Introduction to FPGA

Kevin Yang
[1]Computer Science, Arkansas State University, Jonesboro, Arkansas, USA

Abstract – *Field Programmable Gate Arrays (FPGAs) where first introduced almost two and a half decades ago but they are commonly used in industry and academia due to their area, speed, and power benefits over their homogeneous counterparts. FPGA market is expected to grow to $9.5 billion USD by 2023 [3]. Since their invention by Xilinx in 1984, FPGAs have gone from being simple glue logic chips to actually replacing custom application-specific integrated circuits (ASICs) and processors for single processing and control applications. Why is this technology been so successful? The growing demand for advanced driver-assistance systems (ADAS), the growth of Internet of things and reduction in time-to-market are the key driving factors for the FPGA market. This paper provides an introduction to FPGAs and the internal structure of a generic FPGA. Finally, we finish with a discussion of the benefits that make FPGAs unique. [1][2]*

Keywords: FPGAs, ASICs, ADAS

1. Introduction

FPGA, stand for Field Programmable Gate Array, provide the next generation in the programmable logic devices. An FPGA is an integrated circuit (IC) that can be programmed and configured by the embedded system developer in the field after it has been manufactured [4]. FPGA is a semi-conductor device, which is not limited to any pre-defined hardware function; it is rather highly flexible in its functionality and may be configured by the developer according to his/her design requirements. i.e. The user can program what the logic gate does (be it a NAND or NOR or some form of SUM-PRODUCT implementation) or an adder, you as a user, can "program" the chip to perform that logic function. Now we can add another layer of user programmability--you can program how these logic gates are connected together. In that way we have a general programmable logic chip. Unlike the microprocessor where the program is just the instruction to fix digital hardware, here you can program the hardware itself [5]. Therefore, FPGAs use pre-built logic blocks and programmable routing channels for implementing custom hardware functionality depending upon how embedded system developer configure these devices. Here is another example, a rear-view camera designed for a car. If your camera system take 250 milliseconds from the time to image sensor sees the image until the image frame actually appears on the display, and a change in government regulations requires that this delay or latency be no more than 100 milliseconds, you could find ways to adjust the image signal processing pipeline in an FPGA to comply with the new latency requirements. This would be almost impossible to do with a microprocessor-based system. In this example, a company can gain a big advantage using an FPGA because it does not have to redesign parts or buy all new processors [5]. On the other hands, in microcontrollers, the chip is designed for a customer and they have to write the software and compile it to hex file, which is a file format that conveys binary information in ASCII text form. It is commonly used for programming microcontrollers, to load onto microcontroller. This software can be easily re-placed as it is stored in flash memory. However, in FPGAs, there is no processor to run the software and we are the one designing the circuit. We can configure an FPGA as simple as an AND gate or a complex as the multi-core processor [6]. In addition, FPGAs are programmed and configured using hardware description languages (HDL), which is of two types--Verilog and VHDL. Then the HDL is synthesized into a bit file using a BITGEN to configure FPGA. FPGA stores the configuration in RAM that is the configuration lost when there is no power connectivity. Hence, they must be configured every time power is supplied.

Those features of FPGA make it unique from ASIC technologies such as standard cells. ASICs typically take months to fabricate and cost hundreds of thousands to millions of dollars to obtain the first device; FPGA on the other side, take less than a minute to configure and they cost anywhere around a few hundred dollars to a few thousand dollars. FPGA requires approximately 20 to 35 times more area than a standard cell ASIC, has a speed performance roughly 3 to 4 times slower than an ASIC and consumes roughly 10 times as much dynamic power. These disadvantages arise largely from an FPGA's programmable routing fabric, which trades area, speed, and power in return for "instant" fabrication. Despite these disadvantages, FPGAs present a compelling alternative for digital system implementation based on their fast-turnaround and low volume cost. For small enterprises or small entities with in large corporations, FPGAs provide the only economical access to the scalability and performance provided by Moore's law [7]. As Moore's law progresses, the ensuing difficulties brought about by state-of-the-art deep submicron processes make ASIC deign more difficult and expensive [8].

The remainder of the paper is organized as follows: Section 2 gives an overview of FPGA history. Section 3 describes the architecture of FPGAs, which include

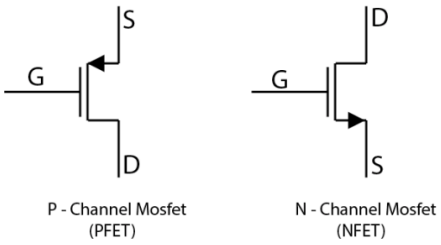
programmable logic blocks, resources for interconnection, and I/O blocks. Section 4 gives the list of major manufactures. Section 5 explores some application of FPGA. Section 6 talks about the benefits of FPGA, which compare to ASIC and CPU.

2. History of FPGA

The origins of the contemporary Field-Programmable Gate Array are tied to the development of the integrated circuit in the early 1960s; however, there is still one predecessor before IC, which is called--MOSFET.

A Brief Timeline of the steps leading to FPGA development [13]:

1960--First MOSFET, Metal-Oxide-Semiconductor Filed-Effect Transistor) are one of the most basic elements in an FPGA. In the world of FPGAs, they are usually referred to as gates. They are used similar to switches. Depending on what value they receive, 0 or 1, they will either block or allow the flow of current. There are two main types of MOSFETs--PFETS and NFETS.



The NFET and PFET shown with arrows denoting the type.

NFETS turn on, allow current, and when past a 1, PFETS act in the opposite way. When you program the FPGA chip, it has to arrange all these MOSFETs so they provide the required output. One example of how they work is in a NAND gate. A NAND gate performs the not-AND operation. Sometimes, NFETS and PFETS are shown without the arrows but with a circle on the gate to denote a PFET. In the image, you can see that if A and B are past a 0, the two PFETs on the top will be turned on and the two NFETs connected to VSS or ground will be off, therefore, VDD will be passed, or logically it will pass a 1. Since the PFETs are in parallel, if at A or B is 0, VDD will be passed and ground will be disconnected. Hence, this acts as a NAND gate should be 0 and it will only be passed if ground is connected. Ground is connected only if both A and B are 1 and the NFETs are turned on.

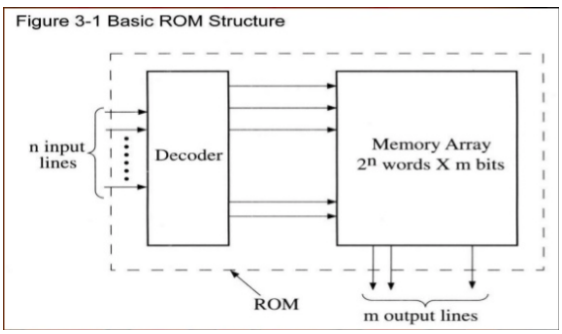
1961--First Communication IC. An IC is a small chip that has a set circuit on it. For instance, you can buy an IC that just has a mux on it. ICs were an upgrade from using transistors to build circuits, because they are generally printed. This way you can print an entire circuit in one shot, rather than having to assemble the circuit with possibly thousands of transistors; it saves time and space. When you write an FPGA project in HDL, you instantiate modules in code. This would be analogous to ordering ICs of each of the modules that you need and physically wiring them together. No doubt that you do not have the protection that FPGA chips offer, and it is very easy to burn out your ICs, but ICs were one of the major building blocks in FPGAs for this reason. Engineers had a way to "customize" circuits by putting together multiple ICs, nonetheless, this process not only tedious and exhaustive but also not as automated as coding the blocks. On top of that, they are not as reusable. People cannot simply reprogram their circuit like someone could do on FPGA. Another problem with ICs is that the circuit of itself become huge and very quickly. However, ICs are not irrelevant. In fact there are quite a few in the chipKIT Starter Kit, such as the mux IC. An example of how this used can be found in [9, 10].

1962--First TTL. Transistor-Transistor Logic is a family of ICs, rather than being made of MOSFETs like the ICs, a TTL is made with BJTs (Bipolar Junction Transistors). The main advantage of the TTL over the IC is that it consumes less power and is less sensitive to damage from electrostatic discharge [11].

1963--First CMOS. Complementary Metal-Oxide Semiconductor is again an improvement on the IC technology, but his time an improvement on how they are constructed. The PFESTs and NFETs in COMS circuits are paired so that one in each pair is always off. This reduces waste heat as there is no standing current during transition.

1965--Moore's Law. This law is based on the prediction that the number of transistors in a circuit doubles every year, and is now used in industry and research to set goals. Unfortunately, soon Moore's Law may not hold, as technology is getting to the point where transistors are a number of silicon atoms wide. Fore more information [12].

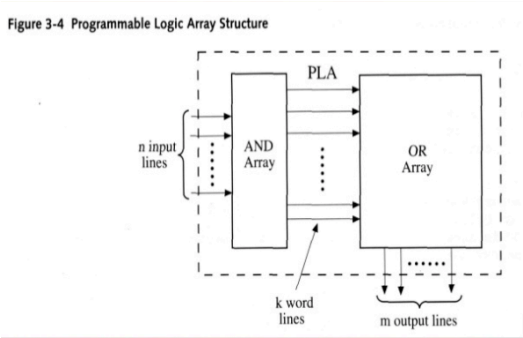
1970--PROM. Programmable Read-Only Memory was one of the first types of programmable memory. Previously there had been ROM (Read-Only Memory) where the data was set at manufacturing. Although PROM was a huge step in programmable technology, there was still improvement to be made, as it was not re-programmable. Once you set the memory, it was permanent [14].



1971--EPROM. Erasable Programmable Read-Only Memory was the second step in programmable memory, in that you could erase it. To erase an EPROM, you had to shine UV-light directly on the EPROM. However, this type of erasing is not the most practical, as you would have to take the EPROM out of the circuit to erase it.

1972--DST. Depleted-Substrate Transistor is similar to a MOSFET, except that it has no voltage difference from its gate to source. This provides higher gain and low noise as compared to a normal MOSFET.

1975--PLA. Programmable Logic Array is made up of programmable AND gate planes and programmable OR gate planes, connected to product a desired output. With this type of programmable logic you could most directly implement POS (Product Of Sums), OR gate AND gate, and SOP (Sum Of Products), AND gate then OR gates, which are now only used as a formalism in design, and teaching tool. This allowed for programmable logic, but nothing near the complexity of design that FPGAs allow [14].



1978--PAL. Programmable Array Logic is similar to PLA; however, rather than having two programmable planes, it instead has one PROM array, a fixed OR plane and a programmable AND plane. This only allows for sum of products logic equations with feedback from the outputs.

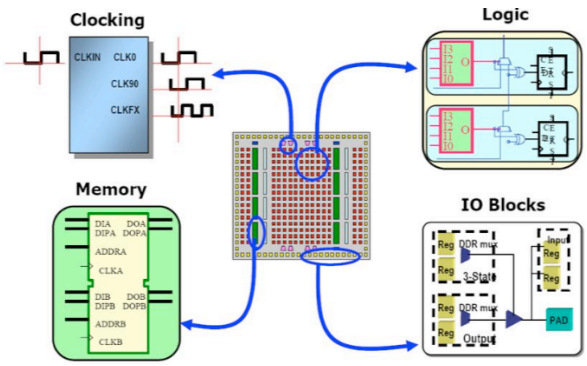
1983--EEPROM. Electrically Erasable Programmable Read-Only Memory offered a huge improvement from EPROM, in that you did not have to shine UV light on the memory to erase it. It was memory that was finally erasable electrically. This allowed for data to be read, erased, and reprogrammed. However, EEPROMs only allow for a certain number of times it can be reprogrammed.

1983--GAL. Generic Array Logic was a major improvement on PAL. GAL is completely erasable and reprogrammable.

1984--FLASH. Flash memory is a type of EEPROM, and it is the most commonly used non-volatile memory. One major improvement from the EEPROM is that FLASH memory can be erased in blocks. So, rather than having to erase all of the memory and rewrite it, you can choose what memory to erase.

Xilinx introduced the first modern-era FPGA in 1984. It contained the now classic array of CLB, Configurable Logic Blocks. From that first FPGA which contained 64 logic blocks and 58 inputs and outputs, FPGAs have grown enormously logic blocks and around 1100 inputs and outputs, In addition, a large number of more specialized blocks have greatly expanded the capabilities of FPGAs. These massive increases in capabilities have been accompanied by significant architectural changes.

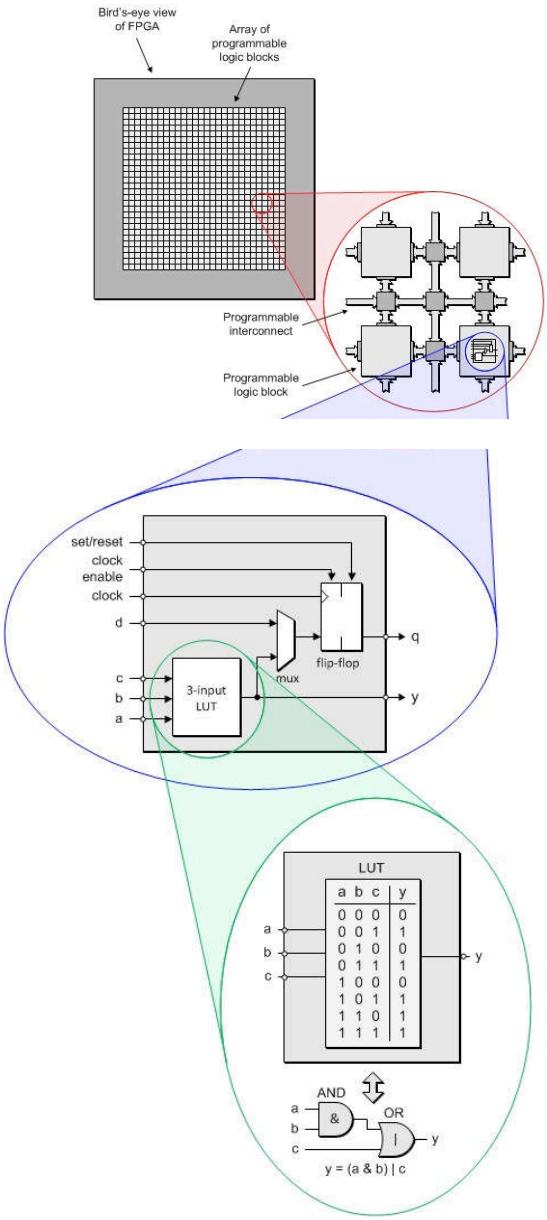
3. Architecture of FPGA



A typical modern FPGA provides the designer with programmable logic blocks that contain the pool of combinatorial blocks and flip-flops to be used in the design. Moreover, vendors acknowledge the fact that logic is often used in conjunction with memory, and typically include variable amounts of static Random Access Memory (RAM) inside their chips. Clock conditioning has also become commonplace, and support in the form of Delay Locked Loops (DLLs) and Phase Locked Loops (PLLs) is aslo provided inside the same silicon chip. Finally, an FPGA chip does not lead a solitary life isolated from the rest of the world. It needs to be easily interfaced to other chips or external signals. In order to make this interfacing easier, FPGA vendors have invested a great deal of effort in enhancing the flexibility of the input/output blocks behinds the chip pad. Each pad can serve as an input, and output, or both. The list of electrical standards supported is extensive, and novel techniques for maximizing bandwidth, such as clocking data in using both edges of the clock, are widely supported [15].

3.1 Simple FPGA fabric

In the context of an IC, it is common to hear the term *fabric*, which is used to refer to the underlying structure of device. If we were to peer inside the FPGA's package, the programmable fabric is presented in the form of an array of programmable logic blocks as shown in the image below. If we zoom in with a metaphorical magnifying glass, we see that this fabric comprises "islands" of logic, the programmable logic blocks, basking in a "sea" of programmable interconnect.

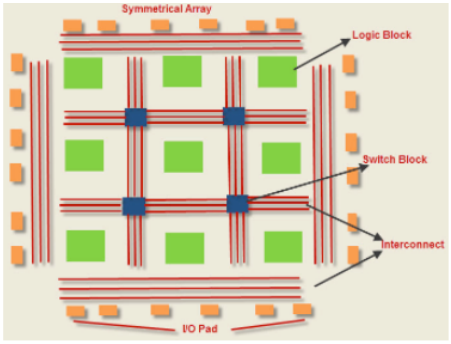


A generic representation of fundamental FPGA programmable fabric.

If we zoom in further, we see that each of the programmable blocks contains a number of digital functions. In this example, we see a 3-input Lookup Table (LUT), a multiplexer, and a flip-flop, but it is important to realize that the number and types and sizes of these functions varies from family to family. The flip-flop can be configured to act as a register or latch; the multiplexer can be configured to select an input to the block or the output from the LUT, and the LUT can be configured to represent whatever logic function is required [16].

3.2 Internal Architecture of FPGAs

There are several different families of FPGAs, manufactured by different semi-conductor companies like Xilinx, Altera and Actel etc., that are currently available in market. These device families, when compared with one another, show slight variations in their features and architecture. Despite the slight differences in their architecture and features, on the whole, these device families follow a common design methodology.



The general FPGA architecture consists of three types of modules. They are I/O blocks or Pads, Switch Matrix/Interconnection Wires and Configurable Logic Blocks (CLB). The basic FPGA architecture has two-dimensional arrays of logic blocks with a means for a user to arrange the interconnection between the logic blocks. The functions of an FPGA architecture module are discussed below [17]:

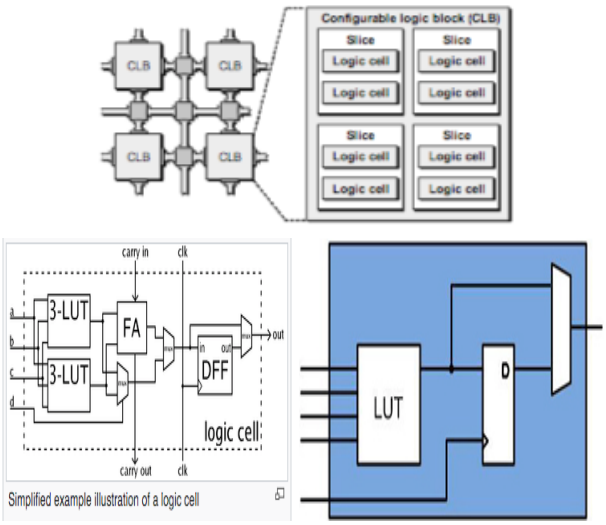
- Configurable Logic Block (CLB) includes digital logic, inputs, and outputs. It implements the user logic.
- Depending on the logic, switch matrix provides switching between interconnects.
- I/O pads used for the outside world to communicate with different applications.

3.3 Configurable Logic Blocks

A Configurable Logic Block (CLB) is a basic component of an FPGA that provides the basic logic and storage functionality for a target application design. In order to provide the basic logic and storage capability, the basic component can be either a transistor or an entire processor. However, these are the two extremes where at one side the basic componment is very fine-grained (in case of transistors) and requires large amount of programmable interconnected which eventually results in an FPGA that suffers from area-inefficiency, low performance and high power consumption. On the other side (in case of processor), the basic logic block is very coarse-grained and cannot be used to implement small functions as it will lead to wastage of resources. In between these two extremes, there exists a spectrum of basic logic blocks. Some of them include logic blocks, that are made of NAND gates, an interconnection of multiplexors, lookup table and PAL style wide input gates.

In general, a logic block consists of a few logical cells (called ALM, LE, Slice etc.). A logic cell is composed of three basic components: a small LUT, a D-Flip-flop, and a

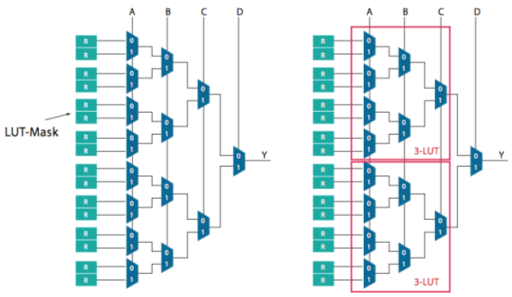
Multiplexer. (different manufacturer may generate different number of multiplexer or the number of input LUT)



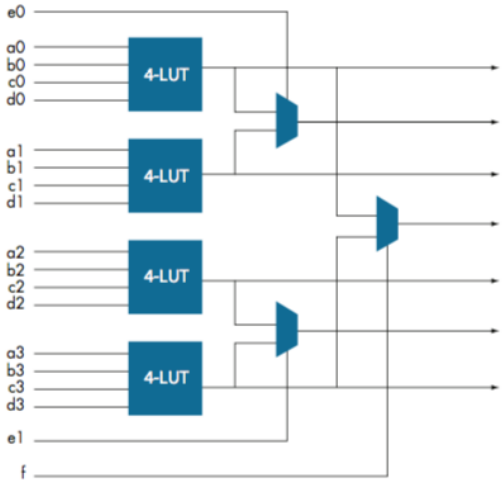
If required, the Multiplexer may be used to bypass the flip-flop. Flip-flops are binary shift registers, which are used to synchronize logic and save logical states between clock cycles. On every clock edge, a flip-flop latches a 1 or 0 value on its input and holds that value constant until the next clock edge. A LUT may be regarded as a small RAM, which is capable of implementing any logic function. Each logic-cell when taken individually may of little use. Nevertheless, a large number of logic-cells when connected together may be employed for implementation of complex logic functions. FPGAs are equipped with fairly a large number of routing resources (Multiplexers or wires placed around the logic cells) for connecting together a huge number of logic cells for implementation of complex and useful logical functions.

3.4 A closer look at Lookup Tables

An overview of how LUTs are built helps describe the key innovations in the ALM. A LUT is typically built out of SRAM bits to hold the configuration memory (CRAM) LUT-mask and a set of multiplexers to select the bit of CRAM that is to drive the output. To implement a k-input LUT (k-LUT)--a LUT that can implement an function of k inputs--2^k:1 multiplexer are needed.



The figure shows a 4-LUT, which consists of 16 bits of SRAM (FPGA may be implemented using antifuses, Flash memory cells, or SRAM memory cells) and a 16:1 multiplexer implemented as a tree of 2:1 multiplexers. The 4-LUT can implement any function of 4 inputs (A, B, C, D) by setting the appropriate value in the LUT-mask. To simplify the 4-LUT in this figure, it can also be built from two 3-LUTs connected by a 2:1 multiplexer. Similarly, larger LUTs can be built out of smaller ones, as shown in figure below. For example, a 5-LUT can be built with two 4-LUTs and a multiplexer, while a 6-LUT can be built with two 5-LUTs and a multiplexer.



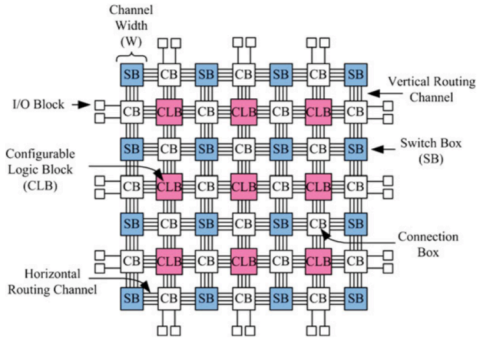
Even though larger LUTs can be built from smaller LUTs, it is important to differentiate between FPGA architectures designed for 4-LUTs and for 6-LUTs. With a different size LUT as the base logic block, the number of LUTs clustered together in each architecture and the number of inputs available to the LUTs, and delay optimization through the LUTs will vary. While 6-LUTs can be built on architectures that support 4-LUTs, this structure is inefficient. For instance, four 4-LUTs together with either a 4:1 multiplexer or 2 more 4-LUTs can be used to build a 6-LUT as shown in figure above, but the implementation uses only 6 of the 16 available inputs and creates extra delays between the various LUTs. Clearly, having the ability to built 6-input LUTs is not enough; the entire architecture needs to be optimized specifically for 6-LUTs as the base logic block [18].

3.5 Routing Architecture

As discussed earlier, in an FPGA, the computing functionality is provided by its programmable logic blocks and these blocks connect to each other through programmable routing network. This programmable routing network provides routing connections among logic blocks and I/O blocks to complete a user-designed circuit. It consists of wires and programmable switches that form the desired connections. These programmable switches are configured using the programmable technology. Routing interconnect must be very flexible, because FPGA architectures claim to be potential

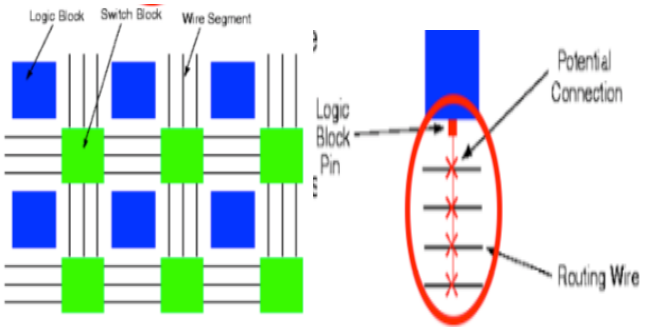
candidate for the implementation of any digital circuit. So that they can accommodate a wide variety of circuits with design goals of speed performance and power consumption. Although the routing demand of logic circuits varies from design to design, certain common characteristics of these circuits can be used to optimally design the routing interconnect of FPGA architecture. For example, most of the designs exhibit locality, hence requiring abundant short wires. But at the same time, there are some distant connections, which leads to the need for sparse long wires. So it needs to be taken into account while designing routing interconnect for FPGA architectures where we have to address both flexibility and efficiency. Additionally, circuits also contain a number of signals such as clocks and resets that must be widely distributed across the FPGA. Modern FPGAs all contain dedicated interconnect networks that handle the distribution of these signals. Typically, these networks are carefully designed to be low skew for use in distributing clock signals. They generally can be directly connected to flip-flops and the networks can only be driven by a limited number of resources on the FPGA. The arrangement of routing resources, relative to the arrangement of logic blocks of the architecture, plays a very important role in the overall efficiency of the architecture. This arrangement is termed here as global routing architecture whereas the microscopic details regarding the switch topology of different switch blocks is termed as detailed routing architecture. In recent years, the issue of single-driver versus multiple-driver wires, which gives rise to wires that send signals in a specific direction, has also arisen as an important part of detailed routing architecture. FPGA global routing architectures can be characterized as either island-style or hierarchical. But we only discuss the island-style in this paper.

3.6 Island-Style Routing Architecture



As shown in figure above, island-style FPGAs (also called mesh-based FPGA architecture) logic blocks are arranged in a two-dimensional mesh with routing resources evenly distributed throughout the mesh. An island-style global routing architecture typically has routing channels on all four sides of the logic blocks. The number of wires contained in a channel, W, is pre-set during fabrication, and is one of the key choices made by the architect. Island-style routing architectures generally employ wire segments of different lengths in each channel in an attempt to provide the most appropriate length for each given connection. They also typically stagger the starting point of the wire segments so that each logic block has a chance of connecting at the beginning of a wire of the most appropriate length.

Currently, most commercial SRAM-based FPGA architecture use island-style architectures. This routing structure offers a number of desirable properties. Since routing wires of different lengths are in close physical proximity to logic blocks, efficient connections for a variety of design net lengths can be formed. By staggering the start and end points of channel segments of the same length, the physical layout for each logic block and surrounding routing channels can be optimized to form a single tile.

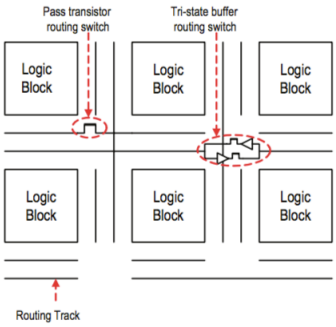


This combined logic and routing tile can be replicated in two-dimensions to form the FPGA array. As a result of this regularity, the minimum feasible routing delay between logic blocks can quickly be estimated.

The routing network of an FPGA occupies 80-90% of total area, whereas the logic area occupies only 10-20% area. The flexibility of an FPGA is mainly dependent on its programmable routing network. A mesh-based FPGA routing network consists of horizontal and vertical routing tracks, which are interconnected through switch boxes (SB). Logic blocks are connected to the routing network through connection box (CB). The flexibility of a connection box (Fc) is the number of routing tracks of adjacent channel, which are channel to the pin of a block. The connectivity of input pins of logic blocks with the adjacent routing channel is called as Fc(in); the connectivity of output pints of the logic blocks with the adjacent routing channel is called as Fc(out). An Fc(in) equal to 1.0 means that all the tracks of adjacent routing channel are connected to the input pin of the logic block. The flexibility of switch box (Fs) is the total number of tracks with which every track entering in the switch box connects to. The number of tracks in routing channel is called the channel width of the architecture. Same channel width is used for all horizontal and vertical routing channels of the architecture.

Many FPGA architectures have been developed that use pass transistors and tri-state buffers as routing switches. Figure below illustrates a routing architecture, which contains both pass transistors and tri-state buffers. Both of these switch implementations support bidirectional wire segments since each segment can be driven by switches in multiple switch

block. The relative usage of each type of switch dictates FPGA area and performance. Pass transistors minimize area consumption and are faster than buffered connections for short wiring paths that pass through a small number of switches.



Generally, tri-state buffers provide faster interconnect for connections that pass through many switches. As result, FPGA devices that intersperse pass transistors and tri-state buffers in the routing fabric provide better delay characteristics with the same area consumption as those that provide only one type of switch.

3.7 Input/Output Architecture

We refer to the I/O pad and surrounding supporting logic and circuitry as an input/output cell. These cells are important components of an FPGA both because this interface sets the rate for external communication and these cells along with their supporting peripherals consume a significant portion of an FPGA's area. A crucial consideration in I/O cell design is the selection of which interface standards to support.

The major challenge in input/output architecture design is the great diversity in input/output standards. For example, different standards may require different input voltage thresholds and output voltage levels. To support these differences, different I/O supply voltages are often needed for each standard. They may also require a reference voltage to compare against the input voltages. Other standards require clamping diodes, which allow specific abnormally high or low voltages to be tolerated. Many standards also rely on differential signaling to improve noise immunity and enable increased data transmission speeds. Proper termination is also essential for maintaining signal integrity but different standards have different termination requirements.

One of the most significant decisions in input/output architecture design is the selection of the standards that will be supported. This involves carefully made trade-offs because, unlike general-purpose logic structures, such as LUT, which can implement any digital function, an I/O cell can generally only implement the standards selected by the I/O cell designer. However, the decision regarding which standards to support is far from straightforward. Supporting a greater number of standards can increase the silicon area required for the I/O cells significantly. Additionally, the pin capacitance may increase with each additional supported standard, which can limit performance. Nevertheless, the usefulness of FPGAs depends on their flexibility, including the ability to support different signaling standards. Given these conflicting factors, the final selection of standards often depends heavily on business factors as opposed to technical factors, and therefore the choice of which standards to support is typically made by a marketing research arm of an FPGA vendor.

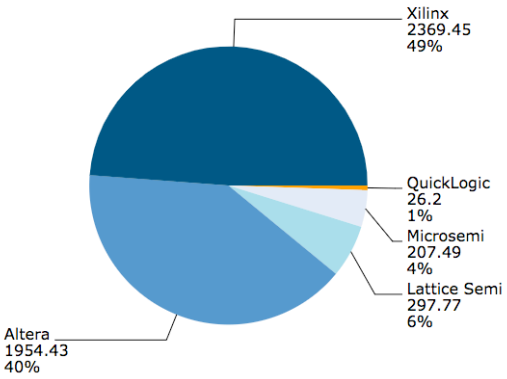
Once the I/O standards to be supported are known, it is necessary to determine, which input/output pins will support each standard. At one extreme every pin can support every standard and feature. This approach is clearly the most expensive and may result in implementation difficulties as a large capacitance will be attached to each pin. Nonetheless, this generality gives the printed circuit board designer who uses the chip the most flexibility. At the other extreme, different standards can be limited to different groups of I/O pins. This approach may result in easier electrical design and lower-cost (to the FPGA vendor) I/O cells, but it may limit flexibility for the printed circuit board designer.

It is desirable to make all the input/output pins in an FPGA equivalent. Until recently, this was generally the case for commercial FPGA offerings. However, the increase in the number of input/output pins on an FPGA and the number of standards for inter-chip communication has made full equivalency impractical. Many standards have conflicting requirements such as differing output voltages. Given such differences it would be impractical for every pin to be able to independently support every standard. Instead, most modern FPGAs have adopted an I/O banking scheme in which input/out cells are grouped into predefined banks. Each bank shares supply and reference voltage supplies. A single bank therefore cannot support all the standards simultaneously, but different banks can have different supplies to support otherwise incompatible standards. We have briefly discussed some of the many issues that must be considered in the design of an FPGA's I/O architecture. While modern commercial FPGAs provide some solutions to these design questions, FPGA I/O architecture remains a relatively unexplored area. A better understanding of the appropriate granularity for I/O banks is needed, and the extent to which equivalent I/O banks are necessary should be explored. Both issues require an understanding of the needs of state-of-art printed circuit board design.

4. Major Manufacturer

With the top two FPFA companies taking up 89% of the FPGA market, people can be forgiven for thinking there was no one else out there. Xilinx and Altera have done a good job for defending the duopoly but a few companies are gradually winning market share by targeting specific applicaitons and

sub-markets. Here is a list of the top 5 FPGA companies by revenue in 2010 [19].



4.1 Xilinx

The leader in FPGAs for many years, Xilinx has a good range of FPAGs in terms of cost and performance. In recent years, the popular Spartan series has covered the low-two-mid-end market while the Virtex series has covered the high-end. Recently, Xilinx released the "7" family of FPGAs which are built on 28-nm process and for the first time introduced the Artex-7 and Kintex-7 series which provide better coverage of the lower and mid-end applications previously covered by the Spartan series. The Kintex-7 recently won the "Highly Commended Prize" Semiconductor of the year award for 2011 [20].

4.2 Altera

The Altera FPGAs cover the low, mid and upper end markets with the Cyclone, Arria and Stratix series respectively. The most recent offering from Altera is the Cyclone-V, Arria-V and Stratix-V, all build on 28-nm process technology [21].

4.3 Lattice Semiconductor

Lattice Semiconductor tackles the low-power and low-cost market for FPGAs. They market their products as the "high-value FPGAs" of the industry, providing the best performance per cost. With the explosion in portable elecrtonics, this has been a good strategy for Lattice. Lattic claims to have the industry's lowest power and price SERDES-capable FPGA: LatticeECP3. Obviously, they did not trend of naming FPGAs after greek mythology or meteorological phenomena [22].

4.4 Microsemi

Microsemi specializes in low-power an mixed-signal FPGAs. Here are some of Microsemi's claim:

- The industry's lowest power FPGA: the IGLOO
- The industry's only FPGA with hard 32-bit ARM cortex-M3 microcontroller: the SmartFusion [23]

4.5 QuickLogic

QuickLogic's focus is on the mobile devices industry meaning ultra-low power, small form factor packaging, and high design security. Rather than selling "FPGA", they pitch "customizable semiconductors". You will not find the word "FPGA" on the front page of their website [24].

5. Applications of FPGA

Application areas of FPGAs are quite diverse and wide ranging. Broadly speaking, major application areas of FPGAs are: Digital Signal Processing (DSP), video processing, software defined radio, control system engineering, bioinformatics, aerospace and defense systems, computer vision, speech recognition and processing, medical imaging, computer hardware emulation, ASIC prototyping, reconfigurable computing and radio astronomy etc.

Traditionally, FPGAs are utilized in applications where the volume of production is small and development resources and expenses required for creating an ASIC for that low-volume application are prohibitively high. With advancement in FPGA technology the areas of application of FPGAs are growing day by day. FPGAs are particularly suitable in applications or implementation of algorithms where parallel processing offered by the architecture of FPGA may be utilized to deliver high performance. Due to the inherent parallelism offered by the internal architecture and logic resources, FPGAs are capable of delivering high throughput even at low clock rates. Because of the advancement in FPGA technology and availability of sophisticated development tools application areas of FPGAs are growing at a very high rate and it is anticipated that in future this is technology may even replace ASIC.

6. Key benefits of FPGA technology

The global market of FPGAs is growing at an enormously high rate and popularity of PFGAs is growing day by day. A unique feature of FPGA is that it combines the best parts of ASICs and processor-based system. The most compelling advantages of FPGAs are very short time-to-market, low start-up cost, low financial risk and flexibility of updating the design as FPGAs are programmed and configured by the end-user. Some of the key advantages of FPGAs are:

- Compared to CPU, FPGA has the following benefits: performance, throughput, and reliability.
- Compared to ASIC, FPGA has the following benefits: re-configurability, cost, time to market.

6.1 Performance

- FPGA code runs in real time in nanoseconds.
- All the logic blocks inside FPGA can run concurrently in parallel, since they are real hardware circuits.
- If fully utilized , FPGA can implement a high performance many-core system.

6.2 Throughput

- FPGA has many high speed I/O pins to interface with memory, Ethernet, etc. It has dedicated channel for peripherals, unlike CPU whose peripherals share a bus.
- FPGA vendors provide many high speed communication IP cores.

6.3 Reliability

- Software programs are usually built upon several layers of abstractions (driver, OS, etc) to help schedule tasks and share resources. They are at risk of incompatibility, recource contention, deadline violation, etc.
- FPGA circuitry is a hard implementation, which is deterministic and well predictable.

6.4 Re-configurability

- ASIC can only do a specific job. If you want an extra function after an ASIC has been manufactured, design another one. While for FPGA, that is nothing but adding a piece of code and re-compile it.
- Certain bugs are caught after the ASIC has been manufactured and distributed to customers. A recall may put the company into bankrupt. While for FPGA, developers can fix the bug in the HDL code and distribute the update to customers.

6.5 Cost

- ASIC development is usually non-recurring engineering. Product fabrication also costs a lot of money.
- FPGA vendors produce large volume of chips so that end users do not need to eat the fabrication cost. Re-configurability of FPGA makes it reusable for different projects.

6.6 Time to market

- ASIC needs cost many years to design, test, validate, and fabricate.
- FPGA development boards usually come with a rich set of peripherals and IP cores which makes it ideal for rapid prototyping.

7. Conclusions

In this paper we have explored some basic aspect of FPAG such as the concept of FPAG, history, architecture, benefits, and some application. However, there still have a lot of issue that we can do further research like tree-based FPGA architecture, or hierarchical routing architecture and also each component inside the FPGA. Moreover, different manufacturer has their own FPGA architecture, so these devices have more ways to do further. While FPGA has changed dramatically in last two decades, it is clear that many fundamental questions remain, driven by rapid changes in technology and application. Hence, more research on this topic is necessary because this is the trend of futher.

8. References

[1] National Instruments. “Introduction to FPGA technology” (http://www.ni.com/white-paper/6984/en/)

[2] MARKETSANDMARKETS. “FPGA Market by Technology”(www.marketsandmarkets.com/Market-Reports/fpga-market-194123367.html)

[3] FPGA RELATED.com. "Introduction to FPGA Technology." (https://www.fpgarelated.com/showarticle/17.php)

[4] Peter Cheung "Introduction to FPGAs"; Department of Electrical & Electronic Engineering Imperial College London

[5] Andrew Moore and Ron Wilson. "FPGAs For Dummies" 2nd Intel(R) Special Edition.

[6] EDGEFX.in. "Basic FPGA Architecture and its Applications". (https://www.edgefx.in/fpga-architecture-applications/)

[7] Moore's Law Wikipedia (https://en.wikipedia.org/wiki/Moore%27s_law)

[8] Ian Kuon "FPGA Architecture: Survey and Challenges" Foundations and Trends(R) in Electronic Design Automation Vol. 2, No.2 (2007) 135-253

[9] Learn.DIGILENT "Multiplexer Guessing Game" (https://learn.digilentinc.com/Documents/292)

[10] Integrated circuit Wikipedia (https://en.wikipedia.org/wiki/Integrated_circuit)

[11] Transistor-transistor logic Wikipedia (https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic)

[12] Purdue University News Service "One and done: Single-atom transistor is end of Moore's Law; may be beginning of quantum computing". (http://www.purdue.edu/newsroom/research/2012/120219KlimeckAtom.html)

[13] Digilent "History of the FPGA" (https://blog.digilentinc.com/history-of-the-fpga/#prettyPhoto)

[14] InSideShare "Introduction to FPGA, VHDL" (file:///Users/Kevin/Desktop/Graduate%20period/2018Spring/FPGA/Introduction%20to%20FPGA,%20VHDL.html)

[15] J. Serrano "Introduction to FPGA design" CERN, Geneva, Switzerland

[16] Embedded cracking the code to systems development. "The MCU guy's introduction to FPGAs: The Hardware" (file:///Users/Kevin/Desktop/Graduate%20period/2018Spring/FPGA/The%20MCU%20guy's%20introduction%20to%20FPGAs_%20The%20Hardware%20_%20Embedded.html)

[17] Elprocus "Basics of FPGA Architecture and Application" (https://www.elprocus.com/fpga-architecture-and-applications/)

[18] ALTERA white paper "FPGA Architecture"

[19] FPGAdeveloper "List and comparison of FPGA companies" (http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html)

[20] Xilinx (https://www.xilinx.com/)

[21] Intel-Altera (https://www.altera.com/)

[22] Lattice semiconductor (http://www.latticesemi.com/)

[23] Microsemi (https://www.microsemi.com/)

[24] Quicklogic corporation (https://www.quicklogic.com/)