FPGA LinkedIn Article

**Introduction**

Field Programmable Gate Arrays, popularly known as FPGAs, are taking over the market by storm. They are widely used nowadays, due to their simplicity in reusability and reconfiguration. Simply put, FPGAs allow you flexibility in your designs and is a way to change how parts of a system work without introducing a large amount of cost and risk of delays into the design schedule. FPGAs were first conceptualized and fabricated by Xilinx in late 80s, and since then, other big companies such as Altera(now Intel) , Qualcomm, Broadcom have followed suit. From industrial control systems to advance military warheads, from self-driving cars to wireless transceivers, FPGAs are everywhere around us. With knowledge of Digital Designing and Hardware Descriptive Languages, such as Verilog HDL or VHDL, we can configure our own FPGAs. Though first thought of as the domain of only Electronics Engineers, FPGAs can now be programmed by almost anyone, thanks to the substantial leaps in OpenCL (Open Computer Language).

I have tried to lay down the concept in terms of 5 questions, so as to cover the majority of the spectrum.

**What is a FPGAs exactly?**

An FPGA is a semiconductor device on which any function can be defined after manufacturing. An FPGA enables you to program new functions product features and functions, adapt to new standards, and reconfigure hardware for specific applications ever after the product has been installed in the field – hence the term *field programmable.* *Gate arrays* are 2 dimensional logic gates which can be used in any way we wish. An FPGA consists of 2 parts, one customizable (containing programmable logic) and another non-customizable. Simply put, it’s an array of logic gates and wires which can be modified in any way, according to the designer.

**Customizable Part**

As rightfully said by Andrew Moore, you can build almost anything digital with three basic components – wires (for data transfer), logic gates (for data manipulation) and registers (for storage reasons). Customisable part consists of Logic Elements (LEs) and a hierarchy of reconfigurable interconnects that allow the LEs to be physically connected with one another. LEs are nothing but a collection of simple logic gates. From simply ANDing/ORing 2 pulses to sending the latest SpaceX project into the space, logic gates, if programmed correctly and smartly, can do anything.

**Non-customizable Part**

The non-customizable part contains hard IPs (intellectual property) which provides rich functionality while reducing power and lowering cost. Hard IP generally consists of memory blocks (like DRAMs), calculating circuits, transceivers, protocol controllers, and even whole multicore microprocessors. These hard IPs frees the designer from reinventing these essential functions every time he wants to make something, as these things are commodities in most electronic systems.

As a designer, you can simply choose whichever essential functionality you want in your design, and can implement any new functionality from the programmable logic area.

**Why are FPGAs gaining popularity?**

Electronics are entering every field. Consider an example of a car. Nowadays, every function of a car is controlled by electronics. Drivetrain technologies like engine, transmission, brakes, steering and tyres use electronics to control and monitor essential conditions like amount of fuel required, optimal air pressure according to usage and surroundings, lucid transmission and even better brakes are achieved due to this. Infotainment in cars is also gaining popularity, such as real time traffic displays, digital controls, and comfort and cruise control settings according to driver’s conditions. Even modern day driving assistance like lights, back-ups, lane-exits guiding and collision avoidance techniques. We are also using sensors like cameras, LASERs and RADARs for optimal driving and parking conditions.

A lot to digest, isn’t it?

All these technologies are implemented on a SoC (System on Chip). But suppose there comes out a better way for gear transmission, or a better algorithm for predictive parking, or the government changes its guidelines pertaining to speed limit for cruise control situations or fuel usage. We can’t change the entire SoC just for some versions. Moreover, these “updates” come often, and we can’t always build new, custom made SoC every time, as the time required to build a new one would increase, whilst also increasing the design and cost load, and on the top of it all, replacing the entire system.

Our humble FPGA comes to the rescue here. SoC FPGAs which can implement changes in specific parts without affecting the other parts, reducing design and time load, and most important of all, reusability of the same hardware by reconfiguring the requisite changes.

FPGAs are gaining popularity because

1. They are reconfigurable in real time
2. Costs less in long runs as compared to ASICs (Application Specific Integrated Circuits). Though ASICs are faster than FPGAs and consume less power, they are not reconfigurable, meaning once made, we can’t add/remove or update any functionalities.
3. They reduce the design work and design time considerably due to inbuilt hard IPs
4. You can build exactly whatever you need using an FPGA.

**When was the 1st FPGA fabricated?**

FPGA was a product of advances in PROMs (Programmable Read Only Memory) and PLDs (Programmable Logic Devices). Both had the option of being programmed in batches or in field (thereby, *field programmable)*. However programmable logic was hardwired between logic gates.

Altera (now Intel) delivered the industry’s first reprogrammable device – the EP300, which allowed the user to shine a ultra-violet lamp on the die to erase the EPROM cells that held the device configuration.

Ross Freeman and Bernard Vonderschmidt (Xilinx co-founders) invented the 1st commercially viable FPGA in 1985 – the legendary XC2064. The XC2064 had programmable gates and programmable interconnects between gates, which marked the beginning of a new technology and market.

The 90s showed the rapid growth for FPGAs, both in terms of circuit sophistication and volume of production. They were mainly used in Telecommunications and Networking industry, due to their reconfigurablility, as these industries demanded changes often and sometimes, in real time.

By the dawn of the new millennium, FPGAs found their way into consumer, automobile and industrial applications.

In 2012, the first complete SoC (System on Chip) chip was built from combining the logic blocks and interconnects of traditional FPGA with an embedded microprocessor and related peripherals. A great example of this would be Xilinx Zynq 7000 which contained 1.0 GHz Dual Core ARM Cortex A9 microprocessor embedded with FPGA’s logic fabric.

Since then, the industry has never looked back, seeing unforeseen growth and applications in recent years.

**Where are FPGAs used?**

FPGAs are used everywhere where there is a need of frequent reconfiguration or where there is a need of addition of new functions, without affecting other functionalities. The car functionalities discussed earlier is a great example in terms of consumer usage.

They are widely used in industries too. Let’s take an example of SoC FPGA for motor control system, which is used in each and every industry. It includes a built-in processor that manages the feedback and control signals. The processor reads the data from the feedback system and runs an algorithm to synchronize the movement of the motors as well as control their rotation speeds. By using a SoC FPGA, you can build your own IP that can be easily customized to work on other motor controls. There are several advantages to using a SoC FPGA for motor control instead of a traditional microcontroller *viz.*  Better system integrations (remember the customizable areas in FPGAs?), scalable performances (rapid and real time re-configurability) and comparatively better functional safety (computing real time data and taking industrial regulations in mind).

Any problem which is [computable](https://en.wikipedia.org/wiki/Computability) can be solved using an FPGA. Their advantage lies in that they are significantly faster for some applications because of their [parallel nature](https://en.wikipedia.org/wiki/Parallel_computing) and [optimality](https://en.wikipedia.org/wiki/Logic_optimization) in terms of the number of gates used for certain processes.

Another trend in the use of FPGAs is [hardware acceleration](https://en.wikipedia.org/wiki/Hardware_acceleration), where one can use the FPGA to accelerate certain parts of an algorithm and share part of the computation between the FPGA and a generic processor (Bing using FPGA for its search algorithm accelerations)  FPGAs are seeing increased use as [AI accelerators](https://en.wikipedia.org/wiki/AI_accelerator)  for accelerating [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) for [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications.

**How can we configure our own FPGA?**

As we know, to make any chip using logic gates, we need Hardware Descriptive Languages such as Verilog HDL or VHDL. These languages are generally known only by people with Electronics Engineering background, thereby keeping this magnificent machinery away from other engineers, thereby increasing the need of a heterogeneous environment for exploiting hardware. OpenCL (developed by Apple Inc.) a pioneer in this field, is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, DSPs, FPGAs, and other types of processors. OpenCL includes a language for developing kernels (functions that execute on hardware devices) as well as application programming interfaces (APIs) that allow a main program to control the kernels. OpenCL allows you to develop your code in the familiar C programming language. Then, using the additional capabilities provided by it, you can separate your code into normal software and kernels that can execute in parallel. These kernels can be sent to the FPGAs without you having to learn the low-level HDL coding practices of FPGA designers.

Sounds too much? Let’s simplify the stuff.

Many of you have had experience with Arduino or similar small microcontroller projects. With these projects, you usually breadboard up a small circuit, connect it to your Arduino, and write some code in the C to perform the task at hand. Typically your breadboard can hold just a few discrete components and small ICs. Then you go through the pain of wiring up the circuit and connecting it to your Arduino with a bird’s nest of jumper wires.

Instead, imagine having a breadboard the size of a basketball court or football field to play with and, best of all, no jumper wires. Imagine you can connect everything virtually. You don’t even need to buy a separate microcontroller board; you can just drop different processors into your design as you choose. Now that’s what I’m talking about!

Welcome to the world of FPGAs.

References:

1. Intel: <https://www.intel.in/>
2. Wikipedia: <https://en.wikipedia.org/wiki/Field-programmable_gate_array>
3. Makezine: <https://makezine.com/2019/10/11/a-brief-history-of-fpga/>
4. Xilinx: <https://www.xilinx.com/>

