FPGA Architectural Flow: CAD Improvements

Vivek Bhardwaj
Intel Corporation
Penang, Malaysia

ABSTRACT
Field Programmable Gate Arrays have long been seen as a viable alternative to Application Specific Integrated Circuits (ASICs). While ASICs have very sophisticated commercialized EDA tools that deliver very fast and power efficient chips, the FPGA world has unfortunately not seen the kind of software investment the ASIC world has seen. However, with the ever rising demand of FPGA based applications and increasing semiconductor complexity of late, the techniques and efficient algorithms of ASIC software have trickled down to FPGA as well, In this paper, we are going to look at some of these techniques that have resulted in better performance per watt- a key metric in FPGA world. We will also do a brief comparison of ASIC vs FPGA design flow and FPGA architecture, connect the dots and make user better aware of the challenges that are faced by FPGA designers in implementing a certain design technique and how the software tries to overcome those challenges. This paper would be useful for new ASIC developers entering in the FPGA world, or even experienced FPGA developers who can get some ideas from this paper for the betterment of the FPGA compilation process.

General Terms
Programmable Logic, Chip Design flow, computer aided design, system-on-chip design, performance, software tools, turnaround time.

Keywords
FPGA, SOC, VLSI CMOS integrated circuits, Moore's law, Electronic design automation, physical design, timing. Quality of results

1. INTRODUCTION
Field-Programmable Gate Arrays (FPGAs) are becoming increasingly important with the advent of Moore's law and increasing semiconductor complexity. The increased complexity of ASICs(application specific integrated circuits) and SoC(System on Chip) design process that includes implementation, verification and fabrication of these state-of-the art chips have led to a rise in investment that needs to be done to make them and deploy in production scale. Such high cost of engineering and manufacturing has led small scale design companies to look for FPGA based design starts and their subsequent implementation. The obvious disadvantage of FPGAs over ASICs is low performance(or timing), higher power consumption and much bigger area of the chip that is manufactured. Their PPA numbers(Power, Performance, Area metrices) are much worse compared to ASIC because of the architectural issues that an FPGA has. However, FPGA's biggest advantage is reconfigurability, i.e. The ability of the same chip to be programmed and then re-programmed into different functions, unlike the ASIC that once burnt on a chip retains the same logic function for the entire chip life cycle duration. The flexibility of an FPGA derives itself from the pre-fabricated nature of the silicon that has a fixed architecture of cells and wires. The desired logic can easily be burnt(and re-burnt) on this fixed architecture for various logical functions. Also, compared to ASICs, FPGAs are very fast to implement. This is very intuitive to understand, if we compare this to a fact that any custom process is much longer and difficult to achieve than a standard(or fixed) process. The same fastness in implementation comes into play while re-programming an FPGA and offers very distinct advantage over an ASIC.

In this paper, we will try to understand some of new innovations that have been done in the CAD(computer aided design) software flow to improve upon various PPA targets and even faster compilation process. These innovations have helped bridge the gap between an ASIC performance vs FPGA performance, hence making FPGAs even more desirable.

To better appreciate these innovations, we would first look at the basic architecture of an FPGA. This architecture would help in understanding basic FPGA terminology. Also, we would be having a brief overview of the CAD process of an FPGA compilation design flow with reference to the architecture. The design flow helps in understanding what goes on behind the flat compilation process. In the last section, we would go over some of these innovations that have helped the FPGA design process.

2. FPGA Architecture and CAD
In this section we would touch on the basics of architecture and the interrelated flow that would help in understanding how FPGA works.

2.1 Architecture
Figure 1 refers to a very basic representation of a FPGA architectural system. This system consists of the few major components and some minor components. The major ones include a logic block and routing wires that interconnect various elements. Logic blocks contain the core circuits for implementing the logic. These blocks are configurable and programmable(implying the logic could be different). In current FPGA systems, there are clusters of such logic block which are called as LAB(Logic Array Block). Outside the LAB's run the routing channels that connects the various LAB's. Then there are minor components like switch blocks and connection blocks through which the interconnect runs and gets switched in state through various kinds of switch blocks.

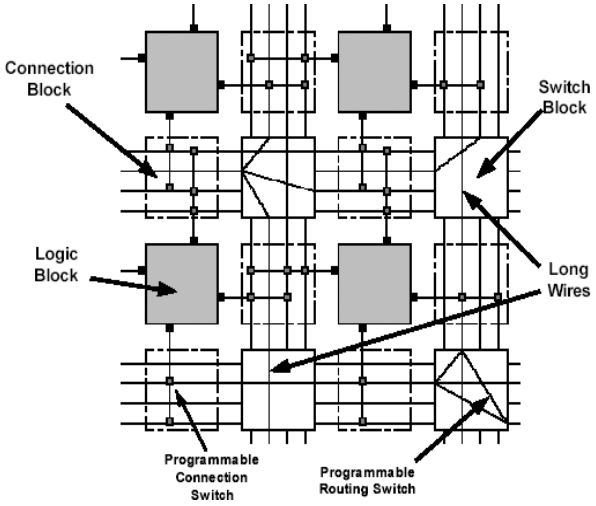

Fig 1: FPGA core diagram [4]

Newer FPGAs(Figure 2- pictorial representation) have a complicated fixed array kind of structure that also contains embedded components that are dedicated to perform various functions like memories, computational elements like DSPs(signal processors) and hard processors, clock generating elements like PLLs . They are also surrounded by IO elements like transceivers and various other kind of IPs(Intellectual property) that serves to implement a complete SoC.
The figure below is one such simple representation. However, this perfect symmetry will always be not true in case of all FPGA systems.

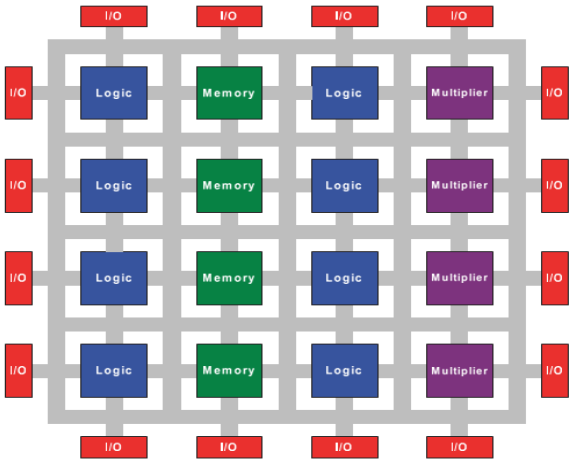

Fig 2: FPGA IO and embedded functions [2]

The architecture of the entire chip is such that it felicitates direct communication between the LABs through dedicated channels in either a one-one connection or through a hex line that runs across the FLGA and all inter-LAB communication are data transfer that happen through these lines rather than classical Manhattan type of routing done in various ASIC style routers. To calculate routing distances and hence timing delays present a challenge in FPGAs compared to ASICs.

A LAB itself is composed of multiple LE's(Logic elements) . The number of LE's could be 4, 16, 32 etc depending on complexity of FPGA). This is shown in Figure 3. Each LE consists of a LUT(Look Up Table) and a register. In the figure, the LE takes I inputs and N outputs. We don't have to go into the details of the LE circuit implementation for the scope of this paper.

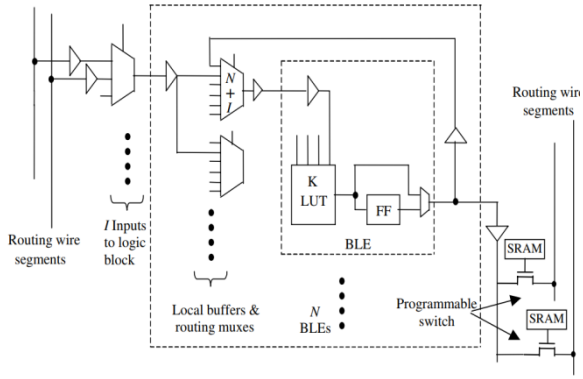

Fig 3: Basic LE representation including a LUT. [1]

So, as we have noted above, an FPGA consists of fixed routing structures and is much more constrained in shapes compared to an ASIC router that can shape as much as possible to meet desired routing metrices.

Another big difference between the ASIC and FPGA architecture is the clock network. While the clock network in an ASIC is primarily free floating and have many degrees of freedom in clock cells placements and routing, the clock architecture of an FPGA is primarily fixed. The clock travels from a predefined location of PLLs and boundary clock signals, through a fixed path of clock network and then bifurcates in an H fashion to the clock root cells(LABs and LE's). Figure 4 shows this pictorially. Hence the problem of clock distribution in an FPGA is essentially of clock assignment rather than a true clock synthesis in an ASIC where the clock is optimized for performance, skew and power. Hence the performance in an FPGA is somewhat limited by this huge architectural constraint. If it is difficult to envision this limitation, one should refer back to the initial statement that FPGAs are more or less fixed in architecture since they need to maintain the idea of re-programmability. However, the newer versions of FPGAs are modified to the best extent possible so that clock tree can be tuned at the last stages of the distribution in order to improve performance.

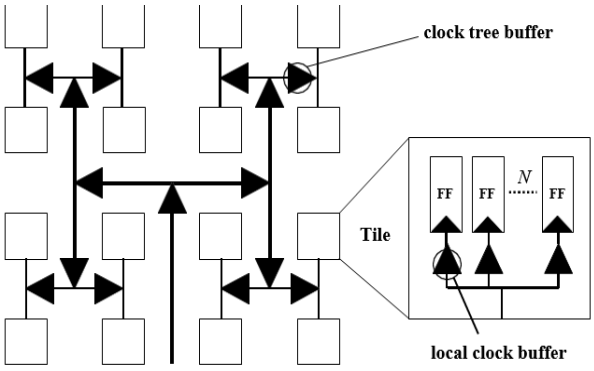

Fig 4: Basic FPGA clock structure [5]

2.2 CAD

The CAD flow or also called as EDA flow(Electronic Design Automation) is the physical design flow that is used to implement or program an FPGA chip. FPGA design flow is somewhat similar to an ASIC flow. Figure 5 illustrates the overall flow that starts with an RTL(Register Transfer Level) .

RTL is nothing but an abstraction of a digital circuit consisting of combinational and sequential elements. RTL is specified in a standard language which is also known as HDL(hardware description language) like Verilog . RTL has always been designed at a functional level . Generally, RTL designers write RTL for their sub-systems by providing optimal logic for functionality and let downstream synthesis and physical design tools manage the performance and power.

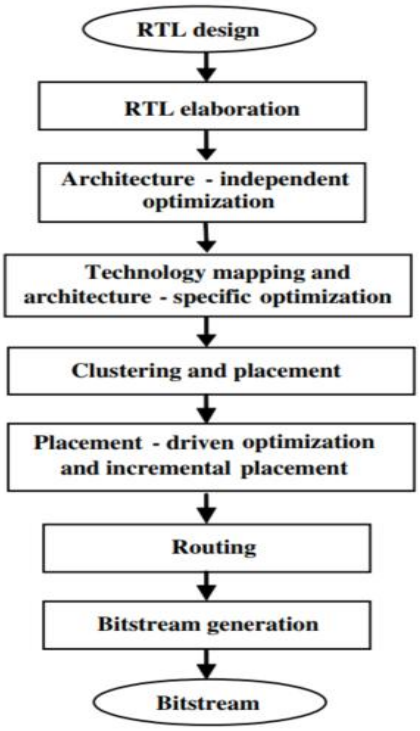

Fig 5: Flat FPGA design flow. [1]

We will not go into details for each step in this paper, but would point out to differences when compared to an ASIC flow, wherever applicable.

Once an RTL is elaborated and synthesized to a gate level netlist, it is converted to an FPGA primitive netlist. One must remember that an FPGA is made up of LE's as fundamental elements and hence the entire netlist is transformed to FPGA understandable one.

The placement and routing steps are conceptually similar to the ASIC flow. However, the placement problem in FPGA is fundamentally different as the locations of LAB's(and all other design elements) are predefined on the FPGA. So, the placement problem is virtually of mapping the netlist to the best possible pre-defined location in order to get the best performance. In ASIC, the gates are free to be placed anywhere on the die as long as it adhering to a legal location. The routing problem also is conceptually same with the difference of pre defined architecture for both data and clock network, unlike an ASIC. Also, in an ASIC flow, the physical optimization is spread across the flow with varying degrees of freedom.

We already have discussed about similar constraints in clock architecture of an FPGA . One additional thing to note is that an FPGA additionally offers multiple type of clock routing resources- E.g. Global network for wide impact clocks but with bigger insertion delay, while regional or local clock networks are offered for last mile clock distribution with smaller insertion delay and is better suited for localized clocks.

Modern FPGA CAD flows are getting seemingly closer to the ASIC flow with the intention of better flexibility and performance.

The flow ends with the generation of bitstream that is nothing a structured combination of 0's and 1's that is programmed onto an FPGA. This part is again specific to an FPGA flow and in integrated in the software CAD flow with various modes of operation. In an ASIC flow, the analogous is the GDS but it is not so tightly integrated into the design flow as it is one time thing only. In FPGA however, if the design change, the bitstream can be regenerated for a brand new FPGA logic.

Another important thing to note is that there are various families of FPGAs provided by the leading providers(Intel provides Aria/Stratix® families while AMD(Xilinx) provides Spartan/Virtex® devices. Each device family has multiple FPGAs that differ in size, performance and power. Different device families differ with each other in architecture, IO, speed etc. A design flow is carried out usually on a specific set of devices within a family and should be carefully chosen based on requirements and cost.

For more details on the CAD flow one can refer to [1] and [2] that describes the flow and architecture in much larger details.

In this section we touched on high level FPGA architecture and the CAD flow that works on that architecture, while outlining some high level differences in ASIC vs FPGA flow. In the next section, we will look at some methodologies and software innovations used to bridge the gap between an ASIC and an FPGA performance.

3. FPGA CAD FLOW INNOVATIONS

In this section we will explore few of the many initiatives FPGA design flow has seen over the past few years to improve the performance and turnaround time of the FPGA compilation flow.

3.1 Exploration modes

FPGA placement quality depends on initial conditions of a design that is determined by various factors like input type and size, compiler settings, operating system used etc. This initial factor is also called as seed . Recent changes in software allows a designer to sweep over various seed values in order to determine the best possible placement of the design. Once a best case seed value is arrived at, the design placement could be locked by various hierarchical flow constraints that we would visit later. Along with the seed sweeping capabilities that give best default values, software also can tune up specific parameters at the cost of other ones. E.g. , the timing effort could be aggressive where the timing weight is higher than power/area and vica-versa. Similarly other efforts could be tuned up as well. This depends largely on the design requirement- some designs need better frequency operations than others, if they are numerically intensive workloads or serve as graphics accelerators. Other operations could be more power consuming and need to scale down the power requirements at the cost of frequency.

Modern ASIC EDA tools let the user customize these settings- and so now the FPGA tools as well.

3.2 Hierarchical flow improvements

Hierarchical flow has reached to a very advanced stage in ASIC CAD flow. There are multiple ways to partition the design and implement block and top portion separately and then assemble

back everything together[9][10]. This is basically due to unrestricted way of handling shapes and locations in an ASIC flow(there are some limitations but not as hard as in FPGAs). In FPGAs however, the flow has been under developed for quite some time. But recently there have been multiple software flows introduced and various new use models provided for the customer to implement a complex SoC on an FPGA in a hierarchical fashion. The design flow mentioned in the previous section is basically a flat flow where small designs are implemented by single designer. However, when it comes to very large and complex SoC where there are many complex core and peripheral logic and functionalities, then a hierarchical flow becomes a must. Lack of an adequate flow hurts the overall turnaround time for a chip or impacts the final timing performance or can lead to both of these issues.

The flow starts in an FPGA starts with the concept of floorplanning the design in a correct fashion. For an FPGA, it is critical to get the locations of the various IPs(like peripheral clocks, transceivers, or core logic memory elements and DSPs) correct in first place. In an ASIC, IPs such as these are hand placed through manual floorplanning. But in FPGA they have pre-defined locations on the chip and a FPGA designer or the CAD tool has to select the best possible location. The design is first divided into partitions and then these partitions are physically locked to a certain area of the chip encompassing certain number of memories or DSPs that need to be used by that partition. Incorrect floorplanning can lead to fatal issues in timing closure and hence it is the first correct step in overall closure and performance. Once the partitions are assigned to the chip, there are various constructs available to implement the chip. The chip can be either implemented in a team based distributed fashion or a single environment fashion. In a team based methodology (which is most close to the true ASIC hierarchical flow), the full design snapshot is available to various teams responsible for implementation of a different portion of the design after budgeting[6][7] including clock and delay budgets[8][11]. Typically, the periphery and various core partitions are implemented by separate teams. As each team comes up with their implementations, they are plugged in another team's version of the design through models or database files. Note that in this flow, true partitioning and subsystem generation[9] is not followed- however it is still a good representation of the hierarchical model.

In a non-team based model, a single user/environment is responsible for timing closure for the entire chip. For a large chip, it is not possible to complete the timing closure of a full design as a single flat flow as it is subject to both runtime bottlenecks and availability of RTL for various sub-blocks. Hence these sub-blocks are partitioned(and floorplanned) just like one does in a team based model. User in most such cases, would like to focus on their most timing critical blocks before closing the other blocks. For exercising this use mode, the software provides the user the option of emptying the entire design except the timing critical block. This is only a virtual empty state as the physical RTL is still present for those blocks. Once the sub-block gets timing closed, its design state can then be locked so that its timing is not impacted by other partitions implementation.

Multiple instantiated modules are not handled by FPGA flows so far. These are modules that have the same logic in multiple partitions. In ASIC flow, these modules are handled in a similar fashion (one block replicating other), but in FPGA all such blocks have to be independently implemented. The software flows are now providing some basic capabilities to implement such modules.

3.3 Register retiming

Register retiming(Figure 6) is a technique used in ASIC flows whereby back-to-back registers are physically shifted in a pipeline kind of design, in such a manner that the timing slacks(delay margins) for each individual stage becomes positive. In other words, excess positive slack is redistributed across the entire pipeline stages, if possible. This technique though common in ASIC flows, finds it difficult to be implemented in FPGA flow because in ASIC, there is very less constraint on the physical movement, while it may be recalled that an FPGA has only fixed locations a new LE(that contains the register) can be mapped to. This was a big bottleneck in FPGA unless recently the architecture of new devices has been changed to accommodate retiming flows. One such architecture is hyperflex® used in Intel family of FPGAs. Introduction of retiming has been very useful in increasing the frequency of operation.

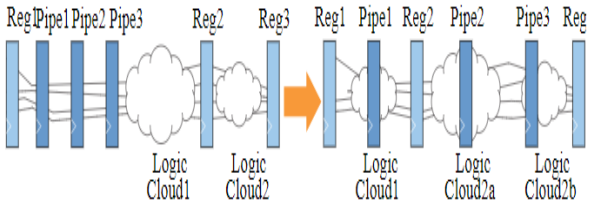

Fig 6: Register retiming in FPGA's

3.4 Other improvements

Other similar architectural changes have been done in other parts of an FPGA E.g redefining the dedicated clock network that distributes the clock in a skew aware fashion. Recent advancements in the FPGA architecture have enabled distribution of clock in a more balanced fashion till the leaf level [3]. For this purpose an entire FPGA has to be divided in sub-regions and clock structure has to be propagated till the sub-region level before arriving at the final root elements.

FPGA flows have borrowed also from the ASIC flows many fancy optimization transforms like register duplication, cloning and declining which gives better performance with the cost of area/power . These transforms have traditionally played a key role in getting optimal QoR(Quality of Results) for ASIC flows.

4. CONCLUSION

Through this paper we have seen and understood basic FPGA architecture and provided references to understand them in detail. We also have gone through the physical flow that helps in generating a bitstream for an FPGA. Finally we have seen some innovations that are being done in the FPGA flow that bridges the gap with the ASIC flow performance, while comparing with the ASIC flow to give a better understanding for people with background of ASIC design. New engineers can get a good perspective of the FPGA flow as well. Through this paper, academia would also benefit by observing the trends seen in the design and automation industry for FPGA design and could provide a pathway for further innovation and research.

5. ACKNOWLEDGMENTS

The author would like to acknowledge various publicly available user guides and reference manuals from Intel detailing about their main FPGA CAD tool- Quartus®

https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/user-guides.html?wapkw=quartus%20user%20guide

The author would like to also acknowledge the hierarchical and quartus® software development team- specially Levitsky O. and Maryan C.

6. REFERENCES

[1] Deming Chen, Jason Cong and Peichen Pan. 2006. FPGA Design Automation: A Survey. Foundations and Trends® in Electronic Design Automation: Vol. 1: No. 3, pp 195-330. http://dx.doi.org/10.1561/1000000003

[2] Ian Kuon, Russell Tessier and Jonathan Rose. 2006. FPGA Architecture: Survey and Challenges. Foundations and Trends® in Electronic Design Automation: Vol. 2: No. 2, pp 135-253. http://dx.doi.org/10.1561/1000000005

[3] Carl Ebeling, Dana How, David Lewis, and Herman Schmit. 2016. Stratix™ 10 High Performance Routable Clock Networks. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Association for Computing Machinery, New York, NY,, 64–73. DOI:https://doi.org/10.1145/2847263.2847279

[4] V. Betz, J. Rose, and A. Marquardt. 1999. Architecture and CAD for Deep- Submicron FPGAs. Kluwer Academic Publishers.

[5] F. Li, D. Chen, L. He, and J. Cong. 2003. Architecture evaluation for power- efficient FPGAs. In ACM International Symposium on Field Programmable Gate Arrays, pages 175–184, Monterey, California.

[6] V. Bhardwaj, O. Levitsky, D. Gupta. 2015. Machine readable products for single pass parallel hierarchical timing closure of integrated circuit designs. US patent 9165098.

[7] V. Bhardwaj, O. Levitsky, D. Gupta. 2013. Flow methodology for single pass parallel hierarchical timing closure of integrated circuit designs. US patent 8365113.

[8] V. Bhardwaj, O. Levitsky, D. Gupta. 2013. Systems for single pass parallel hierarchical timing closure of integrated circuit designs. US patent 8539402.

[9] Vivek Bhardwaj. 2020. Hierarchical Methodology Approach to SOC Design- A comprehensive look.

[10] Vivek Bhardwaj. 2020. Shift Left Trends for Design Convergence in SOC: An EDA Perspective .

[11] V. Bhardwaj, O. Levitsky, D. Gupta. 2015. Methods for single pass parallel hierarchical timing closure of integrated circuit designs. US patent 8935642.