# Physical Design For FPGAs

Rajeev Jayaraman
Xilinx Inc.
2100 Logic Drive
San Jose, CA 95131, USA
rajeev.jayaraman@xilinx.com

## ABSTRACT
FPGAs have been growing at a rapid rate in the past few years. Their ever-increasing gate densities and performance capabilities are making them very popular in the design of digital systems. In this paper we discuss the state-of-the-art in FPGA physical design. Compared to physical design in traditional ASICs, FPGAs pose a different set of requirements and challenges. Consequently the algorithms in FPGA physical design have evolved differently from their ASIC counterparts. Apart from allowing FPGA users to implement their designs on FPGAs, FPGA physical design is also used extensively in developing and evaluating new FPGA architectures. Finally, the future of FPGA physical design is discussed along with how it is interacting with the latest FPGA technologies.

## Keywords
FPGA, Physical design, Placement, Routing.

## 1. INTRODUCTION
Field Programmable Gate Arrays (FPGA) have revolutionized digital system design in the past 15 years. Their programmability and fast time-to-market have made them very popular with digital system designers. About 5 years ago, FPGAs were being used primarily as glue logic in a system. Now, with the arrival of multi-million gate FPGAs and the availability of a variety of system-level features on them, FPGAs are being used to design complete systems.

FPGAs are used in systems for a variety of different reasons. Their use can be classified into four broad categories [10]. They are:

**Production use**: In this category, FPGAs are an integral part of the system in production. Further, due to low volume requirements or rapidly changing market conditions, there is no migration plan to ASICs. Since they are part of production systems, the performance requirements for FPGAs may be very high.

**Pre-production use**: This category of FPGA use is very similar to production use in all respects except one: the use of FPGAs is temporary and only until an equivalent ASIC is deployed. Typically, FPGAs in pre-production use indicate a very tight time-to-market requirement that cannot be met by ASICs. Similar to FPGAs in production use, the performance requirements could be very high.

**Prototyping**: FPGAs in this category are used primarily to prototype a system. The volume requirements are fairly small and the performance requirements may not be stringent.

**Emulation**: Emulation is an effective way of functionally debugging the system and FPGAs are sometimes used to emulate complete systems. The volume requirements are very small and the performance requirements are not critical.

Figure 1. shows the relative usage of FPGAs in these 4 categories.

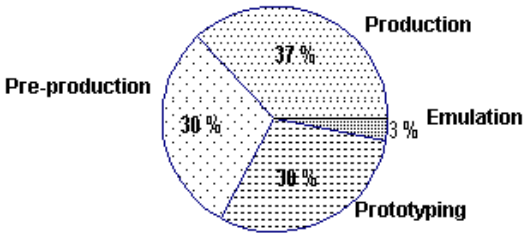


**Figure 1. FPGA Use**

As can be seen in the figure, production and pre-production systems comprise the overwhelming majority of FPGA use: a far cry from the days when FPGAs were primarily used for prototyping and emulation. This directly implies that the time-to-market and high FPGA performance requirements are crucial determinants of FPGA software.

The rapid growth and adoption of FPGAs in digital systems can be traced to three main factors: Business climate, FPGA device features and density, and FPGA software.

**Business Climate**: The business factors that have contributed to the success of FPGAs are reduced time-to-market and lower lifecycle costs. With respect to time-to-market, FPGAs have proved to be very valuable in reducing the system design cycle. Additionally, their re-programmability implies that late feature requirements or bugs caught late in the design cycle are easier and less expensive to fix than ASICs. The re-programmability for FPGAs is also important in lowering the overall lifecycle costs of the system since new features and modifications can be implemented in systems that have already been deployed in the field.

**FPGA device features and density**: Another reason for the popularity of FPGAs among system designers is the addition of several system-level features on FPGAs. Not long ago, FPGAs consisted primarily of configurable logic elements and routing.


*ISPD'01,* April 1-4, 2001, Sonoma, California, USA.

However, in the last 5 years, vendors have started implementing a wide variety of system-level features on their FPGAs such as embedded block RAMs; multiple system clocks with associated clock management circuitry; I/Os that can be configured according to several I/O standards, and embedded processors.

Along with the addition of system-level features, the gate densities of FPGAs have also grown by orders of magnitude in the past 5 years. For example, in 1996, the largest FPGA offered by Xilinx®, the XC4025 consisted of 25000 equivalent user-programmable gates. Today, Xilinx offers the X2V6000 that can implement an equivalent of 6 million user-programmable gates. The combination of system-level features and the large number of user-programmable gates allow FPGAs to implement complete systems on a chip.

**FPGA Software:** Another important reason for the popularity of FPGAs is the FPGA design software. By FPGA software we mean the software that is provided by FPGA and FPGA-CAD vendors to FPGA users to implement their design on an FPGA. Current FPGA software tools give the user the sophistication and capability to start with behavioral and RTL-level descriptions and compile multi-million gate systems in a matter of a few hours. Such fast compile times along with the software's ease-of-use have shortened the FPGA design cycle and have fueled the rapid adoption of FPGAs in systems.

In this paper we will restrict our attention to FPGA physical design software that is used in implementing the user's design on the FPGA. It is important to clarify that we will not be discussing the physical design software used for the actual layout of the FPGA silicon by the FPGA vendors. Instead, we will be discussing the software used in the implementation of the user's design on the configurable logic and routing elements on the FPGA.

Most FPGA vendors rely on third-party EDA vendors to provide synthesis and schematic-based design entry mechanisms. On the other hand, FPGA vendors are typically the primary source for the physical implementation tools such as placement, routing, and configuration programming. The primary reason for this is that the physical implementation software is very closely tied to the FPGA architecture. In fact it is developed simultaneously with new FPGA architectures. The algorithms for these FPGA physical implementation tools started out as algorithmic modifications to classical ASIC physical design algorithms. However, over time, they have evolved in subtle but different ways from classical ASIC physical design algorithms.

This paper will discuss the state-of-the-art in physical design algorithms for FPGAs and contrast them with those algorithms for ASICs. To understand physical design for FPGAs, it is instructive to understand the requirements that drive FPGAs. In the next section, we will discuss the requirements of FPGA physical design software and contrast it with ASIC physical design software. In Section 3 the typical FPGA design flow is described. We define the placement and routing problems for FPGAs along with a discussion of some basic algorithms for them. We conclude with some thoughts on the future of FPGA physical design software.

# 2. ASICs and FPGAs

In this section, we discuss features of FPGAs and FPGA design cycles that have important ramifications for FPGA software.

## 2.1 Design Cycle

The FPGA design cycle can be divided into 3 phases: the evaluation phase, the design and debugging phase, and the production phase. In the evaluation phase, the designer is typically evaluating the FPGA for possible implementation of their design. In this phase, the designers evaluate the estimated ASIC performance of their design and compare it with an estimated FPGA performance of their design. Additionally, users may also evaluate different FPGA architectures and vendors. In this phase, the FPGA physical design tools are required to be very fast and must provide a result that provides a reasonably close estimation of the final system performance.

At the other end of the FPGA design cycle is the production phase, where a design is very close to being complete. Typically, late features or last minute bugs are fixed in this stage. In this phase, FPGA physical design software must focus on getting the best possible performance (or at least a result no worse than before) at the cost of some additional runtime.

The majority of the time in the FPGA design cycle is spent in the debug phase. This is the phase where the designer implements their design, configures the FPGA, and debugs specific functional units of their design. This requires the FPGA software to produce a result with reasonably good performance in a very short time. In this phase, it is more important that the software run fast than it is to produce the highest possible circuit frequency. This is because the time to compile a design to an FPGA is considered "dead time" when the designer cannot look at the results on the bench and hence is not very productive. The "turns-per-day" metric is, therefore, of paramount importance. In other words, it is very important to allow the user to iterate several times a day during this phase. Furthermore, since 90% of the total number of design compilations are done in the debug phase, faster compile times are of paramount importance for FPGA physical design software in this phase.

Depending on the design phase, the primary requirement for the FPGA physical design software changes from extremely fast compile times to extremely good circuit performance by trading off one for the other. In the evaluation and debug phase, runtime is of primary importance and circuit performance, while still being important, is of secondary importance. In the production phase, circuit performance is of primary importance, while runtime of the software is not as important.

While we see that the primary requirements for the FPGA design software can change from very fast compile times to very high frequency implementations it is important to note that the underlying FPGA value proposition of faster time-to-market makes compile times that are greater than 10-12 hours unacceptable. This is a very important requirement for FPGA software since this perceived upper limit on the amount of time a single compilation run can take has not scaled with the increase in the FPGA device and design sizes. For example, in the same time it took to completely place and route the largest FPGA device of 25000 gates five years ago, it is now not only possible but expected that a multi-million gate design be placed and routed.

## 2.2 Deep Sub-micron Effects

A few years ago, FPGAs used process technology that lagged the state-of-the-art. However, in recent years, due partly to their regular structure and high volumes (i.e. volumes in which FPGAs

are manufactured by the vendors), FPGAs have not only become the technology leaders but have actually become the drivers for the latest process advances in the semiconductor fabrication facilities.

Inherent in the state-of-the-art processes come the set of challenges referred to as deep sub-micron effects. With ASICs, the designer has to account for all the deep sub-micron effects in their design. Consequently, ASIC software must provide users with tools to address these deep sub-micron challenges. However, in the case of FPGAs, the FPGA vendors design their FPGAs such that the end user of FPGAs does not have to directly account for many of the deep sub-micron effects. Of course, a result of this design is that some FPGA performance may be sacrificed. However, not having to account for some of these deep sub-micron effects simplifies not only the design cycle for FPGA users but also the development of FPGA software.

Currently, FPGA software does not concern itself with some deep sub-micron effects such as cross-talk and signal integrity to as great an extent as ASIC software. For example, even at 0.13u FPGA software does not have to contend seriously with these deep sub-micron effects. Of course, as geometries get smaller, FPGA software may have to start accounting for these DSM effects since the FPGA architecture itself may not be able to completely shield the user from having to account for them.

While FPGA users do not have to concern themselves with most DSM effects, some DSM effects such as the dominance of routing delays over logic delays are effects that the first FPGAs have had to deal with. In fact, the dominance of routing delays over logic delays in FPGAs is not a result of the sub-micron geometries but more a result of the FPGA architecture. The reason for this is that a typical FPGA connection consists of a combination of metal and one or more programmable interconnect points (PIP) that are usually implemented as pass gates. These pass gates make the routing delay dominate the logic delay in FPGAs.

This dominance of routing delay has influenced architecture decisions of FPGAs. While it is not possible to reduce the routing delay beyond a certain amount, most FPGA architectures attempt to at least make the routing delays highly predictable leading to physical design algorithms that thoroughly exploit this characteristic.

## 2.3 Software Complexity

One of the interesting requirements on FPGA physical design software stems from the fact that most system designers designing ASICs are primarily logic designers. Consequently, separate teams of dedicated engineers handle the physical design aspect of the ASIC design. These separate teams have gathered significant expertise in physical design over time and are expert users of the ASIC physical design software. On the other hand, designers using FPGAs do not have separate teams of engineers dedicated to handling the physical design. Instead, the system designers have to focus on both the logic and physical design aspects of their design. This in no way implies that ASIC designers are not concerned with physical design; they expect physical design concerns and try to mitigate them with their design. FPGA designers, on the other hand, do not expect physical design issues to crop up in their design and expect it to be a fairly "hands-off" process. This situation demands that FPGA physical design software be made as easy to use as possible.

Another factor that forces FPGA physical design software to be simple and require less support is the economics of FPGA software. Given the relatively low cost of FPGA software compared to ASIC design software, the support costs account for a large fraction of the overall cost. This imposes the requirement that FPGA software need as little support as possible. This requirement for FPGA physical design software manifests itself as a tendency to hide a lot of the tool and algorithm complexities from the user.

## 2.4 FPGA Device Densities

One of the important reasons that FPGA physical design software differs from ASIC software is that like mask programmed gate arrays, FPGAs are available only in certain vendor determined gate densities. There isn't a continuum of devices that can be used depending on the size of a given design. This implies that the amount of logic and routing available in an FPGA device is fixed and pre-determined. Even the least bit of over-utilization of device resources forces the user to migrate to the next larger FPGA device, and consequently, increases the dollar cost for the design significantly.

Another way to view this phenomenon is that the marginal cost of using additional resources in an FPGA is zero unless the total demand of resources exceeds capacity of the FPGA, in which case the marginal cost jumps sharply reflecting the difference in the cost of the next FPGA device. This implies that it is not necessary for FPGA physical design software to minimize the resource usage if the capacity limit for the FPGA device is not reached. On the other hand, if the design must be developed with future additions and modifications in mind, it becomes important to minimize resource usage to allow for future expandability.

Consequently, FPGA software must not simply minimize the number of resources (area). Instead, it must focus on the optimal use of the existing resources.

## 2.5 New Architecture Development

To improve the runtimes of FPGA CAD tools and to improve the performance of the implemented designs on new FPGA architectures, FPGA vendors develop CAD tools in parallel with new architectures. Vendors will typically model architecture features in the software and evaluate them even before committing them to silicon. In fact, FPGA software is usually available much earlier than the FPGA silicon. This is in complete contrast to ASIC vendors where the process and the new generations of ASICs are developed with minimal, if any, involvement from the EDA tool vendors.

New FPGA architectures are evaluated for cost, routability, and performance with the help of the FPGA CAD tools. In fact, on several occasions specific features are added or removed from the architecture to facilitate faster runtimes or better performance from the FPGA CAD tools.

This requirement for concurrent development of the architecture and the FPGA CAD tools has important implications for FPGA CAD tools. For example, for new architecture evaluation, FPGA software should include general algorithms that can be easily modified. This is preferred to specialized heuristics that are heavily tuned to specific architectures. While the architecture development CAD tools may be internal tools and not visible to the FPGA user, they are, nonetheless, important in our discussion

since they have greatly influenced FPGA software and architecture.

## 3. FPGA Physical design Software

A typical FPGA design flow is shown in Figure 2. The FPGA implementation flow can divided into 3 main phases, design entry which includes HDL or schematic entry mechanisms; the design implementation phase which typically consists of synthesis and technology mapping, followed by placement, routing, and bit-stream generation; and the design verification phase which consists of various simulation and verification tools.

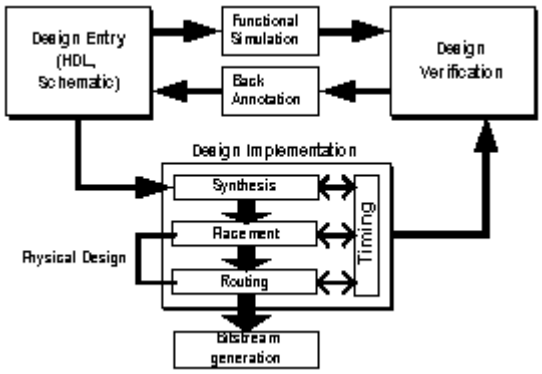


**Figure 2. FPGA Design Flow.**

The design entry phase is identical to the ASIC design flow. Considering the high gate densities of contemporary designs, hardware description languages such as Verilog and VHDL are the current method of choice for design entry. The design verification phase is also similar to the ASIC design flow. Design verification in the form of formal verification or functional simulation can be done directly on the design entry. On the other hand, verification such as back annotation and timing simulation can also be performed on the implemented design.

The part of the FPGA flow that we will concern ourselves in this paper is the design implementation phase. The design implementation phase can be mainly divided into synthesis, placement, and routing. FPGA synthesis tools have been traditionally developed by synthesis vendors rather than the FPGA vendors. On the other hand, as discussed earlier, the FPGA vendors themselves have been the primary developers of the physical design tools such placement, routing and bit-stream generation.

In the design implementation phase, the first task is of synthesis and technology mapping. An input HDL description is synthesized and mapped into logic elements such as Lookup-tables (LUTs), Flip-flops, I/O blocks etc. that are the basic building blocks of the target FPGA architecture. The resulting netlist consists of these logic elements connected together to implement the user design.

This netlist is used as the input to the placement tool that places these elements on the FPGA sites that implement these logic elements. After all the logic elements are placed appropriately on sites on the FPGAs, they are connected together by the routing tool. Once the placement and routing is completed, the FPGA is configured to implement the design. The placement of the logic elements in the design net-list on the logic element sites on the FPGA dictates the configuration of those logic element sites. Similarly, the routing of the logic elements implies that specific routing resources are configured in order to achieve the required connections.

### 3.1 FPGA Model

Before we discuss more details of the physical design flow, let us first describe a traditional FPGA. The traditional model of the FPGA is shown in Figure 3.

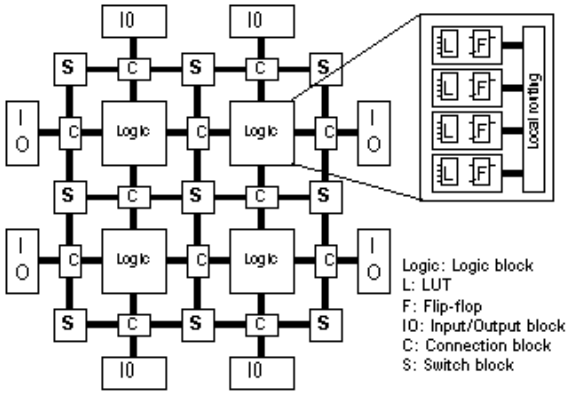


**Figure 3. Traditional FPGA Architecture**

As shown in the figure, an FPGA consists of a 2-dimensional array of logic blocks. These logic blocks can be further decomposed into a hierarchical collection of different logic elements such as LUTs, Flip-flops and Muxes etc. The figure shows each logic element consisting of 4 sub-blocks where each sub-block consists of a 4-input LUT and a Flip-flop. There is a local routing network within the logic block that provides very fast and almost complete connectivity with all other logic elements within the logic block. On the periphery of the FPGA are the programmable I/O blocks through which the FPGA connects to the external world.

Connecting these logic and I/O blocks is a mesh of uncommitted routing resources that can be programmed to achieve different connections. The routing fabric is represented as a set of routing resources and a set of switch and connection blocks. Connection blocks connect the routing resources to the pins of the logic block while the switch blocks connect different routing resources that are incident to the switch block. Typically, there is a hierarchy of routing resources i.e., some resources can connect to switching and connection blocks in adjacent tiles, others can connect to blocks that are a specific distance apart, and some can connect all the blocks in the same row or column on the FPGA.

While FPGA architectures differ in the kinds of logic elements, the number of logic elements, the amount of routing resources, and the routing fabric, they can be abstracted down to the model shown in Fig. 3.

### 3.2 Placement

As discussed earlier, synthesis creates a netlist consisting of a set of logic elements and a set of connections between them. Since, the synthesis step involves technology mapping, these logic elements can be directly mapped to the logic element resources on the FPGA device. Given a list of logic elements, connected to each other by nets, the placement problem can then be defined as placing these logic elements on the available logic element sites on the FPGA such that the connections between them, as specified

by the nets, can be routed completely using the available routing fabric of the FPGA. As one can see, it is almost identical to the classical ASIC placement problem.

Let us now take a detailed look at the objective functions, constraints, and the algorithms for FPGA placement.

### *3.2.1 Placement Metrics*

FPGA placement must place all the instances such that they can be completely routed while achieving the required timing constraints. Typically, this can be achieved with the use of classical ASIC placement objective functions such as minimizing the total wire-length and reducing the maximum congestion.

While complex placement objective functions can be used to closely model the routing fabric, it is generally found that the most efficient metrics for FPGAs are those that are simple yet quite accurate. Therefore, bounding box wire-lengths, cut numbers, and simple congestion metrics are popular FPGA placement metrics. Complex and computationally expensive cost functions are almost always avoided due to the overarching goal of faster run-times. This preference for simple yet reasonably accurate objective functions is even reflected during new FPGA architecture development where one of the primary goals is to allow placement to use simple and fast metrics to accurately reflect the placement on the FPGA architecture.

One of the interesting ways FPGA placement differs from ASIC placement is in their use of routing delay estimation. Unlike ASICs where estimating routing delays may involve detailed RC-tree analysis and computations, FPGA placement algorithms usually use much simpler methods that are unique to FPGAs. In FPGAs, the routing fabric dictates that for optimal routing of a connection, a certain number and type of routing resources must be used. Critical signals must be routed using these optimal routing patterns for best performance. Other non-critical signals may be routed using different routing resources. During placement, it is possible to pre-compute the routing delay for the critical signals based on these optimal routing patterns. For the remaining non-critical signals, a similar approach of pre-computing typical delays can be used. This method leads to very fast yet very accurate routing delay estimation.

An important distinction of FPGA routing delays is that, unlike ASICs, they are non-continuous in nature. This means that they are not necessarily proportional to the length of the connection. This is due to the presence of fixed length routing resources in the architecture.

Since computing fast and accurate routing delays during placement is very essential to achieve good placement results, FPGA vendors pay special attention to this aspect and design architectures where the routing delay estimation can be made more predictable and fast.

### *3.2.2 Placement Algorithms*

FPGA placement uses a variety of different algorithms similar to the ones available for use in ASIC placement. Iterative algorithms such as simulated annealing algorithms are very popular primarily due to the ease with which complex FPGA constraints can be modeled. Another reason iterative algorithms are widely used in FPGA placement is that these algorithms can be easily modified to trade-off execution time and quality. As we discussed earlier, depending on the design phase, the placement algorithm might be either required to run quickly and produce a reasonable result or to produce superior results with longer runtimes.

Typically, the logic elements of FPGAs are arranged in a hierarchical fashion. For example, in the Xilinx Virtex™ family of FPGAs, a combination of 2 look up tables (LUTs) and 2 flip-flops is referred to as a slice. Two such slices make up a configurable logic block (CLB), and the entire FPGA consists of a 2-dimensional array of CLBs. Logic elements at different levels in the hierarchy have different connectivity and configuration specifications. For example, the reason that LUTs and flip-flops are grouped together in a slice is to allow for efficient registering of combinational logic functions. Additionally, the control signals to the LUTs and flip-flops in the same slice may be shared and their configuration may be constrained. This means that not all combinations of LUTs and flip-flops can be placed in a single slice.

An important decision that must be made for FPGA placement is its unit of placement. If the unit of placement is too fine e.g. LUTs and flip-flops, placement will have to deal with a very large number of movable objects which in turn will have a deleterious effect on runtime. Additionally, a coarser unit of placement such as a CLB may not give enough flexibility to attain good results. Table 17 shows the number of LUTs/Flip-flops, slices, and CLBs in the largest family members of the Virtex-E™ and Virtex-II™. There are a couple of things that can be noted from this table: the largest FPGA (Virtex-II, XC2V10000) has close to quarter million LUTs and Flip-flops, and the largest FPGA is at least an order of magnitude larger than the smallest FPGA in the family.

**Table 1. Placement Entities in FPGAs**

| Device | Array | LUTs,FFs | Slices | CLBs |
|---|---|---|---|---|
| XCV50E (Virtex-E) | 16x24 | 3072 | 768 | 384 |
| XCV3200E (Virtex-E) | 104x156 | 129792 | 32448 | 16224 |
| XC2V40 (Virtex-II) | 8x8 | 1024 | 256 | 64 |
| XC2V10000 (Virtex-II) | 128x120 | 245760 | 61440 | 15360 |

The logic element hierarchy is used extensively in the choice of the placement algorithm. For example, min-cut algorithms can be used very effectively for placement of hierarchical FPGAs [7]. By using clustering and changing the unit of placement, min-cut algorithms can yield very good placement. Additionally, even simulated annealing algorithms can employ hierarchical clustering techniques to make use of the inherent hierarchy of logic resources.

While basic FPGA placement can be tackled with the standard set of ASIC placement algorithms, there are some FPGA specific constraints that require variations of these standard algorithms. For example, the presence of architectural constraints that restrict how I/Os can be configured along banks on a side of the Virtex-E FPGA presents a placement problem that is solved using modifications to a basic simulated annealing algorithm [6].

## 3.3 Routing

The FPGA routing problem can be defined as the problem of choosing specific FPGA routing resources to achieve the connections specified in the net-list while meeting the user's timing constraints. The typical model for FPGA routing is illustrated in the following figure.

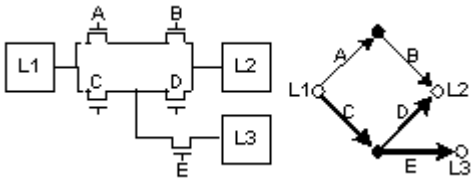


**Figure 4. FPGA Routing Model**

The FPGA is represented as a connectivity graph where the nodes of the graph are the routing segments while the edges are the programmable interconnect points (PIPs). In the example shown, there are 5 PIPs corresponding to 5 possible edges, and 5 nodes corresponding to the routing segments. This is the underlying connectivity graph. A net that connects L1, L2, and L3 can be routed by programming the PIPs C, D, and E. The associated routing graph for this route is shown in the figure as the dark edges on the underlying connectivity graph.

Some of the programmable interconnect points in the FPGA are pass transistors while others are buffered switches. Since the positions of these buffered PIPs are pre-determined by the architecture, FPGA physical design algorithms cannot insert buffers where required, instead they have to judiciously use buffers where they are available in the FPGA routing fabric. This means that normal ASIC buffer insertion methods don't apply to FPGAs.

The size of the routing connectivity graph can be extremely large. Figure 5. shows the total number of nodes and arcs vs. the number of LUTs in the Virtex-II series of FPGAs. Note that the largest Virtex-II FPGA, the XC2V10000 contains close to 60 million arcs and 6 million nodes in the routing connectivity graph. A design that utilizes a high percentage of an FPGA may use as many as 25% of these arcs and nodes. Such large graphs impose serious restrictions on how the connectivity graph can be manipulated by the routing algorithm.

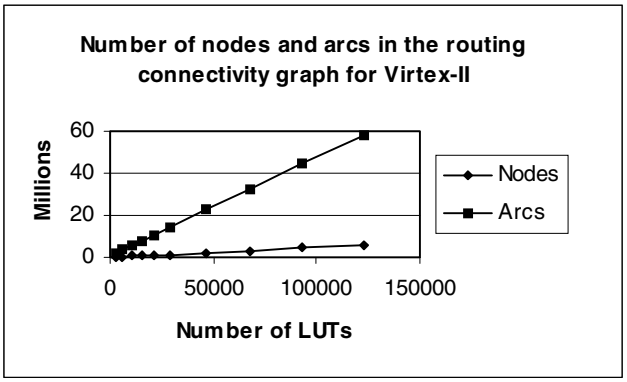


**Figure 5. Size of the Routing Connectivity Graph in Virtex-II**

This representation of the routing graph in FPGAs gives rise to an important distinction between ASIC and FPGA routing algorithms. In ASIC routing, a route is expressed in terms of the underlying rectilinear grid that is sometimes referred to as the Hanan grid. As we have seen, an FPGA does not have a rectilinear grid and the FPGA routing problem, therefore, becomes a problem of embedding the net-list onto this connectivity graph without using the same node twice. This implies that the standard rectilinear grid based routing algorithms must be modified to handle generalized graphs. For example, finding a Steiner point in ASICs implies a rectilinear Steiner point while on an FPGA it implies finding a Steiner point on a general graph.

Routing for ASICs is performed using a two-phased approach. A global routing phase precedes the detailed routing phase. The global routing phase abstracts the details of the routing problem into regions and performs coarse routing on these regions. This is followed by the detailed routing phase that uses the results of the coarse routing and performs detailed routing within the regions.

The key assumption that allows for this two-phased approach is that the regions (typically rectangular channels) are a good abstraction of the underlying routing problem. With FPGAs, this assumption that the coarse routing regions are a good abstraction of the underlying routing structure is not always valid. This leads to the possibility that the coarse routing determined by global routing may not be accurately refined into the underlying detailed routing. Consequently, the two-phased approach of global followed by detailed routing does not apply to all FPGA architectures.

In FPGA routing, instead of a global routing phase, there is often a phase that is referred to as global resource assignment. Some routing resources in the FPGA fabric are designed for special kinds of nets. For example, FPGAs may have special routing structures that allow for low skew and delay when high fanout nets are routed on them. The global resource assignment phase attempts to recognize nets that ought to use global routing resources on the FPGA and assigns them optimally to these global resources.

Typically, this phase is followed by a single detailed routing phase. However, some of the common approaches to detailed routing in ASICs are either not applicable at all (channel routing) or are not very suitable (maze routing). Since the concept of a channel is not explicit in the FPGA, channel routing algorithms are not employed in FPGA routing. While maze routing algorithms are applicable to FPGA routing, they can be inherently slow. Another issue with maze routing is that it does not consider the side effects on other connections i.e., it is net ordering dependent.

Detailed routing algorithms in FPGAs are based on some modification to basic maze routing algorithms. Maze routing and wavefront expansion techniques are employed on the connectivity graph. Different heuristics such as future costs computations, and partial wavefront expansion etc., are employed to speed up the basic maze routing algorithm. However, one of the main drawbacks of using the maze routing algorithm is the fact that it is dependent on the order of routing of nets. For example, a net being routed first does not consider the effect of its routing on the routing of subsequent nets. This problem is exacerbated in FPGAs due to the relatively scarce amount of routing resources.

One very popular FPGA routing algorithm that minimizes the negative effects of the net ordering problem is the Pathfinder [2] algorithm. It is very well suited to FPGAs since it adapts very well to the FPGA connectivity graph. In this algorithm, individual connections are routed to minimum cost on the FPGA connectivity graph; once a connection is routed, the routes are recorded and the connection is then ripped out. This procedure is repeated for the next connection and so on. In a single iteration of this algorithm, each connection is routed in this fashion as if it were the only connection to be routed. In effect, this is equivalent

to routing every connection in the absence of any existing routes or obstacles.

After all the nets are routed once i.e. after a single iteration, the demand for every resource on the FPGA is computed. The demand for a resource is computed as the number of nets that used that resource to complete a route. A demand of 1 on a routing resource implies that only one net required the use of that resource to complete its route. In effect, there is no conflict for the use of the resource. However, a demand greater than 1 implies a routing conflict i.e., more than 1 net requires the use of the resource for a minimum cost route.

In subsequent iterations, the cost of a resource that has high demand is raised and the entire process of routing connections individually is repeated. Raising the cost of resources that have heavy demand ensures that some of the nets that used the resource in previous iterations will complete their routes by using less expensive nodes. The iterations continue, with the costs of the resources having heavy demand getting progressively higher. Routing is complete when the demand for all resources is no greater than one.

This algorithm addresses the slow speed of maze routing since it requires every connection to be routed in an obstacle free environment. Additionally, since every net is routed several times to account for heavily used resources, it avoids the net ordering problem inherent in maze routing.

The presence of the connectivity graph with a finite number of nodes and edges has given rise to some new formulations of the FPGA routing problem. Recent approaches ([4],[5]) attempt to formulate the FPGA routing problem using Boolean Satisfiability. In this approach, variables in the SAT problem instance correspond to the allocation of specific resources to nets. Routing is completed when a set of values can be assigned to the variables causing the SAT problem to evaluate to TRUE. While this approach is not used in practice yet due to runtime and memory concerns, it does, however, present a unique perspective to the FPGA routing problem.

## 3.4 Physical Synthesis

No discussion of physical design can be complete without a discussion of physical synthesis. Physical synthesis for FPGAs must primarily deal with the dominance of routing delay over the logic delay. Other deep sub-micron effects such as signal integrity need not be considered due to reasons discussed earlier. Physical synthesis for FPGAs is a relatively new area of research though the concept of dominant routing delays has been as old as FPGAs itself.

In earlier days, to compute post routing delays, the delays were primarily based on the fan-out of the nets and were statistically estimated. Typically, several designs were placed and routed and the routing delays measured. These measurements were then used to formulate a statistical estimate of the routing delays of nets based on their fanout. This method was notoriously inaccurate and would result in underestimation of routing delays sometimes by 300 to 400%.

While the basic approach of statistically estimating post-routing delays remains the most often used method, one approach [8] below being investigated in physical synthesis by the synthesis vendors is a “two pass” method. In this method, the first pass of synthesis is done independently and followed by the regular place and route. The placed and routed design is then evaluated for feedback to another pass of synthesis. This feedback is used to modify the re-synthesis process to account for the routing delays.

## 3.5 Results

Table 2 illustrates a sampling of results of the Xilinx physical design software (PAR) for a range of Virtex-E devices run on SUN UltraSparc-II. The runtimes are measured when the software is able to achieve the user defined frequency requirement. For the smallest device, the place and route runtimes are less than a minute. On the other hand for the XCV2000E which has 38400 LUTs, the total runtime is under an hour.

**Table 2. Results of Xilinx PAR**

| Design | Slices | Nets | Runtime | | Frequency (Mhz) | |
|---|---|---|---|---|---|---|
| | | | Placement | Routing | Req. | Act. |
| XCV50E | 766 (99%) | 1615 | 00:00:45 | 00:00:50 | 6.67 | 8 |
| XCV1000E | 8635 (70%) | 17468 | 00:09:59 | 00:08:02 | 68 | 72 |
| XCV2000E | 14037 (90%) | 27639 | 00:22:05 | 00:22:27 | 47 | 48 |

It must be pointed out that while the XCV50E design (which is a glue logic design) has a small frequency requirement, it has a large number of logic levels (69). The runtimes, of course, increase as the frequency requirement becomes more stringent or as the design gets more congested.

# 4. Conclusions

FPGAs have evolved into very popular vehicles for designing systems. The increasing numbers of system-level features and user programmable gates have contributed to their popularity. There are some fundamental differences between the ASIC and FPGA design flow. Consequently, FPGA physical design software has evolved differently from ASIC physical design software though their roots are common.

The FPGA design methodology and requirements have resulted in a different set of goals and objectives for FPGA physical design software. Runtime is of supreme importance due to the fast time-to-market value proposition of FPGAs. At the same time, in some situations fast FPGA circuit speeds may become the primary goal of FPGA physical design software at the expense of runtime. This has resulted in the use of algorithms that can be tuned for speed and performance. Due to the design of the FPGAs, deep sub-micron issues such as signal integrity and cross talk do not have to be considered by FPGA physical design software. However, another deep sub-micron issue relating to the dominance and unpredictability of routing delays has been an FPGA characteristic since the invention of FPGAs. FPGA vendors have been addressing this issue through innovative FPGA architectures that improve routing predictability.

FPGA placement algorithms are very similar to classical ASIC placement algorithms. However, they use routing delay estimation strategies that are unique to FPGAs. On the other hand, the FPGA routing problem is very different from that of ASICs due to the underlying representation of the FPGA routing graph. Traditional maze routing algorithms are not very well suited to FPGA routing, but variations on the basic maze routing algorithm such as PathFinder have proven very popular.

# 5. REFERENCES


[1] V. Betz and J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGA Research,” Proc. 7th Intl. Workshop on Field-Programmable Logic and Applications, 1997.

[2] L. E. McMurchie and C. Ebeling, “PathFinder: A Negotiation-Based Path Driven Router for FPGAs”, Proc. ACM/IEEE Intl. Symp. on FPGAs, 1995.

[3] S. K. Nag and R. A. Rutenbar, “Performance-Driven Simultaneous Placement and Routing for FPGAs”, IEEE Trans. on CAD, pp 499 – 518, June 1998.

[4] G. Nam, K. A. Sakallah, and R. A. Rutenbar, “Satisfiability-Based Layout Revisited: Detailed Routing of Complex FPGAs Via Search-Based Boolean SAT”, Intl. Symp. on FPGAs, 1999.

[5] G. Nam, F. Aloul, K. A. Sakallah, and R. A. Rutenbar, “A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints”, Proc. Intl. Symp. on Physical Design, 2001.

[6] J. Anderson, J. Saunders, S. Nag, C. Madabhushi, R. Jayaraman, "A Placement Algorithm for FPGA Designs with Multiple I/O Standards", Proc. 10th Int. Conf. on Field-Programmable Logic and Applications, August 2000.

[7] M. Hutton, K. Adibasmii, and A. Leaver, “Timing-driven Placement for Hierarchical Programmable Logic Devices”, Proc. Intl. Symp. on FPGAs, 2001.

[8] http://www.synplicity.com/products/amplify.html

[9] http://www.xilinx.com/partinfo/databook.htm

[10] “Designers roadmap to system-level integration”, Gartner Group Inc.