Area and Delay Trade-offs in the Circuit and Architecture Design of FPGAs

Ian Kuon and Jonathan Rose
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON
{ikuon,jayar}@eecg.utoronto.ca

ABSTRACT

Field-programmable gate arrays (FPGAs) are used in a wide range of markets that have differing cost, performance and power consumption requirements. It would be advantageous if a single device family could serve these varied needs but the economics of catering to this wide distribution of market demands suggest more than one family is appropriate. Consequently, FPGA vendors have moved to provide a more diverse set of families that sit at different points in the area-speed-power design space.

In this work, our goal is to understand the circuit and architectural design attributes of an FPGA that enable trade-offs between area and speed, and to determine the magnitude of the possible trade-offs. This will be useful for architects seeking to determine the number of device families in a suite of offerings, as well as the changes to make between families.

We have found that varying both architecture and transistor sizing of an FPGA allows the effective area to change by a *factor* of 3.6 from largest to smallest and the speed to change by a factor of 2.6 from fastest to slowest. It is interesting to observe that the range of area and delay trade-offs possible by varying only the transistor sizing of a single architecture is larger than the ranges observed in past architectural experiments. In addition to transistor size, we note that LUT size is one of the most useful parameters for trading off area and delay.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles

General Terms

Design, Measurement, Performance

Keywords

FPGA, Architecture, Optimization

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) have evolved to the point that they are now used in a wide range of markets including consumer electronics, automotive, industrial and high-performance computing in addition to their original primary market of communications. These markets often have very different needs with some requiring the best performance while others are more focused on minimizing cost. These differing requirements make it difficult for a single FPGA family to serve these different market needs. As a result, industry practice has moved to provide different FPGA families to cater to these different market needs. It is now common for FPGA manufacturers to offer a high end, high performance family [2, 15, 32] and a lower cost, lower performance family [3, 16, 31].

This trend is almost certain to continue as new processes require FPGA architects to make increasingly difficult design choices between between cost, performance and power consumption. These choices can dramatically affect the gap between FPGAs and both the full or partially-fabricated application-specific integrated circuits (ASICs) with which they compete. For example, we reported that a pure soft logic FPGA (with no hard memory or other blocks) is 35 times larger, 3 to 4 times slower and consumes 14 times more dynamic power than the equivalent standard-cell ASIC implementation [14]. Given this large gap, the ability to trade-off one attribute for another is particularly important and it would be useful to understand the extent to which any one of the area, performance or power gaps can be narrowed. For different markets, area, performance and power are of different importance and closing one of the gaps could be essential. By exploring the trade-offs possible the limits for FPGAs can be better understood since that will demonstrate the extent to which any of the gaps can be narrowed. Also, if embedded FPGAs become viable, there will be a further increased need for FPGAs that make different cost, performance and power trade-offs.

There has been little exploration of the extent to which area, speed and power can be traded. Past studies have focused almost exclusively on high-level logical architecture changes such as changes to the routing [18], logic block [1, 21] or both [7, 8]. However, the high-level logical architecture is only one variable in the design of an FPGA that can be varied. For every architecture, there are a range of possible electrical implementations that trade cost, performance and power through the use of different circuit structures or transistor sizings which has been largely ignored in past studies [18, 1, 7]. Some electrical design issues such as VDD and V_t optimization have been explored [8] but again little attention has been paid to the issues of transistor sizing and circuit structure. Instead, it has been assumed (possibly for simplicity) that there is only a single circuit structure and

FPGA'08, February 24–26, 2008, Monterey, California, USA.

transistor sizing of interest such as that which minimizes the circuit's area delay product. However, as is often seen in the custom design world, there are a range of logically equivalent but electrically distinct implementations [24]. We believe the same holds true for programmable circuits and, in this paper, we explore the range of area and delay trade-offs that are possible in the design of an FPGA by varying both its logical architecture and its electrical implementation. This exploration can inform architects the extent to which area and delay can be improved for FPGAs and is a step towards understanding how many different FPGA families are necessary. Together, architecture and electrical implementation provide significant leverage that can significantly alter either the performance or cost of an FPGA.

Full manual exploration and optimization of designs within this space is not feasible and, therefore, we have developed a tool to perform much of the optimization. This tool is briefly described in a subsequent section. It is important to note that we aim to observe the size of the design space for *general-purpose* FPGAs. Another degree of freedom in optimization would be to create FPGAs that are heavily tailored towards specific application domains [9]. However, we believe there is a need for different general-purpose FPGAs that occupy different points within the design space.

The remainder of the paper is organized as follows. A brief background and the basic architectural and circuit assumptions on which this work is based are presented in Section 2. Section 3 describes the FPGA-specific transistor-level optimization tool that was developed to assist in exploring the design space. Section 4 outlines the procedure used to measure performance and area within the design space. Section 5 examines the range of trade-offs possible with transistor sizing. The impact of transistor sizing in conjunction with architectural changes is then explored in Section 6 to determine both the magnitude of changes possible and to determine the parameters that provide the most leverage when making area and delay trade-offs. Finally, Section 7 concludes.

2. ARCHITECTURAL AND ELECTRICAL DESIGN ASSUMPTIONS

To keep this work tractable, we place some limitations on the design changes we will explore. First we will focus only on area and delay trade-offs. Power consumption trade-offs will not be considered. This is reasonable since we have confirmed, as has been previously reported [21], that power consumption is closely related to area for many architectural changes. Techniques such as power gating [8] can alter the relationship between area and power consumption but these techniques are not supported by our current computer-aided design (CAD) tools.

Additionally, we will restrict the architecture and circuit structures we will consider. The assumptions we make are described in the following sections.

2.1 Logical Architecture

We focus exclusively on the classic island-style FPGAs consisting of a Cluster-based Logic Block (CLB) surrounded by programmable routing. This structure and the main architectural parameters are shown in Figure 1. We further limit ourselves to a homogeneous routing topology in which all the routing tracks are unidirectional as described in [34, 18, 20] and have the same length. The length of a track, L, is defined as the number of logic blocks it reaches. We assume that there is an equal number of tracks in the horizontal and vertical directions and we refer to this quantity as the channel width, W. A fraction of the tracks in a channel, $F_{c,input}$, connect to each of the logic block's input pins.

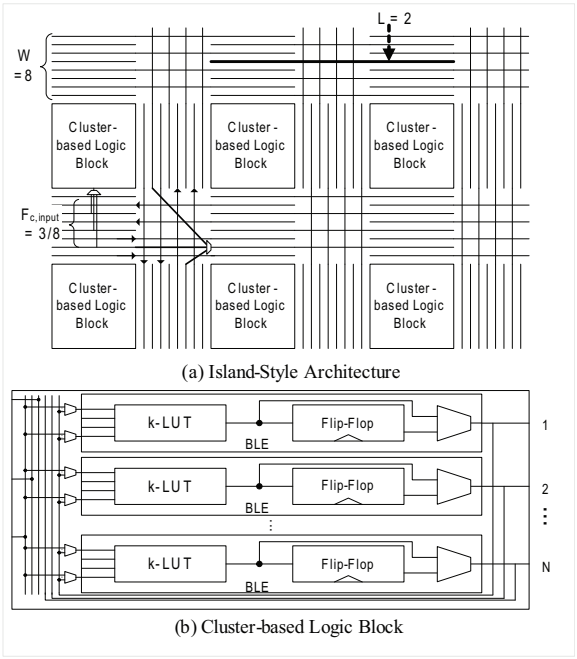

Figure 1: FPGA Logical Architecture

Logic blocks are composed of one or more Basic Logic Elements (BLEs) and each BLE is made up of a lookup table (LUT) of size k and a flip-flop. The number of BLEs in a logic block is defined as the cluster size, N. When logic blocks contain more than one BLE, programmable intra-cluster routing connects the logic block inputs to the inputs of each BLE. The intra-cluster routing is assumed to be fully populated as each BLE input is able to connect to all the logic block inputs and all the BLE outputs. All these assumptions and parameters define the *logical* architecture of the FPGA.

2.2 FPGA Tiles

A single logic block and its neighbouring routing channels must be instantiated thousands of times to create a complete FPGA. It is not practical to individually optimize each logic block and routing track. Instead, in this work a single *tile* consisting of a logic block and the neighbouring routing tracks is designed. When creating a complete FPGA that single tile is replicated.

This use of a single tile places restrictions on the logical architectures that can be explored. In particular, the channel width is restricted to multiples of twice the segment length [18]. (For bidirectional routing tracks, the channel width must be multiples of the segment length.) This quantization of the channel width ensures that every tile is identical with an equal number of routing tracks starting and stopping in each tile. When determining the architectural parameters for our experiments we ensure this quantization is maintained.

2.3 Circuit Assumptions

At the circuit level, the FPGA architectures we will consider consist purely of multiplexers, inverters, configuration memory, and user-circuit flip-flops. The flip-flops are a relatively small part of the design that does not significantly affect an FPGA's performance or area. Therefore, we do not investigate the range of possible flip-flop implementations. The remaining structures all make up the majority of the FPGA's area. The design of the configuration memory and the inverters is straightforward. For the configuration memory, a standard 6-transistor SRAM cell is assumed. However, multiplexers have a range of possible electrical implementations. We assume multiplexers are constructed using NMOS pass transistors. To restore signals to the full rail, a level restoring PMOS is added to the inverters connected to the multiplexer output. A one-level NMOS pass transistor tree is assumed for multiplexers with a width less than 4 (i.e. a one hot encoding) and a two-level NMOS tree is assumed for all larger multiplexers as described in [19, 17].

To keep the scope of this work reasonable, we will only consider changes in transistor sizes within these previously described circuit structures. We will not consider threshold voltage or supply voltage changes such as those in [8] since these are only useful for power trade-offs.

The high-performance 90 nm CMOS process from ST Microelectronics [27] will be used exclusively in this work. We use only standard V_t transistors and assume a supply voltage of 1.2 V. In the future, we plan to consider more advanced process technologies.

2.4 Comparison to Commercial Architectures

While the architecture of modern FPGAs is significantly more complex with multiple types of routing segments, logic blocks with more features such as adders [32, 2] and various different types of logic blocks such as multiplier [32, 2], memory [32, 2] and processor [32] blocks, focusing on the comparatively simple architecture described above is still useful for exploring the area-delay trade-offs we will consider in this work. Despite the new features, the basic LUT and flip-flop is still crucially important as it gives an FPGA its general-purpose capabilities and the trade-offs made in the design of that basic logic and routing continue to have a significant impact on the overall area and performance of FPGAs. If anything, the additional architectural features found in modern devices have the potential to further expand the range of trade-offs possible when designing an FPGA since with each new component comes the capability to adjust its performance or area.

3. TRANSISTOR-LEVEL OPTIMIZATION TOOL

When designing an FPGA at the electrical and architectural level, there are inherent trade-offs between area and delay that must be made. Our goal is to explore these trade-offs and examine what we call the area-delay design space for FPGAs. These explorations require a multitude of different combinations of logical architecture and electrical design of the circuits in that architecture to be considered. It is not feasible to manually size the circuitry for each possibility as has been done in past architectural experiments [7, 1]. Instead, we have developed a transistor-level optimization tool to assist in this exploration and, in this section, we will give a brief overview of this tool. With this tool, it is possible to specify a logical architecture and some high-level electrical parameters as inputs and receive as an output the transistor sizes for a range of different circuit-level implementations at different points in the area-delay space.

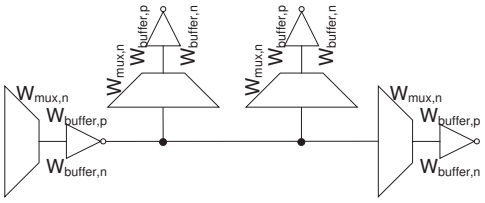

Figure 2: Routing Track Sizing

3.1 FPGA-Specific Optimization Issues

The transistor-level optimization of custom (non-programmable) integrated circuits has been well studied and a variety of approaches have been proposed to automate this process [12, 26, 11]. Programmable circuits and, FPGAs in particular, present unique optimization challenges. The most significant is that, due to the programmability, it is not known what end-user circuit will be implemented on the FPGA. This means that the critical path is not known at design time (of the FPGA itself), and, therefore, improving the performance of the circuit is no longer straightforward because different circuits may place different demands on the various elements within the FPGA.

The second unique feature that must be considered in the design of FPGAs is the large number of logically equivalent components. All these equivalent components must be sized identically to maintain their equivalence. As a result, the designer no longer has the freedom to increase the size of one component to improve performance and, instead, all similar components must be increased in size. An example of this is a routing track and the multiplexers that select the signal driving this track as shown in Figure 2. For performance reasons, increasing the size of the transistors in the multiplexer (up to a point) would be advantageous but, because this same multiplexer also loads the routing channel in more cases than it is used, increasing the transistor sizes may not provide the desired reduction in overall delay. We refer to this effect as *logical self loading* and we note that similar issues are faced in other custom designs that involve repeated circuit structures such as memories. It is possible to alter these logical equivalency requirements by creating different architectures with distinct classes of switches [6, 13]; however, for transistor-level optimization, we treat logical equivalency as a fixed constraint.

This characteristic of repeated structures within the FPGA circuit design also simplifies the optimization problem since there are many fewer independent variables compared to the total number of transistors. This reduction is significant since an FPGA has hundreds of millions of different transistors that can be on a user's critical path but there are only on the order of a hundred unique transistor sizes that must be optimized.

3.2 Optimization Overview

To handle and leverage these challenges and opportunities, a custom optimization tool was developed. The tool is given as inputs the logical architectural parameters described in Section 2.1, high-level electrical parameters such

as the basic structure to use for multiplexers and the desired optimization objective. Based on these inputs the tool produces the optimized transistor sizes for all the components in an FPGA tile. The optimization process is summarized in Figure 3.

The tool sizes the transistors in the FPGA to balance the performance and area of the FPGA according to the desired optimization objective. As described earlier, the performance of the FPGA depends on the users' designs. For the purpose of optimization, it is not possible to directly measure and, hence, optimize that performance. Instead, a representative measure of the design's performance must be created. The approach we use for this measurement is described below. Area is comparatively straightforward to measure and the area measurement methodology is described in Section 4.2; however, during optimization, only the area of a single tile is considered. The complete process described in Section 4.2 will only be used for the final results to reflect the area costs of the transistor sizing and the logical architecture.

3.2.1 FPGA Performance Metric

The speed performance metric for the optimizer must fully capture the performance of all the components of the FPGA. A simple approach to this is to take a set of circuits that make up the critical paths in a collection of benchmark designs. The performance of these circuits could be measured and combined to create the performance metric. However, this approach is inefficient since many of the circuits may be relatively similar. Even taking one critical path circuit could be excessively demanding computationally because the same component such as a LUT may appear in the path multiple times.

To reduce the computational demands, an alternative approach was used for this work. The shortest register to register path within the FPGA that still contains all the unique components within the FPGA tile is used for optimization. The delay of this path is not used as the performance metric since it does not use the circuit components of an FPGA at the same frequency as they occur in end-user's circuits. Instead, the performance metric used for optimization is created by taking a weighted average of the delays of the individual components within this shortest path. The weighting factors are set based on the frequency with which each component was observed to occur in the critical paths in our set of benchmarks.

3.2.2 Optimization Objective

Our goal is to explore the range of trade-offs possible with area and delay and, hence, the optimization tool must produce range of transistor sizings that make these different trade-offs. The tool is driven to create these different designs by varying the objective function that the optimizer attempts to minimize. With different functions, a different balance between the performance of the FPGA and the area required to achieve that performance is reached.

Various forms of optimization objectives were considered for the tool. The final form of the function that was selected was $Area^m Delay^n$ where m and n are real numbers greater than or equal to zero. By varying m and n, different objective functions are created which will lead to a range of different transistor sizings. This form of the function is easy to understand. Setting $m = n = 1$ gives a design that minimizes the area delay product which has conventionally been the goal of circuit design in FPGAs [7, 1].

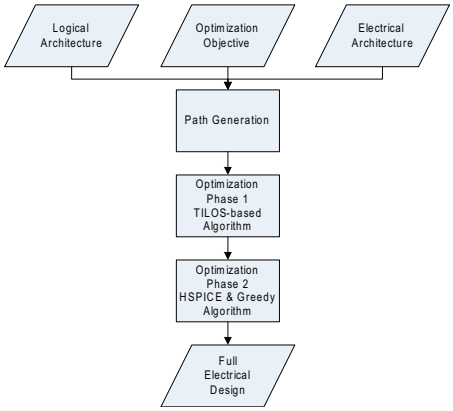

Figure 3: Optimization Methodology

It is also worth noting that in conventional custom circuit design, optimization frequently takes the form of minimizing the area subject to a delay constraint. Such a form is not as useful for FPGA-based optimization again because the performance of the FPGA is not known until user circuits are implemented on the FPGA. Therefore, the absolute value of any delay constraint would not be informative.

3.2.3 Optimization Algorithm

Optimization is performed in two phases. In the first phase, transistors are modelled as simple linear resistances with the resistances inversely proportional to the transistor width. Delay is then computed using the standard Penfield-Rubenstein RC model from [25]. Optimization is then performed using a slightly modified version of the TILOS algorithm [12]. For this delay model, the algorithm should generally be able to achieve an optimal result but it must be emphasized that it is only optimal given the simple RC delay model. Logically equivalent components are defined to have the same transistor sizes and the optimizer always maintains this relationship. Other approaches such as logical effort [28] or related fanout-of-four based sizings could be used but those approaches are not directly suited to handling the effects of logical self-loading.

While the TILOS algorithm can achieve optimality for the simple RC delay model, the linear RC model has long been known to not accurately reflect the behaviour of transistors [23]. Therefore, further optimization, which captures the true behaviour of the transistors, is necessary. For example, compared to an unoptimized design, the sizings produced by TILOS have improved performance by 69 % but with the next phase of the algorithm that will be described below the performance was improved by 77 % overall. Clearly, TILOS delivered most of the gains but a second phase to refine the sizes is necessary.

Using the sizes determined from the TILOS algorithm as a starting point, the second phase of optimization refines the sizes with a greedy iterative sizing algorithm that uses delay measurements from Synopsys HSPICE [29]. As the name implies, this algorithm looks at each parameter (which defines the transistor size for all logically equivalent components) iteratively and explores a range of values for the parameter. The best value is selected and the process repeats

with the next parameter. If the best value of any parameter is changed, the process will repeat over all the parameters again until one complete pass through all the parameters is made without adjusting any sizes. While this simple algorithm produces mediocre results on its own, we found that when applied after the first phase of optimization the result is satisfactory. Testing with simple circuits that could be exhaustively optimized found that this combination gave results that were consistently within 2 % of optimal. The disadvantage of this approach is that significant computing resources are required but this is acceptable since our goal is design space exploration and not fast optimization. In general, we are able to obtain a sizing in under 12 hours.[1] More advanced optimization techniques such as those proposed in [11] would likely enable significant run-time improvements but they either require custom simulators or access to internal computations within standard simulators.

3.3 Optimization Output

The final output from the optimizer defines the transistor sizes of all the components that make up the FPGA logic tile. As described earlier, for each architecture, different optimization objectives are used to produce a range of different sizings that make different trade-offs between delay/speed and area. These sizings only define the electrical implementation of the programmable fabric that makes up an FPGA and, therefore, further steps are required to produce performance and cost measurements of the resulting FPGA. The approach used to take these measurements is described in the following section.

4. AREA AND PERFORMANCE MEASUREMENT METHODOLOGY

The inherent programmability of FPGAs means that until an FPGA is programmed with an end-user's design there is no definitive measure of the performance or area of the FPGA. Only after a circuit is implemented *on* an FPGA is it possible to measure the performance of the FPGA in a meaningful manner. Similarly, determining the effective area of an FPGA also requires the implementation of end-user circuits on the FPGA to accurately determine the resources required. In this section, the specific methodology used to measure the performance and the area of an FPGA implementation is defined.

4.1 Performance

The performance of a particular FPGA implementation is measured experimentally using the 20 largest MCNC benchmarks [33]. Each benchmark circuit is implemented through a complete CAD flow on the input FPGA fabric and a final delay measurement is generated as an output. The geometric mean delay of all the circuits is then used as the figure of merit for the performance of the FPGA implementation. The steps involved in this process are illustrated in Figure 4.

Synthesis, packing, placement and routing of the benchmark circuit onto the FPGA is done using SIS with FlowMap [10], T-VPack [22] and VPR [5] (an updated version of VPR that handles unidirectional routing is used). Placement and routing is repeated with 10 different seeds for placement. Only the placement and routing with the best performance will be used. The tools cannot directly make use of the transistor size definitions of the FPGA fabric and, instead, a simplified timing model must be provided. This timing model is encapsulated in VPR's architecture file and includes fixed delays for both the routing tracks and the paths within the logic block. We generate this file automatically from the transistor size definition.

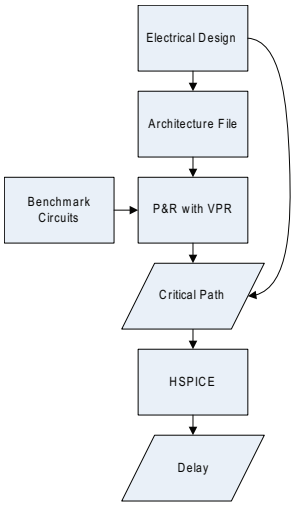

Figure 4: Performance Measurement Methodology

After placement and routing is complete, VPR performs timing analysis to determine the critical path of the design implemented on the FPGA. While this provides an approximate measure of the FPGA's performance, it is not sufficiently accurate for our purposes since the relatively simple delay model does not accurately capture the complex behaviour of transistors in current technologies. To address this, we have created a modified version of VPR that emits the circuitry of the critical path. This circuit is then simulated, with the appropriate transistor sizes and structures, using HSPICE. The delay as measured by HSPICE is used to define the performance of this benchmark implemented on this particular FPGA implementation.

One concern with this method is that routing and timing analysis in VPR is performed with the inaccurate timing model and, as a result, poor routing choices may be made or timing analysis may incorrectly predict the design's critical path. Adequately addressing this problem would likely require simulation of the n-most critical paths in a design to verify the timing analysis results and this quickly becomes computationally infeasible for a reasonable n and a large number of benchmark circuits. Furthermore, if significant discrepancies are observed it suggests that inappropriate routing decisions may also have been made and addressing this challenge would require overhauling the entire timing analysis engine within VPR. However, we do not believe this to be a concern in this work for two reasons. First, we observed for any individual sizing a high degree of correlation between the critical path delays as reported by VPR compared to the critical path delays as reported by HSPICE for the full set of benchmarks. However, for different sizings, the delay measurements between VPR and HSPICE are not as consistent and, hence, the need for HSPICE simulation in

[1]This time could be easily reduced by using fast SPICE simulators such as Synopsys NanoSim [30] but we have elected not to make use of such simulators since run time is not a significant issue

the first place. Secondly, in this work we restrict ourselves to relatively simple routing architectures that are homogeneous. With only a single type of routing interconnect the fidelity of VPR's timing analysis is relatively good.

4.2 Area

The most accurate method for measuring the area of different electrical implementations would be to layout a complete FPGA tile consisting of both the logic and routing. However, this is not feasible since we wish to examine a large number of different architectures and transistor sizings and, instead, we must resort to using a model to determine the area of an FPGA tile. The model considers transistors from two sources: the general circuitry used in the design and the configuration SRAM used to configure the circuitry. The SRAM is treated separately because it is the single most frequently repeated structure in the FPGA. Significant effort is therefore spent optimizing the layout of the 6 transistors that make up a single bit. It is also possible to significantly improve the effective area per bit by merging diffusion regions between bits. In our model, the area of the SRAM component of a design is estimated assuming a fixed area per SRAM bit and this fixed area includes a significant amount of sharing. For the remaining general circuitry, the model estimates the area required for each individual transistor using a method similar to the minimum-transistor-widths approach [7] but with the model re-calibrated to reflect the design rules of our process (STMicroelectronics 90 nm G CMOS [27]). The absolute area is determined by multiplying the minimum width transistor area count by the actual minimum width transistor area. This minimum width transistor area model does not consider the impact of diffusion sharing between transistors or the occasional need to increase the space between devices to allow for inter-transistor routing. To address these inaccuracies we scale the estimated minimum width transistor area estimate by a constant to match the layout area of some sample designs that were created for this purpose.

It was important to obtain a reasonably accurate absolute area measurement and not just one with reasonable fidelity as has been appropriate in the past [7, 1] because the absolute area is needed to determine the length of interconnect segments. For both the routing tracks and the local intra-cluster lines, interconnect is not negligible and, therefore, we use the estimated tile area to determine the length of these lines.

To allow for comparisons across different *logical* architectures (e.g. with different LUT or cluster sizes), the final area metric is the product of the area of an individual tile and the number of tiles (or equivalently clusters) required for all the benchmark circuits. This metric ensures that changes that alter the amount of usable logic are fully reflected in the area measurement. This is particularly important when the number of inputs to a cluster is reduced or when the size of the LUT is changed since those changes alter the amount of usable logic within a tile/cluster.

5. AREA AND DELAY TRADE-OFFS

We now employ the methodology described above to first examine the magnitude of the area and delay changes possible with transistor sizing. Not all trade-offs are useful and, therefore, we discuss and define what we believe to be the boundaries of useful trade-offs.

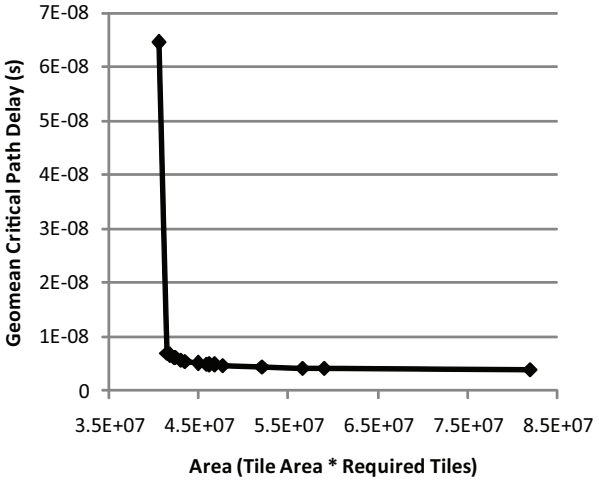

Figure 5: Area Delay Space

Table 1: Architecture Parameters

Parameter	Value
LUT Size, k	4
Cluster Size, N	10
Number of Cluster Inputs, I	22
Tracks per Channel, W	104
Track Length, L	4
Interconnect Style	Unidirectional
Driver Style	Single Driver
$F_{c,input}$	0.2
$F_{c,output}$	0.1
Pads per row/column	4

5.1 Transistor Sizing for a Single Architecture

For any specific architecture, transistor sizing enables a range of implementations between the two extremes of minimum delay and minimum area solutions. The different implementations occupy different points in the area-delay design space. Figure 5 plots these different points that form the delay versus area curve for the cluster size 10, 4-LUT architecture fully described in Table 1. As described in Section 4.1, delay is measured as the geometric mean of the critical paths of the 20 largest MCNC benchmarks [33] when placed and routed on an FPGA with the particular transistor sizing. Area is measured using the model described in Section 4.2 to estimate the size of the tile.

Transistor sizing clearly enables a large range of area and delay possibilities with a range of 2.0 × in area from the smallest to largest design and 16.3 × from the fastest to slowest design. Table 2 compares this area-delay range to the range seen when architectural parameters have been varied in past studies. In each case, the range is measured as the largest area or delay relative to the smallest area or delay observed for the architectures considered. Previously, the largest range was achieved when cluster size and LUT sizes are both varied. In that case, ranges of 3.2 × and 1.7 × were observed in delay and area respectively [1]. While the area range is of a similar magnitude to that seen from transistor sizing, the delay range from architectural changes is considerably smaller than that from transistor sizing indicating

Table 2: Area and Delay Impact of Transistor Sizing and Past Architectural Changes

Variable	Delay Range	Area Range
Transistor Sizing (Full)	16.3	2.0
Cluster Size (1-10) [1]	1.6	1.5
LUT Size (2-7) [1]	2.2	1.5
Cluster & LUT Size [1]	3.2	1.7
Segment Length (1-16) [7]	1.6	1.6

the significant effect transistor sizing can have on performance. It is also important to recognize that architecture and transistor sizing are independent since the transistor sizing for each architecture can be varied to achieve different area-delay trade-offs.

5.2 Reasonable Trade-offs

While the full range of transistor sizing possibilities illustrates the important role sizing plays in determining performance trade-offs, reasonable architects and designers would not consider this full range useful. At the area-optimized and the delay-optimized extremes, the trade-off between area and delay is severely unbalanced. This is particularly true near the minimal area sizing where the large negative slope seen in Figure 5 indicates that, for a slight increase in area, a significant reduction in delay can be obtained. In relative terms, there is a 9.2 × reduction in delay for only a 2.3 % increase in area. Clearly, a reasonable designer would always pay 2.3 % in area to gain the 9.2 × reduction in delay. Our goal, then, is to explore the "*interesting*" and realistic trade-offs between area and delay, and, clearly, regions where the trade-offs are unbalanced are not normally of interest.

Selecting the regions in which the trade-offs are useful is a somewhat arbitrary decision. Intuitively, this region is where the elasticity, defined as

$$\text{elasticity} = \frac{d\text{delay}}{d\text{area}} \frac{area}{delay} \tag{1}$$

is neither too small or too large. Since we do not have a differentiable function relating the delay and area for an architecture, we approximate the elasticity as:

$$\text{elasticity} = \frac{\%\text{ change in delay}}{\%\text{ change in area}}. \tag{2}$$

When the elasticity is -1, meaning a 1 % area increase achieves a 1 % performance improvement, the trade-off between area and delay is certainly interesting. However, based on conversations with a commercial FPGA architect [4], we will view the trade-offs as "interesting" when a 3 % area increase is required for a 1 % delay reduction (an elasticity of -1/3) and when a 1 % area increase improves delay by 3 % (an elasticity of -3). All points within this range of elasticities will make up what we call the *interesting* range of trade-offs. While this restriction only explicitly considers delay and area, it has the effect of eliminating designs with excessive power consumption because those designs would generally also have significant area demands.

With this restriction on the data from Figure 5, the range of trade-offs possible is decreased to a range of 28 % in delay from slowest to fastest and 21 % in area from largest to smallest (The range is expressed here in percent not a multiplicative factor). While this is a significant reduction in the effective design space, the range is still appreciable and it demonstrates that there are a range of designs for a specific architecture that can be useful. Applying this same criteria to the past investigation of LUT size and cluster size [1], we find that the range of useful trade-offs is 17 % from fastest to slowest and 11 % from largest to smallest. This space is smaller than the range observed for transistor sizing changes. From the perspective of designing FPGAs for different points in the design space, transistor sizing appears to be the more powerful tool. However, architecture and transistor sizing need not be considered independently and, therefore, in the next section we examine the size of the design space when these attributes are varied in tandem.

6. TRADE-OFFS WITH TRANSISTOR SIZING AND ARCHITECTURE

For each logical architecture, a whole range of different transistor sizings, each with different performance and area, are possible. In the previous section, only a single architecture was considered, but now we explore varying the transistor sizes for a range of architectures. We considered a range of architectures with varied routing track lengths (L), cluster sizes (N) and LUT sizes (k). A comparison between architectures is most useful if the architectures present the same ease of routing. Therefore, as each parameter is varied, it is necessary to adjust other related architectural parameters such as the channel width (W) and the input/output pin flexibilities ($F_{c,input}$, $F_{c,output}$). We determine appropriate values for the channel width experimentally by finding the minimum width needed to route our benchmark circuits. The minimum channel width is increased by 20 % and rounded to the nearest multiple of twice the routing segment length to get the final width which, as described earlier, is necessary to ensure a tileable design. The input pin flexibility ($F_{c,input}$) is determined experimentally as the minimum flexibility which does not increase the channel width requirements. The output flexibility is set as, $1/N$, where N is the cluster size. For each architecture, the full range of transistor sizing optimization objectives were considered and the results for all these architectures and sizes are plotted in Figure 6. In total, 58 logical architectures were considered. With the different sizings for each architecture, this gives a total of 822 distinctly sized architectures.

Each point in the figure is a different combination of architecture and transistor sizing. Again, it is necessary to consider which points within this design space are "interesting" in the manner defined above. A slight adjustment to our criteria is necessary to handle the discrete nature of architectural changes. We start by assuming the design with minimum area-delay product is of interest. From this point, we extend the region of interest using our standard threshold that the boundary of interest is if a design requires a one percent area increase for a three percent delay improvement or a three percent area increase for a one percent delay improvement. In Figure 6, these boundaries are shown as curves extending from the minimum area-delay point. Points which are below these boundaries (provided they are not dominated by a point with less area for the same delay) are considered to be of interest. This eliminates many of the FPGA implementations as not useful but it still leaves a wide range of interesting possibilities. The

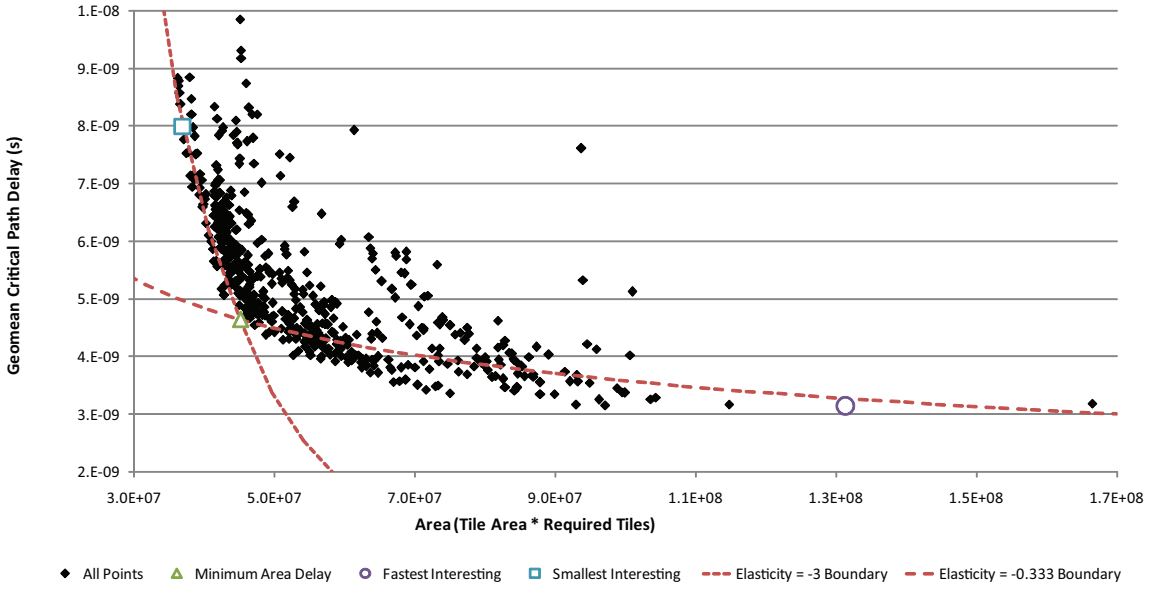

Figure 6: Full Area Delay Space

Table 3: Span of Different Sizings/Architecture

	Area (1E8 μm^2)	Delay (ns)	Area Delay ($\mu m^2 \cdot s$)	Architecture
Fastest Interesting	1.31	3.14	0.41	N=8, K=7, L=4
Min. Area Delay	0.45	4.64	0.21	N=8, K=4, L=6
Smallest Interesting	0.37	8.00	0.30	N=6, K=3, L=4
Range	3.6	2.6	2.0	

smallest interesting design and the fastest interesting design are highlighted in the figure and summarized in Table 3.

Compared to conventional experiments which would have only considered the minimum area-delay point useful we see that in fact there are a wide range of designs that are interesting when different design objectives are considered. The span of these designs is of particular interest and is summarized in Table 3. In terms of area, we see that there is a range of 3.6 × from the largest design to the smallest design and, in terms of delay, the range is 2.6 × from the slowest design relative to the fastest design. It is clear that when creating new FPGAs there is a great deal of freedom in the area-delay trade-offs that can be made and, as can be seen in Table 3, both transistor sizing and architecture are key to achieving this full range. To gain a deeper understanding of this space and how to best make these trade-offs we now explore each of the architectural parameters independently.

6.1 Segment Length

Figure 7 plots the transistor sizing curves for architectures with 4-LUT clusters of size 10 with the routing segment lengths varying from 1 to 8. It is immediately clear

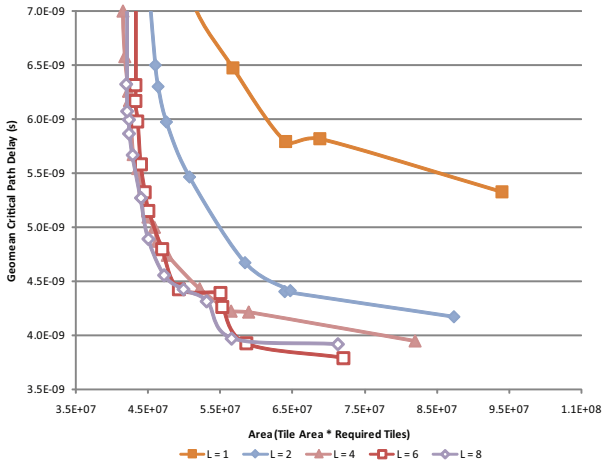

Figure 7: Area Delay Space with Varied Routing Segment Lengths

that the length-1 and length-2 architectures are not useful in terms of area and delay trade-offs. Similar conclusions have been made in past investigations [7]. From the trade-off perspective, the remaining segment lengths are all very similar. Clearly, segment length is not a powerful tool for adjusting area and delay as a single segment length generally offers universally improved performance.

6.2 Cluster Size

A range of cluster sizes from 2 to 12 were examined and the results across the full range of transistor sizings are shown in Figure 8. The routing segment length is 4 and the clusters were composed of 4-LUT BLEs. This is a more promising parameter for making area and delay trade-offs because we see that at different points in the design space, different

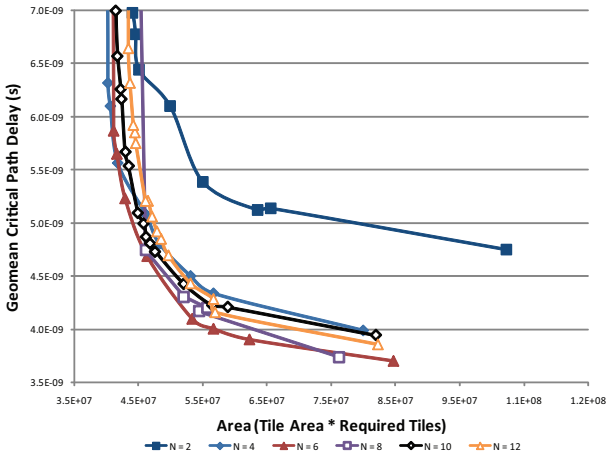

Figure 8: Area Delay Space with Varied Cluster Sizes

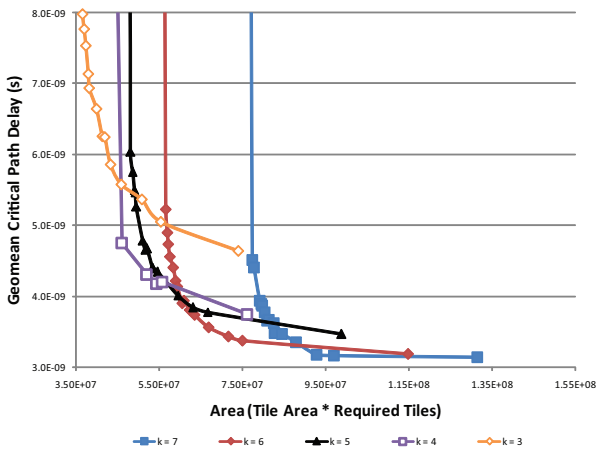

Figure 9: Area Delay Space with Varied LUT Sizing (N=8)

cluster sizes are best. For high performance (and high area), we see that the larger cluster sizes are best but, for smaller area (and worse performance) smaller cluster sizes are best. However, the differences are not extremely large, and we conclude that cluster size is only of limited use when making these design trade-offs.

These results highlight the importance of considering transistor sizing during pure architectural investigations. When optimizing for area, the best cluster sizes were, in order, 8, 6, 12, 10, 4 and 2 but when optimizing for area a completely different ordering of 4, 6, 10, 12, 2, 8 results. Clearly, a specific design objective must be considered during architectural studies and any architectural conclusions may only be valid for that specific objective.

6.3 LUT Size

Finally, for a cluster size of 8 and routing segments of length 4, a range of LUT sizes from 3 to 7 were examined across a full range of transistor sizings. The results are shown in Figure 9. In terms of area-delay trade-offs, we see that LUT size is clearly the most useful architectural parameter to vary of the parameters considered. At different points in the design space different LUT sizes are clearly best. For a minimum area-delay product a LUT size of 4 is best but for better performance larger LUT sizes are advantageous. Similarly, smaller LUT sizes are useful when area is a more important concern. In comparison to the other parameters, LUT size provides the most significant leverage for trading off area and delay. The exploration of the complete design space had also shown this as the LUT sizes between the fastest, smallest, and area-delay optimal designs in Table 3 were all different.

7. CONCLUSION

This paper has explored the trade-offs between area and delay that are possible in the design of FPGAs when both architecture and transistor sizing are varied. An automated transistor design tool was used to create a range of different circuit implementations for every architecture investigated. Compared to past pure architecture studies, we find that varying the transistor sizing of a single architecture offers a greater range of possible trade-offs between area and delay than was possible by only varying the architecture. By varying the architecture along with the transistor sizings, we see that performance could be usefully varied by a factor of 2.6 and area by a factor of 3.6. We observe that LUT size is the most useful architectural parameter for making trade-offs between area and delay. As well, we see that such architectural conclusions could not be properly made if transistor sizing were not explicitly considered.

We plan to extend this work to also examine the trade-offs that are possible between area, delay and power consumption. In addition to this, we hope to explore the possibilities when additional architectural features, such as additional types of routing segment lengths and more advanced logic blocks, are considered. With the addition of power and more advanced architectures, an even larger design space can likely be found.

8. REFERENCES

[1] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):288–298, March 2004.

[2] Altera Corporation. Stratix III device handbook, Nov 2006. ver 1.0.

[3] Altera Corporation. Cyclone III device handbook, Sept 2007. ver. CIII5V1-1.2.

[4] T. Bauer. Xilinx. Private Communication.

[5] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Seventh International Workshop on Field-Programmable Logic and Applications*, 1997.

[6] V. Betz and J. Rose. Circuit design, transistor sizing and wire layout of FPGA interconnect. In *Proceedings of the 1999 IEEE Custom Integrated Circuits Conference*, pages 171–174, 1999.

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

[8] L. Cheng, F. Li, Y. Lin, P. Wong, and L. He. Device and Architecture Cooptimization for FPGA Power Reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(7):1211–1221, July 2007.

[9] K. Compton, A. Sharma, S. Phillips, and S. Hauck. Flexible routing architecture generation for domain-specific reconfigurable subsystems. In *International Conference on Field Programmable Logic and Applications*, pages 59–68, 2002.

[10] J. Cong, J. Peck, and Y. Ding. RASP: a general logic synthesis system for SRAM-based FPGAs. In *FPGA '96: Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*, pages 137–143, New York, NY, USA, 1996. ACM Press.

[11] A. R. Conn, I. M. Elfadel, J. W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan. Gradient-based optimization of custom circuits using a static-timing formulation. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 452–459, New York, NY, USA, 1999. ACM Press.

[12] J. P. Fishburn and A. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *International Conference on Computer Aided Design*, pages 326–328, November 1985.

[13] M. Hutton, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. Patel, B. Pedersen, J. Schleicher, and S. Shumarayev. Interconnect enhancements for a high-speed PLD architecture. In *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pages 3–10, New York, NY, USA, 2002. ACM.

[14] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2), 2007.

[15] Lattice Semiconductor Corporation. LatticeECP2/M Family Handbook, Version 01.6, August 2007. `http://www.latticesemi.com/dynamic/view_document.cfm?document_id=19028`.

[16] Lattice Semiconductor Corporation. LatticeECP2/M Family Handbook, Version 02.9, July 2007. `http://www.latticesemi.com/dynamic/view_document.cfm?document_id=21733`.

[17] E. Lee, G. Lemieux, and S. Mirabbasi. Interconnect driver design for long wires in field-programmable gate arrays. In *IEEE International Conference on Field Programmable Technology*, pages 89–96, December 2006.

[18] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in FPGA interconnect. In *IEEE International Conference on Field-Programmable Technology*, pages 41–48, December 2004.

[19] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose. The Stratix II logic and routing architecture. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 14–20, New York, NY, USA, 2005. ACM Press.

[20] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose. The Stratix^{TM} routing and logic architecture. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pages 12–20. ACM Press, 2003.

[21] F. Li, Y. Lin, L. He, D. Chen, and J. Cong. Power modeling and characteristics of field programmable gate arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(11):1712–1724, Nov. 2005.

[22] A. Marquardt, V. Betz, and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 37–46, 1999.

[23] J. K. Ousterhout. Switch-level delay models for digital MOS VLSI. In *DAC '84: Proceedings of the 21st conference on Design automation*, pages 542–548, Piscataway, NJ, USA, 1984. IEEE Press.

[24] J. M. Rabaey. *Digital Integrated Circuits A Design Perspective*. Prentice Hall, 1996.

[25] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(3):202–211, July 1983.

[26] S. S. Sapatnekar, V. B. Rao, P. Vaidya, and K. Sung-Mo. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(11):1621–1634, November 1993.

[27] STMicroelectronics. 90nm CMOS090 Design Platform, 2005. `http://www.st.com/stonline/products/technologies/soc/90plat.htm`.

[28] I. Sutherland, R. Sproule, and D. Harris. *Logical Effort : Designing fast CMOS circuits*. Morgan Kaufmann Publishers, 1999.

[29] Synopsys. HSPICE. `http://www.synopsys.com/products/mixedsignal/hspice/hspice.html`.

[30] Synopsys. NanoSim. `http://www.synopsys.com/products/mixedsignal/nanosim/nanosim.html`.

[31] Xilinx. Spartan-3e, November 2006. Ver. 3.4.

[32] Xilinx. Virtex-5 user guide, October 2006. UG190 (v2.1).

[33] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, Microelectronics Center of North Carolina, Jan 1991.

[34] S. P. Young, T. J. Bauer, K. Chaudhary, and S. Krishnamurthy. FPGA repeatable interconnect structure with bidirectional and unidirectional interconnect lines, Aug 1999. US Patent 5,942,913.