Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density

Alexander (Sandy) Marquardt, Vaughn Betz, and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{arm,vaughn,jayar}@eecg.toronto.edu

Abstract

In this paper, we investigate the speed and area-efficiency of FPGAs employing "logic clusters" containing multiple LUTs and registers as their logic block. We introduce a new, timing-driven tool (T-VPack) to "pack" LUTs and registers into these logic clusters, and we show that this algorithm is superior to an existing packing algorithm. Then, using a realistic routing architecture and sophisticated delay and area models, we empirically evaluate FPGAs composed of clusters ranging in size from one to twenty LUTs, and show that clusters of size seven through ten provide the best area-delay trade-off. Compared to circuits implemented in an FPGA composed of size one clusters, circuits implemented in an FPGA with size seven clusters have 30% less delay (a 43% increase in speed) and require 8% less area, and circuits implemented in an FPGA with size ten clusters have 34% less delay (a 52% increase in speed), and require no additional area.

1. Introduction

Much of the speed and area-efficiency of an FPGA is determined by the logic block it employs. If a very small, or fine-grained, logic block is used, many connections must be routed between the numerous logic blocks [Rose93]. Since routing consumes most of the area and accounts for most of the delay in FPGAs, a small logic block often results in poor area-efficiency and speed due to the excessive routing required to connect all the logic blocks. If, on the other hand, a very large, or coarse-grained, logic block is employed, the logic block area and delay may become excessive, again resulting in poor area-efficiency and speed [Rose93]. Choosing the best size, or granularity, for an FPGA logic block therefore involves balancing complex trade-offs.

In this work we determine the best size for "cluster-based" logic blocks, which we refer to as "logic clusters". This style of logic block is of interest for several reasons. First, the Altera Flex series FPGAs [Alte98], the Xilinx 5200 and Virtex FPGAs [Xili97, Xili98], and the Vantis VF1 FPGAs [Vant98] all employ cluster-based logic blocks, so research concerning the best size of logic clusters is of clear commercial interest. Second, prior research [Betz98a] has shown that the area-efficiency of large logic clusters is quite competitive with that of FPGAs using single look-up table (LUT) logic blocks. Third, an FPGA composed of large logic clusters requires fewer logic blocks to implement a circuit than an FPGA using a more fine-grained block. This reduces the size of the placement and routing problem, and hence design compile time — an increasingly important concern as the logic capacity of FPGAs rises. Finally, we show in this paper that cluster-based logic blocks can improve FPGA speed compared to single-LUT logic blocks by reducing the number of connections on the critical path that must be routed between logic blocks.

Prior research [Betz98a] has focused only on the area-efficiency of different sizes of logic clusters. In this work, we simultaneously examine both the area-efficiency and the speed of FPGAs using different logic cluster sizes. Since both speed and density are crucial in modern FPGAs, only by examining both issues can we determine the best logic cluster size. As well, we use a more complex and realistic routing architecture than [Betz98a] in our investigations, leading to more accurate architectural conclusions. Finally, we present a new, timing-driven algorithm (T-VPack) to "pack" circuitry into logic clusters. Relative to prior work [Betz97a], this new algorithm not only improves circuit speed, but also reduces the total amount of routing required between logic blocks, resulting in improved area-efficiency.

This paper is organized as follows. Section 2 introduces the structure of cluster-based logic blocks. In Section 3 we outline the experimental methodology used to evaluate the utility of different cluster sizes. Then, in Section 4 we explain why the area-delay product is useful for evaluating the quality of each architecture. Next, Section 5 describes the FPGA architecture and timing models used in our experiments. Section 6 describes a new timing-driven logic block packing algorithm (T-VPack) and explains the enhancements it contains relative to an earlier CAD tool, VPack. In Section 7 we present experimental results comparing VPack and T-VPack, and the effect of various cluster sizes on FPGA area and delay. Section 8 discusses potential sources of inaccuracies. Finally, in Section 9 we present our conclusions.

2. Cluster-Based Logic Blocks

Cluster-based logic blocks, or *logic clusters* are a generalized version of the Logic Array Blocks used in Altera's FLEX 8K and FLEX 10K parts [Alte98]. Figure 1-a shows the structure of a *basic logic element* or *BLE* [Betz98a] which consists of a 4-LUT plus a flip-flop. A logic cluster consists of one or more BLEs, plus the local routing required to connect them together. Figure 1-b shows how the BLEs are connected. For clusters of size greater than one, the architecture used is fully connected: each BLE input can be connected to any of the cluster inputs or to the output of any of the BLEs within the cluster. Clusters of size one (i.e. a cluster contain-

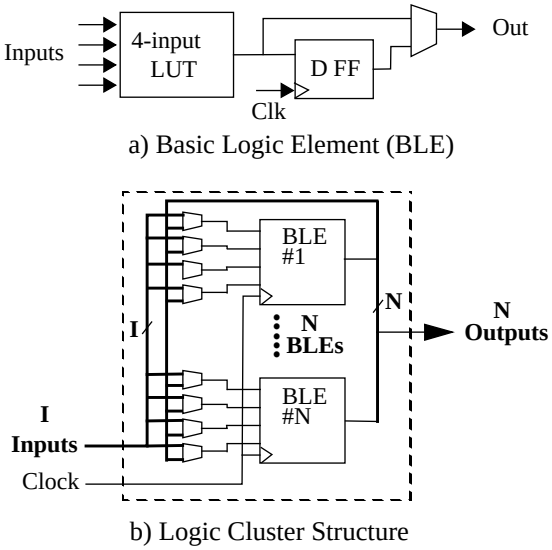

Figure 1. Structure of basic logic element (BLE) and logic cluster.

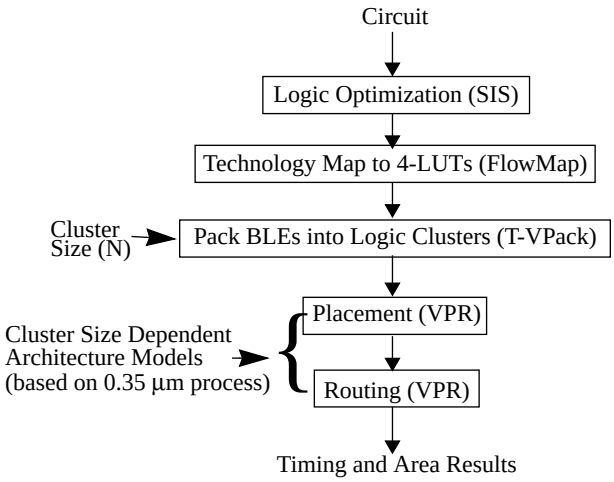

Figure 2. CAD Flow

ing a single BLE) do not contain local routing, and hence have neither multiplexors on the BLE inputs nor local feedback paths.

Following the convention of [Betz97a], we use two parameters to describe a logic cluster, *N* and *I*, where *N* is the number of BLEs per cluster and *I* is the number of inputs per cluster. In [Betz97a] it is shown that setting $I = 2 \cdot N + 2$ is sufficient for complete logic utilization, so we use this relation for all of our experiments.

3. Experimental Methodology

We use an empirical method to explore different FPGA architectures. This involves technology-mapping, packing, placing, and routing benchmark circuits[1] into realistic architectures with clusters of size 1 through 20. We then estimate the area required by each architecture to implement each benchmark circuit, and measure the speed of each implementation. At this point we have enough information to judge the quality of each architecture.

3.1 CAD Flow

Figure 2 illustrates the CAD flow for our experiments. Each circuit we use is logic-optimized by SIS [Sent92] and then technology-mapped into 4-LUTs by FlowMap [Cong94]. VPack [Betz98b, Betz97b, Betz99] or T-VPack is then used to group the LUTs and registers into logic clusters of the desired size. Finally, we use VPR [Betz98b, Betz97b, Betz99] to place and route each circuit. VPR's timing-driven router extracts the elmore delay [Elmo48] of each routed net, and performs a path-based timing analysis to determine the delay of the circuit critical path. Finally, VPR uses a transistor-based area model [Betz98b, Betz99] to estimate the total layout area required by this FPGA.

[1] Our benchmarks consist of 20 of the largest MCNC circuits [Yang91] and 5 University of Toronto benchmark circuits [Leve98, Ye98, Gall98, Padi98, Hame98]. The circuits range in size from 1047 to 8383 4-LUTs. The MCNC circuits used are: alu4, apex2, apex4, bigkey, clma, des, diffeq, dsip, elliptic, ex1010, ex5p, frisc, misex3, pdc, s298, s38417, s38584.1, seq, spla, and tseng. The University of Toronto circuits used are: des_fm, des_sis, marb, grayscale, and wood.

In FPGA architecture and CAD research, it is convenient to have tools which can vary the FPGA dimensions (number of columns and rows) and channel width (number of tracks in each channel). VPR allows this, and it also allows us to find the minimum channel width required to successfully route a circuit. By allowing the channel width to vary, and searching for the minimum routable width, we can detect small improvements in FPGA architectures or CAD algorithms that might otherwise go unnoticed. Compare this to mapping a circuit into a fixed size FPGA — this would only tell us if it fit or not. It is more difficult to draw architectural conclusions from such a "binary" result.

VPR is capable of performing both *high-stress* and *low-stress* routings [Swar98]. A high-stress routing occurs when VPR routes a given circuit into an FPGA with the minimum channel width required for a successful routing. To accomplish this, VPR repeatedly routes each circuit with different channel widths, scaling the architecture accordingly until it finds the minimum number of tracks in which the circuit will route. A low-stress routing occurs when an FPGA has significantly more routing resources than the minimum required to route a given circuit. In our experiments we define a low-stress routing to occur when there are 30% more tracks per channel than the minimum required.

We feel that low-stress routings are indicative of how an FPGA would generally be used (it is rare that a user will utilize 100% of the routing and logic resources), so all of the results that we present are based on low-stress routings. Additionally, the low-stress and high-stress results are very similar, and both cases result in the same conclusions.

4. Architecture Evaluation — Area-Delay Product

One metric that we will use to evaluate the quality of different architectures is the area-delay product. We feel that there are two reasons that this metric makes sense:

1. Intuitively, we want to find the point at which we are sacrificing the least amount of area for the most improvement in speed. Given that we can always trade area for speed (see below), and speed for area, it makes sense to combine these two factors into one curve to see where the best trade-off occurs.

2. Much of the performance gain from using an FPGA is derived from parallelizing functional units, rather than raw clock speed. In this case, *throughput = number of functional units · clock rate*. Another way of looking at this is, *throughput = (1/area per functional unit) · (1/ delay)*. Therefore if we minimize the area-delay product, we will maximize throughput.

There are two main factors which can affect the area-delay product of an FPGA: transistor sizing, and the FPGA architecture. In general, the speed of an FPGA can be increased (to a point) by sizing up the buffers and transistors within the FPGA, but this increases area. Alternatively, the FPGA can be made smaller by sizing down the buffers and transistors, but this degrades the FPGA performance.

Throughout this paper, we will size the transistors in each FPGA architecture to minimize the FPGA's area-delay product. Only by resizing transistors appropriately for each architecture in this way can we fairly compute the speed and area-efficiency of FPGAs with different logic block architectures.

5. Architecture Modeling

To evaluate the speed and area of an FPGA we must choose not only the logic block architecture, but also a routing architecture and transistor sizes. The following sections detail all of our architectural choices, which are provided to VPR in an architecture description file [Betz98b, Betz99].

5.1 Basic Architecture

We investigate island-style FPGAs in which each logic block borders a routing channel on its four sides. Each circuit is mapped to the smallest square FPGA with enough logic blocks and pads to accommodate it. The FPGAs of Xilinx [Xili94], Lucent Technologies [Luce98], and Vantis [Vant98] employ an island-style architecture.

Delays, capacitances, and resistances of the FPGA circuitry are obtained from SPICE [Meta92] simulations of TSMC's 0.35 µm CMOS process.

5.2 Routing Architecture

We define the number of logic blocks which a routing segment spans as the *logical length* of that segment. [Betz98b, Betz99] found that an architecture in which routing segments have a logical length of four, with 50% of the segments connected by tri-state buffers and 50% connected by pass-transistors, provides good area-efficiency and speed for FPGAs containing logic clusters of size four. An example of this routing architecture is shown in Figure 3. We implicitly assume that this routing architecture is good for architectures containing logic clusters of all sizes, and we use this routing architecture in all of our experiments. Ideally, one would like to find the best routing architecture for each FPGA employing a different cluster size, but this would require a huge amount of effort. By basing all of our experiments on this routing architecture, we may slightly favor architectures with size four clusters over other architectures.

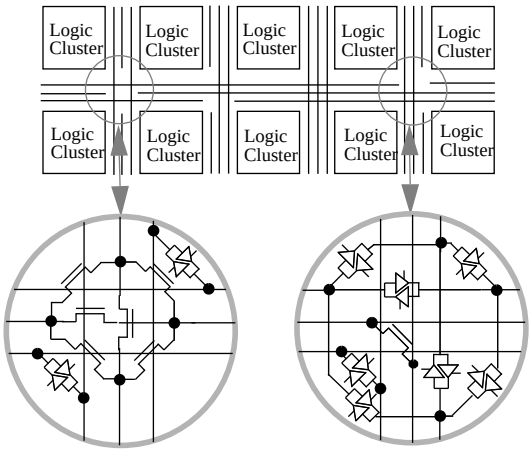

Figure 3. FPGA Architecture with Length 4 Segments, and 50/50 Unbuffered/Buffered Switches.

5.3 Effect of Varying Cluster Size on FPGA Routing Segment Length

As we increase the cluster size, both the logic area per cluster and routing area per cluster grow. The logic cluster and its associated routing is called a tile. Figure 4 demonstrates how a tile grows as cluster size is increased. This increased tile size results in routing segments with the same logical length having physically different lengths for logic clusters of different sizes.

We define the measured length of a routing segment as its *physical length*. There is a linear relation between the physical length of a routing segment, and the resistance and capacitance of that segment. We have experimentally determined the average rate at which the FPGA tiles grow with cluster size, and have used this knowledge to appropriately scale the routing segment resistance and capacitance values for the various cluster sizes.

5.4 Scaling Transistor and Buffers to Compensate for Increased Segment Physical Length

To compensate for differences in the capacitance and resistance of different length routing segments, we scale the routing pass-transistors and buffers. All of our transistor and buffer scaling is in relation to a base architecture that has been area-delay optimized for clusters of size four [Betz98b, Betz99]. From this base architecture, we linearly scale our buffers and pass transistors depending on the relation between the new segment lengths and the base segment length. For example, in an FPGA with size 16 clusters, the physical segment length is approximately 2x longer than in an architecture with size 4 clusters. To maintain roughly the same routing speed, we increase the size of the routing switches connecting to each wire by a factor of 2. In Section 7.2 we verify that this linear scaling of buffers and pass-transistors with segment length provides the best results.

In our architecture models, we account for variations in delay caused by resizing buffers and pass-transistors. Also, changes in

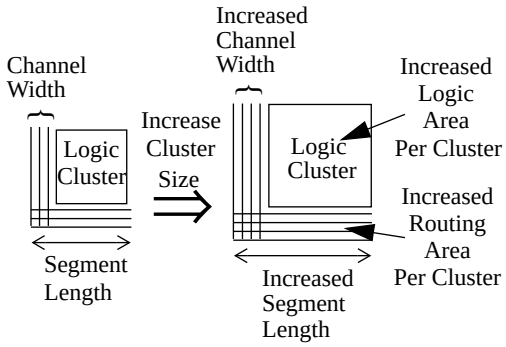

Figure 4. Effect of Increased Cluster Size on Segment Length

area due to the use of different sizes of routing pass-transistors and inverter chains are automatically calculated by VPR.

5.5 Varying $F_{c, in}$ and $F_{c, out}$ with Logic Cluster Size

In [Rose91] it is shown that $F_c = W$ is good for logic clusters of size one; i.e. each logic block pin can be connected to any routing track in an adjacent channel. As cluster size increases, setting $F_c = W$ provides more flexibility than is required, wasting area. In [Betz98b, Betz99] it is shown that setting F_c on the input pins ($F_{c, in}$) to $2 \cdot W/N$ and F_c on the output pins ($F_{c, out}$) to W/N provides a good level of routing flexibility, so all of our experiments use these values for clusters of sizes other than one.

5.6 Detailed Logic Cluster Structure.

In Figure 5 we show the structure of a logic cluster and the circuitry connecting the logic clusters to the main FPGA routing. Table 1 shows delay values for selected cluster sizes. The multiplexor, buffer, LUT, and flip-flop delays were obtained by modeling the structures in SPICE [Meta92] with TSMC's 0.35 µm process parameters.

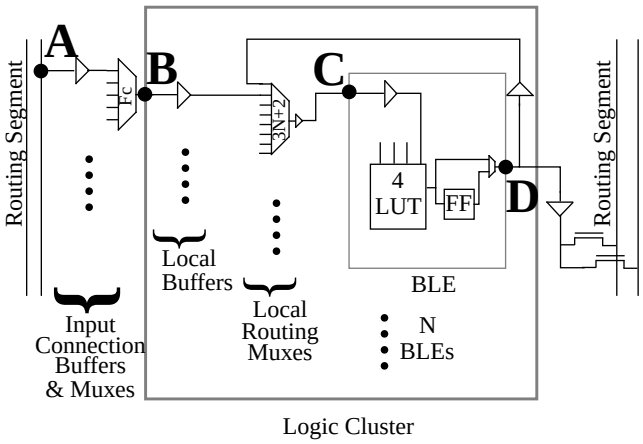

Figure 5. Detailed Logic Cluster Structure

Table 1: Selected Logic Cluster Delay Values (in picoseconds) 0.35 µm CMOS

Cluster Size	A to B	B to C and D to C	C to D	B to D
1 (No Local Muxes)	761	140	379	519
2	761	627	379	1006
4	761	761	379	1140
8	761	902	379	1281
16	761	1056	379	1435
20	761	1084	379	1463

6. Packing Algorithms

The packing step (in Figure 2) takes a netlist consisting of LUTs and flip-flops and produces a netlist consisting of logic clusters. This involves combining the LUTs and flip-flops into BLEs, and then grouping the BLEs into logic clusters.

There are two main constraints that packing algorithms must meet:

1. The number of BLEs must be less than the cluster size, *N*.
2. The number of distinct inputs generated outside the cluster and used as inputs to BLEs within the cluster must be less than or equal to the number of cluster inputs, *I*.

In this section, we present two packing algorithms, VPack [Betz97b, Betz98b, Betz99], and T-VPack. Then we show that our new T-VPack algorithm outperforms the original VPack algorithm in both area and critical path delay.

6.1 Input-Sharing VPack Algorithm

The original VPack algorithm has two optimization goals. The first is to pack each logic cluster to its capacity in order to minimize the number of clusters needed. The second goal is to minimize the number of inputs to each cluster in order to reduce the number of connections required between clusters.

Vpack uses a greedy algorithm to construct each cluster sequentially. At the start of each cluster operation, Vpack selects as a "seed" an unclustered BLE with the most used inputs, and then places this "seed" into a cluster *C*. Then VPack selects a new BLE, *B* to pack into *C* based on the *attraction* that *B* has to *C*. Attraction is determined by the number of inputs and outputs that *B* and *C* have in common:

$$Attraction(B) = |Nets(B) \cap Nets(C)| \quad \textbf{(1.1)}$$

After each cluster reaches capacity, packing begins on a new cluster. The process terminates when there are no more unclustered BLEs left. The time complexity of this algorithm is $O(k_{max} \cdot n)$ (where n is the number of BLEs in the circuit and k_{max} is the fanout of the highest fanout net) which results in an execution time of about four seconds to pack the largest circuit (clma) on a 296 MHz UltraSPARC-II processor.

6.2 Timing-Driven T-VPack Algorithm

Our new packing algorithm is based on the original VPack algorithm, but its optimization goal is minimizing the number of external connections (connections between clusters) on the critical path. The reasoning behind this is that external connections have higher delay than internal connections (connections within a cluster), so by reducing the number of external nets on the critical path, we will reduce the circuit delay. The first stage of this algorithm involves computing which connections are on the critical path. We then sequentially pack BLEs along the critical path into logic clusters and recompute which BLEs are critical.

6.2.1 An Overview of Slack and Criticality Calculation

The first step in determining which nets are critical is to determine the *slack* of each connection [Hitc83, Fran92]. Slack is defined as the amount of delay which can be added to a connection without increasing the delay of the entire circuit.

Calculating slack involves computing the arrival time, $T_{arrival}$ and the required arrival time, $T_{required}$ at all BLE input pins. This is accomplished using two breadth-first traversals of the circuit; the

first traversal propagates $T_{arrival}$ forward from input pins and register outputs (Sources), and the second propagates $T_{required}$ back from output pins and register inputs (Sinks). The slack of a connection driving a BLE input pin, *i*, is defined as:

$$slack(i) = T_{required}(i) - T_{arrival}(i) \quad \textbf{(1.2)}$$

Finally, we define the criticality of the connection driving input *i* as:

$$Connection\_Criticality(i) = 1 - \frac{slack(i)}{MaxSlack} \quad \textbf{(1.3)}$$

where *MaxSlack* is the largest slack amongst all point-to-point connections in the entire circuit.

6.2.2 Delay Estimates of an Unplaced and Unrouted Circuit

To obtain a good packing solution[1] the T-VPack algorithm models three types of delay: The delay through a BLE, or *logic_delay,* the connection delay between blocks within the same cluster or *intra_cluster_connection_delay,* and the connection delay between blocks that are in different clusters, or *inter_cluster_connection_delay.* We experimentally determined that setting *logic_delay*=0.1, *intra_cluster_connection_delay*=0.1, and *inter_cluster_connection_delay*=1.0 results in the clustered circuits having the smallest delay after placement and routing by VPR[2].

6.2.3 The Attraction Function

We extend the attraction function from the original VPack algorithm to include timing information. The first BLE that is placed into a cluster is the unclustered BLE that is driven by the most critical connection in the circuit. Then, based on our attraction function (Equation 1.8, below) we add the most attractive BLEs to the cluster. We repeat this absorbtion until either no more BLEs will fit into the cluster, or all of the cluster inputs are used. Once a cluster is full, we start a new cluster with a new seed, and repeat the process until there are no unclustered BLEs left in the circuit. We next describe how blocks are selected for absorbtion.

We define the base criticality of each unclustered BLE, *B*, or *Base_BLE_Criticality(B)*, to be the maximum *Connection_Criticality* value of all connections joining *B* to BLEs within the cluster currently being packed, *C*. If *B* does not have any connections to *C* then the base criticality score is zero. In Figure 6 we illustrate how the *Base_BLE_Criticality* values are assigned. We have labelled each connection between unclustered BLEs and BLEs within the cluster with a criticality value. Notice how the base criticality of each BLE is assigned the highest criticality value of all its connections to the cluster.

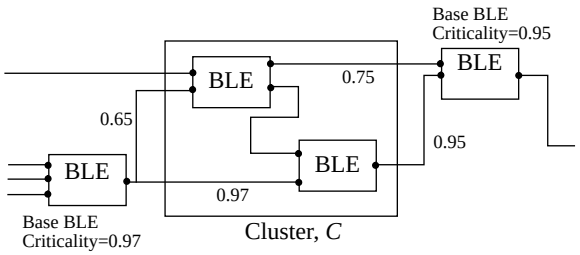

Figure 6. BLE Base Criticality Assignment

When selecting which BLE to absorb into a cluster there is a high potential for multiple BLEs to have the same base criticality value. We use a tie-breaker mechanism to select which BLEs are the most beneficial to pack. This mechanism is based on the desire to pack BLEs together in a manner that most effectively reduces the number of BLEs remaining on the critical paths. This is best illustrated by an example.

In Figure 7 we have darkened connections and BLEs on the critical paths. Notice that when selecting which BLEs to place into a cluster, it is more beneficial to absorb certain critical BLEs over other critical BLEs. In this case, absorbing BLEs H, I, and J would be much more beneficial than absorbing BLEs A, D, and F. We can see that absorbing H, I, and J affects the criticality of seven BLEs (A, B, C, D, E, F, and G), while absorbing A, D, and F would only affect the criticality of three BLEs (H, I, and J). Clearly it is best to cluster BLEs that reduce the criticalities of the most other BLEs.

We define three variables that keep track of the number of critical paths that each BLE in the circuit effects. First we define *input_paths_affected* as the number of critical paths between sources in the circuit and the BLE currently being labelled. Next we define *output_paths_affected* as the number of critical paths between the sinks in the circuit and the BLE currently being labelled. Finally, we define *total_paths_affected* as the sum of the previous two variables. The calculation of these variables is explained below.

The BLE labels in Figure 7 demonstrate the *input_paths_affected* value for each BLE. We assign any sources that are on the critical paths with an *input_paths_affected* value of one, and all other sources are set to zero. Then we perform a breadth-first traversal of the circuit starting at the sources, and define the *input_paths_affected* value as in (1.4).

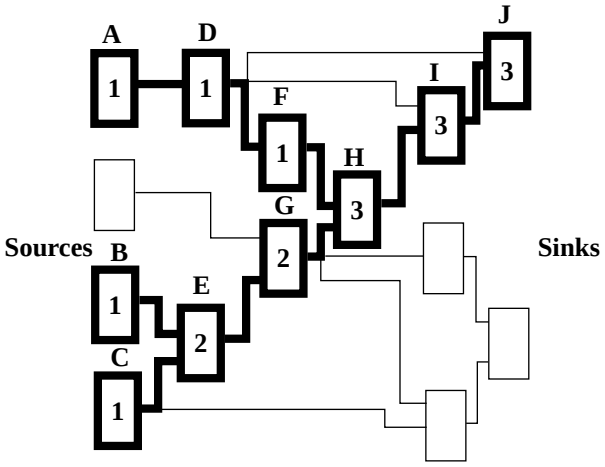

Figure 7. Criticality Tie-Breakers

[1] A good packing solution is one that results in the smallest delay after being placed and routed by VPR.

[2] Note that these delay values are *only used in the packing process*. After packing is complete, VPR places and routes the circuits and extracts the real (elmore) delay of each routed net. All of the delay results that we present in this paper are computed by VPR.

$$input\_paths\_affected(B) = \sum_{\forall D \in critical\ inputs(B)} input\_paths\_affected(D) \quad (1.4)$$

Where *critical inputs(B)* refers to the BLEs driving the connections on *B*'s inputs that are on the critical path.

The *output_paths_affected* variable is calculated in the same manner, but it starts at outputs and works back towards the inputs.

$$output\_paths\_affected(B) = \sum_{\forall D \in critical\ outputs(B)} output\_paths\_affected(D) \quad (1.5)$$

We define *total_paths_affected* as

$$total\_paths\_affected(B) = input\_paths\_affected(B) + output\_paths\_affected(B) \quad (1.6)$$

Criticality(B) is defined as.

$$Criticality(B) = Base\_BLE\_Criticality(B) + (\varepsilon \cdot total\_paths\_affected(B)) \quad (1.7)$$

where ε is a very small value that ensures that the total_paths_affected value acts only as a tie-breaking mechanism.

Finally, we define our new attraction function as follows:

$$Attraction(B) = \alpha \cdot Criticality(B) + (1-\alpha) \cdot \frac{|Nets(B) \cap Nets(C)|}{G} \quad (1.8)$$

Where *G* is a normalization factor which is set to the maximum number of nets to which any a BLE can connect, i.e.

$$G = \#BLE\ inputs + \#BLE\ outputs + \#BLE\ Clocks \quad (1.9)$$

In (1.8), α is a trade-off variable which determines how much we wish the attraction to be affected by criticality vs. input pin sharing. If we set α to 0 then we have a purely pin-sharing based algorithm, and the program functions the same as the original VPack algorithm. If we set α to 1 then we have an algorithm that focuses only on minimizing the critical path with no concern for the number of inputs shared. We experimentally determined that setting α to a value of 0.75 results in clusterings with the least delay.

The time complexity of this algorithm is $O(n^2)$ (where n is the number of BLEs in the circuit) which results in an execution time of about two minutes[1] to pack the largest circuit (clma) on a 296 MHz UltraSPARC-II processor.

7. Area and Delay at Various Cluster Sizes

This section shows the effect of varying cluster size on the area and delay of the benchmarks. This involves packing, placing, and routing the benchmark circuits and comparing the resulting FPGA area and critical path delay. The results that we present are based on low-stress routings (described in Section 3.1).

[1] There is an option in T-VPack which allows the user to specify how many blocks, P, to pack before re-computing the timing information. This reduces the time complexity to $O(n^2/P)$. We have found that performing a timing analysis only once at the beginning (set P=n) does not reduce the quality of the placed and routed circuits. This reduces the complexity to $O(k_{max} \cdot n)$, and requires only a few seconds to pack the largest circuit.

7.1 Cluster Size Comparison using both VPack and T-VPack

In this section we present results from circuits packed with both VPack and T-VPack. We demonstrate that the T-VPack algorithm is superior to the VPack algorithm, and we show the effects of increased cluster size on area and delay.

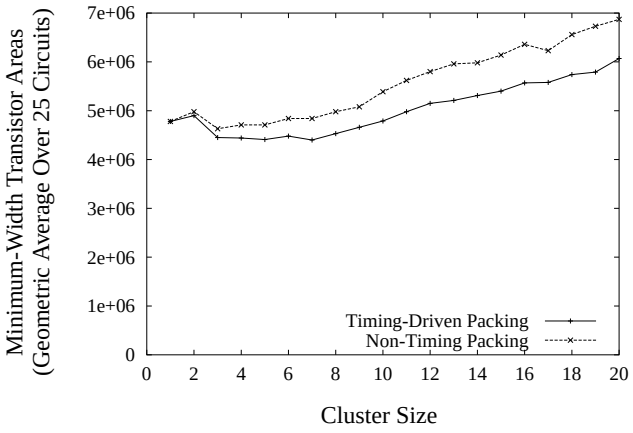

Figure 8. Area vs. Cluster Size

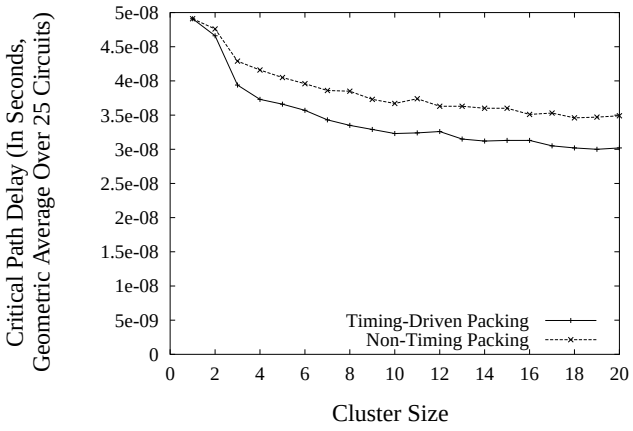

Figure 9. Critical Path Delay vs. Cluster Size

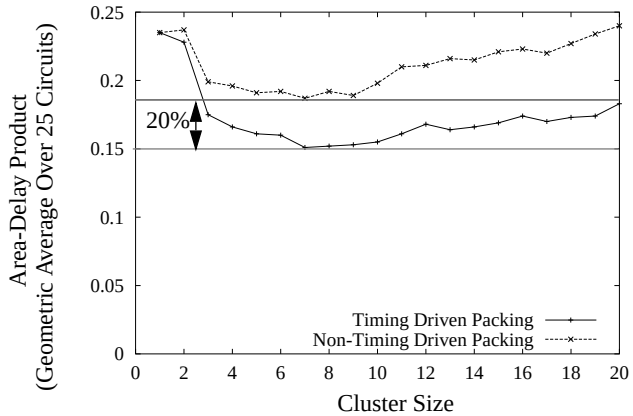

Figure 10. Area-Delay Product vs. Cluster Size

7.1.1 Area and Delay as Cluster Size is Increased

After the benchmark circuits were packed with the two different clustering algorithms, they were placed and routed using VPR to obtain area and critical-path delay estimations. The total area of each circuit (logic plus routing) is given in terms of the equivalent number of *minimum-width transistor areas*. A minimum-width transistor area is the layout area occupied by the smallest transistor that can be contacted in a process, plus the minimum spacing to another transistor above it and to its right [Betz98b, Betz99].

In Figure 8 we show the geometric average of the total circuit area of the benchmarks vs. cluster size. It can be seen that the T-VPack algorithm has significantly improved the area required for each circuit when compared to the original VPack algorithm, particularly for larger cluster sizes (this improvement is explained in Section 7.1.2).

Area is affected by two factors. First, as we increase cluster size we reduce the routing requirements between clusters, so we require less routing area. Second, as we increase cluster size, the total area of the multiplexors within each cluster grows quadratically. For sufficiently large clusters, the area reductions in the routing are overtaken by the increased area required within the larger clusters.

Figure 9 shows the geometric average of the critical path delay of the benchmarks vs. cluster size for both algorithms, and demonstrates that the delay for the T-VPack algorithm is less than the delay for the original VPack algorithm. Additionally, this graph shows that the critical path delay is decreasing as cluster size is increased. This means that for clusters of size one through 20, larger clusters provide better speed (a detailed explanation of why this occurs is given in Section 7.1.3).

In Figure 10 we show the geometric average of the area-delay product of the benchmarks vs. cluster size. Comparing T-VPack to VPack, we can see that T-VPack has improved the area-delay product by about 20% for clusters of size seven through ten. This represents a comparison of both algorithms at their best performance points. At larger cluster sizes the T-VPack algorithm provides even more of a performance gain. This is mainly due to the increased number of nets that the T-VPack algorithm completely absorbs within clusters, resulting in reduced circuit area.

Figure 10 makes an important result visible — clusters of size seven through ten provide the best trade-off between area and delay. Compared to a cluster of size one, a cluster of size seven has an area-delay product that is 36% better, and a cluster of size ten has an area-delay product that is 34% better.

On average, circuits implemented in an FPGA with size seven clusters have 30% less delay (a 43% increase in speed) and use 8% less area than circuits implemented in an FPGA with size one clusters. Circuits implemented in an FPGA with size ten clusters have 34% less delay (a 52% increase in speed), and require no additional area compared to circuits implemented in an FPGA with size one clusters.

All of the individual benchmark circuits tracked these averages quite well (with minor variations, mostly at cluster sizes one and two).

7.1.2 T-VPack Area Improvement over VPack

As Figure 8 shows, T-VPack produces circuits that require less area than circuits packed with VPack. To understand the reason for this surprising result, one must compare the structure of the packed circuits produced by VPack and T-VPack. The criticality term in the T-VPack attraction function (1.8) makes T-VPack prefer to cluster a BLE with BLEs that are in its fan-in or fan-out, rather than with BLEs that it shares inputs with. As a result, T-VPack produces circuit packings in which many low-fanout nets have been completely absorbed into logic clusters[1].

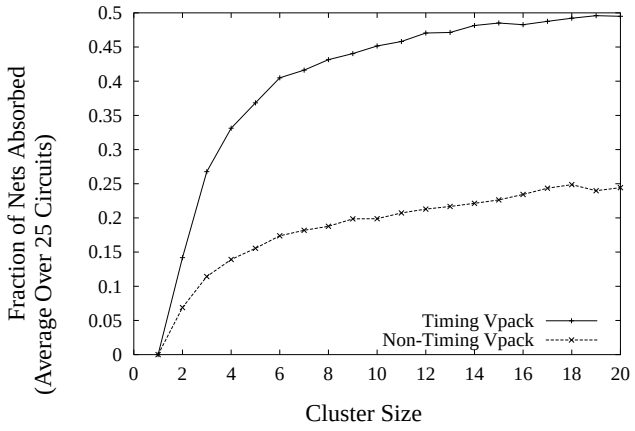

Figure 11. Number of Nets Absorbed vs. Cluster Size

Figure 11 shows the number of nets absorbed vs. cluster size for both VPack and T-VPack. Since T-VPack has absorbed more nets than VPack, it has fewer nets to route between clusters than the output of VPack; however, the average fanout of each inter-cluster net is slightly higher (not shown). The net result is that the circuits packed with T-VPack are somewhat easier to route than the circuits packed with VPack, resulting in a reduction in the routing area required[2].

7.1.3 Explanation of Delay Results

In Figure 12 we show the relationship between the number of internal (intra-cluster — fast) and external (inter-cluster — slower) connections on the critical path. As cluster size is increased the number of internal connections on the critical path is increased, and the number of external connections is decreased. This provides a circuit speedup due to fact that internal connections are faster than external connections[3].

It is interesting to note that for clusters of size greater than four, the number of external (inter-cluster) nets on the critical path does not decrease as much with cluster size as the inter-cluster delay decreases with cluster size (see Figure 13). From size four to size twenty we have a reduction in the number of external nets on the critical path (Figure 12) of about 18%; compare this to the inter-

[1] For a net to be completely absorbed into a cluster, it must have all of its terminals contained within that cluster.

[2] This result shows the importance of using a full CAD flow, including placement and routing, to evaluate many FPGA issues. It would have been difficult or impossible to guess that the output of T-VPack would be easier to route than the output of VPack without actually placing and routing the outputs from both packing algorithms. In fact, since the circuit packings produced by T-VPack have more point-to-point connections to route between clusters (despite having fewer nets), one would likely guess that T-VPack's circuits would be more difficult to route.

[3] As cluster size is increased, internal cluster multiplexor and wiring delays increase. If we were to keep increasing the cluster size, this effect would eventually result in internal delays becoming large enough that any gains obtained from making connections local to the cluster would be lost.

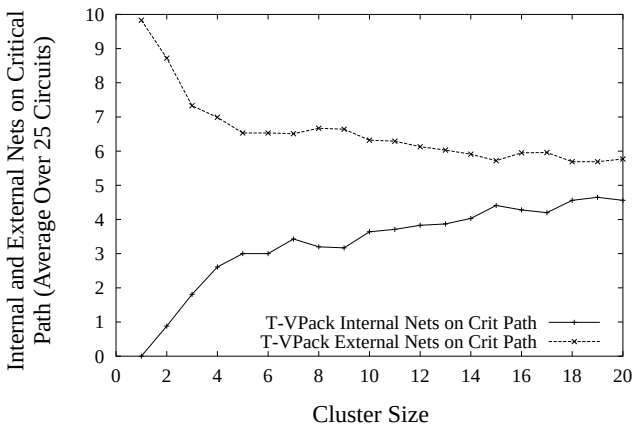

Figure 12. Internal and External Nets on the Critical Path (Post Place and Route, T-VPack Only)

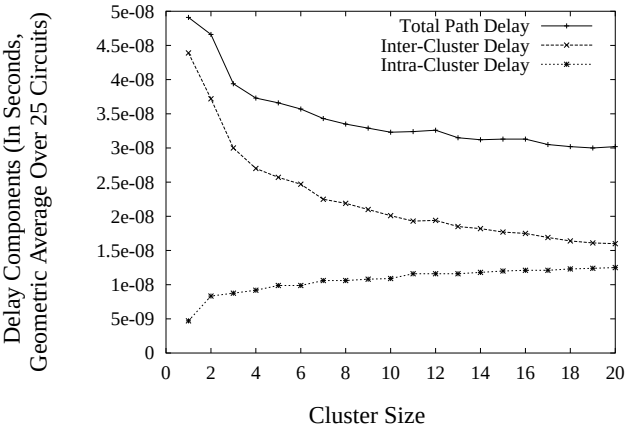

Figure 13. Critical Path External, Internal, and Total Path Delay (Post Place and Route, T-VPack Only)

cluster critical path delay (Figure 13) which has been reduced by 40% over this same range. This means that the circuit speedup visible in Figure 13 for larger cluster sizes is not only caused by a reduction in the number of external nets on the critical path but it is also *caused by inter-cluster connections on the critical path becoming faster.* This is explained below.

The improvement in inter-cluster delay with increased cluster size is caused in part by a reduction in the "logical" manhattan distance between connections in the FPGA as shown in Figure 14. By sizing buffers[1] to compensate for the increased physical length of routing wire segments associated with larger clusters, the delay of each routing segment has remained roughly constant. Since the total number of segments on the critical path has decreased due to the reduction in the "logical" manhattan distance, the result is a

[1] Changes in delay and area due to different size routing buffers is accounted for in VPRs timing and area models.

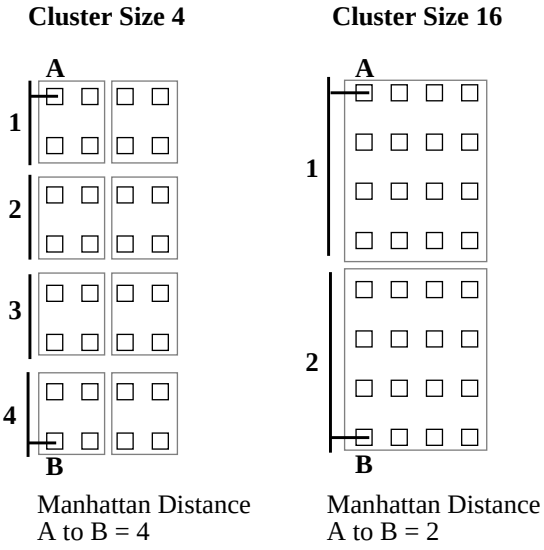

Figure 14. Decreased Manhattan Distance as Cluster Size Increases

greater improvement in critical path delay than the reduction in the number of nets on the critical path would indicate.

7.2 Effect of Routing Transistor Sizing on Critical Path Delay and Area at Various Cluster Sizes

The purpose of this section is to provide a verification that the manner in which we sized buffers and transistors is acceptable, and did not favor one cluster size over another. In this section we use only T-VPack to pack the circuits since we have demonstrated that it is superior to VPack.

We have repeated the experiments described in Section 7.1 using transistor and buffer sizes of one-half and double the sizes used in Section 7.1. The results from these experiments are shown in Figures 15, 16, and 17. These experiments validate the original

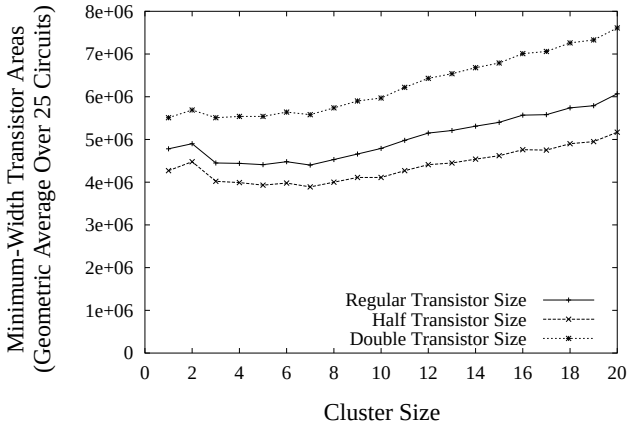

Figure 15. Area vs. Cluster Size for Various Transistor Sizings

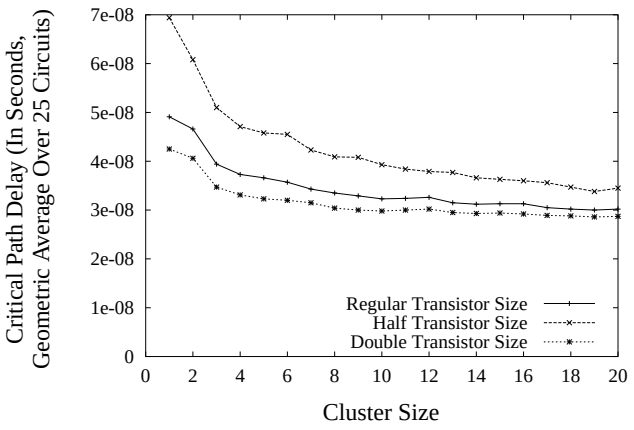

Figure 16. Critical Path Delay vs. Cluster Size for Various Transistor Sizings

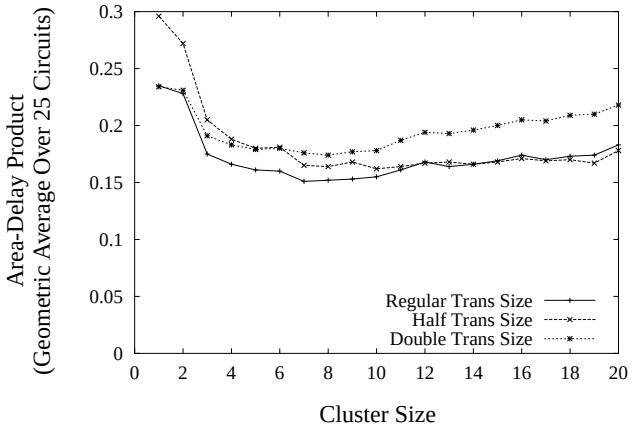

Figure 17. Area-Delay Product vs. Cluster Size for Various Transistor Sizings

transistor sizings that we used since the new transistor sizings do not improve the area delay trade-off.

8. Potential Sources of Inaccuracies

Every effort has been made to ensure that our results are accurate, however, there are three potential sources of inaccuracies.

First, without actually laying out the various FPGA architectures, there is some estimation involved in determining how much area various FPGA implementations will require.

Second, VPR uses the Elmore delay model [Elmo48] to evaluate the speeds of circuits implemented in various FPGA architectures. Generally the delays calculated by VPR are within 9% of SPICE delays [Betz98b, Betz99]. Also, delay results can be affected by our area model since it affects wire lengths and transistor sizings.

Third, area and delay results are affected by the quality of the placement and routing software. The tools used for these experiments have been shown to produce high quality results [Betz98b, Betz99], but it is always possible that the CAD software does a better job for certain architectures over others.

We have taken considerable care to minimize the effects of these potential sources of inaccuracies, and we believe that the our results are of high quality.

9. Conclusions

We presented a new timing-driven packing algorithm, T-VPack and demonstrated that this algorithm provides significant timing and area improvements over the original VPack algorithm. Circuits packed with T-VPack have an area-delay product that is 20% better than circuits packed with VPack for clusters of size seven to ten, and for larger cluster sizes the improvement is even greater.

Using the area-delay product evaluation metric, we demonstrated that clusters of size seven to ten are the best size to use when constructing an FPGA. Compared to circuits implemented in an FPGA with size one clusters, circuits implemented in an FPGA with size seven clusters have 30% less delay (a 43% increase in speed) and use 8% less area, and circuits implemented in an FPGA with size ten clusters have 34% less delay (a 52% increase in speed), and require no additional area. The reason for this improvement in circuit speed at larger cluster sizes is partly due to an increased number of critical connections becoming local within clusters, and partly due to a reduction in the "logical" manhattan distance between BLEs.

10. References

[Alte98] Altera Inc., *Data Book*, 1998.

[Betz97a] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *IEEE Custom Integrated Circuits Conference,* Santa Clara, CA, 1997, pp. 551-554.

[Betz97b] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Int'l Workshop on FPL*, 1997, pp. 213-222.

[Betz98a] V.Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block?," *IEEE Design and Test Magazine,* Spring 1998, pp. 10-15.

[Betz98b] V. Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," *Ph. D. Dissertation, University of Toronto,* 1998.

[Betz98c] V. Betz, "VPR and VPack User's Manual (Version 4.17)," May 5, 1998. (*Available for download from http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html).*

[Betz99] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, (expected publication date: February 1999).

[Brow92] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[Brow96] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial," *IEEE Design & Test of Computers*, Summer 1996, pp. 42-57.

[Cong94] J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD*, Jan. 1994, pp 1-12.

[Elmo48] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *J. Applied Physics*, Vol. 19, January 1948, pp. 55-63.

[Fran92] J. Frankle, "Iterative and Adaptive Slack Allocation

for Performance-Driven Layout and FPGA Routing," DAC, 1992, pp. 536 - 542.

[Gall98] D. Galloway, "Implementation of Grayscale Conversion for Video Image Processing on the Transmogrifier-2a," *Personal Communication.*

[Hame98] I. Hamer, "Implementation of DES on the Transmogrifier-2a," *Personal Communication.*

[Hitc83] R. Hitchcock, G. Smith and D. Cheng, "Timing Analysis of Computer-Hardware," *IBM Journal of Research and Development*, Jan. 1983, pp. 100 - 105.

[Leve98] P. Leventis, "Using edif2blif Version 1.0," June 30, 1998. (*Available for download from http://www.eecg.toronto.edu/~leventi/edif2blif/edif2blif.html*).

[Luce98] Lucent Technologies, *FPGA Data Book*, 1998

[Meta92] Meta-Software, *Hspice User's Manual*, 1992.

[Padi98] K. Padalia, "Implementation of Grayscale Conversion for Video Image Processing on the Transmogrifier-2a," *Personal Communication.*

[Rose90] J. Rose, R. J. Francis, D. Lewis and P. Chow, "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, Oct. 1990, pp. 1217 - 1225.

[Rose91] J. Rose and S. Brown. "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays," *JSSC*, March 1991, pp. 277 - 282.

[Rose93] J. Rose, A. El Gamal and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings IEEE*, vol. 81, no. 7, July 1993, pp. 1013 - 1029.

[Sent92] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Analysis," *Tech. Report No. UCB/ERL M92/41*, University of California, Berkeley, 1992.

[Swar98] J. Swartz, V. Betz and J. Rose, "A Fast Routability-Driven Router for FPGAs," *FPGA*, 1998, pp. 140 - 149.

[Vant98] Vantis Corporation, "VF1 Field Programmable Gate Array," *Preliminary Data Sheet*, 1998.

[West93] N. Weste and K Eshraghian, *Principles of CMOS VLSI Design; A System Perspective; Second Edition*, *Addison Wesley*, 1993.

[Xili94] Xilinx Inc., *The Programmable Logic Data Book*, 1994.

[Xili97] Xilinx Inc., "XC5200 Series of FPGAs", *Data Book*, 1997.

[Xili98] Xilinx Inc., "Virtex 2.5 V Field Programmable Gate Arrays", *Advance Product Data Sheet*, 1998.

[Yang91] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report*, Microelectronics Center of North Carolina, 1991.

[Ye98] A. Ye, "Procedural Texture Mapping on FPGAs", *M.A.Sc. Thesis, in Preparation*, 1998.