Using Sparse Crossbars within LUT Clusters

Guy Lemieux
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 3G4
lemieux@eecg.toronto.edu

David Lewis
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 3G4
lewis@eecg.toronto.edu

ABSTRACT

In FPGAs, the internal connections in a cluster of lookup tables (LUTs) are often fully-connected like a full crossbar. Such a high degree of connectivity makes routing easier, but has significant area overhead. This paper explores the use of sparse crossbars as a switch matrix inside the clusters between the cluster inputs and the LUT inputs. We have reduced the switch densities inside these matrices by 50% or more and saved from 10 to 18% in area with no degradation to critical-path delay. To compensate for the loss of routability, increased compute time and spare cluster inputs are required. Further investigation may yield modest area and delay reductions.

1. INTRODUCTION

A recent trend in FPGA architectural design is to use a *clustered architecture*, where a number of lookup tables (LUTs) are grouped together to act as the configurable logic block. The motivation for using clusters is manifold: to reduce area, to reduce critical path delay, and to reduce CAD tool runtime [1, 2, 9, 10]. This trend is followed by FPGAs from Xilinx's Virtex and Spartan-II families, as well as Altera's APEX and ACEX products. All of these FPGAs are based on clusters of 4-input lookup tables.

In a clustered architecture, the LUT inputs can be chosen from two sources: 1) a set of shared *cluster inputs*, which are signals arriving from other clusters via the general purpose routing, or 2) from *feedback connections*, which are the outputs of LUTs in this cluster. It has been common to assume that these internal cluster connections are *fully populated* or *fully connected*, meaning every LUT input can choose any signal from all of the cluster inputs and feedback connections combined. This arrangement can also be viewed as a full crossbar, where a switch or crosspoint exists at the intersection point of every LUT input and every cluster input or feedback connection.

In this paper, it is assumed that the connections within the cluster are made by multiplexers driving the LUT inputs, called *LUT input multiplexers*. These multiplexers tend to have a large number of inputs and, after including the requisite input buffers and controlling SRAM bits, contribute significantly to FPGA area.

FPGA 2001, February 11–13, 2001, Monterey, CA. Some typographical errors have been fixed (February 20, 2001).

A clustered FPGA is composed of a number of *cluster tiles* which are repeated in a simple array pattern during layout. Each tile is complete in that it includes the cluster logic (the flip-flops, LUTs, and LUT input multiplexers) as well as the general routing to interconnect them. Based on an area model stated later in Section 2, the LUT input multiplexers alone can consume 24 to 33% of the transistor area in a cluster tile. A breakdown of the area estimates for a number of such tiles is provided in Table 1.

The significant amount of area required by the LUT input multiplexers motivated the idea of removing switches from the full crossbar, or *depopulating* it, to result in a sparse crossbar. Naturally, depopulating the cluster raises the following questions:

1. Will depopulation save area, require greater routing area, or create unroutable architectures?
2. Will depopulation reduce or increase routing delays?
3. What amount of depopulation is reasonable?
4. How much area or delay reduction can be attained, if any?
5. What are the other effects of depopulating the cluster?

This paper addresses these questions using an experimental process of mapping benchmark circuits to clustered FPGA architectures and measuring the resulting area and delay characteristics.

1.1 Comparison to Prior Work

The use of fully-connected clusters likely stems from previous work [12] which suggests that inputs of a 4-LUT be fully connected to the routing channel. This provides enough routing flexibility to obtain minimum channel widths in non-clustered architectures, the area metric in use at that time. Since then, clustered architectures have become prevalent, CAD tools have improved, and area metrics have become more detailed.

Reducing the amount of connectivity within the cluster was recently explored using a simple striped switch layout [11]. Rather than modify the router, the T-VPACK packing algorithm was altered in such a way that routability of the cluster was still guaranteed. Unfortunately, the area improvement obtained using this technique was limited to 5% and delays increased up to 30%.

In this work, the packing algorithm was left unchanged. Instead, improved switch patterns were used, spare cluster inputs were added to the cluster, and modifications to the router were made to support these architectural changes. Although these spare inputs contribute to additional area, they also improve routability and reduce channel width requirements. Overall, a net area reduction of up to 18% with *no degradation* to critical-path delay was obtained.

Architecture		Fully Populated Cluster Tile Area (Number of Minimum-Width Transistor Areas)						
LUT size	Cluster size	LUT+FF		Routing		LUT Input Mux		Total
4	6	990	(10.6%)	6050	(65.0%)	2267	(24.4%)	9307
5	6	1840	(16.4%)	6321	(56.2%)	3080	(27.4%)	11241
6	6	3496	(24.4%)	6713	(46.9%)	4109	(28.7%)	14318
7	6	6831	(34.8%)	7645	(39.0%)	5146	(26.2%)	19622
7	10	11358	(32.3%)	12022	(34.2%)	11765	(33.5%)	35145

Table 1: Breakdown of cluster tile area. The routing area is an arithmetic average required to route 20 MCNC circuits.

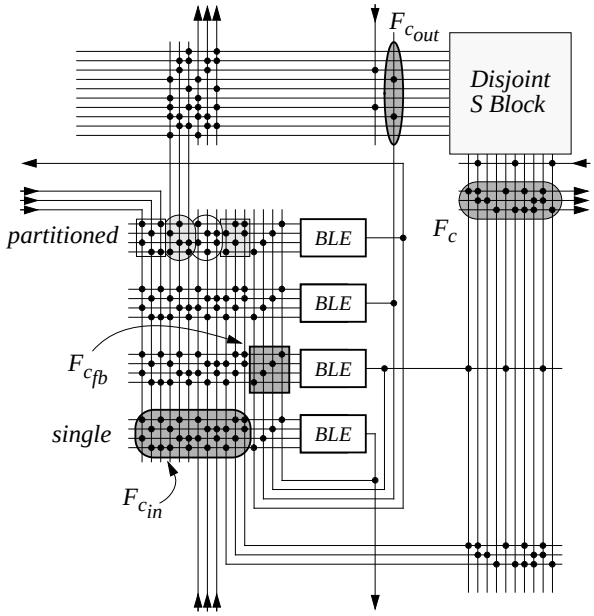

Figure 1: Details of the cluster tile architecture.

k	LUT size
N	cluster size
I	number of cluster inputs
I_{spare}	number of additional cluster inputs, used for routing only

Table 2: Cluster organization parameters.

$F_{c_{in}}$	cluster input to LUT input density
$F_{c_{fb}}$	LUT feedback to LUT input density
F_c	routing channel to cluster input density
$F_{c_{out}}$	cluster output to the routing channel density

Table 3: Switch density parameters.

1.2 Tradeoffs

Sparse clusters give the promise of reduced area, but one important tradeoff that must be made to realize this savings is increased routing time. In our experience, an approximate runtime increase of three to four times was observed. This increase may not be tolerable during early prototyping stages when design changes are frequent, but a less costly device could offset this inconvenience when an FPGA design shifts to volume production. Consequently, the premise of this paper is to evaluate the limits of area reduction that can be obtained using a high degree of CAD tool effort.

The remainder of this paper is organized as follows. In Section 2, the FPGA architecture is described along with the area and delay models. Section 3 discusses the experimental methodology and CAD tools used. Section 4 presents the results, and Section 5 concludes.

2. FPGA ARCHITECTURE

This section describes assumptions made about the FPGA architecture and the area and delay models.

2.1 Architectural Model

The architecture used in this study is a symmetrical, island-style FPGA containing interconnected clusters. The basic FPGA tile formed by a cluster and its routing channels is shown in Figure 1. This tile is drawn in a way to suggest a step-and-repeat layout that is possible, with wires on the left edge of one tile lining up with wires on the right edge of the adjacent tile.

One cluster contains N basic logic elements (BLEs), where one BLE contains a k-input LUT and a register. Each cluster has $I = \lfloor k(N+1)/2 \rfloor$ primary inputs which are used during packing [2]. As well, a cluster has I_{spare} additional cluster inputs which are reserved only for routing. These extra inputs are required to improve routability due to the restrictions imposed by sparse clusters. All of these cluster organization parameters are summarized in Table 2.

The cluster inputs are assumed to be logically equivalent, but they may connect to only some of the LUT inputs. The cluster input (and output) pins, which connect the cluster to the general routing, are evenly distributed on the four sides of the tile. Later in Section 4.3, we shall partition the cluster inputs into four groups based on which side they are placed.

2.2 Routing Architecture Details

Detailed routing architectural parameters were set to be the same as earlier studies [2, 4]. In the detailed routing architecture, 50% of the tracks are length-4 segments using tri-state buffers, the remaining tracks are length-4 segments using pass transistors, and clocks were assumed to be routed on a global resource. The *disjoint* switch (S) block was used, so signals entering the routing on track i must remain on that track number until the destination is reached. The number of i/o pads per cluster tile pitch was set to 5 for $N = 6$, and to 7 for $N = 10$.

The routing switch sizes (*i.e.*, buffer and pass transistor sizes) and wiring RC properties were computed assuming double minimum-spaced wiring and a fully-populated cluster tile size. For the $k = 4, N = 6$ architecture, the buffer was 6.1 times the minimum

size and the pass transistor was 12.2 times the minimum. The other architectures had larger tile sizes and used buffer sizes of 6.6, 7.6, 8.9, and 11.8. The pass transistor sizes were always chosen to be twice the corresponding buffer size.

Within a BLE, the LUT inputs are assumed to be logically equivalent and hence freely permutable. These inputs can select signals from two independent sources: either cluster inputs or feedback connections. The density of switches for these two regions, $F_{c_{in}}$ and $F_{c_{fb}}$, respectively, are independently controlled. These two parameters control the sparseness of switches inside the cluster.

For connections to outside the cluster, the inputs from and outputs to the general routing channels are selected using switch matrices with densities of F_c and $F_{c_{out}}$, respectively. The part of the general routing channel that connects to the cluster is commonly referred to as the connection block or C block.

The parameters controlling switch densities inside and outside of the cluster are summarized in Table 3.

Each BLE output directly drives a cluster output and a local feedback connection. The BLE outputs are assumed to be logically equivalent, allowing any function to be placed in any of the BLEs of the cluster. To achieve this output equivalence, every BLE is given the exact same input switch pattern.[1]

To improve routability, the routing tool takes advantage of the input and output equivalences just described. It may also replicate logic onto multiple BLEs in the same cluster, provided there are empty BLEs available.

2.3 Area Model

The area model used in this paper is the same buffer-sharing model used previously [2, 4], with a few minor changes described below. This model is based on the unit area of a minimum-width transistor (T), including the spacing to an adjacent transistor. As mentioned in [4], discussions with FPGA vendors have suggested that this, and not wiring, is the area-limiting factor.

All of the logic structures in the FPGA are modeled, including BLEs, the LUT input multiplexers, and the cluster routing, but not the padframe. For example, the area contribution of a pass transistor depends on the transistor width, and a buffer chain depends on the number of inverter stages as well as the required drive strength of each stage.

The drive strength requirement for a buffer is based on fan-out and is computed as follows. In general, it is assumed that a size B inverter in a buffer is sufficient to drive another inverter of size $4B$, or a total transistor gate width of $8B$. However, buffers driving the LUT input multiplexers, *i.e.*, the *cluster input buffers*, were sized differently. These buffers must drive a larger load created by the many levels of the LUT input multiplexer tree. This load is larger not only due to the depth of the tree, but also because diffusion is being driven. For these buffers, a size B was selected if the first level fan-out of the buffer [2] was loaded by a total diffusion width of $2B$, with the exception that drive strength was limited to be at least $7x$ and at most $25x$ minimum size. These approximations were made after examining HSPICE results [Ahmed and Wilton, *private communication*].

There were a few additional improvements made to the area model described in [4]. The previous LUT area model was slightly pessimistic and used the largest buffer required for all LUT inputs. In addition, it was optimistic when estimating cluster input buffer load, and this produced slightly understated area results. In this paper, every LUT input buffer and every cluster input buffer was sized according to its unique load requirements, yielding a slightly smaller LUT area but a larger cluster tile area than previously reported.

One simplification made while employing this area model was that the routing switch sizes were chosen beforehand based on the tile size of a fully populated cluster. As a result, the switches are larger than required, since any area saved by employing a sparse cluster would surely shrink the tile size. If recomputed using the smaller sparse cluster tile size, the routing switch sizes, hence the overall tile size, would be reduced. However, this simplification merely implies that area savings reported in this paper are conservative.

2.4 Delay Model

The delay model used here is the same path-based, critical-path delay model used previously [2, 4]. Timing parameters for all delay results were obtained using 0.18μm TSMC process information and detailed HSPICE circuit models. The precise delays along each path are computed in one of two ways, as described below.

Routing delays from the cluster output buffer to the cluster input buffer are computed using the Elmore delay [6] of the RC-tree network. The delays inside a cluster, however, are modeled as constant worst-case delay times (either rise or fall) extracted from HSPICE simulation results of a fully populated cluster. For example, these constant delays measure propagation from a cluster input to a LUT input, or the delay through the LUT.

Delay results in this paper are very conservative and may be overstated for two principal reasons. First, the routing delay results are overestimated because they ignore the tile size shrink that was mentioned in Section 2.3. Consequently, the routing wirelength parasitics and switch sizes are larger than required. Second, due to reduced loading and smaller cluster input buffers, internal cluster delays might be reduced if this simulation was repeated for sparsely populated clusters.

For these two reasons, the delay model used tends to produce pessimistic results for both components of delay: internal cluster routing and general purpose routing. Since internal cluster routing alone accounts for 35% of the critical-path delay on average [14], any savings from either component would lead to a measurable overall delay reduction.

3. METHODOLOGY

In general, the experimental methodology from [2, 4] was used. Benchmark circuits were optimized using SIS [13], mapped into LUTs using FlowMap and FlowPack [5], packed into clusters using T-VPACK [9], and placed using VPR [3, 4, 10] onto the smallest square FPGA that fits the circuit. In all experiments, the same packing and placement was used for each unique combination of circuit, LUT size and cluster size.

Below, the remainder of the CAD process is described, beginning with details about the routing stage, then a description of the router enhancements, and lastly a note on CAD tool parameter selection.

3.1 Routing Step

The last step of the CAD flow involves routing a placed netlist in the detailed routing architecture. The routing tool used here is based on a modified version of VPR 4.30 which was tailored specif-

[1] An alternative architecture with different input switch patterns for each BLE can be built. Such an architecture would require a full permutation stage to reorder all of the BLE outputs to the cluster outputs and feedback connections. This could be done by fixing $F_{c_{fb}} = F_{c_{out}} = 1.0$, for example, or by using N additional $N-to-1$ multiplexers. We did not consider such an architecture here.

[2] Note that this fan-out can be significantly lower in a sparsely populated cluster, and this area savings is counted.

ically for sparse clusters. This version of VPR includes the latest timing-driven packing and placement enhancements [9, 10].

During routing, the minimum channel width required to route, W_{min}, was found using a binary search. Afterwards, a final low-stress routing was done with $W = 1.3 \cdot W_{min}$ tracks to compute area and delay statistics. This procedure models the way FPGAs are actually used; designers are seldom comfortable working on the edge of capacity or routability.

The final low-stress routing actually failed in 34 out of 3980 (0.9%) circuit/architecture combinations, usually due to slow convergence or switch pattern interference.[3] To resolve this, one, two, then three additional tracks were added to the channel. This strategy was sufficient to route all but four of the troublesome cases — the three underlying architectures for these cases were deemed unroutable, so they were abandoned from further consideration in this paper.

Also, if the binary search was unable to find a reasonable minimum channel width ($W_{min} \leq 240$) for any of the circuits, the architecture was deemed unroutable and abandoned. Consequently, every architectural result presented in this paper was obtained by routing all of the benchmark circuits.

All area and delay results are averages obtained from placing and routing the 20 largest MCNC benchmark circuits [7]. Area is computed as the geometric average of the *active FPGA area*, which is defined below. The geometric average ensures that the circuits are all weighted equally, independent of the size of the circuit. Delay results are also the geometric average of the critical-path delay for each benchmark circuit.

Active FPGA area is the area, in units of minimum-width transistor areas, of one cluster tile (including its routing) times the number of clusters actually used by the benchmark circuit. This measurement was used in [1, 2] to better distinguish packing efficiency. We have chosen to use the active FPGA area metric here to be consistent with those results.[4]

3.2 CAD Tool Enhancements

Originally VPR routed only to cluster input pins because fully-connected clusters could guarantee the routability of cluster inputs and feedback connections. Extensive modifications to VPR were necessary to route sparsely populated clusters. For example, the routing graph, timing graph, and netlist structures had to be altered to accommodate the cluster feedback nets and the location of every BLE sink. As well, other changes were necessary to permit nets to enter a cluster more than once to improve routability.

The switch pattern generator from [8] was integrated into VPR to create the switch patterns for the LUT input multiplexers. This generator first distributes switches to balance the fan-in and fan-out of each wire, usually in a random pattern. A greedy improvement strategy is then followed which roughly maximizes the number of distinct output wires reached by every pair of input wires. To accomplish this, switches are randomly selected, first in pairs, then singly, and moved only if the fan-in/-out constraints are kept and the aforementioned cost improves. Using this technique, the switch patterns within a cluster are individually well-designed.

Other switch patterns in the routing fabric, namely the cluster input and output patterns, use the original VPR switch placement generators. Additionally, we *have not* attempted to optimize the cascading of the the cluster input multiplexers and LUT input multiplexers, except as noted below in Section 4.3. This extension to the work is nontrivial and left for future investigation.

[3]Of the failed combinations, 20 of them had $I_{spare} = 0$ and the remainder had $F_{c_{in}} \leq 0.25$.

[4]Note that the results in [2] use a different process technology.

Cluster feedback connections are also sparsely populated, and this may cause some problems during routing. In particular, there may be too few switches to satisfy all possible feedback connection patterns, so feedback signals are also permitted to leave the cluster and re-enter through the cluster inputs. This may cause speed degradation, or some netlists may become unroutable because all cluster inputs are used.

There is no immediate solution for the speed degradation problem, but we address the routability problem by assuming there are *spare cluster inputs* in the architecture. These spare inputs improve routability by providing the router with more choices [8]. The number of spare inputs given, I_{spare}, is specified prior to routing as a fixed part of the architecture. For convenience, the packing tool adds these as part of the netlist and the router automatically uses them.

The effectiveness of the modified VPR router was validated against the original version of VPR. Both routers obtained similar delays, channel widths and area results for fully populated clusters using a variety of cluster and LUT sizes.

3.3 Tool Parameters

In general, the packing, placement and routing tools were run in timing-driven mode using their default parameters. Some non-standard command line switches were used for routing — these are shown in Table 4 and described in further detail below.

The number of router iterations had to be increased beyond the default value of 30, partly because sparsely populated clusters require additional routing effort. As well, large variations were observed in delay results because the router parameter that increases the cost of nets sharing wires between iterations (`pres_fac_mult`) was too high.

The impact of reducing this parameter on runtime and average critical-path delay of the low-stress route can be seen in Figure 2. As well, the range of average delay values (across all benchmarks) for each architecture is shown using error bars. This wide range made it difficult to distinguish architectures with low delay from those with higher delay. Clearly, increased routing effort was required to reduce the delay variation, but we feel this was time well-spent. Without this effort, we would be unable to conclusively compare the delay results of the different architectures.

For this experiment, the maximum number of router iterations was set to 300. On average, however, the number of iterations used increases from 23 to 160 in a manner that very closely follows the increase in runtime. The router values shown in Table 4 were chosen in reference to these results.

4. RESULTS

This section gives the area and delay results from placing and routing 20 MCNC benchmark circuits. In all cases, only the geometric average is used for FPGA area and critical-path delay. Initial experiments determined the best routing parameters, then these parameters were used to evaluate the area and delay of sparse cluster architectures.

4.1 Key to Curve Labels

In the following graphs, each curve represents a family of architectures parameterized along the *x*-axis. Each curve label describes the specific architecture parameters in the following order:

$$k \quad N \quad I_{spare} \quad F_{c_{in}} \quad F_{c_{fb}}$$

These parameters are fully described in Tables 2 and 3. Where the value of a parameter is given as 'X', that simply means the parameter is being varied along the *x*-axis.

Tool	Additional Parameters
T-VPACK	*default*
VPR placement	*default*
VPR binary search	`-pres_fac_mult 1.3 -max_router_iterations 250`
VPR final route	`-pres_fac_mult 1.05 -max_router_iterations 300`

Table 4: CAD tools and non-default parameters used.

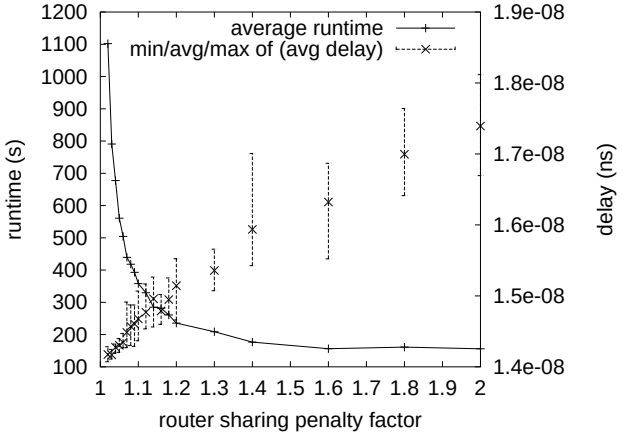

Figure 2: Variation in average critical path delay is shown as a function of the router sharing penalty factor. To reduce the variation (and the delay), longer runtime is required.

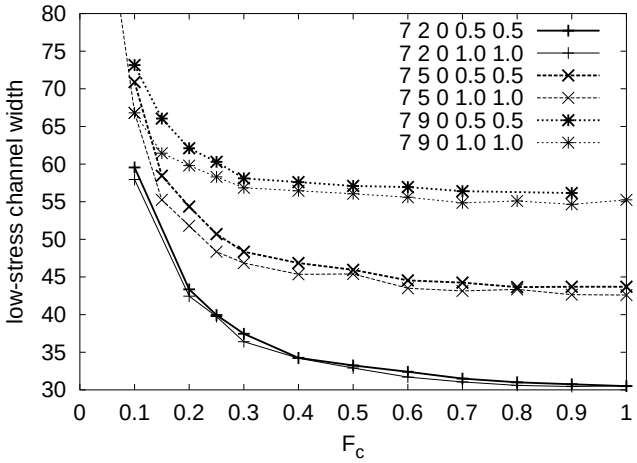

Figure 3: F_c impact on channel width.

4.2 Routing Architecture Selection

To explore the sparse population of switches inside the cluster, it is first necessary to establish a good routing architecture outside of the cluster. Hence, the best values for F_c and $F_{c_{out}}$ need to be selected beforehand. We chose to find the best switch density that would give minimum area rather than delay. To generate the results for this expediently, the number of router iterations was limited to 75, but all other parameters were left at their default values.

4.2.1 Selecting F_c for minimum area

The density of switches connecting channel wires to cluster inputs in the C blocks is called F_c. We wish to determine the value of F_c that would result in a minimum-area FPGA.

The choice of F_c depends on the effectiveness of the CAD tools and the size of the C block, determined by the channel width, W, and the number of cluster inputs, I. It has been our experience that I is the most important factor influencing the choice of F_c.

Routing experiments were done for $k = 7$ architectures, varying N from 2 to 9. This large LUT size was chosen because we are mostly interested in the effects of having a large number of cluster inputs. Both full (100%) and sparse (50%) population levels inside the cluster were tried. The 50% density was chosen because this was almost always routable without adding spare inputs, hence $I_{spare} = 0$ here.

The average low-stress channel width required to route the benchmark circuits, W, is presented in Figure 3 for a variety of F_c values. Only three cluster sizes are illustrated, but the other results are similar. From these results, it can be seen that choosing $F_c > 0.4$ has little impact on channel width. Although not shown, this is particularly true for $N > 3$.

Interestingly, the channel width results are very similar for both sparse and fully-populated clusters. Sparse clusters typically required only 2 to 4 more tracks than the corresponding fully populated ones. Hence, the sparse architecture is still quite routable at the 50% population level.

Although channel width is not hindered by a large value of F_c, having more switches than necessary will contribute to an area increase. Figure 4 also shows the active FPGA area versus F_c for cluster sizes 2 and 9.[5] Again, similar results were obtained for cluster sizes 3 through 8.

One unexpected area result is that the 50% sparse cluster near $F_c = 1.0$ always uses *fewer* transistors than the minimum-area fully-populated cluster. This can be seen in Figure 4 where point B is lower than A, and D is lower than C. This trend holds for the other cluster sizes as well. Hence, it is better to sparsely populate the clusters than the general routing, a non-intuitive result. One reasonable explanation for this is there are about twice as many LUT input multiplexers as cluster input multiplexers, even though the cluster input multiplexers can easily have twice as many inputs (based on the channel width).

Another result shown in Figure 4 is a significantly larger area reduction for $N = 9$ than $N = 2$. The reduction is so large that the $N = 9$ architecture goes from using more area (curve C) than the corresponding $N = 2$ architecture (curve A) to using less area (curves D and B). This result shows how sparse clusters can shift the optimum design point towards larger clusters. For example, in this $k = 7$ architecture, the fully-populated cluster should contain between 4 and 6 LUTs to be area-efficient. However, the 50% sparsely-populated cluster should contain between 4 and 9 LUTs. Further investigation of different cluster sizes is left as future work.

The values of F_c producing the lowest area for each cluster size, *i.e.*, for each value of I, are shown in Figure 5. It is remarkable that

[5]Notice that the sparse $F_c = 1.0$ result is missing for $N = 9$ in Figures 3 and 4 because VPR was unable to route the *clma* circuit under low-stress conditions due to slow convergence.

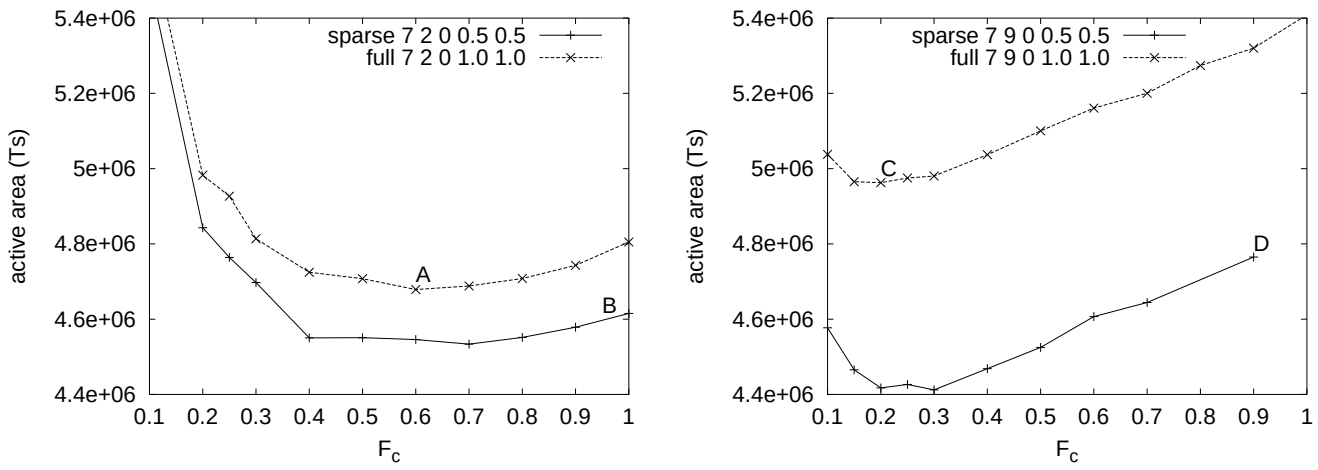

Figure 4: F_c impact on area for cluster sizes of 2 and 9. Intermediate cluster sizes gave similar results.

the sparse and fully populated cluster results are so similar. This can be partly attributed to the relative flatness near the minimum area. For $N = 9$, varying F_c from 0.1 to 0.5 causes less than 5% change in area. Hence, precise F_c selection is not critical, provided it is large enough to be routable, yet not wastefully large.

For the remainder of the results in this paper, it was determined that a fixed value of F_c would not significantly hinder area results. Rather than using the minimum-area F_c values from Figure 5, we felt that having a few more switches in the routing (by having a slightly larger F_c) would be helpful as clusters were made even more sparse (internally). This is especially important because no effort was made to tune the two switch patterns together and we wished to avoid possible interference patterns. Hence, we chose to set $F_c = 0.5$ for the $N = 6$ architectures and $F_c = 0.366$ for the $k = 7, N = 10$ architecture. These particular values were chosen because they were used in previous work [2, 4] and this gives us the most comparable results.

4.2.2 *Selecting* $F_{c_{out}}$

Previous experiments have shown that $F_{c_{out}} = 1/N$ is adequate for routing in fully populated architectures [4]. Considering the similarity of the F_c area results between sparse and fully populated architectures, it was decided that modifying $F_{c_{out}}$ would have insignificant impact in a sparsely connected architecture. Hence, $F_{c_{out}} = 1/N$ was used for all results.

4.3 Partitioning of Cluster Inputs

Additional net delay can be caused by sparsely populated clusters because some LUT inputs may not be reachable from particular sides of the cluster. For example, consider the case when some LUT input connections have already been formed, and the last remaining input signal is being made. A lack of switches inside the cluster may cause that net to enter the cluster from a more distant side. The result is increased delay.

We investigated this problem by trying a *single* switch matrix for all cluster inputs, and one which was *partitioned* into four smaller switch matrices, one for each input side. The partitioned matrix addresses the above problem by ensuring that all of the cluster inputs from any particular side can reach all of the LUT inputs. It also has a weakness though: these smaller switch matrices are not carefully designed to couple together well. Each partitioned matrix is derived from the same basic switch pattern, but each has its own

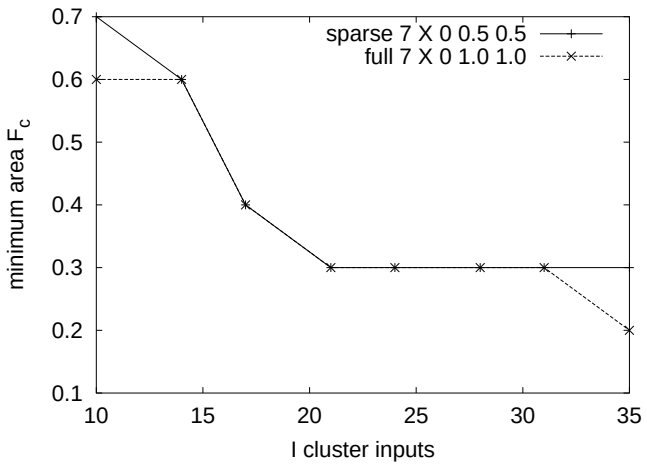

Figure 5: Best F_c corresponding to minimum area as a function of I cluster inputs.

permutation of the rows (or outputs) to balance the fan-in of the LUT inputs. These matrices, but not the permutation pattern, are illustrated in Figure 1.

Both switch designs were routed in a $k = 7$, $N = 10$, $F_{c_{in}} = F_{c_{fb}} = 0.43$ architecture. Both designs required identical transistor area, and the *partitioned* matrix was only about 1% faster. Although this is not significantly faster, it was used for subsequent results in this paper since it may help with some pathological cases.

4.4 Sparse Cluster Area Results

The primary motivation for depopulating clusters is to reduce the area, and subsequently the cost, of an FPGA. In Section 4.2, it was determined that simply depopulating the cluster to 50% is more effective at reducing area than choosing the proper value of F_c. In this section, further depopulation of the cluster is explored.

To reduce the number of routing experiments, it was decided to fix the cluster size to $N = 6$ and vary the LUT sizes from 4 through 7. That particular cluster size was selected because it generated near-minimum area and area-delay results for fully populated clusters with all of these LUT sizes. The larger LUT sizes are especially interesting because they require larger input switch matrices, hence

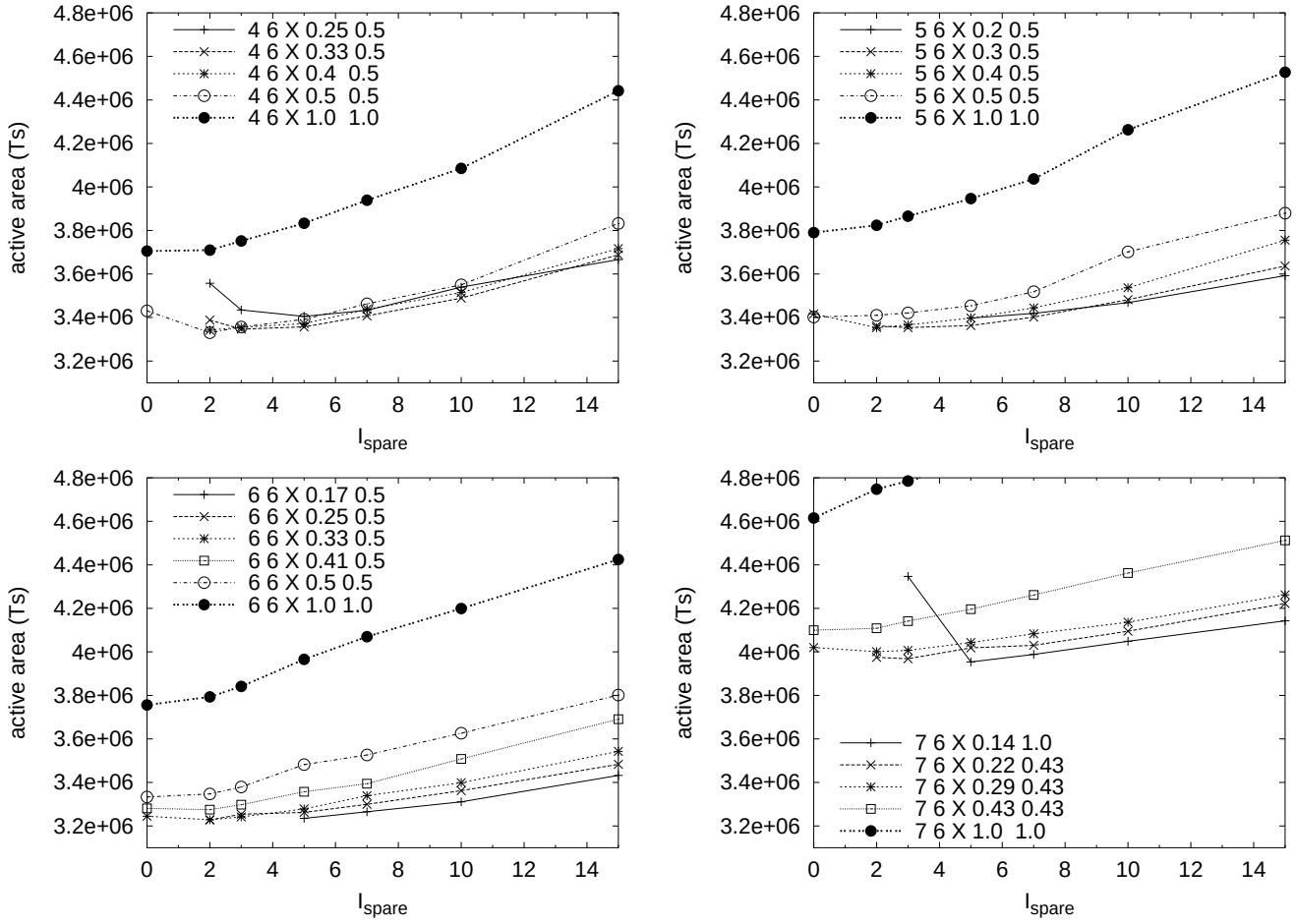

Figure 6: Active FPGA area of fully and sparsely populated clusters.

offering more potential for depopulation. One additional architecture with $k = 7, N = 10$ was chosen to study an even larger number of inputs entering the cluster.

A number of preliminary routing experiments were run with a wide range of values for $F_{c_{in}}$ and $F_{c_{fb}}$. From these results, which are not shown here, it was confirmed that $F_{c_{fb}}$ has less influence on area. As $F_{c_{fb}}$ was reduced below 50%, a number of circuits would no longer route. It was determined that $F_{c_{fb}}$ of 50% (or $3/7 = 43\%$ for $k = 7$) was as low a value as could be tolerated. Similar preliminary sweeps indicated that $F_{c_{in}} \geq 0.5$ was nearly always routable, so area reduction should concentrate on more sparse values.

The area results from routing the four LUT sizes are shown in Figure 6. In these graphs, each curve represents the geometric average of active FPGA area for a fixed value of $F_{c_{in}}$. The number of spare inputs is varied along the *x*-axis. The sparse cluster results should be compared against the bold curve representing the fully-populated cluster area.

The most apparent trend in these curves is a gentle dip, then a general upward climb in area as I_{spare} is increased. The upward trend is an expected result, since the spare inputs will require additional cluster input multiplexers. The dip is caused by a rapid initial decline in average channel width, which then gradually reaches a 5% to 20% reduction (10% is typical).

A number of data points are missing in Figure 6, specifically for small I_{spare} values. This is because one or more benchmark circuits could not be routed on the architecture. Hence, although they contribute to area reduction in only a few cases, it is *essential* to have these spare inputs to make sparse clusters routable. Typically, between two to five spare inputs are required to make the architecture routable and attain the lowest area.

The lowest-area architectures from Figure 6 are summarized in Table 5. As well, the large $N = 10$ cluster architecture is included. With these architectures, a 10 to 18% area savings is achieved. As mentioned earlier, between two and five spare inputs is sufficient to achieve most of this savings, which is surprising since this only about one spare input per side.

A breakdown of the cluster tile area is given Table 6. For 4-input LUTs, there was a slight decrease in routing area because the spare inputs helped reduce average channel width. The 5- and 6-input LUTs cases did not achieve the same benefit because the spare inputs contributed more to area than the amount saved by the slight channel width reduction. The two 7-LUT architectures had an increase in routing area due to the spare inputs and a channel width increase. However, the sparse switch populations produced a net area savings of 14% and 18%, with the larger cluster benefitting more. With respect to the entire tile, depopulating the clusters was very effective at reducing the relative LUT input multiplexer size from the 24–33% range down to 12–18%.

One very interesting result from this data is that a sparse cluster of six 6-input LUTs is slightly more area-efficient (3%) than six

Architecture				Best Sparse Parameters			Channel Width (arith. avg.)		Active FPGA Area ($\times 10^6$ Ts)		
k	N	I	F_c	I_{spare}	$F_{c_{in}}$	$F_{c_{fb}}$	Fully Populated	Best Sparse	Fully Populated	Best Sparse	Savings
4	6	14	0.5	2	0.5	0.5	47.9	45.9	3.71	3.33	10.1%
5	6	17	0.5	2	0.4	0.5	46.4	45.6	3.79	3.35	11.5%
6	6	21	0.5	2	0.33	0.5	44.3	43.5	3.76	3.23	14.0%
7	6	24	0.5	5	0.143	0.43	43.8	44.6	4.62	3.95	14.3%
7	10	38	0.366	10	0.143	0.43	53.7	55.1	4.96	4.03	18.8%

Table 5: Active FPGA area savings obtained by depopulating switches inside the cluster.

Architecture		Tile Area (Number of Minimum-Width Transistor Areas)									
		Fully Populated Cluster					Best-Area Sparse Cluster				
k	N	Total	LUT+FF	Routing	LUT Input Mux		Total	LUT+FF	Routing	LUT Input Mux	
4	6	9307	990	6050	2267	(24.4%)	8380	990	5960	1430	(17.1%)
5	6	11241	1840	6321	3080	(27.4%)	9964	1840	6371	1753	(17.6%)
6	6	14318	3496	6713	4109	(28.7%)	12343	3496	6732	2115	(17.1%)
7	6	19622	6831	7645	5146	(26.2%)	16879	6831	8120	1928	(11.4%)
7	10	35145	11358	12022	11765	(33.5%)	28646	11358	12990	4298	(15.0%)

Table 6: Breakdown of cluster tile area. The routing area is an arithmetic average for all circuits.

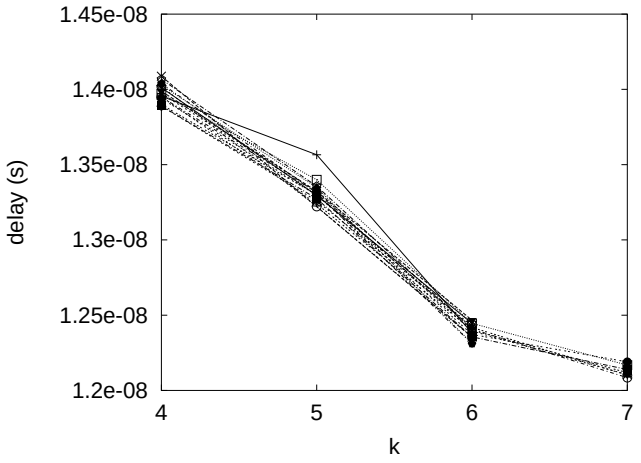

Figure 7: Delay decreases with LUT size.

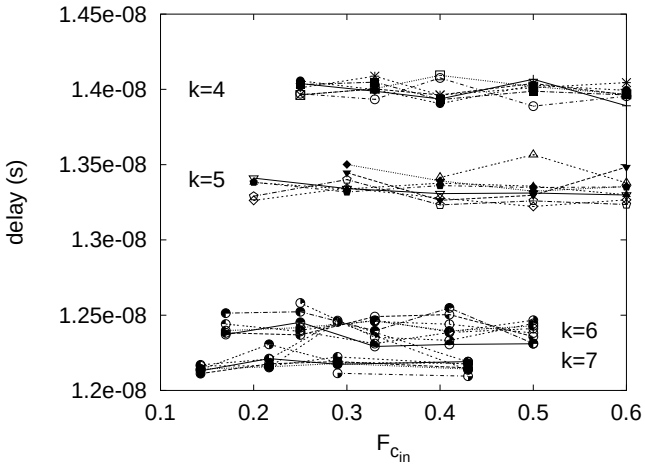

Figure 8: Delay is not influenced by $F_{c_{in}}$. Similar results indicate it is not influenced by I_{spare} or $F_{c_{fb}}$.

4-LUTs in a sparse cluster. This is a departure from previous work which has consistently shown that 4-LUTs achieve lower area, albeit in fully populated clusters. The reason for this difference is simple: larger LUTs provide more opportunity for depopulation. This concept is supported by previous work which has shown that sparse crossbars with more outputs require fewer switches for the same level of routability [8].

4.5 Sparse Cluster Delay Results

As mentioned earlier, reduced switch densities may cause an increase in delay due to an increase in bends or wire use to achieve routability. Although delay may decrease for other reasons such as reduced loading, we chose to be conservative and ignore these possible benefits.

The curves in Figure 7 show the impact that varying the LUT size has on delay for a few of the $N = 6$ architectures. The curve labels identifying the architectures have been omitted for clarity, since only trends need to be observed. The important thing to notice is that, for all architectures, delay goes down as k increases.

Similarly, Figure 8 shows the change in delay as the switch density $F_{c_{in}}$ is varied. It is apparent in the graph that curves of the same LUT size are all grouped together. In particular, the 4- and 5-LUT data is easily distinguished from the 6- and 7-LUT data. The flatness of all of these curves illustrates how little impact $F_{c_{in}}$ has on delay.

Analysis of delay while varying I_{spare} or $F_{c_{fb}}$ shows the same result: delay is independent of these parameters. Even though sparse clusters present a challenge to the router and remove many choices, and even though some feedback connections must leave the cluster and re-enter through the general-purpose routing, the router still has enough freedom to ensure that nets on the critical path remain on the fastest paths to the critical sinks.

4.6 Sparse Cluster Area-Delay Product

The previous two sections presented results indicating the 6-LUT had the lowest area and the 7-LUT had the lowest delay. When the

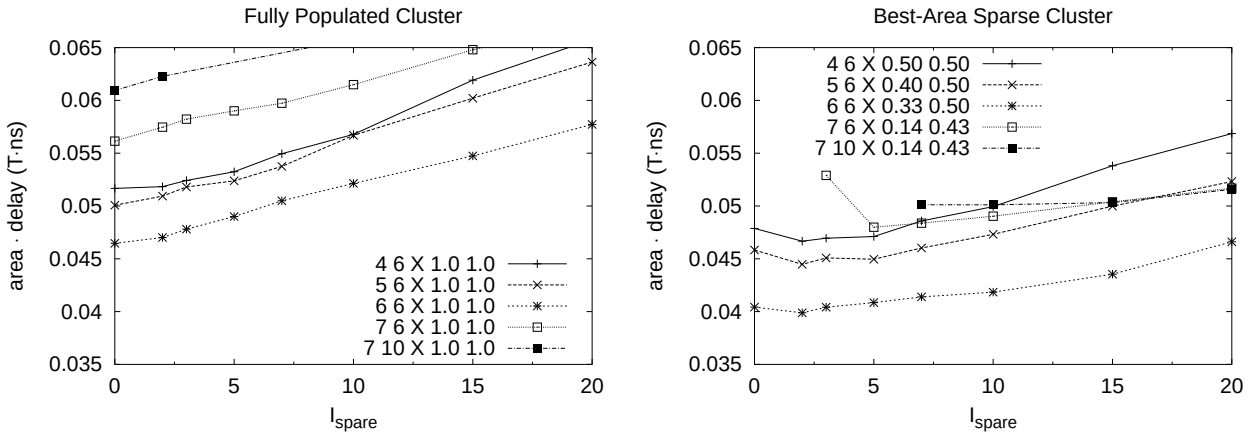

Figure 9: Area-delay product results for fully-populated and best-area sparse architectures.

		Average Runtime (seconds)			Average # Routing Iterations		
		VPR 4.30	Modified VPR		VPR 4.30	Modified VPR	
k	N	Fully Populated		Best Sparse	Fully Populated		Best Sparse
4	6	70	153	150	84	86	86
5	6	72	183	205	93	91	93
6	6	57	177	178	84	86	88
7	6	53	188	350	88	83	109
7	10	43	177	275	96	94	116

Table 7: Average runtime and number of routing iterations for the final low-stress route (arithmetic averages of 20 benchmarks). Runtimes were collected on an 866MHz Pentium III computer with 512MB of SDRAM.

area and delay results are combined in the form of an area-delay product, the 6-LUT emerges as the superior logic block choice. This metric is important because it indicates when the best trade-off is being made between using an additional amount of area for a similar relative gain in clock rate (or vice versa). For example, it is directly useful in FPGA-based computation because the computation rate is a product of both the clock rate and parallelism.

The best sparse area-delay product organizations are compared to their fully-populated versions in Figure 9. The area-delay product improves for every LUT size due to the area reduction. The overall best sparse architecture containing 6-LUTs is about 14% more efficient than one containing 4-LUTs, and about 22% more efficient than the traditional fully-populated 4-LUT cluster.

4.7 Routing Runtime with Sparse Clusters

The removal of switches inside the cluster also removes the routability guarantee of the cluster. Consequently, the router must pay attention to all of the wires and switches within the cluster, so it is expected that additional runtime effort is required to complete the route.

The average runtime and average number of iterations required for routing the different architectures are shown in Table 7. Results are presented for fully populated clusters to compare the original VPR 4.30 to the modified one. As well, the modified VPR can be compared against itself to study the additional impact of routing the best-area sparse clusters.

Generally, the modified VPR currently runs about three to four times slower than the original version when fully populated clusters are used. Even though runtime has increased, the number of router iterations used is practically unchanged. The main reason for the slowdown comes from the increased number of wires and switches in the architecture that must be examined with each iteration: all cluster inputs now have connections to many LUT inputs, and nets are allowed to enter a cluster more than once. This causes the router to evaluate many more routing paths before making a decision.

It is worthwhile to note that having larger LUT sizes and cluster sizes reduces the amount of work that VPR 4.30 must do, so runtime decreases. This benefit was not realized in the modified VPR because the amount of wiring inside the cluster also increases, keeping runtime relatively flat.

The additional runtime needed to route the best-area sparse architectures is also shown in Table 7. For $k = 4,5,6$ the runtime and the number of iterations is similar, for $k = 7$ runtime nearly doubled and the number of iterations increased by 25–30%.[6] This increase in the average is caused by a large increase in four of the normally difficult-to-route circuits. The need for more router iterations indicates these architectures are barely routable, probably because $F_{c_{in}}$ is so low, even though these circuits are being routed using the low-stress channel width.

Increasing routability by increasing I_{spare} to 15 for the $k = 7$, $N = 10$ architecture reduced runtime to 210 seconds and 97 iterations. Hence, the amount of area savings can also be balanced against the runtime effort.

[6]The amount of searching done in each iteration may increase as the search space expands, so each iteration's runtime may increase.

5. CONCLUSIONS

This work has studied the area and delay impact of sparsely populating the internal cluster connections in a clustered architecture. At the expense of three to four times the compute time, an area savings of 10 to over 14% was realized by sparsely populating the cluster internals of 4-, 5-, 6-, and 7-input LUT architectures containing 6 LUTs per cluster. A larger cluster size of ten 7-LUTs obtained an 18% area savings. It was also observed that the additional router effort and reduced routing flexibility did not degrade critical-path delay.

A fixed number of spare inputs were added to each cluster. These inputs are used only by routing, and are not used or required for packing. By adding up to 15 spare inputs, the channel width decreased by about 10% in most architectures, whether full or sparsely populated. Although sparse clusters on their own impose a small increase in channel width, the spare inputs reduce the channel width, resulting in a small, net savings.

The channel width reduction typically produced a net savings in routing area alone when up to seven spare inputs were added, but resulted in a net increase thereafter. Of course, the cluster area (excluding the routing) always increased with the addition of spare inputs. However, this area increased at a slower rate in more sparsely populated clusters, as expected. When added to the routing area, most architectures became less efficient after more than five spare inputs were employed.

The increase in routability and decreases in channel width and area indicate that it is best to force the packing algorithm to leave a few spare inputs (two or three) for the router.

One interesting outcome of this work is that, contrary to popular belief, it is more area-efficient to depopulate only the LUT input multiplexers than it is to depopulate only the cluster input multiplexers (*i.e.*, the C blocks) in the general routing. The reason for this is that, due to input sharing in a cluster, there are about twice as many LUT input multiplexers than cluster input multiplexers. Of course, depopulating both regions provides even more savings.

Another interesting observation is that 6-LUTs become more area efficient than 4-LUTs when sparse clusters are employed. This was entirely attributable to the more sparse pattern that could be used in the 6-LUT case.

The area and delay results in this paper used conservative estimates and ignored secondary effects which would improve results further. In particular, the tile size and the subsequent routing switch size reduction from sparse cluster use should lead to additional area and delay reduction. Delay improvement may also come from reduced loading inside the cluster and by generally using larger cluster sizes, which are more area-efficient when using sparse clusters.

It is reasonable to expect that larger cluster sizes may produce an even larger area savings due to the large amount of area concentrated in the LUT input multiplexers.

Future work in this area will include effort to jointly design the LUT input switch matrices with the cluster input multiplexers to avoid switch pattern interference. Additional constraints such as carry chains or other local routing may impact sparse cluster design and should be evaluated. A wider variety of cluster sizes, particularly the effectiveness of large clusters, should also be explored. The area savings from sparely populated clusters will reduce tile size, but the subsequent area and delay reduction from using smaller routing switches should also be quantified. The delay improvements arising from reduced loading and larger cluster sizes should be investigated. Also, efforts should be made to improve the runtime of the router while still retaining the area savings.

An interesting extension of this work would involve tighter coupling with the packing stage. For example, under special circumstances, it may be reasonable to have the packing tool use the spare inputs reserved for routing. Before doing this, it could first do a routability test to verify whether the potential cluster of logic blocks is routable. Since this shouldn't be a common case, it can be done with reasonable CPU effort. This may increase the usefulness of the FPGA architecture for subcircuits which have wide fan-in (or poor input sharing), such as finite state machines.

6. ACKNOWLEDGEMENTS

The authors wish to thank Elias Ahmed, Mike Sheng, and Steve Wilton for HSPICE timing results and helpful discussions.

7. REFERENCES

[1] E. Ahmed. The effect of logic block granularity on deep-submicron FPGA performance and density. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 2001.

[2] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In *ACM/SIGDA Int. Symp. on FPGAs*, pages 3–12, 2000.

[3] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic*, pages 213–222, 1997.

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

[5] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design*, pages 1–12, January 1994.

[6] W. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, pages 55–63, January 1948.

[7] C. B. Laboratory. *LGSynth93 suite*. `http://www.cbl.ncsu.edu/www/`.

[8] G. Lemieux, P. Leventis, and D. Lewis. Generating highly-routable sparse crossbars for PLDs. In *ACM/SIGDA Int. Symp. on FPGAs*, pages 155–164, Monterey, CA, February 2000.

[9] 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 Int. Symp. on FPGAs*, pages 37–46, 1999.

[10] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *ACM/SIGDA Int. Symp. on FPGAs*, pages 203–213, 2000.

[11] M. I. Masud. FPGA routing structures: A novel switch block and depopulated interconnect matrix architectures. Master's thesis, Department of Electrical and Computer Engineering, University of British Columbia, December 1999.

[12] J. Rose and S. Brown. Flexibility of interconnection structures in field-programmable gate arrays. *IEEE Journal of Solid State Circuits*, 26(3):277–282, March 1991.

[13] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit analysis. Technical Report UCB/ERL M92/41, University of California, Berkeley, May 1992.

[14] M. Sheng and J. Rose. Mixing buffers and pass transistors in FPGA routing architectures. In *ACM/SIGDA Int. Symp. on FPGAs*, 2001.