On Biased and Non-Uniform Global Routing Architectures and CAD Tools for FPGAs

Computer Systems Research Institute (CSRI) Technical Report #358
University of Toronto

Vaughn Betz and Jonathan Rose
{vaughn, jayar}@eecg.utoronto.ca

1 Introduction

In recent years Field-Programmable Gate Arrays (FPGAs) have seen explosive market growth because they offer instant manufacturing and much lower non-recurring engineering costs than Mask-Programmed Gate Arrays (MPGAs). FPGAs enable fast manufacturing and low development costs because all of their logic and routing resources are prefabricated and are customized in the field by the designer [1].

The prefabrication of routing resources in an FPGA implies that the number of routing tracks in each channel is set by the manufacturer. It is vital that these routing resources be distributed in a manner that allows their efficient utilization by the largest class of circuits. If there are too few tracks in some area of the chip then many circuits will be unroutable, while if there are too many tracks, they may be wasted.

This paper addresses several questions concerning the distribution of routing tracks across an FPGA. Essentially we are investigating if the intrinsic properties of circuits lead them to map most efficiently to a certain routing architecture. The first question addressed is whether or not there should there be a directional bias to the routing. If so, what amount of bias is best? Figure 1(a) illustrates a *directionally-biased* FPGA in which the horizontal channels contain more tracks than the vertical channels. Commercial FPGAs with both unbiased routing [2, 3] and biased routing [4, 5] exist, so this question has clear commercial relevance. To ensure our results are applicable to the broadest class of FPGAs, we determine the best directional bias for FPGAs with different logic block pin positions and aspect ratios.

The second question we address is whether all channels in the same direction in an FPGA should be the same width or whether some channels should be wider than others to facilitate rout-

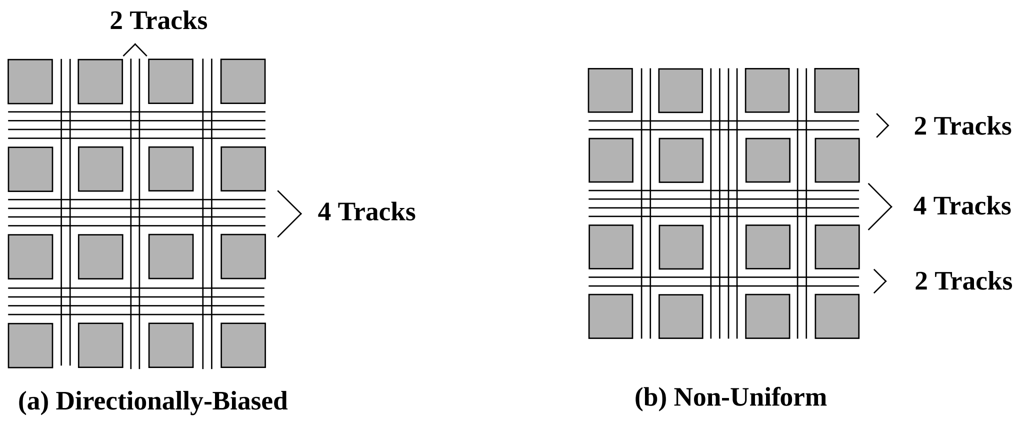


Figure 1: Types of Global Routing Architectures.

ing in congested regions. We refer to architectures in which the channels in some regions are wider than the channels in others as *non-uniform* routing architectures, as illustrated in Figure 1(b)[1]. Many in the FPGA community believe that most routing congestion occurs near the center of an FPGA, and hence channels in this region should be wider than the channels near the edges. In fact, AT & T has designed an extra-wide channel in the center of their latest device to improve routability [6]. In addition, board-level constraints often force designers to fix the position of an FPGA's I/Os, and some believe that this increases congestion near the chip edges so that the channel between the pads and the logic should be made extra wide. Xilinx has an FPGA with a wide channel between the pads and logic, at least partially to improve routability when the I/O locations are fixed [7]. In this paper, we determine the best distribution of tracks across an FPGA both when the I/O assignment to pads is unconstrained and when it is fixed in a poor configuration.

We evaluate FPGA architectures experimentally; benchmark circuits are placed and routed into FPGAs with different global routing architectures to determine the relative area consumed by the circuit in each architecture. In order to obtain meaningful results, the CAD tools used to place and route these circuits must understand and take advantage of the biased and non-uniform nature of these architectures. We have created a new placement and routing tool which reads a parameterized description of an FPGA architecture and aggressively seeks to minimize congestion and fully utilize the channels of the specific architecture during both placement and global routing.

The organization of this paper is as follows. Section 2 outlines the CAD flow used to evalu-

1. Note that any given channel will always have the same number of tracks along its entire length. We did not consider varying the channel capacity along its length as this leads to a very difficult layout problem.

ate the different FPGA architectures. Section 3 describes the algorithms and performance of our placement and routing CAD tools. We evaluate the area-efficiency of FPGAs with differing amounts of directional routing bias in Section 4. In Section 5 we address the uniform vs. non-uniform channel thickness question. Finally, we summarize our results and conclusions.

2 Experimental Methodology

To compare the area-efficiency of the different global routing architectures we technology-map, place and route 26 of the largest MCNC benchmark circuits [8] into each architecture. In this section we describe the CAD flow, the area model, and several important architectural details that were assumed.

2.1 CAD Flow

Figure 2 provides an overview of the CAD flow. First, the SIS [9] synthesis package is used to perform technology-independent logic optimization of each circuit.[2] Next, Flowmap [10] is

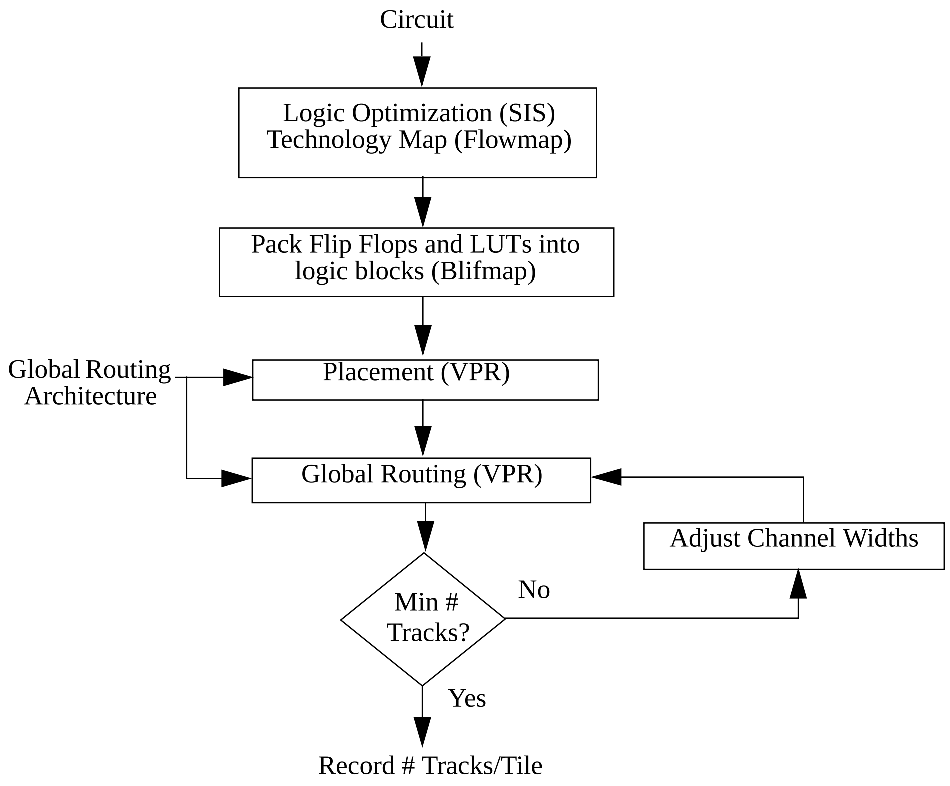


Figure 2: Overview of Architecture Evaluation Flow.

2. The circuit is optimized by both script.rugged and script.algebraic and the smaller result is taken.

used to technology-map each circuit into four-input look-up tables (4-LUTs) and flip flops. The logic block used in these experiments contains a 4-LUT and a flip-flop, in the configuration illustrated in Figure 3. A custom-built program (blifmap) packs the 4-LUTs and flip flops together into these logic blocks.

The netlist of logic blocks and a description of the FPGA global routing architecture are then read into the placement and global routing tool, VPR. This program places the circuit, and then repeatedly routes (or attempts to route) the circuit with different numbers of tracks in each channel (*channel capacities)*. VPR performs a binary search on the channel capacities, increasing them after a failed routing and reducing them after a successful one, until it finds the minimum number of tracks required for the circuit to route successfully on a given global routing architecture. While the absolute number of tracks per channel is adjusted upwards or downwards after each attempted routing, the *relative* numbers of tracks in the various channels across the FPGA are always kept at the values specified by the FPGA architecture. For example, VPR's first attempt at routing a circuit in an architecture with a two-to-one directional bias might assume horizontal channel capacities of twelve tracks and vertical channel capacities of six tracks. If this routing was successful, VPR would then attempt to route the circuit in an FPGA with horizontal channel capacities of six tracks and vertical channel capacities of three tracks. After a small number of such attempted routings one can determine the minimum number of tracks required to successfully route this circuit in this architecture. Thus the fixed variable in these experiments is the relative channel capacities, and the free variable is the absolute number of tracks required to route the circuit successfully.

The benchmark circuits used in this study consist of 14 combinational and 12 sequential MCNC benchmark circuits [8], which vary in size from 222 to 1878 of our logic blocks.

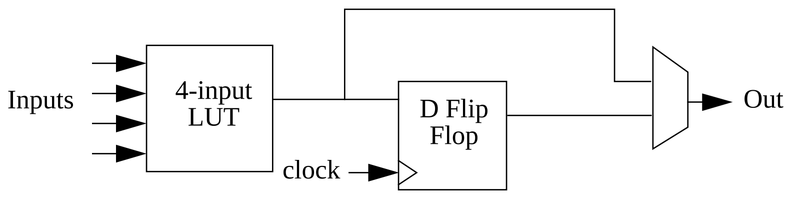


Figure 3: Logic Block Structure.

2.2 Area Model

Our goal is to measure the area-efficiency of different global routing architectures without reference to the detailed routing (segmentation, switch block, etc.) architecture. At this level, it is the amount of "global wiring" that changes as we vary the architecture. A simple track count will not accurately represent the wiring area of rectangular FPGAs, as the tracks in one direction are longer than those in the other. Accordingly, we define a *track segment* to be a prefabricated wire that spans one logic block; a channel of width W tracks that spans L logic blocks contains W x L track segments. The total number of track segments an FPGA must contain to globally route a circuit is a representative metric of the "global wiring" area. In order to average the results from circuits of differing sizes we use the average number of track segments per tile (i.e. per logic block) as our area measure. For example, in a square NxN uniform FPGA with W tracks in each channel, the total number of track segments is $2WN^2$, and the number of tracks per tile is 2W. Note that the routing area is given by the total number of track segments the FPGA contains, and not the number of track segments which are actually used by a circuit.

2.3 Significant FPGA Architectural Details

Several architectural parameters other than the global routing architecture must be specified in order to define an FPGA. We set these parameters to be as close to those of commercial FPGAs as possible.

First, the size of the FPGA array used for a given circuit (i.e. the number of logic blocks) is set to be the *smallest* FPGA with the desired aspect ratio (number of columns / number of rows) with sufficient logic blocks to accommodate the circuit. This situation, in which there is minimal "spare room" in the FPGA, presents the greatest challenge to routing completion, and is normally the case that manufacturers wish to optimize, since users want to buy the smallest FPGA with enough logic to contain their circuit.

In this study the number of I/O pads that can fit into the height or width of a logic block is set to two. This number is commensurate with the relative sizes of I/O pads and 4-LUTs in current FPGAs [2, 3, 5] and ensures that none of the 26 benchmarks is pad-limited.

Finally, we do not route the clock net in sequential circuits, since this net is normally distributed through a special clocking network in commercial FPGAs.

3 Tuned Placement and Routing Algorithms

In FPGA architecture explorations of this kind [1] one must ensure that the CAD tools used are responsive to the architectural parameters being varied. An architectural feature which appears useful but which CAD tools cannot exploit is of very limited utility. Similarly, a primitive CAD tool which does not make use of an FPGA feature as aggressively as possible may lead to a false conclusion about the usefulness of this feature. To ensure a fair comparison between different global routing architectures, we created a new placement and routing tool which understands how the routing resources available vary across the FPGA and tries to make maximal use of the widest channels in both the placement and routing steps. As this CAD tool is capable of mapping to a wide variety of FPGA architectures, we named it VPR, short for Versatile Place and Route.

3.1 Global Routing Resource-Aware Placement

We employ the simulated annealing algorithm [11] for placement. The annealing schedule is based on feedback control of the accepted move rate, which was found to be crucial to obtaining excellent placements in [12, 13]. The key to a routing resource aware placement tool is ensuring that the cost function correctly models the relative difficulty of routing connections in regions with different channel widths. After significant experimentation with many alternatives, we have developed a *linear congestion* cost function which provides the best results in reasonable computation time. Its functional form is

$$Cost_{linear} = \sum_{n=1}^{M} q(n) \times \left[\frac{bb_x(n)}{C_{av,x}(n)^{\alpha}} + \frac{bb_y(n)}{C_{av,y}(n)^{\alpha}} \right]. \tag{1}$$

The summation in (1) is over the M nets in the circuit. For each net, bb_x and bb_y denote the horizontal and vertical spans of its bounding box, respectively. The q(n) factor compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals, as suggested in [14]. Its value depends on the number of terminals of net n; q is 1 for nets with 3 or fewer terminals, and slowly increases to 2.79 for nets with 50 terminals. $C_{av,x}(n)$ and $C_{av,y}(n)$ are the average channel track capacities in the x and y directions, respectively, over the bounding box of net n:

$$C_{av,\mathrm{x}}(n) = \sum_{j = bb_{ymin}(n)}^{bb_{ymax}(n)} \frac{Capacity_x(j)}{bb_{ymax} - bb_{ymin} + 1}, \tag{2}$$

$$C_{av,\mathrm{y}}(n) = \sum_{i = bb_{xmin}(n)}^{bb_{xmax}(n)} \frac{Capacity_y(i)}{bb_{xmax} - bb_{xmin} + 1}. \tag{3}$$

This cost function penalizes placements which require more routing in areas of the FPGA that have narrower channels. The exponent, α, in the cost function allows the relative cost of using narrow and wide channels to be adjusted. When α is zero the linear congestion cost function reverts to the standard bounding box cost function. The larger the value of α, the more wiring in narrow channels is penalized relative to wiring in wider channels; we have experimentally found that setting α to 1 results in the highest quality placements.

Since C_{av} depends only on the channel capacities, which do not change during a placement, and on the maximum and minimum coordinates of the bounding box, we can precompute all possible $C_{av,x}$ and $C_{av,y}$ values and store them in two arrays indexed by the minimum and maximum of the appropriate bounding box coordinate. In fact, we precompute the inverse of $C_{av,x}^{\alpha}$ and $C_{av,y}^{\alpha}$ for all possible bounding boxes for further efficiency. Consequently, the time required to recompute this cost function is virtually the same as that of the traditional bounding box cost function.

In an FPGA where all channels have the same capacity, C_{av} is also a constant and hence the linear congestion cost function reduces to a bounding box cost function. In non-uniform and directionally-biased FPGAs, however, this cost function results in higher quality placements than a bounding box cost function. The exact amount of routability improvement depends on the precise global routing architecture used; as one would expect, those in which there is a large difference between the widths of channels in different regions show the largest improvement. For the architectures we study in this paper the linear congestion cost function typically produces placements which require 5 to 10% fewer tracks to route than placements produced with a bounding box cost function.

We found one cost function which was capable of producing higher-quality placements than the linear congestion cost function, at the cost of greatly increased CPU time. This cost function is based on the work in [14], and we call it a *non-linear congestion* cost function. This cost function divides the FPGA into an array of N x N regions and attempts to model the routing resource demand and supply in each of these regions. When a placement causes the routing resource demand to exceed the supply in some regions, the placement is heavily penalized. The exact functional form we use is

$$C_{nonlinear} = \sum_{i=1}^{N}\sum_{j=1}^{N} Max\left(\frac{D_{x,ij}}{\sigma \times S_{x,ij}}, \left(\frac{D_{x,ij}}{\sigma \times S_{x,ij}}\right)^2\right) + \sum_{i=1}^{N}\sum_{j=1}^{N} Max\left(\frac{D_{y,ij}}{\sigma \times S_{y,ij}}, \left(\frac{D_{y,ij}}{\sigma \times S_{y,ij}}\right)^2\right) \quad (4)$$

where $D_{x,ij}$ and $D_{y,ij}$ are the expected demand for routing resources in region (i,j) in the x and y directions, respectively, $S_{x,ij}$ and $S_{y,ij}$ are the available supply of routing resources in region (i,j) in the x and y directions, respectively, and σ is an optimization parameter between 0 and 1 that controls what fraction of a region's routing resources a placement can use before it is heavily penalized. We have found that the best value of σ is 0.6 -- smaller values do not reduce the solution quality much, but one should not use values much larger than 0.6 The routing supply is in units of tracks and is precomputed for each region before the annealing starts via

$$S_{x,ij} = \sum_{r = ymin_{ij}}^{ymax_{ij}} Capacity_x(r), \text{ and‘} \quad (5)$$

$$S_{y,ij} = \sum_{r = xmin_{ij}}^{xmax_{ij}} Capacity_y(r). \quad (6)$$

$Capacity_x(r)$ and $Capacity_y(r)$ are the capacities of the r^{th} channel in the x and y directions, respectively. The routing resource demand also has units of tracks. The total routing resource demand is the summation of the routing resource demands of all nets, where the resources used by one net in region (i,j) are given by

$$\Delta D_{x,ij} = q \times \frac{O_x \times O_y}{bb_y \times R_x} \quad (7)$$

$$\Delta D_{y,ij} = q \times \frac{O_x \times O_y}{bb_x \times R_y} \quad (8)$$

In the equations above, the bb factors refer to the span of the net bounding box, the R factors refer to the dimensions of each of the regions, and the O factors refer to the overlap in each dimension between the bounding box and the region in which the routing demand is being updated. Figure 4 summarizes the various geometric factors.

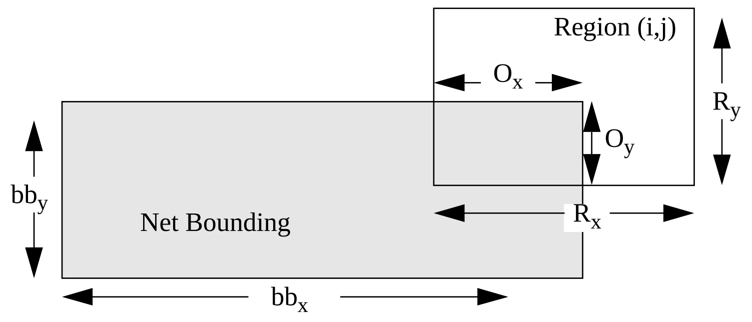


Figure 4: Routing Resource Demand in Region (i,j).

We found that this non-linear congestion cost function, when computed on a 4 x 4 grid (16 regions), generally produces placements which require 2 to 4% fewer tracks to route than those produced by the linear congestion cost function. However, keeping track of the routing resource demand in the various chip regions is computationally expensive, and placement with this cost function requires five times greater CPU time than the linear congestion function. Dividing the FPGA into smaller subregions to make localized congestion more visible did not work well; a non-linear congestion cost function computed on a 16 x 16 grid (256 regions) performs only marginally better than a cost function computed on a 4 x 4 grid, yet consumes sixteen times the CPU time.

We also investigated the performance of a second variant of the nonlinear congestion cost function. The cost is computed as before except that (4) is replaced by

$$\begin{aligned} C_{nonlinear} = {} & \sum_{i=1}^{N}\sum_{j=1}^{N} D_{x,ij} + [Max(D_{x,ij} - \sigma S_{x,ij}, 0)]^2 \\ & + \sum_{i=1}^{N}\sum_{j=1}^{N} D_{y,ij} + [Max(D_{y,ij} - \sigma S_{y,ij}, 0)]^2 \end{aligned} \tag{9}$$

This cost function is equivalent to adding a bounding-box cost to a quadratic penalty term for excessive congestion, with the two terms scaled so that both are significant, regardless of problem size or number of subregions. This cost function gave results that were essentially the same as thsoe of (4).

We considered the reductions in track count achieved by the non-linear congestion cost function too small to warrant the additional CPU time, so the results presented in this study all use the linear congestion cost function. Nonetheless, we did rerun a few of our experiments with the non-linear congestion cost function and found that its use did not change any of the conclusions discussed below.

3.2 Congestion-Driven Global Routing

It is crucial for the global router to be capable of leveraging the differences in the capacities of the various FPGA channels. The global router developed for this study employs a variant of the PathFinder negotiated congestion algorithm [15]. This algorithm consists of routing each net with a maze router [16], then ripping up and rerouting each net in sequence several times. In each of these subsequent routing iterations, the cost of using a node (which is *either* a channel segment or a logic block input pin) is modified, based on both the current and past (in previous iterations) competition for that node. A channel segment is the length of channel that spans one logic block. In an FPGA composed of an N x N array of logic blocks each channel contains N segments. We define the cost of a routing node somewhat differently than [15]; the cost of using routing node *n* is

$$c_n = (1 + h_n \cdot h_{fac}) \times (1 + p_n \cdot p_{fac}) + b_{n,\, n-1}. \tag{10}$$

The p_n term is a measure of the present congestion at this node. It is updated *every time any net* is ripped-up and rerouted. The value of p_n is equal to the overuse of this node that would occur if one more route were to use it, since the decision we are making during routing is whether another net should go through this node or not.

$$p_n = max((demand_n + 1 - capacity_n), 0) \tag{11}$$

For example, consider a channel with a capacity of six tracks and a segment of this channel in which five tracks are currently used. The p_n value of this channel segment is zero, since routing one more net through this channel will not cause any congestion. If, however, all six tracks in this channel were currently used, its p_n value would be one, since routing another net through it would result in an overuse of one. The h_n term accounts for the historical, or past, congestion at this node. It is updated *only after an entire routing iteration* is completed; i.e. after every net in the circuit has been ripped up and rerouted. Initially h_n is set to 0; at the end of each routing iteration h_n is increased by the amount by which demand for this node outstrips its capacity. That is,

$$h_n^{i+1} = \begin{cases} 0\,, & i = 0 \\ h_n^i + max(demand_n^{\ i} - capacity_n^{\ i}, 0), & i \geq 1 \end{cases} \tag{12}$$

where the superscripts refer to the routing iteration number.

The $b_{n,n-1}$ term penalizes bends, since global routes with many bends in them present a more difficult detailed routing problem in FPGAs with segmented routing, and will generally lead to detailed routes that are both slower and require more tracks. The value of $b_{n,n-1}$ is one if making the connection from node n-1 to node n implies a bend (i.e. node n-1 is a horizontal channel segment and node n is a vertical channel segment or vice versa), and is zero otherwise. Including this bend cost in the total cost of using a node produces routes with very few unnecessary bends with little increase in track count.

The key idea of Pathfinder is that the p_{fac} term is 0 for the first routing iteration, and is gradually increased in successive iterations. Hence, each net is initially routed by the shortest path found. In successive iterations, the p_{fac} term is gradually made larger so that congestion becomes more expensive and those nets which have alternate routes move out of the congested areas. The history term, h_{fac}, allows information from previous routing iterations to affect the current routing, further improving the router's ability to find and avoid congestion. By treating both channel segments and input pins as routing nodes, this algorithm makes use of the functional equivalence of LUT input pins in a very natural way. Initially, each connection connects to the logic block input pin which leads to the shortest route. As the cost of congestion increases, nets are gradually forced to ensure that they are each using an input pin that no other net is using.

In our implementation each channel can have a different capacity. Since the cost of a channel segment is based on the amount by which routing demand exceeds its capacity, this router will automatically act to relieve pressure on narrow channels by rerouting nets through wider channels whenever necessary.

Considerable effort was spent tuning the *routing schedule* (the values of p_{fac} and h_{fac} over the course of the iterations) in order to achieve the best results. The best routing schedule we found set p_{fac} to 0 for the first iteration, 0.5 for the second iteration, and 1.5 times the previous p_{fac} value for all subsequent iterations. The value of h_{fac} was set to 0.2 for all iterations; the fact that h_n can only increase from iteration to iteration provides enough increase in the historical congestion penalty by itself. This routing schedule increases the cost of congestion slowly enough that the net ordering is not very important -- nets with the most alternate routes move out of congested areas first. We found that increasing the cost of congestion more slowly than this reduced the number of tracks required only by 1 - 2% while increasing the CPU time by a factor of 2 to 3. Setting h_n to 0 so that the router has no information about past congestion increased the number of tracks required by 15%.

We also experimented with two different speed optimizations to the PathFinder algorithm. First, we tried setting the initial cost for routing segments (i.e. the cost for routing iteration 1) according to the expected demand for each segment as predicted from the final placement by equations (7) and (8) with the number of regions set equal to the number of logic blocks. The idea was that the initial routing would therefore try to avoid regions which were expected to have high congestion, and fewer routing iterations would be needed to find a feasible routing. However, the results thus obtained are significantly worse than those obtained when the cost of routing resources during the first iteration is set by (10) with p_{fac} equal to 0. Setting p_{fac} to 0 for the first routing iteration lets each net initially take a short path because congestion is not penalized; forcing some of the nets out of the highly congested areas is deferred to later routing iterations. This initial routing in which congestion is not penalized provides a considerably better outline of the congestion inherent in the placement than equations (7) and (8), and hence leads to higher-quality final routings.

The second speed optimization was more successful. We constrain the router to route each net without using any routing segments which are more than β logic blocks outside the net bounding box. Setting β to 0 forces all nets to be routed within their bounding box; this results in a speedup of 24% at the cost of a 3% increase in track count versus allowing all routes (i.e. the β = chip size case). Setting β to 3 speeds up the router by 12% while increasing the tracks required by less than 1% over the all routes allowed case. We have found that the congestion-avoidance features of our router result in very little increase in the average length of connections, so the fact that the vast majority of routes can be completed within the bounding box of their terminals is not surprising. The speedups achieved by limiting routings to the net bounding box are fairly modest because high fanout nets take the greatest amount of time to route with our algorithm, and their bounding boxes span essentially the whole chip.

3.3 Validation of Placement and Routing Tool

In order to assess the quality of our new placement and routing tool, we compared the channel density it achieved on thirteen benchmark circuits to that achieved by the existing University of Toronto placement and routing suite. This placement and routing suite consists of the ALTOR [17] placement program, which uses the MinCut placement algorithm [18], and the LocusRoute global routing program [19].

In order to allow direct comparison with the final routings of ALTOR/LocusRoute, the experimental methodology described in Section 2 and used throughout the remainder of this work was altered wherever necessary so that it exactly corresponded to the flow used by ALTOR/LocusRoute. Specifically, the benchmark circuits were all combinational, and were technology-mapped to 4-LUTs by the Chortle [20] program. The size of the FPGA and the number of I/O pads per column were always set to the values used by ALTOR/LocusRoute. Finally, the FPGA has channels which are all of the same width (i.e. uniform and unbiased), as this is the only type of FPGA to which ALTOR and LocusRoute can map.

Table 1 compares the performance of VPR and the ALTOR/LocusRoute tool suite. The channel density for each circuit is the minimum number of tracks each channel must contain for the circuit to successfully route. There are two columns for VPR. In the “reroute only” column VPR was allowed only to reroute the circuit; the placement was performed by ALTOR. In the “replace and reroute” column both the placement and routing were performed by VPR. When VPR is only allowed to reroute a circuit it reduces the channel density by 37% relative to LocusRoute; when the circuit is both placed and routed by VPR the channel density is reduced by 57% compared to that achieved by ALTOR and LocusRoute. Since ALTOR and LocusRoute have been widely used for FPGA research both inside and outside the University of Toronto, we consider the large reduction in channel density achieved by VPR to be a convincing demonstration of its quality.

Table 1: Comparison of VPR to Altor/LocusRoute

Circuit	Channel Density		
	ALTOR + LocusRoute	VPR: Reroute Only	VPR: Replace and Reroute
C499	10	7	5
C880	12	8	5
C1355	11	7	5
alu4	13	8	6
apex7	12	7	4
term1	9	6	4
example2	16	8	5
too_large	11	8	6
k2fix	15	10	7
vda	13	9	6
9symml	10	5	5
alu2	10	6	5
z03D4	14	8	5
Average	12.0 (100%)	7.5 (63%)	5.2 (43%)

4 Experimental Results for FPGAs with Directionally-Biased Routing Resources

The experimental framework and tools described above were employed to answer the questions posed in the introduction to this paper: first, is there an area-efficiency advantage to using a directionally-biased architecture? A directionally-biased FPGA is one in which the numbers of tracks available for routing in the horizontal and vertical directions are not equal. In essence, we are investigating if there is an exploitable directional bias in the basic nature of circuits. Figure 5 (b) shows an example FPGA with a 2:1 directional bias; its horizontal channels contain twice as many routing tracks as its vertical channels. We characterize directionally-biased FPGAs by the ratio of the width of a horizontal channel to the width of a vertical channel, denoted as R_h.

We need to define an additional architectural feature which markedly affects our conclu-

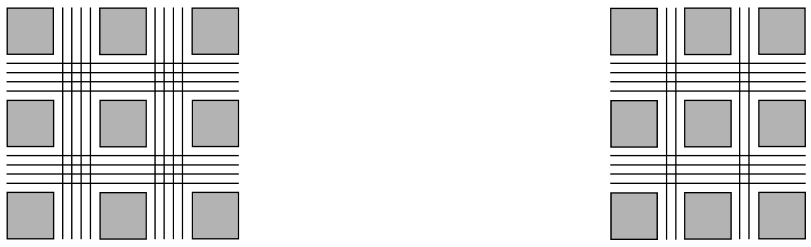

(a) No Directional Bias (Rh = 1)
(b) Directional Bias (Rh = 2)

Figure 5: FPGAs With and Without Directional Bias.

sions: the positioning of the pins on the logic block. The two main cases of interest are illustrated in Figure 6. In Figure 6(a), the logic block input and output pins are distributed evenly around the entire perimeter of each logic block. We call this the *full-perimeter* pin positioning, and it is similar to the pin positioning used in the FPGAs of Xilinx and AT & T [2, 3]. The second alternative, which we call the *top/bottom* pin positioning, restricts the logic block input pin locations to lie only on the top and bottom of the logic block. The top/bottom pin positioning is illustrated in Figure 6(b) and it is similar to the pin positioning used in Actel FPGAs [4]. In all the results we show in this paper, each logic block pin appears (physically) on only one side of a logic block. As discussed in Section ??, we have found that for the channel connectivity values (F_c [1]) found in today's commercial FPGAs this leads to the most area-efficient FPGAs.

Finally, we have also found that the ratio of the number of columns to the number of rows in an FPGA, which we call the aspect ratio, significantly affect area efficiency. Since most FPGAs have the same number of rows and columns, we first present the results for square (aspect ratio 1) FPGAs, before discussing the more general case of rectangular FPGAs in Section 4.2.

4.1 Results for Square FPGAs

The 26 large MCNC circuits were passed through the experimental flow of Figure 2 for values of R_h varying from 1 to 4. As discussed in Section 2, the result for each circuit is the number

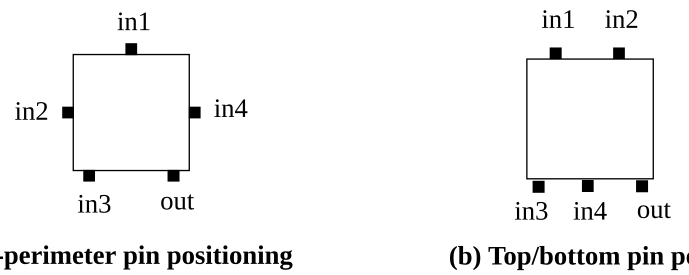

in1
in2
in4
in3
out
(a) Full-perimeter pin positioning
in1
in2
in3
in4
out
(b) Top/bottom pin positioning

Figure 6: Logic Block Pin Position Alternatives.

of track segments *per tile*[3] needed to successfully global route the circuit in an FPGA with the specified value of R_h. Figure 7 is a plot of area-efficiency versus the degree of routing direction bias, R_h, for both types of pin positioning. The vertical axis is the average number of tracks per tile required to successfully route the 26 benchmark circuits.

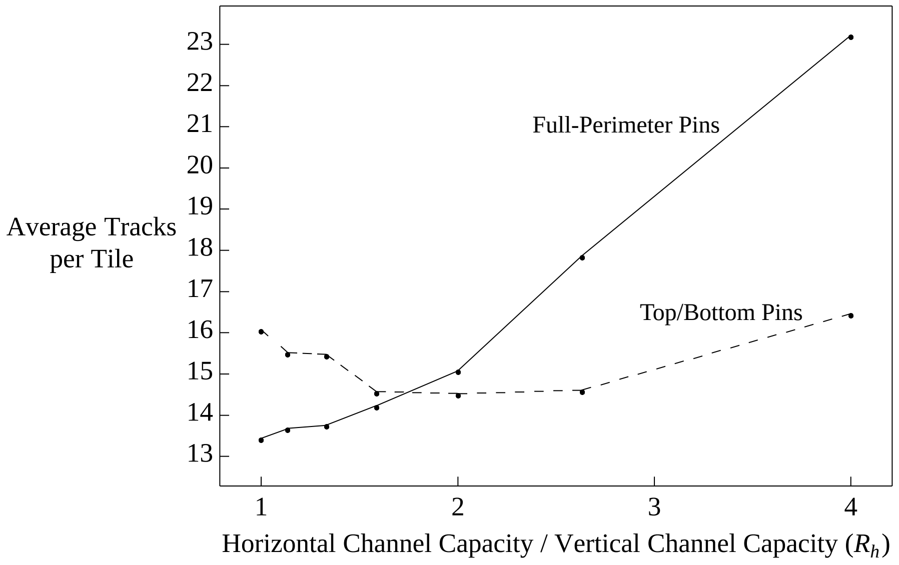


Figure 7: Area-Efficiency vs. Directional Bias for Square FPGAs.

The data shows that for the full-perimeter logic pin positioning, the best architecture is one without any directional bias. However, when the pins are restricted to the top and bottom of the logic block, the most efficient architecture has horizontal channels which are roughly twice as thick as the vertical channels. An important conclusion is that the best full-perimeter architecture is better than the best top/bottom pin architecture. The latter requires about 8% more tracks per tile on average.

The full-perimeter architecture is more area-efficient because there is a greater chance that the block input pins are closer to their desired connections when they are in the full-perimeter configuration than when they are in the top/bottom configuration. For example, consider the two routings of a multi-terminal net shown in Figure 8. The top/bottom pin configuration needs six track segments to route this net, whereas the full-perimeter configuration requires only five. By making use of the functional equivalence of LUT input pins during routing, the router is often able to connect to a logic block pin adjoining a track segment it needs to use for other connec-

3. Track segments are counted whether or not they are actually used, so this is a true representation of the area that must be devoted to routing in the layout.

tions, essentially making the connection to this logic block for free. Since the top/bottom pin configuration has input pins bordering on only the horizontal channels, such “free” connections into logic blocks are less frequent, reducing area-efficiency.

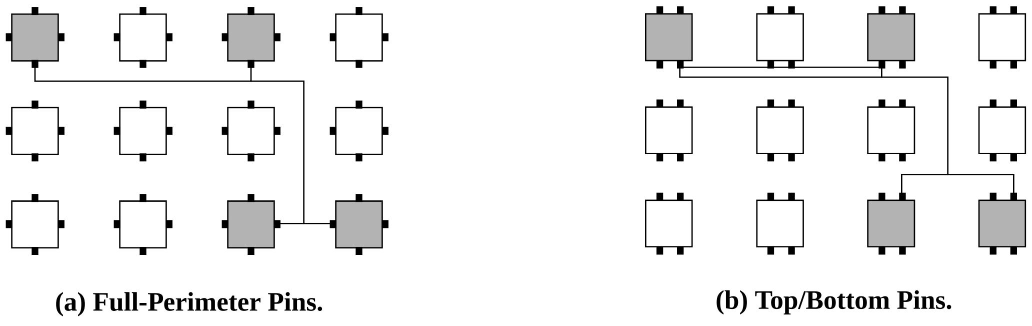


Figure 8: Example Routing of a Multi-Terminal Net Using Different Pin Posi-

The full-perimeter pins configuration achieves highest area-efficiency when there is no directional bias to the routing because this makes the difficulty of routing to each of a logic block’s nearest neighbors roughly equal. Consequently, the placement software can use all the nearby logic block locations equally to cluster the fanout of a net around its driver. Essentially, this allows one to cluster tightly coupled portions of logic in the smallest possible area. The top/bottom pins configuration, on the other hand, prefers a 2:1 directional bias because every connection to a logic block pin must come from a horizontal channel. This extra pressure on the horizontal routing resources is significant, since the typical distance routed between pins is only about 3 track segments.

4.2 Rectangular FPGAs

In order to increase the IO-to-logic ratio, FPGA manufacturers may want to build rectangular FPGAs, as this increases the die perimeter and hence the number of pads. In this case the channels in one direction are longer and have more blocks connected to them than the orthogonal channel, so the best amount of directional bias may change. We refer to the ratio of the number of columns in an FPGA to the number of rows as its aspect ratio. Figure 9 depicts an FPGA with an aspect ratio of two.

Figure 10 is a plot of the required tracks per tile versus R_h for various chip aspect ratios for an FPGA with the full-perimeter logic block pin positioning.

There are two features of interest in Figure 10. First, notice that the minimum of the aspect

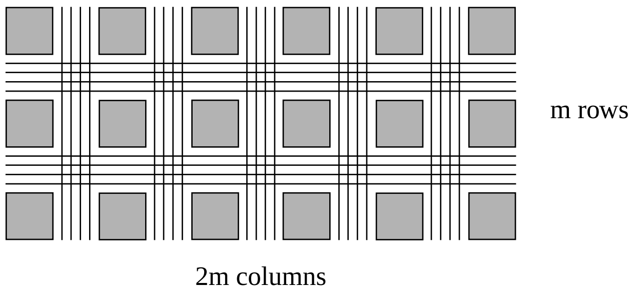


Figure 9: An FPGA with an Aspect Ratio of 2.

ratio = 1 curve is the lowest of the three, indicating that a square FPGA is most area-efficient. Secondly, the value of R_h at which the minimum area occurs *increases* as the aspect ratio increases. As the aspect ratio increases, the horizontal channels become longer than the vertical channels and this results in greater demand for horizontal track segments. The best value of R_h increases from 1 for a square FPGA to 1.33 and 1.59 for aspect ratios of 2 and 3, respectively.

Figure 11 shows how the number of tracks per tile varies with the aspect ratio, again for the full-perimeter logic block pin positioning. The upper curve keeps R_h fixed at 1, which is the best value for a square FPGA. The routing resource requirements increase moderately with aspect ratio; an FPGA with an aspect ratio of 3 requires 18% more tracks per tile than a square FPGA when R_h is 1. The lower curve plots the tracks per tile required by the FPGA with the best value of R_h for each aspect ratio. Clearly, when we alter R_h as the aspect ratio increases to compensate for the greater demand for horizontal routing the increase in tracks per tile with aspect ratio is consid-

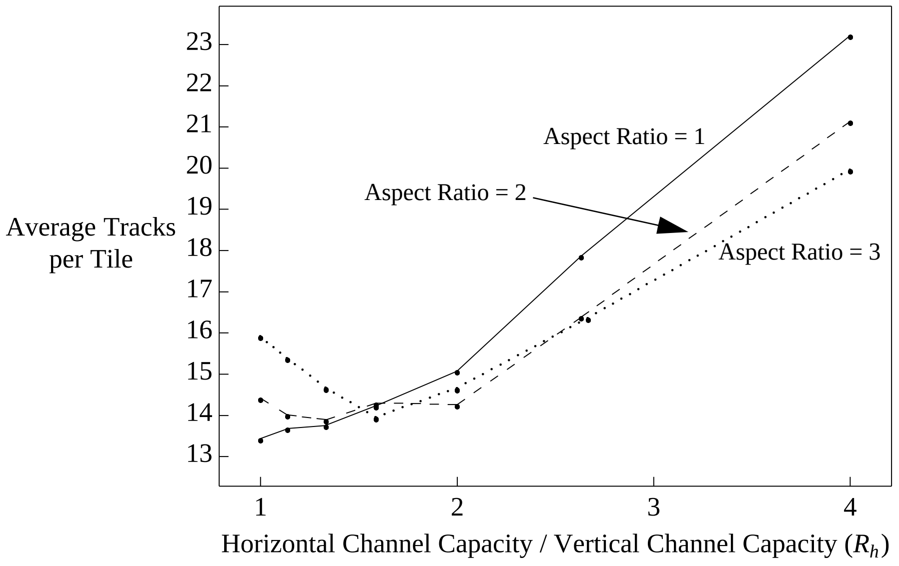


Figure 10: Area-Efficiency of Rectangular FPGAs with Full-Perimeter Pins.

erably less pronounced. In this case, an FPGA with an aspect ratio of 3 requires only 4% more track segments than a square FPGA. Thus we conclude that, as long as the horizontal and vertical channel widths are appropriately balanced, the chip aspect ratios can be increased with little impact on the core area, and so I/O counts can be increased.

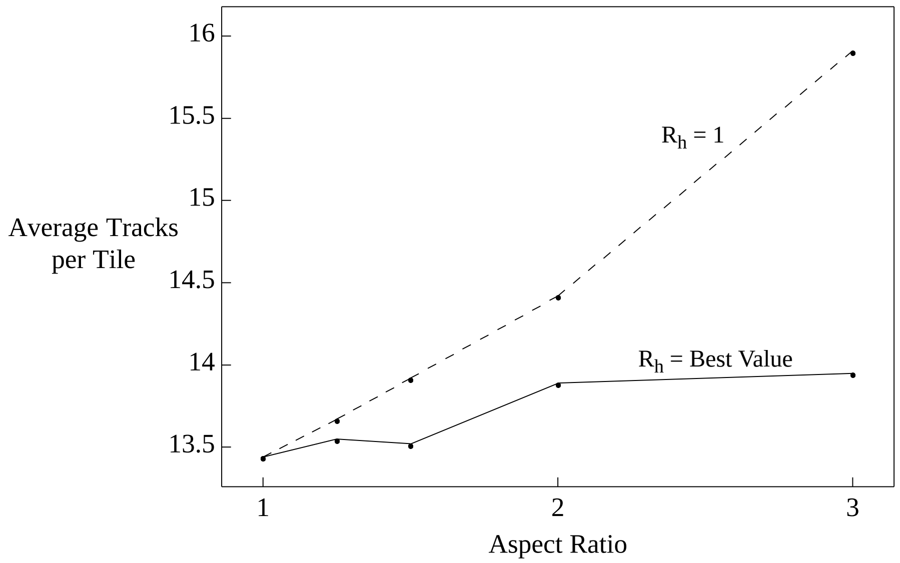


Figure 11: Routing Resource Requirements vs. FPGA Aspect Ratio.

The variation of core routing area with aspect ratio is similar for FPGAs that use the top/bottom logic block pin positioning. In this case an FPGA with an aspect ratio of 3 requires only 5% more tracks per tile than a square FPGA. For FPGAs of this type, however, it is not necessary to increase R_h as the aspect ratio increases; doing so provides only a marginal area-efficiency improvement. This is because the best square FPGA with top/bottom pins has horizontal channels which are twice as wide as vertical channels; the thicker horizontal channels are better able to cope with the increased pressure for horizontal tracks as aspect ratio increases.

4.3 Number of Physical Locations for Each Logic Block Pin

Since the demand for routing tracks is so dependent on the location of the logic block pins, we conducted a study to determine the number of logic block sides on which each pin should appear. We call an input or output to a logic block a *logical* pin; in the 4-LUT based logic blocks we are using there are 4 logical inputs and one logical output. Each logical pin has one or more associated *physical* pins; for example, if input 1 is accessible from both the left and right sides of the logic block it has two physical pins. Brown looked at this issue in [21], but he did not consider configurations where inputs and outputs had differing numbers of physical pins, and the global router used in that study did not make use of the fact that the inputs to a LUT are logically equivalent.

Let T_i and T_o be the number of physical pins for each logical input and output pin, respectively. We will determine the best values of T_i and T_o by estimating the relative number of switches per logic block to which various choices of their values lead. In order to estimate switch counts, we must assume a detailed routing architecture for the FPGA. We will assume that the number of tracks to which each physical pin connects, F_c, is equal to the number of tracks in a channel, W. As well, assume that the number of track segments to which each track segment can connect, F_s, is 3. These values are in line with those in popular FPGAs [2]. Brown [21] found that such an FPGA could be detail routed using only 7% more tracks, on average, than the global router required. Hence, the number of tracks required by the detailed router is proportional to the number required by the global router, and we can use the track count from the global router to make comparisons between the various T_i and T_o.

With these assumptions, we can write the following proportionality relation for the number of switches per logic block, N_s, in an FPGA with no directional bias in its routing

$$N_s \propto (4T_i + T_o)F_c + 2WF_s = (4T_i + T_o + 6)W \tag{13}$$

where W is the number of tracks per channel required by the global router, and the second relation has made use of the fact that $F_c = W$ and $F_s = 3$.

Table 2 shows how N_s varies with T_i and T_o; W in the table is the average over our 26 benchmark circuits. Our values for N_s in Table 2 are really a lower bound, since the W value is taken from the global router.

Table 2: Routing Switches per Logic Block for an Unbiased FPGA.

T_i	T_o	W	N_s
1	1	6.7	74
1	2	6.3	76
1	4	6.2	87
2	2	5.8	93
4	4	5.6	146

The most area-efficient FPGA has one physical pin for each logical pin; i.e. $T_i = T_o = 1$. Note that there may be benefits to having more than one physical pin per logical pin when F_c is less than W; for example, $T_i = T_o = 2$ and $F_c = W/2$ might be a good choice. However, this question is beyond the scope of this study.

In the case of an directionally-biased FPGA, we must generalize (13) to account for the fact that the horizontal and vertical channels have different widths, which we denote as W_h and W_v respectively. A potential switch block for a directionally-biased FPGA (with $R_h = 2$) is illustrated in Figure 12.

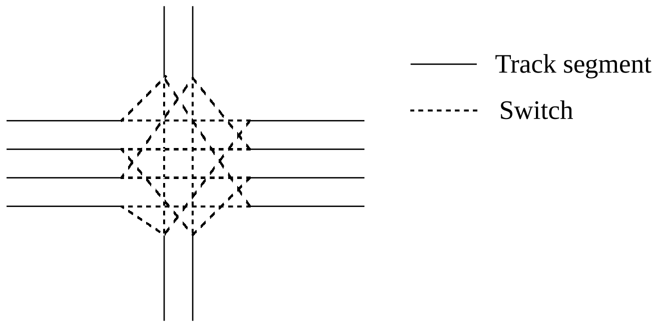


Figure 12: Switch Block for a Directionally-Biased FPGA with $F_s = 3$

The number of switches per logic block when this switch block is used varies as

$$N_s \propto (4T_{i,h} + T_{o,h})F_{c,h} + (4T_{i,v} + T_{o,v})F_{c,v} + W_h + (2F_s - 1)W_v \qquad (14)$$

where $T_{i,h}$ ($T_{i,v}$) refers to the number of logic block sides bordering on a horizontal (vertical) channel on which each input pin appears. $T_{o,h}$ and $T_{o,v}$ are similarly defined for the output pin. By using the fact that $W_v = W_h/R_h$, setting $F_{c,h}$ and $F_{c,v}$ to W_h and W_v respectively, and setting F_s to 3, we obtain

$$N_s \propto \left(4T_{i,h} + T_{o,h} + 1 + \frac{4T_{i,v} + T_{o,v} + 5}{R_h}\right)W_h \tag{15}$$

Table 3 summarizes how the number of switches per logic block varies with the various T parameters for the best directionally-biased FPGA found in Section 4.1, which has $R_h = 2$.

Table 3: Routing Switches per Logic Block for a Directionally-Biased FPGA with $R_h = 2$.

$T_{i,h}$	$T_{i,v}$	$T_{o,h}$	$T_{o,v}$	W_h	N_s
1	0	1	0	8.8	75
1	0	2	0	8.6	82
1	0	2	2	8.4	88
2	0	2	0	7.9	107
2	2	2	2	7.4	137

Again the best architecture has one physical pin for each logical pin. Clearly, using one physical pin for each logical pin in the studies of Sections 4.1 and 4.2 was the correct choice.

5 Experimental Results for FPGAs With Non-Uniform Routing

The second key issue we explore concerns the area-efficiency obtained when the channels in different regions of an FPGA have different capacities. We only investigate FPGAs which use the full-perimeter pin positioning, as the results of the previous section showed that this pin positioning is best.

We define a non-uniform routing architecture to be one in which the number of tracks per channel changes from channel to channel across an FPGA. For example, Figure 13 illustrates a non-uniform FPGA in which the channels at the chip center are wider than those near the periphery. If congested regions of a circuit can be localized and placed in the portions of the FPGA with the widest channels, a non-uniform FPGA could have better area efficiency than a uniform FPGA. We will investigate three types of non-uniform FPGAs in which we vary the center/edge channel capacity ratio, the width of only the center channel, and the I/O channel capacity, respectively.

5.1 Center/Edge Capacity Ratio

There is a widespread belief that most congestion occurs in the center of FPGAs, and hence

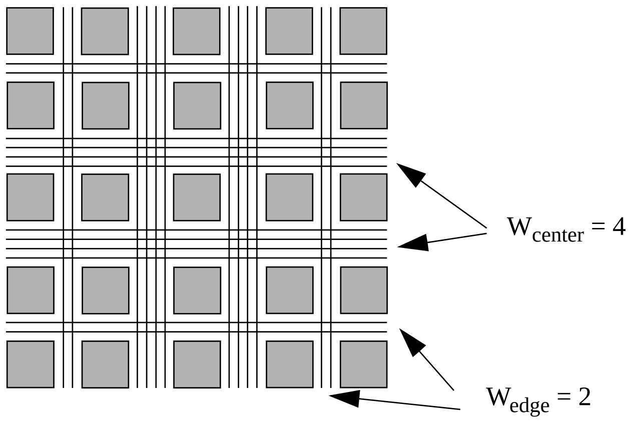


Figure 13: An FPGA with a Non-Uniform Routing Architecture (R_w = 2, R_c = 0.5)

having wider channels near the FPGA center and narrower channels near the edges is expected to improve area-efficiency. To keep the layout problem tractable, we restrict ourselves to FPGAs which use channels of only two different widths, such as the FPGA in Figure 13. We can describe global routing architectures of this form with two parameters. Let R_w be the ratio of the widths of the channels near the center of the FPGA to the widths of the channels near the FPGA edges, i.e. W_{center} / W_{edge}. Let R_c be the ratio of the number of channels with width W_{center} to the total number of channels. With this notation, the FPGA of Figure 13 is described as having R_w = 2 and R_c = 0.5.

Using the flow of Section 2, we again mapped 26 benchmark circuits into several architectures to determine their area-efficiency. We examined FPGAs with R_w equal to 0.75, 1.18, 1.33, and 2, and with R_c values varying from 0 to 1. The relative effectiveness of FPGAs with R_w = 1.33 and R_w = 2 is summarized in Figure 14. Note that the points at which R_c equals 0 or 1 correspond to a uniform FPGA.

The results show that the less uniform the channel widths, the worse the FPGA area-efficiency. The worst area-efficiency with R_w = 2 occurs when R_c is 0.5, meaning that half the FPGA channels are twice as wide as the other half. In fact, only two non-uniform FPGAs show even marginal area-efficiency improvements over a uniform case. Both these FPGAs are very close in architecture to a uniform FPGA. In one, the 10% of channels nearest the center are 33% wider than the other channels, while in the other the 90% of channels closest to the center are 33% wider than the channels nearest the edges. The reduction in tracks per tile over a uniform FPGA is less

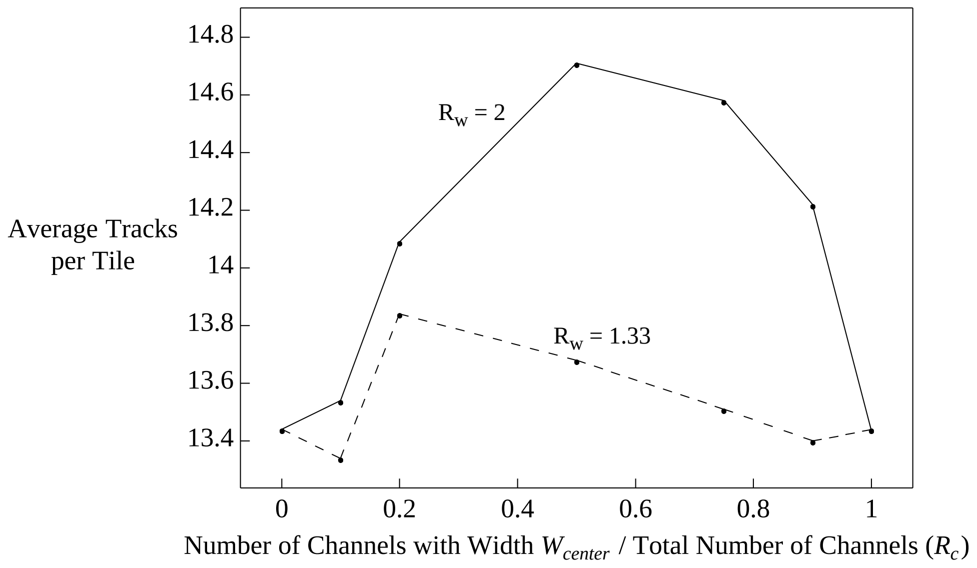


Figure 14: Routing Resources Required vs. Routing Resource Distribution.

than 1% for both of these FPGAs, so the improvement is not sufficient to justify the extra layout effort required.

These results are significant because there is a common belief amongst FPGA architects that there would be significant benefit to these kinds of non-uniform architectures. The fundamental reason that they do not show any benefit is that there is not much more congestion in the center of an FPGA than there is near its edges. In order to determine the "natural" routing demand distribution of circuits, we placed and routed the 26 benchmark circuits with all congestion avoidance features disabled, so that placement minimized wirelength and the router connected each net by the shortest path. Figure 15 plots the maximum and average number of tracks required by the horizontal channels as a function of the channel position within the FPGA, averaged over the 26 benchmark circuits. Notice that the demand for routing tracks is relatively constant over the middle 90% of the FPGA, and there is only a moderate decrease as one gets very close to the chip edges. Figure 16 plots the average and maximum tracks required per channel versus channel position for three representative benchmark circuits. There is some high-frequency variation from channel to channel, since the router is, in this case, not making any effort to route nets around congestion. Nevertheless, it is clear that these circuits closely mirror the behavior of the overall average of Figure 15.

Figure 17 provides another perspective on this issue. It shows the placement and global

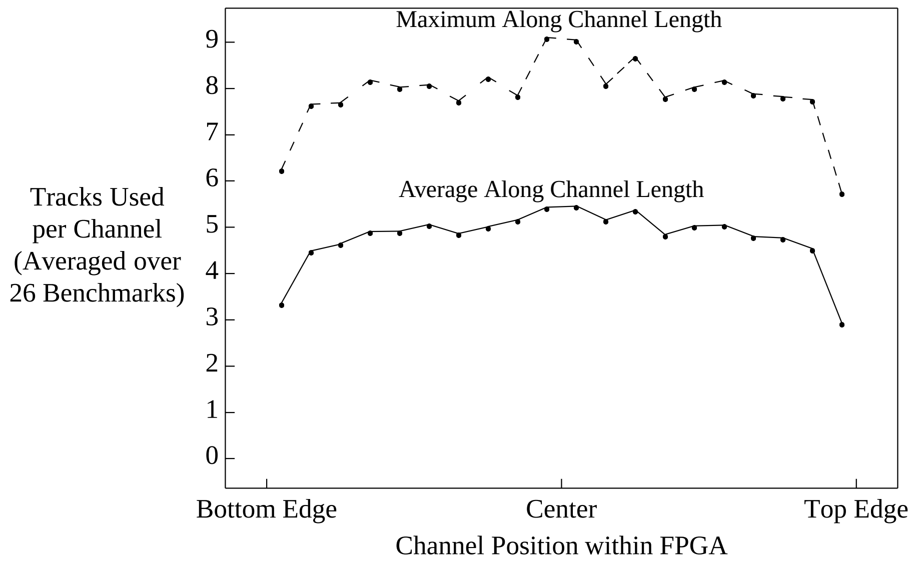
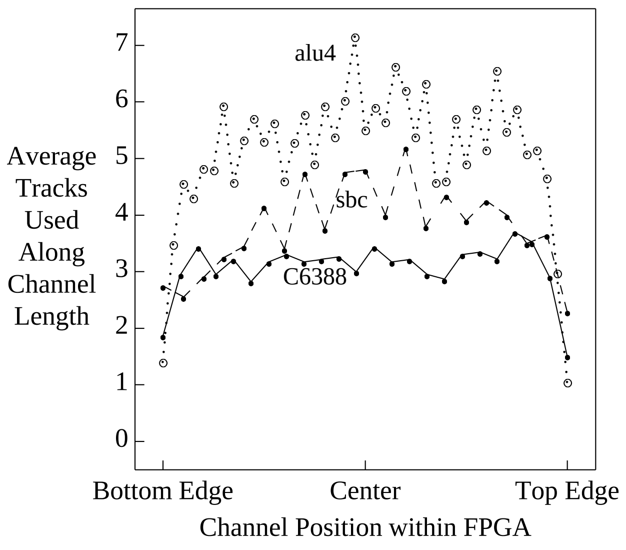
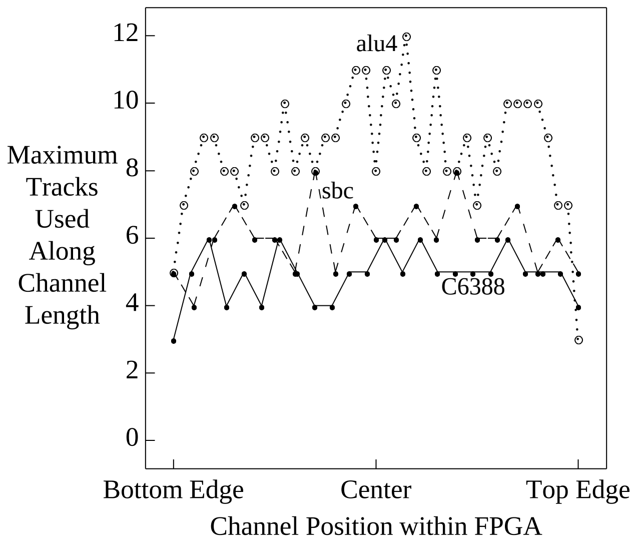


Figure 15: Average over Benchmarks of Horizontal Track Demand vs. Position.

routing (using the usual congestion-driven tools again) of a benchmark circuit on a uniform FPGA. Notice that while there is somewhat greater congestion in the middle of the FPGA than in areas very close to the pads, the trend is not very strong. As well, there are numerous local congestion "hotspots" where small regions have filled all the available channels, and some of these "hotspots" occur quite close to the FPGA edge. Consequently, in order for an FPGA with thicker channels near its center to use fewer routing resources, the placement software must move all of these hotspots into the FPGA center. As discussed in Section 3.1, we spent considerable time



(a) Average Tracks Needed Along Channel



(b) Maximum Tracks Needed Within Channel

Figure 16: Horizontal Channel Track Demand vs. Channel Position for Three Circuits.

investigating placement cost functions that modelled congestion well. The more advanced, and computationally expensive, cost functions, however, improved the performance of the uniform FPGA more than they did the non-uniform FPGA. It is more effective to have CAD tools attempt to spread out congestion as much as possible, rather than trying to localize it to a designated portion of a chip.

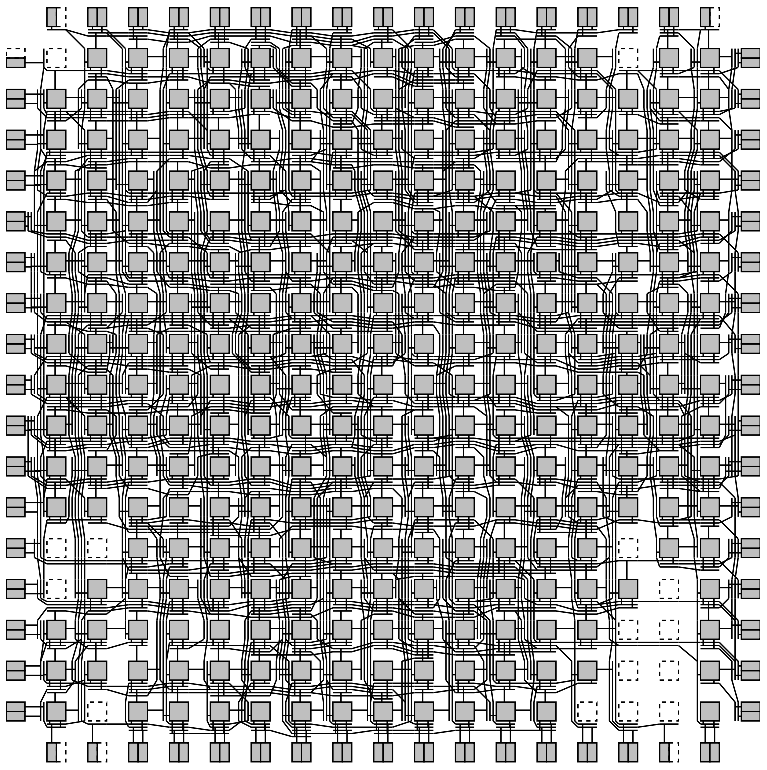


Figure 17: Global Routing of Benchmark Circuit e64.

5.2 Single Center Channel

One major FPGA vendor, in an effort to improve routability, has made one channel in the center of the FPGA in each direction extra wide (called “inter-quad routing” by AT & T) [6]. Figure 18 depicts an example FPGA of this type. We define R_m to be the ratio of the width of these center channels to the width of the other channels. Figure 19 is a plot of Tracks/Tile Required versus R_m for this type of FPGA.

The data shows that the most routable FPGA is one without an extra wide channels in the middle -- i.e. $R_m = 1$. There is a sharp dip in the number of tracks/tile required at $R_m = 2$, indicating an FPGA with routability almost as good as one with $R_m = 1$. This dip occurs at the first point

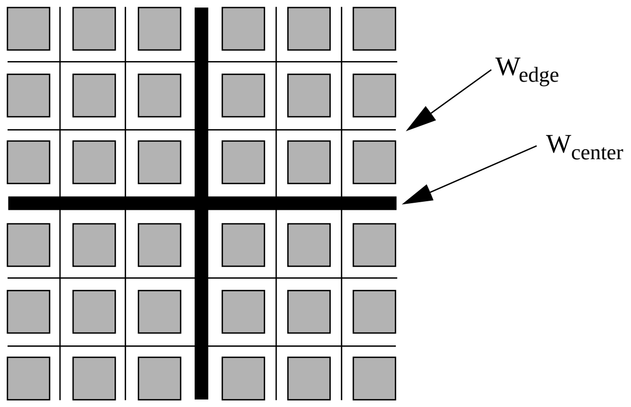


Figure 18: An FPGA with an Extra-Wide Center Channel.

at which the linear congestion cost function considers the cost of routing through narrow channels to connect two adjacent blocks to have the same cost as connecting two blocks separated by one intervening block through the extra wide channel. Consequently, the placer is able to make better use of the extra-wide channel at this point. Note that, as with the non-uniform FPGAs of Figure 14, the best results are obtained by spreading extra routing resources over the entire FPGA rather than by adding them to only one region.

5.3 I/O Channel

We refer to the channel that runs between the I/O pads and the logic array as the I/O channel; Figure 20 depicts its location. Many in the FPGA community believe that when a circuit's I/O locations are fixed by board-level constraints, there is considerable extra pressure on the I/O chan-

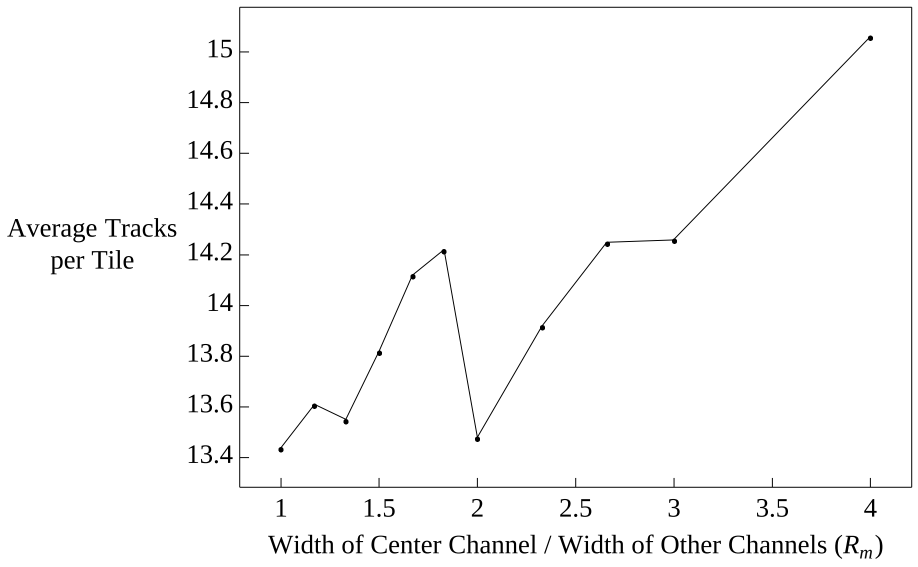


Figure 19: Effectiveness of an Extra-Wide Center Channel.

nel and it should therefore be quite wide. In fact, a major FPGA vendor has added routing resources to this I/O-channel, at least partially to ensure that fixed I/O pad placement does not impact routability and speed [7]. Therefore, we investigated the best width of the I/O channel both when the I/O locations of a circuit can be chosen by the placement software, and when they are locked in a specific configuration.

We investigated FPGAs in which all channels within the logic block array had the same width, W_{logic}, and the I/O channel had a different width, W_{IO}. We describe such FPGAs via a single parameter, R_{io}, which is defined to be W_{IO} / W_{logic}. Figure 21 is a plot of the average tracks/tile required for the 26 benchmarks circuits versus R_{io}. The solid line in Figure 21 shows the trend when the I/O locations are chosen by the placement tool, while the dashed line is found when the I/O pads are “fixed” in a random location, to model the effect of poor (from the FPGA’s point of view) board-level pin constraints.

There are several features of interest in Figure 21. First notice that fixing the I/O locations increases the number of routing tracks required by 12% on average. Architects must take this into account when designing FPGAs. Secondly, the curve where the I/O locations are chosen by the placement tool has its minimum value when R_{io} = 1, again showing that it is best to spread routing resources evenly across the chip. Fixing the I/O pins shifts the minimum in the tracks per tile curve slightly so that it now occurs when R_{io} = 1.25.

In order to determine how the “natural” demand for tracks is altered when the I/O locations of a circuit are fixed in a poor configuration, we repreated the congestion-oblivious placement and routing experiments described in Section 5.1 with the I/O locations fixed in a random configura-

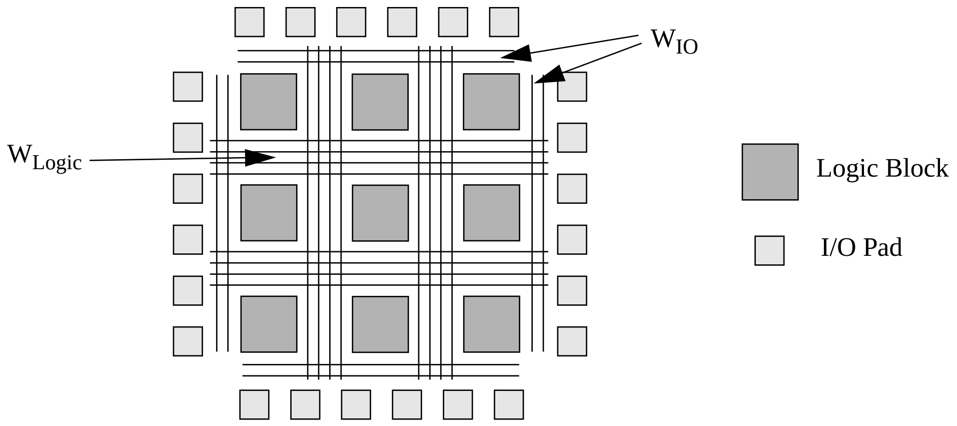


Figure 20: I/O Channel Location

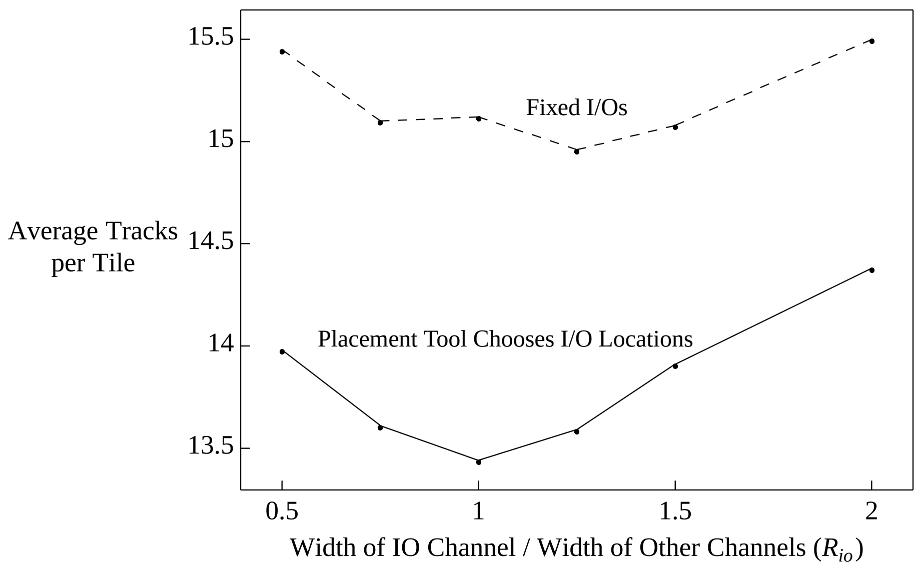


Figure 21: Effect of I/O channel width on routability.

tion. Figure 22 plots the maximum and average number of tracks required by the horizontal channels as a function of the channel position within the FPGA, averaged over the 26 benchmark circuits. Comparing with the corresponding curve obtained with movable I/Os (Figure 15), one sees that the curves have shifted up by approximately half a track, and that the drop off in track demand near the chip edges is significantly less pronounced. Figure 23 shows how the “natural” track demand of three typical circuits vary with channel position. By comparing with Figure 16, one sees that the curve for alu4 has changed little, while the sbc and C6388 curves have each shifted up by about a track and show significantly more demand for routing tracks near the chip

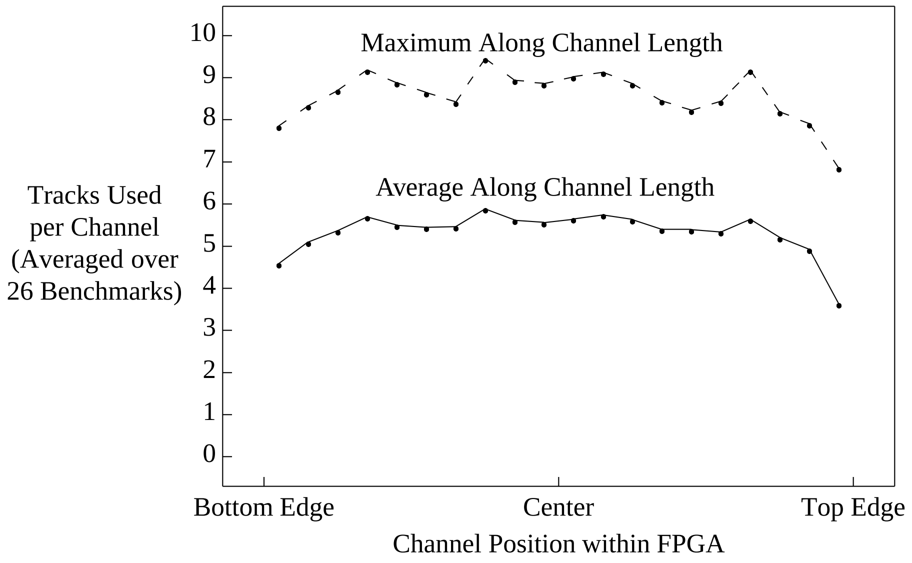
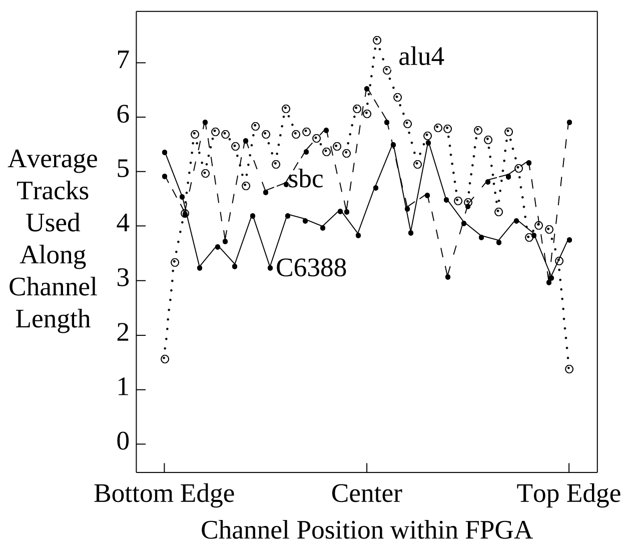
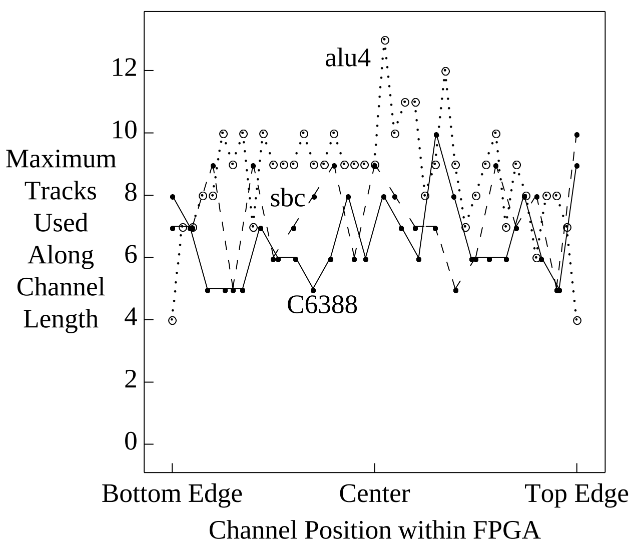


Figure 22: Horizontal Track Demand when I/O Locations are Fixed.



(a) Average Tracks Needed Along Channel



(b) Maximum Tracks Needed Within Channel

Figure 23: Horizontal Track Demand vs. Channel Position with Fixed I/Os.

edges than they did when the I/Os were movable. This is due to the different IO to logic ratios of these three circuits. Alu4 has very few I/Os; it uses only 7% of the I/O pads available in the FPGA to which it is mapped. C6388 and sbc, on the other hand, have considerably more I/O, and use 35% and 61% of the I/O locations available to them, respectively. As one would expect, then, fixing I/O locations has little effect on circuits with few I/Os. On the other hand, circuits with larger I/O requirements show an increase in routing track demand across the entire FPGA, with the greatest increase near the chip edges.

Figure 24 shows the routing (with all the router's congestion avoidance features enabled again) of the benchmark circuit e64 on a uniform FPGA when its I/O locations are fixed. Comparing with Figure 17 one sees that 8 tracks per channel are now required instead of 6, and that there has been some increase in the routing density near the chip edges relative to the routing density near the center. Overall the amount of congestion is fairly uniform across the entire chip.

In summary, while fixing the I/O pins leads to a significant increase in the number of tracks required to route a circuit, this increase is, for the most part, spread over the FPGA and not confined to the channels connecting to the IO pads. Consequently, one should not make very wide channels adjoining the pads in order to improve routability with pin constraints, although a small increase in the I/O channel capacity is a net benefit.

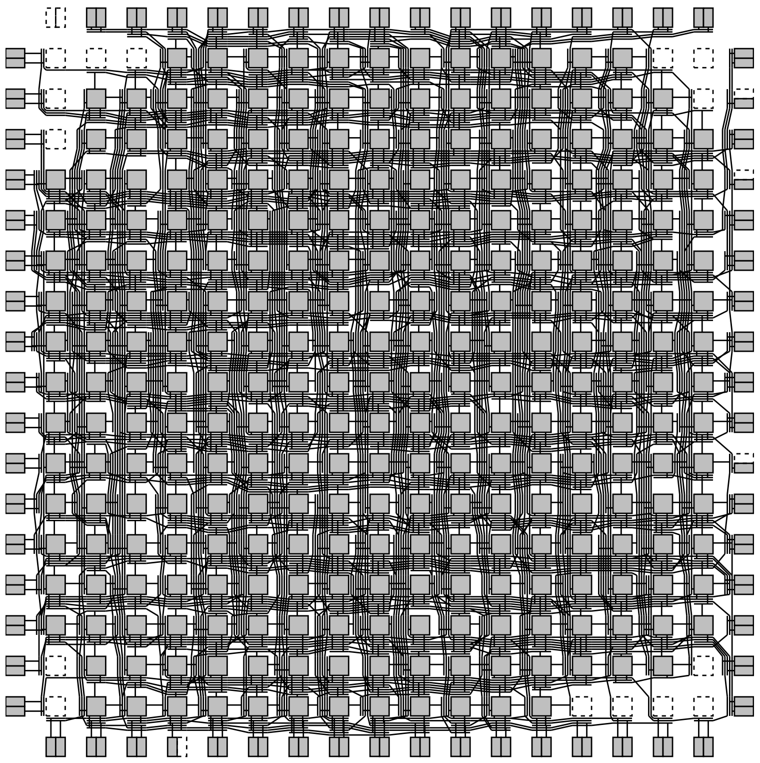


Figure 24: Global Routing of Benchmark Circuit e64 with Fixed I/Os.

6 Conclusions

The most interesting (and unexpected) conclusion of this work is that the most area-efficient global routing structure is one with completely uniform channel capacities, across the entire chip and in both horizontal and vertical directions. The basic reason is that most FPGA circuits “naturally” tend to have routing demands which are evenly spread across an FPGA, so they map best to a uniform routing architecture. The only (slight) exception we found to this “uniform is better” rule occurred when the I/O locations of circuits were fixed by board-level constraints. In this case making the I/O channel 25% wider than the other channels was a net benefit.

Of almost equal note is the fact that the area-efficiency is decreased only slightly by many non-uniform or direction-biased architectures, provided the pin placement on the logic blocks is well-matched to the channel capacity distribution. This means that if such architectures are desirable for other reasons the impact on core area doesn’t preclude their use. For example, one reason for widening the center channel is to re-use an existing tile layout in a larger FPGA (which needs more routing), and hence save vendor layout effort.

More specifically, of the FPGA architectures studied, a full-perimeter pin position FPGA with no directional routing bias and uniform channel widths is most area-efficient. Employing a logic block with the top/bottom pin position requires approximately 8% more routing resources than full-perimeter FPGAs, and the most area-efficient top/bottom FPGA has twice as many horizontal routing tracks as vertical ones. We also found that one can construct rectangular FPGAs which are only slightly less dense than square FPGAs provided one adjusts the degree of directional bias in the routing resources to best match the chip aspect ratio.

Our experimental results in this paper were gathered with the linear congestion cost function in the placement tool because we felt the non-linear cost function was too slow to be commercially viable. However, it is interesting to note that while the non-linear function improved the routability of circuits for all FPGA architectures, it improved routability the most for uniform routing architectures. Apparently it is easier for advanced CAD tools to spread out congested regions than it is to localize them to designated portions of a chip that have extra routing resources. Consequently, we expect that future advancements in CAD tools will tend to slightly increase the advantages of uniform routing architectures over their non-uniform counterparts.

7 References

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

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

[3] AT & T Inc., *ORCA Datasheet*, 1994.

[4] Actel Inc., *FPGA Data Book and Design Guide*, 1994.

[5] Altera Inc., *Data Book*, 1993.

[6] B. K. Britton, Y. T. Oh, W. Oswald, H. T. Nguyen, S. Singh, C. Lee, W. Leung, C. Spivak, J. Steward and C. T. Chen, “Second Generation ORCA Architecture Utilizing 0.5µ Process Enhances the Speed and Usable Gate Capacity of FPGAs,” *IEEE Int. ASIC Conf. and Exhibit*, Sept. 1994, pp. 474-478.

[7] D. Tavana, W. Yee, S. Young, and B. Fawcett, “Logic Block and Routing Considerations for a New SRAM-Based FPGA Architecture,” *CICC*, 1995, pp. 24.6.1 - 24.6.4.

[8] S. Yang, “Logic Synthesis and Optimization Benchmarks, Version 3.0, *Tech. Report*, Microelectronics Centre of North Carolina, 1991.

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

[10] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. Computer-Aided Design*, Jan. 1994, pp. 1-12.

[11] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, May 13, 1983, pp. 671 - 680.

[12] J. Lam and J. Delosme, "Performance of a New Annealing Schedule," *ACM Design Automation Conference*, 1988, pp. 306 - 311.

[13] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells", *ICCAD*, 1990, pp. 336 - 339.

[14] C. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *ICCAD*, 1994, pp. 690 - 695.

[15] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on VLSI*, Dec. 1995, pp. 473 - 482.

[16] C. Y. Lee, "An Algorithm for Path Connections and its Applications," *IRE Trans. Electron. Comput.*, Vol. EC-10, 1961, pp. 346 - 365.

[17] J. S. Rose, W. M. Snelgrove, Z. G. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," *Canadian Conf. on VLSI*, 1985, pp. 169-173.

[18] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. Computer-Aided Design*, Jan. 1985, pp. 92 - 98.

[19] J. Rose, "Parallel Global Routing for Standard Cells," *IEEE Trans. on CAD*, Oct. 1990, pp. 1085 - 1095.

[20] R. J. Francis, J. Rose, K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *ACM Design Automation Conference*, 1990, pp. 613-619.

[21] S. D. Brown, "Routing Algorithms and Architectures for Field-Programmable Gate Arrays," PhD Dissertation, University of Toronto, 1992.

Appendix A

Table 4: Benchmark Circuit Statistics.

Circuit	Nets	4 LUTs	Flip Flops	Inputs	Outputs
alu4	1536	1522	0	14	8
apex2	1916	1878	0	38	3
apex4	1271	1262	0	9	19
bbrtas	418	406	7	5	2
C6388	559	527	0	32	32
cordic	489	466	0	23	2
cps	781	757	0	24	109
daio-rec	408	311	81	16	46
dalu	575	500	0	75	16
diffeq	1935	1494	377	64	39
e64	339	274	0	65	65
ecc	451	330	109	12	14
ex4p	529	445	0	84	28
ex5p	1072	1064	0	8	63
k2	564	519	0	45	45
misex3	1411	1397	0	14	14
mm30a	591	467	90	34	30
parker	871	660	161	50	9
s1423	313	221	74	18	5
s1488	311	296	6	9	19
s5378	772	576	160	36	49
s9234.1	625	461	135	29	39
sbc	452	384	27	41	56
scf	453	418	7	28	56
seq	1791	1750	0	41	35
table3	494	480	0	14	14