A Fast Routability-Driven Router for FPGAs

Jordan S. Swartz, Vaughn Betz, and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{jsswartz,vaughn,jayar}@eecg.toronto.edu

Abstract

Three factors are driving the demand for rapid FPGA compilation. First, as FPGAs have grown in logic capacity, the compile computation has grown more quickly than the compute power of the available computers. Second, there exists a subset of users who are willing to pay for very high speed compile with a decrease in quality of result, and accordingly being required to use a larger FPGA or use more real-estate on a given FPGA than is otherwise necessary. Third, very high speed compile has been a long-standing desire of those using FPGA-based custom computing machines, as they want compile times at least closer to those of regular computers.

This paper focuses on the routing phase of the compile process, and in particular on routability-driven routing (as opposed to timing-driven routing). We present a routing algorithm and routing tool that has three unique capabilities relating to very high-speed compile:

1. For a "low stress" routing problem (which we define as the case where the track supply is at least 10% greater than the minimum number of tracks per channel actually needed to route a circuit) the routing time is very fast. For example, the routing phase (after the netlist is parsed and the routing graph is constructed) for a 20,000 LUT/FF pair circuit with 30% extra tracks is only 23 seconds on a 300 MHz Sparcstation.
2. For low-stress routing problems the routing time is near-linear in the size of the circuit, and the linearity constant is very small: 1.1 ms per LUT/FF pair, or roughly 55,000 LUT/FF pairs per minute.
3. For more difficult routing problems (where the track supply is close to the minimum needed) we provide a method that *quickly* identifies and subdivides this class into two sub-classes: (i) those circuits which are difficult (but possible) to route and will take significantly more time than low-stress problems, and (ii) those circuits which are impossible to route. In the first case the user can choose to continue or reduce the amount of logic; in the second case the user is forced to reduce the amount of logic or obtain a larger FPGA.

1. Introduction

The success of CPLDs and FPGAs is in part due to the instant manufacturability of a programmable device. As device sizes increase, however, increasing compile times have reduced the impact of instant manufacturing. This is particularly true among impatient hardware designers, emulation/rapid prototyping system users who have many FPGAs to compile, and users of FPGA-based custom computing machines who want compile times closer to those of their competition, a microprocessor.

The complete end-to-end compile time of modern large FPGAs (those with approximately 5000 or more LUT/Flip-flop pairs) is threatening to become so long that it may take a significant portion of a day to compile, or even to declare failure of compilation. For a subset of designers, these large compile times may reduce or eliminate the advantages of FPGAs.

In this paper we focus on the routing phase of the compile process, and as a first step, explore ways of making a fast routability-driven router. Although it is clearly necessary to develop a fast timing-driven router, it is first important to understand what "fast" means in the context of routability before moving on to timing-driven routing.

We envision the following scenario as the context for fast compile: the user has just designed a circuit (to the netlist level) and initially targets an FPGA of a particular size. He/she would like to quickly receive a routing (and subsequently a programming file) for that FPGA *or* be told that the routing will take significantly more time (with a time estimate) or be told that the routing task is impossible. In the latter two cases the designer has several options, depending on the circumstances. Users of a rapid-prototyping system (such as the Transmogrifier-2 [Lewi97] or the Aptix System Explorer [Apti96]) could reduce the size of the design by moving part of the circuit into a different FPGA. If the FPGA is designed into a socket that can accept FPGAs with differing logic capacities, then the designer could choose to try routing the circuit on a larger FPGA. Alternatively, the designer could remove part of the design, making it smaller, as this is typically possible in the FPGA-based computing world, where the amount of parallelism (and hence hardware) can be parameterized. Note that this scenario also requires a fast placement tool; another part of the Fast Compile Project at the University of Toronto is currently at work on this issue.

To precisely define the notion of fast compile, we have set the following goal for our router: to be able to route a 20,000 LUT/flip-flop pair circuit in 10 seconds on a modern processor. We furthermore require a running time that is linear in the size of the circuit, with a low linearity constant. Finally, since some routing problems are inherently difficult or impossible, we require that our router be able to quickly identify both of these cases in order to alert the user.

Most previous work on FPGA routing [Brow92a] [Lemi93] [Wu94] [Betz97] has focussed on achieving routes within the fewest number of tracks per channel. To our knowledge, there is no previous work which has a primary focus on fast compile time and fast identification of hard routing problems. There has been various reports of techniques for speeding up maze routing [Alle76] [Souk78] [Palc92], some of which we build on here.

This paper is organized as follows: Section 2 describes the routing algorithm and difficulty prediction approach. Section 3 compares the speed and quality of the new algorithm with those of VPR [Betz97], and Section 4 concludes.

2. Routing Algorithm

In this section we give a brief overview of the previous work upon which our algorithm is built. We assume that the reader is familiar with the basic maze routing approach [Lee61]. We then describe modifications to the basic algorithm to increase its speed and give a method to classify each routing problem as low stress, difficult, or impossible.

2.1 Base Algorithm

Our routing algorithm is based on the PathFinder algorithm [Ebel95], which is an iterative maze-type router [Nair87]. Nets are routed sequentially, and once a track segment has been used for one net, other nets *are* allowed to use that segment, but must pay a higher cost. Consequently, nets tend to avoid overuse of a segment unless it is necessary or particularly efficient. At the end of the first iteration (after all nets have been routed), either there are no segments overused and the routing is successful, or some segments are overused and more routing iterations are executed to try and resolve the contention. In each of these subsequent routing iterations, every net is ripped up and re-routed. Since the cost of over-used track segments is increased every routing iteration, they become increasingly expensive and are less likely to be used by more than one net. This gradual reduction in routing violations is a very successful routing approach.

In order for the router to have a very short run time, two conditions are necessary: all of the nets have to be routed in as few iterations as possible (ideally one iteration). Secondly, each net has to be routed as efficiently as possible, without exploring all of the possible paths exhaustively.

2.2 Speed Enhancements

We have implemented two enhancements to increase the speed of the basic breadth-first search maze router. The first is to employ a depth-first search which directs the router to head towards specific targets [Rubi74]. The second is to reduce the amount of activity on the routing expansion list for higher-fanout nets by only placing segments on the expansion list that are in the neighborhood of the target.

2.2.1 Directed, Depth-First Search

Most basic maze routers use a breadth-first search of the routing graph to make connections. While this guarantees the best connections (for two-pin nets), it means the router spends much of its time exploring paths in the wrong direction. A depth-first search (which we will call a *directed* search as it is a more evocative term for the two-dimensional routing problem) uses a narrower wavefront in order to expand in the direction of the target pin to be connected. Ideally, the directed search will simply start at the source of the net and choose successively closer track segments until the target sink is reached.

There are two key issues in the design of this kind of directed search: modifying the cost function to direct the expansion from the source towards a specific target, and choosing the correct target from the choices available in a multi-terminal net.

A form of directed search, known as the A* algorithm, was tried as part of the Pathfinder algorithm [Ebel95]. Our directed search algorithm is similar, but our choice of cost function makes our directed search more aggressive in seeking the target.

Cost Function

The following cost function is used to measure the cost of a route from the source node to a specific track segment:

$$\text{Cost} = \text{Cost}_{\text{prev}} + C_0 + \alpha(\Delta D) \qquad (1)$$

$\textbf{Cost}_{\textbf{prev}}$ is the cost of the previous track segments on the path from the source to this track segment, i.e. the cost of the track segments used to reach this one.

$\mathbf{C_0}$ is the base cost of using the segment under consideration. The base cost for a track segment is initially one, but is made very large when the track segment has already been used. By increasing the base cost quickly, we reduce the number of router iterations to resolve congestion. In the original PathFinder algorithm [Ebel95], the base cost is increased more gradually, requiring more iterations to resolve congestion.

$\mathbf{\Delta D}$ is the change in the Manhattan distance remaining to the target sink for the track segment under consideration. If the track segment is closer to the target sink than the previous track segment, then ΔD is negative, reducing the overall cost of using the track segment. A track segment that is further from the target sink will have a positive ΔD.

α is called the "direction factor", it determines how aggressively the router "drives" towards the target sink. With $\alpha = 0$ the search is equivalent to a breadth-first search. A very large α, on the other hand, will often result in excessively long connections since the nearness to the target is considered much more important than wirelength. We found experimentally that a direction factor of 1.5 produced the best results for large circuits. For $\alpha > 1$, the router is not guaranteed to find the shortest source-sink connection;

however, we have found that setting $\alpha = 1.5$ leads to a large speedup, with no measurable quality degradation.

Target Selection

The second issue in a directed search is target selection for multi-terminal nets. Since the directed search needs a specific target (in order to calculate **D** in the above equation), each sink of the net must be connected in a separate routing step. We route to the targets in order from the closest sink to the source, to the farthest sink from the source. This avoids the creation of long trunks that are not well re-used, as illustrated in Figure 1. Figure 1 (a) shows the net routing when the sink closest to the source is routed first. The further sink simply extends the existing net further, without using extra wiring unnecessarily. Figure 1 (b) shows what often happens when the furthest sink is routed first. The portion of the routing created to reach the furthest sink does not pass close by the other sink. Consequently, the net in (b) uses more track segments than the net in (a).

Figure 2 gives the pseudocode for routing a multi-terminal net. After a sink is reached, each of the track segments already assigned to the net is placed on the expansion list with a cost equal to alpha times its distance to the target sink. This guarantees that the routing will continue from the closest track segment that is already part of the net.

Net Order

The nets are routed in order of decreasing fan out. The highest fanout nets are routed first because they tend to span the whole FPGA and are much easier to route when there is no existing congestion. Low fanout nets tend to be localized, so routing them later in the routing process is not too difficult even in the presence of congestion.

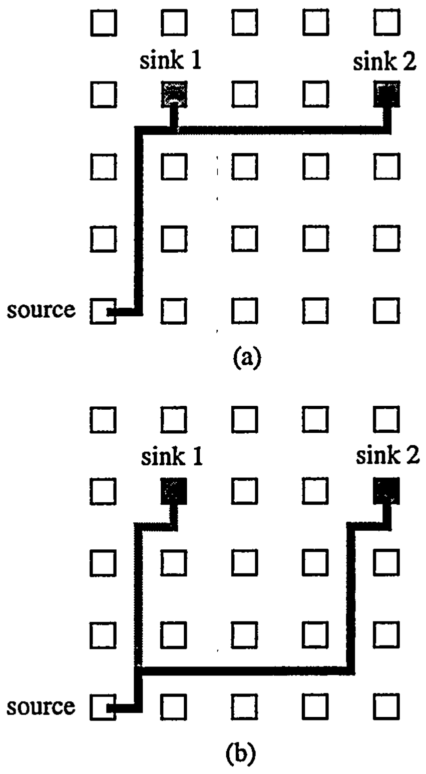

Figure 1. Two methods of routing a multi-terminal net: (a) closest sinks first; (b) furthest sinks first

2.2.2 Reducing Activity on the Expansion List

The above algorithm is somewhat inefficient because it places the entire net routed so far on the expansion list when starting to route to each sink of a net. This is often unnecessary because for higher fanout nets most of the net is

```
Sort the sinks in order from closest to furthest from the source
Target = sink closest to source
Put tracks segments attached to source onto expansion list
Remove lowest cost track segment from expansion list
While the target has not been reached
          Put neighbors of this track segment onto expansion list with cost given by (1)
          Remove lowest cost track segment from expansion list
Endwhile
Empty the expansion list
While still more sinks to route for this net
          Target = next closest (to source) unconnected sink.
          Put the whole net created up to this point onto expansion list with cost = αD
          Remove lowest cost track segment from expansion list
          While the target has not been reached
                    Put neighbors of this track segment onto expansion list
                    Remove lowest cost track segment from expansion list
          Endwhile
          Empty the expansion list
Endwhile
```

Figure 2. Pseudocode for Routing a Multi-terminal Net

unlikely to be involved in any particular connection. The expansion list is essentially used to sort the track segments in order of increasing distance to the sink, so that the first track segment removed from the expansion list is the closest one to the sink. In the worst case, for an FPGA of containing N logic blocks and a net with N sinks, the routing algorithm would exhibit $O(N^2)$ behavior. Since many circuits have at least a few extremely high fanout nets, this typically slows the router significantly.

To overcome this effect, we devised a technique called *binning*. The key idea is that only the portions of the net routed so far which are closest to the current target sink need to be placed on the expansion list. Figure 3 illustrates a simple example of the binning technique. In this example there are four bins, each containing one quarter of the total track segments. A net with fanout three is being routed, and two of the three sinks have already been routed. When routing the last sink, instead of placing the entire net on the expansion list, only those parts of the net in bin 4 are placed on the expansion list, thus reducing the number of expansion list operations. For relatively low fanout nets, binning does not save many expansion list operations. However, when used on very high fanout nets, binning significantly reduces the number of expansion list operations. We determined experimentally that binning is most effective for nets with fanout greater than 50.

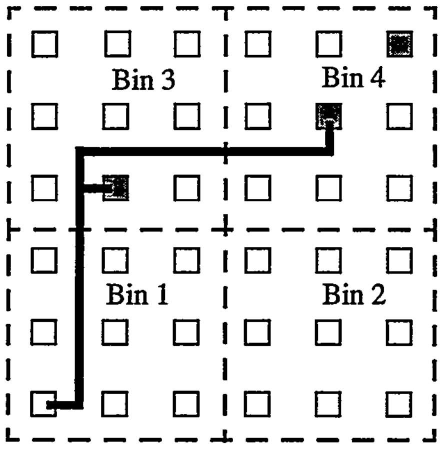

Figure 3. The Binning Technique

There are two key issues that have to be addressed with binning: the size of the bins, and what to do when a bin containing a sink does not contain any part of a net's routing.

Bin Size

The bin size is very important. If it is too small (in the extreme case the segments in just 1 logic block tile), then the quality of the routing degrades since an insufficient amount of the prior route is available as potential "start points" for the connection to the sink in that bin. If the bin size is too large (in the extreme case the entire FPGA), then unnecessary segments will be put on the expansion list and the routing time will increase. Since the average distance between sinks can vary for different nets, our router computes the proper bin size to use for each net. Before a net is routed, the average area per sink is calculated as the area of the bounding box of the net terminals divided by the number of sinks. We determined experimentally that choosing a bin size of four times the average area per sink for each net provides the best results.

Empty Bins

If the bin containing a sink does not contain any part of the route so far, then the portions of the net in its eight neighboring bins are added to the expansion list. The neighboring bins may contain parts of the route relatively close to the target sink. If the neighboring bins are also empty, then the entire existing net is placed on the expansion list.

2.3 Difficulty Prediction

A key aspect of high-speed routing is the ability to quickly predict when the routing problem is very hard (and hence will take a longer time to complete) or impossible. In both of these cases, it is important to inform the user that the result will either be a long time coming, or simply isn't possible to achieve. In order to predict these cases (after placement), we need to correlate routing difficulty with parameters that we can quickly determine from the target FPGA and the user circuit. For a given circuit, with a specific placement, we define W_{min} as the minimum number of tracks per channel that our router would require in order to successfully route the circuit. Let the number of tracks per channel in the FPGA be W_{FPGA}. Table 1 gives a reliable prediction of routing difficulty in terms of W_{min} and W_{FPGA}. Typically, when W_{FPGA} is 10% greater than W_{min} then the routing problem is not difficult and can be solved very quickly using the directed search approach. As the number of tracks per channel decreases below this threshold, the problem rapidly becomes more difficult. Note that this 10% figure is a rough estimate of when the onset of difficulty occurs, from experimental data.

Definition		Predictor
Classification	Time (seconds)	Typical Range of Tracks Per Channel in FPGA
Impossible	--	$W_{FPGA} < W_{min}$
Difficult	> 60	$W_{min} \leq W_{FPGA} < 1.1W_{min}$
Low Stress	< 60	$W_{FPGA} \geq 1.1W_{min}$

Table 1: Definition of Routing Classes

Since W_{min} is not known before-hand, we need a method for rapidly estimating W_{min} to make any useful difficulty predictions from Table 1. We call $W_{estimate}$ the estimate of

W_{min}. In order to calculate $W_{estimate}$ we use a placement wirelength model based on the approach of [Chen94].

The wirelength needed to route each net is estimated from the half-perimeter of the bounding box of the net terminals multiplied by a fanout-based correction factor. The correction factor compensates for the fact that the bounding box half-perimeter underestimates wiring for nets with more than 3 terminals. The correction factor is 1 for nets with 2 or 3 terminals and slowly increases with net fanout, reaching 19 for nets with 3000 terminals. We can obtain an estimate of the total wirelength by summing the expected wirelength of each net.

[Chen94] contains correction factors for nets with up to 50 terminals; we re-print them in Table 2. To determine the correction factors for higher fanout nets, we routed the larger MCNC benchmark circuits ignoring congestion, and recorded the actual wirelength for each net. By dividing the actual wirelength by the bounding box half perimeter, we obtained average correction factors for nets with up to 3000 terminals. Instead of storing discrete values for all the correction factors for nets with 50 to 3000 terminals, we fit equations (2) and (3) to the data. In (2) and (3), C is the correction factor and k is the number of terminals. With these correction factors, our estimate of total wirelength was within 5% of the actual wirelength for all of our benchmark circuits. Simply linearly extrapolating the [Chen94] correction factors led to estimates of total wirelength that were up to 25% to high.

Num Terminals	Correction Factor	Num Terminals	Correction Factor
1 ~ 3	1.00	15	1.69
4	1.08	20	1.89
5	1.15	25	2.07
6	1.22	30	2.23
7	1.28	35	2.39
8	1.34	40	2.54
9	1.40	45	2.66
10	1.45	50	2.79

Table 2: Correction Factors up to 50 [Chen94]

$$C(k) = 0.026 \cdot k + 1.49 \quad \text{for } 50 < k < 85 \qquad (2)$$

$$C(k) = -0.0000018 \cdot k^2 + 0.011 \cdot k + 2.79 \quad \text{for } k \geq 85 \qquad (3)$$

An FPGA consisting of N logic blocks contains $2 \cdot N \cdot W_{FPGA}$ track segments. Consequently, we can estimate the required channel width to route as:

$$W_{estimate} = \lceil \text{total estimated wirelength} / (2 \cdot N \cdot U) \rceil \qquad (4)$$

U is the "track segment utilization" -- the fraction of the total number of track segments in the FPGA that a router can typically use before congestion results in some unroutable nets. We have determined experimentally that for our router targeting the FPGA architecture described in Section 3.1, U is 0.56.

Since we know W_{FPGA} before routing a circuit and we have a method to calculate $W_{estimate}$ from the placement, we can use Table 1 to predict the difficulty of a routing problem.

3. Results

In this section we describe several experiments comparing the new router with VPR [Betz97] and illustrate its speed, near-linear complexity, the effectiveness of the enhancements, and difficulty prediction.

3.1 FPGA Architecture

We employed an island style architecture with unit-length track segments, connection block flexibility $F_c = W$ and a switch block flexibility of $F_s = 3$. The logic block consists of a 4-input look-up table and a single D flip-flop [Betz97]. The switch block used was the Wilton switch block [Wilt97], which is a non-planar switch block that provides improved routability.

3.2 Benchmark Circuits

The benchmark circuits used are listed in Table 3. These come from three sources, two of which are the MCNC suite [Yang91], and the RAW benchmark suite [Babb97]. Since we are principally interested in large circuits, we also used the synthetic benchmark circuit generator developed at the University of Toronto [Hutt97] to create several very large benchmarks. Although the latter circuits are actually somewhat more difficult than real circuits, we believe they are perfectly reasonable test cases for the compile time issue.

Each of the MCNC and RAW benchmark circuits was synthesized with the SIS [Sent92] package and technology mapped using Flowmap [Cong94]. These were packed into logic blocks using VPACK [Betz97]. The synthetic circuits were only packed into the logic blocks using VPACK [Betz97] as they are generated in technology-mapped form. The circuits range in size from 3,556 logic blocks up to 19,600 logic blocks.

Each circuit was placed using the VPR tool, which uses a simulated-annealing algorithm [Betz97]. Table 3 lists the minimum number of tracks per channel required by the new router and VPR. For both placement and routing, VPR was run using the "fast" to flag reduce the run-time while giving up a small amount of quality. VPR currently holds the world record for track count on a number of standard benchmark circuits when run without the "fast" flag. Our router is clearly of high quality, since it is on average only 2% worse than VPR for minimizing track count.

3.3 Routing Time

In this section we establish the speed of the router over a range of track counts. Table 4 lists the track counts and the time it took the new router to route each circuit. The track

Circuit	Source	Num. Logic Blocks	New Router Min. Track Count (W_{min})	VPR Min Track Count
beast10k	GEN	9800	22	22
beast12k	GEN	11760	23	23
beast14k	GEN	13720	26	24
beast16k	GEN	15680	23	23
beast18k	GEN	17640	26	25
beast20k	GEN	19600	30	29
bubble sort	RAW	12293	10	9
clma	MCNC	8383	12	12
elliptic	MCNC	3604	12	12
ex1010	MCNC	4598	14	13
frisc	MCNC	3556	12	12
pdc	MCNC	4575	16	16
s38417	MCNC	6406	8	9
s38584.1	MCNC	6447	9	9
spla	MCNC	3690	14	14

Table 3: Benchmark Circuits Data

counts range from W_{min} up to $W_{min} + 40\%$. Recall that W_{min} is the minimum number of tracks per channel our router needs to successfully route a circuit. Execution times were measured on a 300 MHz UltraSPARC 3200 with 1 GByte of memory, and do not include the time to parse the netlist and generate the routing graph. For the largest circuit the parse and graph generation time was 20 seconds. The largest circuit (beast20K) required 200 MBytes of memory.

Notice that we have not yet achieved our informal goal of a 10 second routing time for a 20,000 logic block circuit, as it requires 23 seconds to route the circuit beast20k with 30% extra tracks, but we're getting close.

It is instructive to observe how the routing time of the new router changes as the available track count, W_{FPGA}, increases. Figure 4 plots the routing time for the new router and VPR versus the number of tracks available, for the 8383 logic block circuit clma. It is clear that once there are sufficient tracks the new router completely routes the circuit in about 6 seconds, independent of the number of tracks. The speedup as W_{FPGA} increases comes from two factors: fewer routing iterations (eventually, only 1) are needed to resolve congestion; and the directed search can more rapidly route each net when there is little congestion to detour around. Observe that the VPR router takes a great deal more time, and the time increases as W_{FPGA} increases (for large W_{FPGA}) because of the breadth-first search nature of the VPR router.

3.4 Experimental Complexity Measurement

A key goal for the high-speed routing project is to achieve linear time complexity for low stress situations, with a very small linearity constant. Figure 5 shows a graph of the W_{min} + 40% routing times versus the number of logic blocks for

Circuit	W_{min}		W_{min}+10%		W_{min}+20%		W_{min}+30%		W_{min}+40%	
	Track Count	Time (s)	Track Count	Time (s)	Track Count	Time (s)	Track Count	Time (s)	Track Count	Time (s)
beast10k	22	96	25	30	27	18	29	8	31	9
beast12k	23	372	26	41	28	24	30	12	33	12
beast14k	26	175	29	31	32	15	34	15	37	16
beast16k	23	291	26	63	28	30	30	14	33	14
beast18k	26	330	29	39	32	19	34	18	37	20
beast20k	30	430	33	83	36	50	39	23	42	24
bubble sort	10	56	11	16	12	10	13	11	14	5
clma	12	909	14	33	15	36	16	12	17	6
elliptic	12	34	14	7	15	4	16	2	17	2
ex1010	14	31	16	8	17	5	19	2	20	2
frisc	12	173	14	15	15	7	16	5	17	2
pdc	16	928	18	17	20	8	21	8	23	4
s38417	8	79	9	15	10	8	11	5	12	3
s38584.1	9	33	10	16	11	19	12	9	13	4
spla	14	91	16	11	17	5	19	2	20	2

Table 4: Routing Times

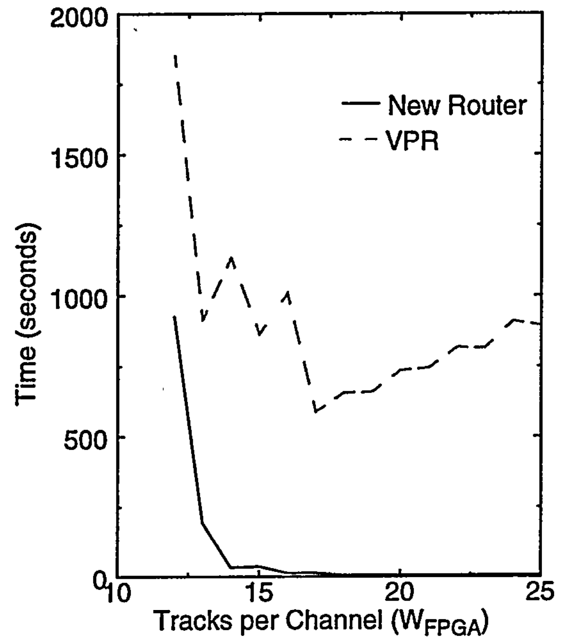

Figure 4. Routing Time vs. Available Tracks for clma (8383 Logic Blocks)

all of the benchmark circuits. The dashed line shows the least-squares line that best fits the data. The fitted equation of the line is:

$$\text{run time} = 0.0011 \cdot N - 1.2 \qquad (5)$$

where N is the number of logic blocks. The correlation coefficient for this linear approximation is 0.95, strongly suggesting that the run time is indeed essentially linear. If we use the time-constant of (5), 0.0011, we can effectively route 55,000 logic blocks per minute.

3.5 Effectiveness of Enhancements

In order to measure the effectiveness of the two router enhancements, directed search and binning, each of the benchmark circuits was routed using three different routers: (i) the VPR breadth-first router, (ii) the new router with directed search only, and (iii) the new router with directed search and binning (the version used to get the timing results in previous sections).

The two metrics used to compare the three routers were the minimum track count needed to route the circuit and the low stress routing time. Experimental results, geometrically averaged across all fifteen benchmark circuits, are given in Tables 5 and 6. In terms of average minimum track count, all the routers performed almost equally well; the directed search with binning requires only 2% extra tracks per channel. The low stress routing times were measured by routing each circuit with W_{min} + 30% tracks. Most of the speedup is obtained from the directed search, which is 52 times faster than the breadth-first search. The addition of binning provides an extra speedup of 2 over the directed search.

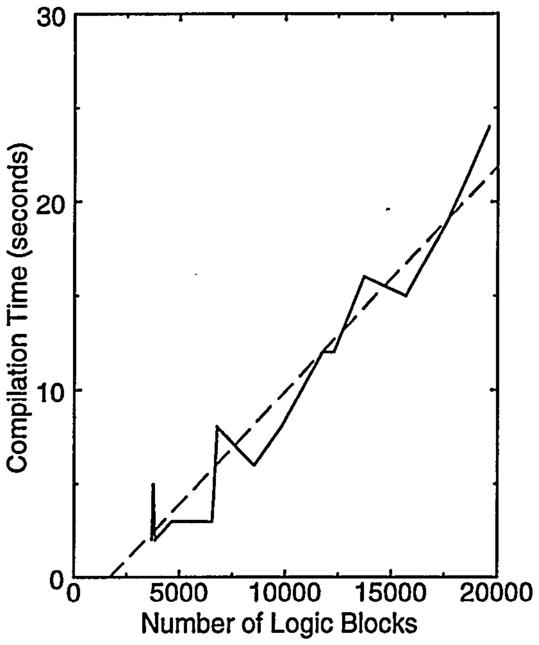

Figure 5. Compile Time vs. Circuit Size

Algorithm	Average Minimum Track Count
Breadth-first Search	15.5
Directed Search	15.5
Directed Search with Binning	15.8

Table 5: Average Minimum Track Count Results

Algorithm	Average Low Stress Routing Time (s)
Breadth-first Search	731
Directed Search	14
Directed Search with Binning	7

Table 6: Average Low Stress Routing Time Results

3.6 Difficulty Prediction

The final important feature of the new router is its ability to detect the difficulty of the routing task. To test the difficulty prediction scheme, we ran the router for each circuit to get the estimated tracks per channel using (4). $W_{estimate}$ took less than less than one second to calculate for the largest benchmark circuit, providing the user with feedback on the problem classification very quickly. Table 7 lists the actual minimum tracks per channel and the estimated tracks per channel for each benchmark circuit.

Circuit	W_{min}	$W_{estimate}$	Difference
beast10k	22	22	0
beast12k	23	24	1
beast14k	26	25	-1
beast16k	23	23	0
beast18k	26	26	0
beast20k	30	29	-1
bubble sort	10	8	-2
clma	12	13	+1
elliptic	12	11	-1
ex1010	14	12	-2
frisc	12	13	+1
pdc	16	16	0
s38417	8	8	0
s38584.1	9	8	-1
spla	14	14	0

Table 7: Track Count Estimates

The last column in Table 7 shows the difference between $W_{estimate}$ and W_{min}. For thirteen of the circuits, the estimates were within ±1 track per channel, for the remaining two circuits the estimates were under by two tracks per channel. These inaccuracies will result in some mistakes by the prediction scheme of Table 1.

In order to illustrate the effect of the inaccuracies in predicting difficulty, we ran the router on each benchmark circuit using five different track counts: the minimum required by the circuit W_{min}, W_{min}+1, W_{min}-1, W_{min}-2, and W_{min}-3. We chose these values because it is within this range that inaccuracies in $W_{estimate}$ will affect the routability predictor. Table 8 lists for each circuit: the correct (Crct) difficulty level for each circuit based on the definition from Table 1, the reported (Rpt) difficulty by the router using $W_{estimate}$ and applying the predictor from Table 1, and the routing time. The following key is used: LS=low stress, DF=difficult, and IM=impossible.

There are two types of errors in Table 8, difficult/impossible errors and low-stress/difficult errors. We only concern ourselves with the difficult/impossible errors because they can cause the user to think that their circuit can be routed even though it is impossible. The worst outcome of a low-stress/difficult error is that the user ends up waiting a few minutes for a circuit to route, even though the router classified the problem as low stress. The difficult/impossible errors are highlighted with shading in Table 8. Out of the 75 test cases, 11 were difficult/impossible errors, resulting in an accuracy of 84%.

We can reduce the severity of the difficult/impossible errors by providing the user with fuzzy feedback when $W_{estimate}$ is within -1 to +2 tracks per channel of W_{FPGA}. Table 9 shows how we can redefine our difficulty prediction scheme. When W_{FPGA} is less than $W_{estimate}$-1, we can say with near certainty that the problem is impossible. When W_{FPGA} is

Circuit	W_{min}+1			W_{min}			W_{min}-1			W_{min}-2			W_{min}-3		
	Crct	Rpt	Time (s)	Crct	Rpt	Time (s)	Crct	Rpt	Time (s)	Crct	Rpt	Time (s)	Crct	Rpt	Time (s)
beast10k	LS	DF	57	DF	DF	96	IM	IM	--	IM	IM	--	IM	IM	--
beast12k	DF	DF	61	DF	IM	372	IM	IM	--	IM	IM	--	IM	IM	--
beast14k	DF	DF	140	DF	DF	175	IM	DF	--	IM	IM	--	IM	IM	--
beast16k	DF	DF	95	DF	DF	291	IM	IM	--	IM	IM	--	IM	IM	--
beast18k	DF	DF	181	DF	DF	330	IM	IM	--	IM	IM	--	IM	IM	--
beast20k	DF	DF	326	DF	DF	430	IM	DF	--	IM	IM	--	IM	IM	--
bubble sort	LS	LS	16	LS	LS	56	IM	DF	--	IM	DF	--	IM	IM	--
clma	DF	DF	191	DF	IM	909	IM	IM	--	IM	IM	--	IM	IM	--
elliptic	LS	LS	25	LS	DF	34	IM	DF	--	IM	IM	--	IM	IM	--
ex1010	LS	LS	5	LS	LS	31	IM	DF	--	IM	DF	--	IM	IM	--
frisc	LS	DF	39	DF	IM	173	IM	IM	--	IM	IM	--	IM	IM	--
pdc	DF	DF	99	DF	DF	928	IM	IM	--	IM	IM	--	IM	IM	--
s38417	LS	LS	15	DF	DF	79	IM	IM	--	IM	IM	--	IM	IM	--
s38584.1	LS	LS	16	LS	LS	33	IM	DF	--	IM	IM	--	IM	IM	--
spla	LS	DF	22	DF	DF	91	IM	IM	--	IM	IM	--	IM	IM	--

Table 8: Correct and Reported Difficulty (LS=low stress, DF=difficult, IM=impossible)

equal to $W_{estimate}$-1, we can classify the problem as impossible, but inform the user that the problem may be difficult. When W_{FPGA} is at least equal to $W_{estimate}$ but less than $W_{estimate}$+2, we can classify the problem as difficult, but warn the user that the problem may be impossible.

Definition		Predictor
Classification	Time (seconds)	Typical Range of Tracks Per Channel in FPGA
Impossible	--	$W_{FPGA} < (W_{estimate} - 1)$
Probably impossible, but could be difficult.	--	$W_{FPGA} = (W_{estimate} - 1)$
Probably difficult, but could be impossible	> 60	$W_{estimate} \leq W_{FPGA} < (W_{estimate} + 2)$
Difficult	> 60	$(W_{estimate} + 2) \leq W_{FPGA} < 1.1(W_{estimate} + 2)$
Low Stress	< 60	$W_{FPGA} \geq 1.1(W_{estimate} + 2)$

Table 9: Fuzzy Definition of Routing Classes

When a routing problem is given a fuzzy classification, it is up to the user to decide whether to try and route the circuit or to stop and change the circuit or the target FPGA. We expect that in most cases W_{min} will lie somewhere outside the fuzzy prediction region. In such cases, where W_{min} is considerably greater than or less than W_{FPGA}, our predictor is highly accurate.

4. Conclusions and Future Work

We have presented a fast routability-driven router for FPGAs. The router is of particular interest to users who are willing to accept slightly lower quality results in exchange for extremely short routing times. The routing algorithm has three distinguishing capabilities. (i) In low stress situations, where the channel width is at least 10% greater than the minimum channel width actually needed, the router is very fast. For example, it can route a 20,000 logic block circuit in 23 seconds on a 300 MHz sparcstation. (ii) For low stress routing problems, the routing time scales very close to linearly with circuit size, with a linearity constant of 0.0011. (iii) For difficult routing problems, the router is able to predict that the problem is difficult, with high accuracy.

In the future we plan to try our router with segmented FPGA architectures. We are also going to develop a fast timing-driven router.

5. Acknowledgments

The authors would like to thank Russell Tessier for providing the bubble sort benchmark circuit. We also wish to acknowledge funding from Lucent Technologies and MICRONET.

6. References

[Alle76] J. R. Allen, "A Topologically Adaptable Cellular Router," *DAC*, 1976, pp. 161-167.

[Apti96] Aptix Corporation, *Product Brief: The System Explorer MP4*, 1996. This and other documents are available on the Aptix web site: http://www.aptix.com.

[Babb97] J. Babb, M. Frank, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, P. Finch, and A. Agarwal, "The RAW Benchmark Suite: Computation Structures for General Purpose Computing," *FCCM*, 1997, pp. 161-171.

[Betz97] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," Int'l Workshop on *FPL*, 1997, pp. 213-222.

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

[Brow92a] S. Brown, J. Rose, and Z. G. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," *IEEE Trans. on CAD*, May 1992, pp. 620-628.

[Chen94] C. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", ICCAD, 1994, pp. 690-695.

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

[Corm90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The Massachusetts Institute of Technology, 1990, pp. 469-485.

[Ebel95] 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.

[Hutt97] M. Hutton, J. Rose, and D. Corneil, "Generation of Synthetic Sequential Benchmark Circuits," *FPGA*, 1997, pp. 149-155.

[Korn82] R. Korn, "An Efficient Variable Cost Maze Router", Proc. 19th DAC, June 1992.

[Lee61] C. Y. Lee, "An Algorithm for Path Connections and its Applications," *IRE Transactions on Electronic Computers*, Vol. EC=10, 1961, pp. 346-365.

[Lemi93] G. Lemieux and S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," ACM/SIGDA Physical Design Workshop, 1993, pp. 215-226.

[Lewi97] D. M. Lewis, D. R. Galloway, M. van Ierssel, J. Rose, and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System," *FPGA*, 1997, pp. 53-61.

[Nair87] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Trans. on Computer-Aided Design*, vol. CAD-6, no. 6, March 1987, pp. 165-172.

[Palc92] M. Palczewski, "Plane Parallel A* Router and its Application to FPGAs," *DAC*, 1992, pp. 691-697.

[Rubi74] F. Rubin, "The Lee Path Connection Algorithm", IEEE Trans. Computers, vol c-23, no. 9, Sept. 1974.

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

[Souk78] J. Soukup, "Fast Maze Router," *Proc. 15th Design Automation Conf.*, June 1978, pp. 100-102.

[Wilt97] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories," *Ph.D. Dissertation*, University of Toronto, 1997.

[Wu94] Y.-L. Wu and M. Marek-Sadowka, "An Efficient Router for 2-D Field-Programmable Gate Arrays," *EDAC*, 1994, pp. 412-416.

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