Negotiated A* Routing for FPGAs[*]

Russell Tessier
MIT Laboratory for Computer Science
Cambridge, MA 02139

Abstract

In the next few years, logic capacities for field-programmable gate arrays are expected to exceed one million gates per device. While this expansion of FPGA device resources offers the promise of exceptional fine-grained performance for developing technologies such as ASIC prototyping and FPGA computing, supporting computer-aided design tools have yet to be developed to target these devices rapidly and efficiently. This paper addresses the compilation time issue for routing array FPGAs with segmented routing architectures. By treating the routing problem as an A$*$ search, it is possible to trade additional device routing resources for decreased router run-time by converting an exhaustive breadth-first maze route into a shorter depth-first route. It is shown that for the depth-first case, the sparse nature of FPGA routing switches in commerical architectures, such as the Xilinx XC4000 family, necessitates an additional localized search near net inputs, called *domain negotiation*, to aid in directing the route of each design net onto a set of routing resources most likely to lead to a successful route. For a set of large FPGA benchmarks, a route time speedup of over an order of magnitude for an iterative maze router configured for depth-first routing is shown when compared to the same router configured for a breadth-first search.

1 Introduction

A major limitation in the use of contemporary multi-FPGA systems is the amount of time required to place and route circuit components inside the individual FPGA devices. For many systems, this compile time is on the order of hundreds of CPU hours as opposed to tens of minutes for typical microprocessor-based computing platforms. Such long turn-around time from conceptual development to physical implementation significantly limits the on-the-fly modification capability of existing FPGA computing applications. In many cases maximum device performance and utilization is of less importance than getting a feasible solution implemented as soon as possible with given resources.

Most commercial routers in use today are variants of the Lee maze routing algorthm for path determination between two vertices on a planar rectangular grid [9]. This algorithm attempts to locate the shortest path between two points while avoiding grid obstacles in the form of used routing resources. By varying the number of neighboring grid points examined for each evaluated grid location in the route, the search process can be varied between a full breadth-first search in which nearly every candidate grid point between source and destination is examined and one that is depth-first in which the first, best route is taken.

Previously [8] [13], it was observed that the process of selectively expanding candidate grid points from source to destination in the maze route is actually a special case of a more general A$*$ search algorithm [12]. In this paper the relationship between routing run time and required routing resources is examined by varying both routing track widths for given designs and the grid point expansion rate along a given route. While the basic routing algorithm used in this new router is similar to the one described recently in [15], an important extension has been made to support routing structures commonly found in commercial devices such as those in the Xilinx XC4000 series. In this paper, it is shown that a localized search near net inputs prior to net routing is necessary to identify routing resources most likely to allow route completion. Without this step, A$*$ routing targetted to these existing devices is frequently inefficient and can be infeasible in some cases.

The organization of this paper is as follows. Section 2 reviews the FPGA architecture used for this study, outlines existing approaches to FPGA routing, and draws comparisons to A$*$ routing. In Section 3, an optimization called *domain negotiation* which significantly reduces the number of nodes evaluated in

[*]Appears in Proceedings of The Fifth Canadian Workshop on Field-Programmable Devices, 1998 (FPD98).

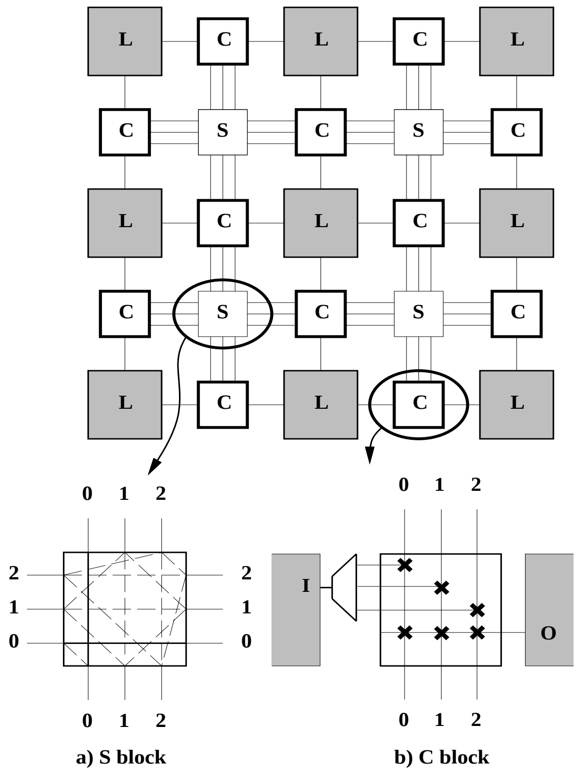

Figure 1: Array-based FPGA Model

depth-first routing is presented. Section 4 discusses the details of the iterative maze routing implementation. In Section 5, experimental results obtained by applying both breadth-first and depth-first routers to a collection of large FPGA circuits are presented. Finally, Section 6 summarizes this research and outlines directions for future work including suggestions on how the placement-time problem may be addressed.

2 Background

2.1 FPGA Architecture

The model for FPGA architecture used in this paper is the same as that used for previous studies [6] [7] and closely models the architecture of several commercial FPGAs [1] [2]. The contents of logic blocks (L) are a single look-up table/flip-flop pair. As illustrated in Figure 1, routing channels of width W (in this case 3) are connected to logic blocks through a set of programmable switches, referred to as connection or *C blocks*, at the intersection of logic block IO terminals and channel tracks. For this paper, it is assumed that the connection block is flexible enough to connect logic block IOs to any routing track in the channel ($F_c = W$). A distinctive architectural feature of the FPGA is how each C block is constructed [10]. If each logic block input connection is implemented as a pass transistor, then two or more connections to the pin may be activated to permit a routing dogleg, where the pin and connected wires are shorted together to form a single electrical path. However, as seen in Figure 1b, if the input connections are implemented as a multiplexor, only one connection to the tracks can be made and doglegs are not possible. To maintain consistency with previous studies, the latter, no-dogleg case for logic block inputs is assumed.

Wire segments in the routing channels span one or more logic blocks in the horizontal or vertical dimension. Switchboxes, or S blocks, allow a predefined set of programmable connections between wires at the intersection of horizontal and vertical track channels. Figure 1a shows that each switchbox is sparsely connected so that each horizontal or vertical wire entering the switchbox can connect to only three possible destinations ($F_s = 3$). For wire segments that span multiple logic blocks, such as those labelled 0 in Figure 1a, wiring passes directly through the S-block and is represented as a solid line. Programmable S-block connections between segments are represented with dashed lines. The limited connectivity of the switchbox topology divides routing tracks into disjoint routing *domains*. In Figure 1, a total of three domains are indicated. With the given S block and no-dogleg restrictions, a net beginning in a given track domain at the net output pin is restricted to wire segments in that domain, no matter which S block switches it goes through or net input pins it touches. The sparseness of the switchbox coupled with the inability to switch tracks at logic block inputs leads to the constraint that routing domain changes can only occur at net *outputs* even for nets with high fanout.

2.2 A* Routing

In general, FPGA routing may be defined as a graph problem [11]. Routing resources in an FPGA and their connections may be represented by a graph $G = (V, E)$ where V represents the routing nodes or tracks and E represents the connections between the wires or switches. Additionally, each node has an associated cost, c_i, which indicates its current usage, or *occupancy*, among nets targetted to the device. For successfully route completion, each node should have have an occupancy of at most one net.

Algorithm A* [12] may be applied to this routing problem [8] by considering an evaluation function f at each node n_i in the partial route from a two-point net source to destination as:

$$f_i = g_i + d_i \tag{1}$$

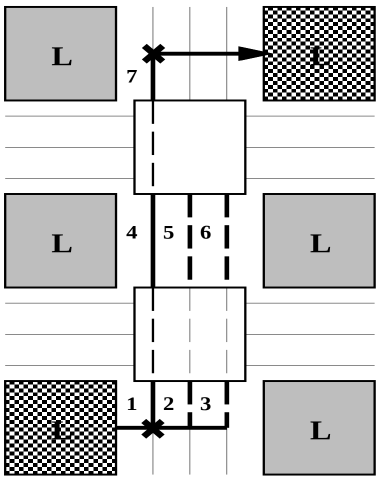

Figure 2: Breadth-first Route

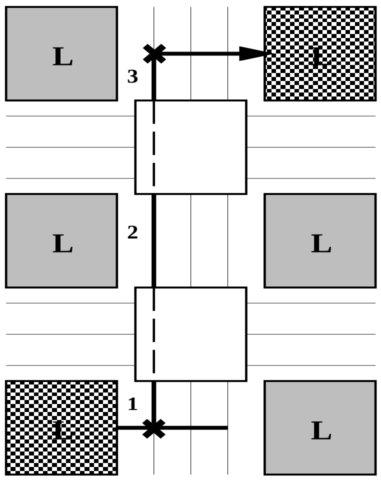

Figure 3: Depth-first Route

where g_i is the cost of the path from the source through n_i and d_i is the *estimated* cost of the path from n_i to the destination.

Value g_i is represented in most maze routing algorithms [9] [11] as the total cost of the previous path f_{i-1} plus the cost of the next candidate node or:

$$g_i = f_{i-1} + c_i \tag{2}$$

Typically, the estimate of the path cost from node to destination d_i is ignored, giving $f_i = g_i$. Since maze routing algorithms proceed by expanding around the lowest cost path (f_i) under consideration, the net effect of considering only g_i is a *breadth-first* search, leading to a minimum-cost, shortest-path route. If instead, the preceding path cost (f_{i-1} in Equation 2) is ignored and the path cost estimate (d_i in Equation 1) is set to the Manhattan distance from node to destination, maze route expansion of the lowest cost path will lead to expansion of the lowest-cost node closest to the destination. By following this rule, a sub-optimal, but much faster, *depth-first* search is performed.

Searches between depth-first and breadth-first can be created by weighting the effect of g_i and d_i via a scaling factor α between 0 and 1:

$$f_i = (1 - \alpha) \times (f_{i-1} + c_i) + \alpha \times d_i \tag{3}$$

The node cost, c_i, is used to avoid the use of nodes occupied by previous routes.

3 Domain Negotiation

The transition from a breadth-first route to a depth-first one requires the disjoint nature of the routing architecture be taken into account. Consider a breadth-first route from a two-point source to destination assuming that all routing nodes have the same cost c_i and span a single logic block. Figure 2 illustrates that node expansion takes place across *all* track domains at the same Manhattan distance from the destination prior to expansion of adjacent nodes due to the restriction of always expanding shortest paths. Thus, the final route is completed using tracks in a domain found to have a feasible connection along the source-destination path.

Alternately, in the depth-first case, the node closest to the destination is expanded first before additional points at the same Manhattan distance are expanded. As seen in Figure 3, the net effect of this expansion approach and the disjoint nature of the routing switches is a directed route confined to the same track domain from net output to input. If routing along an initial domain fails, subsequent depth-first routes can be attempted on different domains until route completion is achieved.

A key issue in this depth-first route is deciding the domain order in which routes are attempted. Since domains can not be switched in the course of the route, it is desirable to first attempt routing in domains that have a high likelihood of successful completion so that expansion in additional domains will not be necessary. To perform this domain selection the concept of *domain negotiation* is introduced. Prior to routing each net, domains are ranked based on the occupancy of domain tracks adjacent to net input pins. This localized search ranks domains as more likely to succeed in a depth-first route if the current track occupancy around the inputs in a domain is small and less likely to succeed if occupancy is high.

Details of the domain negotiation algorithm are shown in Figure 4. Since logic blocks have multiple

```
Loop over all nets $N$
    Loop over track domains
        Initialize domain cost $C_d$ to 0
        Loop over each destination input block
            Loop over tracks adjacent to input pins
                Add track occupancy to cost $C_d$
            End
            If all tracks adjacent to inputs have
                occupancy $> 1$
                Add penalty $P$ to $C_d$
        End
    End
    Order domains by costs $C_d$
    Assign each domain a rank $r_d$ based on
        cost $C_d$
End
```

Figure 4: Algorithm: Domain Negotiation

input pins that are logically equivalent, occupancy of a single domain track adjacent to a logic block input pin does not prevent route completion in a given domain, but does make it more difficult. Competition for routing resources is reflected in a cost value C_d assigned to each domain. As each net input logic block is visited, domain C_d values are incremented with the occupancy of domain tracks adjacent to logic block input pins. If tracks for all logic block input pins in a given domain are occupied, the route can not be completed in the domain without creating at least one track with occupancy greater than one, a non-feasible physical implementation. To reflect this undesirable situation, domain C_d values are incremented by a penalty factor P for each input logic block that has all adjacent input pin domain nodes occupied by at least one net. Through experimentation it was found that a penalty value of $P = 2000$ worked best in minimizing the number of node expansions required for routing completion and resulted in minimum track width routes.

As a first step in depth-first routing following domain negotiation, all domains are ranked based on their cost value C_d. The routing domain that has the minimum C_d value is given the smallest rank r_d, while the domain with the largest C_d value is given the largest r_d value. Other intermediate domains are labelled with rank values indicating their relative C_d value.

```
Order the sinks using Prim's Algorithm.
Perform Domain_Negotiation.
Target = sink closest to source.
Put track segments attached to source onto expansion list
    with cost given by (4).
Remove lowest cost track segment from expansion list.
While the net input has not been reached.
    Put neighbors of this track onto expansion list
        with cost given by (3).
    Remove lowest cost track segment from expansion list.
Endwhile
Empty the expansion list.
While still more sinks to route for this net.
    Target = next sink determined from Prim's Algorithm.
    Put whole net created to this point onto expansion list
        with $cost = \alpha \times d_i$.
    Put track segments attached to source onto expansion list
        with cost given by (4).
    Remove lowest cost track segment from expansion list.
    While the net input has not been reached.
        Put neighbors of this track onto expansion list
            with cost given by (3).
        Remove lowest cost track segment from expansion list.
    Endwhile
    Empty the expansion list.
Endwhile
```

Figure 5: Pathfinder Iteration for a Multi-terminal Net

4 Router Implementation

To support experimentation, the iterative maze router in the Versatile Place and Route (VPR) tool suite [4] was modified to support depth-first routing and domain negotiation in addition to its original support of breadth-first routing. This router is based on the PathFinder negotiated congestion algorithm [11] which consists of routing each net with a maze router and then ripping up and rerouting each net in sequence a number of times. The cost of each routing node in the routing grid is updated not only after the route of each net but also after an entire iteration in which every net is routed. This additional cost update allows for migration of routes away from congested areas of the device to those more sparsely populated through use of a non-decreasing historical cost factor.

The modified maze router differs significantly from the breadth-first original in its evaluation of multi-terminal nets. In the depth-first case, a specific target input must be specified to calculate the distance d_i in Equation 3. As a result, each input must be connected in a separate routing step. In an attempt to minimize overall wire length, the depth-first router orders target inputs using Prim's shortest-path algorithm [14]. The first target input is chosen to be the one closest to the net output. Subsequent targets are selected by choosing the input with the shortest path to the net output or to the inputs already chosen. As suggested in [15], nets are routed in order of decreasing fanout. In general, high fanout nets are easier to route when there is less existing routing congestion.

The details of the PathFinder algorithm modified for depth-first routing are shown in Figure 5. An expansion list is used to maintain a list of possible tracks for expansion and their related costs. For each net input, the expansion list is initialized to the existing route of the multi-fanout net including the output pin. If routing fails to complete in the domain used by previous net inputs, a new path back to the output pin of the the multi-pin net must be created to allow for a domain change.

The order in which domains are searched is controlled by the rank, r_d, of a given domain. As determined during the domain negotiation stage, domains with lower congestion will have a lower rank, r_d, thus promoting routing in less-congested domains first. This rank may be added to tracks attached to the net source by modifying the cost function in Equation 3 to:

$$f_i = (1 - \alpha) \times (f_{i-1} + c_i) + \alpha \times d_i + r_d \qquad (4)$$

All other tracks are added to the expansion list using the cost function in Equation 3. It was found experimentally that an α value of 0.6 significantly accelerated routing time without creating a significant loss in routing quality.

Excluding the italicized domain negotiation steps, the routing iteration shown in Figure 5 is similar to the one discussed in [15]. This previous router was targetted to an enhanced FPGA routing architecture that did not contain the disjoint switchbox commonly found in commercial architectures such as the Xilinx XC4000 family. In the next section it is shown that domain negotation plays an important role in achieving efficient results for the disjoint switchbox case.

5 Results

Both the breadth-first and depth-first versions of the iterative router were applied to a number of large FPGA benchmarks. Each design was placed and routed in the smallest square FPGA which could contain it. Target FPGAs had the following track length distribution for each routing channel: 44% of channel tracks span one logic block, 22% span two logic blocks, and 33% span the entire array. This length distribution is the same as that found in devices from the Xilinx XC4000 family.

Circuit	Source	Logic Blocks	DFS-neg Min. Tracks	BFS Min. Tracks
fft16	RAW	11860	12	12
ssp96	RAW	12041	13	12
spm16	RAW	6632	11	11
bubble	RAW	8453	8	8
frisc	MCNC	3556	15	15
s38417	MCNC	6406	11	11
s38584.1	MCNC	6447	11	12
clma	MCNC	8383	16	16
elliptic	MCNC	3849	13	13
pdc	MCNC	4631	21	21
total		–	131	131

Table 1: Benchmark Circuits Data

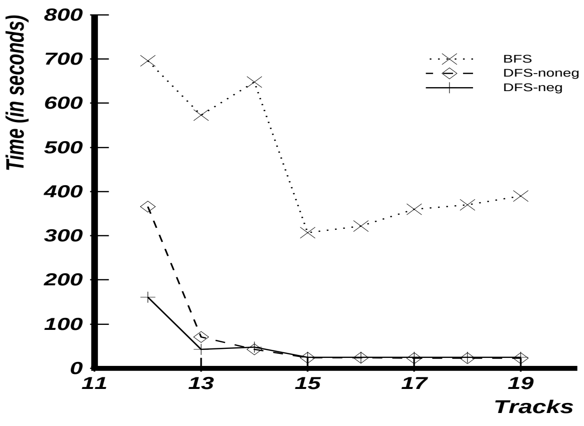

Figure 6: Route Time vs. Track Width - fft16

The first four benchmarks in Table 1 are from the RAW Benchmark Suite [3]. These benchmarks were placed using the VPR placer based on simulated annealing. The remaining six benchmarks and associated placements are from the FPGA Challenge [5]. In only one case, ssp96, was the minimum track width that achieved a successful route, W_{min}, less for breadth-first routing than for depth-first routing with domain negotiation. The fact that the sum of the minimum track widths for breadth-first and depth-first routing was the same indicates the efficiency of the depth-first routing approach.

All run time results were obtained using a 140 MHz UltraSparc 1 with 288Mb of memory. Figure 6 illustrates the importance of domain negotiation in the depth-first routing of array-based architectures. In the non-negotiated case, the lack of domain selection caused many separate paths from the net output pin to input pins on different domains thus leading to the overuse of routing resources. For track widths near the minimum track width, depth-first routes with domain negotiation showed a speedup (as much as 2X) over routes performed without negotiation. In general, the effect of domain negotiation was less as the track widths were increased due to a large increase in possible routing paths for both cases.

	Average Route Time (s)	
	Tracks: W_{min}	Tracks: W_{min}+40%
BFS	709	269
DFS-noneg	647	20
DFS-neg	333	18

Table 2: Average Route Times

Table 2 shows average route times achieved across all designs both at minimum track widths and at track widths with 40% additional tracks. It can be seen that route time decreases to under a minute for increased routing channels for the depth-first case but continues to be several minutes on average for the breadth-first case. This would indicate that if FPGA device manufacturers created devices with the same logic capacity but additional routing resources, depth-first routing could allow for device routing in less than a minute.

6 Conclusion

As FPGAs continue to grow, the need for fast compile solutions to the place and route problem becomes important. In this paper we have investigated one approach, converting a traditionally breadth-first router into a depth-first one by treating the route as an A* search. We have shown that this tradeoff reduces routing run time by over and order of magnitude on average for devices with sufficient routing resources, while maintaining minimum track counts in most cases. The disjoint nature of the array-based FPGA creates the need to order domains for depth-first routing so that routes in domains with a high probability of completion are attempted first. It is shown that this domain negotiation selection process has an important effect on routing by allowing routes to complete in less than one minute even for designs containing thousands of logic blocks.

As an added step, the depth-first router could be integrated iteratively into a macro-based floorplanner to allow for tightly-coupled placer-router interaction.

7 Acknowledgements

The author would like to thank Jordan Swartz and Vaughn Betz from the University of Toronto for discussions regarding this work and for providing the VPR software.

References

[1] *Field-Programmable Gate Arrays Data Book.* Lucent Technologies, 1996.

[2] *The Programmable Logic Data Book.* Xilinx Corporation, 1996.

[3] J. Babb, M. Frank, V. Lee, E. Waingold, and R. Barua. The RAW Benchmark Suite: Computations Structures for General Purpose Computing. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines*, Napa, Ca, Apr. 1997.

[4] V. Betz and J. Rose. Directional Bias and Non-Uniformity in FPGA Global Routing Architectures. In *ICCAD*, San Jose, Ca, 1996.

[5] V. Betz and J. Rose. VPR: A New Packing, Placement, and Routing Tool for FPGA Research. In *Proceedings, Field Programmable Logic, Seventh International Workshop*, Oxford, UK, Sept. 1997.

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

[7] S. Brown, M. Khellah, and Z. Vranesic. Minimizing FPGA Interconnect Delay. *IEEE Design and Test of Computers*, pages 16–23, 1996.

[8] G. W. Clow. A Global Routing Algorithm for General Cells. In *Proceedings, ACM/IEEE 21st Design Automation Conference*, 1984.

[9] C. Lee. An Algorithm for Path Connections and its Applications. *IRE Transactions on Electronic Computers*, Sept. 1961.

[10] G. Lemieux, S. Brown, and D. Vranesic. On Two-Step Routing for FPGAs. In *Proceedings: International Symposium on Physical Design*, Napa, Ca., Apr. 1997.

[11] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1995.

[12] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing, Palo Alto, Ca., 1980.

[13] M. Palczewski. Plane Parallel A Maze Router and its Application to FPGAs. In *Proceedings, ACM/IEEE 29th Design Automation Conference*, 1992.

[14] R. Prim. Shortest Connecting Networks and Some Generalizations. *Bell Syst. Tech. J.*, 1957.

[15] J. Swartz, V. Betz, and J. Rose. A Fast Routability-Driven Router for FPGAs. In *6th International Workshop on Field-Programmable Gate Arrays*, Monterey, Ca, Feb. 1998.