VLSI DESIGN
1996, Vol. 4, No. 4, pp. 275–291
Reprints available directly from the publisher
Photocopying permitted by license only

Segmented Routing for Speed-Performance and Routability in Field-Programmable Gate Arrays

STEPHEN BROWN*, MUHAMMAD KHELLAH, and GUY LEMIEUX

Department of Electrical and Computer Engineering, University of Toronto, Canada

This paper addresses several issues involved for routing in Field-Programmable Gate Arrays (FPGAs) that have both horizontal and vertical routing channels, with wire segments of various lengths. Routing is studied by using CAD routing tools to map a set of benchmark circuits into FPGAs, and measuring the effects that various parameters of the CAD tools have on the implementation of the circuits. A two-stage routing strategy of global followed by detailed routing is used, and the effects of both of these CAD stages are discussed, with emphasis on detailed routing. We present a new detailed routing algorithm designed specifically for the types of routing structures found in the most recent generation of FPGAs, and show that the new algorithm achieves significantly better results than previously published FPGA routers with respect to the speed-performance of implemented circuits.

The experiments presented in this paper address *both* of the key metrics for FPGA routing tools, namely the effective utilization of available interconnect resources in an FPGA, and the speed-performance of implemented circuits. The major contributions of this research include the following: 1) we illustrate the effect of a global router on both area-utilization and speed-performance of implemented circuits, 2) experiments quantify the impact of the detailed router cost functions on area-utilization and speed-performance, 3) we show the effect on circuit implementation of dividing multi-point nets in a circuit being routed into point-to-point connections, and 4) the paper illustrates that CAD routing tools should account for *both* routability and speed-performance at the same time, not just focus on one goal.

Keywords: Field-programmable Gate Arrays, FPGAs, CAD, Routability, Speed-performance

1. INTRODUCTION

Over the past several years, Field-Programmable Gate Arrays (FPGAs) have become widely accepted as an attractive means of implementing moderately large digital circuits in a customized VLSI chip. A number of different styles of FPGAs are commercially available and one of the most important types is the *array-based* architecture, which consists of rows and columns of logic blocks with horizontal routing channels between the rows and vertical channels separating the columns. First introduced by Xilinx, in [1] and later in [2] and [3], variations of the array-based architecture are also found in FPGAs produced by AT&T [4], and QuickLogic [5].

*E-mail: brown@eecg.toronto.edu

Array-based FPGAs are available with very high logic capacities, approaching the equivalent of 15,000 logic gates (a *logic gate* is usually defined as the 4-transistor cell that is the basic building block in some Mask-Programmable Gate Arrays; in simpler terms, it can be thought of as a NAND-gate). With such large devices, the design of the interconnect in the routing channels has a crucial impact on both the percentage of the chip's logic capacity that can be effectively utilized and the speed-performance of circuits implemented in the FPGA. In early array-based FPGAs [1] [2], interconnect comprised mostly short wire segments that spanned the length or width of a single logic block, and longer wire segments were available only by interconnecting the short segments via programmable routing switches. While such architectures allow for efficient utilization of the wire segments in terms of area (since short connections never waste area by using long wire segments), requiring that long connections pass through several routing switches in series severely impairs speed-performance. This follows because routing switches are user-programmable and hence have significant series resistance and parasitic capacitance. To address these issues, recent architectures contain *segmented routing channels* that comprise a mixture of both short and long wire segments. If CAD tools carefully utilize these variable-length segments when implementing circuits, segmented routing channels can greatly enhance speed-performance [6].

It is clear that implementing any non-trivial circuit in a complex FPGA requires sophisticated Computer-Aided Design (CAD) tools. A typical design system [7] [8] [9] would include support for the following CAD steps: initial design entry, logic optimization, technology mapping, placement, and routing. This paper focuses on the final stage of the CAD process, investigating most of the important issues associated with routing for array-based FPGAs. Routing is studied by using CAD routing tools to implement a set of benchmark circuits in FPGAs, and measuring the effects that various parameters of the CAD tools have on the implementation of the circuits. In the experiments, both of the key metrics for routing tools are studied, namely 1) the effective utilization of the available interconnect resources in the FPGA, and 2) the speed-performance of the final result.

The overall routing strategy used is the traditional two-stage approach in which global routing is followed by detailed routing. The *global router* assigns each of the required connections in a circuit to specific routing channels, and then the *detailed router* allocates the FPGA's wire segments and routing switches within the channels to complete the connections. Since global routing for FPGAs is similar to that for other technologies, it is considered only briefly in this paper, but detailed routing, which for FPGAs requires a novel approach, is discussed at length. In fact, we present a new detailed routing algorithm that has been developed specifically for the types of routing architectures found in the most recent generation of array-based FPGAs[1].

The rest of this paper is organized as follows. Section 2 provides background information on the category of FPGA used in this study. Section 3 gives an overview of the CAD tools used for implementing circuits and describes in detail the global and detailed routing algorithms (most of the focus is on detailed routing). Section 4 presents experimental results that explore the effects of the routing tools on both the area-utilization of FPGA routing resources as well as the speed-performance of implemented circuits, and Section 5 summarizes our research contributions.

2. BACKGROUND INFORMATION

This section provides background information in two areas: it describes the model of array-based FPGAs used for this study, and it defines the CAD routing problem for this type of FPGA. Also, previous research on routing algorithms is discussed.

2.1. FPGA Model Used in This Study

The model for FPGAs assumed in this paper is similar to that in other studies on FPGA architecture [6] [10] [11] [12] and CAD algorithms [13] [14]. As il-

lustrated in Figure 1, the FPGA consists of a rectangular array of $N \times M$ logic blocks with both horizontal and vertical routing channels. In terms of commercially available devices, the structure depicted in the figure is most similar to that found in Xilinx FPGAs [1] [2] [3], but it is more general. For the small example in Figure 1, the FPGA has two pins on each side of a logic (L) block and three tracks per channel. For this paper, no assumptions are necessary about the internal details of the logic blocks, except that each block has some number of pins that are connected to the channels by routing switches. The channels comprise two kinds of blocks, called Switch (S) and Connection (C) blocks, described below. The S blocks hold routing switches that can connect one wire segment to another, and the C blocks house the switches that connect the wire segments to the logic block pins. Because of its widespread use, being offered in FPGAs manufactured by Xilinx, Altera [15], and AT&T, this paper assumes that routing switches are pass-transistors controlled by StaticRAM cells[2]. Note that the blocks in Figure 1 are numbered along the left and bottom sides for later reference as a means of describing connections to be routed.

The general nature of an S block is illustrated in Figure 2a. Since wire segments in the routing channels may be of various lengths, some tracks pass

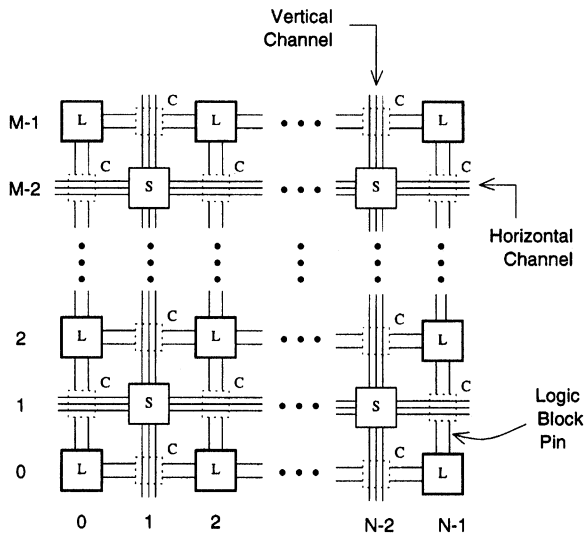


FIGURE 1 An $N \times M$ FPGA.

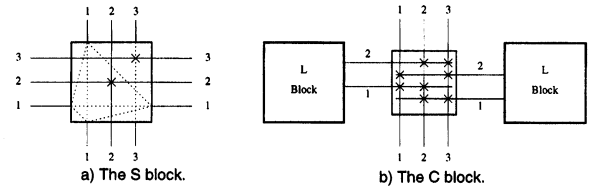


FIGURE 2 Examples of S and C Blocks.

straight through the S block, while other tracks are broken by routing switches. There are two representations for switches in the figure, either as a dotted line for connecting the ends of two wire segments, or as an X for a wire segment that passes straight through the S block. For the example in the figure, the S block switches allow the horizontal tracks numbered 1, 2, and 3 to connect to the vertical tracks with the same numbers. Although Figure 2a provides a specific example, the FPGA model treats the S block as a general four-sided switch block that can be configured in any way.

There are two parameters of the FPGA architecture that determine the layout of routing switches in an S block. The first is the segmentation of the channels; by allowing customizing of the S blocks, the model supports virtually any channel segmentation scheme (for the CAD routing tools described later in this paper, the user can specify channel segmentation by any number of "groups" of tracks that have specific segmentation lengths or a probability distribution of lengths). The second architectural parameter affecting an S block is called its *flexibility* and is set by a parameter, F_s, which defines the number of other wire segments that a wire segment that ends at an S block can connect to. For the example shown in Figure 2a, the wire segment at the top left of the S block can connect to three others and so F_s is 3. Note that F_s alone does not determine the number of routing switches in an S block, since tracks that pass uninterrupted through the block have fewer associated switches.

Figure 2b illustrates a C block. The tracks pass uninterrupted through the C block and can be connected to the logic block pins via a set of switches. The flexibility of a C block, F_c, is defined as the

number of wire segments in the C block that each logic block pin can connect to. For the small example shown in the figure each pin can be connected to 2 vertical tracks, and so F_c is 2 (in a C block, routing switches are drawn as an X). Our FPGA model allows complete customizing of the C block.

The main advantage provided by the FPGA model described above is its generality, which supports a wide range of routing architectures by changing the number of tracks per channel and the contents of the C and S blocks. Since the CAD routing tools are based on this general model, they provide a research vehicle for studying the architecture of FPGAs. Studies based on our earlier CAD tools examined the effects of the F_c and F_s parameters, and can be found in [10] [11] and [12]. A recent architectural study on channel segmentation has been carried out using the tools described in Section 3 of this paper, and is reported in [6].

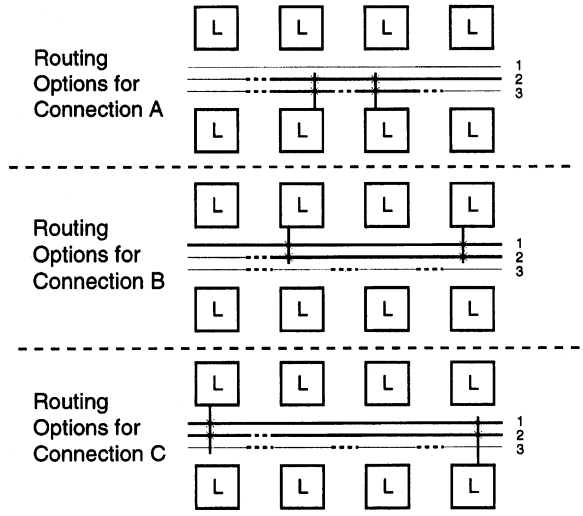


FIGURE 3 An Example of an FPGA Routing Problem.

2.2. The Routing Problem in Array-based FPGAs

Since numerous routing algorithms for VLSI chips have been created over the years, it is prudent to explain why array-based FPGAs with segmented channels represent a new type of routing problem. To begin with, routing in FPGAs with any style of routing architecture can be more difficult than classical detailed routing [7] [8] because the segments available for routing are already in place and connections between segments are possible only where routing switches exist. To illustrate the issues involved, consider the example described below.

Figure 3 shows three views of a section of a routing channel in an array-based FPGA (note that, for clarity, the vertical channels are not shown in the picture). In each view, the figure illustrates the routing options available in this channel for three different connections, called A, B, and C. In the figure, a *wire segment* in the channel is shown as a solid horizontal line, and a wire segment that is usable for a particular connection is highlighted as a bold line. A routing switch that joins two horizontal wire segments is drawn as a dashed line, and a switch that joins a horizontal segment to a logic (**L**) block pin is shown as an X. Finally, logic block *pins* are drawn as vertical lines. As depicted in Figure 3, the routing architecture in this FPGA has three tracks and the routing switches are distributed such that only tracks 2 and 3 can connect the required logic block pins for Connection A, and only tracks 1 and 2 can be used for Connections B and C. The discussion below considers this routing problem, first from the perspective of just completing all three connections, and then also considering the usage of the wire segments according to their lengths.

Assume that a router completes connection A first. If it chooses to route Connection A on track 2, then one of B and C will fail because they both rely on a single remaining option, namely track 1. On the other hand, if the router had chosen track 3 for A, then B could use track 1 and C track 2, or vice-versa. This simple example illustrates that, even when there are only three connections involved, routing decisions made for one connection can unnecessarily block others. Such conflicts for routing resources are the main reason why detailed routing for FPGAs can be more difficult than classical detailed routing.

The above routing solution satisfies the goal of completing all three connections, but only one of the two choices for B and C makes the best use of the available wire segments. Specifically, it is clear from examining the routing channels that Connection B should be assigned to track 2, since the wire segment there exactly matches the connection's length. This also leads to the best solution for Connection C since it requires only one wire segment in track 1 but would need two segments in track 2. Matching the lengths of wire segments to connections is a new problem that does not exist for classical mask-programmed technologies, where there is complete flexibility to create metal wires of any length. While Figure 3 shows only connections within one small routing channel, the problem is much more complex where many connections compete for wire segments and when both horizontal and vertical channels are involved.

A key issue illustrated by the above example is that routing algorithms for FPGAs must consider not only the successful completion of all required connections, but must also account for the number of wire segments allocated *per* connection. The former of these goals is concerned with the *routability*, or *area-performance* of circuits implemented with the routing algorithms, and the latter goal determines the *speed-performance* of circuits.

In terms of previous research, common approaches for detailed routing in other types of devices are not suitable for FPGAs. Classic Maze routing [16] is ineffective because it is inherently sequential and so, when routing one connection, it cannot consider the side-effects on other connections. The example in Figure 3 illustrates why this is important. Channel routers [17] are not appropriate for array-based FPGAs because it is very difficult to subdivide the routing problem into independent channels. Channel routing algorithms are used in [18] and [19] for row-based FPGAs [20] [21]. This is possible for these types of FPGAs because the logic blocks are arranged in rows separated by routing channels and the routing switches are such that each logic block pin can connect to all the wire segments in the channels above and below it and each horizontal wire segment can connect to all the vertical wire segments that cross it. This routing flexibility cannot be assumed for array-based FPGAs (like those from Xilinx), and so it is not clear how channel routing algorithms could be adopted for such devices.

There is a limited number of previous publications concerning routing for array-based FPGAs. The earliest [13] [14] is the predecessor of the detailed routing algorithm described in this paper. The earlier algorithm addressed the problem of considering the side-effects that routing one connection has on others. However, it was intended for routing architectures consisting of short wire segments only and so it did not have the ability to properly utilize wire segments of variable lengths according to the lengths of connections to be routed. While this is not especially important for achieving good routability, it can have significant effects on speed-performance. In Section 4, we contrast the routing results, in terms of the speed-performance of resulting circuits, produced by the router in [14] to the new algorithm described in this paper. Alternative approaches to routing in array-based FPGAs can be found in [22], [23] and [24]. No direct comparison is available to [22] or [23], but [24] shows similar area-performance results to [14] and this is about the same as the area-performance results from the new router described in this paper. [24] describes a multi-point, as opposed to two-point, router and shows that it uses fewer wire segments than the router in [14]; however, the effect of this optimization on speed-performance is not measured.

3. IMPLEMENTATION PROCEDURE

This section describes the CAD tools that are used in this research to implement a set of benchmark circuits in array-based FPGAs. The next subsection provides an overview of the entire CAD system, after which the global and detailed routers are described in greater depth. At the end of this section, we describe the method that it used to measure the speed-

performance of a circuit after it has been implemented by the CAD tools.

3.1. Overview of CAD Tools

To implement the benchmark circuits described later in the paper, the following CAD steps, which would be included in any typical FPGA development system [9], were involved: 1) the benchmark circuits, which were originally targeted for standard cell implementation, were *technology mapped* into FPGA logic cells using the Chortle algorithm [25], 2) the logic cells in the multi-point netlist resulting from technology mapping were *placed* into specific locations in the FPGA using an implementation of the min-cut algorithm [26], 3) finally, the logic cells were interconnected during *routing*. The approach used for routing is the traditional [8] two-stage method of *global* routing followed by *detailed* routing, allowing the separation of two distinct problems: balancing the densities of the routing channels, and assigning specific wire segments to each connection.

The CAD stages preceding routing were performed only once for each benchmark circuit, but routing was performed multiple times, for different parameters of the global and detailed routers. The results after routing were evaluated in two ways: 1) were the routing tools able to successfully complete 100 percent of the required connections for the circuit?, and 2) if all of the connections were successfully routed, what is the speed-performance of the final result? The answer to question 1) is easily obtained from the detailed router, and to answer question 2) we estimate routing delays of signals using the method that will be described in Section 3.4. The following subsections provide more details on the global and detailed routing algorithms.

3.2. The Global Router

Since global routing does not necessarily require detailed knowledge of the contents of the routing channels, it is possible to adapt algorithms from previous technologies for use with FPGAs. The global router employed in this study is an adaptation of the LocusRoute global routing algorithm for standard cells [27]. This global router divides multi-point nets in the circuit being routed into two-point connections (the implications of this step are discussed later in this section) and finds minimum distance paths through the routing channels for each connection. The algorithm's main goal is to distribute the connections among the channels so that the channel densities are balanced. Intuitively, this is a sensible goal for FPGAs, because the capacity of each channel is strictly limited. In addition to balancing the channel usage, the global router can also (optionally) minimize the number of bends that each of the two-point connections incurs [12]. A *bend* occurs at an S block where a connection has to turn to reach its destination. Reducing bends is important because connections are better able to utilize longer wire segments if they travel further along a single channel before turning. The results in Section 4 will show that bend reduction can have a significant effect on the speed-performance of routed circuits.

An example of the output of the global router, which is called a *coarse graph* [8], G, for a single connection routed in a very small FPGA is illustrated on the left-most side of Figure 4. The vertices and edges in G are identified by the coordinates shown in the figure for the FPGA and define the sequence of channels that the global router has chosen to connect the logic block at location (0,4) to the one at (4,0).

Since the global router splits all multi-point nets into two-point connections, the coarse graphs always have a fan-out of one. However, some connections

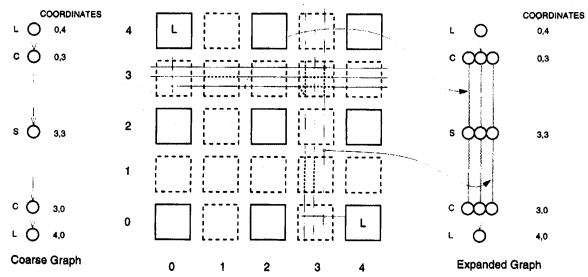


FIGURE 4 Expanded Graph, D, Showing the Alternative Detailed Routes for G.

that are part of the same net might overlap within a routing channel, and this could lead to wasted wire segments after the entire circuit is routed. The results in Section 4 will show that by decomposing multi-point nets into two point connections the global router can adversely affect speed-performance. In Section 3.3, we describe a method that can be used during detailed routing to "re-construct" the multi-point nets that are broken by the global router.

3.3. The Detailed Router

A new detailed routing algorithm has been developed for this study and is called **SEGA**, for SEGment Allocator. Designed specifically for array-based FPGAs with segmented channels, **SEGA** includes novel features that allow it to produce a routing result that is optimized *either* for the best achievable area-utilization of the FPGA's routing resources, or the best achievable speed-performance of the implemented circuit. **SEGA** is parameterized to support any FPGA architecture that fits the general array-based model that was illustrated in Figure 1. In terms of its overall organization, **SEGA** is similar to a previously published detailed router described in [14]. However, the new algorithm is fundamentally different from its predecessor in the treatment of wire segments according to their lengths. By properly accounting for the lengths of wire segments during all stages of routing, **SEGA** is able to achieve a significantly better result (as much as 25%) than the earlier algorithm with respect to the speed-performance of implemented circuits.

To route a circuit, **SEGA** first creates a representation of the FPGA, from a set of user-specified parameters, and then reads the output from the global router. A coarse graph is created in an internal data-structure for each required connection. Detailed routing then proceeds in two main phases: in phase 1, the router examines the wire segments and routing switches present in the FPGA and enumerates all of the alternatives for the detailed route of each coarse graph. Then, in phase 2, specific routing decisions are made for each connection. The decisions taken in phase 2 are driven by cost functions (to be described in Section 3.3.2) that reflect either the routing delay associated with each choice, or the effect that each alternative would have on the routability of the overall circuit.

3.3.1. Phase 1: Enumerating the Detailed Routes

During phase 1, **SEGA** enumerates all of the detailed routes that are available in the FPGA to implement each global route. The alternative detailed routes for each coarse graph, G, are recorded in an *expanded* graph, called D. As illustrated in Figure 4, each edge in D represents specific wire segments (one or more) that can be used to implement the corresponding edge in G. As the figure shows, D has the same vertices as G, but there is one instance of each vertex for each *path* in the FPGA that leads from the root vertex to the leaf vertex. The edges of D are drawn as shaded lines to indicate that they are not simple edges. Each edge, e, in D may imply the use of multiple wire segments, in which case multiple shaded lines are shown. It is important to realize that the length of a wire segment referenced in e is not necessarily the same as the length of the corresponding edge in G, since a wire segment may be either longer or shorter than the edge itself. Each e has associated with it one or more labels, one for each wire segment that it references. The labels identify the corresponding wire segments in the FPGA, examples of which are indicated by the two curved lines pointing from wire segments in the FPGA to edges in D.[3]

3.3.2. Phase 2: Path Selection

After phase 1, each D may contain a number of alternative paths. **SEGA** places all of the expanded graphs into a single *connection-list*. Based on cost functions (defined shortly), the router then selects a path to define the detailed route for each connection in the list. Because **SEGA** expands all the coarse graphs before making any routing decisions, when optimizing for routability it can consider the side effects that a deci-

sion made for one connection has on others. For reasons given in Section 2, this is important in FPGAs. Alternatively, if speed-performance is the primary goal the router can base its decisions on the lengths of the wire segments represented in D as they compare to the lengths of the edges in G. Phase 2 proceeds as follows (the basis for *sorting* the connection-list and the method for evaluating the *cost* of a path will be defined shortly):

put all connections (expanded graphs) into a single *connection-list*
while the connection-list is not empty **do** {
  sort the *connection-list*; **select** the connection at the head of the list
  route the selected connection, using the path with lowest **cost**
  mark the connection as routed, and remove all paths in this connection from the *connection-list*
  find all paths that would conflict with the selected path (i.e. all paths that are part of *different* nets but reference the wire segments just allocated to the selected path) and remove them as alternatives for the corresponding connections. If a connection loses its last remaining path, that connection is deemed unroutable[4]
  update the cost of all affected paths
}

Two key details are not explained in the above pseudo-code: the metric used to *sort* the connection-list, and the definition of the cost function that assesses the *cost* of a path. In both cases, this depends on whether **SEGA** is being used to 1) optimize for area or 2) optimize for speed, as follows. For area optimization, **SEGA** first sorts (note: in this paper, *sort* means to scan through the list from head to tail and make a selection based on some metric) the connections according to the number of possible alternatives (number of paths in each expanded graph), so that connections that have fewer possible routes will be given priority. Once a connection has been selected by this sorting procedure, **SEGA** uses a cost function called *Demand*(p), described below, to evaluate the cost of each available path, p, and chooses the path with the minimum cost (if more than one connection ties for having the fewest alternatives after sorting, **SEGA** evaluates the costs of the paths in all of these connections). In speed-performance mode, **SEGA** first sorts the connections according to their lengths[5] (so as to prioritize long connections and enable them to take advantage of long wire segments), and then makes the path selection based on a cost function called *Delay*(p). The cost functions *Demand*(p) and *Delay*(p) will now be described.

3.3.2.1. Routability (or Area)-based Cost Function

The area-based *Demand*(p) cost function was originally defined in an earlier router for array-based FPGAs, called CGE [13] [14]. Its purpose is to allow the router to select a path for one connection such that it has the least negative effect on other connections from a routability point of view. For **SEGA**, this cost function engenders successful routing of 100% of the connections in a circuit using a minimal number of tracks per channel. *Demand*(p) is defined by a summation that calculates the 'demand' among the connections in a circuit for each wire segment associated with p. To calculate the demand for an individual wire segment, w, **SEGA** counts the number of instances of w that are in expanded graphs for other nets. However, some instances are less likely to be selected when the corresponding connection is routed because there are alternative wire segments in parallel with w. Thus, if a path p contains a wire segment w that has j other instances (w_1, w_2,...w_j), then *Demand*(w) is given by:

$$Demand(w) = \sum_{i=1}^{j} \frac{1}{alt\,(w_i)} \qquad (1)$$

where $alt\,(w_i)$ is the number of wire segments in parallel with w_i. *Demand*(p) is then the summation of *Demand*(w) for all wire segments in p.

3.3.2.2. Speed-Performance-based Cost Function

The purpose of the *Delay*(*p*) cost function is to allow **SEGA** to select whichever path represents the best choice in terms of speed-performance. Different paths may incur larger or smaller delays because they might have different numbers of wire segments or their wire segments may be of different lengths. For the purpose of comparison, two methods for evaluating *Delay*(*p*) can be used in **SEGA**. The first method considers the number of wire segments assigned to each connection and the lengths of those segments, while the second method employs an analytical model to estimate real routing delays (the analytic model is described in Section 3.4). When measuring *Delay*(p) based on the number and lengths of wire segments in a path, Delay(p) is calculated as follows:

$$Delay(p) = c_1 \times NumSeg\ (p) + c_2 \times SegLen\ (p) \tag{2}$$

where *NumSeg*(*p*) is similar to the cost function defined in [18] and [19] and its purpose is to minimize the *number* of wire segments assigned to a connection. The cost terms are normalized so that they range from 0 to 1, and thus *NumSeg*(*p*) is defined as the quotient of "the actual number of segments in *p* minus the minimum possible[6]" divided by "the actual number of segments in *p*". *SegLen*(*p*) is similar to the function used in [19]. Its purpose is to minimize the wastage due to assigning long wire segments to short connections. Thus, *SegLen*(*p*) is defined as the quotient of "the total wasted length of the wire segments in *p*[7]" divided by "the total length of wire segments in *p*". The c_1 and c_2 factors in Equation (2) are binary weights used to turn either term on or off.

Equation (2) provides one way of measuring *Delay*(*p*), using cost functions defined in previous publications [18] [19]. A different approach to assessing the speed-performance of paths is to use an analytic model to estimate real propagation delays, rather than counting segments and segment lengths. When measuring *Delay*(*p*) based on real propagation delays, *Delay*(*p*) is defined as:

$$Delay(p) = \frac{ActualDelay(p) - MinimumDelay}{ActualDelay(p)} \tag{3}$$

where *ActualDelay*(*p*) represents the total routing delay that would be seen by the corresponding connection if routed using path *p*. *MinimumDelay* is the theoretical minimum routing delay for the connection, if it were routed using the fastest possible routing resources in the FPGA. Both *ActualDelay*(*p*) and *MinimumDelay* are calculated by using the mathematical model described in Section 3.4.

3.3.2.3. Modifying SEGA to Route Multipoint Nets

SEGA produces good results for both area and speed-performance with the above algorithm and cost functions. However, some improvements should be possible if the algorithm considered which connections are part of multi-point nets, rather than just routing two-point connections. We have performed extensive experiments to investigate this issue and have found that it is not particularly important when optimizing for routability, because the *Demand*(*p*) cost function tends to merge two-point connections that are part of the same net if they overlap. However, for speed-performance accounting for multi-point nets can have a significant effect, due to the extra RC-load that is added to these nets when their constituent two-point connections overlap but do not share wire segments. The key issue is that it is advantageous for **SEGA** to "share" wiring resources among connections that are electrically part of the same multi-point net. To address this issue, the following is a modified version of **SEGA** that can be used instead of the above algorithm when optimizing for speed-performance:

Phase 2: Path Selection

put all connections (expanded graphs) into a single *connection-list*
group connections in the *connection-list* by nets
while the connection-list is not empty **do** {
    sort the *connection-list* according to net

```
        length[8]; select the longest net
    sort connections in the selected net according to their lengths; select the longest connection
    while there are unrouted connections for the current net do {
        if this is the first connection routed for the net then
            route the connection with the fastest available path
        else {
            route the connection using the path that has the maximum number of shared segments with the already routed part of the net
            if such a path is not available then
                route the connection with the fastest available path
        }
        mark the connection as routed, and remove all paths in this connection from the connection-list
        find all paths that would conflict with the selected path (i.e. all paths that are part of different nets but reference the wire segments just allocated to the selected path) and remove them as alternatives for the corresponding connections. If a connection loses its last remaining path, that connection is deemed unroutable
        update the cost of all affected paths
    }
    mark the net as being routed
}
```

The key idea behind the above pseudo-code is that it tries to maximize the sharing of wire segments among connections that are part of the same net. Referring to the code, for this scheme **SEGA** routes all of the connections in a particular net before moving on to another net. The nets are sorted by length, so that long nets can take advantage of long wire segments. Once a net has been selected, its individual connections are further processed by length so that long connections have the most opportunity to use long wire segments. Referring to the inner-most "**while**" loop in the code, the first connection routed for each net is mapped to its fastest available path according to Equation (2) or (3). Any subsequent connections, however, will be routed using the path that has the maximum number of shared segments with the already routed part of the net, if such a path exists. Otherwise, the graph will be routed using its fastest possible path. In Section 4, we will show that since the above algorithm tends to minimize resistive and capacitive loading on nets, it results in significant speed-performance improvement. Finally, experiments have shown that **SEGA** requires the same amount of time (about 40 msecs per connection on a SUN/4 model ELC) whether routing by nets or two-point connections.

3.3.3. Summary of SEGA Cost Functions

The preceding sub-sections have described several cost functions that are available in the **SEGA** detailed router. In Table I, these functions are summarized and assigned a name for reference later, in Section 4.

Table I **SEGA**'s Cost Functions.

Cost Function	Description
Area	optimize for routability only
Seg_Len	minimize lengths of wire segments used
Num_Seg	minimize the number of wire segments used
Seg_Len + Num_Seg	combination of the above two cost functions
Analytic_Model	use an analytic model to find delays
Net_Routing	optimize for speed, but also focus on re-use of wire segments for connections on same net

3.4. Delay Model for Estimating Speed-Performance of Routed Circuits

For this research, there are two purposes for which it is necessary to measure the propagation delays of routed connections in FPGAs. Firstly, and most importantly, once a circuit has been fully implemented we need to measure the speed-performance of the final result in order to assess the quality of the solution produced by the CAD tools. Secondly, when using the **Analytic_Model** cost function, the detailed router calculates actual routing delays of alternative paths in order to make routing decisions. For both of these situations, we use an efficient analytic modelling technique to quickly and accurately estimate signal propagation delays.

To estimate routing delays in FPGAs, an adaptation of the analytic modelling technique presented in [28] is used, in which MOS transistors are modelled as constant RC-elements. Although the original publication [28] stated that the model is not applicable for pass-transistors, in [29] we show that by carefully choosing values of resistance and capacitance it is possible to use it for that purpose. The input to the analytic model is an *RC-tree*, in which resistors represent routing switches that signals pass through in series, and capacitors correspond to parasitic capacitance due to both routing switches and wire segments. As output, the model produces an estimate for the delay from the source node of the network to each of the sink nodes, where the source-to-sink delay is defined as the time it takes for an ideal step input at the source to reach half its value at the sink[9]. In Figure 5, two examples of RC-trees for detailed routes, called "path #1" and "path #3", connecting from the L block at the lower left of the figure to the block at the upper right are shown. Referring to the FPGA channels in the figure, note that each routing switch in series with a detailed route (path) contributes both a resistor and capacitor to the RC-tree, and wire segments in the path as well as routing switches that "hang off" the wire segments add capacitance. Figure 5 also shows that a source resistance and capacitance, as well as a load capacitance are included for each net.

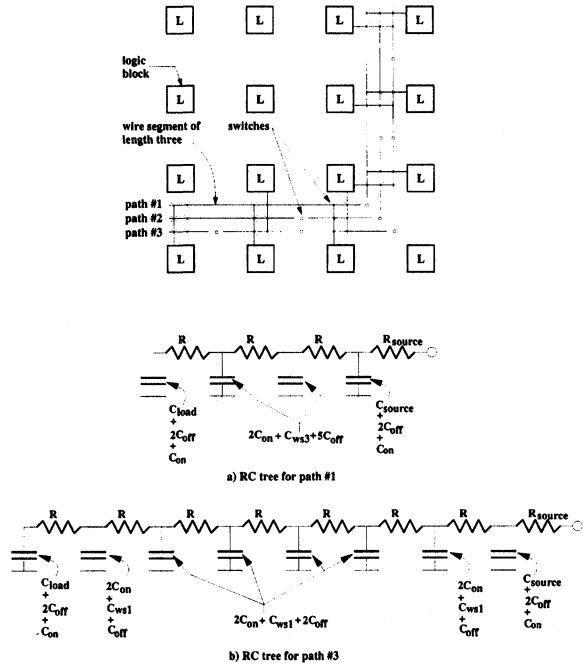


FIGURE 5 Examples of RC-trees for Connections Routed in FPGAs.

For the results presented later in this paper, R and C are calculated assuming a 0.8-micron BiCMOS process. The particular values used can be found in [29], and are summarized as follows: R for an "ON" switch is 915 ohms, C for an "ON" switch is 25 fF, C for an "OFF" switch is 13 fF, and C for a wire segment is 3 fF per unit length. Using these parameters, the speed-performance of individual nets can be calculated directly by the analytic model. The delay of a net is defined as the largest delay from the net's source to any of its sinks. We then define the speed-performance of an entire circuit implemented in an FPGA as the *average of the net delays in the circuit.*

4. EXPERIMENTAL RESULTS

This section presents experimental results that illustrate the effects of various parameters of both the global and detailed routers on the implementation of circuits. Following the procedure outlined in Section 3, the experiments are based on a set of benchmark cir-

cuits summarized in Table II. The table shows the name of each circuit and its size in terms of the number of logic blocks, number of nets, and number of two-point connections. All of the circuits (except the largest one) are from the MCNC benchmark suite.

4.1. Effect of the Global Router on Implementation of Circuits

Recall from Section 3 that besides balancing channel densities, the global router can also minimize the number of bends that connections pass through. In this section, we will show that this is an important goal that can affect circuit implementation.

With a reduced number of bends, connections traverse longer distances in a routing channel before turning at an S block. To quantify this effect, we routed each benchmark circuit twice: once with the bend reduction feature of the global router turned off, and then with bend reduction turned on. For each routed circuit, we measured the lengths of the straight sections of connections, called *section length*. Table III gives the average section length for the connections in each benchmark circuit and shows that the average length is 22% greater when bend reduction is turned on.

To evaluate the effect of bend reduction on area utilization, we used **SEGA** to perform detailed routing of each global router solution using the Area (see Table I) cost function[10]. The purpose of the experiment was to determine the minimum number of tracks per channel needed to successfully route the circuits with and without bend reduction for a range of different channel segmentations in the FPGA. Thus, for each circuit, the methodology used was to set the number of tracks per channel, W, in the FPGA to a small value (equal to the maximum channel density after global routing) and attempt detailed routing with **SEGA**. As long as detailed routing failed, W was incremented by one until eventually 100% of the connections in the circuit were routed. This was performed for different segmentation lengths in the FPGA ranging from 1 to 8. In each case, all tracks had the same segment lengths. The results are shown in Figure 6, in which the horizontal axis represents segment length and the vertical axis shows the number of tracks needed to route the circuits, on average, above the channel densities.

Referring to Figure 6, for all segment lengths the bend reduced circuits result in fewer required tracks per channel for the detailed router. Also, as segment length increases the two curves diverge. This makes intuitive sense, since connections in the bend-reduced circuits have longer straight sections and so they waste less area as the segment length increases. This experiment shows that from an area perspective it is a clear advantage to reduce the number of bends if the FPGA's channels are segmented.

Having observed the effect of bend reduction on area utilization, we now wish to study the effect on routing delays. For this experiment, to ensure that 100% of the connections in each circuit can be com-

Table II Characteristics of Benchmark Circuits

Circuit Name	# of Logic Blocks	# of Multi-point Nets	# of Two-Point Connections
9symml	72	79	259
too_large	156	186	519
apex7	80	126	300
example2	120	205	444
vda	210	225	722
alu2	143	153	511
alu4	255	256	851
term1	56	88	202
C1355	110	145	360
C499	110	145	360
C880	120	174	427
k2	360	404	1256
z03D4	586	608	2135

Table III The Average Section Length for all Circuits.

Circuit Name	Section Length with Bend Reduction Off	Section Length with Bend Reduction On
9symml	1.5	1.9
too_large	1.8	2.3
apex7	1.7	2.1
example2	1.9	2.5
vda	2.0	2.6
alu2	1.7	2.2
alu4	1.9	2.5
term1	1.6	2.0
C1355	1.8	2.4
C499	1.7	2.3
C880	1.8	2.4
k2	2.2	3.0
z03D4	1.8	2.3
Average	1.8	2.3

pleted by the detailed router, the number of tracks per channel is set to a high value (30). Rather than using a single segment length for all tracks as was done for the previous experiment, in this case each channel contains a mixture of tracks with segments of length 1, 2, or 3. Over many combinations of channel segmentation, the benefits of bend reduction were assessed by detailed routing the global routing solutions both with and without bend reduction. Also, the experiments were repeated using all of the different cost functions available in the detailed router. Table IV provides a summary of the average results for all segmentation schemes for each detailed router cost function. Each number in the table represents the average net delay for the circuits, in nsecs.

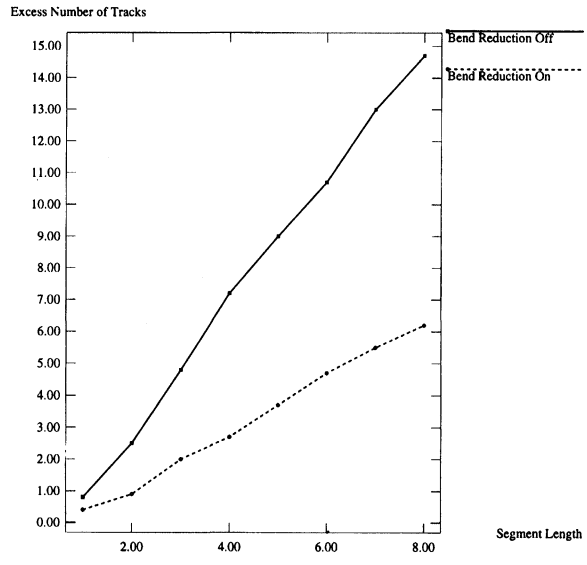


FIGURE 6 Effect of Bend-reduction on Area Performance.

Referring to Table IV, enabling bend reduction clearly produces better speed-performance results, since for all detailed router cost functions the bend-reduced circuits provided better speed-performance. This result occurs because connections in the bend-reduced circuits have longer straight sections, and this allows the detailed router to make use of longer wire segments. Lower propagation delays result because connections routed with a smaller number of long wires need to pass through fewer switches than if they were routed with a larger number of short wire segments.

From the experiments presented in this section, it is clear that bend reduction is a good strategy that can be used to improve both the speed- and area-performance of implemented circuits. Unless otherwise stated, for the rest of the experiments presented in this paper all circuits are bend-reduced.

4.2. Effect of the Detailed Router on Implementation of Circuits

In this section, our aim is to determine which detailed router cost function produces the "best" speed-performance results and which function produces the "best" area-performance results. From the data al-

Table IV Effect of CAD Routing Tool Cost Functions on Routing Delays.

SEGA Cost Function	Global Router without Bend Reduction	Global Router with Bend Reduction
Area	16.7	12.8
Seg_Len	19.0	16.9
Num_Seg	14.9	11.9
Seg_Len + Num_Seg	16.4	13.3
Analytic_Model	14.8	11.9
Net_Routing	13.0	10.1

ready presented in Table IV, it is clear that the detailed router cost function has a significant effect on speed-performance. Referring to the table, the various cost functions in **SEGA** yield different average routing delays. The Area cost function shows that focusing only on routability gives less than minimum routing delays, as would be expected. The **Seg_Len** row indicates that very poor speed-performance results if the router considers only the lengths of wire segments. The intent of this function is to prevent the assignment of long wires to short connections to minimize capacitive loading, but comparison to the **Num_Seg** row shows this to be a poor strategy. Minimizing the number of segments that connections pass through yields among the lowest delays; this seems to be the most important goal since combining it with **Seg_Len** worsens the results. Since for the **Analytic_Model SEGA** calculates accurate estimates of real delays, comparing **Num_Seg** to **Analytic _Model** shows that the simple cost function that counts the number of switches traversed by a connection is a good approach.

Finally, comparing the bottom row in Table IV with the other rows shows that considering multi-point nets instead of just two-point connections has positive effects on speed-performance. This occurs because when multi-point nets are ignored, the router may use more wire segments and switches than is actually needed where two-point connections on the same net overlap. This results in an increase in parasitic capacitance seen by the net and adds to its propagation delays. For **Net_Routing**, **SEGA** tries to reassemble multi-point nets by focusing on not only speed-performance (using **Analytic_Model**), but also on re-using wire segments for multiple connections that are part of the same net. The results in Table IV show that **Net_Routing** is important because it achieves the "best" speed-performance results.

Recall that it was mentioned earlier that **SEGA** is an enhanced version of an earlier FPGA router described in [14]. Since the earlier algorithm used ostensibly the same cost function as **SEGA** when optimizing for routability, the **Area** cost function in **SEGA** achieves approximately the same results as that in the earlier algorithm. However, the router in [14] did not have the ability to optimize for speed, so a comparison between **SEGA**'s speed-performance optimization and that of its predecessor can be made by contrasting the **Area** cost function result in Table IV with the **Net_Routing** result. The data shows that **SEGA** achieves about a 25 percent improvement in speed-performance over the earlier algorithm.

Table IV gives only average results over a wide range of different channel segmentations. To provide a more detailed view, Table V shows the performance of the detailed router cost functions for specific channel segmentation schemes. In the table, the horizontal axis represents the percentage of tracks in the FPGA that are of length 3, the vertical axis is percentage of length 2, and the remaining tracks are of length 1. Each entry in the table represents the average net delay produced by a particular segmentation scheme, and two of the detailed router cost functions are represented: the shaded columns show the speed-performance achieved by the **Net_Routing** cost function, and the unshaded columns represent the **Area** cost function. Comparing the shaded and unshaded columns, it is apparent that speed-performance is significantly affected by **SEGA**'s cost functions for all segmentation schemes.

Table V Routing Delays of Segmentation Schemes.

100	9.2	8.3																		
90	9.5	8.4	9.0	8.2																
80	9.9	8.4	9.8	8.1	8.9	8.0														
70	10.2	8.6	9.9	8.3	9.2	8.0	8.8	7.9												
60	10.5	9.0	10.0	8.5	9.6	8.2	9.3	7.9	8.7	7.7										
50	10.9	9.5	10.3	8.9	9.8	8.4	9.5	7.9	8.9	7.7	8.6	7.6								
40	11.6	10.3	10.9	9.5	10.3	8.7	9.8	8.2	9.1	7.9	8.9	7.5	8.3	7.5						
30	12.1	11.1	11.4	10.1	10.8	9.4	10.3	8.7	9.7	8.2	8.9	7.7	8.8	7.5	8.2	7.4				
20	12.7	11.9	12.0	11.0	11.3	10.2	10.7	9.3	10.0	8.6	9.4	8.0	9.2	7.7	8.7	7.4	8.3	7.5		
10	13.5	12.4	12.6	11.7	11.9	10.9	11.1	10.1	10.6	9.2	9.8	8.3	9.5	7.9	9.1	7.6	9.0	7.4	8.2	7.5
0	13.9	12.8	13.3	12.5	12.5	11.8	11.6	10.8	10.9	9.9	10.3	9.0	9.9	8.4	9.4	7.9	9.0	7.5	8.5	7.4	8.1	7.5
	0		**10**		**20**		**30**		**40**		**50**		**60**		**70**		**80**		**90**		**100**	

Tables IV and V show that the detailed router cost function can significantly affect speed-performance, but they do not consider the effects on area-performance. To illustrate the effects of **SEGA**'s cost functions on area, Figure 7 compares the area requirements of the routability-oriented **Area**, and speed-oriented **Net_Routing** cost functions[11]. To obtain the results shown in the figure, we used single length segments in the FPGA following the same method that was described with respect to Figure 6. Figure 7 shows that although the **Net_Routing** cost function has the "best" speed-performance results (Table IV), it produces the poorest area-performance. On the other hand, though the **Area** function achieves less than optimal routing delays (Table IV), it has the best area-performance results.

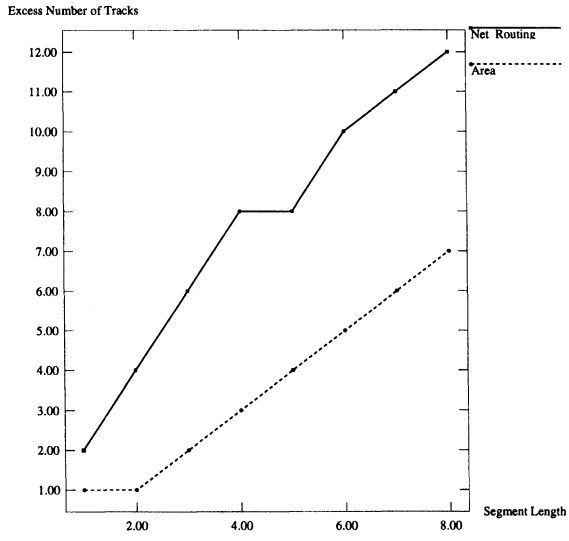


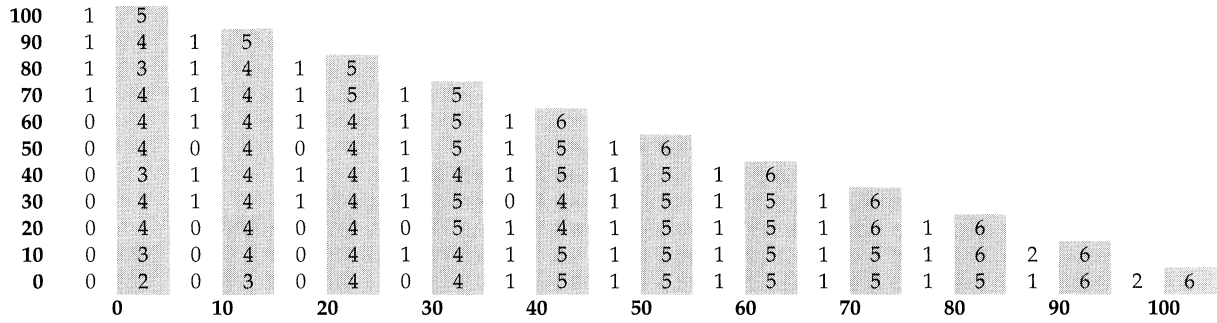
FIGURE 7 Effect of **SEGA**'s Cost Functions on **Area**.

The above area-performance experiment was repeated with channels having combinations of segments of length 1, 2, and 3 and the results appear in Table VI, which lists the number of excess tracks above channel density for the same segmentation schemes in Table V. Consider first only the numbers in the columns shaded grey in Table VI, which provide the excess tracks for the same **SEGA** cost functions used in Table 5: the shaded columns correspond to **Net_Routing**, and the unshaded columns are the **Area** cost function results. Referring to the table, as many as 6 extra tracks are needed for **Net_Routing**; for the benchmark circuits, this corresponds to a significant increase in tracks of about 30 percent. Now consider the unshaded columns in Table VI, which show that for the **Area** cost function at most 2 extra tracks are required. An intuitive conclusion from these results suggests that a "good" CAD routing tool should consider both speed-performance and area utilization, not just focus on one goal. This could be accomplished in practice by having the router use a speed-performance cost function for nets identified as being time-critical, and use area optimization for other nets.

5. CONCLUDING REMARKS

This paper has investigated most of the important issues associated with routing for array-based FPGAs with segmented routing channels. Experiments presented show that the global router can significantly affect speed-performance, depending on whether it selects global routes that traverse short distances

Table VI Area Penalty for Segmentation Schemes.



through multiple channels, or travel longer distances straight along a single channel. We have presented a new detailed routing algorithm for array-based FPGAs and have shown that it achieves much greater speed-performance of circuits than previously published routers. Also, it has been shown that the detailed router cost function can greatly affect both speed- and area-performance of implemented circuits. Finally, routing tools for FPGAs should consider both routability and speed-performance, not just focus on one goal.

References

[1] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo and S. L. Sze, "A User Programmable Reconfigurable Gate Array," *Proc. 1986 Custom Integrated Circuits Conference*, May 1986, pp. 233–235.

[2] H. Hsieh, K. Duong, J. Ja, R. Kanazawa, L. Ngo, L. Tinkey, W. Carter and R. Freeman, "A Second Generation User-Programmable Gate Array," *Proc. 1987 Custom Integrated Circuits Conference*, May 1987, pp. 515–521.

[3] H. Hsieh, W. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey and R. Kanazawa, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *Proc. 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.2.1–31.2.7.

[4] AT&T Microelectronics, "Optimized Reconfigurable Cell Array (OA) Series Field-Programmable Gate Arrays," *Advanced Data Sheet*, February 1993.

[5] Quicklogic, "An Introduction to Quicklogic's pASIC Devices and SpDE Development Environment," *Data Sheet from Quicklogic*, April 1991.

[6] M. Khellah, S. Brown, and Z. Vranesic, "Minimizing Interconnection Delays in Array-based FPGAs," *Proc. 1994 Custom Integrated Circuits Conference*, San Diego, CA, May 1994.

[7] J. Soukup, "Circuit Layout," *Proc. of the IEEE*, Vol. 69, No. 10, pp. 1281–1304, October 1981.

[8] Chapter 5 of "Physical Design Automation of VLSI Systems," B. Preas and M. Lorenzetti, Ed., Benjamin/Cummings.

[9] Stephen D. Brown, Robert J. Francis, Jonathan Rose and Zvonko G. Vranesic, "Field-Programmable Gate Arrays," Kluwer Academic Publishers, 222 pages, 1992.

[10] J. Rose and S. Brown, "The Effect of Switch Box Flexibility on Routability of Field-Programmable Gate Arrays," *Proc. 1990 Custom Integrated Circuits Conference*, pp. 27.5.1–27.5.4, May 1990.

[11] J. Rose and S. Brown, "Flexibility of Interconnection Structures in Field-Programmable Gate Arrays," *IEEE Journal of Solid State Circuits*, Vol. 26 No. 3, pp. 277–282, March 1991.

[12] B. Tseng, J. Rose and S. Brown, "Using Architectural and CAD Interactions to Improve FPGA Routing Architectures," *First International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, pp. 3–8, February 1992.

[13] S. Brown, J. Rose and Z. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," *Proc. IEEE International Conference on Computer Aided Design*, pp. 382–385, Nov. 1990.

[14] S. Brown, J. Rose and Z. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 11, No. 5, pp. 620-628, May 1992.

[15] ALTERA Corporation, "FLEX Programmable Logic," *Product Information Bulletin*, September 1992.

[16] C. Lee, "An algorithm for path connections and its applications," *IEEE Transactions on Electronic Computers*, VEC-10, pp. 346–365, Sept. 1961.

[17] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 8th Design Automation Conference, pp. 155–163, 1971.*

[18] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented Channel Routing," *Proc. 27th Design Automation Conference*, pp. 567–572, June 1990.

[19] K. Roy and M. Mehendale, "Optimization of Channel Segmentation for Channelled Architecture FPGAs," *Proc. 1992 Custom Integrated Circuits Conference*, pp. 4.4.1–4.4.4., May 1992.

[20] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, "An Architecture for Electrically Configurable Gate Arrays," *IEEE Journal of Solid State Circuits*, Vol. 24, No. 2, April 1989, pp. 394–398.

[21] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, S. Kap-

tanoglu, K. Dharmarajan, L. Hutchings, S. Ku, P. McGibney, J. McGowan, A. Samie, K. Shaw, N. Stiawalt, T. Whitney, T. Wong, W. Wong and B. Wu, "An FPGA Family Optimized for High Densities and Reduced Routing Delay," *Proc. 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.5.1–31.5.4.

[22] Mikael Palczewski, "Plane Parallel A* Maze Router," *29th ACM/IEEE DAC*, pp. 691–697, June 1992.

[23] Jon Frankle, "Iterative and Adaptive Slack Allocation for Performance-driven Layout and FPGA Routing," *29th ACM/IEEE DAC, pp. 536–542, June 1992.*

[24] Amit Chowdhary and Dinesh Bhatia, "Detailed Routing of Multi-Terminal Nets in FPGAs," *Proc. Intl. Conf. on VLSI DESIGN*, Calcutta, India, Jan. 1994, IEEE Computer Society Press.

[25] R. Francis, J. Rose and Z.Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th DAC*, pp. 227–223, June 1991.

[26] M. Breuer, "Min-Cut Placement," *Journal of Design Automation and Fault Tolerant Computing*, pp. 343–362, Oct. 1977.

[27] J. Rose, "Parallel Global Routing for Standard Cells," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 10, pp. 1085–1095, Oct. 1990.

[28] J. Rubinstein, P. Penfield and M. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-2, No. 3, July. 1983.

[29] M. Khellah, S. Brown, and Z. Vranesic, "Modelling Routing Delays in SRAM-based FPGAs," *Proc. 1993 CCVLSI*, Banff, Canada, pp. 6B.13–6B.18, Nov. 1993.

Endnotes

1 The detailed router is called **SEGA**, for SEGment Allocator, and is available via the world-wide web at http://www.eecy.toronto.edu/^lemieux/sega or via anonymous ftp from ftp.eecg.toronto.edu in pub/software/SEGA. **SEGA** is written in ANSI-C.

2 Although this assumption does not significantly impact the CAD routing tools, it does affect the speed-performance of implemented circuits, and it dictates the method used to measure speed-performance (Section 3.4 describes our method for measuring speed-performance).

3 The graph expansion procedure is similar to that described in [14], except that in [14] all wire segments are assumed to be of length 1. Explicitly recording the lengths of wire segments allows **SEGA** to later make routing decisions that result in much greater speed-performance of the final result.

4 It would be desirable for the router to have some means of trying other alternative solutions when a connection fails to route. For example, the router could perform another iteration on the problem, trying different combinations of the cost function terms (described shortly) for the channels that contain unrouted connections.

5 The length of a connection is defined as the number of logic (L) blocks it spans.

6 The minimum possible is the number of edges (not including the two L block pins) in the coarse graph.

7 This corresponds to the total length of wire segments in p minus the total length of the edges in the coarse graph.

8 The length of a net is defined as the summation of the lengths of the two-point connections in the net.

9 Since we will assume an NMOS pass transistor switch, we measure the rising time of signal rather than its falling time because the former is the "worst case".

10 Similar *relative* performance results were obtained for **SEGA**'s other cost functions.

11 Other cost functions produced results that fall between these two extremes.

Authors' Biographies

Stephen Brown received the Ph.D. degree from the University of Toronto in 1992. He is currently an Assistant Professor in the Dept. of Electrical and Computer Engineering at the University of Toronto. His research interests include architecture, CAD tools and mathematical modelling for Field-Programmable Gate Arrays, as well as architecture and systems software for shared-memory multiprocessors. He is a member of the IEEE.

Muhammad Khellah received the M.A.Sc. degree from the Department of Electrical and Computer Engineering at the University of Toronto, in 1994. His research interests include architecture and CAD for FPGAs.

Guy Lemieux received the M.A.Sc. degree in 1996 from the University of Toronto, where he is currently a Ph.D. candidate. During 1991-1992, he worked for IBM Canada in the Image Systems Laboratory. His research interests include FPGAs, multiprocessors, and computer architecture. He is a student member of the IEEE.