*VLSI DESIGN*
1996, Vol. 5, No. 1, pp. 11–21
Reprints available directly from the publisher
Photocopying permitted by license only



# Greedy Segmented Channel Router

DINESH BHATIA[a,*] and V. SHANKAR[b]

[a]*Design Automation Laboratory, Department of Electrical and Computer Engineering & Computer Science, P.O. Box 210030, University of Cincinnati, Cincinnati, OH 45221-0030;* [b]*Mail Stop: JF1-61, 2111 N.E. 25th Ave., Intel Corporation, Hillsboro, OR 97124-5961*

*(Received 3 August 1994; In final form 3 February 1996)*

An efficient solution to the generalized detailed routing problem in segmented channels for row-based FPGAs is presented. A generalized detailed routing allows routing of each connection using an arbitrary number of tracks, i.e., doglegs are allowed. This approach is different from the normally followed method where each connection is routed on a single straight track. We present a router that performs generalized segmented channel routing using a greedy approach to route channels. The router also renders itself to limited tolerance against faults in the routing architecture.

*Keywords:* Row-based FPGAs, segmented channel, channel routing

## 1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are becoming increasingly popular for *rapid prototyping* of *Application Specific Integrated Circuits* (ASICs). FPGAs exist in various layout styles which include matrix architecture, row based architecture, sea of gates, and complex PLDs. A row-based FPGA is composed of rows of configurable logic modules. Two rows of logic modules are separated by a routing channel. The width of routing channel is fixed. In order to incorporate user configurability within the interconnection path, a wiring track is organized as a set of *segments* of various lengths. The segments are arranged adjacent to one another in horizontal routing channels. Two or more adjacent segments can be connected electrically to form a long wiring path. Vertical wiring segments run over the logic modules. Each pin of a module connects to a dedicated vertical segment. Vertical segments that are not dedicated to the pins of modules serve as feed-throughs between the routing channels. *Antifuses* are provided at the intersection of vertical and horizontal segments. An Antifuse, when programmed, provides a low resistance path between two segments [6]. Fuses and pass transistors are also provided between two adjacent horizontal/vertical segments. When programmed, pass transistors provide a low resistance path between two adjacent segments. Figure 1 illustrates the channeled field programmable gate array architecture. Commercially such FPGAs are available from Actel [4, 8, 7].

*Corresponding author. Tel.: (513)-556-2570. Fax: (513)-556-7326. E-mail dinesh.bhatiaUC.EDU.

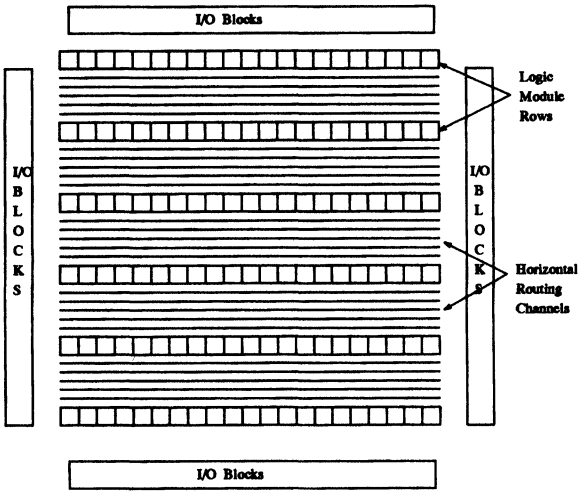


FIGURE 1 General Architecture of Channeled FPGA

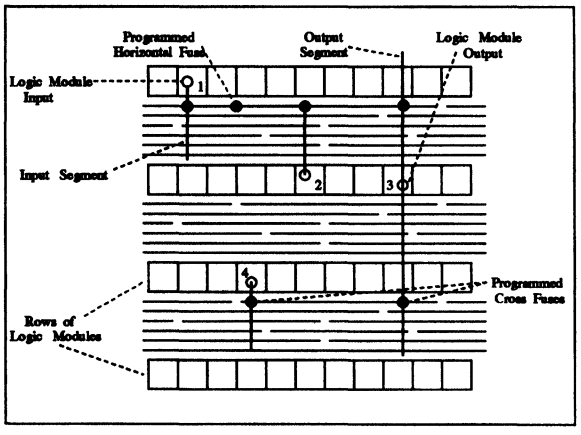


FIGURE 2 An example of routing in channeled FPGAs

There are various programming technologies available and each has its own advantages and disadvantages. One common disadvantage that appears across all the programming technologies is that they have moderately high programmable fuse *resistance*. A typical programmable fuse can provide a resistance from 500 ohm to 2K ohm. Such high resistance results in serious performance degradation when a wiring path has many programmable fuses. The more the number of fuses in any one routing path, the worse is the performance. One obvious way to reduce the number of fuses per routing path is to provide long segments within the channel. However, such a performance enhancement is obtained at the cost of low channel utilization and reduced circuit routability. Normally, routing is performed by achieving a good compromise between segment lengths and the overall design performance.

## 2. RELATED WORK

The segmented channel routing problem is NP-complete [11]. The channel routing problem for segmented channels is much more restrictive than the conventional channel routing problem [11] which in itself is NP-complete. An example of routing in a channeled FPGA is illustrated in Figure 2. The output of module 3 is brought on to a horizontal segment through means of a programmed switch, which in turn is connected to the input pin of module 4 through another programmed switch. Two horizontal segments are connected together to form a longer path so as to reach the inputs of cells 1 and 2 and form a connection.

The first known theoretical results on the combinatorial complexity and algorithm design for segmented channel routing were presented by Roychowdhury, Greene, and El Gamal in [11]. *Unlimited Segment Routing* (*USR*) that involves finding a routing when each connection can use an arbitrary number of segments is known to be NP-complete. *k-Segment Routing* (*k-SR*) is a restriction on the unlimited segment problem where each connection can occupy at most $k$ number of segments. $k$-segment routing is strongly NP-complete even when $k = 2$. If all tracks are segmented *identically*, *i.e.*, the locations of the switches in each track is the same then it is easy to see that *USR* and *k-SR* problems can be solved using the *left-edge algorithm* [9] in $O(MT)$ time, where $M$ is the number of connections to be routed and $T$ is the number of tracks. *Optimal Routing* (*OR*) tries to find a routing that is optimal with respect to some criteria. A general $O(T!M)$ time complexity algorithm using dynamic programming has been presented for solving the *USR*, *k-SR*, and *OR* problems [11]. An important point to be noted for all these results is that a connection is always restricted to be routed using single wiring track.

*Generalized Segmented Channel Routing* (*GSCR*) allows a connection to be assigned to more than one track. In other words, unlike previous discussion where each connection is restricted to one track, generalized segmented channel routing allows insertion of *doglegs*, *i.e.*, a connection can partially route on a wiring track using some wiring segments, change over to another track using a vertical segment, and continue to route using more wiring segments that are dispersed on various tracks. This problem can have many formulations. A generalized routing that uses at most $k$ segments for routing, or that uses at most $l$ different tracks for routing, or the one where connection can switch tracks only at pre-determined columns are some of the formulations. Preliminary results have been presented for the generalized segmented channel routing problem in [11]. It is also shown that a polynomial-time algorithm exists for generalized segmented channel routing problem if the number of tracks is bounded. Zhu and Wong [14] have presented a heuristic algorithm for $k$-segment routing. They reduce the problem of segmented channel routing to the problem of finding a *maximum independent set* of an undirected graph and then solve it using a greedy heuristic. Burman, *et. al.* [2], have presented segmented channel routing algorithm that has the following features—(1) it minimizes not only the delay of the longest net but also reduces the average delay; (2) while calculating the delay, not only the number of anti-fuse, but also the resistive and capacitance effects of the unused portions of the tracks and the un-programmed switches, are taken into account. A modified version of the Elmore delay model [13] has been used to compute the signal delays. Roy has developed an efficient bounded search algorithm for segmented channel routing[10].

## 3. GENERALIZED SEGMENTED CHANNEL ROUTING

In this section, we describe a router for the generalized segmented channel routing *GSCR* problem. Our router (GSC) is capable of performing *generalized segmented channel routing* [11], *i.e.*, each connection can be routed using more than one track. This is possible when vertical segments are available as part of the routing architecture, so that a connection can lie partly on one track and can take a *dogleg* to another track through a vertical segment. In row based FPGAs, such vertical resources are sometimes present. After *technology mapping* [5] and *placement* of logic modules, input and output wires associated with unused logic blocks can be used for introducing doglegs. Long vertical lines that run across the FPGA architecture can also be used for introducing doglegs. The practice of using long wires for introducing doglegs may not be very practical due to excessive delay introduced by capacitance of long lines. The principal benefit of doglegs is that it can help in increasing the routing capacity of a segmented channel. Figure 3 illustrates an example of a segmented routing problem where for the given set of connections and channel, there exists a routing only if doglegs are allowed. It can easily be seen that the following routing solution involving one dogleg exists: *c2* is assigned to *s21*, *c3* to *s22*, *c1* to *s11* and *s23* (dogleg), *c4* to *s12* and *c5* to *s24*. This is illustrated in Figure 4. The dogleg is introduced for connection *c1* just before the switch between segments *s11* and *s12*. Another solution of routing is identical to one just described except that connection *c4* is assigned to segment *s24* and connection *c5* is assigned to segment *s12*.

The area-performance trade-offs are handled with the help of a cost function which takes into account the number of Antifuses that the net will have to drive and the segment wastage that will occur if a connection is assigned to a track in a column.

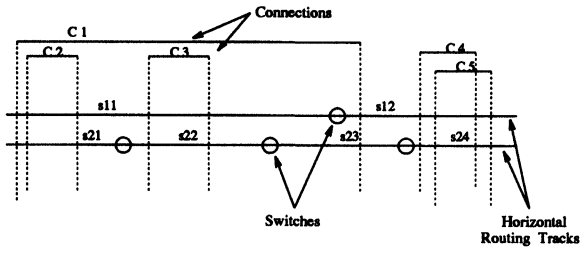


FIGURE 3 A case where routing is not possible without doglegs

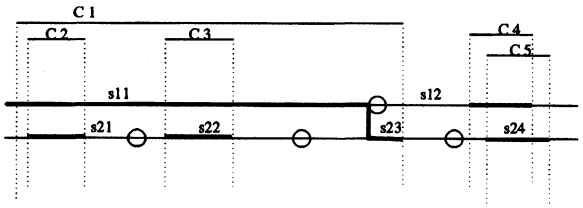


FIGURE 4 Routing solution for the routing instance in figure 3

### 3.1 Associated Terminology

We describe some terms and data-structures before presenting the description of the router. Routing is carried out on a channel divided in $C$ columns, where $C$ is the number of logic modules in a row. A column is a band of width equal to the width of one logic module. Figure 5 illustrates a channel. Columns are numbered 1 through $C$. For each column $j$ a list of connections to be routed is maintained in the *Connections_List(j)*. This list comprises of two types of connections. One type ($L_1$) refers to those connections that start in the column $j$. The second type ($L_2$) comprises of the connections that start in column $i$, where $1 \leq i \leq j - 1$. Figures 6 and 7 illustrate the connections of type $L_1$ and $L_2$ respectively. The routing is performed by successively visiting columns from left to right. Clearly, the connections of type $L_2$ are the ones that can be moved from one track to another using a *dogleg*. While visiting a column $j$, $1 \leq j \leq C$, the connection is moved to a different track using an unused a vertical segment belonging to column $j - 1$. Suppose a connection $c_{j-1}$ belonging to track $t_1$ in column $j - 1$ is to be moved (using a dogleg) to track $t_2$ in column $j - 1$. Such an assignment of $c_{j-1}$ is possible *iff*, (1) track $t_2$ is unoccupied in columns $j - 1$ and $j$, and (2) there exists a horizontal pass transistor on track $t_1$ before the beginning of column $j$ such that track $t_1$ can be used to assign incoming (new)

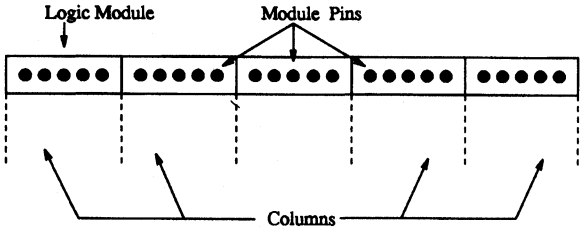


FIGURE 5 A routing channel

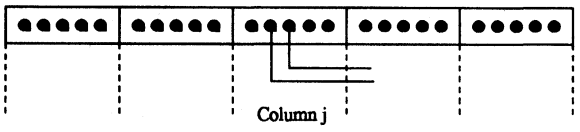


FIGURE 6 Connections of Type $L_1$

connections in column $j$. In other words, if track $t_1$ extends a single metal segment from column $j - 1$ to column $j$, then the doglegging is not useful. In fact, in the absence of horizontal pass transistor, the doglegging introduces two unnecessary fuses in the routing of connection $c_{j-1}$. Similarly, the list of free (unassigned) tracks for each column $j$, $1 \leq j \leq C$, is maintained in *Tracks_List(j)*. The upper bound for the number of connections in the *Connections_List(j)* is also the *channel density* at column $j$.

The routing of connections, *i.e.*, assignment to tracks is maintained in a tree data-structure called the *Routing_Permutations_Tree (RPT)*. An RPT is constructed for each column $j$, $1 \leq j \leq C$, and is denoted by *RPT(j)*. *RPT(j)* is constructed from *Connections_List(j)* and the *Tracks_List(j)*. The root of *RPT(j)* is a node labeled *Col(j)*. *RPT(j)* has $|Connections\_List(j)| + 1$ number of levels, the root being at level 0. Each level $i$, $1 \leq i \leq |Connections\_List(j)|$, denotes all possible assignments of connection $i \in Connections\_List(j)$ to the tracks belonging to *Tracks_List(j)*, $1 \leq j \leq C$. We define $c_p^j \rightarrow t_k^j$ as an assignment of connection $c_p^j \in Connections\_List(j), 1 \leq p \leq |Connections\_List(j)|$, to a track $t_k^j \in Tracks\_List(j)$, $1 \leq k \leq |Tracks\_List(j)|$, $1 \leq j \leq C$. Assignment of connection $c_p^j \in Connections\_List(j)$, $1 \leq p \leq |Connections\_List(j)|$, to a track $t_k^j \in Tracks\_List(j)$, $1 \leq k \leq |Tracks\_List(j)|$, $1 \leq j \leq C$ is said to be *valid_assignment* if,

1. connection $c_p^j$ is of type $L_1$ and track $t_k^j$ is unoccupied in column $j$, or

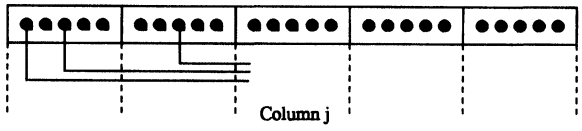


FIGURE 7 Connections of Type $L_2$

2. connection $c_p^j$ is of type $L_2$, continues on using the same track, and track $t_k^j$ is unoccupied in column $j$, or
3. connection $c_p^j$ is of type $L_2$ and switching tracks to $t_k^j$, then $t_k^j$ should be unoccupied in both column $j$ and column $j-1$.

We construct *RPT(j)* in the *breadth first* order, *i.e.*, for all *valid_assignment* $c_1^j \rightarrow t_k^j$, $1 \leq k \leq |Tracks\_List(j)|$, a child of the root node is created with label $t_k$. For each node at level $i-1$, a child with a label $t_k$ is inserted at level $i$ for a *valid_assignment* $c_i^j \rightarrow t_k^j$ under the restriction that ancestors of the inserted node at level $i$ (connections 1 through $c_{i-1}^j$) have not been assigned to track $t_k^j$. It can easily be seen that *RPT(j)* enumerates all possible routings of $c_p^j \in Connections\_List(j)$ within column $j$. In addition to a label indicating the track assignment, a node at any level also maintains *accumulated_cost* of routing. The *accumulated_cost* for a node $A$ at level $i$ is $\Sigma_{k=1}^{i}$ *assignment_cost(k)*, where *assignment_cost(k)* is the cost of assignment $c_k^j \rightarrow t_p^j$, $t_p^j \in Tracks\_List(j)$, and the node representing $c_k^j \rightarrow t_p^j$ belongs to the path from root to $A$. The *assignment_cost* reflects the number of *Antifuses* programmed to route a connection and the waste of metal. The waste of metal occurs because connections do not always end at the end of assigned segments. Since our router scans the channel from left to right, our cost function takes only anticipated metal waste into account, *i.e.*, amount of metal wasted to the right end of a connection. Currently with PLICE Antifuse a resistive load of about 500 ohms is incurred for each programmed fuse. Generally, in practice, a connection should not contain anymore than four programmed fuses.

## 3.2. Cost Function

As stated earlier, the cost function reflects the number of programmed Antifuse as well as the metal waste. *assigmnment_cost* can be written as,

$$\alpha \times fuse + \beta \times excess\_metal$$

where *fuse* is the number of Antifuses (**cfuse** + **hfuse**) programmed for routing a connection and *excess_metal* represents the amount of wasted metal on the right end of the last segment belonging to a connection. The quantity *excess_metal* minimizes, in a greedy fashion, the metal waste as the routing progresses from left to the right end of the routing channel. $\alpha$ and $\beta$ are constants. Figure 8 illustrates the design of the cost function.

The algorithm for routing is described now.

*Procedure* **Route**

*Input*: A segmented channel and a list of connections to be routed.

*Output*: A routing of connections with the assignment of tracks.

*Step* 1: For each column $j$, $1 \leq j \leq C$, do steps 2 to 5.

*Step* 2: Form the *Connections_List(j)*, form the *Tracks_List(j)*, root node labeled *col(j)* for *RPT(j)*.

*Step* 3: For each connection $c_i^j$, $c_i^j \in Connections\_List(j)$ and for each track $t_k^j$, $t_k^j \in Tracks\_List(j)$, if $c_i^j \rightarrow t_k^j$ is a *valid_assignment* then create a new node with a label $t_k$ and add an edge from its parent at level $i-1$ to the new node at level $i$ and asso-

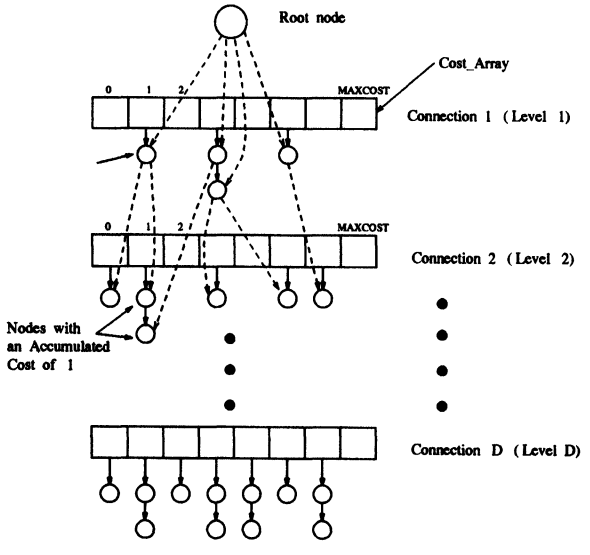


FIGURE 8 The cost function

ciate an *assignment_cost* for the added node. $1 \leq i \leq |Connections\_List(j)|$ and $1 \leq k \leq |Tracks\_List(j)|$.

*Step* 4: Select the node $A$ at level $|Connections\_List(j)|$ with least *accumulated_cost*. The path from $A$ to the root node is the least cost routing of connections in *Connections_List(j)* in column $j$.

*Step* 5: Assign connections to the tracks as per path in step 4 and mark the associated segments in column $j$ as occupied.

## 4. COMPLEXITY OF ROUTING

In this section we give analysis of the worst case space and time complexity of the routing.

The time spent in routing is due to the expansion of the *Routing_Permutations_Tree* for each column. The exhaustive enumeration of all paths can lead to an exorbitantly large tree. In order to control the combinatorial explosion at lower levels of the tree, we constantly *prune* the tree to retain only few best candidate nodes. At each level, before propagating the children to the next level, the number of nodes is pruned to a much smaller number. We define the *window_size*, $W$, which represents the number of nodes that are retained at each level. The rest are freed. The nodes that are retained are not selected randomly but according to their *accumulated_cost*. For this purpose a bucket data-structure called *cost_array* is maintained at each level. The *cost_array* is an array of pointers, each element of the array being a pointer to all the nodes in that level (stored as a linked list), which have an *accumulated_cost* equal to the index value of the bucket element (see Figure 9). Thus the nodes at any level are split up among the elements of the *cost_array* according to their *accumulated_cost* value. At the time of pruning, only the least cost nodes are retained. This is easy since the *cost_array* has the nodes sorted according to their *accumulated_costs*. The *cost_array* helps in speeding up the algorithm considerably, because insertion of a node at the next level just consists of indexing into the *cost_array* using the cost field of the node and inserting the node at the beginning of the list. This is a constant time operation. It also speeds up the pruning as the nodes can be retained starting from the first element of the *cost_array* and proceeding till $W$ number of nodes are reached. The rest of the nodes are freed. The value of $W$ was determined experimentally. It was found that even when the value of $W$ was kept as small as 50 nodes, there was no degradation in the quality of routing when compared against an exhaustive enumeration. This also played an important role in the speed of routing and the low memory requirements of the router.

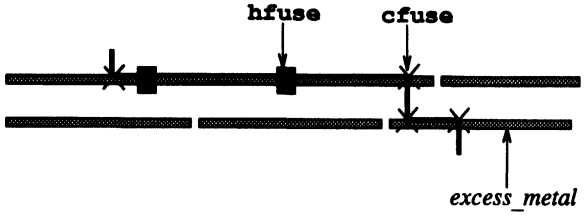


FIGURE 9 The Routing Permutations Tree for the GSC Router

THEOREM 1. *The maximum number of nodes at each level is less than or equal to T! where T is the number of routing tracks in a channel.*

*Proof:* The root node can have at most $T$ number of child nodes. Each of those children can have at most $(T - 1)$ children because the track used by the parent node cannot be used by the child node. Hence, the total number of nodes at the level-1 can be $T$, the level-2 $T(T - 1)$, the level-3 $T(T - 1)(T - 2)$ and so on. At level $i$ there will be at most $T(T - 1)(T - 2)\ldots(T - (i - 1))$ nodes. The maximum number of levels in the tree is at most the channel density, which is at most equal to $T$. Hence the number of nodes in each level is at most $T!$.

The channel density $D$ sets an upper bound on the number of connections in the *Connections_List(j)*, $1 \leq j \leq C$. Thus, there can be at most $D$ levels in the *RPT(j)*, $1 \leq j \leq C$. A node at any level can have at most $t_j$ children, where $T_j$ is the number of tracks in the *Tracks_List(j)*. Before a node is inserted in the *RPT(j)*, $1 \leq j \leq C$, it is checked if the assignment is valid. This can be done in at most $O(D)$ time. Then the overall time complexity of the algorithm is

$O(D^2.T.W.C)$. The memory requirement for the router is bounded by the maximum size of $RPT(j)$, $1 \leq j \leq C$. Worst case complexity for both time and space occurs when $D$, the channel density, is equal to $T$, the total number of tracks, for all the columns and when there is no pruning carried out for the *Routing_Permutations_Tree*. If there is no pruning the maximum number of nodes at any level in the *Routing_Permutations_Tree* will be T! instead of $W$, as established by Theorem 1. And the total number of nodes in all the levels will be $O(T!)$. Hence $D$ will be substituted by $T$ and $W$ by $T!$ and the time and memory requirements will be $O(T^3T!C)$ and $O(T!)$ respectively.

LEMMA 1 *The overall worst case time complexity of the routing algorithm is* $O(T^3T!C)$.

LEMMA 2 *The worst case memory requirements for the router is* $O(T!)$.

## 5. TOLERANCE AGAINST FAULTY ROUTING ARCHITECTURE

The *Actel style* row based FPGA architecture lends itself to limited fault tolerance. This was first noted by Kaushik Roy [10]. In order to make efficient use of such fault tolerance, the configuration tools must have algorithms that can work around the faults. In FPGA configuration, these tools would include programs for *technology mapping*, *placement*, and *routing*. In order for these tools to work around faults, the complete knowledge of fault pattern is required. In case of faulty logic blocks the placement algorithm should avoid using the faulty logic blocks. We are interested in addressing routing fault tolerance, *i.e.*, how to use our router in the presence of faults in the routing architecture. The routing architecture comprises,

1. *wiring segments*,
2. horizontal (**hfuse**) *and cross* **cfuse** fuses.

Antifuses offer extremely high resistance[1] prior to FPGA configuration. Thus a faulty Antifuse (**hfuse** or **cfuse**) would offer a low resistance short circuit path. If a **cfuse** is faulty in column $j$ then there exists a wiring segment in column $j$, which is connected to an input/output pin of the logic block in column $j$ due to the faulty **cfuse**. Let such wiring segment be $s_j$ and let $s_j$ belong to track $t_k$. If the location of faulty **cfuse** and associated connected segment is known *a priori*, our algorithm can route around the faulty *cfuse* with minimal modification. To achieve this we do the following steps,

- while routing for some column $i + 1$ such that $i < j$ and column $i$ has a vertical resource for a possible dogleg, and column $i$ is closest to $j$ satisfying such property, do the following,
  - the *assignment_cost* for each *valid_assignment* of connections in *Connection_List*$(i + 1)$ to track $t_k$ is set to $\infty$.

Later for all columns from $i + 2$ to $j$, the track $t_k$ would not be included in the *Tracks_List*. The step described above forces connection occupying track $t_k$ in column $i + 1$ to some other track using the vertical resource in column $i$ and prohibits any other connection to use track $t_k$ until column $j$. Figure 10 illustrates this step.

In case a horizontal **hfuse** is faulty such that two segments are electrically connected, then the two connected segments should be treated as one long segment while describing the input to our algorithm. This is again based on assumption that the location of faults is known prior to the FPGA configuration. The case when segment becomes faulty due to discontinuity is also easy to handle. The faulty segment has to be avoided and the routing is done using step very similar to the one used for **cfuse** faults. Figure 11 illustrates the step.

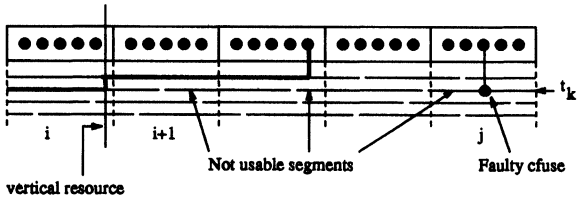


FIGURE 10 Routing in the presence of **cfuse** *fault*

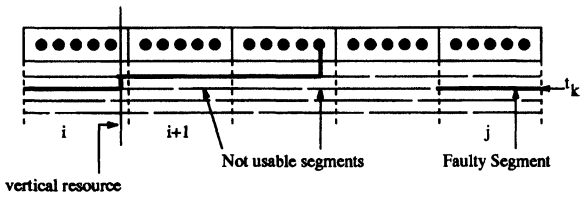


FIGURE 11 Routing in the presence of **segment** *fault*

Thus in all cases, our algorithm provides some degree of fault tolerance. This is a yield enhancing feature. It should be noted that the fault tolerance is achieved only if the location of faults is known. This may not be commercially viable because it requires exposing foundry specific data to the CAD tools.

## 6. IMPLEMENTATION DETAILS

Industrial design circuits from Actel were used to test the router. These circuits were placed on Actel 1280 FPGA using the Actel ALS tool. The global routing for these circuits was performed using the global router developed in [12]. The circuits were mapped on an Actel 1280 FPGA, so the global and detailed router were parameterized for the Actel 1280 architecture [1]. The Actel 1280 FPGA has 17 rows of logic modules, separated by routing channels. There are 82 columns (each comprising of one logic module) in each row. The routing channels have 36 horizontal tracks each, except the first and last channel which have 27 tracks each. The horizontal tracks are segmented in a pre-defined format which was adopted for the detailed router. However the segmentation scheme is just an input parameter and can be changed.

For the sake of comparison, the algorithm presented in [11] for unlimited segment routing (*USR*) was coded and tested against our router. Unfortunately the time and memory requirements for the *USR* grow rapidly with the increase in *T*—the number of tracks, and the pruning heuristics used to reduce these requirements are not known. Hence the two routers were compared using smaller, random netlists. Table I shows the results of executing both

Table I Comparative results of Unlimited Segment Router and Generalized Segmented Channel Router

| USR router | | | GSC Router | | | |
|---|---|---|---|---|---|---|
| No. of tracks | CPU secs. | Fuses | No. of tracks | CPU secs. | Fuses | No. of Doglegs |
| 7 | Not routable | | 7 | 0.2 | 17 | 1 |
| 8 | 4.6 | 16 | 8 | 0.1 | 13 | 0 |
| 9 | 41.4 | 17 | 9 | 0.2 | 15 | 1 |
| 10 | 5.92min | 22 | 10 | 0.3 | 17 | 1 |
| 11 | Out of memory | | 11 | 0.2 | 15 | 1 |
| | | | 15 | 0.3 | 13 | 0 |
| | | | 35 | 0.8 | 6 | 0 |

the routers on four different random netlists for various number of tracks. Here, USR router refers to the *unlimited segment routing* as described in [11], and GSC router refers to our router reported in this paper. Our router has been tested against industrial circuits. Table II gives some details about the circuit examples. These being customer designs, very little functionality details are available.

We report results for representative channels in Table III. Figure 12 illustrates the relationship of execution time against the number of tracks for seven channels.

Figure 13 illustrates the performance in terms of number of fuses programmed against the number of tracks for seven of the routing channels.

### 6.1. Channel Width

The reason why channel width does not improve significantly even when doglegs are allowed in the channel is probably because the segmentation scheme used for the experimentation has been tuned towards

Table II Benchmark examples from industry

| Actel Name | No. of Modules |
|---|---|
| Design 1 | 782 |
| Design 2 | 1142 |
| Design 3 | 1112 |
| Design 4 | 1224 |
| Design 5 | 1056 |
| Design 6 | 1020 |
| Design 7 | 1033 |
| Design 8 | 1086 |

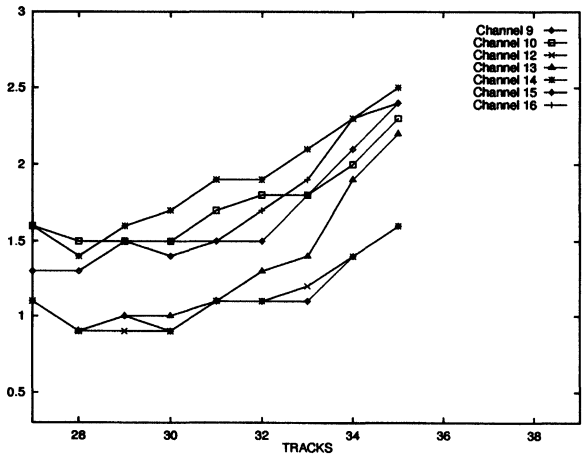


FIGURE 12 Execution Time Vs Number of Tracks

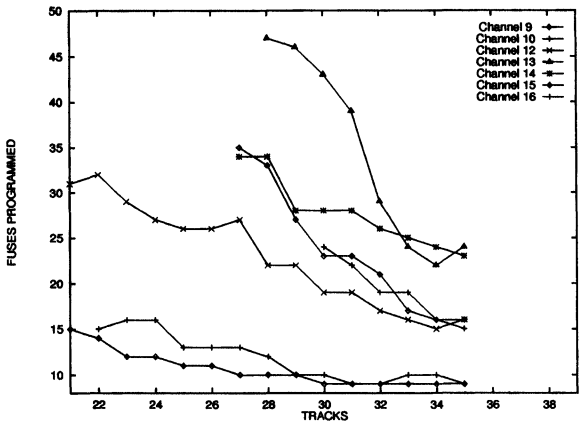


FIGURE 13 Number of Antifuses Programmed Vs Number of Tracks

single-track routing and not towards generalized segmented routing. The design of a segmentation scheme to facilitate doglegs remains an open problem and has not been addressed in this work. The absence of vertical constraints will reduce the number of cases where doglegging is effective in reducing channel width. It must be pointed out that a few doglegs were introduced by our router during routing and it did improve locally the segment utilization. But only in one case ((Table I), Netlist # 1) was it significant enough to cause an overall reduction in channel width. This is indicated by row #1 of Table I.

Table III Routing results for GSC Router on selected channels

| Chan. No. | No. of Nets | No. of tracks | CPU sec. | No. of Doglegs | Fuses |
|---|---|---|---|---|---|
| 2 | 175 | 26 | 1.3 | 1 | 14 |
| | | 27 | 1.6 | 2 | 10 |
| | | 28 | 1.4 | 1 | 10 |
| | | 29 | 1.6 | 1 | 7 |
| | | 30 | 1.5 | 1 | 6 |
| | | 31 | 1.8 | 1 | 6 |
| | | 35 | 2.4 | 0 | 4 |
| 13 | 171 | 28 | 0.9 | 1 | 47 |
| | | 29 | 1.0 | 0 | 46 |
| | | 30 | 1.0 | 1 | 43 |
| | | 31 | 1.1 | 1 | 39 |
| | | 32 | 1.3 | 2 | 29 |
| | | 33 | 1.4 | 2 | 24 |
| | | 34 | 1.9 | 2 | 22 |
| | | 35 | 2.2 | 2 | 24 |
| 14 | 189 | 27 | 2.1 | 3 | 34 |
| | | 28 | 2.3 | 1 | 34 |
| | | 29 | 2.3 | 3 | 28 |
| | | 30 | 2.5 | 2 | 28 |
| | | 31 | 2.7 | 2 | 28 |
| | | 32 | 2.7 | 1 | 26 |
| | | 33 | 2.9 | 1 | 25 |
| | | 34 | 3.4 | 1 | 24 |
| | | 35 | 3.6 | 1 | 23 |

## 6.2. Performance of the Router in Terms of Delay and Run-Time

The main contributor to the net delay in segmented routing is the number of Antifuses that lie in the path of the driving signal. The PLICE Antifuse which is used in Actel family of FPGAs has a resistance (after programming) of about 500 ohms. This is so significant that if more than 3–4 Antifuses are driven by a signal, rapid deterioration of the signal level takes place. Hence, it is highly desirable to reduce the number of programmed Antifuses in the path of the nets.

As seen from Tables I and III, the number of fuses programmed due to routing by our router decreases significantly as the search space, in terms of number of tracks, increases. And this is achieved without a significant increase in run-time.

## 6.3. Memory Requirements

The memory requirements for our router are extremely small and this is another attractive feature of

the router. This is so because of the column-by-column nature of routing. For each column the *Routing_Permutations_Tree* is built and then freed before going on to the next column. Also, due to pruning the size of the *Routing_Permutations_Tree* in the GSC Router is kept quite small.

## 7. CONCLUDING REMARKS

A generalized segmented channel router has been developed for row-based FPGAs. It has a capability of performing generalized segmented channel routing, i.e., the connections can, by means of doglegs, be assigned to more than one track. The memory requirements for the router are very low. The router is extremely fast even for large number of tracks in the routing channel. The design of a segmentation scheme to make better use of this router, especially its capability to put doglegs, remains to be explored.

### *Acknowledgments*

We would like to thank Sinan Kaptanoglu from Actel Corporation for providing us with design examples for testing our router. We also thank Vijayanand Sankarasubramanian and Doug Smith for assisting us in preparing the final manuscript.

### *References*

[1] Actel Corporation, Sunnyvale, California. *ACTA Family Field Programmable Gate Array DATABOOK*, 1993.

[2] S. Burman, C. Kamalanathan, and N. Shervani. New channel segmentation model and associated routing algorithm for high performance FPGAs. In *Proceedings of ICCAD-92*, pages 22–25, 1992.

[3] D. N. Deutsch. A dogleg channel router. In *Proc. 13th Design Automation Conference*, pages 425–433, 1976.

[4] K. A. El-Ayat, A. El Gamal, R. Guo, J. Chang, R. K. H. Mak, F. Chiu, E. Z. H. Amdy, J. McCollum, and A. Mohsen. A CMOS Electrically Configurable Gate Array. *IEEE J. Solid-State Circuits*, 24(3):752–762, 1989.

[5] Silvia Ercolani and Giovanni De Micheli. Technology Mapping for Electrically Programmable Gate Arrays. In *Proceedings of Design Automation Conference*, pages 234–239, 1991.

[6] E. Hamdy *et al.* Dielectric based antifuse for logic and memory ICs. IEDM *Tech. Digest* (Sun Francisco, CA), 1988.

[7] Mike Ahrens *et. al.* An FPGA family optimized for high densities and reduced routing delay. In *Proc. of Custom Integrated Circuits Conference*, pages 31.5.1–31.5.4, 1990.

[8] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat, and A. Mohsen. An architecture for electrically configurable gate arrays. *IEEE J. Solid-State Circuits*, 24:394–398, Apr 1989.

[9] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proc. 8th IEEE Design Automation Workshop*, 1971.

[10] K. Roy. A Bounded Search Algorithm for Segmented Channel Routing for FPGAs and Associated Channel Architecture Issues. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 12(11), 1993.

[11] V. P. Roychowdhary, J. Greene, and A. El Gamal. Segmented channel routing. *IEEE transactions on CAD*, 12(1):79–95, January 1993.

[12] V. Shankar. Routing for Row-Based Field Programmable Gate Arrays. Master's thesis, University of Cincinnati, Department of Electrical and Computer Engineering, December 1993.

[13] Ren-Song Tsay. Exact zero skew. In *Proceedings of the ICCAD-91*, pages 336–339, 1991.

[14] Kai Zhu and D. F. Wong. On channel segmentation design for row-based FPGAs. In *Proceedings of ICCAD-92*, pages 26–29, 1992.

### *Endnotes*

1. High enough that it amounts to an open circuit

### Authors' Biographies

Dinesh Bhatia is an Assistant Professor in the department of Electrical and Computer Engineering and Computer Science at the University of Cincinnati. He also directs the Design Automation Laboratory within the same department. Prior to his current position he was a visiting Assistant Professor of Computer Science and Engineering at the Southern Methodist University in Dallas. His research interests include the architecture and CAD for field-programmable gate arrays, interconnection problems in VLSI, physical design of MCMs and large ICs and, graph theory and its application in VLSI design.

Shankar Venkataraman is a CAD Engineer in Intel Corporation at Hillsboro, Oregon. He is currently working in the Design Technology group at Intel, in the area of Physical Design CAD. He received the

Bachelor of Engineering from the University of Roorkee, Roorkee, India in 1991, and his Master of Science degree in Computer Science and Engineering from the University of Cincinnati, Ohio in 1993.