A Detailed Router for Field-Programmable Gate Arrays

Stephen Brown, *Student Member, IEEE*, Jonathan Rose, *Member, IEEE*, and Zvonko G. Vranesic, *Senior Member, IEEE*

Abstract—**This paper describes a new kind of detailed routing algorithm that has been designed specifically for field-programmable gate arrays (FPGA's). The algorithm is unique in that it approaches this problem in a general way, allowing it to be used over a wide range of different FPGA routing architectures. The detailed routing of FPGA's is a new problem and can be more difficult than classic detailed routing because the wiring segments that are available for routing are preplaced and can only be connected together in specified patterns. In some FPGA's, the routing architecture places exacting limitations on the routing choices for any connection, and in such cases there will routing channels in the FPGA where overlapping routing alternatives of two or more connections create competition for the same wiring segments. Resolving this competition is essential for achieving 100% routing in these FPGA's. The algorithm described here, called the coarse graph expansion (CGE) detailed router for FPGA's, addresses the issue of scare routing resources by considering the side effects that the routing of one connection has on another, and also has the ability to optimize the routing delays of time-critical connections.**

CGE has been used to obtain excellent routing results for several industrial circuits implemented in FPGA's with various routing architectures. The results show that CGE is able to route relatively large FPGA's in very close to the minimum number of tracks as determined by global routing, and it can successfully optimize the routing delays of time-critical connections. CGE has a linear run time over circuit size.

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGA's) represent a new approach to application-specific integrated circuits (ASIC's) that reduces IC manufacturing time from months to minutes, and manufacturing costs from thousands of dollars to under $100. An FPGA has an array of logic cells connected by a general routing structure, like a mask programmable gate array, but it is programmed by the user in the same way as a programmable logic device (PLD). The FPGA was first introduced in [1], with newer versions presented in [2]–[13]. The complexity of FPGA's has reached the point where it is essential to have automatic design tools in order to make effective use of them. This paper focuses on the problem of FPGA routing.

Manuscript received January 17, 1991. This work was supported by NSERC Operating Grants URF0043298 and OGP0005280 and by research grants from Bell-Northern Research and ITRC. This paper was recommended by Associate Editor M. Marek-Sadowska.

The authors are with the Department of Electrical Engineering, University of Toronto, Toronto, Ont., Canada M5S 1A4.

IEEE Log Number 9105724.

Detailed routing for FPGA's can be more difficult than classical detailed routing [14], [15] because connections are made using wiring segments that are already in place and joins between segments are possible only at predetermined places where routing switches exist. The implementation of routing switches could take the form of static RAM controlled pass transistors [2], [3], [7], [9], antifuses [4], [5], [13], EPROM transistors [6], [8], [11], [12], or EEPROM [10].

A key problem in the detailed routing of FPGA's is that the routing of one connection may unnecessarily block another. Consider Fig. 1, which shows three views of the same section of an FPGA. Each view gives the routing options for one of connections A, B, and C. In the figure, a routing switch is shown as an X, a wiring segment as a dotted line, and a possible route as a solid line. Now, assume that a router first completes connection A. If the wiring segment numbered 3 is chosen for A, then one of connections B and C cannot be routed because they both rely on the same single remaining option, namely the wiring segment numbered 1. The correct solution is for the router to choose the wiring segment numbered 2 for connection A, in which case both B and C are also routable. Although this is a simple example, it illustrates the essence of the problems that occur because of limited routing options in FPGA's.

Common approaches used for detailed routing in other types of devices are not suitable for FPGA's. Maze routers [16] are ineffective because they are inherently sequential and so, when routing one connection, they cannot consider the side effects on other connections. Channel routers [17] are not appropriate because the general routing problem cannot be subdivided into independent channels. Note that a channel routing algorithm is used in [18] for Actel-like FPGA's [4], [5]. This is possible for these types of FPGA's because the logic cells are arranged in rows separated by routing channels and the routing switches are such that each logic cell pin can connect to all the wiring segments in the channels above and below it, and each horizontal wiring segment can connect to all the vertical wiring segments that cross it. This routing flexibility cannot be assumed for the general model of the FPGA that is used in this paper.

An earlier version of the work presented here appeared in [19]. The rest of this paper is organized as follows: Section II presents the model used for the FPGA; Section

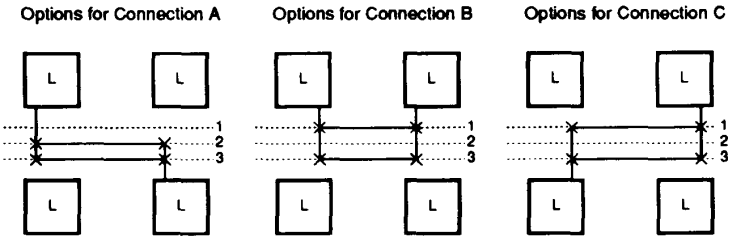


Fig. 1. Routing conflicts.

III defines the detailed routing problem; Section IV describes the CGE routing algorithm; Section V presents the results from tests of the router; and Section VI gives concluding remarks.

II. The FPGA Model

The FPGA is modeled as a two-dimensional array of logic cells interconnected by vertical and horizontal routing channels, similar to [1]. It comprises three major parts: the logic (L), connection (C), and switch (S) blocks, as shown in Fig. 2. The L blocks are programmable cells which house the combinational and sequential logic that form the functionality of a circuit. In general, an L block has a number of pins, each of which may connect to the four adjacent C blocks. The I/O blocks appear as L blocks on the periphery of the chip.

The C blocks are rectangular switch boxes with connection points on all four sides, and are used to connect the L block pins to the routing channels, via programmable switches. Depending on the topology of the C block, each L block pin may be switchable to either all or some fraction of the wiring segments that pass through the C block. The fewer wiring segments connectable in the C blocks, the harder the FPGA is to route. Connections along a routing channel may also pass straight through a C block, but in a typical routing architecture no switch would be involved for such connections.

The S blocks are also rectangular switch boxes. They are used to connect wiring segments in one channel segment to those in another. Depending on the topology, each wiring segment on one side of an S block may be switchable to either all or some fraction of the wiring segments on each other side of the S block. Again, the fewer wiring segments that can be switched to, the harder the FPGA is to route. A connection that passes through an S block may do so through a switch or it may be hard-wired. A connection will have a lower routing delay if it uses hard-wired wiring segments than if it passes through switches.

In Fig. 2, each L block has two pins that appear on all four of its sides, and there are three *tracks* in each routing channel. The figure also defines several terms, such as *channel segment*, *wiring segment*, and *routing channel*. The two-dimensional grid that is overlaid on the FPGA is used later as a means of describing the connections to be routed.

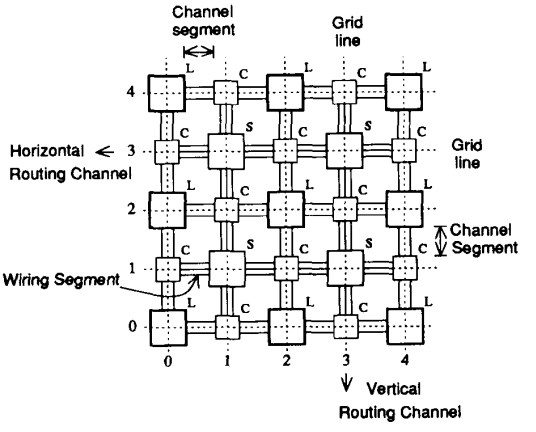


Fig. 2. The FPGA model.

III. General Approach and Problem Definition

As in other design styles, FPGA routing is a complex combinatorial problem. The general approach used here is the usual two-stage method of global routing followed by detailed routing. This allows the separation of two distinct problems: balancing the densities of all routing channels, and assigning specific wiring segments for each connection. The global router used is an adaptation of the LocusRoute global routing algorithm for standard cells [20]. This global router divides multipoint nets into two-point connections and routes them in minimum distance paths. Its main goal is to distribute the connections among the channels so that the channel densities are balanced.

The global router defines a course route for each connection by assigning it a sequence of channel segments. Fig. 3(a) shows a representation of a typical global route for one connection. It gives a sequence of channel segments that the global router might choose to connect some pin of a logic block at grid location 2, 2 to another at 4, 4. The global route is called a coarse graph, $G(V, A)$, where the L block at 2, 2 is referred to as the root of the graph and the L block at 4, 4 is called the leaf. The vertices V, and edges, A, of $G(V, A)$ are identified by the grid of Fig. 2. Since the global router splits all nets into two-point connections, the course graphs always have a fan-out of 1.

After global routing the problem is transformed to the following: for each two-point connection, the detailed

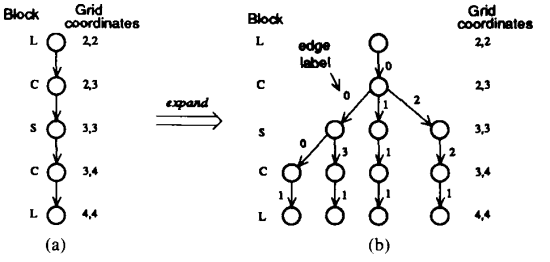


Fig. 3. (a) A typical coarse graph and (b) its expanded graph.

router must choose specific wiring segments to implement the channel segments assigned during global routing. As this requires complete information about the FPGA routing architecture, CGE uses the details of the L, C, and S blocks, as described in the following sections.

IV. The CGE Detailed Router Algorithm

The basic algorithm is split into two phases. In the first phase, it enumerates a number of alternatives for the detailed route of each coarse graph, and then in the second phase, viewing all the alternatives at once, it makes specific choices for each connection. The decisions made in phase 2 are driven by a cost function that is based on the alternatives enumerated in phase 1. Multiple iterations of the two phases allow the algorithm to conserve memory and run time while converging to its final result, as discussed in subsection IV-C.

A. Phase 1: The Expansion of the Coarse Graphs

During phase 1, CGE *expands* each coarse graph and records a subset of the possible ways that the connection can be implemented. For each $G(V, A)$, the expansion phase produces an *expanded* graph, called $D(N, E)$. N are the vertices of D, and E are its edges, with each edge referring to a specific wiring segment in the FPGA. The edges are labeled with a number that refers to the corresponding wiring segment.

In the expansion algorithm, the procedures that define the connection topology of the C and S blocks are treated as *black-box* functions. The black-box function for a C block is denoted as $f_c([d_1, d_2, l], d_3)$ and for an S block as $f_s([d_1, d_2, l], d_3)$. The parameters in square brackets define an edge that connects vertex d_1 to vertex d_2, using a wiring segment labeled l. Such an edge is later referred to as e, where $e = (d_1, d_2, l)$. The parameter d_3 is the successor vertex of d_2. The task of the function call can be stated as: If the wiring segment numbered l is used to connect vertex d_1 to d_2, what are the wiring segments that can be used to reach d_3 from d_2. The function call returns the set of edges that answer this question. As explained in subsection IV-D, this black-box approach is the key to the algorithm's independence with respect to any specific FPGA routing architecture. The result of a graph expansion is illustrated in Fig. 3(b), which shows a possible expanded graph for the coarse graph of Fig. 3(a). The graph expansion process for each coarse graph operates as follows:

Create D and give it the same root as G. Make the immediate successor to the root of D the same as for the root of G.
While traversing D breadth first, enumerate the paths originating at each vertex according to:
  Expand a C vertex in D by calling $f_c(e_C, n) = Z$. e_C is the edge in D that has already been chosen to connect to C from its predecessor. n is the required successor vertex to C (in G) and Z is the set of edges returned by $f_c(\ )$. The call to $f_c(\ )$ adds $|Z|$ edges to D.
  Expand an S vertex in D by calling $f_s(e_S, n) = Z$. e_S is the edge in D that has already been chosen to connect to S from its predecessor. n is the required successor vertex of S (in G) and Z is the set of edges returned by $f_s(\ )$. The call to $f_s(\ )$ adds $|Z|$ edges to D.
Endwhile

B. Phase 2: Connection Formation

After expansion, each $D(N, E)$ may contain a number of alternative paths. CGE places all the paths from all the expanded graphs into a single path list. Based on a cost function, the router then selects paths from the list; each selected path defines the detailed route of its corresponding connection. Because the cost function allows it to consider all the paths at once, CGE can be said to route the connections "in parallel." Phase 2 proceeds as follows (the terms c_f cost and c_t cost will be defined later in this section):

Put all the paths in the expanded graphs into the path list
While the path list is not empty
  If there are paths in the path list that are known to be essential
    Select the essential path that has the lowest c_f cost.
  Else if there are paths in the path list that correspond to time-critical connections
    Select the critical path with the lowest c_t cost.
  Else
    Select the path with the lowest c_f cost
  Mark the graph corresponding to the selected path as routed—remove all paths in this graph from the path list.
  Find all paths that would conflict with the selected path and remove them from the path list (see note). If a connection loses all of its alternative paths, reexpand its coarse graph—if this results in no new paths, the connection is deemed unroutable (see subsection IV-C for a discussion relating to failed connections).
  Update the cost of all affected paths.
Endwhile

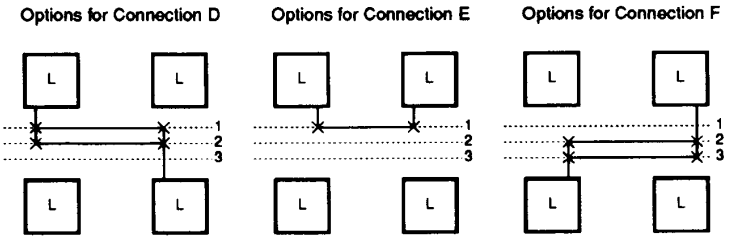


Fig. 4. An essential wiring segment.

Note: When a wiring segment is chosen for a particular connection, it and any other wiring segments in the FPGA that are hard-wired to it must be eliminated as possible choices for connections that are in other nets. This requires a function analogous to $f_c(\ )$ and $f_s(\ )$ that understands the connectivity of a particular FPGA configuration. CGE calls this routine *update* (*e*)—the parameter *e* is an edge in the selected path and *update* (*e*) returns the set of edges that are hard-wired to *e*.

Cost Function Design: Each edge in the expanded graphs has a two-part cost: $c_f(e)$ accounts for the competition between different nets for the same wiring segments, and $c_t(e)$ is a number that reflects the routing delay associated with the wiring segment. Each path has a cost that is simply the sum of the cost of its edges. CGE selects paths based on the c_t cost only if the path corresponds to a time-critical connection. Otherwise, paths are selected according to their c_f cost. The c_f cost has two goals:

1) To select a path that has a relatively small negative effect on the remaining connections, in terms of routability. The cost deters the selection of paths that contain wiring segments that are in great demand.
2) It is used to identify a path that is *essential* for a connection. Such a connection has only one path remaining in the FPGA, because previous path selections have consumed its alternatives.

The reason for using wiring segment demand was illustrated in Fig. 1, where connection A should be routed with wiring segment 2, because wiring segment 3 is in greater demand.

The importance of essential wiring segments is illustrated by the example in Fig. 4. If the router were to complete connection D first, then wiring segments 1 and 2 would be equal candidates according to their demand, since they both appear in one other graph. However, wiring segment 1 is essential for the completion of connection E, and to ensure the correct assignment of the essential wiring segment, connection E should be routed first.

To determine whether an edge, *e*, is in great demand, the router could simply count the number of occurrences of *e* that are in expanded graphs of other nets. However, some occurrences of *e* are less likely to be used than others because there may be alternatives (edges in parallel with *e*). Thus, the c_f cost of an edge *e* that has *j* other occurrences (e_1, e_2, $\cdots$, e_j) is defined as

$$c_f(e) = \sum_j \frac{1}{alt(e_j)},$$

where $alt(e_j)$ is the number of edges in parallel with e_j.

Because of the summing process in $c_f(e)$, the more graphs *e* occurs in, the higher will be its cost. This reflects the fact that *e* is an edge that is in high demand and urges CGE to avoid using *e* when there are other choices. Note that an edge that only appears in its own graph will have a c_f of 0. For the special case where $alt(e_j)$ is 0, e_j is an edge that is essential to the associated connection because there are not alternatives. In this case, any path in the graph that uses e_j is identified as *essential*. When the calculation of a cost reveals that a path is essential, CGE gives that path the highest priority for routing.

C. Controlling Complexity

Although the definition of graph expansion implies that all possible paths in an FPGA are recorded during expansion, this is not practical because the number of paths can be very large in some architectures. For example, consider the connection of two pins on two different L blocks. Assume that each pin can connect to F_c of the wiring segments in the channel segments adjacent to each L block, and that the L blocks are separated by *n* switch blocks. If each wiring segment that enters one side of a switch block can connect to F_s wiring segments on the other three sides, then there are an average of $F_c\,[F_s/3]^n$ different paths from the first pin to the last L block, and assuming *W* tracks in each routing channel, there are an average of

$$\frac{F_c^2}{W}\left[\frac{F_s}{3}\right]^n$$

possible ways to form the connection. Since typical values of F_s should be 3 or greater [21], [22], and the number of connections is large, this complexity must be controlled.

The number of paths in the expanded graphs is reduced by pruning them as they are expanded. The pruning algorithm is parameterized such that the amount of memory space required is controlled and yet the expanded graphs still contain as many alternatives per connection as possible. Maximizing the number of alternatives is important

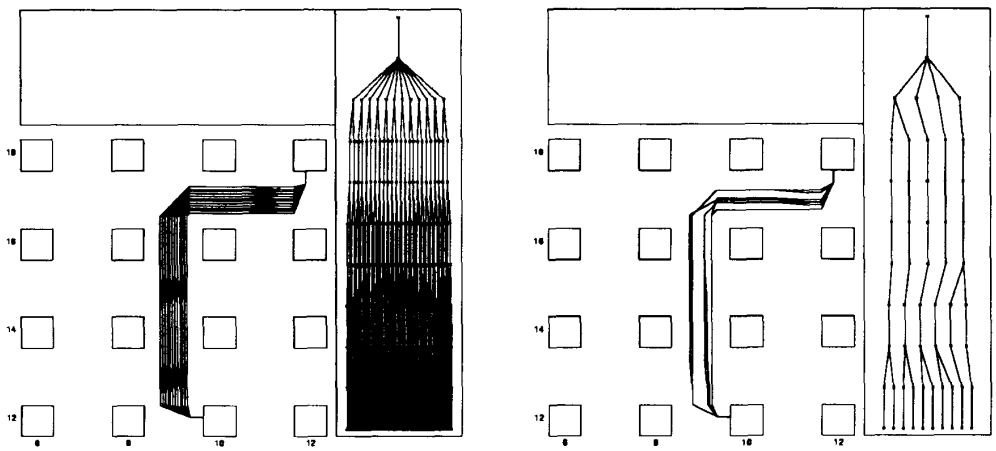


Fig. 5. The effect of pruning.

in the context of resolving routing conflicts. The pruning algorithm is part of the graph expansion process described in subsection IV-A. The general flow follows (the criteria used for pruning is given at the end of this section):

Expand two levels
Prune; keep at most K vertices at this level, and assign each a unique group number. Discard the other vertices and the paths they terminate.
Expand two more levels. Assign each added vertex the group number of its predecessor.
While the leaf level has not been reached.
  Prune; keep at most k vertices with each group number at this level. Discard the other vertices and the paths they terminate.
  Expand two more levels. Assign each added vertex the group number of its predecessor.
Endwhile

The graphs are pruned every two levels because that is where fan-out occurs (after the first C block and after every S block). The parameter K controls the starting widths of the graphs and can take values from 1 to F_c (the number of wiring segments connected to each L block pin). Beyond the maximum value of K, parameter k allows the expanded graphs to further increase in width. The concept of *group* numbers isolates each of the original K paths, which maximizes the number of alternatives at each level of the final expanded graph. The actual values used for K and k are discussed in the next section. The effect of the pruning algorithm is illustrated in Fig. 5. The left half of the figure shows a fully expanded graph from an example circuit, while the corresponding pruned graph is on the right. Also shown are each graph's edges in the FPGA.

The choice of vertices to prune is based on the wiring segment corresponding to their incoming edge, as follows. For the special case of time-critical connections, the wiring segments with the least delay are favored. For other connections, the wiring segments that have thus far been used in the greatest number of other places will be discarded. This helps the c_f cost function discover the wiring segments that are in the least demand.

Note that when paths are discarded because of pruning, they are not necessarily abandoned permanently by the router. In phase 2, as CGE chooses connections, if routing conflicts consume all of the alternatives for some graph, CGE reinvokes the graph expansion process to obtain a new set of paths if some exist.

Iterations: This subsection explains how iterations of the two phases of CGE are used to conserve memory and run time. The iterative approach is linked to the pruning parameters of the graph expansion phase. Setting the pruning parameters to large values allows the router to do a better job of resolving routing conflicts because it sees many alternatives for each connection. On the other hand, with large pruning parameters more memory and run time are required by the algorithm. The key to this routing quality versus memory and time trade-off is the realization that most connections in an FPGA are relatively easy to route and only a small percentage of the connections pose real difficulties. This is because a typical routing problem is likely to have only a few channel segments whose densities are very close to the total number of wires in a routing channel. To exploit this property, the router starts with small pruning parameters and then increases them through successive iterations, but only for the parts of the FPGA that are difficult to route.

For the first iteration the pruning parameters are set to relatively small values, and the entire FPGA is routed. If routing conflicts leave some connections unrouted, then another iteration is required. The procedure is to erase all the routing of any connection that overlaps any part of a failed connection, and then to attempt to route those channel segments again using larger pruning parameters. Only connections that touch some segment of a channel in which a failed connection occurred are routed in the next iteration. Iterations are continued until all connections are routed or until further improvements are not forthcoming

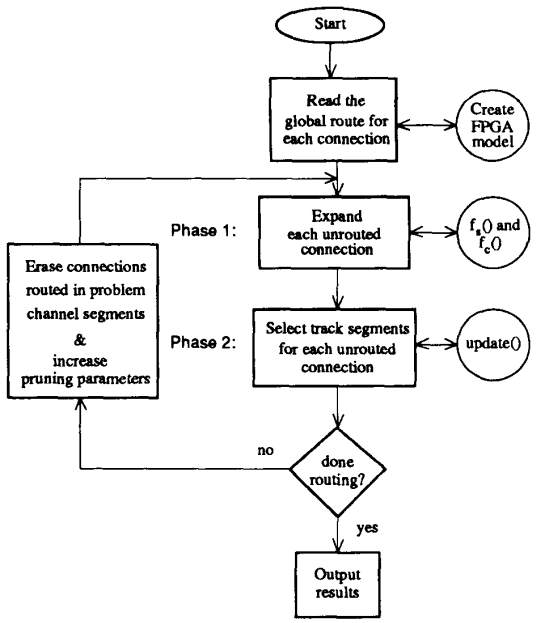


Fig. 6. The organization of CGE.

(note that it would be desirable to try different global routes for connections that are left unrouted after all iterations, but no such failure-recovery mechanism is currently implemented). This approach is a minor variation of classic rip-up and reroute schemes where individual connections would be removed and rerouted to try to resolve routing conflicts. The technique employed here allows CGE's cost function to solve the routing problem, but conserve memory and time where the problem is not difficult and to expend them only where it is required.

The specific values used for the pruning parameters in each iteration affect the total number of iterations required but do not appreciably affect the quality of the final result. This indicates a robustness in the algorithm because the quality of the routing does not depend on the specific values chosen for the program's parameters.

D. Independence of CGE from FPGA Routing Architectures

CGE achieves the ability to route arbitrary FPGA routing architectures by isolating the parts of the code that are architecture-specific. This is illustrated in Fig. 6, which shows the overall flow of the algorithm. The code that is dependent on the routing architecture is enclosed in circles. As shown, the separate code includes the $f_c(\ )$, $f_s(\ )$, and *update* ( ) routines. Any architecture that fits the general model described in Section II can be routed by changing these isolated routines. This generality was used as a research tool in recent papers on FPGA routing architectures [21], [22]. Fig. 6 also shows the organization of the phases of CGE and the feedback path used over multiple iterations.

V. Results

CGE has been used to route several industrial circuits implemented as FPGA's. The routing results shown in this section are based on five circuits from four sources: Bell-Northern Research, Zymos, and two different designers at the University of Toronto. Table I gives the names, size, (number of two-point connections and logic blocks), source, and function of each circuit. For these results, the L block used is the result of a previous study [23], and the S and C blocks will be described in the next subsection. Results are presented for a routing architecture similar to a commercial FPGA.

A. FPGA Routing Structures

Since the routability of an FPGA is determined by the topology and flexibility and its S and C blocks, those used in the tests of the algorithm are presented here. The general nature of the S block is illustrated in Fig. 7(a). Its flexibility is set by a parameter called F_s, which defines the total number of connections offered to each wiring segment that enters the S block. For the example shown in Fig. 7(a), the wiring segment at the top left of the S block can connect to six other wiring segments, and so F_s is 6. Although not shown, the other wiring segments are similarly connected.

Fig. 7(b) illustrates the test C block. The tracks pass uninterrupted through it and are connected to L block pins via a set of switches. The flexibility of the C block, F_c, is defined as the number of tracks that each L block pin can connect to. For the example shown in the figure, each L block pin can connect to two vertical tracks, and so F_c is 2.

B. Routing Results

The familiar yardstick of channel density is used as a measure of the quality of the detailed router. The "channel density" column in Table II shows the maximum channel density over all channels for each circuit. This is the theoretical minimum number of tracks per routing channel required for each example. However, the real track requirements depend on the flexibilities of the routing structures; the maximum flexibility has $F_s = 3W$ and $F_c = W$, where there are W tracks per channel. For the results in Table II, the FPGA parameters are based on the Xilinx 3000 series [24] FPGA's ($F_s = 6$, $F_c = 0.6W$). Table II gives the minimum number of tracks per channel that CGE needs in order to route 100% of the connections. The values for W are slightly greater than the global router minimum, which are excellent results considering the low flexibility of the FPGA routing architecture. Note that if F_c is increased to $0.8W$, CGE achieves the absolute minimum number of tracks for all the circuits.

For comparison purposes, the same problems have also been routed using CGE with its c_f cost facility disabled. In this mode CGE has no ability to resolve routing conflicts and is thus a sequential router, similar to a maze router. At first glance, this may seem to be an unrealistic

TABLE I
EXPERIMENTAL CIRCUITS

Circuit	No. Connections	No. Blocks	Source	Type
BUSC	392	109	UTD1	Bus Cntl
DMA	771	224	UTD2	DMA Cntl
BNRE	1257	362	BNR	Logic/Data
DFSM	1422	401	UTD1	State Mach.
Z03	2135	586	Zymos	8-bit Mult

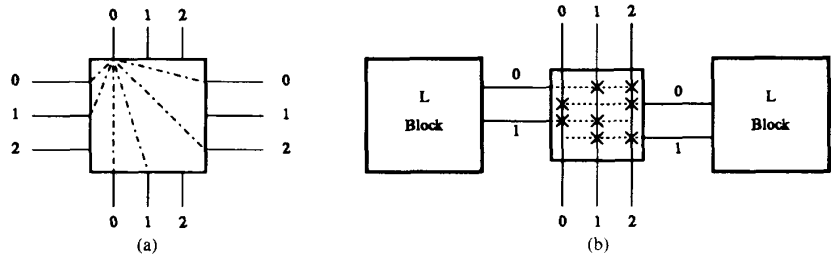


Fig. 7. Definitions of (a) S and (b) C block flexibility.

TABLE II
CGE MINIMUM W FOR 100% ROUTING ($F_c = 0.6W$)

Circuit	F_s	Channel Density	W Required by CGE	W for "Maze"	CPU Seconds
BUSC	6	9	10	15	25
DMA	6	10	10	15	59
BNRE	6	11	12	20	122
DESM	6	10	10	18	103
Z03	6	11	13	18	215

comparison because some maze routers are guided by cost functions that aid in finding good routes for connections. However, the "maze" router used here has, in effect, access to the cost function that was used to solve the global routing, which is based on balancing the densities of all routing channels. Notwithstanding, this is a constrained "maze" router because it is confined within the global route of each connection, and the comparisons are valid only in that context. The second from the right column in Table II gives the number of tracks that the maze router requires to achieve 100% routing. These results show that the maze router needs an average of 60% more tracks than CGE. This shows that resolving routing conflicts is important and that CGE addresses this issue well. Fig. 8 presents the detailed routing for circuit BUSC, with the FPGA parameters in Table II; the L blocks are shown as solid boxes, whereas the S and C blocks are dashed boxes.

C. Routing Delay Optimization for Critical Nets

Table III illustrates CGE's ability to optimize critical connections. For this experiment, several connections in circuit BNRE were marked critical. Then, CGE was used to route the circuit twice; once with CGE's critical net processing turned off, and once with it turned on. To facilitate this experiment, the FPGA was defined to have 18 tracks per channel, with four tracks hard-wired for the entire length of each channel. Connections that use the hard-wired tracks have lower routing delays because they pass through fewer switches (transistors). As Table III shows, a significant reduction in the number of switches in the critical paths was achieved.

Note that a better approach to routing delay optimization would set specific timing requirements that should be met for each critical *path* in a circuit. However, the optimization of nets and their individual connections is a reasonable compromise.

D. Memory Requirements and Speed of CGE

For the examples used here, CGE needs between 1.5 and 7.5 Mbytes of memory. As shown in the rightmost column of Table II, experimental measurements show that CGE is a linear-time algorithm, requiring from 25 to 215 SUN 3/60 CPU seconds for the smallest to the largest of the example circuits. This run-time behavior is due to the pruning procedure, which limits the number of routing alternatives that the algorithm considers for each connection.

VI. CONCLUSIONS

This paper has described a new kind of detailed routing algorithm that is designed specifically for field-programmable gate arrays. The algorithm is able to consider the side effects that routing decisions made for one connection may have on another, and thus to resolve routing conflicts and achieve a high-quality result. The algorithm can be used for a wide variety of FPGA routing architectures. It can route relatively large FPGA's in very close to the absolute minimum number of tracks as determined by global routing, and is capable of optimizing the routing delays of time-critical connections.

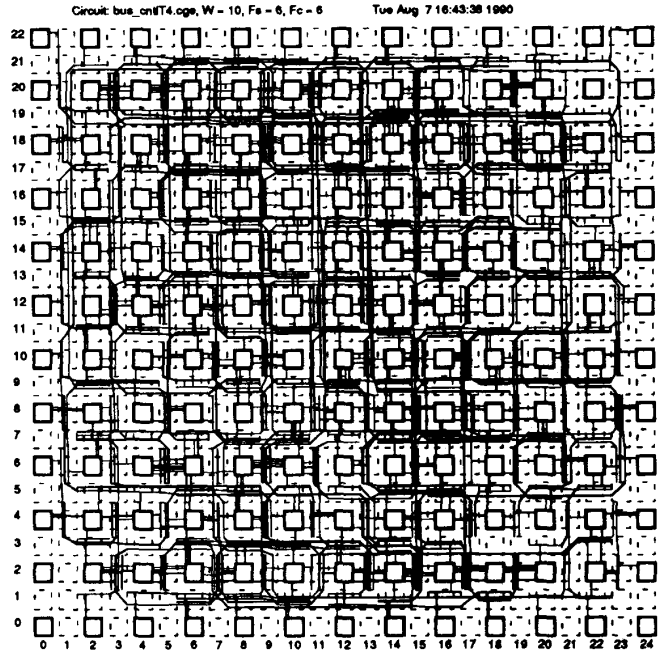


Fig. 8. The detailed routing of circuit BUSC.

TABLE III
CRITICAL CONNECTION ROUTING DELAY OPTIMIZATION

Name of Net	No. Switches Without Critical Processing	No. Switches with Critical Processing
#143	15	5
#144	14	4
#220	10	3
#280	15	2
#351	15	4

ACKNOWLEDGMENT

The authors would like to thank D. Lewis and P. Chow for suggestions on the design of the router.

REFERENCES

[1] W. Carter *et al.*, "A user programmable reconfigurable gate array," in *Proc. 1986 Custom Integrated Circuits Conf.*, May 1986, pp. 233-235.

[2] H. Hsieh *et al.*, "A 9000-gate user-programmable gate array," in *Proc. 1988 Custom Integrated Circuits Conf.*, May 1988, pp. 15.3.1-15.3.7.

[3] H. Hsieh *et al.*, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proc. 1990 Custom Integrated Circuits Conf.*, May 1990, pp. 31.2.1-31.2.7.

[4] A. El Gamal *et al.*, "An architecture for electrically configurable gate arrays," *IEEE J. Solid-State Circuits*, vol. 24, pp. 394-398, Apr. 1989.

[5] M. Ahrens *et al.*, "An FPGA family optimized for high densities and reduced routing delay," in *Proc. 1990 Custom Integrated Circuits Conf.*, May 1990, pp. 31.5.1-31.5.4.

[6] S. C. Wong, H. C. So, J. H. Ou, and J. Costello, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proc. 1989 Custom Integrated Circuits Conf.*, May 1989, pp. 5.8.1-5.8.4.

[7] Plessey Semiconductor ERA60100 preliminary data sheet.

[8] C. Marr, "Logic array beats development time blues," *Electronic System Design Magazine*, pp. 38-42, Nov. 1989.

[9] K. Kawana *et al.*, "An efficient logic block interconnect architecture for user programmable gate array," in *Proc. 1990 Custom Integrated Circuits Conf.*, May 1990, pp. 31.3.1-31.3.4.

[10] *AMD MACH 1 and MACH 2 Device Families Preliminary Data Sheets*, Advanced Micro Devices, 1990.

[11] A. Gupta *et al.*, "A user configurable gate array using CMOS-EPROM technology," in *Proc. 1990 Custom Integrated Circuits Conf.*, May 1990, pp. 31.7.1-31.7.4.

[12] *Plus Logic FPGA2020 Preliminary Data Sheet, 1990.*

[13] Quicklogic, "An introduction to Quicklogic's pASIC devices and SpDE development environment," *Data Sheet from Quicklogic*, Apr. 1991.

[14] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, pp. 1281-1304, Oct. 1981.

[15] B. Preas and M. Lorenzetti, Eds., *Physical Design Automation of VLSI Systems*, Menlo Park, CA: Benjamin Cummings, ch. 5.

[16] C. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346-365, Sept. 1961.

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

[18] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented channel routing," in *Proc. 27th Design Automat. Conf.*, June 1990, pp. 567-572.

[19] S. Brown, J. Rose, and Z. G. Vranesic, "A detailed router for field-programmable gate arrays," in *Proc. Int. Conf. Computer Aided Design*, Nov. 1990, pp. 382-385.

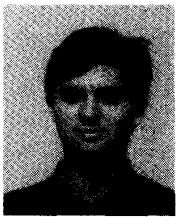
[20] J. Rose, "Parallel global routing for standard cells," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1085-1095, Oct. 1990.

[21] J. Rose and S. Brown, "The effect of switch box flexibility on routability of field-programmable gate arrays," in *Proc. 1990 Custom Integrated Circuits Conf.*, May 1990, pp. 27.5.1-27.5.4.

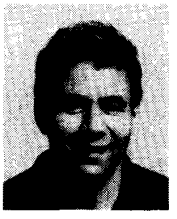
[22] J. Rose and S. Brown, "Flexibility of interconnection structures in field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, pp. 277-282, Mar. 1991.

[23] J. S. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1217-1225, Oct. 1990.

[24] *The Programmable Gate Array Data Book*. Xilinx Co., 1989.

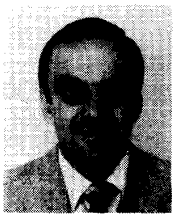


Stephen Brown (S'90) received the B.Sc.Eng. degree from the University of New Brunswick in 1985 and the M.A.Sc. degree from the University of Toronto in 1987, both in electrical engineering. Presently, he is a Ph.D. candidate in the Department of Electrical Engineering at the University of Toronto, Canada. His research interests include CAD and architecture of VLSI Systems.



Jonathan Rose (S'80-M'86) received the B.A.Sc. degree in engineering science in 1980 and the M.A.Sc. and Ph.D. degrees in electrical engineering in 1982 and 1986, respectively, from the University of Toronto, Toronto, Canada.

During the summer of 1983, he was with Bell-Northern Research Ltd., Ottawa, in the Integrated Circuits CAD/CAM group. From 1986 to 1989, he was a Research Associate in the Computer Systems Laboratory at Stanford University. In 1989, he joined the faculty of the University of Toronto, where he is currently an Assistant Professor of Electrical Engineering. His research interests include CAD and architecture for field-programmable gate arrays, automatic layout, and parallel CAD algorithms.



Zvonko G. Vranesic (S'67-M'68-SM'84) received the B.A.Sc., M.A.Sc., and the Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Canada, in 1963, 1966, and 1968, respectively.

From 1963 to 1965 he worked as a design engineer for the Northern Electric Company Ltd., Bramalea, Ontario, Canada. In 1968 he joined the faculty of the Departments of Electrical Engineering and Computer Science at the University of Toronto, where he is now a Professor. During the academic years 1977/78 and 1984/85 he was a Senior Visitor in the Computer Laboratory at the University of Cambridge, England, and at the Institut de Programmation of the University of Paris 6, France. His research interests include computer architecture, VLSI systems, fault-tolerant computing, local area networks, and many-valued switching systems.

Dr. Vranesic is a member of the Association of Professional Engineers of Ontario. He was the Chairman of the 3rd International Symposium on Multiple-Valued Logic in 1973 and of the 18th International Symposium on Computer Architecture in 1991.