3 A Detailed Router for Field-Programmable Gate Arrays

3.1 Introduction

This Chapter presents a new kind of detailed routing algorithm that has been designed specifically for FPGAs. The algorithm is unique in that it approaches this problem in a general way, allowing its use over a wide range of different FPGA routing architectures [Brow90] [Brow91]. This feature is used in Chapter 4, where the router is employed to investigate the effect of the flexibility of routing architectures.

Detailed routing for FPGAs can be more difficult than classic detailed routing [Aker72] [Souk81] [Loren89] because connections are made using wiring segments that are already in place and joins between segments are possible only at pre-determined places where routing switches exist. In some FPGA routing architectures the amount of connectivity available is low, which places exacting limitations on the number of routing choices for a connection. Wherever two or more connections pass through a common routing channel, there may be competition for the routing resources in that channel. In FPGAs that have limited connectivity, resolving such competitions is essential in order to achieve 100 percent routing completion. The algorithm described here, called the Coarse Graph Expansion (CGE) detailed router for FPGAs, addresses the issue of scarce 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 FPGAs with various routing architectures. The results show that CGE is able to route relatively large FPGAs 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.

This chapter is organized as follows: Section 3.2 motivates the development of an FPGA-specific router, Section 3.3 presents the model used for the FPGA, Section 3.4 defines the detailed routing problem, Section 3.5 describes the CGE routing algorithm, Section 3.6 presents the results from tests of the router, and Section 3.7 gives concluding remarks.

3.2 Motivation

A key problem in the detailed routing of FPGAs is that routing choices made for one connection may unnecessarily block another. Consider Figure 3.1, 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. Note that in a regular VLSI channel with full customization of mask layers, this scenario is not a problem because any of segments 1, 2, or 3 could be used for any of connections A, B, or C. Although this is a simple example, it illustrates the essence of the problems that occur because of limited routing options in FPGAs.

Common approaches used for detailed routing in other types of devices are not suitable for FPGAs. Maze routers [Lee61] are ineffective because, as shown in Chapter 2, they are inherently sequential and so, when routing one connection, they cannot consider

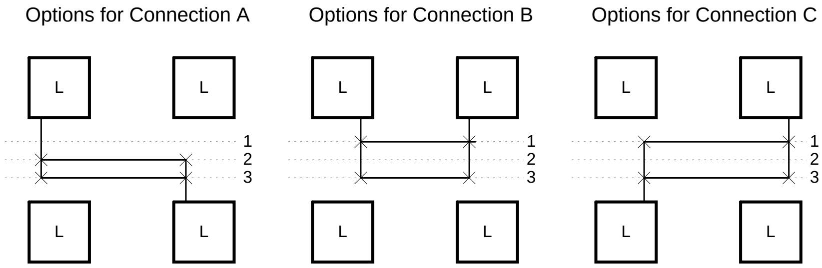

Figure 3.1 - *Routing Conflicts*

the side-effects on other connections. Channel routers [Hash71] are not appropriate because the detailed routing problem in FPGAs cannot generally be subdivided into independent channels. Note that a channel routing algorithm is used in [Green90] for Actel-like FPGAs [ElGa88]. This is possible for these types of FPGAs because the logic cells are arranged in rows separated by routing channels and the routing switches are such that each vertical wiring segment (from a logic cell pin or from another channel) can be connected to any horizontal wiring segment that it crosses in a channel. This routing flexibility cannot be assumed, in general, for an FPGA.

3.3 The FPGA Model

Since the primary purpose of this algorithm is to provide a means of investigating FPGA routing architectures, an appropriate model must be defined for the FPGA. The model that has been chosen has a two-dimensional array of logic cells interconnected by vertical and horizontal routing channels, similar to [Cart86]. Note that this model is much more flexible than the FPGA presented in [Cart86] because it allows the amount of routing resources to be changed over a wide range. The model comprises three major parts: the logic cells (L), Connection blocks (C), and Switch (S) blocks, as shown in Figure 3.2. The logic cells house the combinational and sequential logic that form the functionality of a circuit. In general, a logic cell has a number of pins that may each connect to the four adjacent C blocks. The FPGA's I/O cells appear as logic cells that are 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 logic cell pins to the routing channels, via programmable switches. Depending on the topology of the C block, each logic cell 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

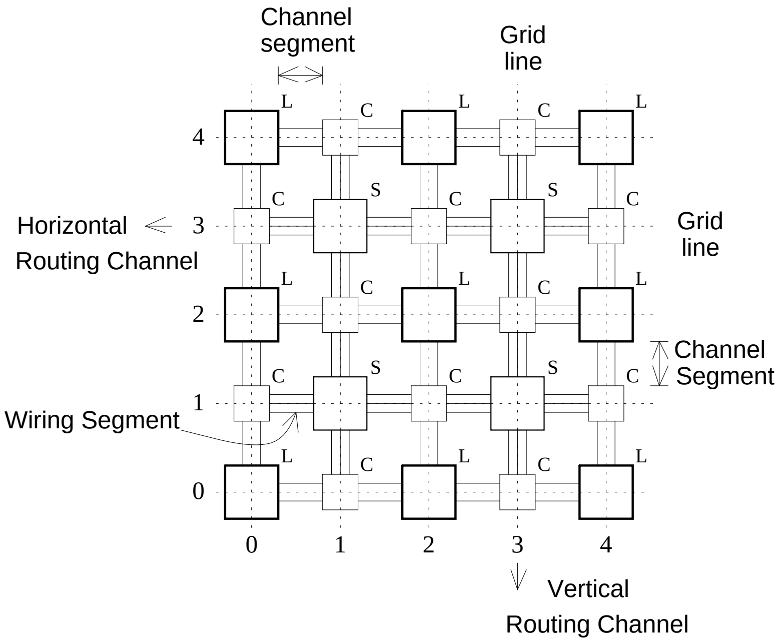

Figure 3.2 - *The FPGA Model*

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 hardwired wiring segments than if it passes through switches.

In Figure 3.2, each logic cell 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 overlayed on the FPGA is used in this chapter as a means of describing the connections to be routed.

3.4 General Approach and Problem Definition

FPGA routing is a complex combinatorial problem. The general approach taken 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, that was described in Chapter 2. The global router divides multi-point 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 coarse route for each connection by assigning it a sequence of channel segments. Figure 3.3a 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 cell at grid location 2,2 to another at 4,4. The global route is called a *coarse graph*, $G(V,A)$, where the logic cell at 2,2 is referred to as the root of the graph and the logic cell at 4,4 is called the leaf. The vertices, *V*, and edges, *A*, of $G(V,A)$ are identified by the grid of Figure 3.2. Since the global router splits all nets into two-point connections, the coarse graphs always have a fan-out of one.

After global routing the problem is transformed to the following: for each two-point connection, the detailed 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 logic cells, C blocks, and S blocks, as described in the following sections.

3.5 The CGE Detailed Router Algorithm

The basic algorithm is split into two phases. In the first phase, it records 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 are used to allow the algorithm to conserve memory and run-time while converging to its final result, as discussed in Section 3.5.3.

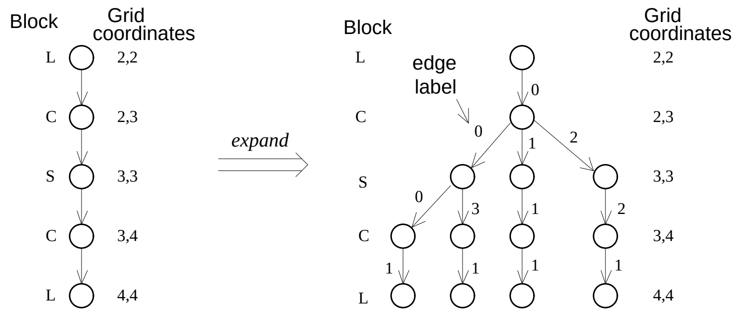

Figure 3.3a. Coarse graph, *G* Figure 3.3b. Expanded graph, *D*

Figure 3.3 - *A Typical Coarse Graph and its Expanded Graph*

3.5.1 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 labelled 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 labelled 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 in G. 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 Section 3.5.4, this black-box approach provides independence from any specific FPGA routing architecture. The result of a graph expansion is illustrated in Figure 3.3b, which shows a possible expanded graph for the coarse graph of Figure 3.3a. An expanded graph is produced by examining the routing switches and wiring segments along the path described by the coarse graph, and recording the alternative detailed routes in the expanded graph. In algorithmic form, 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 added vertex according to:

- **Expand** a C vertex in D by calling $Z=f_c(e_C,n)$. e_C is the edge in D that connects to C from its predecessor. n is the required successor vertex of C (in G) and Z is the set of edges returned by $f_c(\ )$. The call to $f_c(\ )$ adds Z to D.

Expand an S vertex in D by calling $Z = f_s(e_S, n)$. e_S is the edge in D that connects 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 to D.

Endwhile

3.5.2 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. Phase 2 proceeds as follows (as explained later in this section, the terms c_f cost and c_t cost are functions that represents the relative cost of selecting a specific detailed route (path) for a connection, and an *essential* path indicates a connection that should be routed immediately because it has only one remaining option):

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, re-expand its coarse graph - if this results in no new paths, the connection is deemed unroutable (see Section 3.5.3.1 for a discussion relating to failed connections).

Update the cost of all affected paths.

Endwhile

Note: When a wiring segment is chosen for a particular connection, it and any other wiring segments in the FPGA that are hardwired 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 hardwired to e.

3.5.2.1 Cost Function Design

Because the cost function allows it to consider all the paths at once, CGE can be said to route the connections 'in parallel'. 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 costs 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. The reason for using wiring segment demand was illustrated in Figure 3.1, where connection A should be routed with wiring segment number 2, because wiring segment number 3 is in greater demand.

2. It is used to identify a path that is *essential* for a connection. A path is called essential when it represents the only remaining option in the FPGA for a connection, because previous path selections have consumed all other alternatives.

The importance of essential wiring segments is illustrated by the example in Figure 3.4. If the router were to complete connection D first, then wiring segment number 1 or 2 would be equal candidates according to their demand, since they both appear in one other graph. However, wiring segment number 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 $(e_1, e_2, ..., e_j)$ 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 when $alt(e_j)$ is 0, e_j is an edge

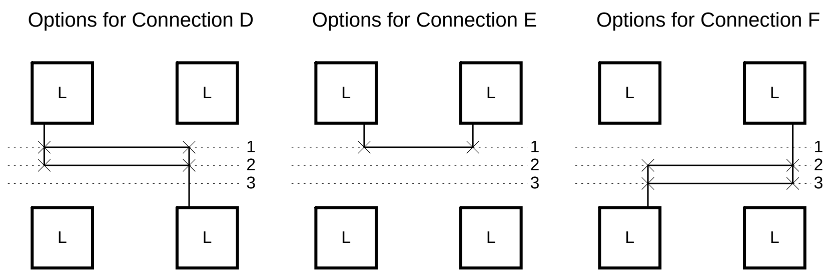

Figure 3.4 - *An Essential Wiring Segment*

that is essential to the associated connection because there are no 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.

3.5.3 Controlling Complexity

Although the above description 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 logic cell, and that the logic cells 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 $F_c\left(\frac{F_s}{3}\right)^n$ different paths from the first pin to the last logic cell, 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 are three or greater, as shown in Chapter 4, and the number of connections is large, a heuristic is employed to reduce the number of paths in the expanded graphs. Some of the paths are pruned as each graph is expanded. The pruning procedure is parameterized so that the number of paths is controlled and yet the expanded graphs still contain as many alternatives as possible. Maximizing the number of alternatives is important in the context of resolving routing conflicts. The pruning procedure is part of the graph expansion process that is described in Section 3.5.1. 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 fanout 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 one to F_c (the number of wiring segments connected to each logic cell 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 Figure 3.5. 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 to prune a vertex is based on the wiring segment that corresponds to its 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 included in the most other expanded graphs will be discarded. This helps the c_f cost function discover the wiring segments that are in the least demand. Note that this introduces an order-dependence in the routing algorithm because the paths that are pruned from each expanded graph depend on the order in which the coarse graphs are expanded.

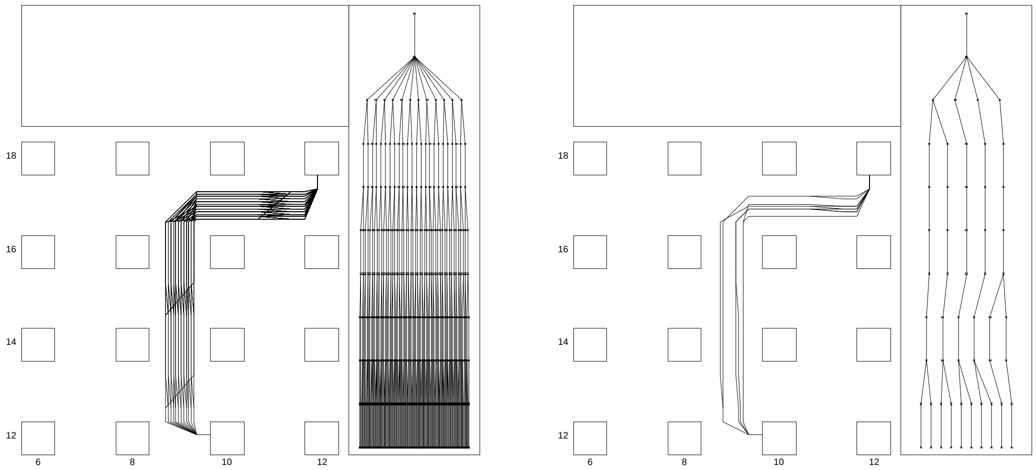

Figure 3.5 - *The Effect of Pruning*

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 the alternatives for some graph, CGE re-invokes the graph expansion process to obtain a new set of paths if some exist.

3.5.3.1 Iterative Improvements

This section 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 longer 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, in a typical routing problem, there are 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 re-routed in the next iteration. Iterations are continued until all connections are routed or until further improvements are not forthcoming. Note that at this point 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 iterative approach is a minor variation of classic rip-up and re-route schemes where individual connections would be removed and re-routed to try to resolve routing conflicts. The technique employed here allows the algorithm's cost function to solve the routing problem, but conserve memory and time where the problem is not difficult and 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. For the results that are presented in Section 3.6, K and k are set to two for the first iteration. K is increased by one for each iteration until it reaches F_c, after which k is increased by one for each subsequent iteration.

3.5.4 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 Figure 3.6, 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 3.3 can be routed by changing these isolated routines. This generality is the key that allows the router's use, in Chapter 4, as a research tool for studying routing architecture flexibility. Figure 3.6 also shows the organization of the phases of CGE and the feedback path used over multiple iterations.

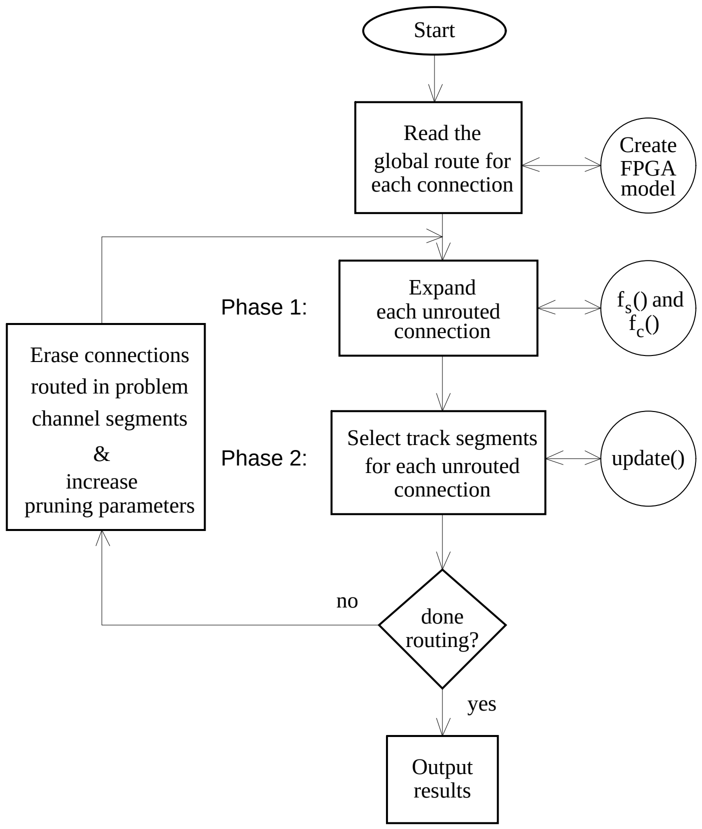

Figure 3.6 - *The Organization of CGE*

3.6 Results

CGE has been used to route several industrial circuits implemented as FPGAs. 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 3.1 gives the name, size (number of two-point connections and logic cells), source and the function of each circuit. For these results, the logic cell used is the result of a previous study [Rose89] [Rose90c], and the S and C blocks will be described in the next sub-section. Results are presented for a routing architecture similar to a commercial FPGA.

3.6.1 FPGA Routing Structures

Since the routability of an FPGA is determined by the topology and flexibility of 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 Figure 3.7a. Its flexibility is set by the parameter F_s, which defines the total number of connections offered to each wiring segment that enters the S block. For the example shown in Figure 3.7a, 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.

Circuit	#Blocks	#Conn	Source	Type
BUSC	109	392	UTD1	Bus Cntl
DMA	224	771	UTD2	DMA Cntl
BNRE	362	1257	BNR	Logic/Data
DFSM	401	1422	UTD1	State Mach.
Z03	586	2135	Zymos	8-bit Mult

Table 3.1 - *Experimental Circuits*

Figure 3.7b illustrates the test C block. The tracks pass uninterrupted through it and are connected to logic cell pins via a set of switches. The flexibility of the C block, F_c, is defined as the number of tracks that each logic cell pin can connect to. For the example shown in the figure, each logic cell pin can connect to 2 vertical tracks, and so F_c is 2.

3.6.2 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 3.2 shows the maximum channel density over all channels for each circuit. This represents a lower bound on the number of tracks per routing channel that is needed for each example. The real track requirements will depend on the flexibility of the routing architecture because there are interactions between one channel segment and another that are not accounted for in channel density measurements. The maximum flexibility has $F_s = 3W$ and $F_c = W$, where there are W tracks per channel. For the results in Table 3.2 the FPGA parameters are based on the Xilinx 3000 series [Xili89] FPGAs ($F_s = 6$, $F_c = 0.6W$). Table 3.2 gives the minimum number of tracks per channel that CGE needs in order to route 100 percent 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

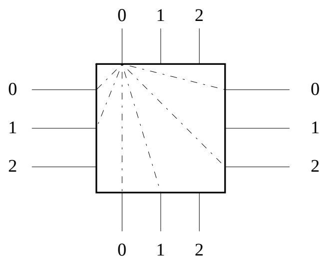

Figure 3.7a. The S block.

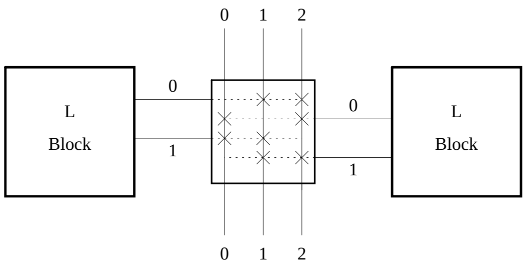

Figure 3.7b. The C block.

Figure 3.7 - *Definitions of S and C Block Flexibility*

that, although not shown, 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 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 to remain within the global route of each connection, and the comparisons are valid only in that context. The rightmost column in Table 3.2 gives the number of tracks that the ’maze’ router requires to achieve 100 percent routing. These results demonstrate that the ’maze’ router needs an average of 60 percent more tracks than CGE. This shows that resolving routing conflicts is important and that CGE addresses this issue well. Figure 3.8 presents the detailed routing for circuit BUSC, with the FPGA parameters in Table 3.2; the logic cells are shown as solid boxes, whereas the S and C blocks are dashed boxes.

Circuit	Channel density	W required by CGE	W for ’maze’
BUSC	9	10	15
DMA	10	10	15
BNRE	11	12	20
DFSM	10	10	18
Z03	11	13	18

Table 3.2 - *CGE Minimum W for 100 % routing ($F_c = 0.6W$, $F_s = 6$)*

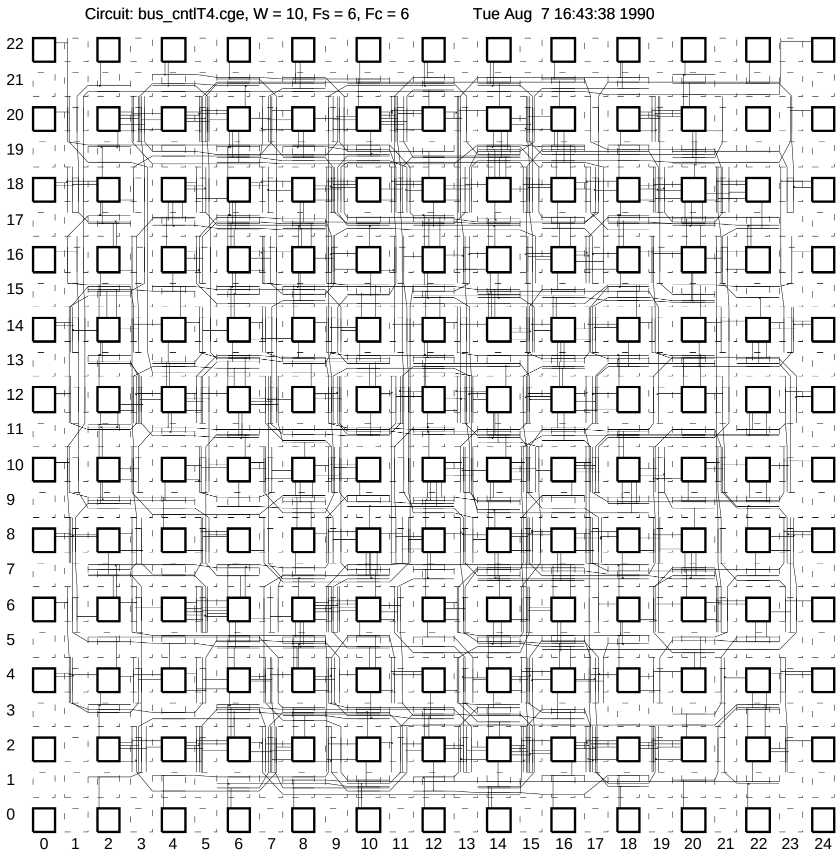

Figure 3.8 - *The Detailed Routing of Circuit BUSC*

3.6.3 Routing Delay Optimization for Critical Nets

Table 3.3 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 hardwired for the entire length of each channel. Connections that use the hardwired tracks have lower routing delays because they pass through fewer switches (transistors). As Table 3.3 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. This exercise is left as future work.

Name of net	# of switches without critical processing	# of switches with critical processing
#143	15	5
#144	14	4
#220	10	3
#280	15	2
#351	15	4

Table 3.3 - *Critical Connection Routing Delay Optimization*

3.6.4 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 Table 3.4, 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.

Circuit	#Conn	Sun 3/60 CPU sec.	msec per connection
BUSC	392	25	63
DMA	771	59	76
BNRE	1257	122	97
DFSM	1422	103	72
Z03	2135	215	99

Table 3.4 - *CGE Run-time*

3.7 Conclusions and Future Work

This chapter 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 resolve routing conflicts and achieve a high quality result. The algorithm can be used over a wide range of FPGA routing architectures. It can route relatively large FPGAs 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.

Future research should improve upon the treatment of time-critical nets. The router could be given specific timing requirements for each critical path in a circuit and should ensure that the routing delays of the nets in these paths do not exceed these times.