A Timing-Driven Global Router for Symmetrical Array Based FPGAs

Kai Zhu and D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

Abstract

In this paper, we present a timing-driven global router for symmetrical array-based architecture FPGAs. The routing resources in symmetrical array based FPGAs consist of segments of various lengths. The timing constraints are specified as delay bounds on source-sink pairs of nets. The algorithm proceeds in a hierarchical top-down manner and is able to utilize various routing segments with global consideration in order to meet the timing constraints. Furthermore, the algorithm can be extended to perform detailed routing simultaneously with global routing. Experimental results on real circuits show that the algorithm is promising in satisfying the timing constraints compared with the conventional global router.

1 Introduction

In the current technologies, the programmable switches used in Field-Programmable Gate Arrays (FPGAs) usually have significant high resistance comparing to the routing metal wires. Interconnection delays in FPGAs account for a large percentage of the total circuit delay. It is not uncommon that over 50% of circuit delay is incurred in interconnections. This situation is very different from that in traditional mask-programmable technologies. Consequently, reducing interconnection delay of FPGAs becomes critical in the effort of boosting circuit performance.

Symmetrical array architecture is used in several popular commercially available FPGAs [1, 9]. Figure 1 illustrates the basic structure of symmetrical array based FPGAs. The routing resources in Figure 1 contain routing channels in both horizontal and vertical directions. To improve circuit performance and at the same time maintain reasonable routability, the routing channels are usually segmented, that is, routing tracks are of various lengths. Shorter segments are intended to be used by shorter nets to avoid waste of routing resources. On the other hand, longer segments are designed to be used by longer nets to reduce interconnection delay. To achieve the goal of improving circuit performance without much sacrifice on routability, it is essential for a routing algorithm to utilize these various routing segments effectively. A timing-driven routing algorithm must have the ability of using different segments for different nets accordingly.

Several routing algorithms for symmetrical array based FPGAs have been reported in the literatures [2, 4, 12, 15, 17]. However, Most of these algorithms aim at routability only, without considering timing constraints [2, 15, 17]. Researches on timing-driven routing for symmetrical array based FPGAs are very limited. In [4], a new routing strategy for improving FPGA performance is proposed. In addition to global and detailed routing stages, a *segment assignment* stage is introduced between global and detailed routing. Routing segments along the global routing paths are chosen for nets based on a greedy strategy in an attempt to reduce routing delays as much as possible. Another routing algorithm considering circuit delay is described in [12]. The algorithm is an extension of the detailed routing algorithm of [2] to handle the selection of different segments for detailed routing. A notable drawback of the approaches in [4] and [12] is that the choice of segments is restricted to the routing paths defined by a conventional global router, and thus the routing results are very unlikely able to utlize the segments efficiently. Furthermore, the timing constraints, which are usually specified as arrival times on primary inputs and required times on the primary outputs [8], are not completely considered in the routing algorithms.

In this paper, we present a timing-driven global routing algorithm for symmetrical array based

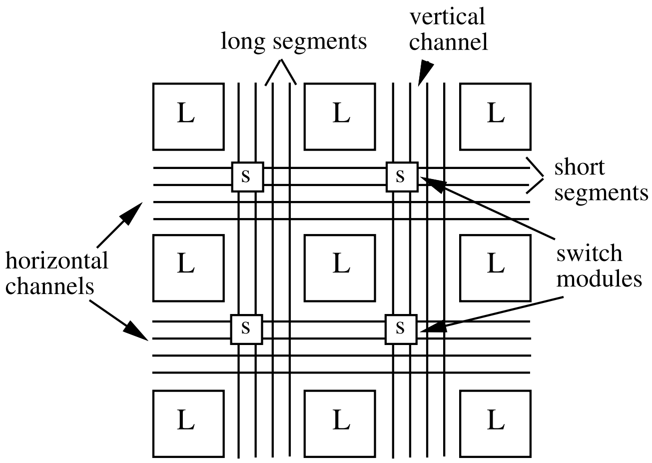

Figure 1: Symmetrical array architecture model.

FPGAs. The algorithm has the following distinct features comparing to the previous approaches. First, the utilization of routing resources are considered with a global viewpoint. The routing algorithm uses a hierarchical top-down approach and consider all the routing resources available in assigning the routing paths for nets. Second, the algorithm explicitly includes timing constraints as its inputs and aims at routing all the nets to meet the timing constraints all together. To perform timing-driven routing, the timing constraints are transferred into the delay bounds on source-sinks of nets. The routing algorithm can be used as a direct follow-up to the timing-driven placement and to honor the timing constraints generated by timing-driven placement [7, 14]. The third distinct feature of our algorithm is that detailed routing can be performed simultaneously with global routing. All nets are routed at the same time and net ordering problem can be avoided at both global and detailed routing level.

The rest of the paper is organized as follows. The routing resource architecture of symmetrical array based FPGAs is described in more details in Section 2 to provide the needed information for the routing algorithm. Based on the routing resource architecture, the timing-driven global routing problem is specified in Section 3. In Section 4, the timing-driven global routing algorithm is described and the incorporation with detailed routing is discussed. Experimental results on real circuits are presented in Section 5, with comparison to traditional routing algorithm. Finally, the paper is concluded in Section 6.

2 Routing Resource Architecture

Conceptuelly, a symmetrical array based FPGA consists of three types of components: *logic modules* (labeled as L), *channels*, and *switch modules* (labeled as S). See Figure 1. The routing resources in a symmetrical array based FPGA include routing channels and switch modules, and we will discuss their detailed structures in turn.

A horizontal (vertical) channel is the routing space between two adjacent rows (columns) of logic modules. A channel consists of a set of *subchannels*. A subchannel contains a set of uniform-length routing segments, and the routing segments in different subchannels in the same channel have different lengths. Assume logic modules are sqaures of the same size. The minimum length unit of a segment spans a logic module. Figure 2 shows an example of a horizontal channel which contains three subchannels. The subchannel on the top contains segments of one unit (called single lines), the subchannel in the middle contains segments of two units (called double lines), and the subchannel at the bottom contains segments of the entire channel (called long lines). This model of channel segmentation is used in several commercial FPGAs [1, 9]. Note that the routing segmentation in the symmetrical array architecture is relatively regular comparing to the the channel segmentation in the row-based architecture [6].

Connections between two crossing vertical and horizontal channels are established by *switch modules*. A switch module is a rectangular box. A track in a channel accessing the switch module on one side can be connected to a set of tracks on other three sides of the switch module. There are two types of switch modules: *switch block* and *switch matrix* [18]. See Figure 3. In a switch block, a programmable switch is required to connect any two tracks, even for the two tracks on the same straight line. The switch block is usually used for connections between short segments, such as segments of lengths of one unit or two units. A switch matrix is a grid of horizontal and vertical tracks. Two crossing tracks can be connected at the intersection point by a switch. However, if a track is not separated into segments by switches, a net can route the switch matrix on that track without going through any programmable switch. Switch matrix is usually used for connections between long segments and other types of segments.

3 Problem Formulation

A circuit consists of a set of logic modules and a set of nets, The timing constraints on the circuit are specified as the *arrival times* at the primary inputs or outputs of storage elements, and the *required times* at the primary outputs or the inputs to storage elements [8]. In timing-driven physical layout,

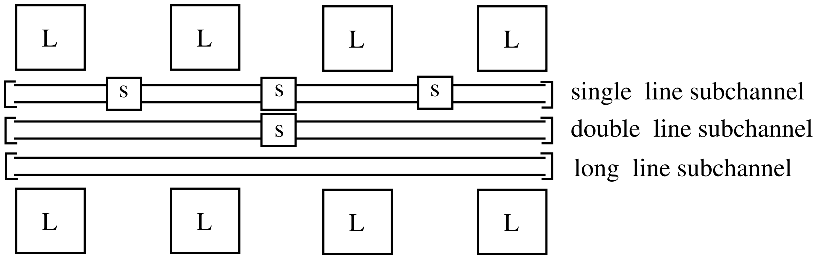

Figure 2: Detailed structure of a horizontal channel.

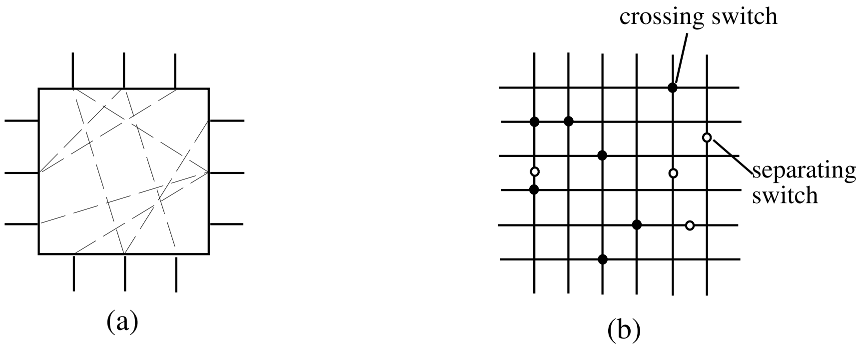

Figure 3: Switch module. (a) switch block model. (b) switch matrix model.

the timing constraints can be transferred into the delay bounds on nets [5, 7]. Every net consists of a *source* pin and a set of *sink* pins. The delay bounds on a net are specified as the delay bounds from the source to each of the sinks of the net. The goal for timing-driven routing is to route the nets and at the same time satisfy the net delay bounds.

To perform timing-driven routing, the net delay bounds need to be transferred into physical layout constraints. It is shown in [10] that the number of segments (and so the number of switches) a net has to pass through is the most critical factor in controlling routing delay. The routing delay model based on the number of switches a net passes through is sufficiently accurate for the purpose of timing-driven routing. Based on such a routing delay model, the delay bounds of a net can be therefore transferred into the upper bounds on the number of switches used between source-sink pairs of the net. Note that for the regular channel segmentation as described in Section 2, if the global routing path of a net is specified in terms of subchannels, then the number of switches used along the routing path of the net is known.

For symmetrical array based FPGA, the timing-driven global routing problem is as follows. The inputs are the routing resources as described in the previous section, a netlist of the placed circuit, and the net delay bounds specified as the upper bounds on the number of switches used between each source-sink pair of nets. The timing-driven global routing is to determine the global routing paths of nets which are specified as the subchannels to be used in the routing, and at the same time to satisfy the net delay bounds for as many source-sink pairs as possible.

In the following discussion, routing delay and delay bounds are measured in number of switches.

4 The Algorithm

4.1 Overview

The overall approach of the routing algorithm follows the hierarchical top-down strategy as described in [11, 13]. The algorithm was originally designed for sea-of-gate and macro-cell architectures. We extend the algorithm to symmetrical array based FPGAs.

The original algorithm in [11, 13] proceeds as follows. Starting from the entire circuit, a cut line is chosen in a certain way (such as max-cut) to divide the circuit into two parts. On the cut line there are *routing sections* which represents routing spaces on the chip acrossing the cut line. See Figure 4 for an illustration of terminologies. A routing section is usually a channel and has a routing capacity, usually specified by the channel width. Each pair of a connection crossing the cut line and a routing section is assigned a cost. The cost is designed to optimize the objective function, such as total wire length and channel densities. The global routing at this level of top-down hierarchy assigns each connection crossing the cut line to a routing section, and this is solved by using the linear assignment algorithm [16]. After finishing global routing at this hierarchical level, both subcircuits created by the cut line are routed by the same method recursively. The algorithm stops when the subcircuits are small enough so that they can be solved easily.

To adapted to timing-driven global routing for symmetrical array based FPGAs, we make the following extensions to the above algorithm.

1. Cut lines are either horizontal or vertical. A routing section on a cut line is a subchannel, instead of a channel. Therefore, the global routing will automatically choose appropriate subchannels for nets at each hierarchical routing level.

2. For each net, a routing tree need to be constructed. Here, a routing tree of a net is a spanning tree of the complete graph defined on all the pins of the net. The minimum spanning tree, while suitable for wire length minimization, may not be good for timing-driven routing. Thus we need to construct routing trees with considerations of net delay bounds.

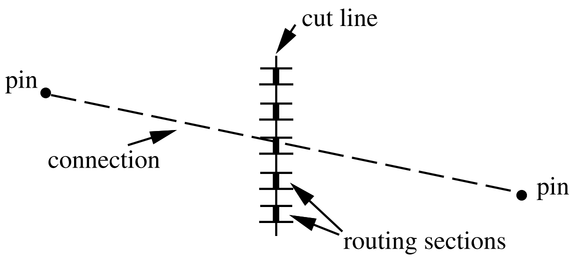

Figure 4: Illustration of terminologies used in the hierarchical routing.

3. After constructing the routing tree, the net delay bounds need to be distributed among the edges of the routing tree in order to perform hierarchical routing.

4. A new cost function for the linear assignment should be designed to include timing constraints. The routing resource architecture of FPGAs should be taken into account in the cost function.

5. At each hierarchical level, the delay bounds for connections crossing the cut line need to be redistributed among the subcircuits for the next level of hierarchical routing. The redistribution of delay bounds should facilitate the routing at the following hierarchical routing levels.

We will discuss the modifications 2 to 5 in details in the following subsections.

4.2 Delay-Bounded Routing Trees

Intuitively, we want to construct a routing tree for every net such that the routing cost is minimized and source-sink delays are within the delay bounds. Routing cost usually includes wire lengths and routing congestions. As in conventional technology, geometric distance is a suitable measurement for the routing cost. However, due to the segmented structure of routing resources in FPGAs, the routing delay is in general not proportional to the geometric distance. Therefore, there should be two costs associated with every edge of the routing tree, one for routing cost and the other one for delay cost.

We first define a few terms. Let $\mathcal{N}$ be a net connecting a set of pins, $V = \{s,\ t_1, \ldots, t_{|V|-1}\}$, where s is the source pin and t_i's are the sink pins. The undirected graph $G = (V, E)$ associated with the net $\mathcal{N}$ is a complete graph on V. A routing tree T of $\mathcal{N}$ is a spanning tree of G. The routing cost on an edge $e \in E$ is denoted by $r(e)$. The delay cost on an edge $e \in E$ is denoted by $d(e)$. For each source-sink pair s and $t_i, \forall t_i \in V$, there is delay bound $U(t_i)$. Given a subgraph G' of G, the shortest path delay between source s and sink t_i measured in delay cost function d is denoted by $D_{G',d}(s,t_i)$. Note that $D_{G,d}(s,t_i)$ is the smallest possible delay between s and t_i on G. The delay bound $U(t_i)$ therefore should not be greater than $D_{G,d}(s,t_i)$.

The routing tree problem, referred to as the *delay-bounded routing tree problem* (DBRT), is stated as follows.

Problem 1 (DBRT) *Given a complete graph $G = (V, E)$, routing cost $r(e)$ and delay cost $d(e), \forall e \in E$, and delay bounds $U(t_i), \forall t_i \in V$, construct a routing tree T on G such that $D_{T,d}(s,t_i) \leq U(t_i)$ for every source-sink pair, and the routing cost $\sum_{e \in T} r(e)$ of T is minimized.*

There are trivial solutions for two special cases of the DBRT problem. If $U(t_i) = D_{G,d}(s,t_i)$, for all source-sink pairs, the routing tree is the union of the shortest paths of all source-sink pairs on G measured in delay cost d. If $U(t_i) = \infty$ for all source-sink pairs, the routing tree is the minimum spanning tree on G measured in routing cost r. The complexity of the general DBRT problem, however, is not known.

For timing-driven global routing, we use the following cost functions. The routing cost of an edge $e = (u, v)$ is defined as the Manhattan distance

$$r(e) = |x_u - x_v| + |y_u - y_v|, \tag{1}$$

where x_u, y_u and x_v, y_v are the coordinates of the pins u and v, respectively. The delay cost function is defined as

$$d(e) = max\{d_{min}, \alpha r(e)\}, \tag{2}$$

where d_{min} is the minimum number of switches required to connect u and v (we will discuss d_{min} in details later), and $0 < \alpha \leq 1$ is a constant which satisfies the following inequality

$$D_{G,d}(s,t_i) \leq U(t_i), \quad \forall t_i \in V, \tag{3}$$

and at the same time maximize the left-hand side of the above inequality. Recall that the minimum length unit of routing segment is the span of one logic module, thus the routing cost function $r(e)$ in the delay function $d(e)$ in equation (2) represents the maximum possible delay between the two pins u and v. The constant α is used to capture the usage of longer routing segments for satisfying stringent delay bounds. Smaller α gives smaller delay cost $d(e)$, and that implies the routing between u and v need to use longer segments. The value of α is computed by inequality (3) on all source-sink pairs of the net. Thus, for the source-sink pairs for which the inequality (3) is strictly less than the corresponding delay bound $U(t_i)$, there will be more alternatives for the routing to meet the delay bounds.

Algorithm: DBRT
Input: $G = (V, E); U(t_i), \forall t_i \in V$.
Output: $T = (V', E')$.
begin
1. $V' \leftarrow \{s\}; E' \leftarrow \emptyset$;
2. **while** $(|V'| < |V|)$ **do begin**
3. pick an edge $e = (u, v) \in E$, $u \in V'$ and $v \in V \setminus V'$ such that
4. $r(e)$ is minimum and $D_{T,d}(s, u) + d(e) \leq U(v)$;
5. **if** such a v is found **then**
6. $V' \leftarrow V' \cup \{v\}; E' \leftarrow E' \cup \{(u, v)\}$;
7. **else**
8. $V' \leftarrow V' \cup \{\text{nodes in } PATH_{G,d}(s, v)\}$; $E' \leftarrow E' \cup \{\text{edges in } PATH_{G,d}(s, v)\}$;
9. **end**;
end.

Figure 5: Algorithm for constructing DBRT.

We now discuss delay lower bound d_{min}. To compute d_{min}, we construct a graph H as follows. Represent each pin and routing segment by a vertex. Connect two vertices by an edge with unit weight if there is a switch between the two vertices. That is, the switch either connects a pin with a routing segment, or connects a routing segment to another segment through switch module. Such a graph H is referred to as *connectivity graph*. The delay bound d_{min} is the length of the shortest path between u and v on H, and can be computed by the shortest path algorithm [3]. Notice that since any pin need to connect to a routing track, it requires at least two switches to connect two pins. Thus $d_{min} \geq 2$. The connectivity graph will be used again in the cost function for linear assignment in Section 4.4.

With the cost functions (1) and (2), we use the following algorithm to construct the delay-bounded routing tree. The algorithm follows the approach of Prim's minimum spanning tree algorithm [3] and is shown in Figure 5. Let $T = (V', E')$ be the routing tree to be constructed. The construction of routing tree T starts from the source $V' = \{s\}$ (line 1). The tree T is then expanded by including nodes not within V' incrementally, until all the nodes in V are included, i.e. $V' = V$. The procedure of routing tree expansion is first to look for two nodes u and v, where $u \in V'$ and $v \in V \setminus V'$, such that the routing cost of $e = (u, v)$ is minimum and the delay bound from s to v is satisfied (lines 3-6). If no such a node v is found, then the shortest path from s to v on G measured in delay cost d, denoted by $PATH_{G,d}(s, v)$, is added to the routing tree (line 8). Note that by the condition (2), the delay bound $U(v)$ can always be satisfied by choosing $PATH_{G,d}(s, v)$.

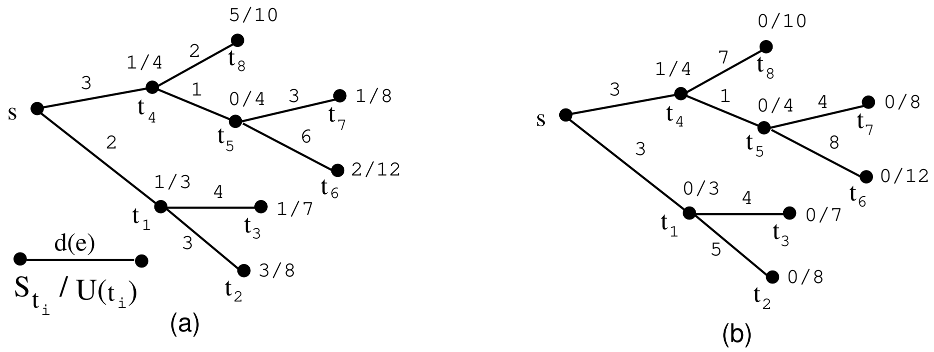

Figure 6: (a) Slack computation. (b) Delay bound distribution.

Since in each iteration of the while loop, at least one vertex is added to the routing tree T, thus the algorithm will stop after at most $|V|-1$ iterations. In each iteration, it requires to compare at most $O(|V|^2)$ pairs of vertices for routing costs and delay costs. Also, the running time of shortest path algorithm is $O(|V|^2)$ [3]. Thus the running time complexity of the algorithm is $O(|V|^3)$.

4.3 Delay Bound Distribution

In this section, we discuss the distribution of delay bounds on the source-sink pairs to the edges of the routing tree.

We first define a few terms. In a routing tree T, there is only one edge between any sink node and its parent node. Thus we can define an edge e_{t_i} in T as the edge between the sink t_i as its parent node. For any sink t_i, there is a uniqe path from the source s to t_i in the routing tree T. Let $U(e_{t_i})$ be the delay bound allocated to the edge e_{t_i}. Define the *slack* for a sink t_i, denoted by S_{t_i}, as the difference between the delay bound $U(t_i)$ and the sum of delay bounds of all edges along the path from s to t_i. That is $S_{t_i} = U(t_i) - \sum_{e \in \text{path}(s,t_i)} U(e)$, where $\text{path}(s,t_i)$ is the path from s to t_i in T. The slack for the source s is defined as zero. Figure 6(a) shows a routing tree, delay bounds on the sinks, a distribution of delay bounds on the edges, and the corresponding slacks.

The delay bound distribution is to allocate the delay bounds on the edges of the routing tree T such that for every sink t_i in T, the slacks $S_{t_i} \geq 0$.

The algorithm for delay bound distribution is listed in Figure 7. First, the algorithm allocates for every edge e_{t_i} in T an initial delay bound $U(e_{t_i}) = d(e_{t_i})$, i.e., the associated edge delay cost (line 1). Note that with such initial delay bound allocation, the slacks of all sinks in the routing tree T constructed by the delay-bounded routing tree algorithm in section 4.2 are guarantee be non-negative. After the initial allocation, it is possible that the delay bounds on the edges can be

Algorithm: DELAY BOUND DISTRIBUTION
begin
1. $U(e_{t_i}) \leftarrow d(e_{t_i}), \forall t_i \in V$;
2. **while** (there exist relaxable sinks) **do begin**
3. Compute slacks $S_{t_i}, \forall t_i \in V$;
4. Label every sink $t_i \in V$ as either relaxable or unrelaxable;
5. Pick a relaxable sink t_i with minimum S_{t_i}, and $U(e_{t_i}) \leftarrow U(e_{t_i}) + S_{t_i}$;
6. **end**;
end.

Figure 7: Algorithm for delay bound distribution.

further relaxed under certain conditions. Let T_{t_i} be the subtree in T rooted at sink t_i. A sink t_i is a *relaxable sink* if the slack of every sink in T_{t_i} is greater than zero. The following is the necessary and sufficient condition to further relax edge delay bounds.

Theorem 1 *The delay bound on an edge e_{t_i} can be further relaxed if and only if the sink t_i is a relaxable sink.*

Proof: (If) Suppose that the sink t_i is relaxable. By definition, all sinks in the subtree T_{t_i} must be also relaxable. Let t_i' be the sink in T_{t_i} with the minimum slack among all the sinks in T_{t_i}. The delay bound of e_{t_i} can be increased by $S_{t_i'}$ and no sink in T will have negative slacks. Thus the delay bound of e_{t_i} can be relaxed. It should be noticed that the existence of sinks with positive slacks is not sufficient to relax the delay bounds on edges. For example, the sink t_4 in Figure 6(a) has slack 1, but t_4 is not a relaxable sink, because one of t_4's children, sink t_5, has zero slack.

(Only if) Suppose a sink t_i is not relaxable. By definition, either $S_{t_i} = 0$ or there is at least one sink in the subtree T_{t_i} which has zero slack. If the delay bound for e_{t_i} is increased, there must be at least one sink in T_{t_i} which will have negative slack. Thus the delay bound for e_{t_i} cannot be relaxed. $\square$

All the relaxable sinks in routing tree T can be easily identified in linear time $O(|V|)$. After allocating the initial delay bounds on edges, the algorithm picks a relaxable sink t_i, if there is any, with minimum slack, and relaxs the the delay bound on e_{t_i} by the amount of S_{t_i} (lines 3-5). The process repeats until there is no relaxable sink in T. Figure 6(b) shows the routing tree after applying the algorithm on the original routing tree in Figure 6(a). Notice that the slack of t_4 does not change, since t_4 is not a relaxable sink.

Each of the steps 3 and 4 can be done in one traversal of the routing tree T. In each iteration, the number of relaxable sinks is reduced at least by one. Thus there are at most $|V|-1$ iterations in the while loop, and the running time of the algorithm is $O(|V|^2)$.

4.4 Cost Function for Linear Assignment

A cut line at a routing hierarchical level cuts through a set of subchannels. A routing section on the cut line is a subchannel. Let C_{ij} be the cost of routing connection i through subchannel j. The cost C_{ij} consists of three terms:

$$C_{ij} = C_{ij}^{(1)} + C_{ij}^{(2)} + C_{ij}^{(3)}.$$

The first term $C_{ij}^{(1)}$ depends on whether the connection i can reach the subchannel j:

$$C_{ij}^{(1)} = \begin{cases} 0 & \text{if connection } i \text{ can reach subchannel } j, \\ \infty & \text{otherwise.} \end{cases}$$

The reachability can be determined by a breadth-first search on the connectivity graph as defined in section 4.2.

The second term intends to utilize the routing segments evenly according to the connection length and its delay bound:

$$C_{ij}^{(2)} = a \left\| \frac{l_i}{U_i} - L_j \right\|,$$

where l_i is the Manhattan distance of the connection i, U_i is the delay upper bound of connection i, L_j is the length of routing segments in the subchannel j, and $a > 0$ is a constant. Note that $\frac{l_i}{U_i}$ is the average Manhattan distance in routing the connection using at most one switch. Thus the closer $\frac{l_i}{U_i}$ to the length of segments in a subchannel, the lower the cost is.

A bend in routing a net requires at least one switch. Thus it is preferable for routing a net with fewer bends in order to meet the delay bounds. The third term $C_{ij}^{(3)}$ of the cost function is defined based on this consideration. The shape of $C_{ij}^{(3)}$ is illustrated in Figure 8. There are two subchannels on the cut line through which the connection will has the minimum possible number of bends. Thus

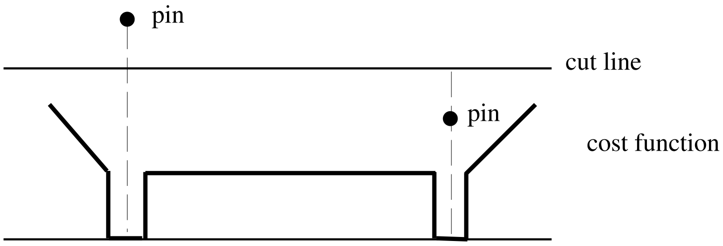

Figure 8: The third term $C_{ij}^{(3)}$ of the cost function for linear assignment.

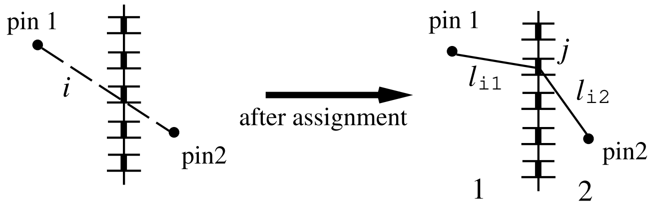

Figure 9: Delay bound redistribution: simple case.

C_{ij} has the minimum value at these two subchannels. The cost of routing through the subchannels within the bounding box of the connection is a constant greater than the minimum cost. The costs routing through subchannels outside the bounding box of the connection are proportional to the Manhattan distances of the subchannels to the bounding box.

4.5 Delay Bound Redistribution

After assigning connections to the subchannels on the cut line, the delay upper bounds for the connections need to be redistributed among the new connections created by the cut line.

We first consider a simple case where the connection crossing the cut line does not share pins with any other connections crossing the cut line. See Figure 9. Assume the connection i is assigned to subchannel j. Let $i1$ and $i2$ be the new connections created on the two sides of the cut line. Let l_{i1} and l_{i2} be the Manhattan distance of $i1$ and $i2$ to the subchannel j, respectively. The delay bound U_i of the connection i is redistributed among the new connections $i1$ and $i2$ proportional to their distances to the assigned subchannel. That is

$$U_{i1} = \frac{l_{i1}}{l_{i1} + l_{i2}} U_i,$$

and

$$U_{i2} = \frac{l_{i2}}{l_{i1} + l_{i2}} U_i,$$

where U_{i1} and U_{i2} are the delay bounds for the new connections $i1$ and $i2$, respectively.

In general, connections of same net crossing the cut line may share pins. If connections which share pins are not assigned to the same subchannel, delay bounds are redistributed for every connection independently as discussed in the above simple case. If several connections which share pins are assigned to same subchannel, the delay bound redistribution will be different and we use an example shown in Figure 10 to illustrate the redistribution procedure. Three connections i, j and k belong to a same net. Connections i and j share a pin on side 1, and connections i and k share another pin on side 2. All three connections are assigned to a same subchannel. Thus the

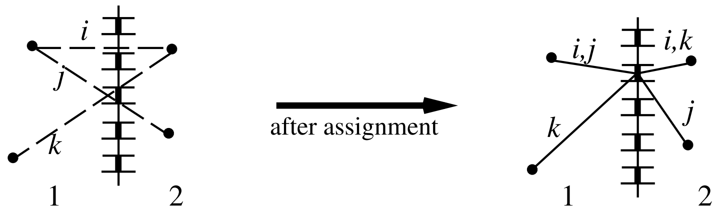

Figure 10: Delay bound redistribution: general case.

new connections $i1$ and $j1$ are the same, as well as the new connections $i2$ and $k2$. The delay bound for a new connections created on the side where the connections share a pin is chosen to be the minimum of the delay bounds should the connections have been treated independently. In the example of Figure 10, the delay bounds for new connections $i1$ ($j1$) and $i2$ ($k2$) are

$$U_{i1} = U_{j1} = min\{\frac{l_{i1}}{l_{i1}+l_{i2}}U_i, \frac{l_{j1}}{l_{j1}+l_{j2}}U_j\}$$

and

$$U_{i2} = U_{k2} = min\{\frac{l_{i2}}{l_{i1}+l_{i2}}U_i, \frac{l_{k2}}{l_{k1}+l_{k2}}U_k\},$$

where U_{i1} is the delay bound for the new connection $i1$, similarly for U_{i2}, U_{j1} and U_{k2}. For a new connection on the side which does not share pins with other connections, the delay bound is the difference between the delay bound for the original connection and the delay bound allocated to the new connection on the other side of cut line. In the example of Figure 10, the delay bounds for the new connections $j2$ and $k1$ are $U_{j2} = U_j - U_{j1}$, and $U_{k1} = U_k - U_{k2}$, respectively.

Notice that assigning connections sharing pins to the same subchannel is equivalent to constructing a steiner point, and thus a steiner tree.

4.6 Remarks

Detailed routing can be easily incorporated with global routing at every hierarchical level. After global routing, the connections assigned to every subchannel are known. For each subchannel, construct a bipartite graph $G = (V_1 \cup V_2, E)$. a connection or a set of connections belonging to same net which are assigned to the subchannel is represented by a node $v_1 \in V_1$. A routing segment in the subchannel is represented by a node $v_2 \in V_2$. There is an edge between a node $v_1 \in V_1$ and a node $v_2 \in V_2$ if and only if all the pins of the connection(s) represented by v_1 can reach the routing segment represented by v_2. Detailed routing is then solved optimally in polynomial time using a bipartite matching algorithm [3]. If detailed routing cannot route all the connections assigned to a subchannel, the capacity of the subchannel is reduced appropriately and the global routing at this hierarchical level is routed again. The iterative procedure continues until either all connections are routed or a predefined routing completion rate is achieved.

One of the advantages of performing detailed routing and global routing simultaneously is that any overestimation in global routing can be corrected immediately and accordingly.

5 Experimental Results

The proposed timing-driven global routing algorithm has been implemented in C on a SUN SPARC 1 workstation. Table 1 is a set of circuits used for testing the algorithm. The circuits are from [4]. The first column are the names of the circuits. The second column are the numbers of connections, i.e., the numbers of source-sink pairs, in each circuit. The third column are the sizes of the circuits in terms of the numbers of logic modules in the arrays.

The routing resources are chosen as follows. There are three subchannels in each horizontal and vertical channel. The first subchannel contains 8 tracks of single lines, i.e., segments of unit length. The second subchannel contains 4 tracks long lines, that is, the lengths of segments are of the entire channel. The third subchannel contains 8 tracks of medium lines, that is, the lengths of segments are between one unit and the entire channel length exclusively. The length of medium lines is chosen as 4 units. Thus there are 20 tracks per channel. These parameters of routing segmentation are close to that in [1]. All three types of subchannels can be connected with each other via switch modules.

A pin in every net of a circuit is chosen randomly as the source pin, and the rest of the pins are chosen as sink pins of the net. Let $l(s, t_i)$ be the Manhattan distance between source s and a sink t_i. The delay bound $U(t_i)$ is chosen randomly and uniformly within the interval $[0.25l(s, t_i), 0.75l(s, t_i)]$. If the delay bound $U(t_i)$ is less than d_{min} [1], then set $U(t_i) = d_{min}$. Each circuit is routed by the algorithm and the percentage of source-sink pairs which meet the delay bounds is computed. The results are shown in the second column of Table 2. For comparison, we also route the circuits by the same routing algorithm, but with cost function for the linear assignment to minimize the wire length and the delay bound distributions/redistributions being turned off. The results are shown in the third column of Table 2. For all the circuits, the timing-driven routing algorithm achieves better results for satisfying the net delay bounds. The improvements are shown in the last column of Table 2. Up to 19% of improvement is obtained. The results show that it is important to consider

[1] Recall in Section 4.2 that d_{min} for two pins is the minimum number of switches required to connect the two pins.

Circuit	# Connections	Circuit Sizes
BUSC	392	12 x 11
X1	436	13 x 12
TooLarge	519	14 x 13
VDA	722	16 x 15
DMA	771	17 x 15
ALU4	851	18 x 16
K2	1256	21 x 19

Table 1: Experimental circuits.

Circuits	Timing-Driven (%)	Traditional (%)	Improvement (%)
BUSC	80.6	61.7	18.9
X1	81.2	70.9	10.3
TooLarge	78.4	68.0	10.4
VDA	83.1	71.6	11.5
DMA	78.9	64.6	14.3
ALU4	86.6	74.4	12.2
K2	87.7	80.6	7.1

Table 2: Experimental results.

the timing constraints during the routing.

It should be noted that the rates of satisfying delay bounds in the experiments depend on the generations of the delay bounds. If the delay bounds generated are too stringent, it is very likely that not all the net delay bounds can be satisfied.

6 Conclusions

We present in this paper a timing-driven global routing algorithm for symmetrical array based FPGAs. The algorithm performs routing with the objective to satisfy timing constraints by utilizing various segments efficiently. The preliminary experimental results show that the algorithm is promising.

Acknowledgements

We would like to thank Steve Brown and Baharam Fallah of University of Toronto for providing us with the benchmark circuits, and Nick Haruyama of AT&T Bell Laboratories for the helpful discussions.

References

[1] AT&T Microelectronics, "AT&T Optimized Reconfigurable Cell Array (ORCA) Series Field-Programmable Gate Arrays Hardware", Product Brief, July 1992.

[2] S. Brown, J.Ross, and Z.G. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays", *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems*, Vol. CAD-11, pp. 620-627, May 1992.

[3] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[4] B. Fallah and J. Rose, "Timing-Driven Routing Segment Assignment in FPGAs", Canadian Conference on VLSI, 1992.

[5] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing", *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 536-542.

[6] A.E. Gamal, et al., "An Architecture for Electrically Configurable Gate Arrays", *IEEE Journal of Solid-State Circuits*, Vol.24, No.2, pp. 394-398, 1989.

[7] P.S. Hauge, R. Nair, and E.J. Yoffa, "Circuit Placement for Predictable Performace", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1987, pp. 88-91.

[8] R.B. Hitchcock, Sr., G.L. Smith, and D.D. Cheng, "Timing Analysis of Computer Hardware", *IBM J. Research and Development*, vol. 26, No. 1, Jan. 1982, pp. 100-105.

[9] H.C. Hsieh, *et al*, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays", *Proc. 1990 Custom Integrated Circuits Conference*, 1990, pp.31.2.1-31.2.7.

[10] M. Khellah, S. Brown, and Z. Vranesic, "Modelling Routing Delays in SRAM-based FPGAs", *Canadian Conference on VLSI*, 1993.

[11] U. Lauther, "Top-down Hierarchical Global Routing for Channelless Gate Arrays Based on Linear Assignment", *Proc. of IFIP VLSI 87*, 1987, pp.109-120.

[12] G. Lemieux and S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs", *ACM Physical Design Workshop*, California, April 1993.

[13] M. Marek-Sadowska, "Route Planner for Custom Chip Design", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1986, pp. 246-249.

[14] M. Marek-Sadowska, "Issues in Timing Driven Layout", in *Algorithmic Aspects of VLSI Layout, Lecture Notes Series on Computing*, D.T Lee and M. Sarrafzadeh ed., World Scientific Publishers, 1993.

[15] M.Palczewiski, "Plane Parallel A* Maze Router and Its Application to FPGAs", *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 691-697.

[16] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.

[17] Y.C. Sun, T.C. Wang, C.K. Wong, and C.L. Liu, "Routing for Symmetric FPGAs and FPICs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 486-490.

[18] K. Zhu, D.F. Wong, and Y.W. Chang, "Switch Module Design with Application to Two-Dimensional Segmentation Design", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 480-485.