A Router for Symmetrical FPGAs based on Exact Routing Density Evaluation

Nak-Woong Eum
Electronics & Telecom. Research Institute
Taejon KOREA

Taewhan Kim Chong-Min Kyung
Dept. of EECS, KAIST
Taejon KOREA

Abstract

This paper presents a new performance and routability driven routing algorithm for symmetrical array based field-programmable gate arrays (FPGAs). A key contribution of our work is to overcome one essential limitation of the previous routing algorithms: *inaccurate estimations of routing density* which were too general for symmetrical FPGAs. To this end, we derive an *exact routing density calculation* that is based on a precise analysis of the structure (switch block) of symmetrical FPGAs, and utilize it consistently in global and detailed routings. With an introduction of the proposed accurate routing metrics, we design a new routing algorithm called *a cost-effective net-decomposition based routing* which is fast, and yet produces remarkable routing results in terms of both routability and path/net delays. We performed an extensive experiment to show the effectiveness of our algorithm based on the proposed cost metrics.

1 Introduction

Due to the low manufacturing cost and time, the FPGA (field-programmable gate array) has become the most popular ASIC for fast system prototyping. The symmetrical array architecture is one of the most widely used classes of FPGA architectures [1]. The architecture consists of two-dimensional (2-D) arrays of configurable logic blocks (CLBs), rows and columns of routing channels, and programmable switch blocks. Each CLB has a look-up table (LUT), which allows to implement any Boolean function constrained by the number of inputs. Routing channels are predefined wire segments of variable length. Switching blocks are programmable to make connections between the wire segments and CLB pins.

Many routing approaches for symmetrical FPGA have been proposed over the past several years. [2] decomposed each net into a set of two terminal nets, and routed them in minimum distance paths. The objective is to distribute the wires among the channels such that the maximum channel density is minimized. [3] proposed Steiner spanning-tree based routing approach (called a negative reinforcement method) to overcome the sequential routing of two-terminal nets. SEGA [4] tried to route critical nets first so that long connections do not suffer long propagation delay through multiple programmable switches. GBP [5] formulated the routing problem into the two-dimensional interval packing problem. It uses two greedy algorithms iteratively to pack the nets into the tracks. IKMB [6] modeled interconnection resources as a graph, and constructed routing trees for the nets by the repeated applications of a Steiner-tree generation algorithm. [7, 8] proposed a new routing approach for 2-D FPGA/FPIC in which the main focus is to minimize the channel density to improve the routability of the designs. TRACER-fpga [10] and TRACER-fpga_PR [11] initially routed nets sequentially according to their criticalities and routabilities. The nets/paths violating the routing resources are then resolved iteratively by employing simulated evolution based optimization technique. The difference between TRACER-fpga and TRACER-fpga_PR is in their cost formulations to be used to serialize the nets to route. Until now, TRACER-fpga and TRACER-fpga_PR have reported the best routing results in terms of nets/path delays and routability. However, one of their drawbacks is a very excessive execution time, which is mainly because of the full application of the multiple component growth wave expansion algorithm [9] to find approximations of the Steiner trees, and the increased complexity of representing a grid graph in the one-step (i.e., combined global and detailed) routing strategy.[1]

In this paper, we present a new performance and routability driven routing algorithm for symmetrical array based FPGAs. One key feature of our approach is to overcome one critical limitation of the previous works, namely, *inaccurate routing density estimations*.

The existing routing algorithms use the "routing density" as the measure of the routability in (global) routing. However, the routing density reflects the degree of net-congestions at the connection blocks of symmetrical FPGA only, and does not take into account the net-congestions at the switch blocks properly.[2] Consequently, this leads to routing solutions with a sub-optimal use of routing resources in the switch blocks. Clearly, maximally utilizing the routing resources in the switch blocks as well as in the connection blocks is a key for achieving an optimal routing result in terms of path/net delay and routability. For this reason, we analyze the interconnection structure of symmetrical FPGA and derive highly reliable cost metrics which not only take into account net-congestions at the connection blocks but also at the switch blocks. We then design a new routing algorithm to utilize the cost metrics more effectively. Our goal for designing a new routing algorithm is to produce more efficient routing solutions in terms of net/path delays and routability while does not spend a long execution time. To this end, we develop a new routing technique, called *a cost-effective net-decomposition based routing*. The basic idea of the technique is essentially drawn from the concept of divide-and-conquer which decomposes an n-terminal net into a set of 2-terminal subnets (using minimum-distance spanning tree), and routes them sequentially, one at a time. However, we employ a more predictive way of finding a minimum-distance spanning tree by estimating the routing density of every point-to-point path in a net with a full considerations of the switch resources. We have conducted a series of experiments to demonstrate the strength of the proposed routing algorithm (will be shown in Sec. 4.).

2 Routing Density for Symmetrical FPGA

2.1 Routing Architecture

Figure 1(a) depicts the architecture of a typical symmetrical FPGA. It consists of three types of components: configurable logic blocks

[1] Note that when the number of tracks is W, one-step routing strategy using a wavefront expansion method requires W^2 times more grid points than that of two-step (global and detailed) routing strategy.

[2] See Figure 1 for the connection resources in symmetric FPGA.

(CLBs), configurable I/O blocks (IOBs), and interconnection resources. As shown in Figure 1(b), the interconnection resources consist of wiring segments and switch points which are organized into vertical and horizontal routing channels. A wire segment (i.e., track) is connected between CLB pins and switch points in the connection and switch blocks. Figures 2(a) and (b) show the programmable switches in the switch points of the connection and switch blocks, respectively.

According to [12], it is shown empirically that a reasonable flexibility of routing resources is achieved when the connection blocks are fully flexible (i.e., $F_c = W$ where W is the number of tracks per channel) and the flexibility of switch blocks is three (i.e., $F_s = 3$). Consequently, the routing architecture we are considering is the one shown in Figure 1(b) and Figure 2 in which $F_c = W$, $F_s = 3$, and all wire-segments are single-length segments.

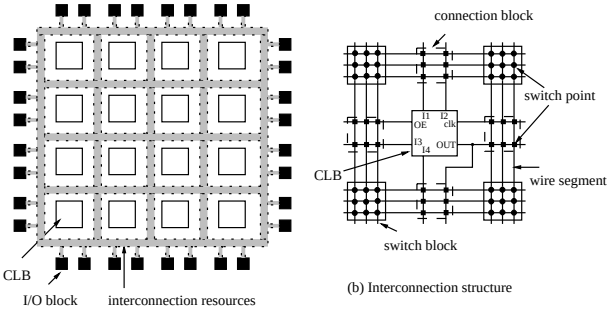


Figure 1: The architecture of symmetrical FPGA.

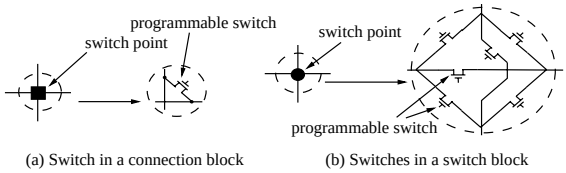


Figure 2: Programmable switches in switch points.

2.2 Classification of Routing Patterns

We classify all possible routing paths around a switch point in a switch block into four classes of routing patterns shown in Figure 3. Routing pattern R_1 and R_2 interconnect two wire segments in horizontal and vertical directions using one (programmable) switch, respectively. Note that R_1 and R_2 can be implemented with a single switch point. Each of routing patterns R_3, R_4, R_5 and R_6 in Class 2 use only one diagonal switch to interconnect a vertical (or horizontal) wire segment to a horizontal (or vertical) segment. Thus, R_3 and R_4 can be implemented with the same switch point, and also R_5 and R_6 can be implemented with the same switch point. However, any of routing patterns $R_7, \cdots, R_{11}$ in Class 3, each of which connects three or four wire segments, can share a switch point with none of the routing patterns in Classes 1, 2 and 3.

Finally, routing patterns R_{12}, R_{13}, R_{14} and R_{15} in Class 4 do not require (programmable) switch at all, but occupy a single wire segment. The wire segment is used by a net connected to a pin of logic blocks next to the switch block to route via another switch blocks. Consequently, the routing patterns in Class 4 can share a switch point with some of the routing patterns in Classes 1, 2, 3 and 4. For example, R_{12} can share a switch point with R_2, R_4, R_6, R_{10}, R_{13}, R_{14} and R_{15}. Table 1 summarizes such switch point sharing with respect to the routing patterns in Class 4.

Now, suppose we have a global (intermediate) FPGA routing solution for a design. That is, a set of routing patterns in every switch block is known. Then, what we want to know (estimate) is *the minimum number of tracks required for each switch block* to route the routing patterns, which has not been well studied in the previous works. The next subsection provide the details on this.

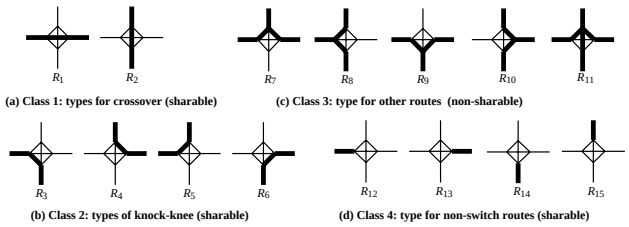


Figure 3: Classes of routing patterns on a switch point in Figure 2(b).

patterns	switch block sharing patterns			
	Class 1	Class 2	Class 3	Class 4
R_{12}	R_2	R_4, R_6	R_{10}	R_{13}, R_{14}, R_{15}
R_{13}	R_2	R_3, R_5	R_8	R_{12}, R_{14}, R_{15}
R_{14}	R_1	R_4, R_5	R_7	R_{12}, R_{13}, R_{15}
R_{15}	R_1	R_3, R_6	R_9	R_{12}, R_{13}, R_{14}

Table 1: Switch point sharing patterns with R_{12}, R_{13}, R_{14} and R_{15}.

2.3 Exact Calculation of Routing Density

In the conventional routers, the term "routing density" refers to the density at the connection blocks only. Since a track in a connection block cannot be shared among the nets, the routing density is simply equal to the number of nets that are in the block.

However, as indicated in Sec. 2.2, the routing density at the switch block is rather complex. We describe the exact calculation of routing density in two steps: (1) formulating the density calculation using the routing patterns in Classes 1, 2 and 3, and (2) completing the formulation incorporating the routing patterns in Class 4. Let $r_i(k)$ (i=1, $\cdots$,15) be the number of nets (i.e., routing path) of routing pattern R_i in switch block k.

Loose routing density: This is the case when r_{12} = r_{13} = r_{14} = r_{15} = 0. We formulate the routing density at switch block k in terms of the numbers of $r_1, \cdots$, and r_{11}. Let $d_s(k)$ be the routing density at switch block k. Then,

$$d_s(k) = \sum_{i=1}^{11} r_i(k) - min\_track(k). \quad (1)$$

where $min\_track(k) = min(r_1(k), r_2(k)) + min(r_3(k), r_4(k)) + min(r_5(k), r_6(k))$.

That is, $d_s(k)$ is the the number of tracks to be used by the nets in switch block k when a maximal sharing of tracks among the nets is assumed.

From the fact that there is no possibility of track sharing among the nets in the connection blocks, when $d_c(j)$ denote the routing density at connection block j, $d_c(j)$ is expressed as

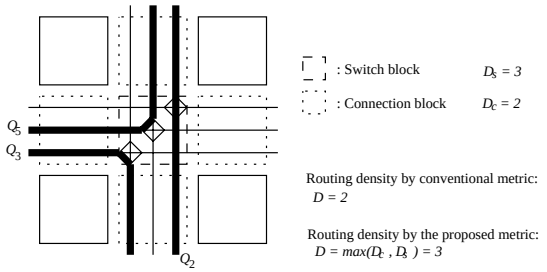
$$d_c(j) = \#\ of\ nets\ in\ connection\ block\ j. \quad (2)$$

Let D_c be the maximum of $d_c(\cdot)$ values over all connection blocks, D_s be the maximum of $d_s(\cdot)$ values over all switch blocks, and let

$$D = max(D_c, D_s) \quad (3)$$

D is the maximum value among all the routing densities at the connection and switch blocks. We refer to D as *the maximum density*, which indicates the minimum number of tracks required for all nets to be routed successfully. Note that for an incremental change to the global routing, we can update the D value in a constant time.

For example, Figure 4(a) shows three nets on the routing region. It is shown that at least three tracks are needed for the nets. Figure 4(b) shows the calculation of the routing densities by the conventional and proposed metrics. Since the traditional metric uses the densities at the connection blocks only, the maximum density (i.e., D_c) becomes 2. On the other hand, because the proposed metric (i.e., $D = max(D_c, D_s)$) takes into account the densities at the switch blocks as well, the maximum density becomes 3. This example indicates that a reliable estimation of routing densities at the switch blocks as well as at the connection blocks could be essential to achieve routing results with both high routability and fast performance of the designs.



(a) (b)

Figure 4: An example of the routing density calculation by using the conventional and proposed metrics.

Exact routing density: We now extend the formulation of $d_s(k)$ to take into account the case when r_{12}, r_{13}, r_{14} and/or r_{15} are nonzero. We define $\bar{r}_i$ (i = 1,$\cdots$,6) as follows: $\bar{r}_1$=r_1-$min(r_1, r_2)$, $\bar{r}_2$=r_2-$min(r_1, r_2)$, $\bar{r}_3$=r_3-$min(r_3, r_4)$, $\bar{r}_4$=r_4-$min(r_3, r_4)$, $\bar{r}_5$=r_5-$min(r_5, r_6)$, and $\bar{r}_6$=r_6-$min(r_5, r_6)$. In other words, $\bar{r}_1$ is the number of nets (i.e., routing paths) of routing pattern R_1 remained after the maximal sharing of the switch block with nets of pattern R_2. ($\bar{r}_2$, $\cdots$, $\bar{r}_6$ also have similar meanings.) Consequently, some of the nets of routing patterns in Class 4 can be used if $\bar{r}_i$ is nonzero. Let $\bar{r}_{12}$ be the number of nets of routing pattern R_{12} remained after the maximal sharing of tracks with nets of other routing patterns, which can be expressed as $\bar{r}_{12} = f(r_{12}, (\bar{r}_2 + \bar{r}_4 + \bar{r}_6 + r_{10})$ where $f(a, b) = a - b$ if $a > b$, and 0 otherwise, since each R_{12} net can share a track with the one corresponding to $\bar{r}_2$, $\bar{r}_4$, $\bar{r}_6$, or r_{10}. (See Figure 3 and Table 1.) Similarly, $\bar{r}_{13} = f(r_{13}, (\bar{r}_2 + \bar{r}_3 + \bar{r}_5 + r_8)$, $\bar{r}_{14} = f(r_{14}, (\bar{r}_1 + \bar{r}_4 + \bar{r}_5 + r_7)$, and $\bar{r}_{15} = f(r_{15}, (\bar{r}_1 + \bar{r}_3 + \bar{r}_6 + r_9)$. Then, the density metric of $d_s(j)$ which redefines the one in *Eq.1* is

$$d_s(k) = \sum_{i=1}^{11} r_i(k) - min\_track(k) + max(\bar{r}_{12}, \bar{r}_{13}, \bar{r}_{14}, \bar{r}_{15}). \quad (4)$$

3 The Routing Algorithms

Conventional physical design approaches divide the routing problem into two subproblems: *global routing* and *detailed routing*. The main reason behind such decomposition is to reduce the problem complexity. However, it is generally true that even solving each subproblem alone is still a timing-consuming process. The essential idea of our routing algorithm is to exploit the fact that using an accurate estimation of routing density helps pruning a significant portion of the unnecessary search space. This can be done by fully taking advantage of our cost metrics proposed in Sec. 2. The key is how to make use of the cost metrics to fit into the context of global and detailed routings without sacrificing the quality of results.

3.1 Global Routing

In global routing, we need to consider two issues: (1) measuring accurately how much relatively hard (or competing) a net to route, and (2) constructing an efficient routing tree for a net. The first corresponds to the *routing tree cost-estimation problem*, based on which the nets are ordered, and a sequence of routing tree construction is performed according to that order, while the second corresponds to the *routing tree construction problem* for a net.

Routing tree cost-estimation problem: The problem is, given an initial (global) routing result for all nets, to measure the degree of difficulty in reconstructing a routing tree for a particular net. We determine the measure based on two factors, *the value of D in Eq.3* and *the amount of routing resources used*. Clearly, it is desirable for a route to have a small value of D to improve routability. Moreover, to reduce the net/path delay the router must use as small number of routing resources as possible, and further, use the routing resources uniformly over the connection/switch blocks. Note that most of the conventional global routing approaches try to minimize D_c only, and/or never have taken into account the minimization of D and the minimization (and evenly distribution) of routing resources together.

First, we measure the degree of routing difficulty in a single connection/switch block. Let $w_c(j)$ and $w_s(k)$ be the measures in connection block i and switch block j, respectively, and defined as

$$w_c(j) = 2^{\lfloor p \times L \times d_c(j)/D \rfloor}, \quad w_s(k) = 2^{\lfloor p \times L \times d_s(k)/D \rfloor} \quad (5)$$

where p ($0 \leq p \leq 1$) is a user defined control parameter. (L is defined later.)

$w_c(j)$ and $w_s(k)$ are nonlinear cost functions. The discontinuous property of the function disables the router to be too much sensitive to a little changes of routing densities. The control parameter p used at a connection/switch block controls the relative importance between the degree of net congestion and the utilization of routing resources in the corresponding block. Thus, by calculating $w_c(\cdot)$ and $w_s(\cdot)$ with a fixed value of p for all the connection and switch blocks, we can tell, among the blocks, which blocks are relatively expensive to use in terms of the (weighted) net congestion and routing resources. (The large value of p implies a relatively more emphasis on the evenly distribution of congestion (i.e., the minimization of D) than on the minimization of routing resources to be used, and vice versa.)

The L is defined to be $L = log_2(L_h + L_v)$ where L_h and L_v is the numbers of CLBs on the horizontal and vertical lines of the corresponding FPGA device, respectively: Since $0 \leq d_c(j)/D \leq 1$ and $0 \leq d_s(k)/D \leq 1$, as $d_c(j)/D$ (or $d_s(k)/D$) approaches to 1, the routing passing through the corresponding connection block j (or switch block k) is very hard, most likely resulting in detouring the block, in which the worst detouring length is bounded by $L_h + L_v$, which is in fact the value of $w_c(j)$ (or $w_s(k)$) in *Eq.5* when $d_c(j)/D = 1$ (or $d_s(k)/D = 1$).

Let $R_c(n_i)$ and $R_s(n_i)$ be the sets of connection and switch blocks in the rectangular region bounded by the terminals of net n_i, respectively. Then, we use the following quantity as a measure of the routing difficulty for net n_i:

$$Route\_Crit(n_i) = \sum_{j \in R_c(n_i)} w_c(j) + \sum_{k \in R_s(n_i)} w_s(k). \quad (6)$$

The example in Figure 5 helps understanding how the quantity of $Route\_Crit(\cdot)$ with different values of p reflects the trade-off

between the resource usage and the routing congestion. Figure 5(a) shows a density configuration with the maximum density D=6 in which a net (v_i,v_j) is to route. Figures 5(b), (c), and (d) show the prediction of routing paths according to the relative emphasis between the resource usage and routing congestion controlled by the value of parameter p. The number shown in each of the switch and control blocks represents the value of the corresponding $w(\cdot)$ in *Eq.5*. Note that the predicted routing path in Figure 5(a) becomes P_a, which longer in length (i.e., passing two more blocks) than an alternative path, P_b. This is because routing the net (v_i,v_j) through path P_a is very likely to be less congested than routing it through P_b.

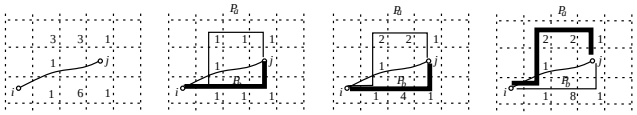


Figure 5: An example of illustrating the trade-off between resource usage and congestion controlled by parameter p in *Eq.5*. (a) A density configuration with $D = 6$, (b) when $p = 1/L$: $\sum w(\cdot)$=5 for P_a, $\sum w(\cdot)$=3 for P_b, (c) when $p = 2/L$: $\sum w(\cdot)$=7 for P_a, $\sum w(\cdot)$=6 for P_b, (d) when $p = 3/L$: $\sum w(\cdot)$=7 for P_a, $\sum w(\cdot)$=10 for P_b.

Routing tree construction problem: Once the net ordering is completed according to the quantities of $Route\_Crit(\cdot)$, we now need to create a routing tree for each net. The commonly used algorithms for routing tree construction are the ones based on the classical maze routing [13]. Our proposed algorithm is also based on the maze routing, but there is a fundamental difference in transforming the original problem into a set of (simple) maze routing problems.

Let us first briefly review the concept of maze routing. In maze routing, the topology of routing resources in symmetrical FPGA is modeled as a grid region, in which each of the connection and switch blocks corresponds to a distinct grid point in the grid region. The terminals of a net have their physical locations of grid points, which we call the initial grid points for the net. The initial grid points are then expanded by adding them one more adjacent grid point.

One inherent limitation of the maze routing is its high time complexity. (i.e., the time complexity increases rapidly with respect to both of the number of terminals in the net and the distance between the grid positions of the terminals.) Consequently, the key issue in designing an efficient routing algorithm is *how to determine "the best" grid point to be included during each expansion*. Our approach (called *a net-decomposition based routing algorithm*), in fact, uses the routing density metrics proposed in Sec. 2 effectively to find out the best grid points to expand, and consists of two steps.

- Step g-1 (*Construct a minimum-cost spanning tree*): We construct a complete graph $G = (V, E)$ for a k-terminal net where V is the set of the initial grid points and E is the set of edges between the points. Each edge in E is assigned with a cost (defined later). Then, we find a minimum-cost spanning tree for G using Prim Algorithm [20]. We formulate the cost of the edge interconnecting two initial points in a way to reflect how much making a route between the two points with less congestion is easy.

  We assign cost $e\_cost_g(n_{ij})$ to the edge of points i and j where n_{ij} is a net with two terminals i and j(which is a subnet of the k-terminal net), and the cost is defined as

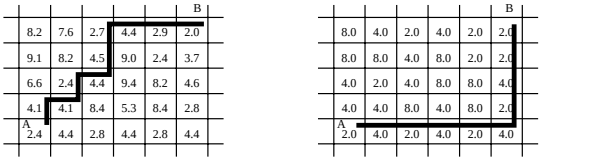
$$e\_cost_g(n_{ij}) = (BLK(n_{ij})/SIZE(n_{ij})) \cdot l(i,j) \qquad (7)$$

  where $BLK(n_{ij}) = \sum_{m \in R_c(n_{ij})} d_c(m) + \sum_{m \in R_s(n_{ij})} d_s(m)$, $SIZE(n_{ij}) = |R_c(n_{ij}) + R_s(n_{ij})|$, $l(i,j)$ is the Manhattan distance (in terms of the number of connection and switch blocks) between i and j, and $R_c(n_{ij})$ and $R_s(n_{ij})$ are the sets of connection blocks and switch blocks that are in the rectangular region bounded by the points i and j, respectively.

  The first term in the formulation of $e\_cost_g(n_{ij})$ represents the average routing density in the region currently concerned for the net to route, and the second term represents the scope of the region. Clearly, as the value of $e\_cost_g(n_{ij})$ increases, making a route between i and j is more likely to be hard, and conversely.

- Step g-2 (*Apply maze routing to the spanning tree*): We arrange the edges in the spanning tree obtained in Step g-1 in decreasing order according to the values of $e\_cost_g(\cdot)$, and apply maze routing to the sorted edges one at a time, with the most critical (i.e., the most hard to route) one first and the least critical one last.

Note that our maze route is driven by the routing measures $w_c(\cdot)$ and $w_s(\cdot)$ in *Eq.5*. Empirically, we found that the discretization of the routing measures have played an important role in reducing the number of bends on the routing path as illustrated in Figure 6. In fact, the use of control parameter p and the discontinuous property of the cost metrics enables our router to explore many alternative (totally different) routing paths, as will be shown in Figure 13.



(a) A maze routing driven by density metric w(.) without discretization of density values: #bends=5.

(b) A maze routing driven by density metric w (.) with discretization of density values: #bends=1.

Figure 6: The maze routing examples using our cost metrics $w_c(\cdot)$ and $w_s(\cdot)$ in *Eq.5* without and with the discretization of cost values.

Our router starts from an initial routing solution or an estimation of the routing distribution for nets. We use the probabilistical concept, called *net-existence*, to estimate the routing distribution.[3] Routing based on net-existence is widely used method for estimating routing distribution [14]; From the information of given terminal positions of each net to be routed, we draw a bounding box that covers the terminals, and then determine, for a particular block, a probability that the net is routed passing through the block. By repeating this process for every net we obtain a probabilistic measure of the degree of routing difficulty for each block. We then apply a *net-by-net* maze routing based on the measure to generate an initial routing solution.

We calculate our basic metrics of routing densities, $d_c(\cdot)$ and $d_s(\cdot)$, from which we compute the routing criticality measure $Route\_Crit(\cdot)$ (*Eq.7*) for each net. Then, we reroute nets iteratively. We generate the $k-1$ most routability/performance efficient edges from the graph corresponding to the net (i.e., Step g-1), and apply maze routing to the edge one by one (i.e., Step g-2). The iteration process repeats until there is no reduction on the $Cost$, in which $Cost$ used for determining the acceptance of the new route is defined as

$$Cost = \alpha_1 \cdot D + \alpha_2 \cdot (tot.\ \#bends) + \alpha_3 \cdot (tot.\ wire\ length) \qquad (8)$$

[3]Also, it is possible to generate an initial routing solution by simply applying any of greedy/random routing methods.

α_1, α_2 and α_3 are weighting factors, $\alpha_1 > \alpha_2 > \alpha_3$, and D is the maximal density defined in Sec. 2.

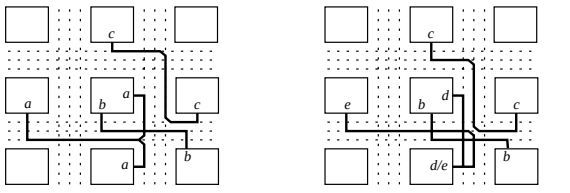
Once routing trees for nets are determined in the global routing, a detailed routing which assigns routing resources to the nets in the corresponding connection/switch blocks will be executed. We perform it in a sequence of three steps, as will be presented in Sec. 3.2. Note that, for a given design, we apply our router iteratively by changing the value of p in *Eq.5* to explore the trade-off between routability and speed performance. (An empirical results on the effectiveness of p in our algorithm is shown in Figure 12.)

3.2 Detailed Routing

In this phase, the problem is to assign the nets in a connection/switch block to the routing resources (i.e, tracks and programmable switches) in a way to minimize the number of routing resources used. The resource assignment can be modeled as a graph coloring problem[5], in which the nodes represent nets and there is an edge between two nodes if the corresponding nets cannot share a track. Finding a minimum colors for the graph will determine the minimum number of tracks to be used by the nets.

We may use any of the conventional detailed routing algorithms for symmetrical FPGA. However, from the analysis of the route types at a switch point shown in Figure 3, it is found that we can utilize tracks more effectively if we can exploit the several possible ways of implementing (shown in Figure 3(c)) nets of types $R_7 \cdots, R_{11}$ in Class 3.

The example in Figure 7 shows the effect of selecting the implementation of a net of type in Class 3 on the minimization of tracks to be used. Figure 7(a) shows the case that net a, which has type R_8, is routed using *a single switch point* with the first implementation in Figure 3. On the other hand, Figure 7(b) shows the case that net a is decomposed into two subnets d and e. This allows the possibility of using *multiple switch points* for routing them, in which their route implementations are determined in a way to minimize the number of tracks used. Consequently, d is implemented with type R_2 and e is with type R_3, which results in saving one track.



(a) Routing results considering one implementation for net a with type R_8 (#tracks_used = 3)

(b) Routing results by decomposing net a into two subnets d and e (#tracks_used = 2)

Figure 7: Effect of decomposing net of type R_8 on the track utilization.

Our proposed detailed routing algorithm will maximally exploit such flexibility of implementing the nets of types $R_7 \cdots, R_{11}$ in Class 3, and is composed of three steps.

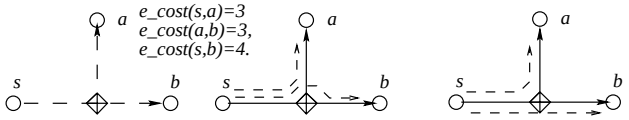
- Step d-1 *Decomposition step*: This step is essentially a variant of the *net-decomposition based routing* technique presented in Sec. 3.1. It decomposes every k-terminal ($k > 2$) net into a set of two-terminal nets. This allows the net to be routed with any of the several possible implementations of $R_1, \cdots, R_{15}$, which otherwise, an implementation of type in Class 3 is chosen. Let n_{ij} be the subnet that interconnects terminals i and j of the k-terminal net. Then, the decomposition is based on the following cost.

$$e\_cost_d(n_{ij}) = \beta_1 \cdot \sum_{m \in R_s(n_{ij})} \Delta d_s(m) \; + \; \beta_2 \cdot l(n_{ij}) \qquad (9)$$

where $\Delta d_s(m)$ is the increment of routing density $d_s(m)$ at the switch block m, and $l(n_{ij})$ is the number of wire segments on the routing path (given by the global routing in Sec. 3.1) between i and j. β_1 and β_2 are weighting factors, and $\beta_1 >> \beta_2$.

The $e\_cost_d(n_{ij})$ is basically similar to the cost formulation of $e\_cost_g(n_{ij})$ in Sec. 3.1. The difference is that $e\_cost_d(n_{ij})$ considers the routing density at the switch blocks only[4] while $e\_cost_g(n_{ij})$ considers the routing density at both the switch and connection blocks. In the same way to find a minimum-cost spanning tree in Step g-1 (using $e\_cost_g(n_{ij})$ cost), we create a complete graph with nodes representing terminals of the net and costs $e\_cost_d(\cdot)$ assigned to the corresponding edges, and find a minimum-cost spanning tree for the graph. Then, the subnets corresponding to the edges in the tree become the set of two-terminal nets.

However, if a net is timing critical, the net should be decomposed with an emphasis on the minimization of interconnection delay rather than the routability. For example, let us consider a net with three terminals a, b (sink), s (source), and edge cost $e\_cost_d(.)$ between the pairs of terminals in Figure 8(a). If the net is not timing critical, it is decomposed into two subnets (s, a) and (a, b), minimizing the total cost of *Eq.9*, thereby creating the route shown in Figure 8(b) where the dotted arrows indicate the delay paths of the net. On the other hand, if the net is timing critical, it is decomposed in a way to minimize the net delay for source to every sink, resulting in the route shown in Figure 8(c). Obviously, the routing delay from s to b in Figure 8(c) is shorter than that in Figure 8(b) because the former uses one switch while the latter uses two switches. In general, given a timing critical k-terminal net, finding the shortest k-1 subnets whose routes coincide with the signal flow originating from source to sink can be done by breadth first search in $O(k)$ time.



(a) Signal flow of a net.

(b) total_cost = 6, but path s->b needs 2 switches.

(c) total_cost = 7, but path s->b needs 1 switch.

Figure 8: Decomposition of a timing critical net.

- Step d-2 *Resource assignment step*: We assign the subnets obtained in the *decomposition step* to routing resources. The objective is to minimize the number of tracks used at each of connection/switch block. We use the efficient graph coloring algorithm proposed in [15].

- Step d-3 *Merging step*: This step tries to merge the connections[5] in different connection/switch blocks to maximize resource sharing. The sharing of tracks will lead to a reduction in the number of switches used, which would then directly contribute to the reduction in the routing delay of the net. The merging process is performed repeatedly for the nets, in which the most timing-critical net is processed first and the least timing-critical net last. The possible merging of subnets is then performed from the subnet containing the source terminal toward the subnets containing the sink terminals.

[4]Note that the concept of our net-decomposition does affect the optimization of resource sharing at the switch blocks, and is nothing to do with that at the connection block.

[5]We call the subnets assigned to tracks *connections*.

The example in Figure 9 depicts our merging process. Figure 9(a) shows the result of track assignment for the subnets a_1, a_2, a_3 and a_4 of a 5-terminal net where $t0$ is the source and $t1$, $t2$, $t3$ and $t4$ are the sink terminals. (We assume that some of the wire segments have already been assigned to other nets.) Consequently, the timing-critical path is $a_4 \rightarrow a_3 \rightarrow a_2 \rightarrow a_1$. First, merging is processed for the connections of subnets a_4 and a_3, and the result is shown in Figure 9(b). Note that the resulting connection at the switch block becomes a route of type R_9 (denoted by (a_3,a_4) in the figure). Connection (a_3,a_4) is then tried to be merged with the connection of a_2, followed by trying to merge with the connection of a_1.

For each net we model the merging problem using a graph, called *resource sharing graph*, in which a node represents a subnet and an edge exists between two nodes if the corresponding subnets have been connected via dogleg. We can now simplify the graph by merging the nodes in a way to maximize the resource sharing. This in trun leads to a reduction of the routing delay without any degradation in routability. For example, Figure 10 shows the resource sharing graphs for the subnets in Figure 9 in which the resources to be merged are specified on the edges of the graphs.

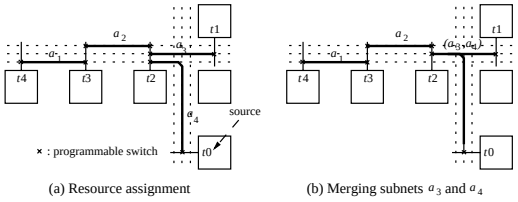


Figure 9: Merging two subnets from the result of an initial resource assignment.

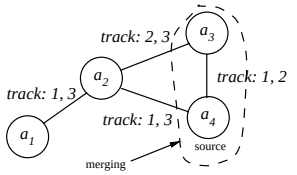


Figure 10: Resource sharing graphs for the subnets in Figure 9.

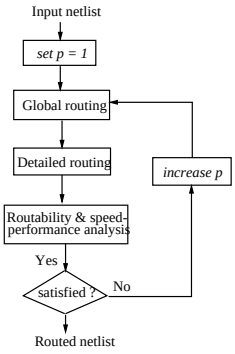


Figure 11: The overall implementation scheme for the proposed routing flow.

4 Experimental Results

We have implemented the proposed routing algorithm in the C programming language on a SUN Sparc 10 workstation. We have tested our algorithm on a set of benchmarks reported in SEGA [4]. We explore the trade-offs between the performance and routing density results using parameter p in *Eq.5* as indicated in the overall implementation scheme for the proposed routing flow in Figure 11.

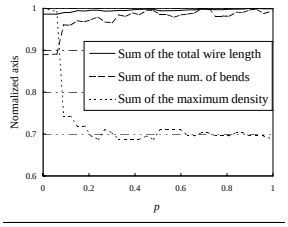


Figure 12: Curve showing the effect of p in *Eq. 5*.

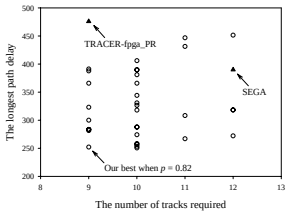


Figure 13: Solution space exploration for too_large.

In all experiments, we set α_1=1000, α_2=10, α_3=1 (*Eq.8* in global routing), and β_1=100, β_2=1 (*Eq.9* in detailed routing). Empirically, we found that p is much more effective in controlling the trade-offs between the performance and routing density rather than the values of α and β. In addition, we set F_c=W and F_s=3 (as in [4, 11]). We performed our experiments in four respects: (1) checking the design space exploration, (2) checking the routing density (3) checking the performance, and (4) checking the degree of the trade-offs between the routing density and performance.

Design space exploration: Figure 12 shows a summary of empirical results on how the trade-offs between the speed-performance and routability are explored by using the control parameter p. The speed-performance is measured by taking into account the delay of the bends (i.e., programmable switches) and the delay of the wire-segments used, and the routability is measured by the minimum number of tracks required like most routing algorithms in the literature. The three curves in Figure 12 corresponds to the results of the ten benchmark examples in the first column of Table 2. The shape of the curves clearly reveals an empirical evidence that p is well fitted into our routing algorithm for an extensive exploration of the trade-off between routability and performance, and suggests that we can find a routing solution that satisfies the performance requirement within a few iterations of the router while minimizing the routing density.

Figure 13 depicts the exploration result for circuit too_large. Within the full range of p, there exists eight different values of p which produce the minimal T, i.e., 9. Even with the minimal T, the longest path delay is widely ranged from 252 to 391. This mainly comes from the capability of our global router to form diverse routing trees under different values of p. Figure 13 also shows the results produced by TRACER-fpga_PR [11] and SEGA [4]. The comparison shows that most of the longest path delays produced by our router are shorter than those by TRACER-fpga_PR and SEGA, indicating an effective exploration of solution space.

Routability (routing density): Table 2 summarizes the maximum required routing density of designs generated by SEGA [4], GBP [5], FPR [16], OGC [17], IKMB [6], TRACER-fpga [10], TRACER-fpga_PR [11], VPR [18, 19] and our algorithm. Note that some of the routers have not reported routing solutions for circuit *z03D4*, which was mainly due to the memory overflow problem. Thus, for a fair comparison we exclude the data of *z03D4* in the parentheses from *total* of the table. The comparison of the results indicates that ours uses much less number of tracks than those by the existing routers except VPR [19]. However, note that VPR considers only the minimization of routing density and never takes into account the minimization of net/path delay. The results clearly indicate that the proposed algorithm based on the exact routing density measures is very effective.

Performance (net/path delay): Table 3 shows the results of the longest path and net delays of designs produced by SEGA [4], TRACER-fpga [10], TRACER-fpga_PR [11] and ours. In summary, our algorithm reduces the longest path and net delays by 30% and 23% (on average), respectively. The last column of the table

circuits	[4]/[5]/[16]/[17]/[6]/[10]/[11]/[18]/[19]	*Ours*
9symml	10/9/9/9/8/7/6/7/6	6
alu2	11/11/10/9/9/9/9/8/8	7
alu4	15/14/13/12/11/11/11/10/9	10
apex7	13/11/9/10/10/8/8/10/8	8
example2	17/13/13/12/11/10/10/10/9	10
k2	17/17/17/16/15/14/14/14/12	13
term1	10/10/8/9/8/8/7/8/7	7
too_large	12/12/11/11/10/10/9/10/8	9
vda	13/13/13/11/12/11/11/12/10	10
z03D4	(14)/-/-/-/-/(11)/-/-/-	(10)
total	118/110/103/99/94/88/85/89/77	80

Table 2: Comparison of the required number of tracks per channel.

shows the comparison of the run time. The results indicate that our algorithm outperforms the best known algorithms, TRACER-fpga [10, 11] and SEGA [4] even with much less execution time. This clearly confirms that the cost metrics used in our algorithm is quite reliable and accurate.

circuits	path delay [4]/[11]/*Ours*	net delay [4]/[10]/*Ours*	CPU (sec) [11]/*Ours*
9symml	320/313/226	57/88/50	56/1.8
alu2	947/714/612	110/354/75	66/4.6
alu4	1392/1037/870	163/458/129	207/18.4
apex7	339/299/137	78/147/59	30/4.6
example2	417/296/213	122/145/64	51/8.7
k2	1278/1483/777	229/745/182	349/44.5
term1	129/136/94	70/113/37	16/1.8
too_large	390/476/252	114/291/90	75/8.6
vda	693/743/662	170/628/122	98/11.4
z03D4	-/-/(2348)	191/642/196	-/(56.9)
total	5905/5497/3843	1304/3611/1004	948/104.4

Table 3: Comparison of the longest path and net delays.

circuits	TRACER-fpga_PR [11] best-route/best-speed #track,delay/#track,delay	*Ours* best-route/best-speed #track,delay/#track,delay
9symml	6, 313 / 10, 239	6, 226 / 7, 210
alu2	9, 714 / 11, 707	7, 612 / 8, 604
alu4	11, 1037 / 15, 1096	10, 870 / 11, 852
apex7	8, 299 / 13, 258	8, 137 / 8, 137
example2	10, 296 / 17, 202	10, 213 / 11, 208
k2	14, 1483 / 17, 1059	13, 777 / 14, 767
term1	7, 136 / 10, 103	7, 94 / 8, 91
too_large	9, 476 / 12, 282	9, 252 / 10, 250
vda	11, 743 / 14, 541	10, 662 / 11, 524
z03D4	-,- / -,-	(10, 2348) / (10, 2348)
total	85, 5497 / 119, 4487	80, 3843 / 88, 3643

Table 4: Trade-offs between performance (path delay) and routability.

Routability/Performance trade-offs: Finally, the results in Table 4 reveals how much the longest path delay can be reduced at the expense of the routing density. Overall, our algorithm reduces the delay up to 5.2% with 10% more tracks while TRACER-fpga_PR reduces the delay up to 18.4% with 40% more tracks. This implies that the results produced by our algorithm is much close to an optimal solution in terms of both the routability and performance of design.

5 Conclusions

We have presented a new performance and routability driven routing algorithm for symmetrical array based FPGAs. The key contributions of our work is the formulations of much reliable and accurate cost metrics which are derived from the careful analysis of the interconnection structure of the switch block. This then leaded us to develop a new efficient routing technique called *a cost-effective net-decomposition based routing* which is well suited to exploit our cost metrics more effectively. Extensive experimental data showed that the proposed routing algorithm is very effective in improving the overall performance of the design as well as the routability. In summary, compared to the results produced by TRACER-fpga_PR and TRACER-fpga for the benchmark circuits, our algorithm reduced the delay of the longest path by 30% (on average) and the delay of the longest net by 23% even with about 9 times less execution time.

Acknowledgement: We would like to thank Prof. S. Brown at the Univ. of Toronto, Canada, for providing us with the source of SEGA and benchmark examples. This work was partially supported by a grant from Ministry of Information and Communications of Korea, and Taewhan Kim was supported by the Korea Science and Engineering Foundation (KOREF) through the Advanced Information Technology Research Center (AITrc).

References

[1] *The Programmable Gate Array Data Book*, Xilinx Inc., San Jose, CA, 1992.

[2] S. Brown, J. Rose, and Z. G. Vranesic, "A detailed router for field-programmable gate arrays," *IEEE TCAD*, May 1992.

[3] F. D. Lewis and W. C.-C Pong, "A negative reinforcement method for PGA routing," *Proc. of DAC*, 1993.

[4] G. G. Lemieux and S. D. Brown, "A detailed routing algorithm for allocating wire segments in field-programmable gate arrays," *ACM Physical Design Workshop*, 1993.

[5] Y.-L. Wu and M. Marek-Sadowska, "An efficient router for 2-D field-programmable gate arrays," *Proc. of EDAC*, 1994.

[6] M. J. Alexander and G. Robins, "New performance-driven FPGA routing algorithms", *Proc. of DAC*, 1995.

[7] Y. Sun and C. L. Liu, "Routing in a new 2-dimensional FPGA/FPIC routing architecture," *Proc. of DAC*, 1994.

[8] Y. Sun, T.-C. Wang, C. K. Wang, and C. L. Liu, "Routing for symmetric FPGA's and FPIC's," *Proc. of ICCAD*, 1993.

[9] T. Ohtsuki, "Maze-running and line-search algorithm," *Layout and Design Verification*, North-Holland, 1985.

[10] C.-D. Chen $et\ al$, "TRACER-fpga: A router for RAM-based FPGA's," *IEEE TCAD*, March 1995.

[11] Y.-S. Lee and A. C.-H. Wu, "A performance and routability-driven router for FPGA's considering path delays," *IEEE TCAD*, Feb. 1997.

[12] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. of Solid-State Circuits*, Vol.26, 1991.

[13] C. Y. Lee, "An algorithm for path connections and its application," *IRE Trans. Electronic Comput.*, Sept. 1961.

[14] T. Sadakane, H. Shirota, K. Takahashi, M. Terai, and K. Okazaki, "A congestion-driven placement improvement algorithm for large scale sea-of-gates arrays", *Prod. of CICC*, 1997.

[15] M. Kubale and B. Jackowski, "A generalized implicit enumeration algorithm for graph coloring," *Communication of the ACM*, April 1985.

[16] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, "Performance-oriented placement and routing for field-programmable gate array", *Proc. of EDAC*, 1995.

[17] Y.-L. Wu and M. Marek-Sadowska, "Orthogonal greedy coupling - A new optimization approach to 2-D FPGA routing," *Proc. of DAC*, 1995.

[18] G. Lemieux, S. Brown, D. Vranesic, "On two-step routing for FPGAs," *Proc. of ISPD*, 1997.

[19] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," *International Workshop on Field-Programmable Logic*, 1997.

[20] Sara Baase, *Computer Algorithms - Introduction to Design and Analysis*, Addison Wesley, 1988.