Short Papers

Matching-Based Algorithm for FPGA Channel Segmentation Design

Yao-Wen Chang, Jai-Ming Lin, and M. D. F. Wong

***Abstract*—Process technology advances have made multimillion gate field programmable gate arrays (FPGAs) a reality. A key issue that needs to be solved in order for the large-scale FPGAs to realize their full potential lies in the design of their segmentation architectures. Channel segmentation designs have been studied to some degree in much of the literature; the previous methods are based on experimental studies, stochastic models, or analytical analysis. In this paper, we address a new direction for studying segmentation architectures. Our method is based on graph-theoretic formulation. We first formulate a problem of finding the optimal segmentation architecture for two input routing instances and present a polynomial-time optimal algorithm to solve the problem. Based on the solution to the problem, we develop an effective and efficient multilevel matching-based algorithm for general channel segmentation designs. Experimental results show that our method significantly outperforms the previous work. For example, our method achieves *average* improvements of 18.2% and 8.9% in routability in comparison with other work.**

***Index Terms*—Detailed routing, interconnect, layout, physical design, routing.**

I. Introduction

Due to their low prototyping cost, user programmability, and short turnaround time, field programmable gate arrays (FPGAs) have become a very popular design style for application-specific integrated circuit (ASIC) applications. With the advances in process technology, multimillion gate FPGAs have become available. A key issue that needs to be solved for the large-scale FPGAs to realize their full potential lies in the design of their routing architectures [11].

Fig. 1 shows the row-based FPGA architecture [1], [4]. The architecture is analogous to the traditional standard-cell model. The logic modules, used to implement logic function, are placed in parallel in predefined locations and channels are settled between two neighboring rows of logic modules. Each logic module is linked with vertical segments for input and output. A vertical segment can be connected to a horizontal segment by programming a *cross switch* (denoted by ⊗) to be ON. The routing tracks are divided into several segments of different lengths. Two neighboring segments can be connected together to establish a longer connection by programming the incident *horizontal switch* (denoted by ◯) to be ON.

Unlike the traditional ASIC, the routing resources in an FPGA are prefabricated in the chip and routing in an FPGA is performed by programming switches to make connections. The switches usually have high resistance and capacitance and, thus, incur significant delays. As shown in the previous works in [2] and [6], the number of segments/switches, instead of wirelength, used by a net is the most critical factor in controlling the routing delay in an FPGA. To achieve better performance, each track should contain fewer horizontal switches (i.e., each segment has longer length and each track contains fewer segments). However, this would reduce routability and waste more wire. On the other hand, if a track contains more horizontal switches (i.e., each segment has shorter length and each track contains more segments), nets can be routed with more flexibility and less waste of wire. However, this would sacrifice performance. This tradeoff between performance and routability presents a segmentation design problem: *How to determine a segmentation distribution to maximize the routability under performance constraints?*

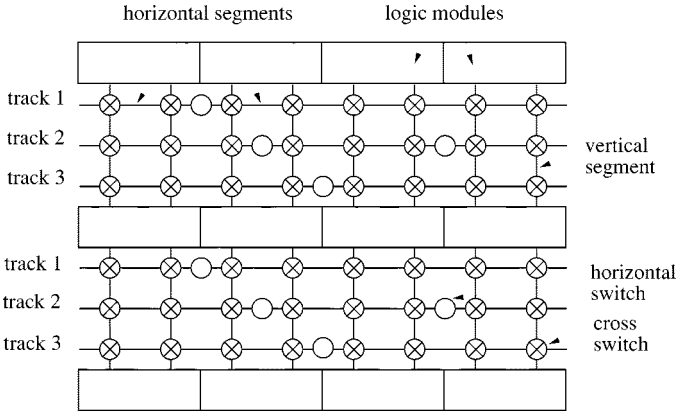

Fig. 1. Row-based FPGA architecture.

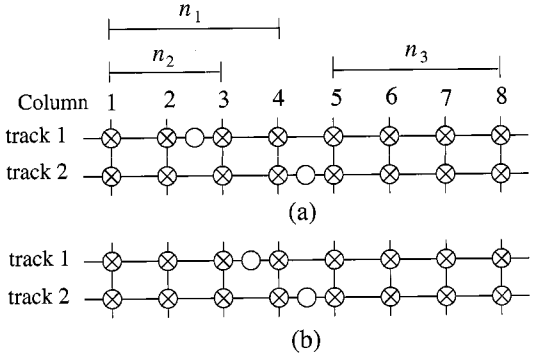

Fig. 2. Two segmented routing examples. (a) Infeasible one-segment routing example. (b) Feasible routing example.

Example 1: Fig. 2 shows a set of three nets n_1, n_2, and n_3 to be routed in two different segmented routing channels, each with two tracks. Each horizontal switch partitions a track into two segments. For example, in Fig. 2(a), track 1 consists of two segments (1, 2) and (3, 8) separated by the horizontal switch located between columns 2 and 3. If each net can use at most one segment for routing, then nets n_1, n_2, and n_3 can not be routed simultaneously using the segmented channel shown in Fig. 2(a); however, they can be routed if each net is allowed to use up to two segments. On the other hand, the three nets are always routable on the segmented channel shown in Fig. 2(b). This

Manuscript received May 28, 1999; revised July 21, 2000. The work of Y.-W. Chang and J.-M. Lin was supported in part by the National Science Council of Taiwan R.O.C. under Grant NSC-87-2215-E-009-041. The work of D. F. Wong was supported in part by the Texas Advanced Research Program under Grant 003658288. This paper was presented in part at the International Conference on Computer-Aided Design, San Jose, CA, November 1998. This paper was recommended by Associate Editor M. Pedram.

Y.-W. Chang and J.-M. Lin are with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: ywchang@cis.nctu.edu.tw; gis87808@cis.nctu.edu.tw).

M. D. F. Wong is with the Department of Computer Sciences, University of Texas, Austin, TX 78712 USA (e-mail: wong@cs.utexas.edu).

Publisher Item Identifier S 0278-0070(01)03540-0.

example shows that segmentation designs could deeply influence the routability of an FPGA.

Rose and Hill [11] emphasized that segmentation distribution would become a key challenge in large-scale FPGA design. They pointed out that physical design for a large-scale FPGA would be difficult because the routing delays and resource utilization could not be handled well and, thus, it is hard to realize the full potential of a large-scale FPGA. A well-designed segmentation can reduce not only routing delays, but also waste of wire lengths. Therefore, the segmentation design problem will become even more important when the age of multimillion gates is coming.

Channel segmentation designs have been studied to some degree in much of the literature [5], [8], [10], [12], [13]. El Gamal *et al.* [5] showed that with appropriate arrangement of segment lengths, a segmented routing channel can achieve comparable routability to a freely customized routing channel (e.g., the routing channel in a standard cell). For the channel segmentation design problem, Roy and Mehendale [12] first presented a stochastic method which approximates a given segment length distribution. Zhu and Wong [13] presented an algorithm for the channel segmentation design problem based also on a stochastic analysis. Given a distribution of nets and routing requirements, they computed the number of segmented tracks of various types required for maximum routability. Pedram *et al.* [10] presented an analytical model for the design and analysis of effective segmented channel architectures . In their approach, the probability density functions for the origination points and the lengths of connections were defined. Based on these functions, they estimated the track number of each type and analyzed the routability of designed segmented channels. Recently, Mak and Wong [8] enumerated the routing patterns for each net and compared the number of tracks required in a channel to accommodate the largest number of patterns to provide high routability and good delay performance for channel segmentation.

Unlike the previous methods that are based on experimental studies, stochastic models, or analytical analysis, we present in this paper a new direction for studying segmentation architectures for row-based FPGAs. Our method is based on graph-theoretic formulation. We first formulate a net matching problem of finding the optimal segmentation architecture for two input routing instances and present a polynomial-time algorithm to solve the problem. Using the solution to the problem as a subroutine, we develop an effective and efficient multilevel matching-based algorithm for general channel segmentation designs. Experimental results show that our method significantly outperforms the previous work. For example, our method achieves average improvements of 8.9% and 18.2% in routability, compared with [8] and [13], respectively. (Note that the most recent work [8] reports the best results among the previous work.)

The remainder of this paper is organized as follows. Section II formulates the segmentation design problem. Section III presents our algorithms for channel segmentation design. Experimental results are reported in Section IV. Finally, we conclude our paper and discuss future research directions in Section V.

II. Problem Formulation

The channel segmentation design problem arising from the row-based FPGA architecture is to determine a channel segmentation architecture to achieve "best" routability under some given constraints (e.g., area and timing constraints). By "best" routability, we mean that the segmentation architecture can accommodate as many routing instances as possible. Here, a routing instance consists of a set of nets for routing, which may correspond to routes in a real circuit design or nets generated from some particular net distribution function.

In this paper, we use the following notations.

- L Length of a channel, measured in the number of columns. We number the columns from 1 to $L+1$.
- T Number of tracks in the channel (area constraint).
- K Maximum number of segments allowed for routing a single net (timing constraint).
- m Number of channel routing instances.
- n Number of nets in each routing instance.

For the channel segmentation, each net is an *interval*, which can be characterized by its leftmost and rightmost points. The leftmost and rightmost points of net i are represented by left_i and right_i, respectively. The *span* of net (interval) i is from left_i to right_i, denoted by $[\mathrm{left}_i, \mathrm{right}_i]$. One net *overlaps* another if the spans of the two nets intersect. A segment *covers* a net (interval) if the span of the net is within the bound of the segment. A set S of segments covers a routing instance I (i.e., a set of nets) if for each net i in I, there exists a segment s in S that covers i and no two nets are covered by the same segment. We use K to model the timing bound for all nets. For the *K-segment routing*, each net can use up to K segments. For $K=1$, a net can be routed on a segment as long as the segment covers the net. When one segment is assigned to a net, the segment is occupied and not allowed to be used for any other net. It is clear that if two nets overlap, they cannot be routed to the same track. For $K \geq 2$, each net can use multiple segments by programming corresponding horizontal switches to connect the segments. However, like one-segment routing, each segment cannot be occupied by more than one net at a time.

The channel segmentation design problem is formulated as follows.

1) *Channel Segmentation Design Problem:* Given L, T, K, m, and n, design a channel segmentation to maximize the success rate for K-segment routing.

For a fixed K, we refer to the problem as the *K-segmentation design problem*. When $K \geq 2$, it is also called the *multisegmentation design problem.*

Note that our formulation is, in fact, more general than most of the previous work. Most previous work considers only some well-defined net distribution functions (e.g., geometric and Poisson distributions, etc.). Ours, however, can not only handle well-defined distributions, but can also deal with arbitrary routing instances.

III. Channel Segmentation Design

We first describe the motivation and framework for our channel segmentation design. Our objective is to construct a segmentation architecture that can accommodate different routing instances in various distribution. To capture the patterns of these input routing instances, we propose a matching-based algorithm to merge routing instances together to generate a super routing instance. Since the intervals in the super routing instance can accommodate those in each routing instance, a segmentation architecture based on the intervals in the super routing instance would lead to good routing success rate for all input routing instances.

More specifically, given m sets of routing instances, each with n_i nets (intervals), $i = 1, \ldots, m$, designing a segmentation to maximize the success rate for one-segment routing is closely related to constructing a set S of segments that can cover each of the m sets of routing instances (one set at a time). It is not difficult to see that using such set S of segments for one-segment routing would result in 100% routing completion. However, there is usually a limitation on the number of tracks T in a routing channel. Therefore, it is not always possible to construct a channel formed by all the segments in S. Nevertheless, the set S of segments still gives a key insight into the optimal segmentation architecture for the given routing instances.

Since S gives the optimal segmentation architecture, our goal is to construct a segmentation structure as close to S as possible. Our method is based on graph-theoretic formulation. We first formulate a net matching problem to obtain a *most economical* set of segments that can cover each of two input routing instances. Based on the weighted bipartite matching theory, we present a polynomial-time optimal algorithm to solve the net matching problem. Using the solution to the problem as a subroutine, we then develop an effective bottom-up matching-based algorithm for the segmentation design for an arbitrary number of input routing instances. We shall first discuss the net matching problem.

A. Net Matching Problem

The net matching problem hopes to find a set of intervals to cover each of two sets of intervals with least length; in other words, if we can optimally solve the net matching problem, we can use it to generate a set of segments with least length for complete routing for each of two routing instances.

Let I be a finite set of intervals (nets). Let $i_1 = [\text{left}_1, \text{right}_1]$ and $i_2 = [\text{left}_2, \text{right}_2]$ be two overlapping intervals. It is obvious that their overlapping length, denoted by $\text{olen}(i_1, i_2)$, can be computed as follows:

$$\text{olen}(i_1, i_2) = \min\{\text{right}_1, \text{right}_2\} - \max\{\text{left}_1, \text{left}_2\}. \quad (1)$$

We define $\text{Merge}(i_1, i_2)$ as the interval $i = [\text{left}, \text{right}]$, where $\text{left} = \min\{\text{left}_1, \text{left}_2\}$ and $\text{right} = \max\{\text{right}_1, \text{right}_2\}$. It is clear that the length of interval i, denoted by $\text{len}(i)$, is given by $\text{right} - \text{left}$ and the total length of all intervals in I, $\text{Length}(I)$ is given by $\sum_{i \in I} \text{len}(i)$.

Let I and J be two finite sets of intervals. A *net matching* M between I and J is a set of ordered pairs of intersecting intervals $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$, where $A = \{i_1, i_2, \ldots, i_k\}$ and $B = \{j_1, j_2, \ldots, j_k\}$ are two sets of distinct intervals from I and J, respectively. We can replace i_1 and j_1 by $\text{Merge}(i_1, j_1)$, replace i_2 and j_2 by $\text{Merge}(i_2, j_2), \ldots$, and replace i_k and j_k by $\text{Merge}(i_k, j_k)$. After the replacement, the set of intervals $I \cup J$ is represented as follows:

$$\text{Union}(I, J) = (I - A) \cup (J - B) \cup \{\text{Merge}(i_1, j_1), \text{Merge}(i_2, j_2), \ldots, \text{Merge}(i_k, j_k)\}. \quad (2)$$

The net matching problem is described as follows.

1) *Net Matching Problem:* Given two finite sets I and J of intervals (nets), find a matching M such that $\text{Length}(\text{Union}(I, J))$ is minimized.

Based on the weighted bipartite matching theory, we present a polynomial-time optimal algorithm for the net matching problem. We reduce the problem to computing the maximum matching in a weighted bipartite graph. Given two finite sets I and J of intervals, we construct a weighted bipartite graph $G = (U, V, E)$ as follows. (See Fig. 3 for an illustration.) For each interval i in I (j in J), we introduce a vertex $u_i (v_j)$ in the set $U(V)$ of vertices. For each pair of overlapping intervals $p, q, p \in I$ and $q \in J$, connect u_p to v_q by an edge $e_{pq} = (u_p, v_q)$ with a weight computed by the weight function $\alpha : E \to Z^+$ defined as follows:

$$\alpha(e_{pq}) = \min\{\text{right}_p, \text{right}_q\} - \max\{\text{left}_p, \text{left}_q\}. \quad (3)$$

Then, we can apply a maximum weighted bipartite matching algorithm [9] on G to solve the net matching problem optimally.

Note that $\alpha(e_{pq})$ gives the overlap length between intervals p and q. Intuitively, this weight function measures the "similarity" between two intervals—the greater the weight, the more similar the two corresponding intervals. By merging intervals with greatest similarity, we can obtain a most economical (i.e., minimum total length) set of segments that covers each of two input interval sets.

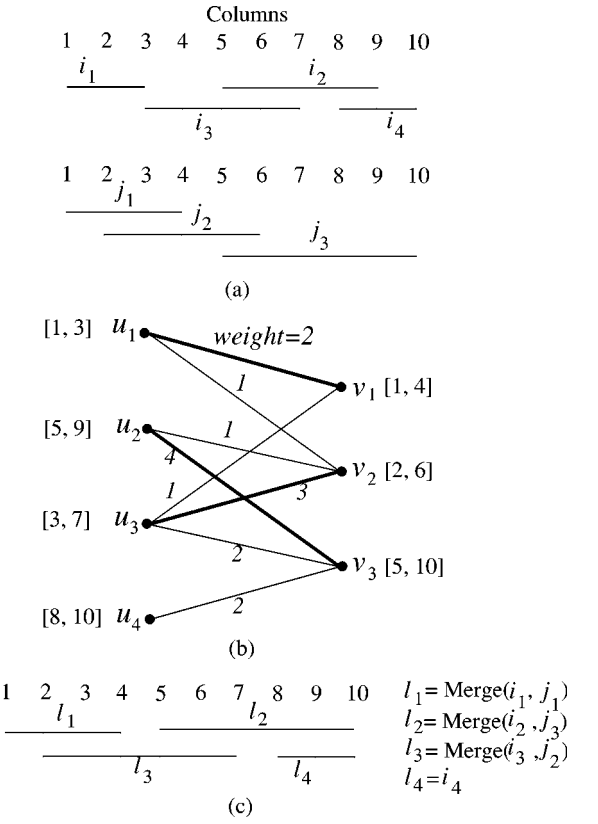

Fig. 3. Matching and merging example. (a) Two sets of nets. (b) Corresponding weighted bipartite graph. (c) Matching result for the two sets of nets.

A *matching* $\hat{M}$ of a *graph* $H = (V, E)$ is a subset of the edges with the property that no two edges of $\hat{M}$ share the same vertex. Edges in $\hat{M}$ are called *matched* edges; otherwise, they are *unmatched*. Let $\text{Matched}(I, J)$ be the set of the matched edges in a weighted bipartite matching on the graph induced by the finite sets I and J of intervals and $\text{Weight}(F)$, $F \subseteq E$ be the total weight of the edges in F. We have the following lemma and theorem.

Lemma 1: $\text{Length}(\text{Union}(I, J)) = \text{Length}(I) + \text{Length}(J) - \text{Weight}(\text{Matched}(I, J))$.

Proof: By (2), $\text{Union}(I, J) = (I - A) \cup (J - B) \cup \{\text{Merge}(i_1, j_1), \text{Merge}(i_2, j_2), \ldots, \text{Merge}(i_k, j_k)\}$, we have

$$\begin{aligned}
&\text{Length}(\text{Union}(I, J)) \\
&\quad = \text{Length}((I - A) \cup (J - B) \cup \{\text{Merge}(i_1, j_1), \\
&\qquad \text{Merge}(i_2, j_2), \ldots, \text{Merge}(i_k, j_k)\}) \\
&\quad = \text{Length}(I) - \text{Length}(A) + \text{Length}(J) - \text{Length}(B) \\
&\qquad + \text{Length}(\text{Merge}(i_1, j_1) + \text{Merge}(i_2, j_2) \\
&\qquad + \cdots + \text{Merge}(i_k, j_k)) \\
&\quad = \text{Length}(I) + \text{Length}(J) - \text{Length}(A) - \text{Length}(B) \\
&\qquad + \sum_{p=1}^{k} \text{Length}(\text{Merge}(i_p, j_p)) \\
&\quad = \text{Length}(I) + \text{Length}(J) - \text{Length}(A) - \text{Length}(B) \\
&\qquad + \sum_{p=1}^{k} (len(i_p) + len(j_p) - \text{olen}(i_p, j_p))
\end{aligned}$$

$$
\begin{aligned}
&= \mathrm{Length}(I) + \mathrm{Length}(J) - \mathrm{Length}(A) - \mathrm{Length}(B) \\
&\quad + \sum_{p=1}^{k} len(i_p) + \sum_{p=1}^{k} len(j_p) - \sum_{p=1}^{k} \mathrm{olen}(i_p, j_p) \\
&= \mathrm{Length}(I) + \mathrm{Length}(J) - \mathrm{Length}(A) - \mathrm{Length}(B) \\
&\quad + \mathrm{Length}(A) + \mathrm{Length}(B) - \mathrm{Weight}(\mathrm{Matched}(I, J)) \\
&= \mathrm{Length}(I) + \mathrm{Length}(J) - \mathrm{Weight}(\mathrm{Matched}(I, J)).
\end{aligned}
$$

■

Theorem 1: The maximum bipartite weighted matching algorithm optimally solves the net matching problem in $O((n_1 + n_2)^3)$ time, where n_1 and n_2 are the numbers of nets in the two input sets.

Proof: Consider two finite sets I_1 and I_2 of intervals with n_1 and n_2 nets, respectively. We apply the maximum weighted bipartite matching algorithm to merge these two sets of nets. According to Lemma 1, the total interval length of the resulting merged set, $\mathrm{Length}(\mathrm{Union}(I_1, I_2))$, is given by $\mathrm{Length}(I_1) + \mathrm{Length}(I_2) - \mathrm{Weight}(\mathrm{Matched}(I_1, I_2))$. Since $\mathrm{Length}(I_1)$ and $\mathrm{Length}(I_2)$ are fixed, $\mathrm{Length}(\mathrm{Union}(I_1, I_2))$ has the minimum value when $\mathrm{Weight}(\mathrm{Matched}(I_1, I_2))$ is maximized. The weighted bipartite matching algorithm guarantees to find such a maximum value and, thus, the net matching problem is optimally solved. The time complexity of the maximum weighted bipartite matching algorithm is $O((n_1 + n_2)^3)$ [9]. The theorem thus follows. ■

Example 2: Fig. 3(a) shows two sets $I = \{i_1, i_2, i_3, i_4\}$ and $J = \{j_1, j_2, j_3\}$ of intervals (nets). The induced weighted bipartite graph is given in Fig. 3(b). The span of net i, $[\mathrm{left}_i, \mathrm{right}_i]$ is shown next to its corresponding vertex. The weight for each edge is computed by the function α and shown beside the edge. The maximum weighted bipartite matching M between $U = \{u_1, u_2, u_3, u_4\}$ and $V = \{v_1, v_2, v_3\}$ is illustrated in Fig. 3(b) by heavy lines. In this example, $M = \{(u_1, v_1), (u_2, v_3), (u_3, v_2)\}$. Note that u_4 is unmatched. Fig. 3(c) shows the resulting configuration of replacing i_1 and j_1 by $\mathrm{Merge}(i_1, j_1)$, i_2 and j_3 by $\mathrm{Merge}(i_2, j_3)$, and i_3 and j_2 by $\mathrm{Merge}(i_3, j_2)$. Let $l_1 = \mathrm{Merge}(i_1, j_1), l_2 = \mathrm{Merge}(i_2, j_3), l_3 = \mathrm{Merge}(i_3, j_2)$, and $l_4 = i_4$. After the replacement, the set of intervals $I \cup J$ becomes $\mathrm{Union}(I, J) = \{l_1, l_2, l_3, l_4\}$. The reader can verify that $\mathrm{Length}(\mathrm{Union}(I, J)) = \mathrm{len}(l_1) + \mathrm{len}(l_2) + \mathrm{len}(l_3) + \mathrm{len}(l_4) = 3 + 5 + 5 + 2 = 15$ is the minimum possible total union length for merging I and J. (Note that $\mathrm{Length}(\mathrm{Union}(I, J)) = \mathrm{Length}(I) + \mathrm{Length}(J) - \mathrm{Weight}(\mathrm{Matched}(I, J)) = (2 + 4 + 4 + 2) + (3 + 4 + 5) - (2 + 4 + 3) = 15$).

B. Segmentation Design Algorithm

Our design algorithm consists of three stages: 1) the matching-and-merging stage; 2) the tuning stage; and 3) the filling stage. In the matching-and-merging stage, we repeatedly apply the aforementioned weighted bipartite matching algorithm to merge input routing instances and find a set I of intervals that can cover each of the input routing instances. In the tuning stage, we find a set I' of intervals from I, $I' \subseteq I$, which can be packed (routed) into T tracks. In the filling stage, we determine the switch locations on the tracks and fill the empty space between each pair of intervals in the T tracks to form a set of segments.

Since the net matching problem guarantees to find a set of intervals with least length to cover each of two routing instances, the length of resulting intervals by repeatedly applying this approach to merge all routing instances would be smaller than total intervals of routing instances and it is more feasible to build segmentation for complete routing from resulting intervals than total interval of routing instances.

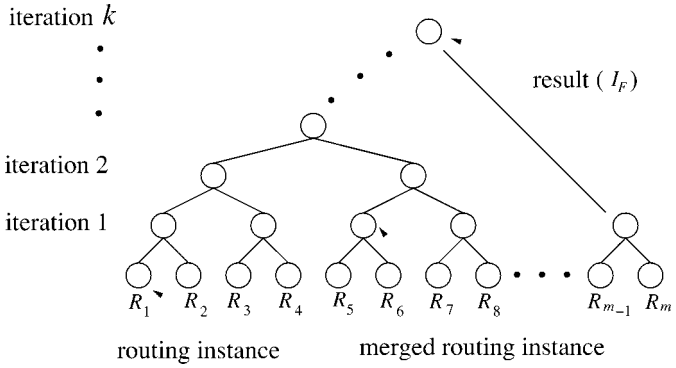

Fig. 4. Matching process.

The matching and merging stage proceeds in a tree-like bottom-up manner. (The whole matching and merging process is illustrated in Fig. 4.) Given m routing instances $R_1, R_2, \ldots, R_m$, each with respective $n_1, n_2, \ldots, n_m$ nets, we apply the aforementioned weighted bipartite matching algorithm to merge R_1 and R_2, R_3 and R_4, and R_5 and $R_6, \ldots$. (See the procedure $\mathrm{Match\_and\_Merge}(\,)$ in Lines 5 and 8 of Fig. 6.) If m is odd, then R_m remains unmerged. After the merge, the number of resulting instances reduces to $\lceil m/2 \rceil$. Then, the same merging process repeats for the new $\lceil m/2 \rceil$ routing instances. The process proceeds level by level in a bottom-up manner until a final merged routing instance is obtained (see Fig. 4).

Let I_F be the set of the intervals in the final merged routing instance. We have the following theorem.

Theorem 2: I_F covers R_i, $\forall i, 1 \le i \le m$.

Proof: Let I_{11} be the resulting set of intervals for matching and merging two routing instances R_1 and R_2 in Iteration 1 (see Fig. 4). Suppose $\{i_1, \ldots, i_p\}$ ($\{j_1, \ldots, j_q\}$) are intervals in R_1 (R_2). Without loss of generality, let i_1 and j_1, i_2 and $j_2, \ldots, i_k$, and j_k be matched intervals $k \le p$ and $k \le q$. Let $A = \{i_1, \ldots, i_k\}$ and $B = \{j_1, \ldots, j_k\}$. It is obvious that $i_1, \ldots, i_k$ $(j_1, \ldots, j_k)$ can be covered by $\mathrm{Merge}(i_1, j_1), \ldots, \mathrm{Merge}(i_k, j_k)$, respectively. If $k < p$ $(k < q)$, for each interval in $\{i_{k+1}, \ldots, i_p\}$ ($\{j_{k+1}, \ldots, j_q\}$), it can be covered by itself in $R_1 - A$ ($R_2 - B$). By (2), we have $I_{11} = \mathrm{Union}(R_1, R_2) = (R_1 - A) \cup (R_2 - B) \cup \{\mathrm{Merge}(i_1, j_1), \ldots, \mathrm{Merge}(i_k, j_k)\}$. Thus, it is obvious that every interval i in R_1 (j in R_2) can be covered by some intervals in I_{11} and no two intervals in R_1 (R_2) are covered by the same segment. Also, R_3 and R_4, R_5 and $R_6, \ldots$ can be covered by $I_{12}, I_{13}, \ldots$, respectively. Similarly, let $I_{21}, I_{22}, \ldots$ be the matching-and-merging results of I_{11} and I_{12}, , I_{13} and $I_{14}, \ldots$. Then, I_{11} and I_{12}, I_{13} and $I_{14}, \ldots$ can be covered by $I_{21}, I_{22}, \ldots$, respectively. This process continues as the matching-and-merging stage proceeds. It is obvious that $R_i, \forall i, 1 \le i \le m$ can be covered by I_F. The theorem thus follows. ■

By Theorem 2, using a set S of segments covering I_F for one-segment routing can route all routing instances $R_1, R_2, \ldots, R_m$. As mentioned earlier, however, there is usually a limitation on the number of tracks T in a routing channel. Therefore, it is not always possible to construct a channel formed by all the segments in S.

In the tuning and the filling stages, we construct a segmentation of T tracks from the final merged routing instance I_F. First, we apply the basic left-edge algorithm [7] to route the intervals in I_F. (See the procedure $\mathrm{Route\_by\_Left\_Edge}(\,)$ in Line 11 of Fig. 6.) We then sort the resulting tracks in the nonincreasing order of their total lengths occupied by the intervals. The first T tracks are chosen for further construction and the tuning stage is done. (See the procedure $\mathrm{Tune\_Track}(\,)$ in Line 12 of Fig. 6.) After the tuning stage, it may contain an empty space between a pair of intervals. In the filling stage, we determine

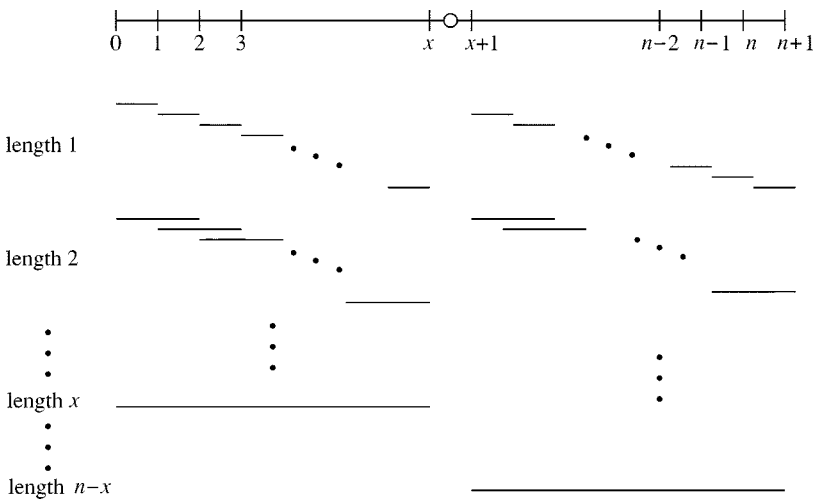

Fig. 5. Routable configurations for the segmented track.

```
Algorithm: Seg_Designer
Input: m—Number of routing instances;
       R[m]—R[i] is the i-th routing instance, 1 ≤ i ≤ m;
       T —Maximum number of tracks in the channel.
Output: S—The designed segmentation.

Stage 1: Matching and Merging
1  iteration ← ⌈log2 m⌉;
2  for i ← 1 to iteration do
3      if (m is even)
4          for j ← 1 to m/2 do
5              R[j] ← Match_and_Merge(R[2j − 1], R[2j]);
6      else
7          for j ← 1 to ⌊m/2⌋ do
8              R[j] ← Match_and_Merge(R[2j − 1], R[2j]);
9          R[j + 1] ← R[m];
10     m ← ⌈m/2⌉;
Stage 2: Tuning
11 Track[t] ← Route_by_Left_Edge(R[1]);
12 Track[T] ← Tune_Tracks(Track[t]);
Stage 3: Filling
13 S ← Fill_Space(Track[T]);
14 return S.
```

Fig. 6. Algorithm for segmentation design.

the switch location to fill each empty space to construct a set of segments. In particular, we intend to optimize not only the routability for the given routing instances (done in the matching-and-merging stage), but also that for unknown instances. We apply the following theorem in the filling stage to guide the placement of horizontal switches to further optimize the routability for the unknown ones.

Theorem 3: For one-segment routing, a two-segment routing track can cover the maximum number of net patterns when the two segments are of equal length (i.e., the switch is placed in the middle of the two segments).

Proof: Consider a track of length $n+1$ (see Fig. 5). The column numbers range from 0 to $n+1$. Each net under the track can be routed on the segmented track. Suppose the switch is placed between Columns x and $x+1$. For one-segment routing, the segment on the left side of the switch can cover x nets of length 1, $x-1$ nets of length $2,\ldots$, or 1 net of length x. Similarly, the segment on the right side of the switch can cover $n-x$ nets of length 1, $n-x-1$ nets of length $2,\ldots$, or 1 net of length $n-x$. Therefore, the segmented track can cover all $x+(x-1)+\cdots+1$ nets on the left side of the switch and $(n-x)+(n-x-1)+\cdots+1$ nets on the right side of the switch. Therefore, the number of combinations of nets that can be covered by the segmented track is given by the following function L:

$$\begin{aligned} L(x) &= (x+(x-1)+\cdots+1) \\ &\quad \times ((n-x)+(n-x-1)+\cdots+1) \\ &= \left(\frac{x(x+1)}{2}\right)\left(\frac{(n-x)(n-x+1)}{2}\right) \\ &= \frac{x^4-2nx^3+(n^2-n-1)x^2+(n^2+n)x}{4}. \end{aligned}$$

$L(x)$ has the extreme values as $L'(x)=0$, $x>0$. We have

$$\begin{aligned} L'(x) &= 4x^3-6nx^2+2(n^2-n-1)x+n(n+1)=0 \\ &\Longrightarrow (2x-n)(2x^2-2nx-(n+1))=0 \\ &\Longrightarrow x=n/2. \end{aligned}$$

To see that $L(x)$ has the maximum value as $x=n/2$, we compute $L''(n/2)$

$$\begin{aligned} L''(x) &= 12x^2-12nx+2(n^2-n-1) \\ L''\left(\frac{n}{2}\right) &= 12\left(\frac{n}{2}\right)^2-12n\left(\frac{n}{2}\right)+2(n^2-n-1) \\ &= 3n^2-6n^2+2n^2-2n-2 \\ &= -n^2-2n-2 \\ &= -(n+1)^2-1 \\ &< 0. \end{aligned}$$

Hence, $L(x)$ has the maximum value when $x=n/2$.

Therefore, a two-segment routing track can cover the maximum number of nets when the two segments have equal lengths (switch is placed in the middle of track) for one-segment routing. ■

By Theorem 3, if there are only two intervals on a track that is separated by empty space, we would better place a horizontal switch on the position that makes the two resulting segments most balanced in length. However, the number of intervals in a track is usually larger than two. We therefore process each pair of neighboring intervals serially from left to right according to Theorem 3. The procedure Fill_Space( ) listed in Line 15 of Fig. 6 finds a better position for placing a horizontal switch in the empty space, if any, between every two neighboring intervals. The whole segmentation design algorithm is summarized in Fig. 6.

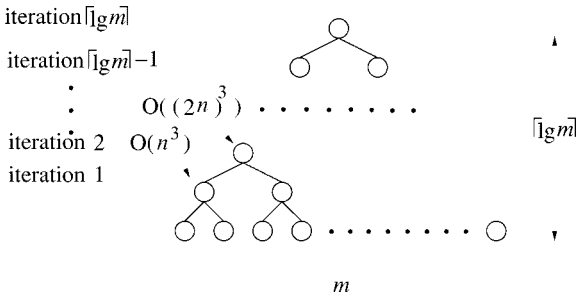

Fig. 7. Time complexity of matching-and-merging process.

Theorem 4: Algorithm Seg_Designer runs in $O(m^3n^3)$ time, where m is the number of input routing instances and n is the maximum number of nets in a routing instance.

Proof: Given m input routing instances, each with at most n nets by Theorem 1, we need $O(n^3)$ time to merge two routing instances in Iteration 1 using the maximum weighted bipartite matching algorithm. (See Fig. 7 for the outline of the matching-and-merging process of our method.) In the worst case, if nets in two routing instances do not match at all, the newly generated routing instance will contain $2n$ nets after Iteration 1. Then, it needs $O((2n)^3)$ time to solve the net matching problem in the next iteration. Similarly, a routing instance has at most $2^{\lceil \lg m \rceil - 1}n$ nets after Iteration $\lceil \lg m \rceil - 1$ and it needs $O((2^{\lceil \lg m \rceil - 1}n)^3)$ time to solve the net matching problem in the final iteration. Therefore, the time complexity $T(m,n)$ of the algorithm is given by

$$
\begin{aligned}
&T(m,n)\\
&\quad \le \left\lceil \frac{m}{2} \right\rceil O(n^3)\\
&\qquad + \left\lceil \frac{m}{4} \right\rceil O((2n)^3) + \cdots + O\left(\left(2^{\lceil \lg m \rceil - 1}n\right)^3\right)\\
&\quad = O\left(\left\lceil \frac{m}{2} \right\rceil n^3 + \left\lceil \frac{m}{4} \right\rceil (2n)^3 + \cdots + \left(2^{\lceil \lg m \rceil - 1}n\right)^3\right)\\
&\quad = O\left(\frac{mn^3}{2} + 2mn^3 + \cdots + 2^{3(\lceil \lg m \rceil - 1)}n^3\right)\\
&\quad = O\left(\frac{\frac{mn^3}{2}(4^{\lg m} - 1)}{4 - 1}\right)\\
&\quad = O\left(\frac{mn^3}{6}(4^{\lg m} - 1)\right)\\
&\quad = O\left(\frac{mn^3}{6}(m^2 - 1)\right)\\
&\quad = O\left(\frac{m^3n^3}{6} - \frac{mn^3}{6}\right)\\
&\quad = O\left(m^3n^3\right).
\end{aligned}
$$

■

By Theorem 4, the time complexity of our algorithm is given by $O(m^3n^3)$, where m is the number of input routing instances and n is the maximum number of nets in an input routing instance. Note that empirically the number of resulting intervals per routing instance grows only linearly as the matching and merging process proceeds, instead of exponentially (in logarithmic steps $n, 2n, 4n, \ldots, 2^{\lceil \lg m \rceil}n$) as shown in the theoretic analysis (see Fig. 7). This will be clear when we show the empirical results in Section IV.

For K-segmentation design ($K \ge 2$), all we need to do is to split each segment into K sections of equal length right after the aforementioned procedures. However, since the minimum length of a segment is one, it is impossible to partition an interval of length smaller than $2K - 1$ into K segments. Specifically, we can partition an interval of length l into at most $\lceil l/2 \rceil$ segments.

TABLE I
ONE-SEGMENT ROUTING RESULTS ($L = 100, T = 36, D = 12, K = 1$)

	$f(l)$	[13]		Ours	
		d_T	d_T/T	d_T	d_T/T
D_1	(1, 1, 1, 1, 1)	24	0.67	31	0.86
D_2	(.1, .3, .5, .8, 1)	32	0.89	34	0.94
D_3	(1, .8, .5, .3, .1)	23	0.64	28	0.78
D_4	(1, .5, .3, .1, 0)	26	0.72	27	0.75
D_5	(1, .5, .3, .5, 1)	20	0.56	30	0.83
D_6	(.2, .5, 1, .5, .2)	29	0.81	33	0.92
D_7	(1, .2, .1, 0, 0)	22	0.61	27	0.75
Ge	γ^l,$\gamma = 0.95$	21	0.58	28	0.78
No	$\mu = 35$, $\sigma^2 = 100$	25	0.70	31	0.86
Po	$\lambda^l e^{-\lambda}/l!$, $\lambda = 20.0$	20	0.56	31	0.86
average		24.2	0.674	30.0	0.833

TABLE II
TWO-SEGMENT ROUTING RESULTS ($L = 100, T = 36, D = 12, K = 2$)

	$f(l)$	[13]		Ours	
		d_T	d_T/T	d_T	d_T/T
D_1	(1, 1, 1, 1, 1)	29	0.81	33	0.92
D_2	(.1, .3, .5, .8, 1)	34	0.94	35	0.97
D_3	(1, .8, .5, .3, .1)	28	0.78	32	0.89
D_4	(1, .5, .3, .1, 0)	26	0.72	32	0.89
D_5	(1, .5, .3, .5, 1)	27	0.75	33	0.92
D_6	(.2, .5, 1, .5, .2)	33	0.92	35	0.97
D_7	(1, .2, .1, 0, 0)	22	0.61	28	0.78
Ge	γ^l,$\gamma = 0.95$	25	0.70	30	0.83
No	$\mu = 35$, $\sigma^2 = 100$	32	0.89	33	0.92
Po	$\lambda^l e^{-\lambda}/l!$, $\lambda = 20.0$	30	0.83	32	0.89
average		28.6	0.795	32.3	0.889

IV. EXPERIMENTAL RESULTS

We implemented our segmentation design algorithms in the C++ programming language on a personal computer with a Pentium 166 microprocessor and 32-MB random access memory. The weighted bipartite matching code was adopted from the public LEDA package. The routability of the architectures designed by our algorithms was tested using the one-segment and the multisegment routing algorithms by Zhu *et al.* [13]. In addition to the notation mentioned in Section II, the following notation is also needed to explain our experimental procedures.

D Maximum number of net terminals at a column.

$f(l)$ Probability that a net is of length l.

The input routing instances were generated by the programs used in [8] and [13]. The first set of ten routing instances is based on the parameters $L = 100, T = 36$, and $D = 12$, which are close to the row-based architectures used by Actel FPGAs [1]. Distributions $D_i, i = 1, 2, \ldots, 7$ are defined as follows. If $f(l) = (p_1, p_2, p_3, p_4, p_5)$, then the probability that a net has length between $0.2(j-1)L$ and $0.2jL$ is equal to $p_j / \sum_{1 \le k \le 5} p_k$. "Ge," "No," and "Po" are geometric, normal, and Poisson distributions, respectively. For each net distribution, 300 routing instances were generated.

The ratio of routing success was measured by the *threshold density* d_T defined in [13]. d_T means threshold for the largest channel densities in one distribution such that larger than 90% routing instances in the distribution with channel density d_T can be successfully routed. Obviously, a larger d_T is more better.

Tables I and II list the respective comparisons for one- and two-segment routing between our designs and those in Zhu *et al.* [13] based on the parameters $L = 100$, $T = 36$, and $D = 36$, which were used

TABLE III
TWO-SEGMENT ROUTING RESULTS ($L = 20, T = 18, D = 6, K = 2$).

	$f(l)$	[13]		[8]		Ours	
		d_T	d_T/T	d_T	d_T/T	d_T	d_T/T
b_1	(1, 1, 1, 1, 1)	15	0.83	17	0.94	17	0.94
b_2	(1, .8, .5, .3, .1)	14	0.78	15	0.83	16	0.89
b_3	(1, .5, .3, .1, 0)	13	0.72	14	0.78	17	0.94
b_4	(1, .5, .3, .5, 1)	13	0.72	16	0.89	16	0.89
b_5	(.2, .5, 1, .5, .2)	16	0.89	15	0.83	17	0.94
b_6	(1, .2, .1, 0, 0)	11	0.61	14	0.78	17	0.94
Ge	$\gamma^l, \gamma = 0.7$	12	0.67	13	0.72	16	0.89
No	$\mu = 4, \sigma^2 = 10$	16	0.89	15	0.83	17	0.94
Po	$\lambda^l e^{-\lambda}/l!, \lambda = 3.0$	13	0.72	15	0.83	16	0.89
Avg		13.7	0.759	14.9	0.826	16.6	0.917

TABLE IV
THREE-SEGMENT ROUTING RESULTS ($L = 50, T = 24, D = 8, K = 3$)

	$f(l)$	[13]		[8]		Ours	
		d_T	d_T/T	d_T	d_T/T	d_T	d_T/T
b_1	(1, 1, 1, 1, 1)	21	0.88	22	0.92	23	0.96
b_2	(1, .8, .5, .3, .1)	17	0.71	21	0.88	24	1.00
b_3	(1, .5, .3, .1, 0)	18	0.75	21	0.88	21	0.88
b_4	(1, .5, .3, .5, 1)	19	0.79	22	0.92	21	0.88
b_5	(.2, .5, 1, .5, .2)	22	0.92	22	0.92	22	0.92
b_6	(1, .2, .1, 0, 0)	17	0.71	20	0.83	22	0.92
Ge	$\gamma^l, \gamma = 0.875$	20	0.83	20	0.83	22	0.92
No	$\mu = 8, \sigma^2 = 15$	21	0.88	20	0.83	21	0.88
Po	$\lambda^l e^{-\lambda}/l!, \lambda = 8.0$	19	0.79	19	0.79	23	0.96
Avg		19.3	0.807	20.8	0.867	22.1	0.924

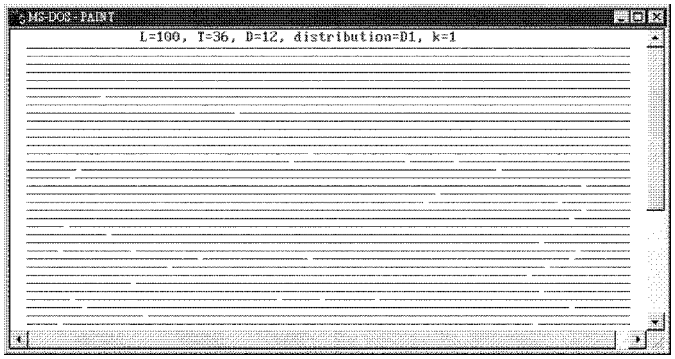

Fig. 8. Channel segmentation designed by our algorithm ($L = 100, T = 36, D = 12, K = 1$, distribution D_1).

in [13]. The results show that our designs outperform those in [13] by *averages* of 24% and 12.9% improvements in routability for one- and two-segment routing, respectively.

The parameters used for net distributions in [8] were $L = 20, T = 18$, and $D = 6$ for two-segment design and $L = 50, T = 24$, and $D = 8$ for three-segment design. The results, reported in Tables III and IV, show that our method significantly outperforms the previous work in [8] and [13]. Our designs achieve *averages* of 17.9% and 8.9% improvements in routability compared with the work in [13] and the most recent work in [8], respectively. Fig. 8 shows our one-segment channel segmentation design for distribution D_1 using the parameters $L = 100, T = 36, D = 12$, and $K = 1$.

We also performed experiments to explore the effect of applying different pairings in the matching-and-merging stage. Table V lists the results for the two-segment design. In the original experiments, we used the pairings $(R_1, R_2), (R_3, R_4), \ldots$ Here, we tested the pairings $(R_2, R_3), (R_4, R_5), \ldots$. Other procedures remain the same. The ordering of merging sequence may affect the routability of our design;

TABLE V
ROUTABILITY COMPARISONS OF MERGING ROUTING INSTANCES WITH DIFFERENT PAIRINGS FOR TWO-SEGMENT ROUTING

	$f(l)$	original pairings		different pairings		difference
		d_T	d_T/T	d_T	d_T/T	
D_1	(1, 1, 1, 1, 1)	32	0.89	33	0.92	1
D_2	(.1, .3, .5, .8, 1)	35	0.97	34	0.94	1
D_3	(1, .8, .5, .3, .1)	32	0.89	31	0.86	1
D_4	(1, .5, .3, .1, 0)	30	0.83	26	0.72	4
D_5	(1, .5, .3, .5, 1)	32	0.89	32	0.89	0
D_6	(.2, .5, 1, .5, .2)	33	0.92	33	0.92	0
D_7	(1, .2, .1, 0, 0)	20	0.56	23	0.64	3
Ge	$\gamma^l, \gamma = 0.95$	30	0.83	26	0.72	4
No	$\mu = 35, \sigma^2 = 100$	32	0.89	33	0.92	1
Po	$\lambda^l e^{-\lambda}/l!, \lambda = 20.0$	31	0.86	32	0.89	1
	average	30.7		30.3		1.6

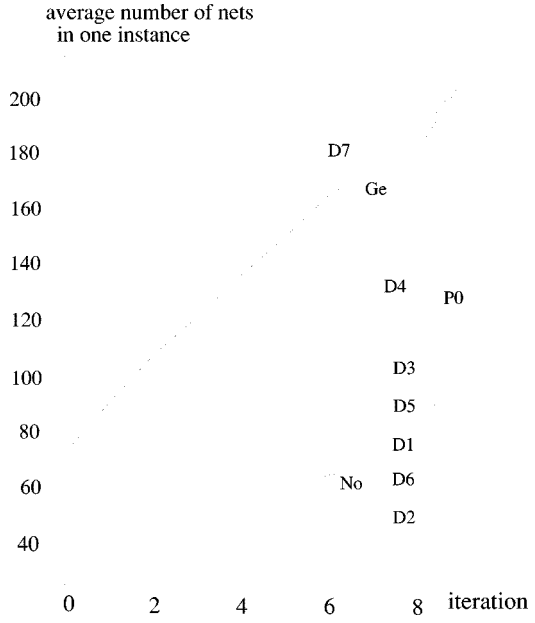

Fig. 9. Average number of nets in one instance at each iteration.

as shown in Table V, there is an average variation of 5% in individual d_T's by using the two pairing schemes. However, the average d_T values for the ten distributions remain about the same for the two schemes. The results show that no matter what pairing scheme is applied, our matching-based approach performs well and the overall routability of the designs is quite stable across different pairing schemes.

Our design algorithms are quite efficient. The empirical runtimes for the largest set of designs ($L = 100, T = 36, D = 12$, and $K = 2$) ranged from 10 s for distribution D_2 to 78 s for distribution D_7 (with an average runtime of about 30 s). Although the theoretic analysis gives $O(m^3 n^3)$-time complexity for our algorithm, the empirical runtime for the m term is close to $O(m \lg m)$ instead of $O(m^3)$. The reason is that most of the nets in two input routing instances were merged together. Therefore, the number of nets in a merged instance grew only linearly instead of exponentially. In Fig. 9, the average number of nets per routing instance is plotted as a function of the number of iterations (in the matching-and-merging stage) for each of the ten distributions listed in Table I. The curves in Fig. 9 exhibit the linearity of the empirical growth rates for the average number of nets per routing instance.

V. CONCLUSION

We have presented a new direction for studying FPGA segmentation design based on the graph-matching formulation. Different from

the previous work that works on some sort of approximation (e.g., to fit into particular distribution functions) in the beginning, our method targets at the optimal architecture directly (e.g., the set I_F of segments resulted from the matching-and-merging stage can cover each input routing instance for one-segment designs—100% routing success rate if no area constraint). We believe that targeting at the optimal architecture directly (ours) is the major factor that leads to substantially better performance than working on the approximation (previous work) in the beginning.

We have shown that the matching-based approach is effective and efficient for the channel segmentation design. In particular, the approach is also very flexible, which makes it a promising alternative to more complex segmentation designs. Future work lies in the extension to higher order (e.g., two- and three-dimensional) segmentation designs. Also, as shown in the experimental results, the pairing of input routing instances in the matching-and-merging stage has some impact on the quality of the channel segmentation design. To explore the best pairing scheme, we propose to apply a general weighted graph-matching algorithm to find the minimum cost pairing among given routing instances.

ACKNOWLEDGMENT

The authors would like to thank Dr. K. Zhu and Prof. W. K. Mak for providing packages for segmentation designs and the anonymous reviewers for their constructive comments.

REFERENCES

[1] *FPGA Data Book and Design Guide*, Actel Corp., Sunnyvale, CA, 1996.
[2] B. Fallah and J. Rose, "Timing-driven routing segment assignment in FPGAs," in *Proc. Can. Conf. VLSI*, Halifax, NS, Canada, Oct. 1992, pp. 18–20.
[3] Y.-W. Chang, J.-M. Lin, and D. F. Wong, "Graph matching-based algorithms for FPGA segmentation design," *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 34–39, Nov. 1998.
[4] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat, and A. Mohsen, "An architecture for electrically configurable gate arrays," *IEEE J. Solid-State Circuits*, vol. 24, pp. 394–398, Apr. 1989.
[5] A. El Gamal, J. Greene, and V. Roychowdhury, "Segmented channel routing is nearly as efficient as channel routing (and just as hard)," in *Proc. Advanced Research VLSI*, Santa Cruz, CA, Mar. 1991, pp. 193–221.
[6] M. Khellah, S. Brown, and Z. Vranesic, "Modeling routing delays in SRAM-based FPGAs," in *Proc. Canadian Conf. VLSI*, Banff, Alberta, Canada, Nov. 1993, pp. 14–16.
[7] M. J. Lorenzetti and D. S. Baeder, "Routing," in *Routing in Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, Eds. Redwood City, CA: Benjamin Cummings, 1988, ch. 5.
[8] W. K. Mak and D. F. Wong, "Channel segmentation design for symmetrical FPGAs," *Proc. IEEE Conf. Computer Design*, pp. 496–501, Oct. 1997.
[9] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[10] M. Pedram, B. S. Nobandegani, and B. T. Preas, "Design and analysis of segmented routing channels for row-based FPGAs," *IEEE Trans. on Computer Aided Design*, vol. 13, no. 12, pp. 1470–1479, Dec. 1994.
[11] J. Rose and D. Hill, "Architectural and physical design challenges for one-million gate FPGA's and beyond," in *ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Monterey, CA, Feb. 1997, pp. 129–132.
[12] K. Roy and M. Mehendale, "Optimization of channel segmentation for channelled architecture FPGAs," *Proc. IEEE Custom Intergrated Circuits Conf.*, pp. 4.4.1–4.4.4, May 1992.
[13] K. Zhu and D. F. Wong, "Segmented channel segmentation design for row-based FPGAs," *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 26–29, Nov. 1992.

On Diagnosis and Diagnostic Test Generation for Pattern-Dependent Transition Faults

Irith Pomeranz and Sudhakar M. Reddy

***Abstract*—We propose a method of modeling pattern dependence as part of the existing delay fault models without incurring the complexity of considering physical effects that cause pattern dependence. We apply the method to transition faults. We define the conditions under which two pattern-dependent transition faults can be said to be distinguished by a given test set. We provide experimental results to demonstrate the diagnostic resolutions obtained under the proposed model. We also present conditions for identifying pairs of indistinguishable pattern-dependent transition faults and propose a procedure for generating diagnostic tests for distinguishable pattern-dependent transition faults.**

***Index Terms*—fault diagnosis, pattern-dependent delay defects, transition faults.**

I. INTRODUCTION

Diagnosis of delay faults is important for identifying and possibly correcting timing-related errors in the design and manufacturing processes of a high-performance chip. Diagnosis of delay faults of various types was considered in [1]–[4]. Path delay faults were considered in [1] and [4] and gate delay faults were considered in [2]. Delays resulting from process variations were considered in [3]. In these works, delay faults are assumed to be independent of the input patterns applied to the circuit. Thus, if two different tests detect the same fault, faulty behavior is expected under both tests in the presence of the fault. In [5]–[7], it was shown that the delays throughout a circuit may be pattern dependent. This is because certain signal transitions may be speeded up or delayed depending on the values of other lines in the circuit. Pattern dependence of delays in the emerging technology of silicon on insulator is considered one of the challenges in using this technology [8]. As a result of delays being pattern dependent, delay defects may show pattern-dependent behavior as well. This implies that two tests detecting the same delay defect may not both result in faulty output values in the presence of the defect. Consequently, the following situation may occur.

Consider two different tests that detect the same fault. Consider a defect that has the same effect as the fault (i.e., it delays the same transitions by the same amount as the fault). However, it exhibits pattern dependence. Assuming that the defect is present in the circuit, it is possible that because of values of other lines in the circuit, the transitions on the faulty lines would occur on time under the first test, resulting in fault-free output values while the same transitions may be delayed under the second test, resulting in faulty output values.

The difficulty in working with pattern-dependent defects results from the fact that the physical effects causing the pattern dependence are complex. Consequently, fault diagnosis, which takes all these effects into account, may be complex if not impossible because of the inability to model accurately chip behavior in the presence of defects.

Manuscript received May 22, 2000; revised November 15, 2000. This work was supported in part by the National Science Foundation under Grant MIP-9725053 and in part by the Semiconductor Research Corporation under Grant 98-TJ-645. This paper was presented in part at the 37th Design Automation Conference, Los Angeles, CA, June 2000. This paper was recommended by Associate Editor R. Aitken.

I. Pomeranz is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA.

S. M. Reddy is with the Electrical and Computer Engineering Department, University of Iowa, Iowa City, IA 52242 USA.

Publisher Item Identifier S 0278-0070(01)03541-2.