Three-dimensional Place and Route for FPGAs

Cristinel Ababei, *Student Member*, Hushrav Mogal, and Kia Bazargan, *Member, IEEE*

***Abstract*—We present timing-driven partitioning and simulated annealing based placement algorithms together with a detailed routing tool for 3D FPGA integration. The circuit is first divided into layers with limited number of inter-layer vias, and then placed on individual layers, while minimizing the delay of critical paths. We use our tool as a platform to explore the potential benefits in terms of delay and wire-length that 3D technologies can offer for FPGA fabrics. Experimental results show on average a total decrease of 25% in wire-length and 35% in delay, can be achieved over traditional 2D chips, when 10 layers are used in 3D integration.**

***Index Terms*—Field programmable gate arrays, three-dimensional circuits, routing, timing-driven placement.**

I. Introduction

SMALLER feature size and increasing transistor counts allow the implementation of more complex and larger designs. However, a number of new design problems emerge and old problems become more difficult to solve. For example, global wires dominate the delay and power budgets of circuits, and signal integrity, IR-drops, process variations, and high temperature gradients pose new difficult design problems. Furthermore, shrinking time-to-market windows and ever-increasing mask costs have reduced profit margins alarmingly. In response to these mounting problems of integrated circuit technology, various research groups have shown renewed interest in 3D IC integration, and a number of successful projects have shown the viability of the technology [15]-[21]. 3D integration can significantly reduce wire-lengths (and hence circuit delay), and boost yield. Furthermore, there has been a trend towards employing IP-based design and structured gate arrays (*e.g.*, FPGA blocks) to partially solve complex signal integrity and noise issues as well as time to market constraints. 3D integration can particularly be useful for FPGA fabrics. It can address problems pertaining to routing congestion, limited I/O connections, low resource utilization and long wire delays [1], [3].

For the standard cell technology, [13] recently proposed a placement and global routing tool as well as a 3D layout editor. The placement algorithm is based on recursive min-cut partitioning of the circuit represented as a hypergraph and follows the same idea as in the Capo placer [23]. Interlayer via minimization is sought by using min-cut partitioning for layer assignment and wire-length (WL) minimization is done by considering the aspect ratio during partitioning. The user can select either hMetis [25] or PaToH [26] as the partitioning algorithm. Their global routing algorithm is a concurrent approach based on the idea in [24]. It was shown that 28% (51%) wire-length improvement could be obtained with two (five) layers, compared to [31] (the improvement is only 7% (17%) when inter-layer via minimization is the main objective). Wire-length reduction of up to 74% was reported in [33]. Deng and Maly showed – using a placement algorithm based on Capo – that the total wire-length can be reduced by 16% compared to flat placement, when two-layer integration is used [27]. It is important to note that current technologies allow for chemical mechanical polishing (CMP) substrate thinning down to about 5-10μm, hence allowing for multiple thin active device layers and interconnect levels be stacked on top of each other, resulting in short inter-layer vias with small aspect ratios [15].

Even though the idea of 3D integrated circuits is not new, recent technological advances have made it a viable alternative. However, there is a lack of efficient 3D CAD tools that can exploit the potential gains that 3D integration can offer. Furthermore, a number of important issues – such as heat dissipation, thermal stress [32], and physical design considerations – remain to be addressed for some 3D architectures.

There has been some previous work proposing 3D FPGA architectures. Borrowing ideas from multi-chip module (MCM) techniques, Alexander *et al.* proposed to build a 3D FPGA by stacking together a number of 2D FPGA bare dies [1]. Electrical contacts between different dies would be made using solder bumps or vias passing through the die. The number of solder bumps that can fit on a die determines the width and separation of vertical channels between FPGA layers. Reference [2] proposed using optical interconnects to construct a multi-layer FPGA. A straightforward extension of a 2D architecture [6] is found in the Rothko 3D architecture, which has routing-and-logic blocks (RLBs) placed on multiple layers [5]. Fine-grained interlayer connections were added outside each RLB, providing connections between cells above and below, using a specially designed technique [8], [9]. An improved version of the Rothko architecture – which advocates putting the routing in one layer and the logic on another for more efficient layer utilization – appears in [7]. It was shown that the percentage of routed connections increases

Manuscript received June 12, 2004, revised November 28, 2004. This work was supported in part by the Office of the Vice President for Research and Dean of the Graduate School of the University of Minnesota, under grant number 1546-522-5980.

The authors are with the Electrical and Computer Engineering Department, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: {ababei,mhush,kia}@ece.umn.edu).

with an increase in the flexibility of switch boxes. Also, computational density is higher compared to a 2D architecture. Universal switch boxes for 3D FPGA design were analyzed in [34]. It is important to point out that all of these works assume that the inter-layer connectivity is provided by vertical wire segments that connect each layer to its adjacent layers only.

There has also been previous work on CAD tools for 3D FPGA integration. Alexander *et al.* proposed 3D placement and routing algorithms [3] for their architecture in [1]. Their placement algorithm is partitioning-based followed by a simulated annealing based refinement for total interconnect length minimization. They reported savings of up to 23% and 14% in total interconnect length at the placement and routing level respectively. An improved version of the placement algorithm appears as Spiffy, which performs placement and global routing simultaneously [3]. In the experimental methodology presented in [7], placement was performed with VPR [10] and routing was performed with a custom routing tool [12].

Our goal is to present an efficient placement and detailed routing tool for 3D FPGAs. Unlike previous works on 3D FPGA architecture and CAD tools, we investigate the effect of 3D integration on delay, in addition to wire-length because wire-length alone cannot be relied on as a metric for 3D integration benefits. Apart from the commonly used single-segment architecture, we also study multi-segment architectures in the third dimension. Our placement algorithm is partitioning-based, and hence scalable with the design size. A circuit is first partitioned for min-cut minimization into a number of sub-circuits equal to the number of layers for the 3D integration. Then, timing-driven partitioning-based placement is performed on every layer starting with the top layer and proceeding towards the bottom layer. The allowable bounding box for nets on a particular layer is decided by the layers above it, in order to minimize the 3D bounding-boxes of the most critical nets. Constraints for any given layer are set by the placement on layers above. The routing algorithm was imported and adapted for the 3D architecture from the leading academic placement and routing tool for 2D architectures, VPR [10]. The main contribution of our work is as follows.

1) **TPR:** We developed a partitioning-based placement and maze routing toolset called TPR (Three-dimensional Place and Route). Its purpose is to serve the research community in predicting and exploring potential gains that the 3D technologies for FPGAs have to offer (similar to the role VPR played in the development of FPGA physical design algorithms). It shall be used as a platform, which can be used for further development and implementation of new ideas in placement and routing for 3D FPGAs.
2) **SA-TPR:** In addition to the partitioning-based 3D placement tool, we have also developed a Simulated Annealing based version of TPR (called SA-TPR) to provide speed / quality tradeoffs.
3) **Architectural analysis**: We analyze potential benefits that 3D integration can provide for FPGAs. More specifically, we place and detailed route circuits onto 3D FPGA architectures and study the variation in circuit delay and total wire-length compared to their 2D counterparts, under different 3D architectural assumptions. The results of this study and similar studies in future can guide researchers in designing high performance 3D FPGA fabric architectures.

II. Overview Of TPR

The philosophy of our tool closely follows that of its 2D counterpart, VPR [10]. The flow of the TPR placement and routing CAD tool is shown in Fig. 1. The design flow starts with a technology-mapped netlist in .blif format, which can be generated using SIS [28]. The .blif netlist is converted to a netlist composed of more complex logic blocks with T-VPack [11]. The .net netlist as well as the architecture description file are inputs to the placement algorithm. The placement algorithm first partitions the circuit into a number of balanced partitions equal to the number of layers for 3D integration. The goal of this first min-cut partitioning is to minimize the connections between layers, which translates into minimum number of vertical (*i.e.*, inter-layer) wires. The reason is that in 3D technologies, the architecture is not isotropic (*i.e.*, due to their higher pitch, vertical vias are not as dense as the 2D channels) and thus the placement and routing tools must judiciously use scarce vertical routing resources. After dividing the netlist into layers, TPR continues with the placement of each layer in a top-down fashion.

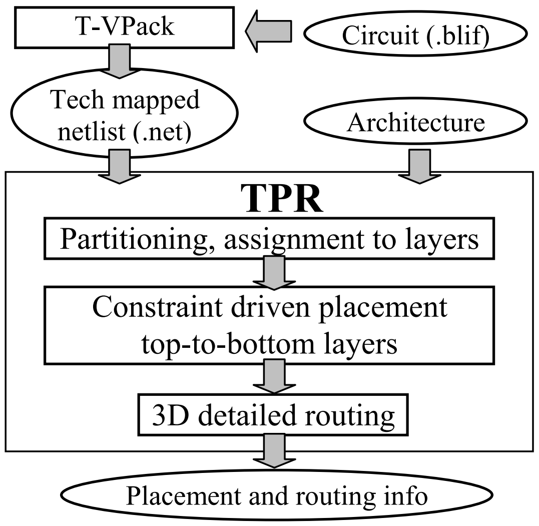

T-VPack
Circuit (.blif)
Tech mapped netlist (.net)
Architecture
TPR
Partitioning, assignment to layers
Constraint driven placement top-to-bottom layers
3D detailed routing
Placement and routing info

Fig. 1. Flow diagram of TPR: 3D placement and routing tool is similar to VPR's flow diagram.

The top layer is placed by unconstrained recursive partitioning. The rest of the layers are then placed in turn by recursive partitioning, but constrained to reduce the delay on timing-critical nets: the terminals of the most critical nets, which span more than one layer, are placed on restricted placement regions. The restricted placement regions are defined by the projection of the bounding-boxes defined by the terminals already placed in the layers above onto the current layer. Hence, the 3D bounding-box of the critical nets is minimized, similar to the 2D terminal alignment proposed in [29]. Finally, global and detailed routing is performed using the adapted 3D version of the VPR routing algorithm.

III. Placement Algorithm

The simplified pseudo-code of the partitioning-based placement algorithm is shown in Fig. 2. In the first stage the circuit netlist undergoes the initial min-cut partitioning and assignment to layers. The second main phase is the actual placement of cells on all layers. In this section we present in more detail the two main steps of our placement algorithm.

```
Input:
       Tech mapped netlist .net G(V,E)
       Architecture description file
Algorithm:
1.  Initial min-cut partitioning into layers
   for via minimization
2.  For all layers i = 0 to L-1 from top to bottom
3.      Do partitioning based placement of layer i
4.          Update timing slacks
5.          Re-enumerate critical paths
6.      Greedy overlap removal
7.      Constraint generation for layers below
8.  Write .p placement output file
```

Fig. 2. Pseudo-code of TPR placement algorithm.

A. Initial Partitioning and Assignment to Layers

The initial partitioning-into-layers step is performed using the min-cut hMetis partitioning algorithm [25] and is further illustrated in Fig. 3. This is motivated by the limitations imposed by current technologies, which require us to minimize the usage of vertical connections (it was also observed in [14] that optimizing inter-layer interconnect is of key importance for 3D integration technologies).

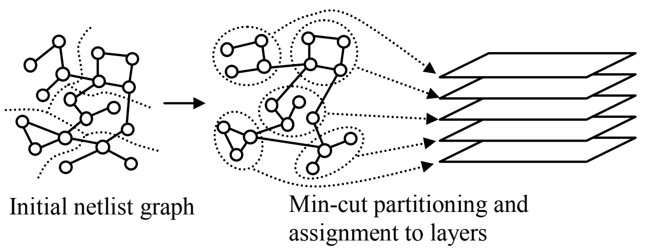


Fig. 3. Illustration of initial partitioning and assignment to layers.

After the initial partitioning into layers we assign blocks (*i.e.*, partitions) to layers using a linear placement technique. The goal of this step is to minimize both the total vertical wire-length and maximum cut between any two adjacent layers. For example, in Fig. 4, we would like to assign the five blocks (as a result of the initial 5-way min-cut partitioning) to layers of the 3D architecture as in the case labeled "Good" rather than in the case labeled "Bad". That is because the good layer assignment minimizes both the total wire-length and maximum cut between adjacent layers.

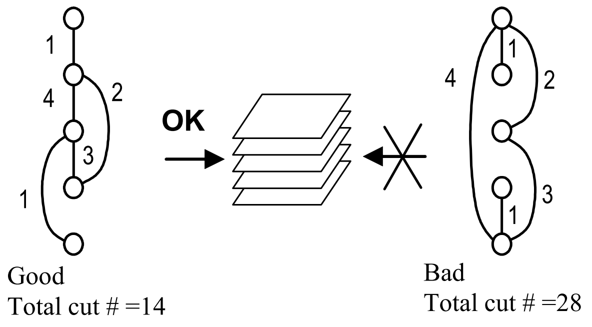
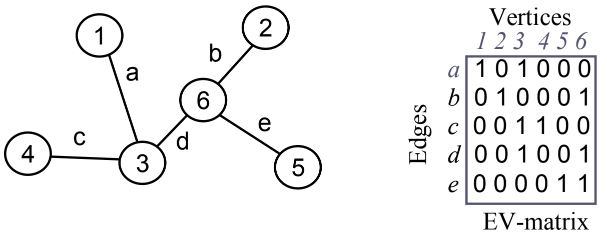


Fig. 4. Illustration of good and bad initial linear placement of partitions into layers.

The objective of minimizing the total vertical wire-length and maximum cut between any two adjacent layers is equivalent to minimizing the bandwidth of the EV-matrix associated to the layer-blocks graph which resulted from the above partitioning [38]. Edges of this graph have weights given by the number of edges of the initial netlist graph, which span different layer-blocks. The EV-matrix is an $m \times n$ matrix where m – the number of rows – is the number of edges in the graph and n – the number of columns – is the number of nodes. An element $a(i, j)=1$ in the matrix is non-zero if the *j-th* node is a terminal of the *i-th* net. If a node is not a terminal for a net, the corresponding *EV-matrix* element is zero. The bandwidth of a matrix is defined as the maximum distance between first and last non-zero entries, among all rows. An example of such a layer-blocks graph and its associated EV-matrix are presented in Fig. 5. In this example the bandwidth equals four due to second row, which has its first non-zero entry in the second column and its last nonzero entry in the sixth column.



$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline a & 1 & 0 & 1 & 0 & 0 & 0 \\ b & 0 & 1 & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 1 & 1 & 0 & 0 \\ d & 0 & 0 & 1 & 0 & 0 & 1 \\ e & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Fig. 5. Layer-blocks graph and its associated EV-matrix.

The bandwidth minimization problem is known to be NP-complete [38] and a solution for the layer assignment problem may not be optimal in terms of both objectives of wire-length and maximum cut between adjacent layers. Therefore, for this step, we use an efficient heuristic [37], which is able to find solutions with very good trade-off between wire-length and maximum cut. This technique is briefly described in what follows using the graph example shown in Fig. 5. We first build the EV-matrix, and then transform it into an as-close-as-possible to a band-form matrix. This problem is denoted as *B(EV-matrix)-min* problem (*i.e.*, minimization of the bandwidth of the EV-matrix problem). The procedure to solve this problem uses row and column flips in order to sort rows

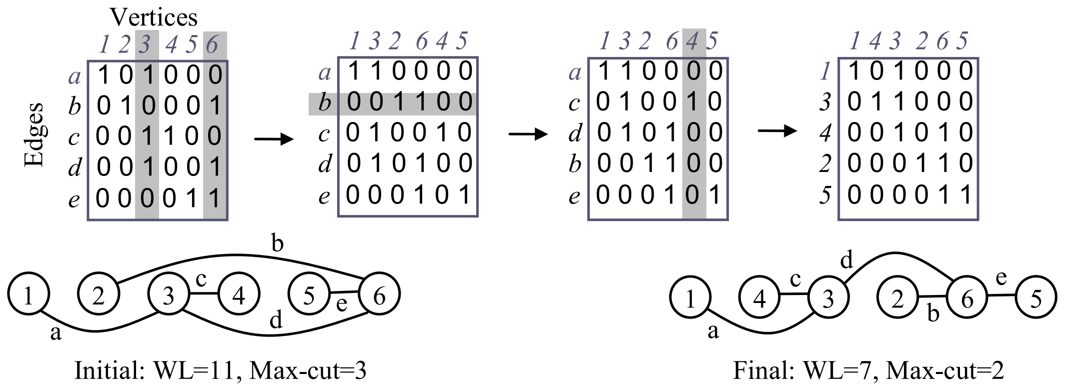


Fig. 6. Illustration of the *EV-matrix* band-width minimization for both wire-length and maximum cut between adjacent layers.

and columns such that non-zero elements are moved towards the main diagonal. For example, for the matrix shown in Fig. 6, in order to "drift" non-zero elements from the upper half towards the main diagonal (i.e., from right to left) column flips are performed between columns 2 and 3 and then column 6 is moved between columns 3 and 4. This technique is performed to move non-zero elements from right to left and from top to bottom (for the upper right half) and from left to right and from bottom to top (for the lower left half). When the above procedure finishes, the example from Fig. 5 becomes as described in Fig. 6. A more detailed description can be found in [37]. The goal of getting the matrix to a band-form (which translates into a best linear ordering) serves two objectives:

--Cutsize minimization – by having all 1's in the matrix clustered along the main diagonal, the cutsize (the number of nets cut by a vertical cut applied between any two consecutive nodes in the linear arrangement) is minimized everywhere in the linear arrangement

-- WL minimization – by minimizing the bandwidth (maximum distance spanned by any of the nets) of the *EV-matrix*, the total wire-length of all nets is minimized.

B. Placement Method

After the initial layer assignment, placement is performed on each layer starting with the top layer (layer *0*) and continuing downwards till the last layer (layer *L-1*). The placement of every layer is based on edge-weighted quad-partitioning using the hMetis partitioning algorithm, and is similar to the approach in [29], which has the same quality as VPR but at 3-4 times shorter run times. Edge weights are usually computed inversely proportional to the timing slack of the corresponding nets. However, we also selectively bias weights of the most critical nets. The set of critical nets is comprised of edges on the current *k*-most critical paths. The placement algorithm has an integrated static timing analysis engine as well as a path enumeration algorithm [30]. The delay of the circuit (and therefore slacks) and the set of the most critical paths are periodically updated based on the delay assigned to all current cut nets by the partitioning engine. This ensures accurate estimation of the circuit delay as the placement algorithm progresses. The rate of delay update and critical paths re-enumeration is dictated by the runtime / estimation accuracy trade-off.

The recursive partitioning of a given layer stops when each placement region has less than four blocks. Complete overlap removal is done using a greedy heuristic which moves non-critical blocks (*i.e.*, not on any critical paths) to the closest available empty location. When the placement of a layer is finished, we propagate placement constraints for the most critical nets. For example, assuming that layer k_0 is the first layer (from the top) in which some terminals of a critical net are placed, the bounding box of the net is cumulatively projected to lower layers k_0+1 through *L-1* as a placement constraint for the rest of the terminals of the net in these layers. In special cases such as when two terminals are placed very closely on layer k_0 and the projected bounding box on layers below is not large enough to accommodate all the terminals, there may be more terminals constrained inside a region which would not have enough CLB's available to accommodate all the constrained terminals. If this situation is relatively relaxed (meaning that there are not too many such terminals or the constraint area is not too small) then they will translate in overlaps which will be removed at the end of the recursion. If the situation is such that more cells cluster together by being constrained in a relatively small area, then they will be eventually split during the partitioning process so that the unbalance factor will be satisfied but they will still be in proximity of each other. In both the above cases projected terminals will end up being close to each other.

In layers that have net bounding box constraints, terminals that have placement restrictions are fixed in appropriate partitions before a call to the hMetis partitioning engine. This technique explicitly minimizes the 3D bounding-boxes of critical nets, which leads to minimization of the total wire-length and circuit delay. Steps 3 to 8 of the algorithm shown in Fig. 2 are performed for all layers, and when the last layer is finished the circuit is completely placed.

IV. SA-TPR: Simulated Annealing Based 3D Placement

In order to analyze the impact of the placement approach (such as partitioning-driven presented in the previous section) on the comparison between 2D and 3D cases we also extended the simulated annealing based placement algorithm of VPR [10] to 3D. Our simulated annealing engine based placement algorithm is fully 3D integrated and is based on the model that VPR uses for its 2D placement. The user specifies the number of layers as well as the annealing schedule to use. As in VPR, our SA engine can place circuits with constraints of both wire-length and timing.

Wire-length of a net is calculated as the weighted sum of its projected 2D bounding box and its vertical span. The weight on the vertical span is set to a high value to discourage usage of scarce vertical vias. The cost of a net *e* is described by the equation below.

$$Cost_{3D}(e) = q \cdot Cost_{2D}(e) + \alpha \cdot Span_z + \beta \cdot Num\_layers(e) \tag{1}$$

where *q* is a correction factor to 2D bounding box computation, which accounts for nets that have more than 3 terminals (the original VPR code uses this factor); $Cost_{2D}$ is the half-perimeter bounding box of the projection of all the terminals of the net; $Span_z$ is the vertical span of the net, and *Num_layers* is the number of layers on which terminals of net *e* are placed. Factors *α* and *β* are used to constrain the maximum length of vertical segments as well as the vertical channel density. To see the importance of using these factors, let us consider the two placements in Fig. 7.

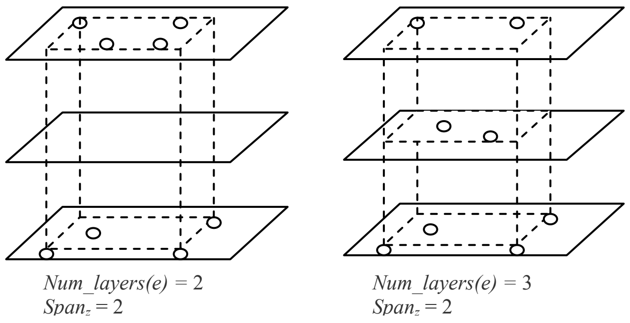


Fig. 7. Two possible placements of the same net, showing different number of layers occupied.

The two placement scenarios would be treated identically if we did not separately consider both the vertical span of a net, and the number of layers in which its terminals are placed. Each of these cost components are scaled by appropriate scaling factors: α, which discourages placing the terminals of a net far apart in the *z* dimension (otherwise the routing of the net would require increased length vertical vias), and β, which restricts the number of vertical vias (vertical channel density is lower than the horizontal channel density and *β* reflects that ratio). The placement on the left in Fig. 7 is preferred to the one on the right, as it could potentially use only one vertical segment of length two to connect the terminals in different layers. The placement on the right is likely to use more vertical routing resources.

Computation of the timing cost of a net essentially follows the approach of VPR (timing criticalities for nets are computed based on slacks). The modification for the 3D case is as follows. First, the source-sink connection (whose delay we compute) is projected onto 2D and its separation Δ*x* and Δ*y* in the two dimensions is calculated. Delay lookup tables similar to VPR are used to calculate these values wherein unlimited routing resources are assumed. To accommodate a 3D structure, the separation of the connection in the third dimension is found and its delay is looked up using only one dimension of the delay tables (*i.e.*, a net that spans a distance of Δ*z* in the vertical dimension, has the same delay as a 2D net with (Δ*z*,0) bounding box). Finally, the annealing engine constrains movement in *x* and *y* directions similar to VPR (initially movement is allowed across the entire dimension of the chip and then gradually it is shrunk to neighboring CLB's). Movement in the third dimension is unrestricted in order to fully explore the vertical dimension.

V. Routing Algorithm

A. Description of the Routing Algorithm

The 3D FPGA architecture – described in the architecture file – is represented as a routing resource graph. Each node of the routing resource graph represents a wire (horizontal tracks in the *x* and *y* channels of all layers and vertical vias in the *z* channels) or a logic block (*i.e.*, CLB) input or output pin. A directed edge represents a unidirectional switch (such as a tri-state buffer). A pair of directed edges represents a bi-directional switch (such as a pass transistor). An example of a routing resource graph construction is shown in Fig. 8.

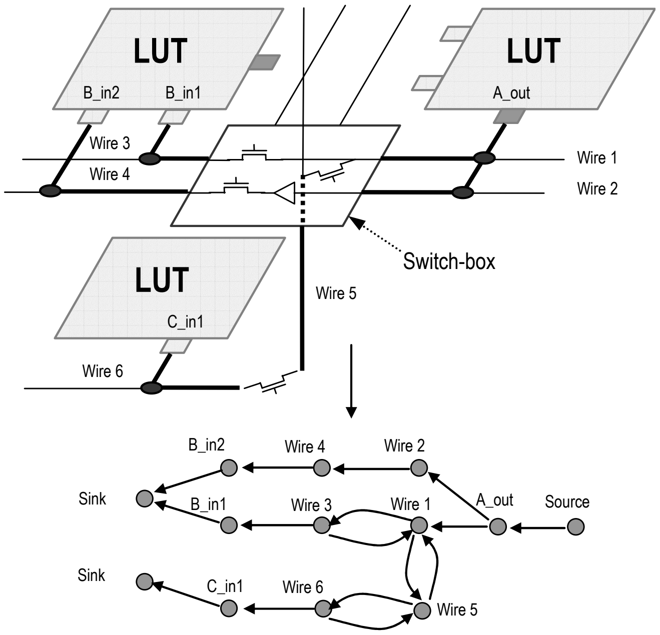


Fig. 8. Illustration of the routing graph construction.

TPR 3D detailed router is based on the Pathfinder negotiated congestion algorithm [35]. The routing is a rip-up and re-route iterative process which routes every net by the shortest path using a breadth-first-search technique. The cost of overused routing resources is gradually increased so that the algorithm forces nets with alternative routes to avoid overused routing resources, leaving behind only the net, which most needs a given resource. We add extra penalties to bends of a route created by a horizontal track and a vertical via as well as to vias themselves in order to discourage the routing engine to prefer vias and therefore to avoid a net placed totally in one layer to be routed using tracks in different layers. This will make, for example, the routing engine find the routing to the left shown in Fig. 9 rather than the routing solution shown to the right.

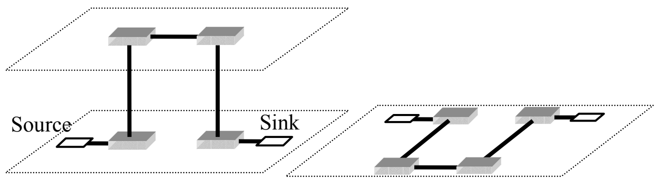


Fig. 9. Illustration of two routings for a two terminal net.

TPR router can find very small horizontal and vertical channel widths (CWs) for which the circuit is fully routable. Vertical channel width starts with a value specified by the architecture file and is incremented every time when the routing fails for a pre-determined number of different values for the horizontal channel width.

B. Computation of Total Foot print Area

In order to be able to compute the foot-print chip area after detailed routing is completed for 3D architectures (simulations results reported in Section 6) we use a new formula, which is the adaptation to 3D of the computations performed in 2D. Footprint area is calculated as total area divided by number of layers. Generally, the overall chip area is the summation of the area taken by logic (CLBs) and routing resources (switch boxes, connection boxes, track segments).

$$Area = Area_{Logic} + Area_{Routing} \tag{2}$$

The above formula reduces to the following simplified equation (we omitted details for brevity), for 2D case (see [36], pages 207-217):

$$Area_{2D} = [Area_{LUT} + C_1 \cdot HCW + C_2 \cdot Area_{mux}(HCW)] \cdot W^2 \tag{3}$$

where:

$$Area_{mux}(HCW) = 6 \cdot \lceil \log_2(HCW) \rceil + 2HCW - 2 \tag{4}$$

represents the area of all minimum-width transistors required by a multiplexer with *HCW* inputs [36]. Such multiplexers are used to connect tracks in the routing channels to the input pins of CLBs and because of that we need multiplexers with a number of inputs equal to the horizontal channel with, *HCW*. C_1, C_2 are functions of track-buffer area, number of input pins within an FPGA tile, number of input pins within a tile which share a common buffer, output pin buffer area, number of pass transistors corresponding to an output pin of a CLB, buffer (inside switch boxes) area, and F_s (F_s=3 for 2D) [36]. $Area_{LUT}$ is the total area of a CLB, which may contain one or more LUTs. *W* is the width of the FPGA chip measured as a multiple of inter-CLB distances (assumed to be equal to the height of the FPGA chip).

The overall chip area computation, extended to the 3D case, is given by the following expression:

$$\begin{aligned} Area_{3D} &= [Area_{LUT} + C_1^{'} \cdot HCW + C_2 \cdot Area_{mux}(HCW)] \cdot L \cdot W^2 \\ &+ [C_3 \cdot (HCW - VCW) + C_4 \cdot VCW] \cdot L \cdot (W-1)^2 \end{aligned} \tag{5}$$

$C_1^{'}, C_2, C_3, C_4$ are functions of track-buffer area for input or output pins of a CLB, number of input pins within an FPGA tile, number of input pins within a tile which share a common buffer, area of all pass transistors corresponding to an output pin of a CLB, buffer (inside switch boxes) area, and F_s (F_s=3 for 2D and F_s=6 for 3D). *HCW*, *VCW* are horizontal and vertical channel widths. *L* is number of layers for 3D architectures.

VI. Simulation Results

A. 3D Architectures

Our goal is to study the variation of the circuit delay and the total wire-length as a function of the number of layers when the delay of an inter-layer wire (*i.e.*, vertical via) has different values. We considered two different architectures: *Sing-Seg* and *Multi-Seg.* In both architectures, each horizontal layer has a routing architecture that resembles a simplified version of the Xilinx Virtex II architecture (they have wire segments of lengths 1, 2, and 6, as well as long lines in each layer). However, *Sing-Seg* has vertical (inter-layer) vias of length one only, while *Multi-Seg* has vertical vias that span 1, 2, and all planes. Length one vertical segment is assumed to have the same delay and wire-length as 2D unit-length segments. This is a reasonable assumption, because 3D fabrication methods such as [15] can create inter-layer vias that are merely 5-10μm long. In such vertical segments, the switch delay dominates the delay of the segment, which is similar to the 2D case. Our architectures have one 4-input LUT per CLB as this is a common assumption made in most previous works. We do not include a description of the LUT because it is essentially the same as for 2D cases and there are many previous works describing them such as [36]. Vertically, our architectures are unique by being vertical in the third dimension, but they are also along the same lines as traditional FPGAs: either containing vias of length one or vias spanning more layers. The switch-box in our architecture is different from the 2D case by the fact that some tracks are connecting to the vertical vias. The switch-box will be described in more detail later on (see Fig. 12). We would like to mention that these architectures are very complex and our implementation is very flexible supporting any combination of segment and via lengths as well as any known type of switch-box. We randomly placed IO terminals on the periphery of every layer after the partitioning and assignment to layers is performed. However, the user can provide locations for IO terminals.

B. Delay Results of TPR and SA-TPR Algorithms

We cannot compare our results to any of the previous works for a couple of reasons. First, our place and route tool is the first to report comprehensive results on wire-length and circuit delay as well as on all other metrics such as chip area, horizontal and vertical channel widths, and run-times on all twenty circuit benchmarks of the VPR package. We cannot compare to the only previous existing results reported in [3] or [4] because the authors of [3] used only six circuit benchmarks (unavailable to us) different from those we use (except *Apex2*). Moreover, the authors of [3] report only wire-length and minimum channel width results obtained for a very simple architecture, which only contains horizontal and vertical routing segments of length one. This is in contrast with our architectures, which have mixed routing resources both horizontally and vertically. The authors of [3] use only four layers whereas we report results for a range between two and ten layers. The authors of [4] report results for wire-length only for a number of layers between two and four, using unknown circuit benchmarks and architecture, most likely similar to the one used in [3] and different from our more complex architectures. Since we are providing the source code and benchmarks that we use in this paper on the World Wide Web (see the Conclusion Section), other researchers can easily compare their results to ours.

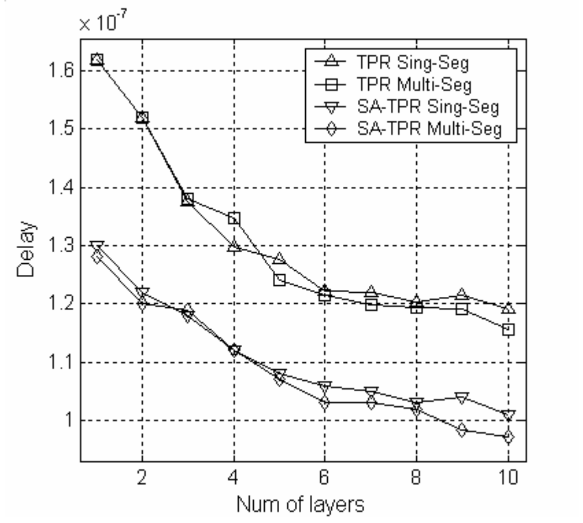


Fig. 10. Variation of delay as reported after detailed routing.

We placed and detailed routed all circuits (see Table I) on a number of layers varied between one (the 2D case) and ten. We recorded the average circuit delay and the average total wire-length of four different runs for each circuit. The comparison between the variation of the average delay values obtained using partitioning-based (TPR) and SA-based (SA-TPR) placement algorithms is illustrated in Fig. 10. We observe that delay decreases by about 30% for both architectures using either placement algorithm compared to the 2D case, although architecture *Multi-Seg* shows slightly better delays, as the number of layers increases beyond six. However, the difference is not large, mainly because the routing algorithm considers fully buffered routing resources, which leads to comparable delay values for both architectures. Delay achieved using the SA-based placement algorithm is smaller compared to the delay achieved using the partitioning-based placement algorithm, which is not surprising, because annealing takes longer runtimes.

We noticed that the amount of delay decrease compared to the 2D case for different circuits and same number of layers can vary. For example, delay decreases by 27% for *Pdc* benchmark but only 18% for *Spla* benchmark when nine layers are used. We suspect this is due to the internal structure (such as higher connectivity) of *Pdc* as opposed to *Spla*, which leads to a larger fraction of nets spanning different number of layers. During our experiments we also noticed that benchmarks with large number of terminals (relative to the number of cells), such as *Des* (see Table I), tend to lead to more delay decrease compared to 2D case. This can be explained by the fact that in the 2D case, the chip size necessary to accommodate all terminals is bigger than if the circuit had less terminals (i.e., final chip will have more "white-space") and therefore delays of nets involving input or output terminals will have larger routing delays in the 2D case.

C. Wire-length Results of TPR and SA-TPR Algorithms

Wire-length results of TPR on *Multi-Seg* architecture are

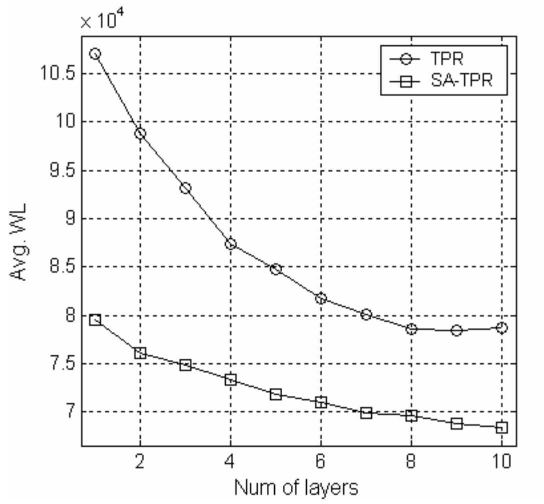


Fig. 11. Variation of average wire-length after detailed routing as a function of number of layers, using both TPR and SA-TPR placement algorithms.

shown in Fig. 11.

Results for architecture *Sing-Seg* and using the SA-based placement algorithm are similar. We observe that wire-length after detailed routing decreases by 25% on average as the number of layers increases. If the length of the inter-layer via increases, then the total wire-length decrease will be less. That is mainly because the fraction of the vertical wire-length relative to the total wire-length will become significant and also the average net delay will increase due to bending (i.e., switches) of nets spanning more layers. Wire-length achieved using the SA-based placement algorithm is smaller compared to the wire-length achieved using the partitioning-based placement algorithm.

We note that the decrease in total wire-length can have favorable impact on the routing congestion (hence channel width), as well as power dissipation (especially due to the fact that most of the power dissipated in FPGAs is due to interconnects, which account for more than 80% of the total area) as predicted in [22].

The decrease in total WL is directly related to a decrease of the average net wire-length, which in turn relates to the overall usage of routing resources and therefore to the circuit delay. Variations of the average net wire-length, and other metrics of interest are shown in Table II and Table III. Routing area counts the exact number of pass transistor and SRAM cells that control them, as suggested in [36]. We observe that overall area (*i.e.*, chip foot-print area multiplied by the number of layers) slightly increases for a small number of layers. This increase is due to the higher connectivity inside of a switch box (*i.e.*, a track entering a 3D switch box will have to connect to 5 correspondents as opposed to only 3 in the 2D case; see Fig. 12). However, routing area decreases as the number of layers increases for *Multi-Seg* architecture, as a direct consequence of a decrease of the horizontal channel width (HCW) required for successful routing. Routing area increase is overall less than 10% when SA-based placement algorithm is used. Except for cases where only few layers are used, CW decreases significantly. The reason is that the number of vertical connections is minimized. In other words, the partitioning algorithm is able to find the clustering structure of the circuit and practically divide the initial netlist into a number of smaller circuits with similar internal structure (in terms of connectivity or Rent's parameter) to the initial one. As a consequence, the horizontal channel width for each layer will be in the same range as when the initial netlist is placed in 2D.

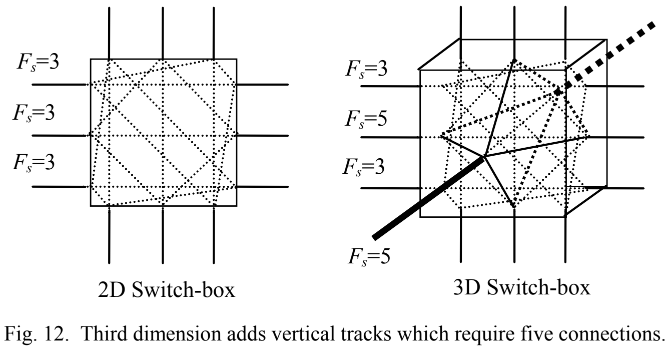


Fig. 12. Third dimension adds vertical tracks which require five connections.

We observe that, overall, run-times of SA-based placement are about twice the run-times of detailed routing (see Table 1) and about an order of magnitude longer than run-times of partitioning-based placement. Therefore, partitioning-based placement can be used for efficient solution space (especially for big circuits) and different architectural assumptions exploration. Also, the vertical channel widths reported in Table I are 1/5 of the horizontal channel widths, which demonstrates that our layer partitioning and linear placement as well as the routing algorithm are very well tuned to minimize the use of vertical tracks. Another advantage of using fewer vertical tracks greatly reduces the required area for switchboxes.[1]

D. *Experiments Using Mixed Partitioning and SA Based Placement Algorithm*

We also implemented a mixed partitioning and simulated-annealing placement algorithm. The reason for that is that the initial partitioning and assignment to layers does a very good job at minimizing the amount of vertical vias. This technique combined with SA-based placement on each individual layer (under the restriction of not moving cells between layers) leads to high quality placements with minimum vertical connectivity, which is desirable due to the limitations imposed by current technologies, which require us to minimize the usage of vertical connections. As we can see in Table II and Table III, this strategy indeed leads to a decrease in wire-length whereas delay is virtually the same compared to full SA placement, which results into slightly smaller horizontal channel width. These results show that the quality of our layer partitioning and linear placement is very good. These experiments also demonstrate that a good initial solution for the annealing based algorithm can be very important and can lead to better results as opposed to a randomized initial placement.

VII. Conclusion

Benefits which 3D FPGA integration can offer were analyzed using a new placement and detailed routing tool. Placement can be done using either partitioning-based or simulated annealing based approach. Simulation experiments, after detailed routing, showed potential total decrease of 25% for wire-length and 35% for delay using either the partitioning-based algorithm or the SA-based algorithm. We observed that the Multi-Seg architecture shows slightly better delay characteristics compared to the Sing-Seg architecture.

Source code and documentation of the implementation of the algorithms presented in this paper are available for download at: http://www.ece.umn.edu/users/kia.

References

[1] A. J. Alexander, J. P. Cohoon, Jared L. Colflesh, J. Karro, and G. Robins, "Three-Dimensional Field-Programmable Gate Arrays," *Proc. Intl. ASIC Conf.*, 1995, pp. 253-256.

[2] J. Depreitere, H. Neefs, H. V. Marck, J. V. Campenhout, D. B. R. Baets, H. Thienpont, and I. Veretennicoff, "An Optoelectronic 3-D Field Programmable Gate Array," *Proc. Intl. Workshop on Field-Programmable Logic and Applications*, 1994.

[3] A. J. Alexander, J. P. Cohoon, Jared L. Colflesh, J. Karro, E. L. Peters, and G. Robins, "Placement and Routing for Three-Dimensional

[1] A vertical/horizontal intersection with F_s=3 requires 6 (4 choose 2) switches, while an intersection with F_s=5 requires 15 (6 choose 2) switches. Each switch has a pass transistor, an SRAM cell, and possibly a buffer.

FPGAs," *Fourth Canadian Workshop on Field-Programmable Devices*, pp. 11-18, 1996.

[4] J. Karro and J. P. Cohoon, "A spiffy tool for the simultaneous placement and global routing for three-dimensional field-programmable gate arrays," *Ninth Great Lakes Symposium on VLSI*, 1999, pp. 226-227.

[5] M. Leeser, W. Meleis, M. Vai, S. Chiricescu, W. Xu, and P. Zavracky, "Rothko: A Three-Dimensional FPGA," *IEEE Design Test Computers*, 1998, pp. 16-23.

[6] G. Borriello, C. Ebeling, S. Hauck, and S. Burns, "The Triptych FPGA Architecture," *IEEE Trans. On VLSI Systems*, Vol. 3, No. 4, 1995, pp. 491-501.

[7] S. Chiricescu, M. Leeser, and M. M. Vai, "Design and Analysis of a Dynamically Reconfigurable Three-Dimensional FPGA," *IEEE Trans. VLSI Systems*, Vol. 9, No. 1, Feb. 2001, pp. 186-196.

[8] P. Zavracky, M. Zavracky, D. Vu, and B. Dingle, "Three-Dimensional Processor Using Transferred Thin Film Circuits," *U.S. Patent Application*, 08-531-177, Jan. 1977.

[9] P. Sailer *et al.*, "Three-Dimensional Circuits Using Transferred Films," *IEEE Circuits and Devices*, Vol. 13, No. 6, Nov. 1997, pp. 27-30.

[10] V. Betz and J. Rose, "VPR: A New Packing Placement and Routing Tool for FPGA Research," *Field-Programmable Logic App.*, 1997, pp. 213-222.

[11] A. Marquardt, V. Betz, J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," *FPGA*, 1999, pp. 37-46.

[12] S. Chiricescu, "Parametric Analysis of a Dynamically Reconfigurable Three-Dimensional FPGA," *Ph.D. Dissertation*, Northeastern Univ., Boston, MA, Dec. 1999.

[13] S. Das, A. Chandrakasan, and R. Reif, "Design Tools for 3-D Integrated Circuits," *Proc. ACM/IEEE ASP-DAC*, 2003.

[14] S. Das, A. Chandrakasan, and R. Reif, "Three-Dimensional Integrated Circuits: Performance Design Methodology and CAD Tools," *Proc. ACM/IEEE ISVLSI,* 2003.

[15] R. Reif, A. Fan, K. - N. Chen, and S. Das, "Fabrication Technologies for Three-Dimensional Integrated Circuits," *Proc. International Symposium on Quality Electronic Design (ISQD)*, 2002.

[16] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: A novel chip design for improving deep submicron interconnect performance and systems-on-chip integration," *Proceedings of the IEEE*, Vol. 89, May 2001, pp. 602-633.

[17] J.A. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S.J. Souri, K. Banerjee, K.C. Saraswat, A. Rahman, R. Reif and J.D. Meindl, "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proc. IEEE*, Vol. 89, Mar. 2001, pp. 305-324.

[18] "SoCs are 'dead' Intel manager declares," February 12, 2002. [Online]. Available: http://www.eet.com/semi/news/OEG20030212S0038.

[19] J. Burns, L. McIlrath, J. Hopwood, C. Keast, D. P. Vu, K.Warner, and P. Wyatt, "An soi-based three dimensional integrated circuit technology," *IEEE International SOI Conference*, 2000, pp. 20-21.

[20] K. W. Lee, T. Nakamura, T. Ono, Y. Yamada, , T. Mizukusa, H. Hashimoto, K. T. Park, H. Kurino, and M. Koyanagi, "Three-dimensional shared memory fabricated using wafer stacking technology," in *Technical Digest of the International Electron Devices Meeting*, 2000, pp. 165-168.

[21] K. W. Guarini, A. W. Topol, M. Leong, R. Yu, L. Shi, M. R. Newport, D. J. Frank, D. V. Singh, G. M. Cohen, S. V. Nitta, D. C. Boyd, P. A. O'Neil, S. L. Tempest, H. B. Pogpe, S. Purushotharnan, and W. E. Haensch, "Electrical integrity of state-of-the-art 0.13u m SOI CMOS devices and circuits transferred for three-dimensional (3D) integrated circuit (IC) fabrication," in *Technical Digest of the International Electron Devices Meeting*, 2002, pp. 943-945.

[22] A. Rahman, S. Das, A. Chandrakasan, and R. Reif, "Wiring Requirement and Three-Dimensional Integration of Field-Programmable Gate Arrays," *Proc. ACM/IEEE SLIP*, 2001.

[23] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?," *Proc. ACM/IEEE DAC*, pp. 477-482, 2000.

[24] M. Burnstein and R. Pelavin, "Hierarchical Wire Routing," *IEEE Trans. CAD*, Vol. 2, No. 4, 1983, pp. 223-234.

[25] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multi-level Hypergraph Partitioning: Applications in VLSI Design," *Proc. ACM/IEEE DAC*, 1997, pp. 526-529.

[26] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based Decomposition for Parallel Sparse-matrix vector Multiplication," *IEEE Trans. Parallel and Distributed Systems*, Vol. 10, No. 7, 1999, pp. 673-693.

[27] Y. Deng and W. P. Maly, "Interconnect Characteristics of 2.5-D System Integration Scheme," *Proc. ACM/IEEE ISPD*, 2001. , pp. 171-175

[28] E. M. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis," *Technical Report UCB/ERL M92/41*, University of California, Berkeley, 1992.

[29] P. Maidee, C. Ababei and K. Bazargan, "Fast Timing-driven Partitioning-based Placement for Island Style FPGAs," *Proc. ACM/IEEE DAC*, 2003, pp. 598-603.

[30] Y-C. Ju, R.A. Saleh, "Incremental Techniques for the Identification of Statically Sensitizable Critical Paths," *Proc. ACM/IEEE DAC*, 1991.

[31] P. H. Madden, "Reporting of Standard Cell Placement Results," *Proc. ACM/IEEE ISPD*, 2001, pp. 30-35.

[32] B. Goplen and S. Sapatnekar, "Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach," *Proc. ACM/IEEE ICCAD*, 2003, pp. 86-89.

[33] S. T. Obenaus and T. H. Szymanski, "Gravity: Fast Placement for 3-D VLSI," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, Vol. 8, No. 3, July 2003, pp. 298-315.

[34] G. – M. Wu, M. Shyu, and Y. – W. Chang, "Universal Switch Blocks for Three-Dimensional FPGA Design," *Proc. FPGA*, 1999, pp. 254-259.

[35] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and Routing Tools for the Trptych FPGA," *IEEE Trans. VLSI Systems*, Vol. 3, No. 4, Dec. 1995, pp. 472-483.

[36] V. Betz, J. Rose, and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs," Kluwer Academic Publishers, 1999.

[37] C. Ababei and K. Bazargan, "Non-contiguous Linear Placement for Reconfigurable Fabrics," *Proc. Reconfigurable Architectures Workshop (RAW)*, 2004.

[38] J. Díaz, J. Petit, M. Serna, "A survey of graph layout problems," *ACM Computing Surveys Journal*, 2002, pp. 313-356.

TABLE I
SIMULATED CIRCUITS: STATISTICS, VERTICAL CHANNEL WIDTH (VCW), AND RUN-TIME

Circuit	No. CLBs	No. IOs	VCW TPR	VCW SA-TPR	CPU (s) Par-based placement	CPU (s) SA-based placement	CPU (s) Detailed Routing
Ex5p	1064	71	6	5	7	85	77
Apex4	1262	28	6	5	8	105	76
Misex3	1397	28	6	5	9	55	84
Alu4	1522	22	5	5	16	145	61
Des	1591	501	5	5	11	227	69
Seq	1750	76	5	5	12	212	114
Apex2	1878	42	5	5	13	243	148
Spla	3690	62	5	6	37	912	532
Pdc	4575	56	7	7	60	1354	1039
Ex1010	4598	20	4	5	56	1238	273
Dsip	1370	426	5	5	28	154	34
Tseng	1407	174	5	5	8	70	14
Diffeq	1497	103	5	5	14	154	46
Bigkey	1707	426	5	5	22	209	48
S298	1931	10	5	5	23	208	53
Frisc	3556	136	5	5	56	881	227
Elliptic	3604	245	5	5	74	818	142
S38417	6406	135	5	5	133	2000	210
S38584.1	6447	342	5	5	230	2034	268
Clma	8383	144	5	5	199	892	950
				Sum	1016	11996	4465

TABLE II
AVERAGE OF DELAY, WIRE-LENGTH (WL), HORIZONTAL CHANNEL WITH (HCW), AND ROUTING AREA FOR SING-SEG ARCHITECTURE

Num layers	TPR Delay ($\times 10^{-7}$)	TPR WL	TPR Routing area	TPR HCW	SA-TPR Delay ($\times 10^{-7}$)	SA-TPR WL	SA-TPR Routing area	SA-TPR HCW	Combined Partitioning + SA Place Delay ($\times 10^{-7}$)	Combined WL	Combined Routing area	Combined HCW
1	1.62	107087.6	1	1	1.3	79553.76	1	1	1.27	79626.34	1	1
2	1.52	98808.3	1.088	0.92	1.22	76072.43	1.016	0.95	1.18	73663.53	1.113	0.99
3	1.37	93162.93	1.097	0.89	1.18	74786.49	1.096	0.96	1.13	69913.86	1.114	0.90
4	1.3	87410.88	1.096	0.85	1.12	73277.69	1.091	0.93	1.1	67746.77	1.108	0.85
5	1.28	84741.11	1.041	0.77	1.08	71817.41	1.117	0.93	1.06	68154.17	1.119	0.85
6	1.22	81707.36	1.089	0.74	1.06	70975.92	1.106	0.91	1.05	67045.21	1.121	0.81
7	1.22	80143.2	1.065	0.70	1.05	69902.95	1.116	0.89	1.03	65753.97	1.119	0.78
8	1.2	78589.86	1.005	0.71	1.03	69589.44	1.096	0.87	1.01	67361.31	1.117	0.80
9	1.21	78456.85	1.018	0.67	1.04	68800.65	1.100	0.87	1.04	66559.84	1.117	0.77
10	1.19	78643.86	1.072	0.69	1.01	68411.58	1.084	0.85	1.02	66185.75	1.122	0.77

TABLE III
AVERAGE OF DELAY, WIRE-LENGTH (WL), HORIZONTAL CHANNEL WITH (HCW), AND ROUTING AREA FOR MULTI-SEG ARCHITECTURE

Num layers	TPR Delay ($\times 10^{-7}$)	TPR WL	TPR Routing area	TPR HCW	SA-TPR Delay ($\times 10^{-7}$)	SA-TPR WL	SA-TPR Routing area	SA-TPR HCW	Combined Partitioning + SA Place Delay ($\times 10^{-7}$)	Combined WL	Combined Routing area	Combined HCW
1	1.62	107087.6	1	1	1.28	79410.53	1	1	1.27	79626.34	1	1
2	1.52	98808.3	1.088	0.92	1.2	76088.11	1.015	0.96	1.18	73663.53	1.113	0.99
3	1.38	93162.93	1.096	0.89	1.19	75304.16	1.068	0.96	1.14	70729.52	1.112	0.91
4	1.35	87410.88	1.053	0.83	1.12	73796.21	1.075	0.95	1.07	67383.43	1.107	0.86
5	1.24	84741.11	1.025	0.78	1.07	72228.37	1.045	0.90	1.06	66798.35	1.113	0.82
6	1.21	81707.36	1.038	0.74	1.03	70888.9	1.064	0.90	1.05	66682.31	1.115	0.80
7	1.2	80143.2	1.003	0.70	1.03	70710.4	1.052	0.88	1.03	65605.21	1.115	0.77
8	1.19	78589.86	0.942	0.70	1.02	69849.53	1.054	0.87	0.989	67049.56	1.113	0.81
9	1.19	78456.85	0.967	0.67	0.9.84	69190.21	1.062	0.87	0.987	65536.51	1.114	0.77
10	1.16	78643.86	0.981	0.68	0.971	68840.17	1.058	0.85	0.987	65439.38	1.115	0.77