Timing-driven Partitioning-based Placement for Island Style FPGAs

Pongstorn Maidee, Cristinel Ababei, Kia Bazargan, *Member, IEEE*

***Abstract*— In traditional FPGA placement methods, there is virtually no coupling between placement and routing. Performing simultaneous placement and detailed routing has been shown to generate much better placement qualities, but at the expense of significant runtime penalties [19]. We propose a routing-aware partitioning-based placement algorithm for FPGAs in which a looser but effective coupling between the placement and routing stages is used. The placement engine incorporates a more accurate FPGA delay model and employs effective heuristics that minimize circuit delay. Delay estimations are obtained from routing profiles of selected circuits that are placed and routed using the timing-driven versatile place and route (TVPR) [6][7]. As a result, the delay predictions during placement more accurately resemble those observed after detailed routing, which in turn leads to better delay optimization. An efficient terminal alignment heuristic for delay minimization is applied during placement to further optimize the delay of the circuit. These two techniques help maintain harmony between placement and routing delay optimization stages. Simulation results show that the proposed partitioning-based placement combined with more accurate delay models and the alignment heuristic can achieve post-routing circuit delays comparable to those obtained from TVPR while achieving a 4-fold speedup in total placement runtime. In another experiment, we augmented the original TVPR algorithm with the terminal alignment heuristic, and achieved on average 5% improvement in circuit delay with negligible runtime penalty.**

***Index Terms*— Field programmable gate arrays (FPGA), FPGA placement, timing-driven placement, partitioning-based placement, delay estimation.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) have become important implementation platforms because of their flexibility and cost benefits. Changes to design can be made late in the design cycle or even after the chip is employed in the final consumer product, hence increasing design flexibility and helping reduce time-to-market windows. Furthermore, mask and fabrication costs can be amortized over a wide range of applications that use the same FPGA architecture [22].

On the other hand, the increasing capacity and complexity of the FPGAs resulted in new challenges for CAD developers. Recent FPGAs can accommodate more than 16 times larger circuits than their predecessors as a result of new hierarchical architectures, more advanced routing structures, and dedicated on-chip circuitry (*e.g.*, adder carry chains and multipliers). This significant increase in size and complexity of the FPGA chips demands much more efficient CAD tools that can deliver shorter compilation times.

In the last decade, there have been significant improvements in placement and routing algorithms for standard cell and full custom designs. Traditionally, these algorithms have been adopted for FPGAs, although they have not been well-tuned to account for the special and limited routing resources on FPGAs. In standard cell routing, channel width inflation and over-the-cell routing can be employed to resolve congestion. Neither of these techniques is available in FPGAs where the number of wires, channel capacity, and the internal structure of switching boxes are fixed. As a result, placement and routing for FPGAs is more challenging.

High quality FPGA placement tools are based on Simulated Annealing (SA), which has the ability to escape from local optima [7][10]. This ability allows SA-based tools to explore larger regions of the solution space. As a result, high quality solutions are obtained, but at the expense of longer runtimes. However, considering the exponential increase in FPGA size, SA-based methods are becoming unsustainable. To avoid long runtimes, partitioning-based placement approaches have been proposed for standard cell placement [11][13]. However, its limited search space makes the partitioning-based placement approach inferior. It is desirable to achieve the lower computational complexities of divide-and-conquer methods (*e.g.*, partitioning-based / hierarchical) while obtaining the high quality of SA-based placement techniques. There have been many efforts that target developing fast placement and routing tools for FPGAs [8][9]. However, quality of placement is often sacrificed in such methods: they can achieve 50-fold speedup with a 33% quality degradation penalty [9]. Other approaches have tried to parallelize the placement or to develop a dedicated hardware implementation [3][4]. Our goal in this paper is to develop a fast placement method for FPGAs without losing placement quality or using more processing power.

Traditional FPGA placement methods suffer from two shortcomings: (a) the placement stage is not tightly coupled to the routing phase, which could result in the routing algorithm to partly nullify optimizations done at the placement level, and (b) routing delay models used in FPGA tools are inherited from their ASIC counterparts. The half perimeter bounding box model for the delay of a net is well suited for ASIC designs, but is not an accurate representation of the segmented-routing architecture employed in modern FPGAs

Manuscript received October 18, 2003, revised February 16, 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: {pongstor,ababei,kia}@ece.umn.edu).

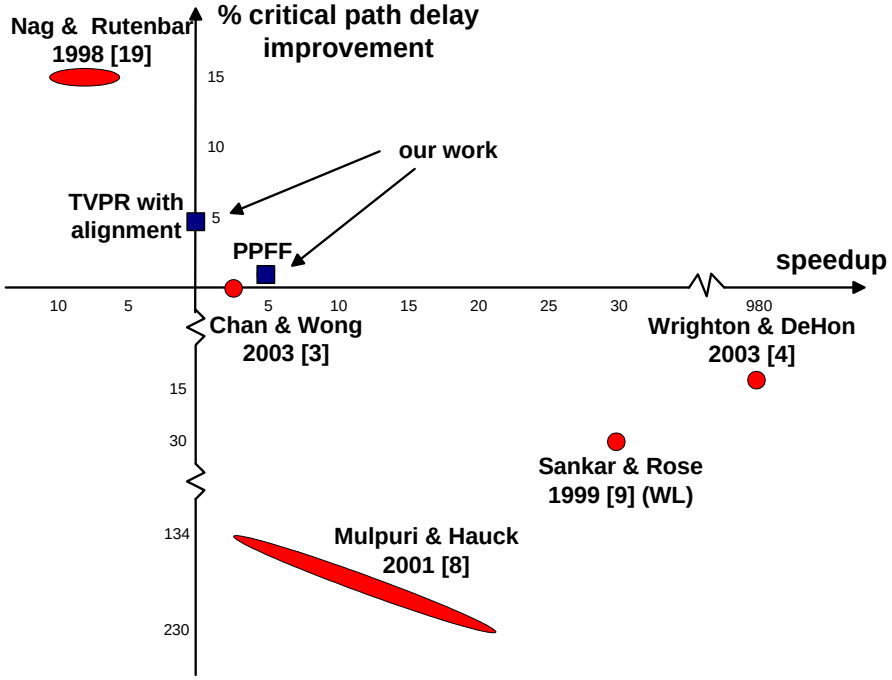


Fig. 1. Graphical comparison of different placement approaches proposed in the literature.

[15][16]. This makes the router behavior harder to predict compared to the ASIC case. Therefore, a placement algorithm, which does not take the router behavior into account, tends not to exploit the high performance capability of modern FPGAs.

To deal with the first shortcoming, both placement and routing can be concurrently considered. Such merging creates a comprehensive view of the problem and is shown to provide better results [14][18][19]. In these approaches, the circuit performance is improved by up to 15%, however the runtime is 6 to 11 times longer compared to TVPR [7].

A direct comparison between previous approaches cannot be made due to the differences in their assumptions and settings. However, we can depict a rough graphical comparison in Fig. 1. Note that the comparison to [9] is in terms of wire-length only. Also note that the authors in [4] report the same critical delay while using wirelength optimization for both TVPR (although in the "–fast" mode) and their proposed parallel placement method as well as using 35% more tracks than TVPR. They achieve 980x speedup using 2645 LUTs forming a systolic array to perform the placement optimization. As seen in Fig. 1, the existing placement algorithms can provide speedups at the expense of circuit quality. In this paper, we propose a looser coupling between placement and routing (*e.g.*, compared to [19]) to improve the performance in a partitioning-based formulation and still achieve speedup. To the best of our knowledge, our placement is the first placement that can provide both speedup and slightly better circuit performance compared to TVPR.

Our contributions can be summarized as follows:

- Use routing profiles to derive a model for estimating the routing resource usage as a function of net criticality and terminal distance. The devised model has to accurately capture routing delay and congestion of a placement and also has to be fast to evaluate so that it can be used within placement iterations.
- Use a net terminal alignment technique to minimize critical path delay. The alignment is used within the placement engine to provide a loose coupling between placement and routing in such a way that the small computation time of traditional design flow is maintained while benefiting from simultaneously considering both placement and routing.
- Determine the order in which partitions in a partitioning-based placement algorithm are placed to minimize placement constraints on terminals that belong to critical nets.

The rest of this paper is organized as follows. Section II cites some relevant works and summarizes the assumptions used in the discussions throughout the rest of the paper. In Section III our algorithm is presented by describing each step in detail. Then, simulation results are presented in Section IV. We conclude our paper and present directions for further work in the last section.

II. Preliminaries

Timing-driven placement for FPGAs can be classified into two main categories: net-based and path-based approaches. Net-based methods translate path criticalities (criticality is inversely proportional to slack) to net weights and treat nets independently. Path-based methods consider paths explicitly. Generally, path-based approaches are more accurate but slower than net-based approaches. Marquardt, et al., presented TVPR [7], the timing-driven version of VPR [6], which incorporates path-based timing analysis and connection-based analysis within the SA algorithm. They obtained delay

improvement at the expense of increase in wiring and runtime. In order to achieve better runtimes, weighted-edge partitioning is the choice for the placement methodology proposed by Hutton [17], although for hierarchical FPGA architectures.

We use a Virtex II [2] like FPGA architecture. The core of the FPGA is composed of an array of configurable logic blocks (CLBs), and routing resources. A CLB consists of one or more *n*-input lookup tables. Each lookup table can output either latched or unlatched signals. In this paper, we assume that the CLB has only one 4-input lookup table. However, our methodology can be used for FPGAs with CLBs containing

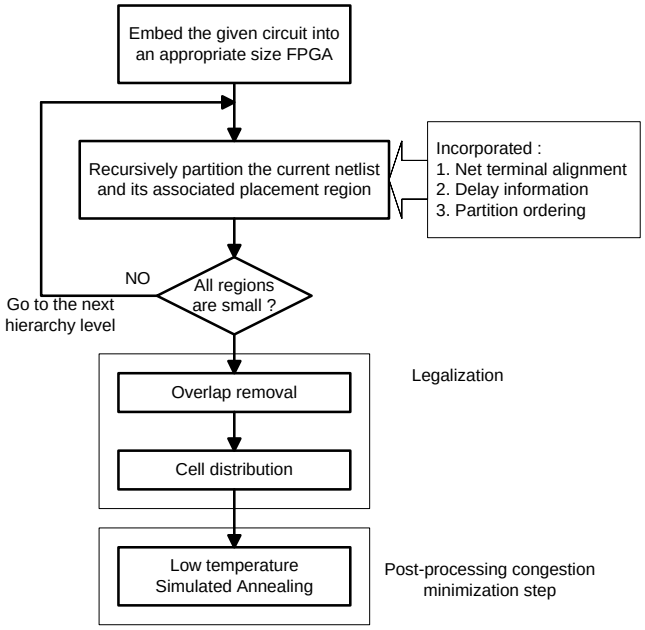


Fig. 2. Schematic diagram of proposed algorithm.

arbitrary number of inputs and any number of lookup tables. In such a case, technology mapping is used to group several lookup tables into a single CLB, which in turn is treated as a single cell at the placement level. Throughout the paper, we interchangeably use "node" or "cell" to refer to CLBs. Each CLB can access the routing segments using connection blocks. Switch boxes are used to connect wires either back-to-back or perpendicularly. Routing segments can have lengths of one, two, six or extend the entire width or height of the chip (called long wires). We also assume that wire segments are fully buffered. Therefore, fanout has a negligible impact on the delay of a net, especially during placement in which the exact routing is unknown. This assumption is consistent with modern FPGAs in which buffers are used to improve the net delay and make it more predictable [2][20]. As a result, we can treat a multi-terminal net as a set of two- terminal nets, each connecting the source to a sink. This means that we assume that all edges are independently routed and therefore, their delays are independent.

III. PPFF: Partitioning-based Placement For FPGAs

In this section we describe our partitioning-based placement framework, which simultaneously performs delay and congestion minimization. We adopted the net-based approach in which the path delay is taken into account by translating a path's slack to individual net criticalities (the criticality of a net is inversely proportional to the smallest slack of the paths that pass through the net).

The overall flow of our PPFF algorithm is shown in Fig. 2. We recursively partition the design and place it hierarchically during global placement. Because after placement there can be cell overlaps, we perform the legalization and cell distribution using a greedy method, followed by a post-processing detailed-placement step that employs low temperature SA.

Placement is done by recursively partitioning the circuit using the hMetis partitioning tool [1]. During partitioning we maintain a tight connection between the circuit graph and the placement (as coordinates of all cells on the FPGA fabric). This connection is key to the success of applying the net terminal alignment heuristic (defined in Subsection III.A) as well as to the accuracy of delay computations. Recursive partitioning is done until each leaf in the hierarchical partition tree contains less than four cells. All edges in the circuit graph to be partitioned by hMetis are weighted. The weights represent timing criticality of the edges calculated using the timing slack values:

$$criticality_i = 1 - \frac{slack_i}{\max\limits_{all\ edges} slack} \quad (1)$$

where max slack is positive.

We also define criticality of a node as the maximum criticality of its incident edges. Using timing criticality as edge weight discourages the partitioning engine from cutting edges with high criticalities. Therefore, critical nets will be kept short and the circuit will have a smaller delay. The process of delay assignment and slack/criticality update is performed at every partitioning level. Hence, timing criticalities will be more accurate and a better, tighter connection between timing-driven partitioning and placement is developed and maintained. A number of novel ideas used in our recursive partitioning steps are listed below:

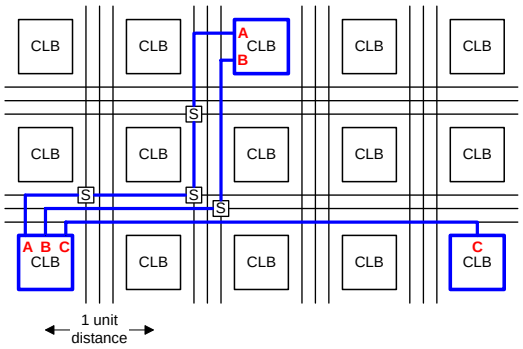


Fig. 3. Illustration of the terminal alignment of a generic two-terminal net.

- Net terminal alignment (presented in Subsection A).
- Partitioning order (covered in Subsection C).
- Net delay and timing criticality modeling based on

routing analysis (discussed in Subsection E).

In the following subsections we will discuss more details of each of these features.

A. Net terminal alignment for delay minimization

FPGA placement tools widely use wire length as the guide for minimizing circuit delay. However, wire length, calculated as the net bounding box, does not accurately capture the delay of a net because of the segmented routing architecture of modern FPGAs. It has been shown that the number of switches along a net could be as important as the Manhattan distance between its terminals, sometimes even dominating the distance [5][14]. Fig. 3 illustrates this point. Connections A, B and C have the same wire length, but exhibit different delays due to the difference in the number of switches they used. It can be observed that net C has the smallest delay.

At the placement level, we cannot predict – with 100% accuracy – the number of switches that the router will use to route a net. However, given the same placement as the terminals of nets A and B, the router is more likely to use fewer switches to route the more timing critical net between the two. Furthermore, we can certainly say that at least one switch is needed to connect the horizontal segments of the route to the vertical ones. On the other hand, the placement of the terminals of net C provides the opportunity for the router to avoid using any switches at all.

Terminals of a net are aligned if they are placed in the same row or column of the CLB array. For example, terminals of net C in Fig. 3, are aligned. Terminal alignment provides a loose coupling between placement and routing, by allowing the router to use fewer switches on timing critical nets. Note that wire length minimization will not distinguish between nets A, B and C, as they all have the same half-perimeter bounding box value. To study the effect of alignment on circuit timing and routability, we integrate the alignment technique into TVPR. We can define the alignment cost of a vertex as the weighted sum of square of its edge criticalities.

$$alignment\_\ cost(v_i) = \sum \delta(e) \cdot (criticality(v_i))^2 \quad (2)$$

$$\text{, where} \quad \delta(e) = \begin{cases} 0 & \text{, if } (v_i^x - v_j^x)(v_i^y - v_j^y) = 0 \\ 1 & \text{, if otherwise .} \end{cases}$$

Note that we square the criticality to give high priority to cells with higher criticalities. An example of the alignment cost computation is shown in Fig. 4.

The auto-normalization cost function used in TVPR is

$$\text{cost} = (1 - w_t) \cdot \frac{\Delta BB\_cost}{BB\_cost} + w_t \cdot \frac{\Delta\ timingCost}{timingCost} \quad (3)$$

$$\text{, where} \quad 0 \le w_t \le 1$$

Where *BB_cost* is the wirelength bounding box cost. We add the alignment cost to the cost function, while preserving the auto-normalization property:

$$\cos t = (1 - w_t - w_{al}) \cdot \frac{\Delta BB\_cost}{BB\_cost} + w_t \cdot \frac{\Delta\ timingCost}{timingCost} + w_{al} \cdot \frac{\Delta alignmentCost}{alignmentCost} \quad (4)$$

$$\text{, where} \quad 0 \le w_t, w_{al} \le 1 \quad \text{and} \quad 0 \le w_t + w_{al} \le 1$$

Even though we do not modify the random moves in TVPR to directly enforce alignment, we use w_{al} to favor moves that improve alignment. The effect of alignment on circuit delay was studied by varying a combination of cost function weights (see Equation (4)). The experiment was performed on the set of benchmark circuits introduced in Section *V*. TVPR's default weights for timing and bounding box costs are 0.5. The improvement over the original TVPR is shown in Fig. 5. The improvement on combinational circuits is larger than that on sequential circuits. The peak improvement is about 4.6% when timing, alignment and bounding box weights are 0.3, 0.2 and 0.5, respectively. Notice that the improvement decreases with w_{al}. This is because the alignment technique is not distance aware. In other words, if two terminals are aligned horizontally or vertically, their separate distance is irrelevant as seen by (2). Therefore, to successfully apply alignment, timing optimization based on distance should also be performed so that a balance is struck between alignment, timing criticality and routability.

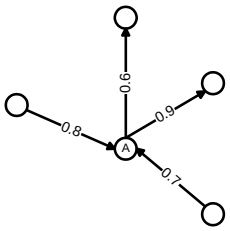


Fig. 4. The example of alignment cost computation. Alignment cost at node A = 1·0.7+1·0.8+0·0.6+1·0.9.

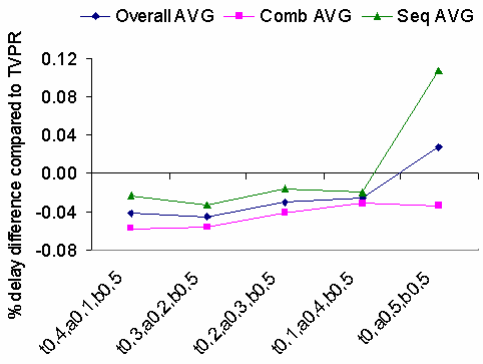


Fig. 5. Effects of alignment on circuits.

B. Alignment implementation in partitioning-based placement

In this subsection, we describe how we integrated the net terminal alignment technique into the partitioning-based placement. This is key to obtaining the most out of the alignment technique while maintaining the time complexity of

the partitioning-based placement. Partitioning-based placement recursively partitions a circuit in a breath-first manner. Each sub-circuit corresponds to one node in a recursion tree. All nodes with the same distance from the root node constitute one level in a partitioning-based placement. The alignment will be performed among the terminals within the same level only. Therefore, alignment also proceeds in a level-by-level basis.

We mentioned in Section III.A that terminal alignment allows the router to minimize the delay of the aligned net. However, excessive alignment can negatively affect congestion. Therefore, we will align only the most critical connections (*i.e.*, the connections with criticality values abave a specified threshold).

We define *anchors* as vertices of regions already partitioned at the current partitioning level. These vertices serve as references for other terminals that are going to be partitioned. Consider nodes *X* and *Y* of a two-terminal net shown in Fig. 6. Assume that node *Y* is an anchor. If node *X* is about to be placed in one of four partitions, *A*, *B*, *C* and *D*, placing *X* in region *A* or *B* can provide the alignment for the connection (*X,Y*). We would like to note that terminal alignment behaves differently from a wire length minimization heuristic or a traditional "terminal propagation" method used in most partitioning-based placement algorithms.

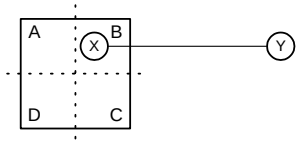


Fig. 6. Illustration of the terminal alignment of a generic two-terminal net.

If terminal propagation were used, a dummy node would be added to partition *B* and *X* is likely to be placed in *B* because *X*-*Y* is a critical connection. However, placing *X* in *A* or *B* does not necessarily make any difference in the delay of the connection (*X,Y*) due to the segmented routing architecture of the FPGA. Furthermore, if we used the wirelength metric, placing *X* in *A* or *C* would not make any difference, while it does for alignment.

As mentioned above, alignment does not distinguish

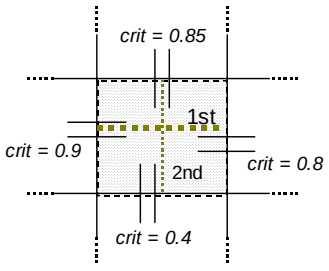
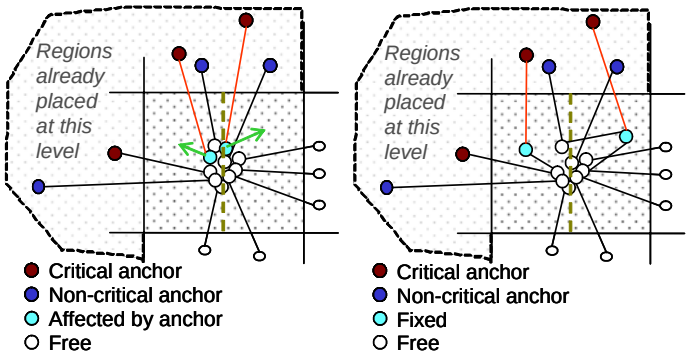


Fig. 7. Deciding the cut direction.

between *A* and *B* (or *C* and *D*). This means that the alignment of a net can only be done in one cut direction. For example, in Fig. 6, it only makes sense to consider alignment of *X*-*Y* when partitioning along a horizontal cut. We should also note that terminal alignment and terminal propagation are orthogonal techniques. For example, in Fig. 6, terminal propagation can be used in conjunction with terminal alignment. For a vertical cut, terminal propagation gets activated for net (*X,Y*), whereas terminal alignment only gets activated for this net when a horizontal cut is being made.

In a region to be partitioned into four regions, we perform two consecutive bi-partitionings, as opposed to one quadrisection. The reason is that by doing bi-partitioning, we can decide which direction (horizontal or vertical) to cut first to give priority to critical terminals. Let us call the left and right (top and bottom) borders of a partition region the vertical (horizontal) borders. Ideally, we would like to let the most



(a) (b)

Fig. 8. Net terminal alignment step during bi-partitioning.

critical net gain the most from alignment. Therefore, we perform the horizontal cut first if the maximum criticality crossing vertical borders is higher than that of the horizontal borders and do vertical cut otherwise. For example, in Fig. 7, the largest timing criticality among nets crossing the vertical borders of the placement region (*i.e.*, max{0.8,0.9}=0.9) is larger than that of nets crossing the horizontal borders (*i.e.*, max{0.85,0.4}=0.85). Therefore, the first bi-partitioning is a horizontally cut and the next two bi-partitionings in the newly created regions have to be done vertically.

The alignment implementation is summarized in Fig. 8, which shows how critical nodes are aligned with nodes placed in regions that are already partitioned. For demonstration purposes, we assume that all regions to the top and to the left of the region under partitioning in Fig. 8-a have already been partitioned. Among the six anchors, only two have critical connections to terminals in the region under consideration. Assume that the vertical cut will be carried out. Therefore, only the critical nodes that have vertical connections to anchors will be aligned as shown in Fig. 8-b. These nodes will be fixed inside the partition that aligns them, and the rest of the nodes are assigned to either partition by hMetis. The fixed nodes can affect how other nodes will be divided into the partitions, because of their connectivity to them.

Note that the partitioning-based placement imposes the placement restriction for a sub-circuit (*i.e.*, the cell cannot migrate to other sub-circuits). Therefore, it is easy to show that if the terminals are not aligned at a level, they cannot be aligned at lower levels.

C. Partition Ordering

After a region is partitioned, its nodes serve as anchors to other nodes in regions to be partitioned. To demonstrate the effect of partitioning order, we assume that partitioning is done from top to bottom and left to right as shown in Fig. 9. The regions in the third and forth columns have anchors in only the horizontal direction, while the regions in the second column have anchors in both directions. It can be seen that if we change the order in which regions are partitioned, the anchors available to a particular region would be different. We refer to this effect as region dependency. It can be observed that region 1 has the maximum flexibility in assigning nodes to its sub-partitions. Conversely, the region in the bottom-right corner of the chip will have the maximum number of alignment constraints.

We define dependency between two regions as follows: regions *a* and *b* are "directly dependent" if they are in the same row or column, and there is at least one critical edge connecting terminals of the two regions. When two regions are directly dependent, partitioning of one of them into four regions creates anchors for the other. Examples of dependent regions in Fig. 11-a are {*a,b*}, {*a,d*} and {*c,f*}, but not for example {*p,f*}. We also define "transfer dependency" as a transitive dependency between two regions. When regions *X* and *Y* have transfer dependency, there is a chain of direct dependent regions starting with *X* and ending in *Y*. For example, region *a* in Fig. 11-a has a length-4 transfer dependency with region *g* through *a,b,c,f,g* or *a,d,c,h,g*. We can consider the direct dependency as a transfer dependency of length one. Intuitively, the influence of a dependency decreases as its length increases. We have found that considering only dependencies of length one (direct dependency) provides good placement quality / runtime tradeoff. For direct dependency, we can solve the partition ordering problem in linear time.

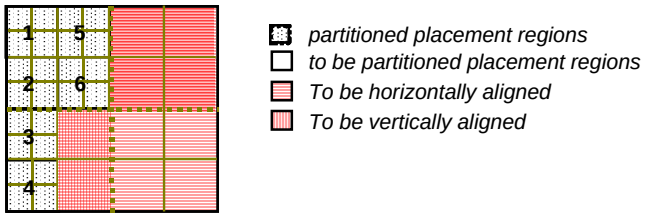


Fig. 9. Effect of a simple partitioning order.

For a set of regions, we can construct the corresponding undirected dependency graph *G*(*V,E,W*), where:

- *V* is the set of regions.
- $(a,b) \in E$ iff *a* and *b* are directly dependent regions.
- w_{ab}, the weight of edge (*a,b*), defined as the summation of the criticality of all critical connections between *a* and *b*. A connection is critical if its timing criticality is above a user-defined value.

Finding the best partitioning order in terms of dependency can be translated to a linear placement problem with minimum sum of incoming edge weights as follows.

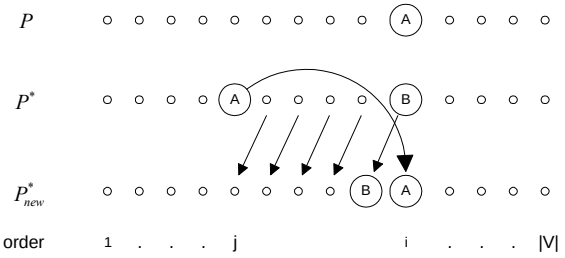


Fig. 10. The swap used in the optimality proof of the greedy algorithm.

Minimum incoming weight linear placement (**MIWLP**): For a given weighted-edge graph *G*(*V,E,W*), find a labeling function $L:V \rightarrow N$, that determines the order in which regions are partitioned, such that

$$\min \max_{v \in V} \sum_{L(u)<L(v)} w_{uv}, where\ u,v \in V\ and\ (u,v) \in E \tag{5}$$

Note that $w_{uv} = w_{vu}$. We can define $C(v)$ and get the cost function

$$\min \max_{v \in V} C(v) = \sum_{L(u)<L(v)} w_{uv}, \tag{6}$$

This problem can be solved optimally using a greedy algorithm that iteratively labels one node at a time. We label nodes from last to first. Let *S* be the set of unlabelled nodes and *S'* be the set of nodes that have been labeled. Since any node in *S* will have a smaller label than any node in *S'*, (6) can be rewritten as:

$$c(v) = \sum_{all\ u \in S,(u,v) \in E} w_{uv}, \quad for\ v \in S' \tag{7}$$

Note that initially $S' = \Phi$. Let us define S_i and S_i' as the set of non-labeled and labeled nodes at stage *i*, *i.e.* when only $|V|$ - *i* nodes have been ordered. Note that S_0 and S_0' are equivalent to *S* and *S'*, respectively. Therefore, the cost of node *v* at stage *i* can be expressed as:

$$c^i(v) = \sum_{all\ u \in S,(u,v) \in E} w_{uv} \tag{8}$$

The proposed algorithm for solving MIWLP problem is as follows.

Algorithm MIWLP

$S_{|v|} = \Phi, S'_{|v|} = |V|$

For $i = |V|$ down to 1 // NOTE the reverse order

a. Compute $C^i(v)$ for each $v \in S_i$.
b. b. Select node v_i in S_i such that $C^i(v_i)$ is smallest.
c. Label v_i as *i*, $S'_{i-1} = S''_i \cup \{v_i\}$, $S_{i-1} = S_i \setminus \{v_i\}$.

Lemma: If $L(v)=i$, then $C^k(v) = C^i(v)$, $\forall k<i$. In other words, the cost of a node will not change after it is labeled.

Proof: We can divide the edges incident to *v* into two groups: those connecting *v* to nodes with smaller labels ("incoming" edges), and those connecting *v* to nodes with larger labels ("outgoing" edges). The latter edges do not count in either $C^k(v)$ or $C^i(v)$. On the other hand, the contribution of an edge (*u,v*) for $L(u)<L(v)$ is independent of the actual value

of $L(u)$, as long as $L(u)<L(v)$. At stage i, we only know that $L(u) < L(v)$. At stage k, we might know the exact value of $L(u)$, but the inequality still holds, and as a result, the C value does not change. □

A direct result of this lemma is that the final value of the cost of a node (*i.e.*, $C^l(v)$) is the same as $C^i(v)$, where $L(v)=i$.

Claim: The above algorithm yields the optimal solution to MIWLP

Proof: To prove the claim, we start by showing that there exists an optimal ordering for the problem that contains node $A=v_i$, the node with the smallest current cost at stage i in the above algorithm as the i-th order.

Let P be the solution obtained by our algorithm with objective value of X. Let P^* be the optimal solution to the problem with objective cost X^*. Let i be the first position (in reverse order, *i.e.*, from $|V|$ down to i) that the ordering of the nodes differs in P and P^*. That is, nodes with labels i+1, i+2, ..., $|V|$ are the same in P and P^*. Assume that node B with $L^*(B)=i$ in P^* is the first node that is different from its corresponding node A in P, where $L(A)=i$ (see Fig. 10). Let the label of node A in P^* be $L^*(A)=j$.

Our goal is to convert P^* to P^*_{new} by replacing B with A, and show that the new objective function X^*_{new} is not larger than X^*. By recursively applying this transformation, we can change P^* to P and show by induction that X is also optimum.

Let us pick A out from the P^* solution, and fill the gap by decreasing (by one) the order of the nodes that have labels in the [j+1 ,i] range. The new ordering makes a new solution P^*_{new}. Because of the shifting of the nodes, the order of B will be $i - 1$ in P^*_{new}. Then, put A at the i-th order in P^*_{new} as shown in Fig. 10. Therefore, now $L^*_{new}(A) = i$ and $L^*_{new}(B) = i - 1$. The nodes with labels greater than i or less than j are the same in P^* and P^*_{new}.

From (8), we can infer the new cost of each node as follows:

1. $C^*_{new}(v) = C^*(v) \le X$: if $L^*(v) = L^*_{new}(v) > i$, because the incoming edges are the same (the order of the source nodes of the incoming edges might have changed, but that does not have any effect on the cost. Refer to the above lemma).

2. $C^*_{new}(v) = C^*(v) \le X$: if $L^*(v) = L^*_{new}(v) < j$, because the incoming edges did not change.

3. $C^*_{new}(v) \le C^*(v) \le X$: if $j < L^*(v) \le i$, because by moving A to the i^{th} order, we might have changed some incoming edges of v to outgoing, and hence decreased $C(v)$. Furthermore, we did not add any edges to the set of incoming edges of v.

4. However, $C^*_{new}(A) \le C^*(A)$, because some of the nodes with labels between j+1 and i in P^* might have edges to A, and these edges will become incoming for v in P^*_{new}.

5. From the lemma, $C^{*i}(B) = C^*(B) \le X$, and $C^{*i}_{new}(A) \le C^*_{new}(A)$.

6. Since the set of nodes with labels greater than i is the same in P, P^* and P^*_{new}, we have $C^{*i}(B) = C^{*i}_{new}(B)$.

7. By definition of node A, $C^{*i}_{new}(A) \le C^{*i}_{new}(B) = C^{*i}(B) \le X$. Therefore, $C^{*i}_{new}(A) \le X$.

Therefore, $X^*_{new} \le X^*$. However, since P^* is an optimum solution. Thus, $X^*_{new} = X^*$ and P^*_{new} is another optimum solution. Since the node with labels greater than i is the same for P and P^*_{new}, their costs are the same. If we repeat this transformation from any node down to the node at label 1, the above argument can be applied recursively. Finally, we will obtain a P^*_{new} which is optimum and has the same order of the nodes as P. Therefore, P is an optimal solution. □

A simple example of a dependency problem, its dependency graph, and its optimal MIWLP solution is shown in Fig. 11. Note that there may be many optimum solutions to the problem. For example, Fig. 11-c shows the optimum solution given by our algorithm and Fig. 11-d shows another optimum solution with the same incoming weight at each node.

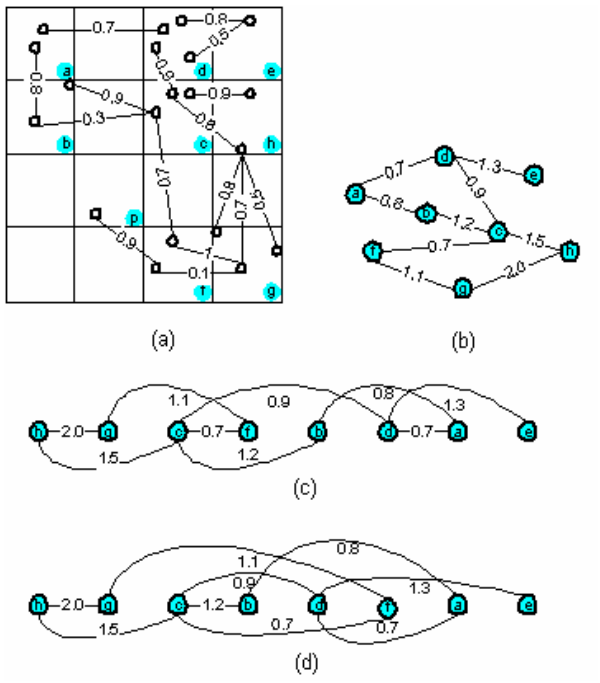


Fig. 11. The example of region dependency, its corresponding dependency graph and its MIWLP solution. Two solutions are shown in parts (c) and (d).

D. Overlap removal and cell distribution

The partitioning-based placement is effective only for coarse-level netlists. In our method, the partitioning process stops when the region size reaches a threshold (4 in our case). Once we stop the partitioning process, the resulting placement is illegal in two respects: First, some regions may have more nodes than they can accommodate. This is the result of the imperfect balancing of the partitions created by hMetis. Second, all the cells in the coarse region (*e.g.*, 2x2) are still at the center of the regions and have to be placed at individual CLB locations in the region. As shown in Fig. 2, we legalize the placement in two steps: overlap removal and cell distribution, both of which consider alignment during legalization.

In the overlap removal step, we move the least critical cells from an overcrowded coarse region to the closest region that can accommodate the cells and also provides the best alignment. After the overlap removal procedure is finished,

every region has enough space for its nodes. However, each region, occupying an array of CLBs (*e.g.*, 2x2), contains a number of cells, all of which are located at the center of the region. We use the distribution step to distribute cells into a CLB which best preserves their alignments. Distribution is performed in the following steps.

1. Order placement regions according to their external horizontal or vertical criticalities.
2. For each region, order its cells according to their external horizontal or vertical criticalities.
3. For each cell that has criticality greater than a threshold, find the best-aligned position and place it.
4. Randomly place the remaining cells.

Finally, the placement is refined by running TVPR's low temperature annealing algorithm on the whole circuit to further minimize wire length and delay (see Fig. 2).

One may argue that overlap removal can be done entirely by the low temperature SA, without the need for the legalization step. However, modifying the SA algorithm to address the overlaps would make the refinement step too complicated, as it would have to optimize many cost functions simultaneously. As a result, it might take much longer for the low-temperature SA to converge to a legal solution.

E. Delay Model and Timing Criticality Update

It has been shown that the number of segments traveled by a routed net is a more important factor in determining the delay of the net rather than the traditional geometric distance [5] [14]. As a result, delay estimation based only on Manhattan distance may be optimistic or pessimistic in an FPGA device with variable length segmented routing architecture [15] [16]. In standard cell designs, connection delay is closely related to its distance while its criticality is almost irrelevant provided that there is no congested area in the chip. Unless buffers are optimally inserted, the net delay is strongly dependent on the fanout of the net. In contrast, for fully-buffered FPGAs, the connection delay is largely independent from the fanout and this allows us to easily estimate the connection delay by the connection distance [20]. However, multi-length segmented FPGAs provide numerous combinations of segments to complete a connection, each with a different delay characteristic. The critical connection is required to be routed with the delay-optimal segment combination, while the non-critical can take longer delay routes. Therefore, two connections with the same length but different criticalities can have different delays.

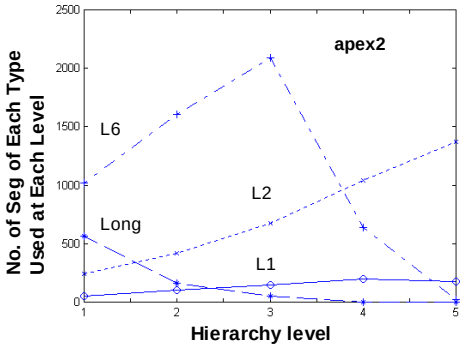


Fig. 12. Typical routing resource usage plot.

In this section, we address two delay estimation problems:

- The first tries to understand how the routing algorithm works so that the placement algorithm can tailor its optimization process to conform to routing optimization methods. Routing delay analysis is discussed in Section E.1.
- Next, we try to develop models, which can estimate the delay of the nets during partitioning-based placement. Section E.2 discusses issues related to delay estimation.

E.1 TVPR Routing Delay Analysis

We first perform an analysis of the TVPR router in order to better understand its behavior. We start with a profiling step for the routing resource usage. We use TVPR to place and route selected circuits and then superimpose an imaginary grid on the FPGA fabric, which represents the partitioning cut lines at different levels had the placement been done using a partitioning-based method as described in Section III. Nets crossing the grid lines corresponding to level *i* are counted to get the usage profile of every type of routing resources at that level. The characteristics of the set of circuits that we used in our experiments are shown in Table I of Section IV.

The key point of this step is that we noticed a common trend in the way routing resources are used by the TVPR routing tool. A typical routing resource usage is shown in Fig. 12. We can see that long segments are used extensively for routing nets that would be cut at higher levels of partitioning, while double-length segments are used mostly for nets cut at lower levels. Single-length segments are used almost uniformly across all levels. The shape of these plots is preserved irrespective of what placement tool is used. We performed experiments with three different placement engines: our placement algorithm, VPR, and random placement.

The main conclusion of the above analysis is that routing resource usage and therefore net delay is predictable. This allows us to adopt a lookup-table delay estimation technique tailored for the routing method that follows the placement. Therefore, for a particular architecture, it is reasonable to extract the delay from placed-and-routed circuits and store them according to their distances and criticalities. However, we have to discretize the criticality to reduce the table size. As the criticality of the partitioning-based placement is not accurate by nature (refer to Section III.E.2), dividing the criticality into 10 regions is enough to provide good delay estimation.

These delay lookup-tables store information about the average delay of nets with a given criticality, which span a given minimum length. These tables are then used inside our partitioning-based placement algorithm for delay assignment to nets cut at different partitioning levels.

We should point out that the delay after routing is what matters in determining the performance of a circuit. That is the reason that we gather delay profiles after routing and try to build a model that can capture routing behavior at placement level. If we are successful, the optimizations done by the placement algorithm are in line with what the routing is inclined to do. As a result, the estimation and optimization processes at the placement level will eventually be more effective.

E.2 Delay Estimation During Partitioning-based Placement

During recursive partitioning, edges are cut at different partitioning levels. Delay assignment to cut edges is done as follows. The minimum distance spanned by a cut net is determined by the level at which it is cut. We estimate the delay of a cut net based on its timing criticality at the time of partitioning and the minimum distance it spans. Our delay model takes into account both the number of segments used for routing as well as the segment lengths through the use of delay lookup-tables.

Nets cut at the first partitioning level are assigned delays corresponding to a single-length segment which is the smallest delay among all routing resources. During the subsequent partitioning levels, the (x, y) coordinates of all CLBs are more accurately known and so is the minimum length spanned by a net. Therefore, every net will be reassigned a delay according to its updated distance and criticality.

IV. Simulation Results

In Section III.A, we have shown the effectiveness of terminal alignment when incorporating it into simulated-annealing-based placement, TVPR (as shown in Fig. 5, about 5% improvement in circuit delay compared to TVPR's original placement). However, when we apply this technique to partitioning-based placement, we inevitably limit the range of alignment to only within sub-circuits. Therefore, we could expect to gain less improvement in this case. To verify the effectiveness of the proposed technique, we performed three sets of experiments and compared the results with those obtained by using TVPR. The aim of the first experiment is to test whether the information from the TVPR router can be fed to a post-processing placement algorithm to improve timing (*i.e.*, smaller delays after routing). This set of experiments is discussed in Subsection IV.A.

The second set of experiments compares our overall flow as a stand-alone tool to TVPR to see if the quality of the proposed method surpasses TVPR's. Subsection IV.B demonstrates the results. Finally, the third set of experiments tries to find the contribution of each of the heuristics that we used in our flow (*i.e.*, alignment, delay tables and partition ordering) to the delay improvements. Subsection IV.C covers these experiments.

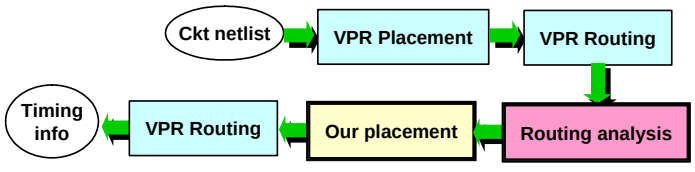


Fig. 13. Flow of the first set of experiments.

All simulations are averaged over six runs and are performed on a Linux machine with Pentium II Xeon 450Mhz and 2Gb memory.

A. Validating the Effectiveness of Feeding Routing Profiles to Placement

The purpose of this set of experiments is to first validate the usefulness of the routing profile information for the placement tool. It also presents our method as a post processing algorithm to improve timing. The flow of this set of experiments is shown in Fig. 13.

We first placed each circuit of the benchmark set using the original TVPR, then we used the TVPR router to route all circuits (the three top boxes in Fig. 13). Delay, runtime and channel width of each circuit after running TVPR is shown in columns 2, 3 and 4 of Table II. The overall delay (one to last row) is calculated as harmonic mean of all delay values while the overall runtime and channel width (CW) are the sum of all circuits.

Then, we extracted the delay information of each circuit as mentioned in Section III.E.1 (the box labeled "routing analysis" in Fig. 13). The delay information of each circuit was then used inside our partitioning-based placement algorithm to redo the placement of the same circuit. Finally we route the circuit using TVPR's router and compare its delay to that of the original TVPR's delay. The results are shown in columns 5-10 of Table II. The numbers in the group of columns labeled "*% delay difference compared to TVPR*" are calculated as 100 × (our_delay – TVPR_delay) / TVPR_delay. Negative numbers show improvement over TVPR while positive numbers show worse results compared to TVPR.

For each circuit placed using PPFF, we apply low temperature simulated annealing (*i.e.*, starting temperature is 10% of the original TVPR), with different cost function weights (as described in Section III.A, t and a represent w_t and w_{al}, respectively). Note that the t0.5, a0 case is equivalent to the original TVPR. It can be observed that circuits placed by

TABLE I
Statistics of Benchmark Circuits

Circuit	No. of CLBs	No. of I/Os	Circuit	No. of CLBs	No. of I/Os
ex5p	1064	71	ex1010	4598	20
misex3	1397	28	tseng*	1047	174
alu4	1522	22	diffeq*	1497	103
des	1591	501	dsip*	1370	426
seq	1750	76	s298*	1931	10
apex2	1878	42	bigkey*	1707	426
spla	3690	62	elliptic*	3604	245
pdc	4575	56	s38417*	6406	135

(* denote sequential circuit)

TABLE II
COMPARISON BETWEEN OUR RESULTS OBTAINED WITH UR PPFF AND TVPR
(TVPR router analysis information used for each circuit separately)

Circuit	TVPR			PPFF with delay data for individual circuit					
				% delay difference compared to TVPR				Time (/TVPR)	CW*
	Delay (e-8)	Time (sec)	CW	t0.5,a0	t0.4,a0.1	t0.3,a0.2	t0.2,a0.3		
ex5p	8.02	179	22	-5.20%	-4.20%	-6.90%	5.10%	0.30	24
misex3	7.48	243	19	-0.50%	4.40%	1.10%	9.70%	0.28	20
alu4	6.84	265	19	3.70%	-5.50%	-0.80%	12.00%	0.27	19
des	9.52	372	22	13.60%	0.60%	-1.60%	0.80%	0.20	22
seq	8.10	339	23	3.40%	-9.90%	-5.40%	-1.00%	0.27	23
apex2	8.91	402	22	-3.20%	-4.60%	1.30%	-0.20%	0.25	22
spla	13.60	1122	30	-9.80%	-5.70%	-6.10%	-3.70%	0.23	30
pdc	15.40	1567	32	0.00%	5.10%	15.30%	2.20%	0.23	32
ex1010	15.10	1355	22	-12.50%	-1.00%	2.20%	2.20%	0.26	22
tseng	4.77	188	18	7.90%	9.70%	5.40%	9.70%	0.28	18
diffeq	5.35	288	16	0.60%	6.80%	6.20%	12.50%	0.28	18
dsip	6.42	291	20	-3.70%	-14.90%	-0.90%	-19.10%	0.40	20
s298	9.65	348	18	16.10%	5.40%	7.60%	-0.70%	0.31	18
bigkey	5.45	374	19	-3.00%	3.00%	-2.10%	-7.90%	0.33	19
elliptic	7.59	1013	23	25.20%	7.20%	15.60%	10.00%	0.28	23
s38417	5.92	2224	18	3.90%	9.30%	12.70%	20.00%	0.24	18
Overall	7.66	10570	343	-0.20%	-1.10%	0.50%	3.70%	0.28	348
Avg Imp				-2.37%	-2.86%	-1.49%	-2.04%		

*CW is channel width

TABLE III
COMPARISON BETWEEN OUR RESULTS OBTAINED WITH UR PPFF AND TVPR
(TVPR router information used as the average of three circuits in bold face)

Circuit	PPFF with delay data for individual circuit					
	% delay difference compared to TVPR				Time /TVPR	CW*
	t0.5,a0	t0.4,a0.1	t0.3,a0.2	t0.2,a0.3		
ex5p	-6.40%	-10.40%	-1.20%	7.20%	0.300	24
misex3	1.00%	8.20%	4.40%	-0.70%	0.278	20
alu4	1.70%	5.40%	6.60%	12.60%	0.270	19
des	3.10%	0.60%	0.20%	2.30%	0.255	22
seq	-2.10%	-11.40%	-1.60%	0.10%	0.273	23
apex2	-3.40%	-3.80%	-4.10%	-1.10%	0.248	22
spla	4.90%	-8.60%	1.20%	-4.40%	0.236	30
pdc	-4.00%	8.00%	-3.20%	-2.40%	0.234	32
ex1010	-7.80%	-3.30%	-1.10%	-2.80%	0.257	22
tseng	10.00%	5.80%	7.40%	8.20%	0.280	18
diffeq	4.00%	7.10%	6.70%	7.10%	0.279	18
dsip	-18.80%	-4.20%	-12.60%	-17.00%	0.404	20
s298	10.10%	-4.70%	0.20%	0.30%	0.315	18
bigkey	2.90%	-3.40%	1.40%	-2.70%	0.331	19
elliptic	12.20%	30.80%	9.10%	6.90%	0.280	23
s38417	11.30%	13.50%	9.40%	15.90%	0.240	18
Overall	-0.30%	-1.00%	0.80%	1.90%	0.280	348

*CW is channel width

our placement algorithm have comparable delays to TVPR when timing and alignment weights are 0.4 and 0.1, respectively. The post processing step can lead to improvements as high as 19% (*dsip* circuit placed with timing and alignment weights of 0.2 and 0.3). Although the best weight combination for each circuit is different, from the overall improvement, we can conclude that the best combination is achieved when a moderate alignment weight is used. The decrease in improvement as the alignment weight increases, confirms the result obtained in Section III.A.

Note that we had to increase the channel width for some circuits (*e.g.*, *ex5p*, *misex3*, and *diffeq*) in order to achieve full routability. Therefore, we require five more tracks for the whole set of benchmark circuits which is about 1.5% increase in total number of channels. This number of tracks is also used for the second simulation setup.

Our experiments show that the delay results of a purely partitioning-based placement that does not consider the routing profiles and does not utilize low temperature annealing legalization is about 50% worse than TVPR, while using 34% more routing channel width. From the experimental results shown in Table II, we can conclude that using the routing profile tables in our partitioning-based placement provides better coupling between placement and routing. It confirms that the routing profile indeed helps the placement engine generate better results that conform to optimizations done at the routing stage.

Furthermore, Fig. 13 shows TPVR followed by our "post-processing" step. One can run our placement algorithm after TVPR, hence incur a 30% increase in runtime on average (in Table II, the average runtime of PPFF is 0.28 of TVPR). If the extra PPFF step results in deterioration of circuit delay, we can discard changes that PPFF made to the output of TPVR. Otherwise, we can keep the output of PPFF as the final placement. The last row of Table II follows the same logic. Entries in this row are the average of negative values. It shows that by using the flow of Fig. 13, we can achieve about 2% improvement over TVPR, with a penalty of 30% increase in runtime.

B. Comparing PPFF and TVPR

Previous experiments always ran TVPR followed by routing analysis and partitioning-based placement for each circuit. However, doing so is against our original goal of speeding up the placement process. We would like to find out whether the routing profile generated from a subset of circuits would have the same benefits for all circuits. If the answer to this question is positive, then we can run TVPR on only a few *representative* circuits, perform the routing analysis and store the results in a table. Then, we can avoid running TVPR altogether and use the routing profile information within our proposed PPFF flow. The flow of this set of experiments is shown in Fig. 14.

The second experiment is performed by using information about the TVPR router analysis as average of three different representative circuits, which are shown in bold face in Table III. The representative circuits were selected randomly from

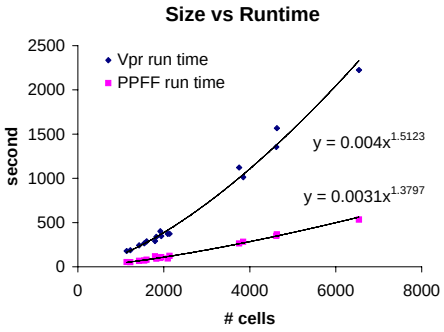


Fig. 15. Regression analysis of the runtimes.

small, medium and large circuits.

All circuits are placed with our algorithm and successfully routed with the TVPR router. It can be seen that the overall improvement exhibits the same trend as when the delay data of a circuit is used for itself and the best overall delay is approximately the same at the same cost function weight combinations. It is worth noting here that the parallel placement for FPGA can provide practically the same delay with a 2.3 fold speedup [3]. However, their experiments were done only for the combinational circuits and used many processors (our combinational results are better than our sequential circuits).

Since we use the TVPR profiling on a few circuits only, our

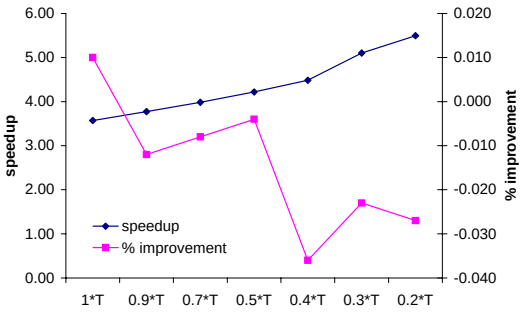


Fig. 16. % delay difference for difference starting temperature.

However, the benchmark circuits used in this simulation are small compared to the commercial circuits. To project the speedup that PPFF can obtain for those commercial circuits, the trend lines are plotted for both TVPR and PPFF as shown in Fig. 15. It can be seen that the order of growth of PPFF is smaller than that of TVPR which indicate that PPFF can achieve even more speedup for large circuits.

For some applications in which the quality is not the prime objective, we can further achieve more speed up by reducing the starting temperature in the low temperature Simulated Annealing phase. Fig. 16 shows the percentage delay difference when tuning the starting temperature with fixed cost function weights as 0.4 for timing and 0.1 for alignment. It can be seen that we can achieve 5.5 speedup with no more than 4% delay degradation.

C. Determining the Relative Effect of Our Heuristic Methods

As mentioned in Section III.E.2, the routing profiles are similar in shape regardless of the placement algorithm used. Therefore, it is possible to use the delay which is extracted by the circuit placed by PPFF which require less runtime than TVPR. To illustrate this idea, we performed an experiment

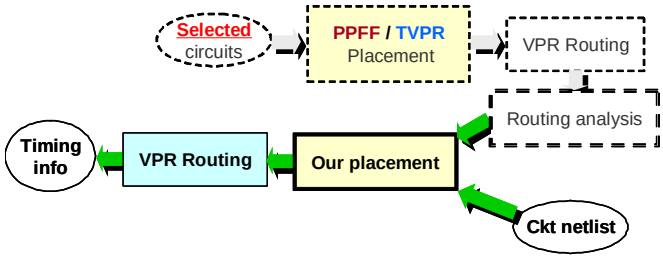


Fig. 17. The flow to determine the effect of different delay information.

following the flow shown in Fig. 17.

Two sources of delay information are studied: the delay information extracted from the PPFF placement method (shown in Fig. 14), and the delay information that is used inside TVPR. Note that the delay table used inside TVPR does not consider net criticalities: only wire length is considered in estimating the delay of a net. The experiment is performed by

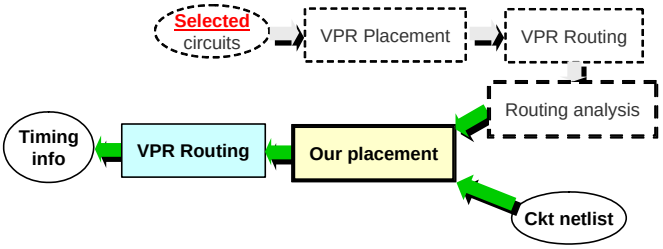


Fig. 14. The flow of the second set of experiments.

using the same channel widths as the ones used in Section IV.A and IV.B. When the delay tables of TVPR are used during our flow, the post-routing delay degrades by 2.3% on average compared to the second column in Table III (t0.4, a0.1). The above experiment showed that better delay estimations would improve the results of our method.

Now we would like to know the effect of each of the two

major components of our method: alignment and partition ordering. The contribution of each component is studied by disabling each in turn and comparing the results after routing. The channel width used for each circuit in this experiment is the same as that used in the experiment shown in Fig. 14, hence, fixing one parameter in the design quality metrics space to provide a meaningful comparison between different methods. The base case in these experiments is the post-routing delay of the second column of Table III (t0.4, a0.1).

We used the following methods to test the effectiveness of each of our techniques:

- Alignment: we disable the alignment during partitioning based placement, overlap removal, distribution and low temperature annealing (see Fig. 2). On average, this resulted in 3.4% quality degradation compared to the base case.
- Partition ordering: the best partition order determined by the algorithm proposed in Section III.C was replaced by random ordering. This resulted in 1.4% average quality loss compared to the base case.

V. Conclusions

We presented a partitioning-based timing-driven placement algorithm for FPGAs. We achieved almost 4x speedup over TVPR, with comparable circuit delays and a penalty of 1.5% increase in total channel width. We pointed out the importance of analyzing the behavior of the routing algorithm and using that model in the placement engine to generate placements that conform to routing optimization. We proposed an alignment technique, which facillitates placements, which are superior to the ones generated by the traditional bounding box minimization techniques. When TVPR is augmented with this technique, its results improve by 5% with virtually no extra runtime. Better placement qualities were achieved by using the proposed alignment technique in a partitioning-based placer compared to a traditional min-cut partitioning-based placer. We also proposed a technique to find the best partitioning order among placement regions in order to minimize alignment dependency during the integration of the terminal alignment into the partitioning based placement,.

We did not explicitly consider congestion while performing alignment, but implicitly reduce congestion by using partitioning (minimize cut between partitions, hence reducing the number of nets that connect them) and low temperature simulated annealing (minimizing wire length implicitly reduces congestion). We believe that performing a congestion-aware alignment can reduce the load of low temperature simulated annealing and allow faster convergence to high quality placements.

References

[1] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar , "Multilevel Hypergraph Partitioning: Application in VLSI domain," in Proc. Design Automation Conf., pp. 526-529, 1997.

[2] Xilinx Inc., The Programmable Logic Data Book, 2002.

[3] P. K. Chan and M. D. F. Wong , "Parallel Placement for Field-Programmable Gate Arrays," in Proc. ACM Int. Symp. FPGAs, pp. 43-50, 2003.

[4] M. G. Wrighton and A. M. DeHon , "Hardware-Assisted Simulated Annealing with Application for Fast FPGA Placement," in Proc. ACM Int. Symp. FPGAs, pp. 33-42, 2003.

[5] Y.-W Chang, K. Zhu and D. F. Wong , "Timing-Driven Routing for Symmetrical Array-Based FPGAs," ACM Trans. Design Automation Elec. Syst., vol. 5, no. 3, pp. 433-450, July 2000.

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

[7] A. Marquardt, V. Betz and J. Rose, "Timing-Driven Placement for FPGAs," in Proc. ACM Int. Symp. FPGAs, pp. 203-213, 2000.

[8] C. Mulpuri and S. Hauck , "Runtime and quality tradeoffs in FPGA Placement and routing," in Proc. ACM Int. Symp. FPGAs, pp. 29-36, 2001.

[9] Y. Sankar and J. Rose , "Trading quality for compile time: ultra-fast placement for FPGAs," in Proc. ACM Int. Symp. FPGAs, pp. 157-166, 1999.

[10] W. Swartz and C. Sechen , "Timing Driven Placement for Large Standard Cell Circuits," in Proc. Design Automation Conf., pp. 211-215, 1995.

[11] D. J.-H. Huang and A.B. Kahng , "Partitioning-based Standard-cell Global Placement with an Exact Objective," in Proc. Int. Symp. Physical Design, pp. 18-25, 1997.

[12] V. Betz, J. Rose and A. Marquardt, Architecture and CAD for Deep-submicron FPGAs, Boston, MA: Kluwer Academic Publishers, 1999.

[13] M. Wang, X. Yang and M. Sarrafzadeh , "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits," in Proc. IEEE-ACM Int. Conf. Computer-Aided Design, pp. 260-263, 2000.

[14] Y.-W. Chang and Y.-T. Chang, "An Architecture-Driven Metric for Simultaneous Placement and Global Routing for FPGAs," in Proc. Design Automation Conf., pp. 567-572, 2000.

[15] M. Khellah, S. Brown and Z. Vranesic , "Minimizing Interconnection Delays in Arrays-based FPGAs," in Proc. IEEE Custom Integrated Circuits Conf., pp. 181-184, 1994.

[16] V. Betz and J. Rose , "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density," in Proc. ACM Int. Symp. FPGAs, pp. 59-68, 1999.

[17] M. Hutton, K. Adibsamii and A. Leaver , "Timing-Driven Placement for Hierarchical Programmable Logic Devices," in Proc. ACM Int. Symp. FPGAs, pp. 3-11, 2001.

[18] N. Togawa, M. Sato and T. Ohtsuki , "A Simultaneous Placement and Global Routing Algorithm with Path Length Constraints for Transport-Processing FPGAs," in Proc. Asia South Pacific Design Automation Conf., pp. 569-578, 1997.

[19] S. K. Nag and R. A. Rutenbar, "Performance-driven simultaneous placement and routing for FPGAs," IEEE Trans. Computer-Aided Design, Vol. 17, No. 6, pp. 499-518, June, 1998.

[20] E.S. Ochotta, et.al., "A Novel Predictable Segmented FPGA Routing Architecture," in Proc. ACM Int. Symp. FPGAs, 1998, pp 3-11.

[21] R. Jayaraman, "Physical Design for FPGAs," in Proc. Int. Symp. Physical Design, pp. 214-221, 2001.

[22] P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen and B. Troxel, "A Hybrid ASIC and FPGA Architecture", International Conference on Computer-Aided Design, pp. 187 – 194, 2002.

Pongstorn Maidee (S'03) received the B.E. degree from King Mongkut's Institute of Technology Thonburi, Bangkok, Thailand, in 1993 and M.S. from University of Minnesota, Minneapolis, in 2003, both in electrical engineering. He is currently pursuing the Ph.D. degree in electrical engineering at University of Minnesota.

His research interests include computer-aided design for VLSI, especially physical design, and combinatorial optimization.

Cristinel Ababei (S'01) is a Ph.D. candidate at University of Minnesota. He received the M.S. degree in Electrical Engineering from University of Minnesota in 2002. He received the B.S. and M.S. degrees in Electrical Engineering from University of Iasi, Isai, Romania, in 1996 and 1998, respectively. His research interests include CAD for layout and logic synthesis for robust high-performance low-power VLSI circuits, FPGA synthesis and reconfigurable systems.

Kia Bazargan (S'97 M'00) received his B.S. in Computer Science from Sharif University in Tehran, Iran, and his M.S. and PhD in Electrical and Computer Engineering from Northwester University in Evanston, IL in 1998 and 2000 respectively. He is currently an Assistant Professor in the Electrical and Computer Engineering at University of Minnesota. He has served on the technical program committee of a number of IEEE sponsored conferences (e.g., ISPD, ICCAD, GLSVLSI). He is a guest co-editor of ACM Transactions on Embedded Computing Systems (ACM TECS), Special Issue on Dynamically Adaptable Embedded Systems. He was a recipient of NSF CAREER award in 2004.