# Timing-Driven Placement Based on Monotone Cell Ordering Constraints

Chanseok Hwang
Department of Electrical Engineering-Systems
Univ. of Southern California, Los Angeles, CA 90089
Tel : 1-213-740-4472 Fax : 1-213-740-9803
Email : chanseoh@usc.edu

Massoud Pedram
Department of Electrical Engineering-Systems
Univ. of Southern California, Los Angeles, CA 90089
Tel : 1-213-740-4458 Fax : 1-213-740-9803
Email : pedram@usc.edu

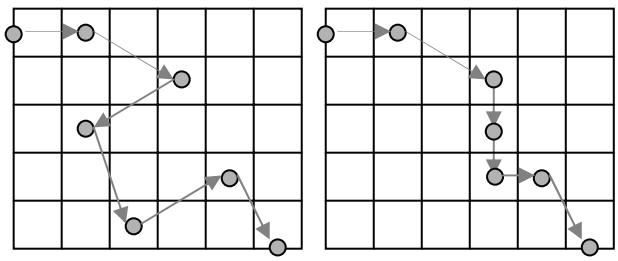
**Abstract– In this paper, we present a new timing-driven placement algorithm, which attempts to minimize zigzags and crisscrosses on the timing-critical paths of a circuit. We observed that most of the paths that cause timing problems in the circuit meander outside the minimum bounding box of the start and end nodes of the path. To limit this undesirable behavior, we impose a physical constraint on the placement problem, i.e., we assign a preferred signal direction to each critical path in the circuit. Starting from an initial placement solution, by using a move-based optimization strategy, these preferred directions force cells to move in a direction that maximizes the monotonic behavior of the timing-critical paths in the new placement solution. To make the direction assignment tractable, we implicitly group all circuit paths into a set of input-output conduits and assign a unique preferred direction to each such conduit. We integrated this idea into a recursive bipartitioning-based placement framework with a min-cut objective function. Experimental results on a set of standard placement benchmarks show that this approach improves the result of a state-of-the-art industrial placement tool for all the benchmark circuits while increasing the wire length by a tolerable amount.**

## I. Introduction

Timing optimization during placement has been an active area of research and development. This is in part due to the increasing ratios of the interconnect delays to the gate delays in deep submicron designs and the huge impact of cell placement on wire lengths, and therefore, longest path delays in the circuit. In general, a "good" timing-aware cell placement tool can positively influence the timing closure of the circuit, and thus, greatly reduce the overall design turn-around-time. There is therefore a need for efficient timing-driven placement algorithms especially for the design of high-performance ASICs.

Many techniques have been developed to optimize circuit delay during placement. These techniques may be broadly classified into two categories depending on whether they modify the netlist or not. Circuit delay during placement can be optimized by using buffer insertion, logic replication, or retiming techniques [1-4]. On the other hand, many techniques [5-12] do not alter the circuit netlist. These techniques often give high weights to or specify physical length constraints for the edges that lie on the critical timing paths of the circuit. These methods therefore require an a priori classification of signal nets into critical and non-critical ones based on a static timing analysis of the circuit. Most of the reported works use slack values to identify critical nets, and decide the net weights or net length constraints. Since net weights do not bear a direct relation to the circuit delay, it has been quite difficult to stabilize the net weights in order to achieve good timing convergence [6]. Net length (or size of net bounding box) constraints have a more direct relation to the timing constraints. However, it has been difficult to effectively incorporate these constraints in a placement tool without creating "solution oscillation" problems whereby the constraints on the current set of critical nets are satisfied at the expense of making some other nets timing-critical. In addition, these techniques tend to over-exert the current set of constraints by making the lengths of the critical nets much shorter than what they have to be in order to satisfy the current timing constraints. A number of researchers [7][8] have used the signal direction as an indicator of the *timing gain* function during the move-based partitioning process. Examples include "backward edges" [7] and "V-shaped nodes" [8]. These early results motivate the use of signal direction to guide the performance-driven placement process (see also the last paragraph of Section III(A))

In this paper, we introduce a novel approach to timing-driven placement, which employs a new type of physical constraint imposed on the circuit. More precisely, we impose constraints that specify preferred signal directions for the timing-critical *input-output conduits* in a circuit (see Section III for a formal definition of I/O conduits). These constraints then guide the cell placement so that timing-critical paths satisfy a type of monotonicity property in their cell ordering. Figure 1 depicts a critical path which has (a) non-monotone cell ordering and (b) monotone cell ordering. Clearly, the path with the monotone cell ordering will have a lower delay than the other path. This notion of monotonic path has also been used in logic synthesis to consider interconnect delay [13][20]. In [4], the logic replication was used to make such paths "straightened" for FPGA applications. Unlike their approach which uses logic replication, we employ the new physical constraint specifying preferred signal directions of the timing-critical *input-output conduits*. This idea has been integrated into a recursive bipartitioning-based placement framework with min-cut objective, which is a general top-down placement algorithm like that in [15]. The notion of the preferred signal directions of input-output conduits was described in [21]. The focus of that paper was however on circuit partitioning and does not consider two-dimensional placement in any form.



(a) Non-monotone cell ordering (b) Monotone cell ordering

Figure 1. An example of a critical timing path.

## II. Problem Statement

In this section, we describe our basic approach for timing optimization during a recursive partitioning-based placement.

Consider a sequential circuit, represented by a directed graph $G=(V, E)$. Each node $v_i \in V$ represents a combinational cell or flip-flop in the design. It has a weight $w(v_i)$ which specifies its

0-7803-9451-8/06/$20.00 ©2006 IEEE.

layout area. Let's denote the set of primary inputs of a circuit as PI, the set of primary outputs as PO, and the set of flip-flops as FF. We assume that the target chip area is known a priori and that PI and PO are placed at the boundary of the chip and remain fixed during placement. A path in the circuit is defined as the set of nodes and edges that connect a $p_i \in$ PI (or FF) to a $p_o \in$ PO (or FF). Path delay $d(\pi)$ can be calculated by the summation of delays of all the edges and nodes along path $\pi$. The minimum cycle time of graph $G$ is denoted by $\Phi_G$ and is equal to $Max\ d(S)$ where $S$ is a set of all paths in a circuit. The primary objective of a timing-driven placement tool is to minimize the cycle time of a circuit.

The timing optimization procedure in the context of recursive partitioning-based global placement engine typically consists of weighted wirelength-driven partitioning (WWP) and static timing analysis (STA.) More precisely, critical nets in the circuit are first identified based on STA and assigned higher weights. Next WWP decomposes the given placement instance into smaller instances by dividing the placement region into two sub-regions, and assigning cells to one or the other sub-region such that the weighted wire length is minimized and a balance condition on the total cell area of each sub-region is satisfied. This process continues until each region contains fewer then a certain number of cells.

## III. Proposed Approach

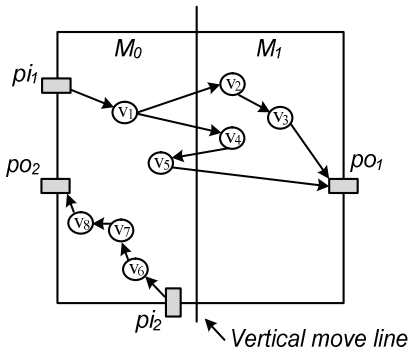
### *A. Signal Direction Constraints*

For completeness, we review here the notion of a signal direction of *input-output conduits* from [21], and the resulting constraint, which will be used to straighten critical paths in order to optimize circuit delay.

An input-output (I/O) conduit is defined as the set of all paths from some input node (in PI or FF) to some output node (in PO or FF). An I/O conduit, $\sigma$, is simply identified by the corresponding input ($p_i \in$ PI or FF) and output ($p_o \in$ PO or FF.) Notice that the maximum number of I/O conduits in a sequential circuit netlist is $(n_I+n_F).(n_O+n_F)$ where $n_I$, $n_O$ and $n_F$ denotes the cardinality of PI, PO and FF, respectively. An I/O conduit then belongs to one of the following types: PI→PO, PI→FF, FF→FF, or FF→PO.

In our approach a timing constraint is not explicitly specified for an individual path. Instead, it is defined for an I/O conduit (thereby it implicitly represents a constraint on a large number of paths.) We denote a timing constraint for an I/O conduit $\sigma$ by $c(\sigma)$. The delay of a I/O conduit is $d(\sigma) = \max d(\Pi)$ where $\Pi$ denotes the set of all paths between $p_i$ and $p_o$ of the I/O conduit. Then *critical I/O conduits* are defined as the set of I/O conduits $\Gamma$, such that for every $\sigma \in \Gamma$, $d(\sigma) \geq c(\sigma)$.

Signal direction constraints for critical I/O conduits are illustrated in Figure 2. A critical I/O conduit $\sigma_1$ from $pi_1$ to $po_1$ comprises of two critical paths $pi_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow po_1$ and $pi_1 \rightarrow v_1 \rightarrow v_4 \rightarrow v_5 \rightarrow po_1$. To achieve a monotone cell ordering of these paths, the signal directions of edges of $\sigma_1$ should be from part $M_0$ to part $M_1$. Let $P(v_i)$ denote the part that node $v_i$ is assigned to i.e., $P(v_i) = 0$ if $v_i$ is put in $M_0$, otherwise, $P(v_i) = 1$. Notice that $P(v_i)$ of the source node $v_i$ of an edge $e$ of $\sigma_1$ should not be any larger than $P(v_j)$ of the target node $v_j$ of that edge, and then both critical paths in $\sigma_1$ have a monotone cell ordering.




$\sigma_1$: $pi_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow po_1$, $e_1(pi_1,v_1)$, $e_2(v_1,v_2)$, $e_3(v_2,v_3)$, $e_4(v_3,po_1)$
$pi_1 \rightarrow v_1 \rightarrow v_4 \rightarrow v_5 \rightarrow po_1$, $e_1(pi_1,v_1)$, $e_5(v_1,v_4)$, $e_6(v_4,v_5)$, $e_7(v_5,po_1)$
$\sigma_2$: $pi_2 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow po_2$, $e_8(pi_2,v_6)$, $e_9(v_6,v_7)$, $e_{10}(v_7,v_8)$, $e_{11}(v_8,po_2)$

*Signal Direction Constraints*:

$P(s(e_i)) \leq P(t(e_i))$, $1 \leq i \leq 7$ for $\sigma_1$
$P(s(e_i)) = P(t(e_i)) = 0$, $8 \leq i \leq 11$ for $\sigma_2$

*where $P(v_i)$ is a part number (0 or 1) of $v_i$, and $s(e_i)$ and $t(e_i)$ are a source and target nodes of edge $e_i$, respectively.*

Figure 2. Signal direction constraints of critical I/O conduits.

This means that all critical paths in a critical I/O conduit $\sigma$ have a monotone cell ordering if and only if all edges of such critical paths satisfy signal direction constraints for $\sigma$. Notice that in the Figure 2 edge $e_6$ violates the signal direction constraints for $\sigma_1$, resulting in a non-monotone cell ordering. In addition, for I/O conduit $\sigma_2$, comprising of a single path $pi_2 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow po_2$, both source and target nodes of edges on $\sigma_2$ should be put in $M_0$ in order to satisfy the signal direction constraint of $\sigma_2$.

Let L, R, B, and T denote left, right, bottom, and top, respectively. Based on the above discussion, we define *signal direction constraints* (SDC's) for a *vertical move line* as follows:

$SDC^1$: if $SD(\sigma)=LL$, $\forall\ e_i \in \sigma$, $P(s(e_i)) = P(t(e_i)) = 0$

$SDC^2$: if $SD(\sigma)=RR$, $\forall\ e_i \in \sigma$, $P(s(e_i)) = P(t(e_i)) = 1$

$SDC^3$: if $SD(\sigma)=LR$, $\forall\ e_i \in \sigma$, $P(s(e_i)) \leq P(t(e_i))$

$SDC^4$: if $SD(\sigma)=RL$, $\forall\ e_i \in \sigma$, $P(s(e_i)) \geq P(t(e_i))$

where $SD(\sigma)$ denotes the signal direction of $\sigma$, which is one of *LL*, *RR*, *LR*, or *RL* for a vertical move line. Clearly, *LL* (*RR*) implies that both start and end nodes of the conduit are located in $M_0$ ($M_1$), whereas *LR* (*RL*) means that the start node of the conduit is in $M_0$ ($M_1$) while the end node of the conduit is in $M_1$ ($M_0$). The SDC's for a *horizontal move line* are obtained similarly (by replacing LL with BB, RR with TT, LR with BT, and RL with TB in the above equations.) For the remainder of this paper, we will only refer to vertical move lines since the case of a horizontal move line is really the same.

Based on the above definitions, each edge of every path in an I/O conduit has the same preferred signal direction. Therefore, although many paths of a conduit can go through an edge, the edge will have only one signal direction constraint (SDC) for the conduit. However, a placement solution that satisfies all of the SDC's associated with the timing-critical I/O conduits seldom exists for any realistic netlist. This is because, in general, an edge may belong to several critical conduits in the circuit, each assigning a preferred signal direction to the edge. Therefore, we

give up on the idea of trying to strictly impose SDC's. Instead we resort to minimizing a cost function which is proportional to the number of SDC violations.

We denote a violation of an SDC by SDV, which stands for a *signal direction violation*. To manage the circuit delay as a scalar objective function rather than as a set of signal direction constraints, we make use of the violation counts of signal directions as defined above. More precisely, in the framework of move-based local neighborhood search algorithm which is used during partition-based placement, we define a *timing gain,* $TG(v_i)$, to exactly quantify the desirability of moving $v_i$ from $M_0$ to $M_1$. The timing gain for a node $v_i$ is thus obtained by summing the number of SDV's of each edge $e_i$ connected to node $v_i$ as follows.

$SDV^1$: if $v_i = s(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 0$, then
$TG(v_i)$ −= $(SDC^1\text{-}cnt(e_i) + SDC^3\text{-}cnt(e_i))$

$SDV^2$: if $v_i = s(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 1$, then
$TG(v_i)$ −= $(SDC^2\text{-}cnt(e_i) + SDC^4\text{-}cnt(e_i))$

$SDV^3$: if $v_i = s(e_i)$ and $P(s(e_i)) > P(t(e_i))$, then
$TG(v_i)$ += $(SDC^1\text{-}cnt(e_i) + SDC^3\text{-}cnt(e_i))$

$SDV^4$: if $v_i = s(e_i)$ and $P(s(e_i)) < P(t(e_i))$, then
$TG(v_i)$ += $(SDC^2\text{-}cnt(e_i) + SDC^4\text{-}cnt(e_i))$

$SDV^5$: if $v_i = t(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 0$, then
$TG(v_i)$ −= $(SDC^1\text{-}cnt(e_i) + SDC^4\text{-}cnt(e_i))$

$SDV^6$: if $v_i = t(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 1$, then
$TG(v_i)$ −= $(SDC^2\text{-}cnt(e_i) + SDC^3\text{-}cnt(e_i))$

$SDV^7$: if $v_i = t(e_i)$ and $P(s(e_i)) > P(t(e_i))$, then
$TG(v_i)$ += $(SDC^1\text{-}cnt(e_i) + SDC^4\text{-}cnt(e_i))$

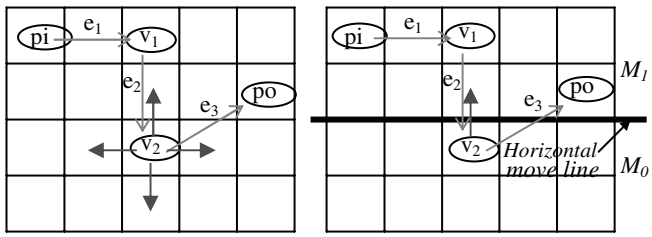
$SDV^8$: if $v_i = t(e_i)$ and $P(s(e_i)) < P(t(e_i))$, then
$TG(v_i)$ += $(SDC^2\text{-}cnt(e_i) + SDC^3\text{-}cnt(e_i))$

where $SDC^*\text{-}cnt(e_i)$ represents the number of signal direction constraints of type * (ranging from 1 to 4) for edge $e_i$, that is, the number of timing-critical I/O conduits with the corresponding signal direction that go through the edge. Notice that these counter values are pre-computed before we begin the cell movements for the purpose of timing optimization. The algorithm for setting the SDC-count is described in Section III(B).

Note that the early works [7][8] that use the signal direction to minimize cutsize cannot solve the problem globally. More precisely, in these references, the authors attempt to optimize local directions of edges without considering the parent path and its criticality. Unlike these methods, we aggregate preferred signal directions for all critical paths that pass through an edge, which in turn enables us to exactly calculate the global signal directions, resulting in maximization of the monotonic behavior of the critical paths.

### *B. Timing Optimization Process*

In a recursive bipartitioning-based timing-driven placement, STA is performed at each level of the partitioning hierarchy in order to first identify the timing-critical nets, and then to assign them higher weights in order to prevent them from being cut at the subsequent partitioning step. From our experimentations, we have observed that timing analysis and optimization at early hierarchical levels are not helpful in reducing the circuit delay.




(a) Move directions (b) $v_2$ is moved to the upper region

$\sigma$: $pi \rightarrow v_1 \rightarrow v_2 \rightarrow po$, edges: $e_1(pi,v_1)$, $e_2(v_1,v_2)$, $e_3(v_2,po)$
$SDC^2$: $SD(\sigma)=TT$, $\forall\, e_i \in \sigma$, $P(s(e_i)) = P(t(e_i)) = 1$
$SDC^2\text{-}count(e_2) = 1$, $SDC^2\text{-}count(e_3) = 1$
$VC_{(V2:P(v_2)=0)} = 2$ // *SDC violations before $v_2$-move*
$\Rightarrow$ *$SDC^2$ violated for $e_2$ and $e_3$.*
$VC_{(V2:P(v_2)=1)} = 0$ // *SDC violations after $v_2$-move*
$\Rightarrow$ *$SDC^2$ violations for $e_2$ and $e_3$ are eliminated.*
$\therefore TG(v_2) = VC_{(V2:P(v_2)=0)} - VC_{(V2:P(v_2)=1)} = 2$

(c) Computation of timing gain for $v_2$-moving to the upper region

Figure 3. An example of a cell move for timing optimization.

This is because the size of net bounding box, which is typically used for calculating the interconnect parasitics, is too rough at such levels where the chip area is divided into only a few sub-regions. Based on this observation, we start our timing optimization process after a few runs of the recursive partitioning with min-cut objective. The starting level of hierarchy for the timing optimization process is obviously a function of the circuit netlist size and the chip bounding box. In this way, we start with *an initial global placement* which has been optimized for minimum wire length objective.

We use an accurate internal STA engine, which uses the Elmore delay model and net-length estimation method proposed in [14] to calculate the wire delay, and a commercial timing library to obtain the gate delays. Based on the results of the timing analyzer, we identify the critical edges and nodes as follows: edges with negative slack values are marked as *critical edges* and nodes which have at least one critical incoming and/or outgoing edge are marked as *critical nodes*. Next, we find *critical I/O conduits* for each critical edge using a modified depth-first-search algorithm (MDFS), which visits only those successor nodes that are connected to their parents by critical edges. We add a source node and a sink node to the directed graph. Next we run a reverse-MDFS to find all transitive PI's and FF's for each node $v_i$ and stored them as set $S_i$ at that node. Similarly, all transitive PO's and FF's are searched for and stored at set $T_i$ at the node during another MDFS. As a result, we can determine, $C_{ij}$, the set of all conduits that go thru any critical edge $e_{ij}$ between nodes $v_i$ and $v_j$ in the circuit graph as the Cartesian product of the sets $S_i$ and $T_j$. Now, we count the number of critical conduits of type *LL*, *LR*, *RL*, and *RR* in $C_{ij}$ for a vertical move line, and thereby, initialize the corresponding SDC-count for all critical edges in the circuit.

We explain the timing gain calculation with the help of example in Figure 3. In the Section III(A), we described the timing gain calculation for the case of a move to a neighboring region over a vertical move line. The timing gain for a move across a horizontal move line can be calculated in a similar manner. Consider moving

a critical node $v_2$ in one of four directions, calculating the timing gain for each direction of movement. $v_2$ will be moved in the direction with the highest gain. The timing gain calculation for $v_2$ moving to the top region is shown in Figure 3(c). Signal direction constraint of the critical I/O conduit $\sigma$ passing $v_2$ is $SDC^2$ since both *pi* and *po* of this conduit are in the upper region over the horizontal move line. There are two edges connected to this node. Edges $e_2$ and $e_3$ do not satisfy $SDC^2$ of conduit $\sigma$. This is because $SD(\sigma)=TT$ but the source and target nodes of these two edges are not in $M_1$. The number of *SDC* violations is thus 2. After $v_2$ is moved to the upper region, $SDC^2$ can be satisfied for both $e_2$ and $e_3$. As a result, the total number of *SDC* violations are reduced by two, i.e., the timing gain for the $v_2$-*move* is two, $TG(v_2) = 2$. We calculate timing gains for other directions in the same way, resulting in $TG(v_2) = -2$ for $v_2$ moving to the left region, $TG(v_2) = 0$ for $v_2$ moving to the right region and $TG(v_2) = -2$ for $v_2$ moving to the bottom region. The maximum timing gain of $v_2$ is then 2.

After computing the timing gains for all critical nodes, we put them into a gain heap where nodes are sorted by their gain (highest gain move is root of the heap.) Next we extract the root node from the heap. Whenever a node $v_i$ moves to its preferred region $r_p$, we update gains of nodes connected to $v_i$ which are remaining in the heap and re-order it so that the root is the node with highest gain. If the remaining capacity of the region $r_p$ is zero, then we choose a node $v_j$ among non-critical nodes in that region based on the computation of wirelength gains for those nodes, and move it to the region where $v_i$ is coming. This process continues until the timing gain heap is empty. The running sum of the total timing gain for the moves is constructed during this process in order to identify a sequence of moves that produces the maximum total gain. Moves that are not part of the accepted move sequence are reversed. We call these steps as a *pass*, which is similar to the mechanism used in a general FM partitioner[16]. We go through multiples passes until no further timing gain can be achieved. Figure 4 shows the pseudo-code for the proposed timing optimization flow.

### C. Timing-driven Placement

In this section, we describe the flow of our proposed timing-driven placement. The placement framework is based on a recursive bipartitioning-based placement algorithm, which comprises of a hierarchical bipartitioning, terminal propagation and legalization. Our proposed timing optimization process is integrated into this framework. We used hMetis [17] as a bipartitioning algorithm, which consists of three phases: coarsening, initial partitioning and uncoarsening phases.

After each bipartitioning, those cells in a sub-region which are connected to external cells are propagated to the boundaries of the corresponding sub-regions. This terminal propagation is performed in a straight-forward manner based on shortest path (or a low-cost Steiner tree) connection of connected terminals. Finally, we allocate all cells that are contained in each sub-region into placement rows when the recursive bipartitioning reaches a certain pre-specified *end level*. This step is typically called *legalization*. We employed a simple technique whereby we divide each row to several equal-sized *bins*, and then, assigned cells in a sub-region to bins according to their coordinates. This assignment may cause unbalances in the total cell size of each bin. To reduce the unbalance, we move cells from "overfilled" bins to "underfilled" bins, by a technique similar to that in [18]. Next, within each row, cell positions are adjusted to eliminate any cell overlaps. This whole procedure is described in Figure 5 and the layout hierarchy of bipartitioning-based placement is shown in Figure 6.

First, we calculate the start hierarchy level based on the target size of the smallest region before we start the timing optimization procedure. We obtained the target initial size of a region by experimentation, and from that size, calculated the start level. The end level is reached when the size of a sub-region of a hierarchy level becomes smaller than ten times the average cell size in the design. Next we ran a wirelength-driven bipartitioning-based placement algorithm until we reached the start level. This step resulted in the *initial global placement*.

Timing _Optimization_PSD (*P*,*T*)

*P* : An initial hierarchical placement solution with J regions
*T* : Timing constraints

1. Perform static timing analysis;
2. From *T*, find critical edges, nodes, and I/O conduits (initialize corresponding SDC-count for all critical edges);
3. Compute initial timing gains for all critical nodes;
4. Put all critical nodes into a timing gain heap;
5. While (heap != empty)
6. Extract root node $v_i$ from the heap and move it in its preferred direction to a neighbor region in *P*;
7. If the region capacity is violated, select a non-critical node in the region and move it back to the parent region of $v_i$;
8. Update timing gains and restructure the heap as needed;
9. Find a sequence of moves that produces max_total_gain;
10. Undo moves that are not in the selected sequence;
11. If max_total_gain > 0 then goto step 3;
12. Else exit;

Figure 4. Flow of the proposed algorithm for timing optimization with preferred signal directions.

PSD_Placement (*G*, *T*)

*G* : A directed graph representing a sequential circuit
*T* : Timing constraints

1. Calculate the start and end levels of timing-driven global placement;
2. Do initial wirelength-driven global placement from level one to start level;
3. While (start_level ≤ *i* ≤ end_level)
4. While (*j*=0; *j* < number of sub_regions in level *i; j*++)
5. Generate a bipartitioning-based placement $P_{i,j}$ of subregion *j*;
6. Do Timing_Optimization_PSD(*Pi* ,*T*);
7. Do the legalization;

Figure 5. Flow of the proposed preferred signal direction placement algorithm.

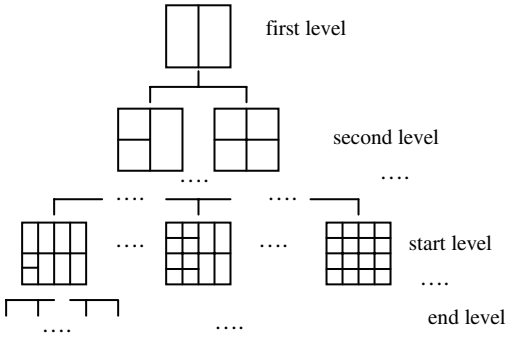



Figure 6. The layout hierarchy of bipartitioning-based placement with level descriptions.

Next we applied the *Timing_Optimization_PSD* to each level of the hierarchy between the start and end levels. Note that the timing optimization procedure is performed only once per hierarchical level on placement solution $P_i$, which itself comprises of $J=2^i$ sub-regions. In Figure 6, for example, at the start level, first eight bipartitionings are performed to divide the chip area into 16 equal-sized sub-regions. Next, the timing optimization is done on the global placement solution with 16 sub-regions. After reaching the end level, to allocate cells into placement rows without overlaps, a legalization step is performed.

## IV. Experimental Results

We have implemented the proposed timing optimization algorithm and bipartitioning-based hierarchical placement flow in C++ on a Sun Ultra Sparc II machine, and tested it on six industry circuits. Four of them, matrix, vp2, mac1 and mac2, are among the ISPD 2001 Circuit Benchmarks that first appeared in [19]. These circuits are also used in [5]. The characteristics of the benchmark circuits are summarized in Table 1. We call our timing-driven placement approach as PSDP (stands for Preferred Signal Direction Placement). We compared PSDP with Capo-boost [22], which attempts to improve circuit delay by reducing the number of global interconnects, and an industrial placement tool, which we call QuadP[1]. We use a 0.18µm standard-cell library to report the delay results.

TABLE 1. The characteristics of benchmark circuits

| Circuits | #Cells | #Nets | #IOs |
|---|---|---|---|
| indust1 | 5931 | 5969 | 179 |
| indust2 | 20193 | 21699 | 351 |
| matrix | 3,083 | 3,200 | 117 |
| vp2 | 8,714 | 8,789 | 321 |
| mac1 | 8,902 | 9,115 | 211 |
| mac2 | 25,616 | 26,017 | 415 |

[1] *QuadP* represents the virtual name of a commercial state-of-the-art placement tool.

Let *total negative slack*, TNS, denote the sum of the slacks of all paths with negative margins. Table 2 compares TNS between the non-timing mode and the timing-driven mode of PSDP. PSDP in non-timing mode (wirelength-driven) is the same as algorithm in Figure 5 with step 6 removed. To obtain the TNS values, we used our STA engine (which uses a commercial timing library to obtain the gate delays and relies on the Elmore delay calculation for interconnects) and assigned the clock cycle time of each circuit as the maximum of "no-wiring path delays [6]" in that circuit. The "no-wiring path delay" accounts for the delay of all gates on the path, but sets the corresponding wire delays to zero. We achieved an average of 44.5% improvement in TNS by using PSDP timing-driven mode.

TABLE 2. Comparison of TNS (total negative slack of all timing endpoints) between wirelength-driven and timing-driven mode of PSDP with the zero-loading delay clock cycle.

| Benchmark circuits | Clock cycle | Wirelength-driven mode | Timing-driven mode | % Improvement |
|---|---|---|---|---|
| indust1 | 5.54 | -38.2 | -24.4 | 36.1% |
| indust2 | 8.75 | -204.5 | -93.1 | 54.5% |
| matrix | 3.23 | -5.8 | -4.3 | 25.9% |
| vp2 | 3.67 | -68.3 | -25.1 | 63.3% |
| mac1 | 2.07 | -21.4 | -13.5 | 36.9% |
| mac2 | 2.35 | -125.4 | -62.7 | 50.2% |
| **Average** | | | | **44.5%** |

Table 3 compares PSDP with QuadP in wirelength-driven mode and in timing-driven mode, and Capo-boost in terms of the post placement wirelength (HPWL) and post routing wirelength (RWL), and the post-routing *worst negative slack* (WNS). We perform Cadence WarpRoute to route the placements obtained from each placer, extract RC values, and run Pearl to perform static timing analysis (STA). We use the values in [5] as the clock cycle for the corresponding four circuits. The other two circuits are available in complete LEF/DEF/GCF format. The wirelength and worst negative slack are represented in microns and in nanoseconds, respectively.

We observe that PSDP in timing-driven mode improved WNS for all circuits compared to QuadP in the wirelength-driven mode, on average, by 31%, while increasing the total wirelength of post placement and post routing, on average, by 5% and 4%, respectively. In addition, PSDP usually has a better result in terms of WNS compared to the other two placers, QuadP in timing mode and Capo-boost; our placer outperformed those placers for all benchmark circuits except one. PSDP runs on average 48% slower than QuadP in non-timing mode, but PSDP is on average 58% faster than QuadP in timing-driven mode.

## V. Conclusions

The paper integrates wire planning into timing-driven min-cut placement. It formulates a new kind of constraint on cell locations based on preferred signal directions. These preferred directions are deduced by grouping all paths from one major source to one major sink into I/O conduits. All paths in the entire circuit are grouped into these conduits. Constraints are computed for all cells in this way, and they are then used to guide the optimization step

by forcing the cells to move in a direction such that the timing-critical paths exhibit a monotonic behavior in their cell ordering. The advantage of the new methodology has been confirmed by experimental results; our placer achieves on average 31% improvement on WNS compared to a leading industry placer at the expense of wirelength increase, on average, by 5%.

## Reference


[1] J. Cong and X. Yuan, "Multilevel Global Placement with Retiming", In *Proceedings of the ACM/IEEE DAC*, 208-213, 2003.

[2] M. Hrikic, J. Lillis and G. Beraudo, "An Approach to Placement-Coupled Logic Replication", In *Proceedings of the ACM/IEEE DAC*, 711-716, 2004.

[3] P. Saxena and B. Halpin, "Modeling Repeaters Explicitly Within Analytical Placement", In *Proceedings of the ACM/IEEE DAC*, 699-704, 2004.

[4] G. Beraudo and J. Lillis, "Timing Optimization of FPGA Placements by Logic Replication", In *Proceedings of the ACM/IEEE DAC*, 196-201, 2003.

[5] X. Yang, B. Choi and M. Sarrafzadeh, "Timing-Driven Placement using Design Hierarchy Guided Constraint Generation", In *Proceedings of the IEEE ICCAD*, 177-180, 2002.

[6] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdlhary and B. Halpin, "Timing Driven Force Directed Placement with Physical Net Constraints", In *Proceedings of the ACM/IEEE ISPD*, 60-66, 2003.

[7] J. Cong and S.K. Lim, "Performance Driven Multiway Partitioning", In *Proceedings of the ACM/IEEE ASP-DAC*, 441-446, 2000.

[8] A. B. Kahng and X. Xu, "Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-driven bipartitioning." In *ACM/IEEE ISPD*, 81-86, 2003.

[9] A. B. Kahng S. Mantik and I. L. Markov, "Min-Max Placement for Large-Scale Timing Optimization", In *Proceedings of the ACM/IEEE ISPD*, 143-148, 2002.

[10] W. Choi and K. Bazargan, "Incremental Placement for Timing Optimization", In *Proceedings of the IEEE ICCAD*, 463-466, 2003.

[11] B. Halpin, C. Y .Chen and N. Sehgal, "Timing Driven Placement using Physical Net Constraints", In *Proceedings of the ACM/IEEE DAC*, 780-783, 2001.

[12] S. Hur, T. Cao, K. Rajagopal, Y. Parasuram and B. Halpin, "Force Directed Mongrel with Physical Net Constraints", In *Proceedings of the ACM/IEEE DAC*, 214-219, 2003.

[13] W. Gosti, A. Narayan, R. K. Brayton and A. L. Sangivanni-Vincentelli, "Wireplanning in Logic Synthesis", In *Proceedings of the IEEE ICCAD*, 26-33, 1998.

[14] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, "Multi-objective Circuit Partitioning for Cutsize and Path-based Delay Minimization", In *Proceedings of the IEEE ICCAD*, 181-185, 2002.

[15] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection produce routable placements", In *Proceedings of the ACM/IEEE DAC*, 477-482, 2000.

[16] C. Fiduccia and R. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", In *ACM/IEEE DAC*, 175-181, 1988.

[17] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph partitioning", In *Proceedings of the ACM/IEEE DAC*, 526-529, 1997.

[18] N. Viswanathan and C. C. Chu, "FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local refinement and a Hybrid Net Model", In *Proceedings of the ACM/IEEE ISPD*, 26-33, 2004.

[19] Y. Chou and Y. Lin, "A Performance-driven Standard-Cell Placer Based on a Modified Force-Directed Algorithm", In *Proceedings of the ACM/IEEE ISPD*, 24-29, 2001.

[20] S. Iman, M. Pedram, C. Fabian, and J. Cong, "Finding uni-directional cuts based on physical partitioning and logic restructuring", *In Proceedings of the 4th ACM/IEEE* Physical Design Workshop, 187-198, 1993.

[21] C. Hwang and M. Pedram, "PMP: Performance-driven multilevel partitioning by aggregating the preferred signal directions of I/O conduits", In *Proceedings of the ACM/IEEE ASP-DAC*, 428-432, 2005.

[22] A. B. Kahng, I. L. Markov and S. Reda, "Boosting: Min-Cut Placement with Improved Signal Delay," In *Proceedings of the IEEE DATE*, 1098-1103, 2004.


TABLE 3. Timing-driven results of PSDP for six industry circuits with comparison to QuadP and Capo-boost.

| Benchmark circuits | Clock cycle | QuadP (wirelength-driven mode) | | | QuadP (timing-driven mode) | | | Capo-boost | | | PSDP (timing-driven mode) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | HPWL | RWL | WNS | HPWL | RWL | WNS | HPWL | RWL | WNS | HPWL | RWL | WNS |
| indust1 | 6.60 | 350134 | 461533 | -1.23 | 358551 | 465394 | -1.22 | 354437 | 472033 | -1.85 | 357728 | 479551 | -0.89 |
| indust2 | 15.50 | 1573453 | 2754704 | -4.31 | 1567428 | 2810432 | -3.81 | 1638655 | 2866492 | -3.52 | 1606993 | 2906574 | -3.17 |
| matrix | 3.89 | 104695 | 116987 | -2.2 | 107921 | 120481 | -2.06 | 105133 | 115670 | -2.04 | 111958 | 122867 | -2.01 |
| vp2 | 4.57 | 370677 | 450872 | -3.02 | 377096 | 453074 | -3.21 | 364578 | 482548 | -3.21 | 381118 | 489366 | -2.95 |
| mac1 | 3.85 | 443460 | 506880 | -0.56 | 444704 | 509136 | -0.49 | 476643 | 523736 | -0.41 | 481045 | 524894 | -0.30 |
| mac2 | 7.67 | 2247603 | 3244264 | -14.46 | 2249426 | 3297112 | -3.63 | 2354646 | 2948992 | -1.01 | 2408205 | 3123254 | -3.73 |
| **Ratio** | | **1.00** | **1.00** | **1.00** | **1.01** | **1.01** | **0.83** | **1.03** | **1.01** | **0.85** | **1.05** | **1.04** | **0.69** |