OWARU: Free Space-Aware Timing-Driven Incremental Placement

Jinwook Jung†, Gi-Joon Nam‡, Lakshmi Reddy‡, Iris Hui-Ru Jiang§, and Youngsoo Shin†
†Dept. of EE, KAIST, Daejeon 34141, Korea
‡IBM Research, Yorktown Heights, NY 10598, United States
§Dept. of EE, National Chiao Tung University, Hsinchu 30010, Taiwan
jinwookjung@kaist.ac.kr, {gnam, reddyl}@us.ibm.com, huiru.jiang@gmail.com, youngsoo@ee.kaist.ac.kr

ABSTRACT

This paper proposes a powerful new technique called "OWARU"[1] that re-places and re-sizes multiple gates simultaneously to improve the most critical paths of a design. In essence, it is an incremental timing-driven placement technique integrated with gate sizing optimization that runs in conjunction with static timing analysis to guarantee a WYSIWYG [2] property. The OWARU technique offers several key advantages over previous techniques such as geometrical path straightening via the Bézier-curve algorithm, free space awareness to guarantee a legal placement solution, and an accurate true timing mode. The Bézier-curve geometric smoothing algorithm is extended with new *anchor* placement techniques to further improve the path placement. Free space aware placement algorithm is further enhanced with multiple gate optimization. The preliminary results are promising. We applied the OWARU technique at the end of industrial strength physical synthesis optimization on high performance microprocessor designs. The technique was extremely effective in improving the most critical path of the tested designs. On timing critical paths that were not fully closed from the previous physical synthesis optimization, the WS (worst slack) is improved by 5.3% of the total clock period and the TNS (total negative slack) improved by 91.3% on average.

1. INTRODUCTION

Timing closure in VLSI design is an optimization process to meet the target timing performance while satisfying geometric constraints of a design. In modern design flows, this concept is extended to *design closure* by including power, thermal and manufacturability constraints that become more important as technology continues to shrink. Nonetheless, *timing* is still the primary objective (and constraint) that determines the quality of a design. Timing closure itself consists of several optimizations such as placement, routing, sizing transistors or gates, buffer insertion and sometimes logic re-structuring. All these so-called physical synthesis optimization problems have an abundant research history of their own, but the ultimate goal of physical synthesis optimization has not changed: *"how can the timing of a design be improved by massaging the given netlist?"*. In this context, a slew of timing-driven optimization techniques have been introduced. Timing-driven placement and timing-driven routing are two good examples.

Despite these efforts, modern timing closure tools hardly achieve first-pass timing closure [1]. It commonly takes several iterations of logical and physical optimization interleaved with manual efforts to gradually (and slowly) converge to a fully timing closed solution. These numerous iterations of physical synthesis optimization are one of the most significant problems in modern IC design and layout flows. This is dubbed as a "TAT" (Turn-Around-Time) issue. There are numerous causes for failure of first-pass timing closure, but one of the biggest culprits is the loose integration of (crude) timing models with optimization. As noted in [2], there is a long debate regarding whether most timing-driven optimization algorithms in the literature should be labeled as *timing-influenced* rather than *timing-driven* techniques. This argument is particularly true in timing-driven placement research. Due to the complexity of computation, most of so-called *timing-driven* placements deploy a crude timing model. The notion of net weighting is a good example. It prioritizes nets based on timing criticality and then a placement algorithm minimizes the weighted sum of total wire length of all nets. Despite its simplicity, net-weight based timing driven placement is still one of the most popular global placement methods deployed in modern timing closure flows [3].

Meanwhile, there has been a great deal of focus on incremental timing-driven placement techniques with a higher degree of timing accuracy. The favored approach is to formulate the placement problem via linear programming (LP) with timing constraints embedded [4, 5]. Due to the scalability of linear programming formulation, however, incremental placement (as opposed to global placement) is formulated as a linear programming problem. Moreover, LP is flexible to have different objective functions (worst slack or total negative slack or the combination of them) with different timing models. While LP is effective for finding the ideal locations of gates according to the timing model employed, placement legalization is still required afterward, which often introduces quite significant perturbation to the previous linear programming solution.

In recent years, the incremental approach has been used to integrate placement optimization with various physical optimizations [6] such as gate sizing [2], V_t assignment, or even buffering [7]. Placement optimization alone on the physically optimized netlist incurs immediate timing degradation due to wire and sink load changes. In the integrated approach, subsequent physical optimization can quickly recover the timing degradation caused from the placement changes by adjusting the size of gates, V_t types or even buffer in-

[1]The word OWARU is a Japanese meaning "finish" or "close".
[2]What You See Is What You Get.

ICCAD '16, November 07–10, 2016, Austin, TX, USA
 ISBN 978-1-4503-4466-1/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967062

sertion or removal. These additional optimizations are accompanied by a placement legalization step. Among all the incremental timing-driven optimizations, only [2] considered a discrete set of pre-legalized locations to avoid an explicit placement legalization.

In this paper, we introduce a powerful new path-based incremental timing-driven placement technique, "OWARU", that can be applied at the late stage of timing closure flow. Its primary goal is to improve the timing of the most critical paths of a design. OWARU is integrated with gate sizing optimization that runs in conjunction with accurate static timing analysis. The OWARU technique offers several key advantages over previous techniques:

1. **Bézier-curve based geometric algorithm.** OWARU effectively straightens crooked critical paths for timing improvement via Bézier curve smoothing.
2. **Anchor placement for path segment optimization.** OWARU identifies a set of critical common gates (anchors) on multiple paths and places them first via a quadratic placement algorithm. These gates serve as anchors for subsequent critical path segment optimization.
3. **Integrated optimization of placement with other physical synthesis techniques.** Gate sizing, buffering, and V_t assignment can be easily integrated into the OWARU framework.
4. **Free space aware discrete optimization.** OWARU only considers a finite set of pre-legalized candidate locations to avoid subsequent placement legalization. The candidate location selection algorithm can factor in any integrated physical optimizations such as gate sizing, buffering, or V_t assignment.
5. **Flexible timing mode.** Each optimization move is evaluated by an explicit timing analysis. In conjunction with free space-awareness, a WYSIWYG property is guaranteed. Virtually any degree of timing accuracy can be employed into this technique.
6. **Multiple gate optimization.** OWARU is further extended to multiple gate optimization on the same critical path while guaranteeing a WYSIWYG property.

We applied OWARU to a suite of high performance microprocessor designs after a full strength conventional industrial physical synthesis flow. Hence, the inputs to OWARU can be considered *hard-to-improve* critical paths. Nonetheless, OWARU was effective in improving the timing of those paths further. On average, the WS (worst slack) improved by 5.3% of the target clock period and the TNS (total negative slack) improved by 91.3% over those of input netlist.

The rest of the paper is organized as follows. Section 2 describes preliminary concepts for the paper. Section 3 presents a simple free space-aware timing-driven placement algorithm in a single gate usage model. We give the details of the OWARU algorithm for multiple gate optimization in Section 4. Section 5 presents experimental results, and Section 6 provides some concluding remarks.

2. PRELIMINARIES

2.1 Circuit Timing Measurement

We use two measures to represent the timing of a circuit, the worst slack (WS) and total negative slack (TNS). In static timing analysis, a circuit is modeled as a timing graph $G = (V, E)$, where each vertex $v \in V$ corresponds to a gate, and each edge $e(u,v) \in E$ represents the connection between a pair of gates $u, v \in V$. Each vertex v is associated with timing slack $SLK(v)$, which is given by

$$SLK(v) = RAT(v) - AT(v), \tag{1}$$

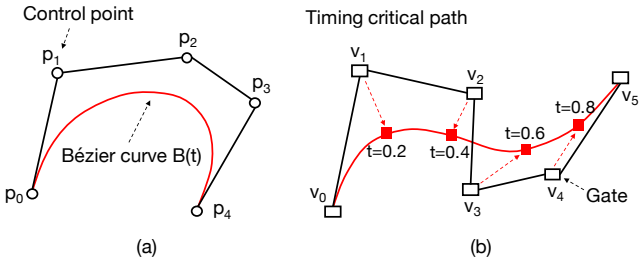

Figure 1: (a) An example of a Bézier curve and (b) critical path smoothing using a Bézier curve.

where $RAT(v)$ and $AT(v)$ are the required arrival time and the arrival time of the vertex v. The timing of a circuit can be represented by the worst slack (WS), which is the most critical path timing of a design. Given a slack threshold SLK_{th} that is the target slack that a design has to meet, $WS(G)$ is defined as:

$$WS(G) = \min_{v \in V} (\min (SLK(v), SLK_{th})). \tag{2}$$

The overall timing of a design can be represented by the total negative slack (TNS):

$$TNS(G) = \sum_{v \in V} \min (SLK(v) - SLK_{th},\ 0). \tag{3}$$

The TNS is the sum of the differences between SLK_{th} and the slack of a gate v that is only smaller than SLK_{th}. The SLK_{th} is normally set to 0. However, it is not uncommon to set a positive SLK_{th} value due to increasing process variations.

2.2 Bézier Curve Smoothing

A Bézier curve is a parametric curve defined by a set of control points. It is widely adopted in the computer graphics field to model smooth curves. Given a set of $n+1$ control points, p_0 through p_n, the corresponding Bézier curve $\mathcal{B}(t)$ is defined as:

$$\mathcal{B}(t) = \sum_{i=0}^{n} p_i\, b_{i,n}(t), \quad 0 \le t \le 1 \tag{4}$$

where $b_{i,n}(t)$ is the Bernstein polynomial of degree n [8]. Fig. 1 (a) shows an example of a Bézier curve. The curve begins at the first control point p_0 and arrives at the last control point p_4; these two control points serve as the end points of the curve. The remaining intermediate control points act as guidance for the direction the Bézier curve heads toward.

A Bézier curve is completely contained within the convex hull of its control points. In addition, Bézier curves have a variation diminishing property, meaning that Bézier curves are smoother than the polylines formed by their control points. These two properties guarantee that the lengths of Bézier curves are always shorter than the sum of the control polyline lengths. Therefore, it is an ideal method for smoothing crooked critical paths [9]. An example is shown in Fig. 1 (b), where the cells $v_0, \cdots, v_5$ on a timing violating path are regarded as the control points of a Bézier curve. The goals of critical path smoothing are 1) to minimize the wire lengths on a critical path, and 2) to evenly distribute the gates along the path [6]. To achieve these, the ideal smooth path produced by a Bézier curve is evenly sampled, and these samples become target points that corresponding gates need to move toward. For instance, the optimal location b_2 on the ideal smooth path for the gate v_2 is obtained by setting t in Eq. (4) to 0.4.

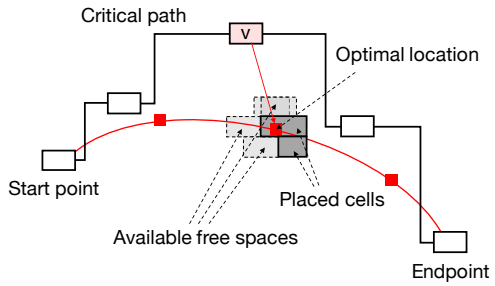

Figure 2: Free-space awareness in critical path smoothing.

2.3 Free Space Awareness within Placement

The literature contains many techniques for incremental timing-driven placement [2–7,10]. Yet, most methods require a subsequent legalization step to guarantee a legal placement solution. This problem is depicted in the example shown in Fig. 2, where a new location for the critical gate v is calculated, but falls into a location occupied by another cell. The resulting placement incurs cell overlap and a subsequent legalization step is required to resolve the cell overlaps. This results in a critical mismatch between the timing seen during the optimization and the timing after the legalization. In [10], several upstream optimizations had to be undone ("Do-No-Harm" property) due to this phenomenon. Free space aware timing-driven placement is important because the mismatch between expected timing improvement and realized timing can be avoided. Free space-aware incremental timing-driven placement considers only a set of locations that guarantee a legal placement solution. The expected timing value during the optimization is the exact final timing realized, which is called a WYSIWYG property. When a placement technique is free space aware, it can be applied even at the later stages of the physical synthesis flow because it avoids any unexpected timing degradation.

3. FREE SPACE-AWARE CRITICAL PATH SMOOTHING

In this section, we state the free space-aware critical path smoothing problem in more detail. This algorithm serves as a foundation for the OWARU algorithm presented in Section 4.

3.1 Problem Statement

Of particular interest is the timing improvement of a placed circuit at the end of a physical synthesis flow for timing closure. We state the free-space aware critical path smoothing problem as follows: Given the netlist of a timing optimized circuit and its timing graph $G = (V,E)$, a set of free spaces $L = \{l_1, \cdots, l_n\}$, where each $l_i \in L$ denotes an empty placement site where a cell can be placed, find new legal locations of gates in a subset of gates from V (i.e., gates on timing critical paths) such that $WS(G)$ and $TNS(G)$ are improved.

3.2 An Algorithm

Fig. 3 shows the free space-aware critical path smoothing algorithm. It is an iterative incremental path-based timing-driven placement technique. At each iteration, it identifies the k most timing critical paths, and improves their timing in the order of their timing criticality while considering only legal locations for each gate. The iterations stop when WS or TNS cannot be improved further, or the number of iterations reaches the given maximum iteration threshold. The optimization for each path is as follows:

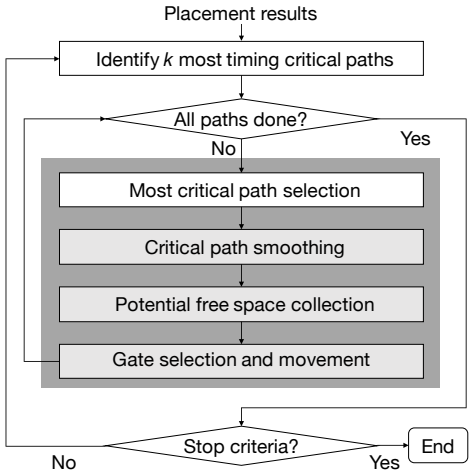

Figure 3: Free space-aware critical path smoothing.

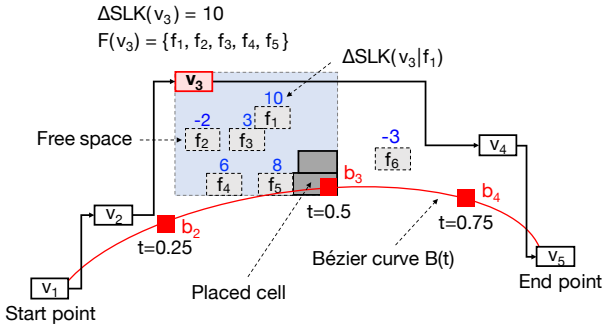

Figure 4: Illustration of the free space-aware critical path smoothing.

Critical path smoothing. For each target critical path, a Bézier curve is used to create a smooth path. As mentioned, Bézier curves have several desirable properties such as the variation diminishing property, a convex hull property leading to a shorter wire length and low computational complexity. In addition, the gates can be easily distributed evenly along the path by properly selecting the parameter t of the Bézier curve. The sampled points on the curve serve as ideal locations for the gates on the critical path. Note that we do not need to compute the entire Bézier curve for each path; only the sampled points (e.g., b_2, b_3 and b_4 in Fig. 4) need to be computed.

Potential free space collection. A free space f is a tuple of consecutive empty placement sites whose size is bigger than the width of the gate v_i. After the ideal location b_i for each gate v_i is identified, a set of the *available free space*s near the ideal location $F_b(v)$ is constructed. Several different algorithms can be used to construct $F_b(v)$ for each gate. In our implementation, we only consider available free spaces within the bounding box of v_i (i.e., the current location) and its corresponding ideal location b_i for each gate. For instance, the available free spaces of v_3 in Fig. 4 are f_1, f_2, f_3, f_4 and f_5[3].

[3]Although free spaces may have placement sites in common, for simplicity, Fig. 4 illustrates that potential free spaces do not share any placement site with each other.

Each available free space f_j of the gate v_i is annotated with the *potential slack benefit*, $\Delta SLK\left(v_i \mid f_j\right)$, which is defined as

$$\Delta SLK\left(v_i \mid f_j\right) = SLK'\left(v_i \mid f_j\right) - SLK(v_i), \tag{5}$$

where $SLK'\left(v_i \mid f_j\right)$ is the expected slack of the gate v_i at the free space f_j. The expected slacks of all the potential free spaces collected are computed using explicit static timing analysis calls. Only the free spaces that have positive potential slack benefit (i.e., timing improvement) are included in the set of *potential free spaces* for v_i. Free spaces for the gate v_i are denoted by $F'_b(v_i)$. In the example of Fig. 4, $F'_b(v_3)$ is $\{f_1, f_3, f_4, f_5\}$.

Gate selection and movement. Given a set of potential free spaces with positive slack benefits, the *maximal slack benefit* and the corresponding best free space of a gate v_i are defined as:

$$\Delta SLK\left(v_i\right) = \max_{f_j \in F'_b(v_i)} \Delta SLK\left(v_i \mid f_j\right), \tag{6}$$

$$f_{best}\left(v_i\right) = \operatorname*{argmax}_{f_j \in F'_b(v_i)} \Delta SLK\left(v_i \mid f_j\right). \tag{7}$$

Among all the gates in the critical path, the gate v_i with the most $\Delta SLK\left(v_i\right)$ is selected, and finally replaced at $f_{best}\left(v_i\right)$. For the example in Fig. 4, the gate v_3 has the most ΔSLK; it is moved to its best free space $f_{best}(v_3) = f_1$, and the slack of the path is improved by 10. Since only the set of free spaces that can accommodate a target gate are considered, subsequent legalization is not required. Moreover, the slack benefit is calculated via an explicit static timing call. Thus, the slack improvement after the optimization is exactly the same slack benefit that is calculated during the construction of $F'_b(v_i)$ for each v_i gate. Note that this algorithm is a "single-gate-at-a-time" technique. In Section 4.5, this algorithm will be enhanced to replace multiple gates simultaneously.

4. THE OWARU ALGORITHM

OWARU consists of several enhancements on top of the algorithm in Fig. 3. In this section, each of those enhanced techniques is presented in details.

4.1 Slack-based Path Segmentation

The previous free space-aware critical path smoothing technique optimizes one path at a time. When a subset of paths is shared by multiple critical paths, the Bézier curve smoothing of each critical path can lead to multiple ideal locations for gates on the sub-paths that are shared by multiple critical paths. Fig. 5 (a) illustrates the situation where paths P_1 and P_2 share part of the path from v_0 to v_2. If we apply Bézier curve smoothing for P_1 and P_2 separately, the gates v_1 and v_2 will have multiple ideal locations for P_1 and P_2. This prohibits effective timing improvement of path P_1 and P_2, since the gates v_1 and v_2 may experience oscillating movements during different iterations.

To uniquely assign a target ideal location for each gate, we decompose timing critical paths using slack-based path segmentation [9]; then Bézier curve smoothing is applied to each path segment. Given a path P, a maximal subset of P where every edge is consecutive and has the same timing slack is defined as a path segment $P(v_i, v_j)$, which starts from gate v_i and ends at gate v_j. Fig. 5 (b) shows an example of the path segmentation. Path P_1, the most timing critical path in this example, results in the path segment $P(v_0, v_4)$; it is the same as the original path since every edge has identical slack. The path P_2 is decomposed into two path segments; $P(v_0, v_2) = (v_0, v_1, v_2)$ and $P(v_2, v_6) = (v_2, v_5, v_6)$. Since $P(v_0, v_2)$ is completely contained in $P(v_0, v_4)$, the resulting set of path segments becomes $\mathbf{P} = \{P(v_0, v_4), P(v_2, v_6)\}$.

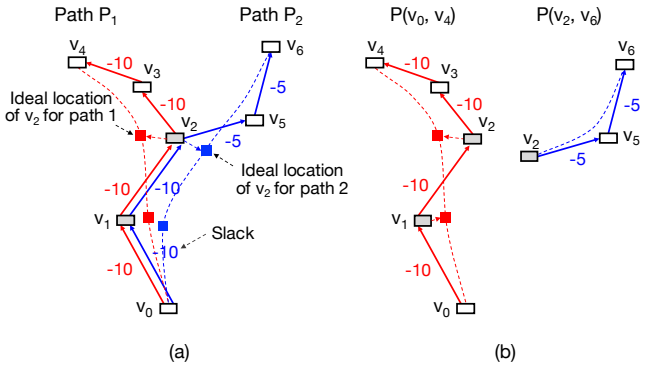

Figure 5: Slack based path segmentation. (a) Two critical paths have some gates and nets in common before the segmentation. (b) The paths are decomposed into edge-disjoint path segments.

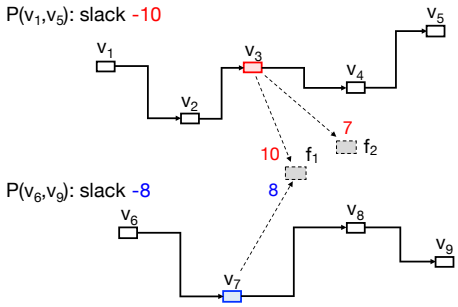

Figure 6: Free space competition problem.

4.2 Free Space Assignment for Multiple Gates

The algorithm in Section 3 just picks the gate with the most maximal slack benefit ΔSLK for each critical path, and moves it to the corresponding free space f_{best}. While it ensures timing improvement of each critical path, the overall amount of timing improvement of the entire design, however, depends on the order by which paths are optimized. Consider the example in Fig. 6. Gate v_3 on path segment $P(v_1, v_5)$ is moved to f_1 by the free space-aware critical path smoothing algorithm with slack improvement of 10 and the slack of path segment $P(v_1, v_5)$ becomes 0. Then, for $P(v_6, v_9)$, the only free space available for gate v_7 is already taken by $P(v_1, v_5)$, thus the worst slack of the circuit still stays -8. However, if f_1 is allocated to v_7 and f_2 is allocated to v_3, respectively, the worst slack of the design can be improved to -3 while the overall slack benefit becomes 15 in this example.

To eliminate the order dependency of path segment optimizations (thus leading to better slack benefits), we introduce a bipartite graph $B = (V_l, F_r, M)$, where V_l and F_r are two disjoint vertex sets (that is, V_l and F_r are each independent sets) such that every edge in M, $m(v_i, f_j)$, connects a vertex $v_i \in V_l$ to a $f_j \in F_r$. An example bipartite graph formulation is shown in Fig. 7. Each vertex v_i in V_l corresponds to the gate having the most ΔSLK for each critical path segment; F_r consists of free spaces, each of which is a potential free space of at least one gate in V_l. An edge $m(v_i, f_j) \in M$ is constructed if f_j is one of the potential free spaces for the gate v_i. Each edge is associated with the cost $c_{i,j} = \Delta SLK\left(v_i | f_j\right)$, the slack benefit when the gate v_i is replaced at the free space f_j.

Via the bipartite graph formulation, we reduce the free space assignment to a maximum cost bipartite matching problem. The max-

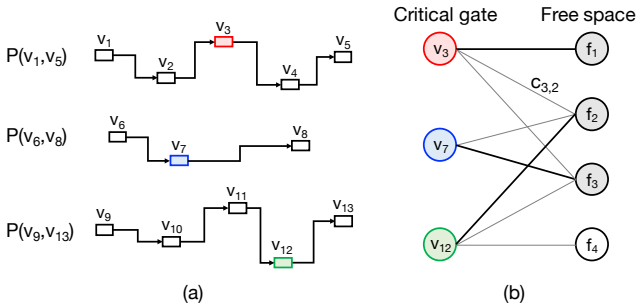

Figure 7: (a) A set of path segments. (b) Bipartite graph formulation for free space assignment.

imum cost bipartite matching problem can be solved in polynomial time using the Hungarian method [11]. The free space assignment is determined by the matching result on B. While it is possible that multiple gates from different path segments compete for the same free space, the bipartie matching solution guarantees that the overall slack benefits of those critical path segments are maximized.

4.3 Anchor Placement

Let us consider the critical path optimization problem in Fig. 8. It consists of two path segments, $P(v_1,v_5)$ and $P(v_2,v_8)$. Via bipartie graph matching, the free space f_1 is assigned to gate v_2 for $P(v_1,v_5)$ while free space f_2 is assigned to gate v_6 for $P(v_2,v_8)$. After commitment of those replacements, we expect the slack improvement of 10 for v_6 as the pre-calculated slack benefit of v_6 at f_2 is 10. However, the Bézier curve of the path segment $P(v_2,v_8)$ and the expected slack benefit of v_6 are both computed with the assumption that v_2 is fixed at its original location. As v_2 is moved to f_1, the pre-computed ideal smooth path of $P(v_2,v_8)$ is no longer a valid smooth path, nor is the pre-calculated slack benefit for v_6 at f_2 correct. The move of v_6 to f_2 may adversely degrade the timing of the path segment $P(v_2,v_8)$ because of the move of v_2 to f_1, which is not reflected in the $P(v_2,v_8)$ optimization. This problem occurs because the previous bipartie formulation cannot optimize multiple path segments simultaneously to maximize the overall slack benefits.

To avoid this situation, we introduce the concept of *movable anchors*. We define the start or the end point of a path segment[4] as an *anchor*, which in turn corresponds to the first or the last control point of the corresponding Bézier curve. If an anchor is contained in another path segment, but is not an anchor for the segment, it is defined as a *movable* anchor (e.g., v_2 in Fig. 8). Otherwise the anchor is defined as a *fixed* anchor (e.g., v_1, v_5 and v_8 in Fig. 8).

The goal is to find a placement solution for movable anchors, which can ultimately guide path straightening to generate geometrically straight paths. To achieve this, we identify *movable anchors* first, and place and fix them prior to Bézier-curve path smoothing for each path segment.

An example of anchor-driven path smoothing is shown in Fig. 9. In the example, we are given four path segments and their initial placement (Fig. 9 (a)). An ideal pre-placement of the movable anchors, shown in Fig. 9 (b), allows for straight paths for all path segments after several steps of Bézier curve smoothing.

In our implementation, we formulate the *anchor placement* problem as a quadratic placement. Given a set of path segments **P**, we

[4]Note that the start and end points of path segments are not necessarily latches or flip flops.

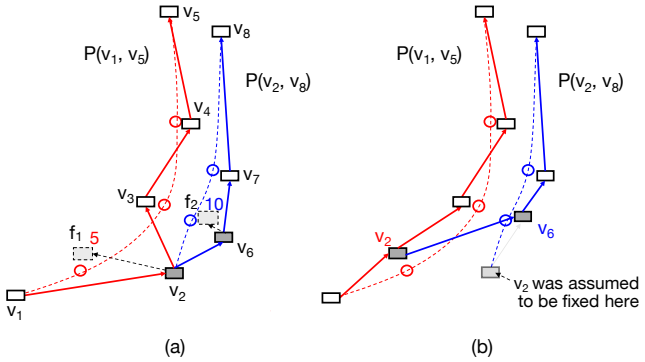

Figure 8: Anchor placement problem. (a) Initial placement before critical path smoothing. (b) Anchor of the segment $P(v_2,v_6)$ assumed to be fixed is displaced to a new location f_1.

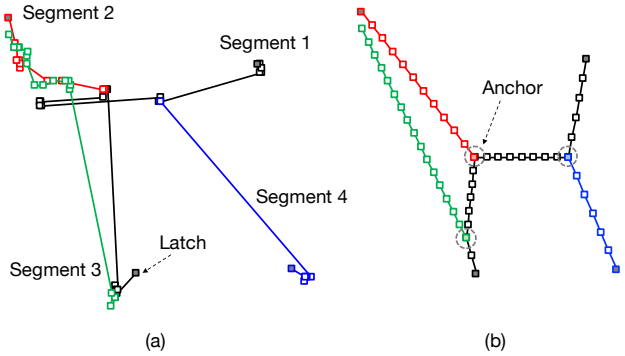

Figure 9: Ideal movable anchor placement. (a) An initial placement. (b) Geometrically ideal anchor placement.

first identify the movable and fixed anchors; the degree of every movable anchor is 3 or higher, so they can be identified from **P** by simply checking the number of incident edges for each gate. The fixed anchors are either the start or the end gate of path segments. Then, the placement graph $G' = (V', E')$ is built where V' only consists of the movable and fixed anchors. An edge is created between a fixed anchor v_i and a movable anchor v_j if they belong to the same path segment. Each edge $e(v_i,v_j)$ is associated with a weight $w_{i,j}$, which is defined as

$$w(v_i,v_j) = \frac{|SLK\left(P(v_i,v_j)\right) - SLK_{th}|}{|P(v_i,v_j)|+1}, \tag{8}$$

where $SLK(P(i,j))$ is the slack of $P(v_i,v_j)$ and $|P(v_i,v_j)|$ denotes the number of non-anchor movable gates in $P(v_i,v_j)$. The numerator of Eq. (8) represents how large the negative slack of the path segment is, and the denominator is the number of wires the path segment has. The weight $w(v_i,v_j)$ has higher value if the path is more critical. However, if more movable gates exist on $P(v_i,v_j)$, the criticality of the segment is diluted by these gates.

Then, we solve the minimization of weighted quadratic wirelengths on G':

$$\text{Minimize} \sum_{e(v_i,v_j)\in E} w(v_i,v_j) \times \left((x_i-x_j)^2 + (y_i-y_j)^2\right), \tag{9}$$

where x_i and y_i (x_j and y_j) are the x and y coordinates of v_i (v_j),

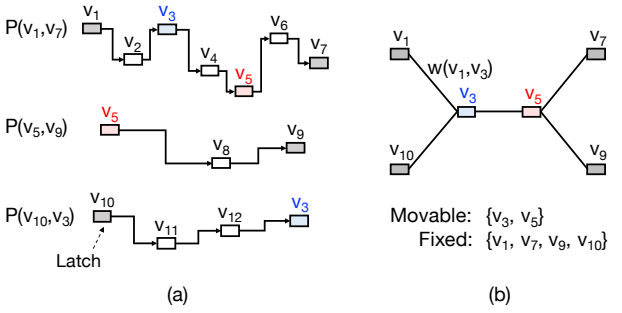

Figure 10: Placement graph formulation for anchor placement. (a) A set of given path segments. (b) Placement graph.

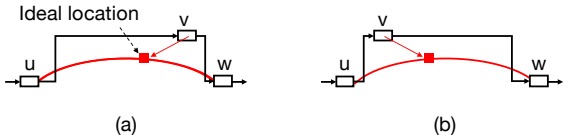

Figure 11: Load capacitance of (a) the gate v, or (b) the driver u increases after critical path smoothing.

respectively.

Fig. 10 shows an example of the anchor placement formulation. Given three path segments, the movable anchors, v_3 and v_5, are identified first. The placement graph G' is built using the set of movable anchors $\{v_3, v_5\}$ as movable objects, and the set of fixed anchors $\{v_1, v_7, v_9, v_10\}$ as fixed objects. Edges are created and associated weights are calculated by Eq. (8). The quadratic placement is then formulated, and the ideal locations of movable anchors are determined by solving Eq. (9).

The best free space is chosen for each movable anchor by the free space-aware critical path smoothing technique from Section 3. We replace the anchors to the best free spaces chosen and they are considered as fixed points. New path segments are constructed with anchor placement and Bézier curve based path smoothing is applied to the new set of path segments.

4.4 Gate Sizing

Bézier curve based path smoothing is an incremental placement technique. Whenever there is a placement change, the sizes of the replaced gate and its neighbor gates from prior gate sizing optimization become stale. Let us consider an example shown in Fig. 11 (a), where the ideal location of the gate v is identified. Moving v to its ideal location reduces the wire length between v and its driver u, while the load capacitance of v is adversely increased. Similarly, in the example of Fig. 11 (b), v's load capacitance after critical path smoothing decreases, while the wire load of its driver increases. In both cases, the sizes of v and u need to be re-optimized.

We integrate gate sizing optimization right after critical path smoothing in a *free space-aware* manner. Only gate sizes that the current free space can accommodate are considered as candidate sizing options, to avoid incurring additional legalization steps. By performing gate sizing simultaneously with critical path smoothing, further slack improvement (and sometimes area savings) can be achieved.

To this end, we integrate gate sizing into the OWARU algorithm. Recall the potential free space collection stage in Section 3, where

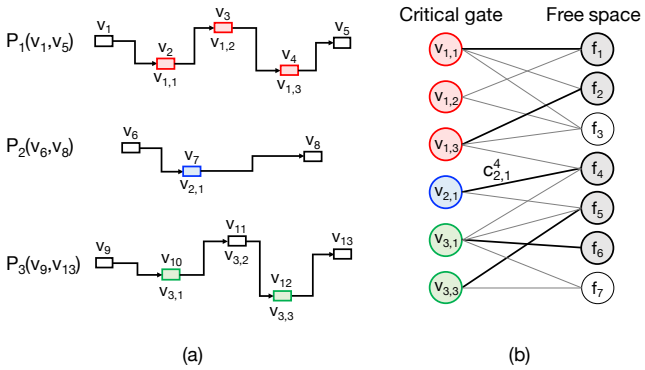

Figure 12: (a) Given critical path segments. (b) Multiple gate extension of the bipartite graph formulation.

the potential slack benefit is computed for each free space f collected for a gate v. Prior to computing the potential slack benefit, we now calculate the change in the wire load of v when it is moved to f. If the wire length of v is increased after movement to f, we also consider a new sizing candidate of v, which is denoted as $s(v|f)$. Among the sizing candidates that can be accommodated into f, the one with the most slack benefit is selected as $s(v|f)$. Similarly, if the load capacitance of u is increased after the movement of v, a viable sizing option for u is found in a similar manner. If this sizing option can be accommodated into the current location of u, it is denoted as $s_{v|f}(u)$. The potential slack benefit $\Delta SLK(v|f)$ is now computed using the newly selected sizing candidates of the gates v and u.

Every available free space f of a gate v is now associated with a tuple $\left(\Delta SLK(v|f), s(v|f), s_{v|f}(u)\right)$. After free space assignment, we examine the associated sizing candidates for each allocated free space. If the current size of a gate is different from the associated sizing candidate, it is bound to the new gate size after movement to the newly assigned free space by the bipartie matching algorithm.

4.5 Multiple-Gate Optimization

Although multiple path segments are optimized simultaneously, note that the OWARU algorithm described thus far only replaces a single gate per path segment per iteration. To more efficiently improve the timing of a circuit, we extend OWARU to optimize multiple gates per segment at each iteration. First, each path segment is assigned a unique index in the ascending order of its timing slack. In addition, each gate v in the i-th path segment is now assigned two indices i and j, and denoted by $v_{i,j}$; the second index j is obtained by numbering vertices in the path segment sequentially from the start point as shown in Fig. 12 (a). Three path segments are given in the example where the path segment with the worst slack $P_1(v_1, v_5)$ is assigned the smallest index.

The formulation of the bipartite graph $B = (V_l, F_r, M)$ presented in Section 4.2 is extended with all the gates in a path segment. The set of left vertices V_l now contains every gate $v_{i,j}$ which has at least one available free space f_k such that $\Delta SLK(v_{i,j}|f_k)$ is larger than zero. Such gates are represented by the shaded boxes in Fig. 12 (a), and the vertices in the left partition of the bipartite graph shown in Fig. 12 (b). Correspondingly, all the available free spaces are included in the right partition F_r, and edges are created between the gate and the free spaces following the method described in Section 4.2. The cost of an edge between gate $v_{i,j}$ and free space f_k is now denoted as $c^k_{i,j}$, which is set to $\Delta SLK\left(v_{i,j}|f_k\right)$.

The free space assignment problem for multiple gates is now formulated as an integer linear programming (ILP) problem:

$$\text{Maximize} \quad \sum_{v_{i,j} \in V_l} \sum_{f_k \in F_r} c^k_{i,j} x^k_{i,j}, \tag{10}$$

$$\text{Subject to} \quad x^k_{i,j} \in \{0,1\}, \qquad v_{i,j} \in V_l,\ f_k \in F_r \tag{11}$$

$$\sum_{v_{i,j} \in V} x^k_{i,j} \leq 1, \qquad \forall f_k \in F_r \tag{12}$$

$$\sum_{f_k \in F_r} x^k_{i,j} + \sum_{f_k \in F_r} x^k_{i,j-1} \leq 1, \qquad \forall v_{i,j} \in V_l \tag{13}$$

$$\sum_{f_k \in F_r} x^k_{i,j} + \sum_{f_k \in F_r} x^k_{i,j+1} \leq 1. \qquad \forall v_{i,j} \in V_l \tag{14}$$

Eq. (10) to Eq. (12) formulates the maximum cost bipartite matching problem for gates. Note that the potential slack benefit of a gate is computed with the assumption that its neighboring gates are fixed. To preserve this property, the constraints of Eq. (13) and Eq. (14) ensure that the matching result does not include the consecutive movements of neighboring gates.

Instead of directly solving the time-consuming ILP problem, we adopt a post-processing heuristic in which the maximum cost bipartite matching is solved on B, and then the matching results violating the constraints of Eq. (13) and/or Eq. (14) are filtered out. The matching results between gates and free spaces are sorted in descending order of the cost of the edge (i.e., the maximal slack benefit), and then in ascending order of the indices of the path segments. The free space assignment results are committed from top to bottom, in decreasing order of slack benefits. If two consecutive gates are committed (i.e., the violation of the constraints of of Eq. (13) and/or Eq. (14) in ILP), the second move is rejected. In other words, with this heuristic, only up to half of gates are repositioned per path segment.

4.6 Overall Algorithm

Fig. 13 gives a sketch of the entire OWARU algorithm. It takes a parameter k, the number of critical paths optimized per iteration, as well as *iteration_limit*, the number of maximum iterations. OWARU first finds the k most critical paths, and decomposes them by slack-based path segmentation (L3). The movable and fixed anchors are identified from the set of path segments $\mathbf{P}$. The anchor placement graph $G' = (V', E')$ is constructed, and the anchor placement problem is solved via quadratic placement (L4–L5). The algorithm moves to Bézier curve smoothing and bipartite graph formulation (L6–L13). For each gate v of a path segment, the ideal location on the Bézier curve and the available free spaces $F_b(v)$ are found(L7–L9). The potential slack benefit $\Delta SLK(v|f)$ for each available free space f is computed. If the slack benefit is positive, the edge between v and f is added to the bipartite graph B (L11–L13). After all the critical path segments are investigated, the maximum weight bipartite matching is performed on B (L14). Each gate is moved and re-sized based on the matching result (L15). This optimization is repeated until no further timing improvement is made or the iteration limit is reached.

5. EXPERIMENTS

The OWARU algorithm is implemented in C++. For the anchor placement, a quadratic placement using a conjugate gradient solver is implemented. The Hungarian method [11] is implemented for the maximum cost bipartite matching algorithm used for free space assignment. To measure accurate timing values during the optimization, an industrial static timing analyzer with sign-off accuracy is used.

Input: Circuit $G = (V, E)$, number of paths k, *iteration_limit*
Output: Circuit G after critical path smoothing

```
L1:  V_l ← ∅, F_r ← ∅, M ← ∅
L2:  repeat for iteration_limit
L3:      P ← critical_path_segments(G,k)
L4:      Build anchor placement graph G' = (V',E')
L5:      Solve anchor placement on G'
L6:      foreach path segment P(v_i,v_j) ∈ P do
L7:          foreach gate v ∈ P(v_i,v_j) do
L8:              b ← Ideal location of v on Bézier curve of P(v_i,v_j)
L9:              F_b(v) ← Available free spaces of v
L11:             foreach free space f ∈ F_b(v) do
L10:                 Compute ΔSLK(v|f) using a timer
L12:                 if ΔSLK(v|f) > 0 then
L13:                     V ← V_l ∪ v, F_r ← F_r ∪ f, M ← M ∪ m(v,f)
L14:     Do maximum cost bipartite matching on B = (V_l,F_r,M)
L15:     Move and re-size gates based on matching result
L16:     if no timing improvement then
L17:         Restore best solution found and exit
```

Figure 13: Pseudo code of the proposed OWARU algorithm.

Table 1: Timing improvement

		Worst slack			TNS		
Design	#Gates	Before	After	Diff (ps)	Before	After	Diff (%)
C1	109k	6.6	10.0	3.4	−143	0	100.0
C2	153k	5.1	10.0	4.9	−288	0	100.0
C3	244k	−3.5	8.5	12.0	−512	−8	98.4
C4	316k	−8.4	8.6	17.0	−685	−5	99.2
C5	360k	−9.3	1.4	10.6	−1222	−502	58.9
Average				9.6			91.3

For the evaluation, several 14nm circuits from commercial high-performance microprocessor designs are used. The number of gates for each design is listed in the first column of Table 1. The target clock period and the slack threshold SLK_{th} for all designs are set to 180ps and 10ps, respectively. The OWARU algorithm is invoked at the end of the full-strength timing closure flow. The number of iterations and the number of path segments to optimize per iteration, *iteration_limit* and k in Fig. 13, are set to 50 and 10, respectively.

Table 1 presents the evaluation results. The second and third columns show the worst slack of each design before and after the OWARU algorithm. On average, the worst slack improved by 9.6ps, which is about 5% of the target clock period (180ps) of the designs. Considering that the inputs of the algorithm are *hard-to-improve* critical paths that were not fully closed by previous full-strength physical optimizations, we view the additional timing improvement by OWARU as significant. In addition, the total negative slack TNS, which represents the overall timing quality of designs, is noticeably improved by 91.3%. Two designs, C1 and C2 in Table 1, are fully timing-closed by OWARU while for the rest of the designs, the worst slacks become positive. For C3, C4 and C5, TNS is still negative. This is because the slack threshold SLK_{th} is set to 10ps.

To demonstrate the impact of individual techniques presented in Section 4, we took the circuit C4, and applied OWARU enabling only a subset of the techniques. First, OWARU with only free space-aware critical path smoothing described in Section 3 is applied and the timing improvement is measured per iteration

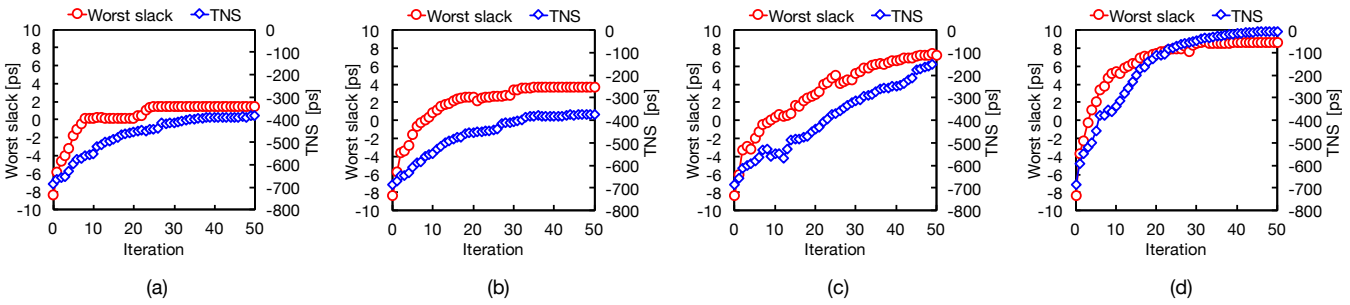

Figure 14: Improvement of WS and TNS during the iterative flow. (a) Result of the baseline free space-aware critical path smoothing (described in Section 3). Results obtained by enabling (b) free space allocation and anchor placement, (c) gate sizing, and (d) multiple-gate optimization.

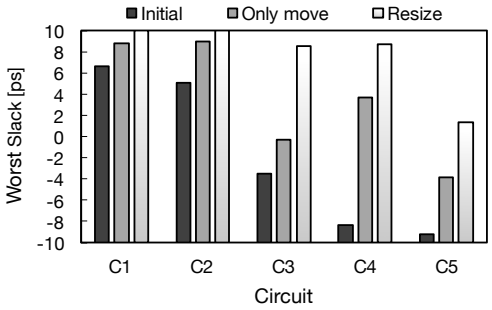

Figure 15: Comparison of the worst slack improvement with and without gate sizing.

(Fig. 14 (a)). The initial worst slack and TNS of circuit C4 is -8.4ps and -685ps with the slack threshold of 10ps. After 50 iterations of primitive free space-aware critical path smoothing, WS and TNS improved by 9.8ps and 43%. Fig. 14 (b) shows the timing improvement per iteration when the anchor placement is applied on top of free space-aware critical path smoothing. Although the TNS is not significantly improved (10ps), the WS is further improved by 2.3ps compared to the result of Fig. 14 (a). The TNS improvement from the initial design becomes 12.1ps. Fig. 14 (c) shows the timing improvement with additional gate sizing optimization described in Section 4.4. By integrating the gate sizing into timing-driven placement, we achieved further WS improvement of 3.6ps. The TNS improvement is even more significant, 32% compared to Fig. 14 (b). The effectiveness of the multiple-gate optimization is demonstrated in Fig. 14 (d). We observe that the multiple-gate optimization makes the algorithm converge faster while improving the timing even more. The additional WS improvement is 8.6ps, which results in the overall WS improvement of 17ps compared to the initial design WS. In addition, TNS of the design is now -5.3ps.

Finally, to emphasize the importance of the simultaneous gate sizing with timing-driven placement, Fig. 15 shows the WS comparison before and after OWARU without and with the gate sizing for all circuits. Without the gate sizing (i.e., only with path straightening placement technique), WS improved by 5.3ps, on average. By incorporating gate sizing within path straightening placement, the average WS improvement becomes 9.6ps, 81% larger than the improvement without gate sizing. These results demonstrate the effectiveness and importance of the co-optimization of gate sizing and incremental timing-driven placement.

6. CONCLUSION

In this paper, we present a path-based incremental timing-driven placement algorithm called OWARU. The OWARU algorithm adopts Bézier curve smoothing to straighten out meandering timing critical paths, integrates with gate sizing optimization, and runs in conjunction with an accurate static timing analyzer. In addition, it provides a free space aware capability to guarantee a legal placement solution even without an explicit legalization step. All these features make OWARU an ideal incremental timing-driven optimization technique that can be applied at the later stages of the physical synthesis flow. On a set of high performance microprocessor designs, OWARU demonstrated its effectiveness by improving the worst slack by 5.3% of the target clock period and the total negative slack by 91.3%, on average.

The OWARU algorithm can also be integrated with other physical optimization techniques such as buffering and V_t assignment. This remains as future work. Other promising research directions include blockage-aware geometrical critical path smoothing, and a multi-threaded implementation for faster turn-around time.

References

[1] C. J. Alpert, S. K. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. C. Yildiz, "Techniques for Fast Physical Synthesis," *Proc. of the IEEE*, vol. 95, no. 3, pp. 573–599, Mar. 2007.

[2] M. D. Moffitt, D. A. Papa, Z. Li, and C. J. Alpert, "Path Smoothing via Discrete Optimization," in *Proc. Design Automation Conf.*, 2008, pp. 724–727.

[3] T. F. Chan, J. Cong, and E. Radke, "A Rigorous Framework for Convergent Net Weighting Schemes in Timing-Dirven Placement," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2009, pp. 288–294.

[4] Q. Wang, J. Lillis, and S. Sanyal, "An LP-based Methodology for Improved Timing-Driven Placement," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 2005, pp. 1139–1143.

[5] W. Choi and K. Bazargan, "Incremental Placement for Timing Optimization," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2003, pp. 463–466.

[6] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu, "ITOP: Integrating Timing Optimization within Placement," in *Proc. Int. Symp. on Physical Design*, Mar. 2010, pp. 83–90.

[7] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov, "RUMBLE: An Incremental, Timing-driven, Physical-synthesis Optimization Algorithm," in *Proc. Int. Symp. on Physical Design*, Apr. 2008, pp. 2–9.

[8] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[9] H.-Y. Chang, I. H.-R. Jiang, and Y.-W. Chang, "Timing ECO Optimization Via Bézier Curve Smoothing and Fixability Identification," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 12, pp. 1857–1866, 2012.

[10] H. Ren, D. Z. Pan, C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Hippocrates: First-Do-No-Harm Detailed Placement." in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 2007, pp. 141–146.

[11] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," in *Naval Research Logistics Quarterly*, Mar. 1955, pp. 83–97.