Drive Strength Aware Cell Movement Techniques for Timing Driven Placement

Guilherme Flach, Mateus Fogaça, Jucemar Monteiro, Marcelo Johann and Ricardo Reis
Universidade Federal do Rio Grande do Sul (UFRGS) - Instituto de Informática - PGMicro/PPGC
{gaflach, mpfogaca, jucemar.monteiro, johann, reis}@inf.ufrgs.br

ABSTRACT

As the interconnections dominate the circuit delay in nanometer technologies, placement plays a major role to achieve timing closure since it is a main step that defines the interconnection lengths. In initial stages of the physical design flow, the placement goal is to reduce the total wirelength, however total wirelength minimization only roughly addresses timing. A timing-driven placement incorporates timing information to remove or alleviate timing violations. In this work, we present an incremental Timing-Driven Placement (TDP) flow to further optimize timing violations via single-cell movements. For late violations, we developed techniques to reduce the load capacitance on critical nets and to obtain load capacitance balancing using drive strength. For early violations, we present techniques that rely on clock skew optimization, register swap and interconnection increase. Our flow is experimentally evaluated using the International Conference on Computer-Aided Design (ICCAD) 2015 Incremental Timing-Driven Contest infrastructure. Experimental results show that our flow can significantly reduce timing violations. On average, for long maximum displacement, the quality of results is improved by 67.8% with late Worst Negative Slack (WNS) and Total Negative Slack (TNS) being improved by 2.31% and 10.84%, respectively, early WNS and TNS improved by 68.92% and 76.42%, respectively and congestion metric Average Bin Utilization (ABU) improved by 74.9% compared to the 1st place in the contest. The impact on Steiner Tree Wirelength (STWL) is less than 2.5%.

Keywords

Microelectronics, EDA, Timing-Driven Placement, Timing Closure

ISPD'16, April 03 - 06, 2016, Santa Rosa, CA, USA

ISBN 978-1-4503-4039-7/16/04...$15.00

DOI: http://dx.doi.org/10.1145/2872334.2872359

1. INTRODUCTION

Timing-driven placement incorporates timing information to reduce timing violations which are only roughly addressed by total wirelength minimization. A detailed timing-driven placement works on a globally optimized and legalized solution trying to further improve timing while keeping the solution legalized. Typically a detailed placement will process only a fraction of the total cells of a design, preserving the global properties of the initial solution.

A pure placement flow can only improve timing by changing cell positions and hence the length of the wires that connect them. As the interconnection dominates the circuit delay in nanometer technologies, placement plays a major role to achieve timing closure.

Although the length minimization of a single wire typically improves the local delay, the minimization of total wirelength does not guarantee the best global delay. The reason is that the minimization of the sum of wirelength completely ignores the fact that some wires are more important than others in defining the circuit performance. Therefore a timing-driven placement prioritizes critical interconnections and typically trades-off increase on non-critical interconnection lengths with decrease on critical interconnection lengths.

In this work, we develop a detailed timing-driven flow that can incrementally improve the timing of a legalized placement solution. Our flow is composed by several single-cell movements aiming both early (hold) and late (setup) timing violations. For late violations, we apply load reduction and load balancing to accomplish the wirelength trade-offs. For early violations, we devise methods to increase delay by moving cells away and by skew optimization.

The main contributions of this work can be summarized as follows:

- a set of single-cell movement techniques to mitigate both early and late timing violations;
- load balancing via an analytical formulation to find the optimal position of a cell w.r.t. its driver and sink considering drive strengths;
- clock skew exploration to reduce early (hold) violations;
- a flow for incremental timing-driven placement validated with ICCAD 2015 contest infrastructure.

Our experimental results show that there is expressive room for mitigation of timing violations using incremental local search techniques. The Timing-Driven Placement

(TDP) flow was experimentally evaluated using an state-of-the-art set of circuits provided by International Conference on Computer-Aided Design (ICCAD) 2015 Contest.

This paper is organized as follows. Section 2 reviews the state-of-the-art algorithms for TDP optimization. Some definitions are provided on Section 3. The proposed algorithms are discussed in Sections 4, 5 and 6 while in Section 7 we show our incremental TDP flow. The experimental results are discussed in Section 8 and the concluding remarks are made in Section 9.

2. RELATED WORKS

Most of the timing-driven placement techniques are divided into 2 groups: net-based [9, 13, 3] and path-based approaches [11, 15, 16].

The former group prioritizes nets with timing violations by assigning them higher weights during global wirelength-driven placement or by assigning a max wirelength constraint. These techniques can deal with a lot of violations at the same time, keeping a global view of the problem. However, while these nets are optimized, other violations may show up and, thereby, the weights need to be updated and new constraints created. At the end, the problem may be over constrained, and the solution may be a local minima. Over constrained solutions also may lead to congestion and can affect routability.

On the other hand, path-based approaches focus on fixing a set of critical or near critical paths. The idea is to straighten the critical paths in order to reduce their length. The procedure can be done by heuristic local search or linear programing techniques.

ITOP [14] proposes various techniques in order to achieve timing closure. The first one is a netlist transformation in which virtual 2-pin nets are created linking cells in critical paths to raise attraction between them in global placement. Furthermore, an incremental path smoothing algorithm locally moves critical modules trying to achieve local improvements. Unlike most algorithms, after changing the solution, small movements are performed to mitigate congestion and to ensure routability. Finally, the authors combine other techniques, like buffering and sizing (repowering), to further improve the solution quality.

A set of local search algorithms was proposed by [2]. Their work rely on two strategies: path straightening and clustering. The goals of clustered movement are to speed up the execution time and to escape from suboptimal solutions. The idea is to minimize the euclidean distance between the most critical upstream and downstream pins of a cluster. A formulation using Lagrangian Relaxation to mitigate TDP timing violations was proposed by [5]. The proposed technique updates dynamically net's weights according to Lagrange multipliers.

3. DEFINITIONS

In this section, some concepts used throughout this paper are presented.

An *early timing violation* occurs whenever a signal propagates too fast reaching a registers before the previous signal had been captured. An early violation is also referred to as a hold violation. A *late timing violation* occurs whenever the signal takes too long to propagate reaching a register after the time-frame necessary to store the signal. A late timing violation is also referred to as a setup violation. In this work both early and late timing violations are handled.

3.1 Drive Strength

To obtain a measure of the drive strength of cells, more specifically of timing arcs, we compute a representative driver resistance R for each timing arc. The delay of a timing arc is estimated as in Equation (1),

$$d = RC + p \tag{1}$$

where C is the load capacitance being driven and p is the parasitic delay. The drive resistance is then computed via least square approximation of the delay values for several different loads.

3.2 Criticality and Centrality

The **criticality** $\{criticality \in \mathbb{R} \mid 0 \leq criticality \leq 1\}$ of a pin is the negative slack of the pin divided by the worst negative slack found in the design. The normalized **centrality** $\{centrality \in \mathbb{R} \mid 0 \leq centrality \leq 1\}$ of a pin is a rough measure of how many critical endpoints are affected by the pin. It can be seen as the importance of such pin to the Total Negative Slack (TNS).

The centralities are computed by traversing the design in the reverse topological order. By definition, the centrality at endpoints is set as the endpoint's criticality. The centrality of a driver pin is simply the sum of centralities of its sink pins. The centrality of the output (driver) pin is then proportionally distributed among the input pins of the respective cell according to the input pin criticalities. Centrality values can be seen as the endpoint criticalities flowing through the circuit, which is a standard technique used by timing driven optimization methods based on Lagrangian Relaxation [1] to obey the flow conservation as implied by the KKT optimally conditions.

4. LATE OPTIMIZATION

In this section, we present a set of techniques that targets to decrease wire load capacitance and resistance of the critical nets. We also propose an analytical formulation to explore driver strength in critical nets to reduce late violations. We devise an analytical formulation to obtain the position where the late timing violation is locally minimized.

4.1 Clustered Movement

As demonstrated by [2], moving one cell at a time may lead to suboptimal solutions. To avoid this problem we implemented an algorithm that performs a Breadth-First Search (BFS) finding topological neighbor cells with timing violations within a given range. For each cluster, we find an ideal position and shift all cells toward that position. These two operations are described below.

4.1.1 Cell Clustering

Algorithm 1 presents the proposed clustering algorithm. Two inputs must be specified: *initialCell* and *maxDistance*. The former refers to the cell with which the algorithm will begin the BFS and the latter is the maximum Manhattan distance a cell can be from *initialCell* in order to be clustered. At the beginning, the *cluster* is empty (Line 2) and the queue that controls the BFS, called *neighbors*, is initialized with *initialCell* (Line 3).

Lines 4-11 show the algorithm's main loop. In each iteration, the first element from the queue is stored in *current* variable and then removed (Line 5). If the current element is critical, i.e., has negative slack, it is clustered (Line 9) and its topological neighbors are added to the queue (Line 10). The loop continues until *neighbors queue* is empty (Line 11).

Algorithm 1: Cell clustering

```
   input : initialCell, maxDistance
   output: cluster
1  begin
2      cluster = ∅
3      neighbors.push( initialCell )
4      repeat
5          current ← neighbors.pop()
6          if minSlack ≥ 0 or
7          dist( current, intialCell ) > maxDistance then
8              continue
9          cluster.insert( current )
10         neighbors.pushAll( current.neighbors() )
11     until neighbors.empty();
```

4.1.2 Clustered Movement

Once a cluster is formed, we must decide in which direction to move it. In this work, we opted to shift the cluster towards the center of mass of critical neighbor pins weighted by their negative slack, as shown in Figure 1.

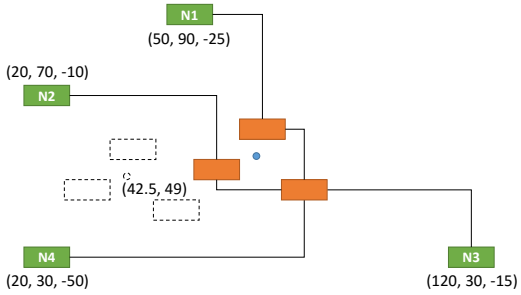


Figure 1: The target position for a cluster according to Equation (2). For each neighbor node N a tuple (x, y, slack) is specified.

So, the target position of cluster cells is computed as:

$$target\_pos(cluster) = \frac{\sum_{i=1}^{n} pos(N_i) \times slack(N_i)}{\sum_{i=1}^{n} slack(N_i)} \quad (2)$$

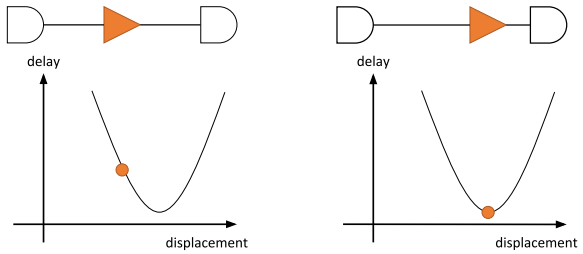
where n is the number of critical neighbor pins of the cluster, $pos(N_i)$ and $slack(N_i)$ are the position and the slack of the pin associated to the neighbor cell i. Then, for each cell in the cluster we obtain a new position as:

$$new\_pos(cell) = pos(cell) + [target\_pos(cluster) - center(cluster)] \quad (3)$$

where $pos(cell)$ is the current cell position, $target\_pos(cluster)$ is the cluster center target position computed in Equation (2) and $center(cluster)$ is the current central position of the cluster.

4.2 Buffer Balancing

After buffer insertion, the circuit may contain several buffer chains. However placement is not always aware of the different drive strengths of cells that compose the chain, which may degrade timing. The general idea of buffer balancing is shown in Figure 2 where the delay of the path segment is reduced if the buffer is placed closer to its sink.



(a) Initial (b) Optimized

Figure 2: Buffer balancing technique finds the buffer position that minimizes the segment path delay.

To find the displacement where the delay is minimum, an analytical formula is devised. This formula takes into account the cell strengths assuming that the interconnection is modeled as an RC tree and its delay is computed via Elmore delay [4]. We assume the buffer's driver and its sink are fixed while the buffer can freely move between them. Moreover the driver is assumed to drive only the buffer and the buffer to drive only one sink. This idea can be applied iteratively so that buffer chains with arbitrary number of buffers can be handled. In our experiments, only a few iterations are necessary to align all the buffers in the design.

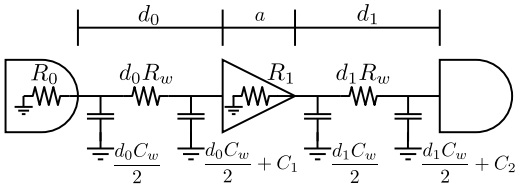


Figure 3: Buffer balancing modeling.

Figure 3 shows a single buffer chain, whose delay, D can be described as in Equation (4) using the Elmore delay model.

$$D = R_0 (C_1 + d_0C_w) + d_0R_w \left(C1 + \frac{d_0C_w}{2}\right) + p_0 + R_1 (C_2 + d_1C_w) + d_1R_w \left(C2 + \frac{d_1C_w}{2}\right) + p_1 \quad (4)$$

where R_0 is the resistance of the buffer's driver, C_1 is the input pin load capacitance of the buffer, d_0 is the wirelength from driver to buffer, R_w is the wire resistance per unit-length, C_w is the wire capacitance per unit-length, R_1 is the buffer resistance, d_1 is the wire length from the buffer to its sink, C_2 is the load capacitance on the input sink pin, p_0 and p_1 are parasitic delays of the driver and buffer, respectively.

Considering that $d = d_0 + a + d_1$ where a is the distance of input and output buffer pins, the minimum delay is obtained by setting $\frac{\partial D}{\partial d_0} = 0$ as described by Equation (5), which for practical purposes is clamped in the range $[0, d]$.

$$d_0 = \frac{C_w\left(R_1 - R_0\right) + R_w\left[C_2 - C_1 + C_w\left(d - a\right)\right]}{2C_wR_w} \quad (5)$$

Equation (6) defines the optimal position of the buffer w.r.t. its driver. As Manhattan routing is used, this may lead to multiple optimal positions. However by placing the buffer on the straight line connecting the driver and sink may help straightening the path which is a very common way to improve delay. Therefore, the buffer is placed on the straight line by setting its position to

$$P_b = P_d + \frac{d_0}{d} \times \left(P_s - P_d\right) \quad (6)$$

where P_b is the new buffer position, P_d is the driver position and P_s is the sink position.

4.3 Cell Balancing

In this section, we extend the formulation of buffer balancing to handle more general cases, i.e, non-buffers cells with multiple input pins and multiple sinks. To do so, we first compute the cell position for each timing arc individually and then combine the results to obtain the best cell position.

We restrict the region of a cell movement between the point it connects to its driving tree, here called driver point, and the point it connects to its sink tree, sink point, as shown in Figure 4.

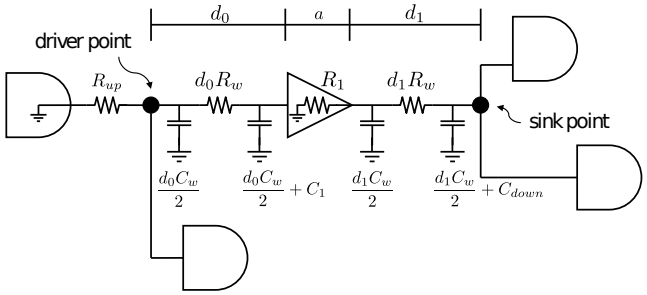


Figure 4: Cell balancing modeling.

Let R_{up} be the upstream resistance of the driver point (i.e. the sum of the resistance from the driver point up to the root of the tree, which includes the driver resistance). Let D_{up} be the delay at the driver point when the branch from the driver point to the cell is removed. Let C_{down} be the downstream capacitance of the sink point excluding any capacitance added by the branch connecting the cell to the sink point (i.e. sum of all capacitances from the sink point down to all leaf points including pin capacitances). Then the delay, D, from the driver cell and the sink point is given by Equation (7)

$$D = D_0 + D_1 \quad (7)$$

where

$$\begin{aligned} D_0 =& D_{up} + R_{up}\left(C_1 + C_wd_0\right) \\ &+ d_0R_w\left(C_1 + \frac{d_0C_w}{2}\right) + p_0 \end{aligned} \quad (8)$$

is the delay from the driver cell to the input of the current cell and

$$D_1 = R_1\left[C_{down} + d_1C_w\right] + d_1R_w\left[C_{down} + \frac{d_1C_w}{2}\right] + p_1 \quad (9)$$

is the delay from the current cell to the sink point, C_1 is cell input pin capacitance, d_0 is the wirelength between the driver point and the cell, d_1 is wirelength from the cell to the sink point, R_1 is the cell resistance and p_0 and p_1 are the driver and cell parasitic delay, respectively.

To a reason that will be apparent later D_0 and D_1 are weighted by w_0 and w_1 respectively so that the weighted delay is given by Equation (10).

$$D = w_0D_0 + w_1D_1 \quad (10)$$

Considering that $d = d_0 + a + d_1$ where a is the distance of input and output cell pins, the minimum delay is obtained by setting $\frac{\partial D}{\partial d_0} = 0$ as described by Equation (11) which for practical purposes is also clamped in the range $[0, d]$.

$$\begin{aligned} d_0 =& \frac{w_1C_wR_1 - w_0R_wC_1 + w_1R_w\left[C_w(d-a) + C_{down}\right)]}{R_wC_w(w_0 + w_1)} \\ &- \frac{w_0R_{up}C_w}{R_wC_w(w_0 + w_1)} \end{aligned} \quad (11)$$

Note that Equation (11) reduces to Equation (5) for a single buffer chain. The final position is obtained in the same way as in the buffer alignment technique.

Since we may have several target positions, one for each input pin, they are combined by their weighted average. Where the weight of each position is the importance of the input pin, which is set to $2 \times centrality + criticality$ in this work.

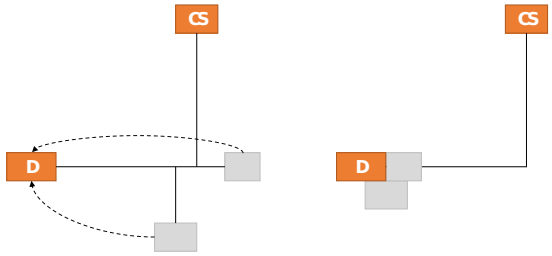
The reason to weight the partial delays is due to the effect on the delay of side cells. By minimizing the delay of a tuple driver-cell-sink we may degrade the delay of other cells nearby. For instance, if the critical sink of the driver is not the cell we are handling and if the cell moves away from the driver it will probably increase the delay on the critical cell due to the increased load capacitance. Here we use the driver's output pin importance as w_0 and the cell's output pin importance as w_1. Note that if the driver is more critical than the sink, the cell will likely get close to the driver, reducing its load capacitance and hence improving its delay.

4.4 Load Optimization

For critical nets with more than two cells, the sink cells with no late violations (i.e. positive slack) are moved closer to their driver cells in order to improve timing as shown in Figure 5. The main idea behind this approach is to reduce the interconnection load capacitance of critical nets and therefore improve the delay of the driver cell. Note that, by moving non-critical sinks closer to the root of the tree, besides reducing the total tree capacitance, we also are reducing the cumulative impact of the sink capacitance in the downstream nodes of the routing tree. Since the sinks moved are non-critical, the paths passing through them are likely to not generate new violations.

Nevertheless the movements are accepted only if they actually are likely to reduce timing violations in critical nets and do not cause timing violation in the sink cells. Otherwise, non-critical sink cells are kept in their initial position.

To accomplish that, after routing trees are re-built, the timing is updated locally.



(a) Initial (b) Optimized

Figure 5: Load Reduction applied to a critical net. Non-critical sinks (lighter cells) are moved closer to their driver cell (D).

5. EARLY OPTIMIZATION

In this section, techniques for early violation mitigation during the placement are also presented. We present four algorithms targeted to minimize early violations. The proposed algorithms explore wire load capacitance and resistance of the critical nets and useful clock skew to minimize early timing violations.

Let us consider a timing path between two registers. The register at the beginning of the path is called input register and the register at the end, output register. The early slack in a register-to-register path is defined by Equation (12)

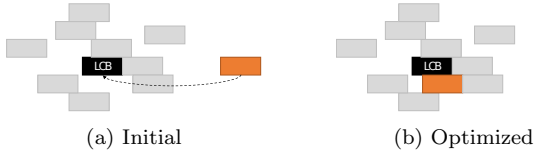
$$\begin{aligned} slack_D^{early} &= at_D^{early} - rat_D^{early} \\ slack_D^{early} &= l_i^{early} + d_{path}^{early} - l_o^{late} - t_{hold} \end{aligned} \quad (12)$$

where at_D^{early} and rat_D^{early} are the early arrival and required time respectively at the data input pin of the output register, l_i^{early} and l_o^{late} are the early and late clock latency at the clock pin of input and output registers respectively, d_{path}^{early} is the early delay among the registers and t_{hold} is the hold time of the output register.

According to Equation (12), the early slack can be improved by (i) increasing the path delay, (ii) increasing the clock latency at the input register, (iii) decreasing the clock latency on the output register and (iv) decreasing hold time. In this work, hold time is considered constant. The difference among the clock latencies is called clock skew.

5.1 Skew Optimization

The early slack can be improved by decreasing the clock latency on the output register. One way to achieve that is by moving the register closer to the clock source (e.g. a local clock buffer) as depicted in Figure 6.



(a) Initial (b) Optimized

Figure 6: Clock skew optimization by moving registers closer to LCBs.

Although the latency on the moved register is typically reduced, there might be side effects as latency changes on other registers and on other data path delay. Also a register can be both the start and end point of different paths. So a reduction on the latency may improve the slack on the incoming path, but may worsen the slack on the outgoing path. However, our experimental results showed that this technique is effective to improve early slack, on average.

5.2 Iterative Spreading

The iterative cell spreading tentatively moves all cells with early timing violation to north, south, east and west. The displacement from current position is set initially to 10% of a maximum displacement. If a better position is not found, the search area is gradually increased. The cost of a position is calculated updating timing locally and checking if the arrival time in the involved pins have increased.

5.3 Register Swap

Register swap tries to avoid the side effect on clock latency present in clock skew optimization (Section 5.1). Assuming that the registers are all the same (e.g. same size, V_{th}), by swapping the registers driven by a same clock source, the clock tree and its timing characteristics will not change. Hence the latency on each tree endpoint can be seen as constant.

The register swap is modeled as an assignment problem similar to [6], which can be optimally solved in polynomial time by the Hungarian algorithm [10]. The current register positions are seen as the slots to where the register should be assigned as illustrated by Figure 7. The goal is to minimize the total cost of the assignment.

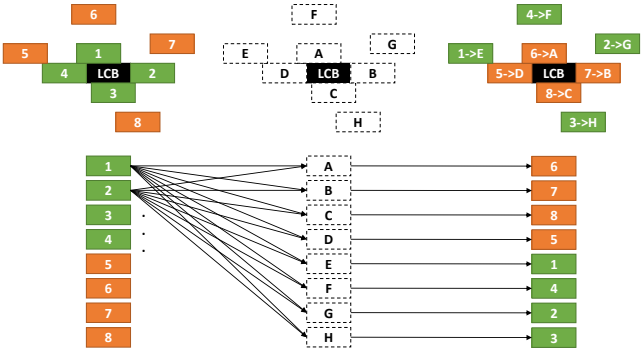


Figure 7: Register swap by optimal assignment.

The cost to assign a register i to a slot k is set as in Equation (13) where $criticality_D^{early}$ and $criticality_{CK}^{early}$ are the criticality of the data and clock pins of the register, respectively. The maximum displacement constraint can be modeled by setting an infinity cost whenever an assignment violates the maximum allowed displacement.

$$cost(i,k) = l^{late} criticality_D^{early} - l^{early} criticality_{CK}^{early} \quad (13)$$

The idea behind this cost function is as follows. When the register acts as the output register (path ends at the data pin), according to Equation (12), its clock latency should be decreased to improve slack. In terms of assignment cost, a larger latency should imply a larger cost ($+l^{late} criticality_D^{early}$). Similarly, when the register acts as the input register (path

starts at the clock pin), its clock latency should be increased. From an assignment cost point of view, a larger latency should imply a smaller cost ($-l^{early} criticality_{CK}^{early}$). The latencies are weighted by pin criticalities to optimize latency based on the influence of registers on timing violations.

5.4 Register-to-Register Path Fix

A common source of hold violations is a path connecting directly two registers, i.e. no combinational logical cells between them, as show Figure 8.

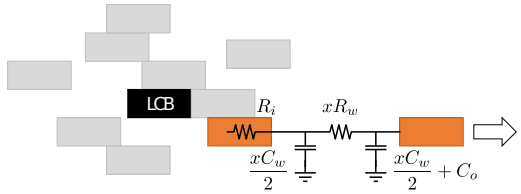


Figure 8: Register-to-register early violation path fix.

Besides skew optimization, early (hold) violations can be fixed by increasing the timing path delay. By setting the early slack to zero in Equation (12), the path delay that eliminates the violation is given by Equation (14).

$$d_{path}^{early} = l_o^{late} + t_{hold} - l_i^{early} \tag{14}$$

In the case of a direct path between registers, the timing path delay is simply composed by the input register delay plus the wire delay and it can be increased by moving the registers apart. Assuming that the cell delay is modeled via its driver resistance and the wire via Elmore delay, Equation (14) can be rewritten as in Equation (15) where x is the distance between the input and output registers and $K = l_o^{late} + t_{hold} - l_i^{early}$.

$$K = R_i \left(xC_w + C_o\right) + xR_w \left(\frac{xC_w}{2} + C_o\right) \tag{15}$$

Assuming that the latencies do not change as the registers are moved apart and that the hold time is also constant (i.e. K is constant), Equation (15) can be solved w.r.t. x as in Equation (16).

$$x = \frac{\sqrt{2C_wR_wK + {C_o}^2{R_w}^2 + {C_w}^2{R_i}^2} - C_oR_w - C_wR_i}{C_wR_w} \tag{16}$$

Once the optimum wirelength, x, is calculated, the input register is moved away from the output register following the straight line formed by the two registers.

6. ABU REDUCTION

We implemented an algorithm to minimize Average Bin Utilization (ABU) area violation. All cells with positive slack inside of bins with ABU violation are ranked. The cells with highest positive slack are moved first to the nearest bin with enough room. The source and target bin utilization are update incrementally after each cell movement. After each cell is moved, the routing trees are rebuilt and the timing updated locally. We only accept a cell movement if its estimated slack is still positive and the target bin utilization is lower than the maximum area utilization allowed. The process continues until all bins reach their target density or no more cell can be moved.

7. FLOW

The techniques presented in this work are combined in an incremental flow for timing-driven detailed placement, as shown in Figure 9. The diamond shape indicates that the steps are run until the quality of the result is not improved. The circle shape indicates that the quality of the result can degrade a certain number of times before exiting. The best solution found is restored.

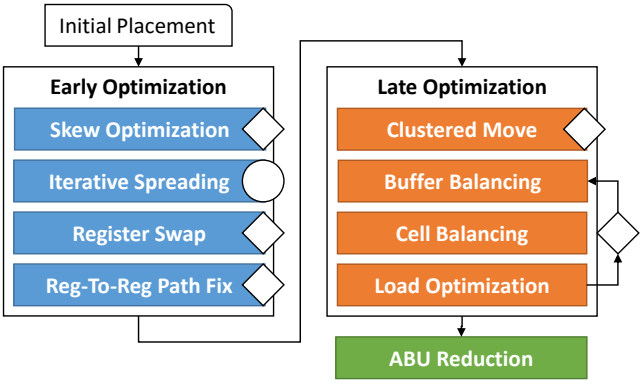


Figure 9: Our incremental timing-driven detailed placement flow.

The flow is divided into three main phases: early optimization, late optimization and ABU reduction. Early and late phases are divided into several steps.

In the early optimization phase, all early critical registers are tentatively moved closer to their local clock buffers in the Skew Optimization step. Registers are processed in increasing order of criticality. Next the Iterative Spreading is performed on all early critical cells. This is the only step allowed to degrade the quality of result. The rationale is that the clock skew has a large influence on early violations which may introduce a large noise in the estimated timing changes. Experiments show that it is better to allow degradation and to keep track of the best solution than to stop immediately when a degradation occurs. If the Iterative Spreading is not able to remove all the early violations, the Register Swap and Register-to-Register Path Fix are executed.

The late optimization phase begins by applying the Clustered Movement on top most late critical cells in increasing order of criticality. Once a cell is moved, it is not moved again in the same iteration. Next a loop composed of one iteration of Buffer Balancing, Cell Balancing and Load Optimization is executed. Buffer and Cell Balancing are applied for all late critical cells in increasing order of criticality. Load Optimization is performed for all non-critical sinks of critical nets. Cells are processed in increasing order of their driver cell's criticallity.

Finally, the ABU Reduction is performed to spread non-critical cells and hence to reduce the number of regions with high cell density.

7.1 Legalization

After a move is executed, the cell is likely to overlap other cells and a legalization needs to be performed. Current

designs have a lot of free space intentionally added to improve routablity and to make easier to perform incremental changes such as sizing and detailed placement. Therefore, to avoid disturbing the position of other cells, the legalization, in this work, is performed by searching for the nearest whitespace available around the intended target position. Our legalization relies on Jezz [12] legalizer.

7.2 Filtering out Bad Moves

A cell movement may cause the routing trees connected to the cell to change drastically and hence huge timing variation may occur, which misleads some optimization methods. To avoid timing degradation caused by such changes and also by legalization noise, local timing is evaluated and the move is optionally committed only if the local timing violation is reduced.

The degradation is computed as $\Delta cost$ where the cost is the sum of the weighted arrival times of the neighboring pins of the moved cell. In this work, the weight is set to $2 \times centrality + criticality$, which gives more importance to TNS-critical pins. This weighting function also helps to avoid focusing too much on Worst Negative Slack (WNS) improvement which may cause side effect degradation on TNS.

8. EXPERIMENTAL RESULTS

Our flow is implemented in C++11 and executed on an Intel® Core™ i7-4790K CPU @ 4.00GHz × 8 CPU with 32GB running Ubuntu 14.04 LTS (64-bit). Our flow relies on a built-in timer.

It is empirically validated using the ICCAD 2015 Incremental Timing-Driven Placement Contest infrastructure [7]. Eight mixed-size benchmarks are available ranging from approximately $700k$ to $2M$ elements. Initial placement solutions are provided for all benchmarks and they are optimized using two configurations for the maximum cell displacement: *short* and *long*. The quality of results is measured using *quality score* [8], which takes into account the early and late slack improvement and the ABU change.

Table 1 presents the benchmark characteristics and the results of our flow along with the initial solution and the results from the 1st place in the ICCAD 2015 Contest. The timing information, Steiner Tree Wirelength (STWL) and ABU are reported by the evaluation script provided by the contest organizers. Runtimes were measured in different machines and they are mostly shown for reference. However it is worth pointing out that our flow is about 4.6× faster, on average, than the 1st place.

For *long maximum cell displacement*, compared to the 1st place, our flow provides improved quality score results for all benchmarks. Considering metrics individually, with the exception of STWL, our flow also provides improved results in almost all cases. The exceptions are early violations in superblue7 and late violations in superblue5. Our flow is also able to achieve zero early violations in 5 out of 8 benchmarks.

For *short maximum cell displacement*, our flow provides improved quality score results on average. However results are mixed for both circuits and metrics. Results from the 1st place are particularly better on early WNS. This probably comes from the fact that they perform Local Clock Buffer (LCB) reallocation where registers can be connected to a different LCB. Note that for short maximum displacement the exploration space is reduced, which tends to flatten the improvements obtained among different approaches.

8.1 Move Gains

Table 2 shows the average impact of each move over all ICCAD 2015 benchmarks for short and long displacement. The results were obtained by applying only one pass of each technique directly to the initial solution. Note that the improvements of some moves may be better if they are combined with other moves. Also some moves may benefit more than others when multiple passes are performed.

As it can be seen the most effective method for early timing violation reduction is the Iterative Spreading. For late timing violation reduction, the Cell Balancing technique is the most effective one. The benefit of the Cell Balancing over other moves is particularly prominent in the short displacement.

Although very simple, the Load Reduction can achieve a significant improvement on quality score for this set of benchmarks. This comes at cost of increased density as measured by ABU, however such an increase is easily mitigated by the ABU Reduction step. The step ABU Reduction does not present quality score gains, in spite of its large gains in the ABU penalty as the ABU improvement works as a scaling factor in the quality score metric.

9. CONCLUSIONS

In this paper, we presented an incremental TDP flow composed by several single-cell move techniques able to reduce early and late timing violations.

Our results show that our drive strength aware load balancing technique is very effective to reduce late timing violations. Although load balancing can also be achieved via cell sizing, by finding a more balanced cell position during placement, one can reduce delay with no or minor impact on power. Moreover, our techniques for early slack improvement are able to remove violations for almost every benchmark.

Our TDP flow was empirically validated using the ICCAD 2015 contest infrastructure. It achieves, on average, best timing closure compared to the 1st place team on the contest particularly when cells are allowed to move greater distances.

10. ACKNOWLEDGMENT

This work is partially supported by Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) and by the National Council for Scientific and Technological Development (CNPq).

11. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] A. Bock, S. Held, N. Kämmerling, and U. Schorr. Local search algorithms for timing-driven placement under arbitrary delay models. In *DAC*, pages 29:1–29:6. ACM, 2015.

[3] M. Burstein and M. Youssef. Timing influenced layout design. In *DAC*, pages 124–130, June 1985.

Table 1: Experimental results of our incremental timing-driven placement flow on ICCAD 2015 contest benchmarks.

Benchmark Cells / Macros Max Disp.	Solution	Short								Long							
		ABU	StWL (μm) $\times 10^7$	Early (ps) WNS $\times 10^0$	Early (ps) TNS $\times 10^0$	Late (ps) WNS $\times 10^3$	Late (ps) TNS $\times 10^5$	Run-time (min)	Quality Score	ABU	StWL (μm) $\times 10^7$	Early (ps) WNS $\times 10^0$	Early (ps) TNS $\times 10^0$	Late (ps) WNS $\times 10^3$	Late (ps) TNS $\times 10^5$	Run-time (min)	Quality Score
superblue1	Initial	0.054	9.59	-9.34	-317	-4.98	-4.6	-	-	0.054	9.59	-9.34	-317	-4.98	-4.6	-	-
1.21M / 3787	1st Place	0.058	9.6	-3.83	-41.6	-4.67	-3.74	40.5	447.59	0.056	9.61	-16.7	-80.9	-4.57	-3.51	37.4	346.64
40μm / 400μm	Ours	0.008	9.72	-9.25	-50.6	-4.61	-3.82	6.5	391.65	0.011	9.9	-9.25	-36.7	-4.46	-3.41	5.4	508.41
superblue3	Initial	0.029	11.4	-78.4	-1460	-10.1	-15	-	-	0.029	11.4	-78.4	-1460	-10.1	-15	-	-
1.21M / 2074	1st Place	0.031	11.4	-65.7	-684	-9.44	-13.7	19.4	243.18	0.031	11.5	-13.1	-214	-8.71	-11.6	22.4	551.74
40μm / 400μm	Ours	0.007	11.5	-30.3	-434	-9.4	-13.4	7.8	351.05	0.008	11.6	-10.7	-91	-8.38	-9.33	7.6	755.54
superblue4	Initial	0.044	7.15	-12.6	-519	-6.22	-34.8	-	-	0.044	7.15	-12.6	-519	-6.22	-34.8	-	-
796k / 3471	1st Place	0.045	7.15	-6.08	-174	-5.94	-32	16.7	287.55	0.048	7.16	-12.3	-53.8	-5.76	-24.6	18.6	507.31
20μm / 400μm	Ours	0.032	7.21	-11.8	-145	-5.98	-31.2	5.2	276.19	0.040	7.55	0	0	-5.68	-23.6	5.7	665.55
superblue5	Initial	0.021	10.8	-36.8	-591	-25.7	-69.7	-	-	0.021	10.8	-36.8	-591	-25.7	-69.7	-	-
1.09M / 1872	1st Place	0.022	10.8	-36.8	-586	-25.1	-67.8	14.7	40.67	0.021	10.8	-36.8	-618	-24.3	-58.4	15.9	179.53
30μm / 400μm	Ours	0.000	10.8	-35.8	-291	-25.3	-67.2	6.3	149.11	0.000	10.9	0	0	-24.6	-59.9	3.2	469.52
superblue7	Initial	0.030	14	-7.65	-1990	-15.2	-18.6	-	-	0.030	14	-7.65	-1990	-15.2	-18.6	-	-
1.93M / 4910	1st Place	0.031	14	-6.75	-1940	-15.2	-17	42.8	98.48	0.031	14	-6.75	-1960	-15.2	-15.1	53.1	200.72
50μm / 500μm	Ours	0.007	14.2	-7.53	-1970	-15.2	-16.1	8.1	141.32	0.007	14.2	-7.53	-1970	-15.2	-13.8	7.8	264.42
superblue10	Initial	0.042	20.5	-8.62	-621	-16.5	-332	-	-	0.042	20.5	-8.62	-621	-16.5	-332	-	-
1.88M / 1696	1st Place	0.043	20.5	-8.62	-361	-16.2	-325	22.5	111.87	0.043	20.6	-5.15	-374	-16.1	-315	25.3	181.33
20μm / 500μm	Ours	0.020	20.6	-6.8	-383	-16.3	-319	11.8	142.67	0.012	21.1	0	0	-15.7	-280	9.3	492.91
superblue16	Initial	0.033	9.33	-10.7	-114	-4.58	-7.76	-	-	0.033	9.33	-10.7	-114	-4.58	-7.76	-	-
982k / 419	1st Place	0.041	9.36	-8.38	-30.7	-4.36	-5.14	22.8	524.72	0.040	9.37	-7.55	-37.6	-3.85	-2.66	32.0	894.76
30μm / 400μm	Ours	0.000	9.4	-0.548	-0.9	-4.34	-4.72	6.5	735.00	0.000	9.54	0	0	-3.46	-1.96	3.8	1,209.42
superblue18	Initial	0.040	5.77	-19	-283	-4.55	-10.3	-	-	0.040	5.77	-19	-283	-4.55	-10.3	-	-
768k / 653	1st Place	0.043	5.77	-3.81	-69.4	-4.12	-9.44	22.8	365.26	0.045	5.78	-1.95	-6.86	-3.82	-7.76	27.0	613.07
20μm / 400μm	Ours	0.013	5.83	-14.7	-81.9	-4.14	-9.48	4.5	302.28	0.010	5.9	0	0	-3.81	-6.45	3.6	780.90
Avg Change	Initial	-72.82	0.89	-26.41	-60.80	-4.43	-13.24	-	-	-72.92	2.64	-73.60	-85.36	-10.75	-32.90	-	-
(%)	1st Place	-74.00	0.79	45.25	-19.13	0.08	-2.31	-	48.55	-74.90	2.43	-68.92	-76.42	-2.31	-10.84	-	67.80

Table 2: Average impact of each technique over all ICCAD 2015 contest benchmarks for short and long maximum displacement.

Move	Goal	Short								Long							
		Quality Score	Run-time (s)	ABU	StWL	Early WNS	Early TNS	Late WNS	Late TNS	Quality Score	Run-time (s)	ABU	StWL	Early WNS	Early TNS	Late WNS	Late TNS
Iterative Spreading	Early	56.31	0.52	0.0%	0.4%	-6.7%	-24.8%	0.0%	0.0%	128.71	0.62	0.0%	0.5%	-26.2%	-51.3%	0.0%	0.0%
Clock Skew Opto	Early	43.32	0.03	0.0%	0.4%	5.9%	-24.7%	0.0%	0.0%	125.57	0.03	-0.1%	0.5%	-27.9%	-48.8%	0.0%	0.0%
Register Swap	Early	54.32	0.41	0.0%	0.5%	-10.3%	-22.0%	0.0%	0.0%	71.27	0.41	0.0%	0.5%	-14.1%	-28.6%	0.0%	0.0%
Reg-to-Reg Path Fix	Early	5.13	0.03	0.0%	0.4%	1.6%	-3.4%	0.0%	0.0%	29.86	0.04	0.0%	0.5%	0.6%	-15.2%	0.0%	0.0%
Clustered Movement	Late	15.90	9.29	0.1%	0.4%	0.0%	0.0%	-0.7%	-1.2%	52.87	9.33	0.1%	0.4%	0.0%	0.0%	-2.8%	-3.9%
Buffer Balancing	Late	44.26	0.58	0.8%	0.5%	0.0%	0.0%	-1.8%	-3.6%	89.32	0.81	0.0%	0.5%	0.0%	0.0%	-3.9%	-7.0%
Cell Balancing	Late	98.65	8.15	0.6%	0.5%	0.0%	0.0%	-3.2%	-8.2%	194.86	8.76	-0.2%	0.6%	0.0%	0.0%	-6.3%	-16.4%
Load Reduction	Late	42.65	7.60	2.1%	0.6%	0.0%	-0.1%	-0.9%	-3.8%	116.81	7.65	6.6%	1.8%	0.0%	-0.3%	-2.3%	-10.5%
ABU Reduction	ABU	-0.03	8.67	-52.4%	0.6%	0.0%	0.0%	0.0%	0.0%	-0.02	8.27	-58.5%	0.8%	0.0%	0.0%	0.0%	0.0%

[4] W. C. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics*, 19(1):55–63, Jan. 1948.

[5] C. Guth, V. Livramento, R. Netto, R. Fonseca, J. L. Güntzel, and L. Santos. Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression. In *ISPD*, pages 141–148. ACM, 2015.

[6] S. Held and U. Schorr. Post-routing latch optimization for timing closure. In *DAC*, pages 7:1–7:6. ACM, 2014.

[7] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *ICCAD*, pages 921–926, Nov 2015.

[8] M.-C. Kim, J. Huj, and N. Viswanathan. Iccad-2014 cad contest in incremental timing-driven placement and benchmark suite: Special session paper: Cad contest. In *ICCAD*, pages 361–366, Nov 2014.

[9] T. Kong. A novel net weighting algorithm for timing-driven placement. In *ICCAD*, pages 172–176, Nov 2002.

[10] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[11] D. Papa, T. Luo, M. Moffitt, C. Sze, Z. Li, G.-J. Nam, C. Alpert, and I. Markov. Rumble: An incremental timing-driven physical-synthesis optimization algorithm. *TCAD*, 27(12):2156–2168, Dec 2008.

[12] J. Puget, G. Flach, M. Johann, and R. Reis. Jezz: An effective legalization algorithm forminimum displacement. In *SBCCI*, Sept 2015.

[13] R.-S. Tsay and J. Koehl. An analytic net weighting approach for performance optimization in circuit placement. In *DAC*, pages 620–625, June 1991.

[14] 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 *ISPD*, pages 83–90. ACM, 2010.

[15] Q. B. Wang, J. Lillis, and S. Sanyal. An lp-based methodology for improved timing-driven placement. In *ASP-DAC*, volume 2, pages 1139–1143 Vol. 2, Jan 2005.

[16] C. S. William Swartz. Timing driven placement for large standard cell circuits. In *DAC*, pages 211–215, 1995.