Design Optimization by Fine-grained Interleaving of Local Netlist Transformations in Lagrangian Relaxation

Apostolos Stefanidis[†], Dimitrios Mangiras[†], Chrysostomos Nicopoulos[‡], David Chinnery[*], Giorgos Dimitrakopoulos[†]
[†]Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[‡]Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus
[*]Mentor, a Siemens Business, Fremont, USA

ABSTRACT

Design optimization modifies a netlist with the goal of satisfying the timing constraints at the minimum area and leakage power, without violating any slew or load capacitance constraints. Lagrangian relaxation (LR) based optimization has been established as a viable approach for this. We extend LR-based optimization by interleaving in each iteration techniques such as: gate and flip-flop sizing; buffering to fix late and early timing violations; pin swapping; and useful clock skew. Locally optimal decisions are made using LR-based cost functions, without the need for incremental timing updates. Sub-steps are applied in a balanced manner, accounting for the expected savings and any conflicting timing violations, maximizing the final quality of results under multiple process/operating corners with a reasonable runtime. Experimental results show that our approach achieves better timing, and both lower area and leakage power than the winner of the TAU 2019 contest, on those benchmarks.

CCS CONCEPTS

• **Hardware → Physical design (EDA)**; **Physical synthesis**.

KEYWORDS

Optimization; Lagrangian relaxation; sizing; buffering; useful skew

ACM Reference Format:
Apostolos Stefanidis, Dimitrios Mangiras, Chrysostomos Nicopoulos, David Chinnery, Giorgos Dimitrakopoulos. 2020. Design Optimization by Fine-grained Interleaving of Local Netlist Transformations in Lagrangian Relaxation. In *Proceedings of the 2020 International Symposium on Physical Design (ISPD '20), March 29-April 1, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3372780.3375566

1 INTRODUCTION

Gate-level optimization is needed through the design flow, to fix early and late timing violations, while reducing the area and the leakage power of the design[14][18]. To achieve this, there are various netlist transformations, such as: cell resizing and threshold voltage selection; buffer insertion for reducing delay when driving large capacitances, or for additional delay to slow down fast paths that violate early timing constraints; pin swapping; logic restructuring; and useful clock skew. Such netlist transformations are accompanied by several other steps that place and route the design. Gate-level optimization must be usable incrementally in various parts of the flow to achieve significant performance, power, and area improvements with a reasonable runtime.

Though the independent application of netlist transformations has offered improvements to the performance of designs, the transformations' combined application in the same LR optimization loop has not been explored. For the first time (to the best of our knowledge), we enclose all relevant netlist transformations inside the same LR-based timing-driven optimization.

We extend the well-known LR-based optimization formulation used for gate sizing [1], by interleaving in each iteration additional netlist transformations, including also register resizing. Each netlist transformation is smoothly integrated, without being disruptive to the overall optimization process. For instance, each resizing decision drives buffering additions in the same or following iterations, and each buffering addition guides the next sizing choices. In this way, buffers are added gradually and cell sizes adapt smoothly to it. The LR-based optimization orchestrates these heuristics with the goal of achieving global convergence.

In all cases, locally optimal decisions are taken using just the LR-based cost functions, without the need for incremental timing updates [8]. This increases the modularity of the proposed approach: any local netlist transformation that can be applied incrementally could be included in the set of available transformations.

To speed up the optimization, each transformation is applied to a critical subset of the design's cells. In each LR iteration, the cells are selected based on sensitivity criteria that identify the cells that are more promising to improve timing or reduce area/power.

This approach has been successfully applied to the TAU 2019 multi-corner design optimization contest benchmarks [2]. In all cases, our approach successfully optimizes the designs in all given corners and achieves (a) lower clock period, or (b) reduced area and leakage power at the same clock period as the contest's winner, without violating any slew or maximum capacitance constraints.

The contributions of this work are summarized as follows:

- The LR-based formulation is enhanced to handle the sizing of both registers and gates using the same cost function.
- Timing conflicts, where late and early timing violations lead to contradictory sizing decisions using LR-based cost function, are identified and handled appropriately.
- To reduce runtime, sizing is applied only to a subset of cells selected by sensitivity metrics. This is the first time that sensitivity based selection is combined with LR gate sizing.
- In each local iteration, we integrate LR-based buffering, useful skew, and logic restructuring, without the need for incremental timing updates per iteration.

ISPD '20, March 29-April 1, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7091-2/20/03…$15.00
https://doi.org/10.1145/3372780.3375566

The rest of the paper is organized as follows: Section 2 reviews state-of-the-art design optimization techniques. Section 3 presents the Lagrangian Relaxation formulation and the overall optimization flow. The proposed netlist transformations, the treatment of timing conflicts, and the sensitivity-based selection of the most critical cells are analyzed in Section 4. The experimental results are presented in Section 5, while conclusions are drawn in Section 6.

2 RELATED WORK

Early work on gate sizing focused on continuous transistor sizing [6]. To handle discrete gate sizes, Nguyen et al. [20] used posynomial models for timing-driven resizing and then used linear programming to assign delays to gates that would lead to maximum power savings. Chinnery et al. [3] improved this by accounting for the delay changes on the neighboring gates due to a resize, with accurate timing analysis using standard-cell library lookup tables. Held et al. [10] assigned slew targets, instead of delay targets. The above methods resize gates to the discrete version that achieves the closest slack-slew values to those optimally calculated. Fatemi et al. [5] presented sensitivities that can be used for timing, power, area, and slew optimization across multiple corners.

LR has been widely used for gate sizing. It was first used by Chen et al. [1] for continuous wire and gate sizing. Hu et al. [11] used LR to define optimal continuous gate sizes that they cluster to discrete sizes based on proximity to the optimal size, and try solutions from different clusters. Ozdal et al. [21] formulated the LR sub-problem to trade-off leakage power and fix timing violations, choosing gate sizes with dynamic programming (DP). Flach et al. [8] sized each gate to locally minimize the LR cost. This provided great leakage-power savings, with faster runtime than other approaches, achieving the best results on the ISPD 2012 and 2013 benchmarks. Sharma et al. [25] extended this work with multithreading and a new LM update method to reduce the number of iterations to converge, achieving a 15× speedup. Reimann et al. [22] applied LR gate-sizing on industrial designs. Roy et al. [23] extended the LR formulation to handle multiple modes and corners. Daboul et al. [4] used a resource sharing formulation, which is a specialization of LR. Shklover et al. [26] added clock skew to the LR formulation and resized both gates and clock buffers. Sharma et al. [24] combined LR gate sizing and slack-based clock skew assignment, achieving significant power savings on the ISPD 2012 contest benchmarks. Mangiras et al. [19] extended the LR formulation for timing-driven placement to include flip-flops, gates, and local clock buffers.

Buffer insertion can help fix early timing violations by increasing the path delay. The main concerns are to avoid increasing late timing violations, and to minimize the added area and power for the hold buffers. Huang et al. [13] used an LP formulation to minimize the number of added hold buffers. Tu et al. [28] tackled hold violations across different power modes in ultra-low voltage designs. Wu et al. [31] presented an LP approach to model setup and hold constraints and assign delays that should be inserted on each node to solve hold violations. They then used DP to perform buffer insertion. Han et al. [9] proposed an integer linear programming hold-buffer insertion approach that achieves hold timing closure across multiple corners.

Buffers can also be used to reduce delay and drive large nets. Van Ginneken [29] found the optimal buffer positions given a set of arrival timing constraints. Lillis et al. [16] extended Van Ginneken's algorithm to account for multiple buffer types, while Wang et al. [30] presented a lower complexity buffering algorithm. Jiang et al. [33] formulated simultaneous transistor sizing and buffer insertion on the output of setup critical gates, driving all or part of their fanout, to isolate critical paths. The buffer position choices were enhanced by sensitivity functions that modeled the expected leakage power and area gain from transistor sizing. Liu et al. [17] used an LR-based cost function for buffer insertion on each net, and to decide how many buffers to add to each net, but they only used one buffer size from the library. Ho et al. [32] utilized LR for buffer insertion, accounting for multiple buffer options and integrating density in the cost function. Finally, Hu et al. [12] proposed an approximation method that guarantees polynomial runtime complexity with minimum impact on the result.

3 LR DESIGN OPTIMIZATION

The proposed optimization engine aims to minimize the design's leakage power and area, while respecting timing and design constraints. The overall optimization problem can be written as:

$$\begin{aligned}
\textbf{min}: & \sum_{c\in cells} P(c) + A(c) - \sum_{j\in POs} slk_j^L - \sum_{j\in POs} slk_j^E \\
\textbf{s.t.:}\ & slk_j^L \le 0 \text{ and } slk_j^E \le 0, \forall j \in POs \\
& slk_j^L \le r_j^L - a_j^L \text{ and } slk_j^E \le a_j^E - r_j^E, \forall j \in POs \\
& a_i^L + d_{i\to j}^L \le a_j^L \text{ and } a_i^E + d_{i\to j}^E \ge a_j^E, \forall cells
\end{aligned}$$

where $P(c)$, $A(c)$ are the leakage power and area of cell c; slk_j is endpoint j's negative slack; a_j and r_j are pin j's arrival and required times; and $d_{i\to j}$ is the arc delay from pin i to pin j, including the wire delay. The indices E and L represent early and late timing.

Lagrangian relaxation incorporates the constraints into the cost function, multiplied by weights called Lagrangian Multipliers (LMs), as shown in (1). The higher the LM value, the more critical the respective constraint is.

$$\begin{aligned}
&\textbf{min}: \\
&\sum_{c\in cells} P(c) + A(c) - \sum_{j\in POs} slk_j^L - \sum_{j\in POs} slk_j^E + \\
&\sum_{j\in POs} \left(\lambda_{j0}^L slk_j^L + \lambda_{j0}^E slk_j^E\right) + \\
&\sum_{j\in POs} \left(\lambda_{j1}^L (slk_j^L - r_j^L + a_j^L) + \lambda_{j1}^E (slk_j^E - a_j^E + r_j^E)\right) + \\
&\sum_{c\in cells}\left(\sum_{i\in fanin_j} \lambda_{i\to j}^L (a_i^L + d_{i\to j}^L - a_j^L) + \lambda_{i\to j}^E (a_j^E - a_i^E - d_{i\to j}^E)\right) \quad (1)
\end{aligned}$$

By differentiating (1) with respect to the arrival times, according to the Karush-Kuhn-Tucker optimality conditions, we end up with the following LM flow-conservation rules:

$$\sum_{i\in fanin(j)} \lambda_{i\to j}^L = \sum_{k\in fanout(j)} \lambda_{j\to k}^L \text{ and } \sum_{i\in fanin(j)} \lambda_{i\to j}^E = \sum_{k\in fanout(j)} \lambda_{j\to k}^E \quad (2)$$

The LMs for the output pin of a cell are proportionally distributed to the LMs of the cell's input-output arcs. For example for gate G in Figure 1, net 6's outgoing LM is propagated to the LMs into net 6, preserving the equalities: $\lambda_{4\to 6}^L + \lambda_{5\to 6}^L = \lambda_{6\to 7}^L$, $\lambda_{4\to 6}^E + \lambda_{5\to 6}^E = \lambda_{6\to 7}^E$. For flip-flops, the LMs added at their Q pin are distributed to the LM of the $CK \to Q$ arc. For the example shown in Figure 1, this translates to $\lambda_{13\to 9}^L = \lambda_{9\to 11}^L + \lambda_{9\to 12}^L$, $\lambda_{13\to 9}^E = \lambda_{9\to 11}^E + \lambda_{9\to 12}^E$.

Substituting (2) into (1), we simplify the objective to (3).

$$\min \sum_{gates} P(c) + A(c) + \sum_{i\to j} \lambda_{i\to j}^L d_{i\to j}^L - \lambda_{i\to j}^E d_{i\to j}^E \quad (3)$$

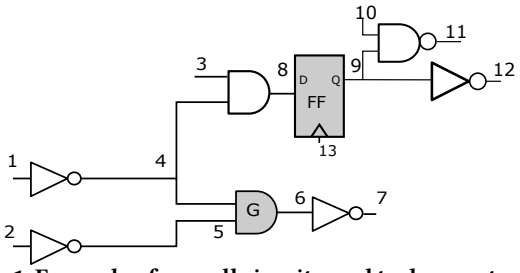


Figure 1: Example of a small circuit, used to demonstrate LM propagation and local cost calculation

From (3), it can be observed that each LM highlights the criticality of its associated timing arc. A large late LM means that the delay of the arc should be decreased to minimize the cost function, while a large early LM means that the delay of the arc should be increased. Whereas, a low value for an LM means that the delay of the arc is not critical to the optimization.

3.1 Overall optimization Flow

The proposed optimization flow is presented in Figure 2. Initially, all cells are sized to the smallest size that does not violate any maximum load or slew constraints [15]. The LMs for every arc are initialized to 1, and the iterative optimization process begins.

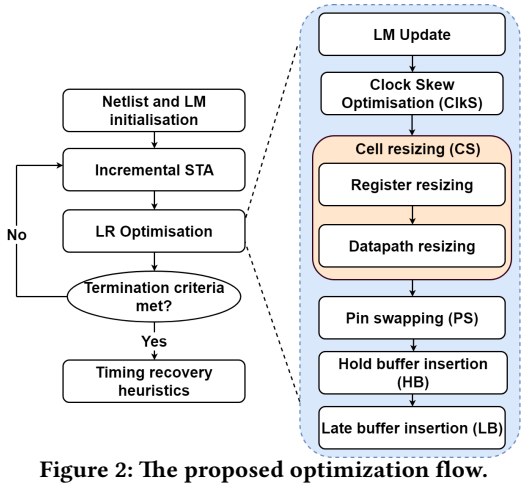


Figure 2: The proposed optimization flow.

At the start of each iteration, an incremental timing update for all corners is performed. Then, we use timing information for only two corners: the most critical early and late corner. The critical late corner is that with the highest total negative slack (TNS), or the corner with the lowest late total slack, if no corner has late violations. The same applies for the most critical early corner. Where a timing variable is used for a mode (early, late), it is implied that its value is calculated from the respective critical corner. The critical corners are redetermined after every timing update.

Each LR optimization step begins by updating the LMs according to the process detailed in Section 3.2. Then, the selected netlist transformations are executed one after the other. Clock-skew assignment inserts or removes a fixed delay from the clock pins of the registers, in order to help LR cost minimization. Register and gate resizing choose an appropriate version for the most critical cells in the circuit. Pin swapping attempts to swap the sink pin of the most timing critical net of the gate with another equivalent pin, in order to help the timing critical net with its violations. Hold-buffer insertion finds the positions and number of buffers that need to be inserted to reduce early violations. Late buffer insertion chooses if a buffer should be inserted on the sink of cells with high fanout-loads-to-input capacitance ratios, and the size of the buffer if insertion proves beneficial. All of the above methods are driven by the LM values and try to optimize the LR cost using slack information only to ensure that they are not degrading the timing violations. So, accurate slack information is not necessary, which allows for the whole series of transformations within an iteration to take place without the need for any new incremental timing update.

Firstly, we apply the transformations that affect the timing startpoints and endpoints of the design, which are clock-skew optimization and register resizing. Then, gate sizing and pin swapping, that traverse the netlist in topological order, are applied. This also allows them to propagate the updated timing information from the startpoints to the endpoints with local timing updates. The transformations that are applied on intermediate topological levels of the circuit are executed last. This order ensures that the local timing updates will propagate timing information rather accurately, thus removing the necessity of an incremental timing update after the application of each transformation.

Optimization stops when the maximum number of iterations is reached, or if the solution quality doesn't improve for 2 consecutive iterations. If timing constraints are satisfied, the quality of the solution is equal to the area/power improvement, whereas, if timing violations are present, termination is judged by TNS improvement. In the end, a final brute-force timing recovery step is executed.

3.2 LM update

The LMs for each primary output and flip-flop D pins are updated using the modified sub-gradient optimization proposal in [27]:

$$\lambda^L_{j0} = \lambda^L_{j0}\left(\frac{a^L_j}{r^L_j}\right), \quad \lambda^E_{j0} = \lambda^E_{j0}\left(\frac{r^E_j}{a^E_j}\right) \ \forall i \in endpoints$$

$$\lambda^L_{i\to j} = \lambda^L_{i\to j}\left(\frac{a^L_i + d^L_{i\to j}}{a^L_j}\right), \quad \lambda^E_{i\to j} = \lambda^E_{i\to j}\left(\frac{a^E_j}{a^E_i + d^E_{i\to j}}\right) \ \forall arc_{i\to j}$$

The delays for early and late timing analysis are calculated from the corresponding critical corner, as mentioned in Section 3.1.

After updating the output LMs, their values must be distributed to all nets satisfying the flow conservation conditions (2). The distribution is performed by traversing the circuit in reverse topological order. At each visited cell, the sum of LMs at the output pins is distributed to the LMs of the input pins. When an LM value needs to be distributed to multiple incoming arcs, this distribution is done based on the ratio of the LMs of the corresponding timing arcs. Such distribution increases the LMs on critical paths, and, therefore, the WNS is also expected to be minimized. Also, since LMs are accumulated at each branching point, the higher the number of violating endpoints affected by an arc, the higher the value of the corresponding LM.

4 NETLIST TRANSFORMATION OPTIMIZATIONS

4.1 Cell Resizing

The cell resizing algorithm (Algorithm 1) examines different versions for each cell and selects the one that minimizes the local cost

Algorithm 1: Cell resizing algorithm

```
foreach cell c ∈ candidate_cells in topological order do
    best_cost ← localCost(c) ;
    best_version ← cellVersion(c) ;
    neg_slack ← localNegativeSlack(c) ;
    foreach version v ∈ equivalent_versions do
        resize cell c to v;
        if (violates_load_constraints(c)) then
            skip version;
        end
        update timing locally;
        new_neg_slack ← localNegativeSlack(c) ;
        if new_neg_slack < γ · neg_slack then
            skip version;
        end
        new_cost ← localCost(c);
        if new_cost < best_cost then
            best_cost ← new_cost ;
            best_version ← new_version ;
        end
    end
    resize cell c to best_version ;
    update timing locally ;
end
```

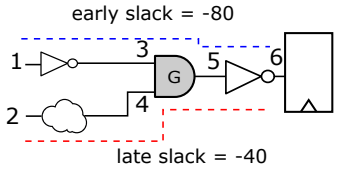
function. Every cell is visited in topological order. If a cell is a candidate for resizing (resizing candidate choice is analyzed in subsection 4.1.2), then every size of the cell is explored. After trying all sizes, the cell is resized to the version that minimizes the local cost function without introducing load violations and without degrading the slack of its nets over a threshold γ [8]. Similar to [8] and [25], the cost function (4) used for the selection is a localized version of the global cost function (3) that includes only local timing arcs:

$$LC(v) = P(v) + A(v) + \sum_{i \to j \in local\_arcs} \lambda^L_{i\to j} d^L_{i\to j} - \lambda^E_{i\to j} d^E_{i\to j} \quad (4)$$

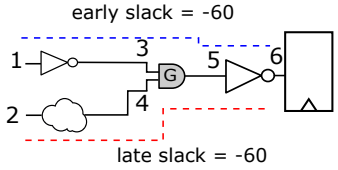
with $P(v)$ and $A(v)$ being the leakage power and area, respectively, of size v of the examined cell.

The local timing arcs for each cell include the arcs of the resized cell, its immediate fanin and fanouts, and the arcs that share a common fanin with the resized cell. For gate G in Figure 1, the local arcs include: arcs of G $(4 \to 6, 5 \to 6)$, the fanin arcs $(1 \to 4, 2 \to 5)$, the fanout arcs $(6 \to 7)$, and the common fanin arcs $(4 \to 8)$. Flip-flops are handled similarly to gates. For flip-flop FF in Figure 1, the local arcs include arcs $(13 \to 9)$, $(3 \to 8, 4 \to 8)$ and $(9 \to 11, 9 \to 12)$.

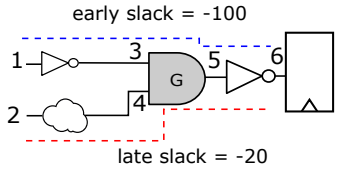
4.1.1 Handling of conflicting constraints. Relying on the local cost function (4) may lead to contradicting sizing decisions when a timing arc has early and late violations. Consider the case depicted in Figure 3a. In this case, gate G has an early violating path passing through pin 3, and a late violating path through pin 4, ending up with conflicting violations on pin 5. No matter which decision the sizer takes, there is no choice that will reduce both timing violations. More specifically, the LR algorithm would try to reduce the violations on the side with the higher LM. For example, if the early LMs are higher than the late ones, it would try to slow down the



(a) Example of conflicting violations.



(b) Result of normal resizing.



(c) Promoting late timing for conflicting arcs.

Figure 3: Example of resizing gate G with conflicting violations. It is seen that late slack is minimized in case (c) and, if early slack is handled by buffer insertion, case (c) will end up with the least amount of negative slack.

arc. This is shown in Figure 3b, where the sizer downsized gate G to increase its delay. However, this choice worsens late slack. So, eventually, the algorithm balances the slacks on both (early-late) sides, without truly solving any violations.

In order to tackle this, we decided to focus on one side of the violations only when an arc has conflicting violations. Since early violations can always be fixed by other means, such as hold-buffer insertion, we decided to focus on late violations only. This can be done by omitting the terms $\lambda^E_{i\to j} d^E_{i\to j}$ from the cost function (4).

So, in case of conflicts, the LR optimization focuses on speeding up arcs with conflicting violations. Based on this strategy, in our example shown in Fig. 3c, the sizer would upsize gate G to reduce late violations, hoping that a hold-buffer insertion algorithm would later insert buffers on the fast path.

4.1.2 Filtering candidate cells. Attempting to resize every cell in the circuit can be very computationally expensive, especially for large circuits. Identifying which cells are critical for improving circuit performance is an important step for reducing the runtime without compromising the final quality of results.

Therefore, before resizing any cell, we make a list of the cells that participate on the top k late and early critical paths, of the cells placed on the fanouts of late critical paths, and cells with significant positive slack and large power and area savings potential. From this list of cells, we need to find the ones that are the most sensitive to resizing. In other words, their resizing greatly affects the cost function. This can be done by calculating a sensitivity for each cell of the list and examine the resizing of only the N cells with the largest sensitivities. Depending on the type of each cell, we employ a different sensitivity factor:

- *Cells on the late critical path*: Cells on the late critical path will usually be upsized to speed up the path, while trading

off power and area. So, the cells that will be resized should be the cells that have the best cost-power-area tradeoff. To calculate the sensitivity for those cells, we compare the local cost difference, as well as the power and area difference between the current and the next bigger size.

- *Cells on the early critical path*: Cells on the early critical path will try to slow down the path, usually by downsizing. Since this does not create a trade-off (downsizing on an early critical path will save power, area, and slack), we do not involve power and area in the sensitivity function. For this set of cells, only early slack minimization matters. In this case, we compare the value of the local cost function (4) between the current size and the next smaller one, after neglecting the power and area of this choice.
- *Cells on the fanout of late critical paths*: Those cells should be downsized to reduce the output load of the cells on the critical path, and thus area and power are certainly reduced. Therefore, we compare the value of the local cost function (4) between the current size and the next smaller one, after neglecting the power and area of each choice.
- *Cells with positive slack*: When a cell has both positive early and late slacks, it will be downsized to save leakage power and area, providing this does not create new slack violations. In this step, we want cells that have both a large margin for leakage and area minimization, and enough positive slack to allow a downsize. To achieve this, we calculate the power-area difference of the current size and the next smaller size for cells that have enough positive slack.

Please note that, even though we assume that a cell should be upsized or downsized for its sensitivity calculation, during true resizing all possible sizes for the selected cells are tried with the full local cost function, including area and leakage.

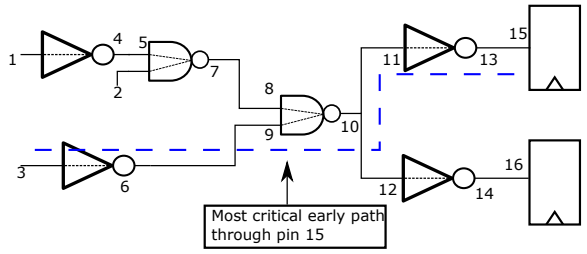


Figure 4: Hold buffer insertion example. Pin 10, which is part of the early critical path, has the highest $\lambda^E - \lambda^L$ difference and receives extra buffers for alleviating hold-time violations.

4.2 Buffer insertion

Buffering is a versatile design optimization that can be efficiently applied for various design targets. In this work, we employ two forms of buffering that target early and late timing violations.

4.2.1 Early timing violations. Solving early/hold timing violations requires slowing down the signal propagation on violating paths. This can be achieved by appropriate buffer insertion. Hold buffers should be inserted in positions where they will affect many hold paths, minimizing the number of buffers that need to be added (and, consequently, the power and area overhead). On the other hand, inserting a buffer on a pin that affects a late critical path will introduce late slack overhead. So, hold buffer insertion should be broken down to two sub-problems: choosing the position where we want to insert buffers, and choosing how many buffers to insert.

The proposed method for inserting hold buffers is based on the LM values. A high early LM value on an arc means that this arc is either very critical for early, or that it drives a large number of hold critical arcs (or both). A low late LM on an arc means that this arc is not critical for late. Thus, intuitively, pins with a high early/late LM difference would be the best candidates for buffer insertion.

The delay added should be sufficient to fix hold violations, but should not worsen late timing. If there is positive late slack available, it can be used to reduce hold violations. Using all of the late slack in one iteration is suboptimal, as LM values change after each iteration, so the pin indicated as candidate by the early-late LM difference might change. Thus, it is best to avoid inserting many buffers at once, as this would not allow further optimization.

For every hold violation endpoint , we store the worst early path through it. The paths' pin with the highest early-late LM value is stored in a list of candidates. For example, in Fig. 4, the candidate pin for the worst early path through endpoint pin 15 could be pin 10. For every candidate pin, the following steps are executed:

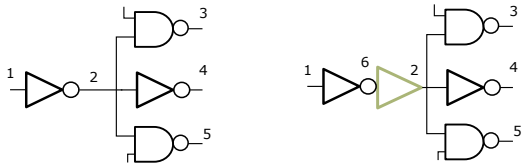
Step 1: Finding a buffer with an appropriate input capacitance. A buffer size with an input capacitance similar to the output load its fanin cell is driving is identified. If the pin where the buffer will be inserted is an input pin, the buffer chosen has an input capacitance similar to the input capacitance of the pin (for example, in Fig. 4, a buffer inserted on pin 8 should have a $C^{in}_{buf} = C^{in}_{8}$). If the pin where the buffer insertion will take place is an output pin, the chosen buffer should have an input capacitance similar to the load driven by the pin (for example, in Fig. 4, a buffer inserted on pin 10 should have a $C^{in}_{buf} = C_{net} + C^{in}_{11} + C^{in}_{12}$). This is done to avoid big timing changes on the rest of the timing arcs, which would be expensive to calculate accurately, and to prevent speeding up the fanin cell, which would worsen early timing.

Step 2: Finding how many buffers are required. After selecting a buffer size, we can calculate its delay, as the input slew and output capacitance are known. Because the buffer chain consists of the same buffer sizes, every added buffer will be driving the same load, so we assume that they will also have the same delay (small differences in delay might occur due to different input slew). So, for a known buffer delay and a known early negative slack, the number of buffers required to be added by early mode on pin p is equal to $N^E_{buf} = -slk^E_p / d^E_b$, where slk^E_p is the slack on the pin where the buffers will be inserted, and d^E_b is the delay of the buffer.

Step 3: Finding how many buffers late slack allows. Similar to the previous step, we calculate how many buffers can be added on the pin before making its late slack negative. The number of buffers allowed to be added by late mode is equal to $N^L_{buf} = slk^L_p / d^L_b$, where slk^L_p is the negative slack on the pin where the buffers will be inserted, and d^L_b is the delay of the buffer.

Step 4: Insert the appropriate number of buffers. After calculating how many buffers early mode requires and late mode allows, the number of buffers added is equal to $\min(N^E_{buf}, N^L_{buf}/2)$. This way, the immediate consumption of all late slack is avoided, while still allowing the insertion of an appropriate number of buffers that would reduce hold violations.

4.2.2 Late timing violations. Buffering helps reduce the output load of cells with a large fanout, by decreasing their delay and helping reduce late slack violations.



(a) Candidate for buffer insertion pin 2 (b) After buffer insertion

Figure 5: Example of late buffer insertion on pin 2. The local cost without a buffer is λd sum (Eq. (4)) over the arcs for pin 2: $1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5$. The local cost with a buffer inserted is the cost of the arcs around the buffer: $1 \rightarrow 6, 6 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5$, and the buffer's power and area. If the new cost is lower than the original, the buffer is kept.

Firstly, the cells with negative late slack are sorted by their C_{out}/C_{in} ratio. Then, buffer insertion is attempted on the source of the fanout net of the top 100 cells. The local cost of net n (eq. (4)), where the buffer will be inserted, is stored (without including the leakage power and area, since there will be no resizing). The arcs used for calculating the local cost are the arcs connected to net n. These will be the local arcs around the new buffer after its insertion. For example, in Fig. 5, the arcs included in the local cost of net 2 will be $1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4$ and $2 \rightarrow 5$.

Then, buffer insertion for every buffer size in the library is attempted at the source of net n. For instance, in the example shown in Fig. 5, the buffer would be inserted at a source of net 2. Subsequently, the local cost around the buffer is calculated in the same way that the local cost of cells is calculated during resizing, using equation (4). After trying all the buffer sizes, the best score from all sizes is kept. If the best buffer score is smaller than the score without the buffer, a buffer of that type is inserted.

4.3 Pin Swapping

When logic functionality allows, the nets connected to the inputs of a gate can be swapped, in order for the most critical net to take advantage of the lower input-output delay of the gate.

The pin swapping algorithm visits combinational gates in topological order. For each gate, the most critical pin is identified, with the most negative early or late slack. For all pins on the gate that are equivalent to the critical pin, a swap is attempted. After each swap, the timing is updated locally and the local cost function of the gate is recalculated. After every swap is done, the swap with the best local cost is kept and the next gate is processed.

4.4 Clock Skew Assignment

Clock skew assignment is an important step for reducing timing violations in a design. By speeding up or slowing down a clock pin arrival time, slack on one side of a register is traded off for slack on the other side. For a register with an input D pin and an output Q pin, the slacks are calculated by the following equations:

$$slk_D^{L'} = slk_D^L + d_{CLK}, \quad slk_Q^{L'} = slk_Q^L - d_{CLK}$$
$$slk_D^{E'} = slk_D^E - d_{CLK}, \quad slk_Q^{E'} = slk_Q^E + d_{CLK}$$

where d_{CLK} is the delay added on the clock pin. Slowing down the clock pin favors slk_D^L and slk_Q^E and degrades slk_Q^L and slk_D^E, respectively. Speeding up the clock pin has the opposite effect.

The added clock pin delays should be scaled across corners. Since the clock pin offsets would be implemented using the available clock buffers of the library, we compute a scaling factor for each corner by approximating the ratio of the clock buffer delay of that corner compared to the buffer delay of the typical corner, for multiple input slew and output load values. So, when a delay d_{CLK} is added on a clock pin for the typical corner, a delay $d_{CLK} \dot{r} atio_c$ will be added for corner c, where $ratio_c$ is the average buffer delay ratio for corner c and the typical corner.

The clock skew assignment algorithm visits every register in the design. For each register, the worst early and late slacks on the D pin and the most critical on the Q pin are computed. Then the LM propagated towards the clock pin are calculated. Based on the KKT conditions, they are equal to $\lambda_{CLK}^L = \lambda_Q^L + \lambda_D^E$, $\lambda_{CLK}^E = \lambda_Q^E + \lambda_D^L$. We can decide if the clock pin requires a speedup or a slowdown from the sign of $\lambda_{CLK}^L - \lambda_{CLK}^E$. If it is positive, a delay should be added on the clock pin, whereas, if it is negative, a delay should be subtracted from the clock pin. After deciding if a delay will be added or subtracted based on LM values, a slack check is performed. We calculate the worst out of all 4 slacks (early-late, D-Q side) and ensure that the decision made based on LM values does not degrade the worst slack. If the worst slack is not degraded, a delay d_{CLK} is added, or subtracted, and scaled for all corners. d_{CLK} should not be too small, as it will lead to a very slow convergence, nor too large as it will dramatically change the timing profile of the circuit and lead the LR method to divergence.

4.5 Timing Recovery Heuristics

The timing recovery steps resizes the driver cell affecting the most violating endpoints [8]. The sizes tested are the immediately smaller and bigger size. After either move, we perform local timing update on the critical corner and calculate the new local negative slack on the cell's output. If it is improved compared to the initial slack, we perform an incremental timing update and ensure that the TNS has improved too. If it has, this cell version is kept and the process is repeated. If the timing has not improved we revert the change and move on to the next most critical cell. Timing recovery stops if all timing violations are solved, if the TNS stops improving, or if a certain number of incremental timing updates is reached. This recovery step is performed twice: once for the remaining late timing violations, and once for the early timing violations. For the last remaining early timing violations, a buffer chain is inserted in front of every hold violating endpoint, until the violations are fixed. This is the last optimization step performed, ensuring that the final design is hold-violation free.

5 EXPERIMENTAL RESULTS

The proposed method was implemented in C++ inside the open-source RSyn framework [7] after extending it for multi-corner timing analysis. The results of this work were validated using the TAU 2019 contest benchmarks and compared against the winner of the contest [2]. We extracted the presented results by running the executable the TAU 2019 winner sent to us, which was the one submitted to the contest. The 6 benchmarks of the contest include SPEF and SDC files for each netlist, and 5 different standard-cell library files, one for each corner. The designs only have timing constraints on register-to-register paths, leaving primary inputs and outputs unconstrained. All benchmarks also include clock trees that are mostly unrealistic, since they do not have RC parasitics, and the clock arrival times are highly unbalanced. For this reason, the TAU 2019 contest winner removed the clock buffers from the clock tree and rebuilt it without including any RC parasitics of the nets connecting the clock-tree buffers. Note that this feature of assuming ideal wires on the clock tree was, indeed, permitted by the contest.

Therefore, in order to allow for a more realistic comparison (assuming that designs are compared in a pre-CTS stage), we removed all clock buffers from both the initial set of benchmarks fed to our algorithm, and from the final netlist of the TAU 2019 winner. Nevertheless, for the latter, we kept the assigned clock pin arrival times (acting as useful clock skew values), so that the timing performance of the TAU 2019 winner remained unaffected. Even though we removed the unrealistic – due to the lack of RC parasitics – clock trees, we retained their useful effect on timing.

Table 1: The leakage and area under the best period reported by the contest for typical corner.

Design	#Cells	Period (ps)	Leakage (uW)		Area (um^2)	
			Ours	Winner	Ours	Winner
s1196	584	334	12	13	545	565
systemcdes	2825	696	68	82	3190	3565
usb_funct	10535	828	352	402	17058	17831
vga_lcd	87958	684	2826	3125	143168	146802
leon2_iccad	793286	1483	24925	30249	1232760	1311985
leon3mp_iccad	649191	1661	19589	23806	961941	1028117

In Table 1, we show the results of running both flows for only the typical corner, and for the clock period specified by the contest as the one where the contest winner achieves timing closure for the typical corner. Our flow also achieves timing closure for these clock periods, saving 17% more leakage power and 6% more area.

Table 2: The best clock period achieved for all corners

Design	Period (ps)		Leakage (uW)		Area (um^2)	
	Ours	Winner	Ours	Winner	Ours	Winner
s1196	1040	918	12.1	12.6	550	569
systemcdes	1777	1788	87	96	3665	3975
usb_funct	2166	2306	419	402	18579	17812
vga_lcd	1871	2826	3215	3106	149033	146257
leon2_iccad	4532	4677	25170	30354	1237510	1312960
leon3mp_iccad	3878	5246	20045	23816	971773	1030860

Secondly, in Table 2, we present the best clock period for which each flow achieves closure for all corners. On average, our flow achieves 14% better clock period, while, simultaneously, saving 15% in leakage and 5% in area. More specifically, our flow results in a better clock period for all designs except one, with the benefits reaching up to 35%. The exception is s1196, where our clock period is 5% worse. In every other design, our flow achieves a better clock period with minimal, or no power and area overhead.

Furthermore, Table 3 depicts the results after running both flows on the same clock period for which both achieve timing closure, i.e., the clock period of the slower. Our flow achieves on average 16% lower leakage power and 6% lower area, and results in smaller leakage and area for every benchmark.

These improvements are the result of the smooth cooperation between the transformation optimizations. Sizing identifies the most critical cells and resizes them to the size that locally reduces the LR score. Resizing targets the 2000 highest sensitivity cells for the 1000 most critical paths. Early slack is allowed to degrade when resizing cells with conflicting violations, in favor of late slack optimization, and it is then reduced by cell sizing on early violating paths, hold-buffer insertion, and clock-tree optimization. Thus, hold violations stay under control, while late violations are constantly reduced. This is highlighted in Fig. 6 for vga_lcd. The late TNS starts off as very negative and slowly converges to 0 as iterations progress. Early TNS is always 0 at first for all designs, since removing the clock tree solves all hold violations. Early violations appear during optimization as paths become faster and clock skew assignment trades off early for late slack. Appropriate cell sizing and hold-buffer insertion keep early slack under control, usually reducing it back to 0 at the end of the iterative optimization.

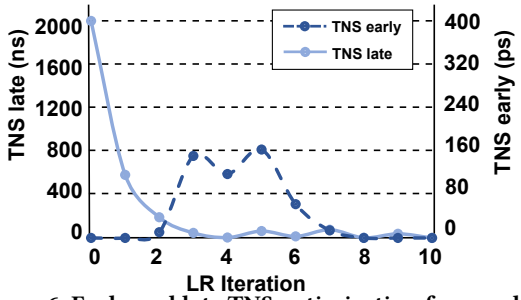


Figure 6: Early and late TNS optimization for vga_lcd.

The runtime of the compared methods are presented in Table 3 on columns 13 and 14. They were run on a 3.6GHz Intel Core i7-4790 Linux workstation with four cores and 32GB of RAM. The iterations that the proposed LR-based optimization needed per benchmark are shown on column 7 of Table 3. In the smaller benchmarks, the runtime of the two methods is almost the same. However for the larger benchmarks, the winner of the contest is significantly faster. We cannot tell if the latter experiences a runtime-vs-quality-of-results tradeoff, i.e., if the results could improve with more runtime. Our future goal is to adopt a multi-threaded approach for each transformation method to reduce runtime. The runtime contribution of each method is detailed in columns 8 to 12 of Table 3. The lion's share belongs to cell sizing and clock skew assignment. Early and late buffer insertion heuristics are applied on a small number of pins, so they are fast.

To quantify the effectiveness of each transformation, we reran the flow excluding one of the available transformations in each run. Due to space limitations, Table 4 presents the results just for the indicative design vga_lcd. It shows the TNS, WNS, power, and area for the best clock period achieved by our flow (the clock period reported in Table 2) for the initial netlist and the optimized netlist. Since hold timing constraints are satisfied in every case, TNS and WNS refer to late violations only.

It is observed that every method is required in the flow in order to achieve the best clock period reported. Cell sizing is required for leakage power and area minimization, since the leakage and area results for the run without cell sizing is significantly larger. Clock skew assignment has the most significant contribution to timing violation minimization, and the rest of the methods further improve the final results without a significant runtime overhead.

Regarding hold buffer insertion, it is observed that early timing achieves closure even when it is not applied in the optimization loop, due to the early timing recovery that runs in the end of the optimization. Nevertheless, omitting it results in a worse late timing, since the iterative LR based method picks better positions for hold buffer insertion.

6 CONCLUSIONS

This work introduces two novel aspects with respect to timing-driven design optimization. The traditional LR-based gate sizing algorithm is enhanced to include flip-flops and to handle cases where

Table 3: Comparison of the leakage and area at the clock period where both flows achieve closure for all corners.

Design	Period (ps)	Leakage (uW)		Area (um^2)		LR iters	Runtime (s)						
		Ours	Winner	Ours	Winner		CS	PS	LB	HB	ClkS	Total	Winner
s1196	1040	12	13	550	569	4	1	0.02	0.01	0	0.01	2	2
systemcdes	1788	85	96	3603	3975	8	13	0.92	0.10	0.12	0.03	17	21
usb_funct	2306	397	402	18002	17812	10	43	1.14	0.21	0.88	0.41	50	52
vga_lcd	2826	3075	3106	145752	146257	8	312	9.02	2.92	5.47	21.81	455	24
leon2_iccad	4677	24996	30354	1234180	1312960	10	2341	23.58	10.48	19.91	1291.11	4471	452
leon3mp_iccad	5246	19632	23816	962764	1030860	9	2046	13.12	8.11	22.04	812.95	3862	362

Table 4: Methods' contributions to the results for vga_lcd.

Netlist		TNS (ns)	WNS (ns)	Leakage (uW)	Area (um^2)
Initial		-19731.00	-12.60	4841	192551
Initial downsizing		-18945.00	-11.80	3078	145692
Full flow		0	0	3215	149033
Opt removed	CS	-50.26	-0.34	3599	161287
	PS	-0.02	-0.01	3205	148836
	LB	-0.11	-0.11	3128	147369
	HB	-2.63	-0.17	2927	145558
	ClkS	-1598.54	-0.44	3174	147832

pure LR-based gate sizing leads to conflicting decisions. Moreover, for the first time, interleaved inside the LR optimisation loop are multiple netlist transformation algorithms that operate on similar local cost functions based on the LM weights. The results show that this approach is a scalable alternative for smoothly integrating multiple transformations that, in the past, have been applied independently to the design. Most importantly, this allows for a modular customization of the optimization flow, whereby transformations are added or removed, based on their effectiveness for a design, without altering the global optimization framework that guides the entire process. The adaptive selection of the order of application of each transformation is left for future work.

ACKNOWLEDGMENTS

Dimitrios Mangiras is supported by the Onassis Foundation - Scholarship ID: G ZO 014-1/2018-2019. This research has been supported by a research grant from Mentor, a Siemens Business to DUTH.

REFERENCES

[1] C.-P. Chen, C. C. N. Chu, and D. F. Wong. 1999. Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. *IEEE TCAD* 18, 7 (Jul 1999), 1014–1025.
[2] H-H Cheng, T-W Lin, Y-C Lin, Iris H-R Jiang, and P-Yu Lee. 2019. Design Optimization Contest, Team iTimer, TAU workshop 2019. https://sites.google.com/view/tau-contest-2019/home
[3] D. G. Chinnery and K. Keutzer. 2005. Linear Programming for Sizing, Vth and Vdd Assignment. In *Int. Symp. Low Power Electronics and Design*. 149–154.
[4] S. Daboul, N. Hähnle, S. Held, and U. Schorr. 2018. Provably Fast and Near-Optimum Gate Sizing. *IEEE Trans. on CAD* 37, 12 (Dec 2018), 3163–3176.
[5] H. Fatemi, A. Kahng, H. Lee, J. Li, and J. Pineda de Gyvez. 2019. Enhancing sensitivity-based power reduction for an industry IC design context. *Integration* 66 (Feb 2019).
[6] J. P. Fishburn and A. E. Dunlop. 1985. Tilos: A Posynomial Programming Approach to Transistor Sizing. In *Int. Conf. Computer-Aided Design*. 326–328.
[7] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis. 2017. Rsyn: An Extensible Physical Synthesis Framework. In *Int. Symp. On Physical Design*. 33–40.
[8] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis. 2014. Effective Method for Simultaneous Gate Sizing and V th Assignment Using Lagrangian Relaxation. *IEEE Trans. on CAD* 33 (April 2014), 546–557.
[9] I. Han, D. Hyun, and Y. Shin. 2016. Buffer insertion to remove hold violations at multiple process corners. In *Asia and South Pacific Design Automation Conf.* 232–237.
[10] S. Held. 2009. Gate Sizing for Large Cell-based Designs. In *Design, Automation and Test in Europe*. 827–832.
[11] S. Hu, M. Ketkar, and J. Hu. 2007. Gate Sizing For Cell Library-Based Designs. In *Design Automation Conf.* 847–852.
[12] S. Hu, Z. Li, and C. J. Alpert. 2009. A fully polynomial time approximation scheme for timing driven minimum cost buffer insertion. In *Design Automation Conf.* 424–429.
[13] S. Huang, G. Jhuo, and W. Huang. 2010. Minimum buffer insertions for clock period minimization. In *Int. Symp. Computer, Communication, Control and Automation*. 426–429.
[14] L. Lavagno, I. Markov, G. Martin, and L. Scheffer. 2016. *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*. Taylor and Francis group.
[15] L. Li, P. Kang, Y. Lu, and H. Zhou. 2012. An efficient algorithm for library-based cell-type selection in high-performance. In *ICCAD*. 226–232.
[16] J. Lillis, C.-K. Cheng, and T. Y. Lin. 1996. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE J. of Solid-State Circuits* 31 (March 1996), 437–447.
[17] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou. 1999. An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation. In *Int. Conf. Computer Design*. 210–215.
[18] N. D. MacDonald. 2010. Timing Closure in Deep Submicron Designs. In *Design Automation Conference*.
[19] D. Mangiras, A. Stefanidis, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. 2019. Timing-Driven Placement Optimization Facilitated by Timing-Compatibility Flip-Flop Clustering. *IEEE Trans. on CAD* (2019).
[20] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer. 2003. Minimization of Dynamic and Static Power Through Joint Assignment of Threshold Voltages and Sizing Optimization. In *Int. Symp. Low Power Electronics and Design*. 158–163.
[21] M. M. Ozdal, S. Burns, and J. Hu. 2012. Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs. *IEEE Trans. on CAD* 31 (2012), 1558–1571.
[22] T. J. Reimann, C. N. C. Sze, and R. Reis. 2016. Cell Selection for High-Performance Designs in an Industrial Design Flow. In *Int. Symp. Physical Design*. 65–72.
[23] S. Roy, D. Liu, J. Singh, J. Um, and D. Z. Pan. 2016. OSFA: A New Paradigm of Aging Aware Gate-Sizing for Power/Performance Optimizations Under Multiple Operating Conditions. *IEEE Trans. on CAD* 35 (Oct 2016), 1618–1629.
[24] A. Sharma, D. Chinnery, and C. Chu. 2019. Lagrangian Relaxation Based Gate Sizing With Clock Skew Scheduling - A Fast and Effective Approach. In *Int. Symp. Physical Design*. 129–137.
[25] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu. 2019. Fast Lagrangian Relaxation Based Multi-Threaded Gate Sizing Using Simple Timing Calibrations. *IEEE Trans. on CAD* (2019).
[26] G. Shklover and B. Emanuel. 2012. Simultaneous Clock and Data Gate Sizing Algorithm with Common Global Objective. In *Int. Symp. Physical Design*. 145–152.
[27] H. Tennakoon and C. Sechen. 2008. Nonconvex Gate Delay Modeling and Delay Optimization. *IEEE TCAD* 27 (Sept 2008), 1583–1594.
[28] W. Tu, C. Chou, S. Huang, S. Chang, Y. Nieh, and C. Chou. 2013. Low-power timing closure methodology for ultra-low voltage designs. In *Int. Conf. Computer-Aided Design*. 697–704.
[29] L. P. P. P. van Ginneken. 1990. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Int. Symp. Circuits and Systems*, Vol. 2. 865–868.
[30] X. Wang, W. Liu, and M. Yu. 2015. A distinctive O(mn) time algorithm for optimal buffer insertions. (04 2015), 293–297.
[31] P. Wu, M. D. F. Wong, I. Nedelchev, S. Bhardwaj, and V. Parkhe. 2014. On timing closure: Buffer insertion for hold-violation removal. In *Design Automation Conf.* 1–6.
[32] Y.-J. Ho and W.-K. Mak. 2008. Power and density-aware buffer insertion. In *Int. Symp. On VLSI Design, Automation and Test*. 287–290.
[33] Y. Jiang, S. S. Sapatnekar, C. Bamji, and J. Kim. 1998. Interleaving buffer insertion and transistor sizing into a single optimization. *IEEE Trans. VLSI Systems* 6, 4 (Dec 1998), 625–633.