

Physical Placement Driven by Sequential Timing Analysis

Aaron P. Hurst¹

Philip Chong²

Andreas Kuehlmann^{1,2}

¹ University of California at Berkeley, CA, USA

² Cadence Berkeley Labs, Berkeley, CA, USA

Abstract

Traditional timing-driven placement considers only combinational delays and does not take into account the potential of subsequent sequential optimization steps. As a result, the potential of re-balancing path delays through post-placement applications of clock skew scheduling and in-place retiming cannot be fully realized. In this paper we describe a new placement algorithm that is based on a tight integration of sequential timing analysis in the inner loop of an analytic solver. Instead of minimizing the maximum path delay, our approach minimizes the maximum mean delay on any circuit loop, thus enabling the full optimization potential of clock skew scheduling and in-place retiming. We present two versions of the new algorithm: one approximates sequential criticality and weights wires accordingly [1], the other extends this with the inclusion of explicit wire-length constraints for loops that limit the final clock period. Our algorithms are implemented using a hybrid, GORDIAN-style sequence of analytical placement steps interleaved with cell partitioning [2]. Our experiments on a set of large industrial designs demonstrate that the presented placement algorithm can minimize the contribution of interconnection delays to the clock period on average by 23.5% compared to a solution based on combinational delays.

1 Introduction

Sequential optimization techniques have the potential to significantly improve the performance, area, and power consumption of a circuit implementation to a degree that is not achievable with combinational synthesis methods. The goal is to balance the path delays between registers and thus to maximize the circuit performance without changing its input/output behavior.

Practical sequential optimization methods of interest are retiming [3, 4] and clock skew scheduling [5]. Retiming is a structural transformation that moves the registers in a circuit without changing the positions of the combinational gates. Retiming, although algorithmically well studied, has gained only limited use because of its impact on the verification flow and the inability to accurately model the load changes caused by register moves. When applied before placement, retiming can perform a coarse balancing of paths delays using wiring estimates. In-place retiming is applied after physical placement and incrementally repositions individual registers based on a precise evaluation of the timing impact including the change of interconnect delays. In-place retiming is limited to local perturbations of the placement and cannot correct for global problems.

Clock skew scheduling preserves the circuit structure by applying non-zero delays to the register clocks — thus virtually moving them in time. In recent years, clock skew scheduling has gained practical acceptance in multiple design flows, typically applied as

a post-placement optimization technique. Implementation strategies vary from clock tree topology construction [6, 7] to clock tree routing algorithms [8]. Recent work on multi-domain clock skew scheduling [9] has demonstrated that even with a very limited number of skew possibilities, almost all of the benefits can be realized; implementing a non-zero clock skew schedule in hardware may be easier than commonly believed. For the purposes of this work, however, we remain implementation independent.

The optimization potential of retiming and clock skew scheduling is bounded by the *critical cycle*, which is among all structural cycles of a circuit the one with the maximum value for $\phi_{\text{cycle}} = \text{total_delay} / \text{number_registers}$. If the combinational delays of all paths along this cycle are perfectly balanced by retiming or clock skew scheduling, then the design can be clocked with a period $\phi \geq \phi_{\text{cycle}}$.

Traditional timing-driven placement (e.g. [10]) minimizes the overall wire-length with the additional constraint that no combinational path is timing critical, i.e. the sum of the gate and interconnect delays on every path between two registers does not exceed the clock period ϕ . This notion of timing criticality is confined to the paths between single sets of registers and does not adequately capture the timing picture of the circuit when registers positions can be moved. A cyclic set of paths may be non-critical with respect to a flexible register clocking even if some of the individual paths significantly exceeds the cycle period. Likewise, combinational paths that have significant combinational slack may be part of the critical cycle. The combinational delay view of a design does not reveal much information about its true sequential criticality and may easily mislead the placer with respect to the overall optimization problem.

Previous work has addressed the integration of selective sequential optimization techniques and placement but remains incomplete. In [1] a partitioning and floorplanning approach is presented that considers retiming moves for registers on long interconnects. This work introduces the concept of *sequential slack* to express the sequential mobility of a register. These slacks are used as weights in the partitioning algorithm to approximate the sequential criticality of an edge. This work is extended in [11] for a multilevel placement algorithm using simulated annealing. However, as further discussed in Section 3, the sequential slack may dramatically underestimate the edge criticality in the presence of multiple critical regions and thus lead to suboptimal placements; the version of sequential slack computation using the reference point of [1] entirely misses the critical cycle in several of our examples. The first version of our algorithm presented in Section 4.1 builds on this notion of sequential slack. Unlike the work in [1], we guarantee that the most critical cycle will always be identified, since we choose a register which lies on this cycle as the reference point for the sequential slack computation. The second version of our algorithm

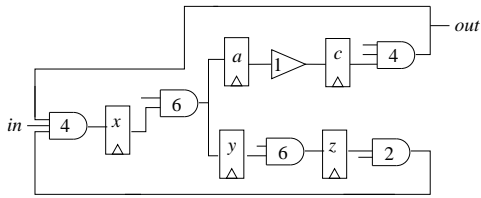


Figure 1: An example of two sequential cycles with the gate delays labeled. The lower set of registers $\{x, y, z\}$ forms the critical cycle and will limit the clock period after sequential optimization. However, a static timing-driven tool will incorrectly target path $c \rightarrow x$, likely at the expense of other paths.

presented in Section 4.2 corrects the problem for all potentially critical cycles by introducing explicit wire-length constraints for circuit loops that are near critical. We use Lagrangian relaxation to handle these constraints in an analytical placement phase similar to the approach presented in [10]. Repeating the timing analysis after each placement iteration additionally improves the odds that such cycles are identified and algorithmic convergence is achieved.

In [12] a budgeting algorithm is presented that computes delay bounds for a traditional placement algorithm under the assumption that retiming can be applied. In contrast to our work, this approach separates the budgeting and placement phases and as a result cannot take the dynamic interaction between placement, wire delays, and sequential optimization into account. A tight integration of sequential timing and placement is needed to capture this complex interaction.

Work on integrating retiming and placement for field programmable gate arrays is described in [13]. There, the authors also extend on the sequential slack computation technique found in [1]. However, they attempt to overcome the shortcomings of [1] through a process of random sampling of reference points. While this is certainly better than choosing a single arbitrary reference, this will still be ineffective in identifying the critical cycle if the sampling process does not happen to choose a register which lies on that cycle. The authors of [13] also demonstrate good results using a net weighting heuristic similar to our technique. However, given their framework of annealing-based placement, it is unclear how to efficiently include explicit cycle constraints, described here in Section 4.2. Our use of a quadratic programming-based formulation allows easy addition of these powerful constraints through Lagrangian relaxation.

2 Motivating Example

Timing-driven placement makes possible performance improvements over the pure minimization of wire-length. The portions of interconnect that have been identified as being timing critical can be given increased weight or attention to further reduce their lengths. This objective may come at the expense of other wires, but the slack available on these non-critical nets allows the wire delay increase to not affect the final clock period.

With the availability of in-place retiming or clock skew scheduling, the wires that limit the achievable clock period after sequential optimization are not necessarily the ones that limit the clock period beforehand. Figure 1 shows a simple sequential circuit. It is clear that without any changes to the clocking, path $c \rightarrow x$ is period-limiting. However, if the relative arrival of the clock at registers a and c can be moved by 1 delay unit backward and 3 delay units forward, respectively, the paths between x, y , and z will limit the

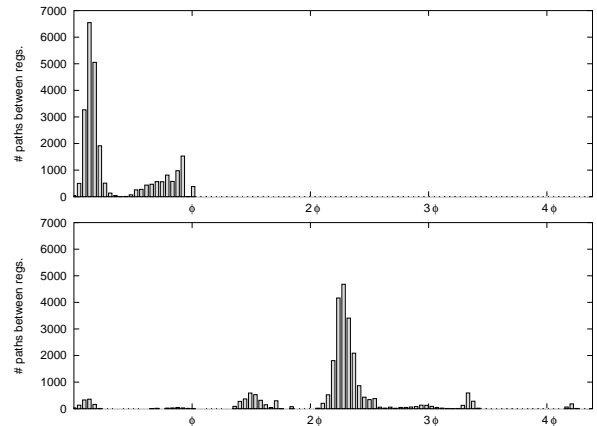


Figure 2: Design ind08 combinational slack distribution plotted above the true sequential slack distribution, both measured in the paths between registers. The slacks are labeled in multiples of the clock cycle after sequential optimization.

period. The information from static timing analysis is essentially meaningless when register boundaries can be moved.

Our experimentation suggests that the failure of static timing analysis to give an accurate picture of the post-sequential optimization timing can be significant in industrial designs. Figure 2 plots the distribution of combinational path slack above the distribution of true sequential path slack in one typical industrial design. It is immediately apparent that they offer very different versions of the timing criticality of the circuit. True sequential slack, described in more detail in Section 3.3, measures the amount of delay that can be added to a path before it becomes critical during post-placement retiming or clock skew scheduling. There is clearly more flexibility when sequential optimization techniques are considered; because slack can accumulate across multiple register boundaries, most of the paths in this design see more than one clock period of slack.

The use of inappropriate timing information to guide the placement process can have deleterious effects for the final design performance. If we reconsider the example in Figure 1, a decision to shorten path $c \rightarrow x$ at the expense of $y \rightarrow z$ appears to be beneficial from a static timing standpoint, however doing so would actually result in an increase in the clock period after sequential optimization. Our experimental results, shown at the end of this paper in Table 1, indicate that this effect is demonstrably present in several industrial designs; timing-driven placement indeed yields a design with a smaller clock period before sequential optimization, but having a larger final clock period after sequential optimization, in some cases even worse than what could be achieved without any timing optimization at all. Our research addresses this problem. Instead of simply trying to correct the results of combinational timing analysis, however, we attempt to fully exploit the potential of in-place retiming and clock skew scheduling.

Figure 3(a) illustrates a placement generated using a traditional QP placement tool based on GORDIAN [2], with the set of paths that are limiting the post-sequential optimization timing highlighted. Figure 3(b) shows the same circuit placed using a timing-driven placer which uses combinational static timing analysis. Figure 3(c) shows again the same circuit placed using CAPO, a state-of-the-art placement tool developed at UCLA [14].

In these three cases, the tools have done a visibly suboptimal job of minimizing the wire segments that are timing-critical; many

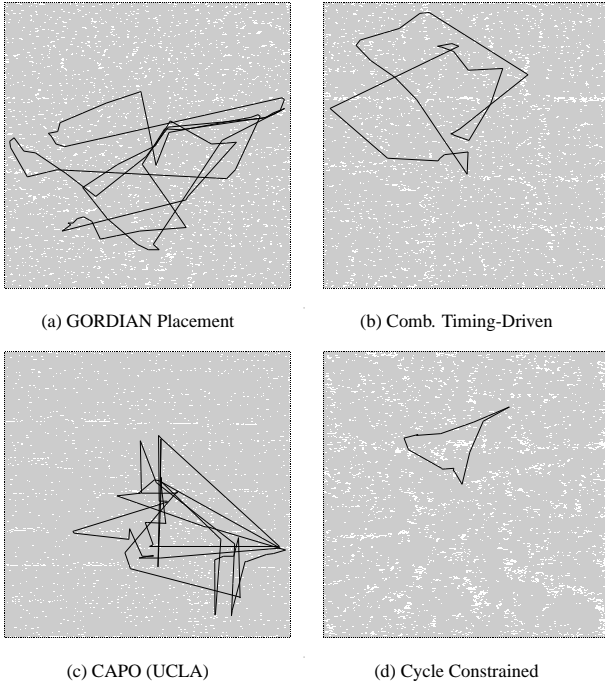


Figure 3: Placements for design ind08. Critical cycles shown in black; standard cells in gray.

of them nearly cross the width of the die. Even with register movement through in-place retiming, it is not likely that the severe mislocation of the critical registers in this particular layout could be fully corrected. In contrast, Figure 3(d) is the resulting placement from our own sequential placement tool. The critical elements are much more localized, and the contribution of interconnect delay to the clock period is dramatically reduced. In this example, the integration of sequential timing analysis into the physical design process has significantly boosted the utility of clock skew scheduling or in-place retiming.

3 Sequential Timing Analysis

Given a circuit with timing information, we are interested in the minimum clock period achievable under sequential optimization to evaluate its fitness, and to characterize the sequential criticality of each component. There are three phases of this analysis: construction of the sequential timing graph, identification of the minimum feasible clock period, and the determination of sequential slacks.

3.1 Constructing the Sequential Timing Graph

The sequential timing graph $G = (V, E)$ is extracted from a circuit as follows. Take V to be the registers of the design, together with an additional vertex v_{ext} representing the primary inputs and outputs. Add an edge (u, v) to E iff there exists a timing path from u to v in the original circuit. Every edge $e = (u, v)$ is labeled with $d(e)$, the maximum delay between u and v in the circuit. During placement, the estimated wire delays are included in $d(e)$.

3.2 Finding the Critical Cycle

Finding the critical cycle is equivalent to computing the *maximum mean cycle* (MMC) for G , which is given by $\max_{\ell \in C} \sum_{e \in \ell} d(e) / |\ell|$, where C is the set of all cycles in G . The MMC is equal to the minimum clock period which may be obtained

using unconstrained clock skew scheduling. In our implementation, we use Howard’s algorithm [15] to compute the MMC. [16] suggests that Howard’s algorithm is the fastest known algorithm for computing MMC. Our own empirical observations concur with those of [16].

The general idea behind Howard’s algorithm is to maintain a small set of edges π which starts as an initial guess of the critical cycle in the graph. The set π then has edges added and removed iteratively to monotonically increase the delays seen at the nodes in its induced subgraph (i.e. the subgraph obtained by discarding all edges except those edges in π). When the delays in the induced subgraph are maximized and can no longer be increased by changing π , then we know that π must contain the critical cycle, and thus we obtain the MMC. More details and a proof of correctness of Howard’s algorithm can be found in [15].

Note that not only is the MMC a lower bound on the clock period for the design, but also that it is always possible to find a clock skew schedule for the registers which achieves the MMC, by linear programming duality. We therefore use the MMC as a metric to evaluate the fitness of a design, since we are assured that, given the freedom of clock skew scheduling, the final clock period will be equal to the MMC.

3.3 Assigning Sequential Criticality

Once the critical cycle has been identified, the relative sequential criticality of the other vertices can be determined. We use a variant of the sequential timing analysis proposed in [1]. There, the concepts of *sequential arrival and required times* and *sequential slack* are presented. Given a target clock period of ϕ , the sequential arrival and required times at all vertices $v \in V$ with respect to a reference vertex v_{ref} can be computed from equations (1) through (3) using a modified version of the Bellman-Ford algorithm.

$$A_{seq}(v, v_{ref}) = \max_{(u,v) \in E} A_{seq}(u, v_{ref}) + d((u, v)) - \phi \quad (1)$$

$$R_{seq}(v, v_{ref}) = \min_{(v,w) \in E} R_{seq}(w, v_{ref}) - d((v, w)) + \phi \quad (2)$$

$$A_{seq}(v_{ref}, v_{ref}) = R_{seq}(v_{ref}, v_{ref}) = 0 \quad (3)$$

The sequential slack is $S_{seq} = R_{seq} - A_{seq}$.

A_{seq} and R_{seq} represent respectively the earliest and latest relative position in time to which a register can be moved (by retiming or clock skewing) while still meeting timing with respect to the reference point. S_{seq} measures the feasible range of temporal positions for v relative to the reference node. Intuitively, sequential slack represents a metric for quantifying sequential criticality, but we note that this criticality is only with respect to the given v_{ref} . A different choice of v_{ref} will impose different constraints. Figure 4 shows an example where the given choice of v_{ref} gives an incorrect value for the criticality of other vertices in the graph.

We define the *true sequential slack* as

$$S_{true}(v) = - \max_{(u,v) \in E} A_{seq}(u, v) + d((u, v)) - \phi \quad (4)$$

$S_{true}(v)$ gives the true sequential flexibility of v . However, in practice, computing S_{true} can be prohibitively expensive; using the above equations, this is equivalent to the all-pairs longest path problem on the sequential timing graph. Thus we use the ordinary sequential slack S_{seq} to approximate S_{true} as an estimate of criticality. Note, though, that the potential difference between S_{true} and S_{seq} can be large; we must choose v_{ref} carefully to minimize the potential error. [1] takes $v_{ref} = v_{ext}$, but this seems to be a poor choice,

as it may completely miss the actual critical cycle. The nodes most likely to impose tight constraints on other nodes are those that are themselves highly constrained, i.e. nodes on the critical cycle itself. We thus use $S_{seq}(v, v_{ref}) \mid v_{ref} \in \text{CRITICALCYCLE}(G)$ to measure sequential criticality.

4 Placement Driven by Sequential Timing

We introduce a placement algorithm that uses sequential timing information to maximize the potential of post-placement retiming or clock skew scheduling. The general procedure outlined in Algorithm 1 involves three phases: sequential timing analysis, the assignment of weights based on sequential criticality, and the introduction of explicit cycle constraints. The algorithm is to some measure independent of the method used to generate placements; the ability to weight nets and to include inequality constraints are the only specific requirements.

Algorithm 1 Sequential Slack Weighting

- 1: sequential timing analysis
 - 2: assign net weights $w(i)$
 - 3: partition $P_0 \leftarrow \text{allcells}$
 - 4: **while** $\exists P(|P| > m)$ **do** {GORDIAN main loop}
 - 5: solve global constrained QP
 - 6: bipartition all P where $|P| > m$
 - 7: solve final global constrained QP
 - 8: (optional) do placement with cycle constraints (Algorithm 3)
 - 9: legalize placement into rows
-

We have implemented a modified version of GORDIAN [2]. In this procedure, phases of global optimization are interleaved with bipartitioning. A quadratic programming (QP) problem is constructed to minimize the total weighted quadratic wirelength

$$\sum \alpha_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2]$$

subject to a set of linear constraints. This problem is solved for the entire chip, and the positions of all cells are updated. Based on this information, the cells in every subregion that contains more than m members are bipartitioned to minimize the total number of wires across the cut and maintain reasonably balanced halves.

We utilize two different partitioning techniques. At the topmost levels, where the partitioning is coarse and the information from the QP solution is less useful to guide the partitioning, we use hMetis [17] to partition the hypergraph without regard to geometry. For finer divisions, we choose a cut-minimizing spectral partition based on the QP solution, similar to the technique described in [18].

The coordinates of the center of each subregion are computed and a linear center-of-gravity (COG) constraint is imposed on its members. The QP is updated to include these new constraints and

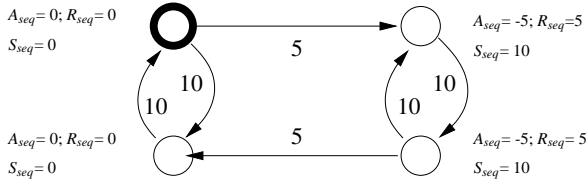


Figure 4: An example of a register timing graph with sequential slacks S_{seq} computed with respect to the solid vertex. The true sequential slack S_{true} for every node is zero; the illustrated result does not offer an accurate picture of sequential criticality.

the global optimization is repeated. GORDIAN is ideally suited to the requirements described above; nets can be easily weighted in both the global optimization and bipartitioning phases, and additional constraints can be imposed on the solution of the QP.

4.1 Sequential Slack Weighting

Each net is assigned a weight proportional to its relative sequential criticality. This is done to give priority to minimizing the lengths of the most critical wires as they are the most likely ones to limit the achievable clock period. After the sequential timing analysis described in Section 3, we have a function S_{seq} that gives an approximation of the sequential flexibility at each timing point; this is the inverse of sequential criticality. We use the following equation to compute the net weight $w(i)$:

$$w(i) = 1 + \frac{\beta}{\gamma + S_{seq}(i)/\phi}$$

The constants β and γ are chosen to tune the distribution of weights between the most and least critical nets. This is then applied to every connection α_{ij} , in addition to scaling based on fanout.

This weighting alone is enough to produce layouts with improved sequential timing characteristics, but its limitations should be recognized. Like their combinational counterparts, sequential slacks are inherently incompatible. Also, without computing the true sequential slacks, the problems described in Section 3.3 can also arise. Both of these problems can be solved with the introduction of cycle constraints. Our iterative algorithm to handle these constraints helps ensure that we catch all critical cycles.

4.2 Explicit Cycle Constraints

Assuming complete flexibility in assigning skew to all registers, for a cycle ℓ in the circuit to satisfy a target clock period ϕ , we must have

$$t_g(\ell) + t_w(\ell) \leq |\ell|\phi$$

where $t_g(\ell)$ is total intrinsic gate delay around ℓ and $t_w(\ell)$ is the total wireload delay around ℓ .

Suppose we have an existing placement P' in which the above constraint is violated. Let $t'_w(\ell)$ be the wireload delay around ℓ for P' , and let $d(\ell)$ be the total delay around ℓ for P' . Then we have

$$t_g(\ell) + t'_w(\ell) = d(\ell) \quad \text{and} \quad \frac{t_w(\ell)}{t'_w(\ell)} \leq \frac{|\ell|\phi - t_g(\ell)}{d(\ell) - t_g(\ell)} \triangleq \mu(\ell)$$

which defines the *wireload delay reduction factor* $\mu(\ell)$ necessary for ℓ to have a valid clock skew schedule for the target period.

Let $(\mathbf{x}', \mathbf{y}')$ be the locations of the cells for the given placement P' . We wish to derive a new placement $P = (\mathbf{x}, \mathbf{y})$ which satisfies the above given delay constraints. As an approximation, we take the wire delay for a cycle as being proportional to the sum of the squared Euclidean distances between cells in that cycle. That is,

$$t_w(\ell) = \eta \sum_{(u,v) \in \ell} (x_u - x_v)^2 + (y_u - y_v)^2$$

where η is a constant. Thus the physical placement constraints are

$$\frac{\sum_{(u,v) \in \ell} (x_u - x_v)^2 + (y_u - y_v)^2}{\sum_{(u,v) \in \ell} (x'_u - x'_v)^2 + (y'_u - y'_v)^2} \leq \mu(\ell) \quad (5)$$

The denominator in inequality (5) as well $\mu(\ell)$ are completely determined from the given placement and timing information. Thus inequality (5) contains only quadratic terms in (\mathbf{x}, \mathbf{y}) . Also note that these constraints are convex.

We justify approximating total wire delay with the sum of square Euclidean distances by our use of an iterative algorithm to solve the constrained system. We aim to make only small changes to the layout during each iteration, so that any error in this approximation can be subsequently corrected. Details may be found below.

4.2.1 Lagrangian Relaxation

To realize the placement constraints, we use *Lagrangian relaxation*, a standard technique for converting constrained optimization problems into unconstrained problems. For brevity, we only present a simplified description of this approach here. More information about Lagrangian relaxation can be found in [19, 20, 10].

Let $f(\mathbf{x}, \mathbf{y})$ be the sum of square wirelengths over all wires in the design for the placement (\mathbf{x}, \mathbf{y}) . Recall that the classical analytic placement formulation is simply the unconstrained problem $\min_{\mathbf{x}, \mathbf{y}} f(\mathbf{x}, \mathbf{y})$. Our constrained problem is then

$$\min_{\mathbf{x}, \mathbf{y}} f(\mathbf{x}, \mathbf{y}) \quad \text{such that} \quad \mathbf{g}(\mathbf{x}, \mathbf{y}) \leq \mathbf{0} \quad (6)$$

where the vector \mathbf{g} represents the placement constraints. For each cycle in the design, there is a single element in \mathbf{g} which corresponds to the constraint inequality (5) for that cycle. We create the *Lagrangian* $L(\mathbf{x}, \mathbf{y}, \mathbf{k}) = f(\mathbf{x}, \mathbf{y}) - \mathbf{k} \cdot \mathbf{g}(\mathbf{x}, \mathbf{y})$, where \mathbf{k} is a vector of *Lagrangian multipliers*; \mathbf{k} can be thought of as “penalties” which serve to increase the value of the cost function whenever a constraint is violated. The *Lagrangian dual problem* is

$$\max_{\mathbf{k} \geq \mathbf{0}} \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k}) \quad (7)$$

Our interest in the dual problem lies in the fact that, for convex problems such as ours, a solution for (7) corresponds directly to a solution for the original problem (6). We use the standard technique of *subgradient optimization* to solve the dual; see Algorithm 2.

Algorithm 2 Subgradient Optimization For Lagrangian Dual

- 1: $\mathbf{k} \leftarrow \mathbf{0}$
 - 2: $\mathbf{x}, \mathbf{y} \leftarrow \arg \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$
 - 3: **while** KKT conditions are not satisfied **do**
 - 4: $\mathbf{k} \leftarrow \max(\mathbf{0}, \mathbf{k} + \gamma \cdot \mathbf{g}(\mathbf{x}, \mathbf{y}))$
 - 5: $\mathbf{x}, \mathbf{y} \leftarrow \arg \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$
-

Note that for a fixed \mathbf{k} , $\min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$ can be solved as an ordinary unconstrained quadratic program. Subgradient optimization works by starting with an initial arbitrary \mathbf{k} , solving the resulting unconstrained QP, then adjusting \mathbf{k} based on the violated constraints which are found. If a constraint is violated, the corresponding penalty in \mathbf{k} is increased, so that subsequent iterations will move to reduce the violations, since the objective function, including the penalty terms, is to be minimized. Heuristics are available to determine an appropriate step size γ to adjust \mathbf{k} ; e.g. [10, 19]. The *Karush-Kuhn-Tucker* (KKT) conditions for stopping the algorithm are described fully in [19]. Roughly speaking, the procedure stops once the penalty multipliers grow large enough to force all constraint violations to zero.

Of course, the design may have many cycles, and thus there may be many constraints involved. We propose an iterative technique, given in Algorithm 3, which reduces the number of cycles under consideration by ignoring non-critical cycles.

In each iteration, we add the critical cycles found in the current placement to the constraint set S . A clock period T is chosen which we use as a target period for determining the cycle constraints for S . T is decreased slowly from T_c , the feasible clock period for the current placement, down to T_f , the final overall target clock period

Algorithm 3 Placement Using Cycle Constraints

- 1: input: an initial placement
 - 2: $T_c \leftarrow$ current MMC, $S \leftarrow \{\text{critical cycles}\}$
 - 3: **while** $T_c > T_f$ **do**
 - 4: choose target clock period $T, T_f \leq T < T_c$
 - 5: **for all** cycles $\ell \in S$ **do**
 - 6: add cycle constraint for ℓ with target T
 - 7: remove all cells in S from COG bins
 - 8: solve QP with cycle constraints (ALGORITHM 2)
 - 9: reassign all cells in S to nearest COG
 - 10: solve QP with cycle constraints (ALGORITHM 2)
 - 11: $T_c \leftarrow$ current MMC, $S \leftarrow S \cup \{\text{critical cycles}\}$
-

for the design. A slow adjustment of T helps ensure that we do not overconstrain the current constraint set S while ignoring other cycles. That is, we do not wish to “squeeze too hard” on those cycles which are currently critical, as this may cause some other cycle not under consideration to violate its timing constraint. Also, as noted before, we wish to perturb the placement only by small amounts, so that any error in our quadratic approximation of the wire delays can be corrected.

A significant benefit to using the iterative technique proposed in Algorithm 3 is that we are able to correct errors in our estimate of the true sequential slack so that subsequent iterations may have a better estimate of the sequential flexibility of each gate. Recall from Section 3.3 that we use $S_{seq}(v, v_{ref})$ to approximate $S_{true}(v)$, the true sequential slack, by choosing v_{ref} to be a vertex on the critical cycle. As the set of critical cycles tends to change with each iteration, this helps to ensure that we compute S_{seq} with respect to several different choices of v_{ref} , so that if we mistakenly identify a critical vertex as non-critical, we will likely correct the mistake in subsequent iterations. In contrast, [1] always takes $v_{ref} = v_{ext}$, and so has no opportunity to correct such errors.

COG constraints are commonly used with analytic placement techniques to ensure that the cells are spread out relatively evenly over the entire die area. We also wish our constrained placement to be appropriately spread out over the die area, but we do not wish the COG constraints to overconstrain our solution. We approach this problem using Steps 7–10 in Algorithm 3, which allows critical cells to “migrate” to appropriate locations on the die to avoid violation of timing constraints.

As a practical point, we also introduce cycles which are near-critical during each iteration, instead of only the critical cycles, to help reduce the number of iterations performed. Also, the main loop is terminated whenever either of the constrained QPs indicate that the problem may have become overconstrained, as no further improvement becomes possible in such case.

Our approach shares some similarity with that of [10], which also uses Lagrangian relaxation in an analytic placement framework to resolve timing constraints. However, there are several key differences between our work and that of [10]. First, and most important, is that we deal with the cyclic timing constraints which arise during clock skew scheduling, rather than simply path constraints. Second, we do not use the analytic placement step itself to perform timing analysis. The practical effect of this is twofold: our approach allows us to use general, nonlinear (and nonconvex) wire delay models, and we also do not encounter the degeneracy problems inherent in the constraints which come from timing analysis, as mentioned in [10]. Finally, we enjoy much greater computational efficiency, as our Lagrangian function can be seen as sim-

ply augmenting the weights of edges between cells by \mathbf{k} . Solving the Lagrangian dual for fixed \mathbf{k} requires no more computation than solving an unconstrained QP for our circuit.

4.3 Row Legalization

We use a greedy approach to detailed placement and legalization of the standard cells into rows. Such an approach has the prime benefit of speed. However, instead of direct minimization of wirelength as the search goal, as is typical of most other placement tools, our legalization technique instead seeks to minimize the total perturbation of the final placement with respect to the solution of the last QP obtained during placement (Step 7 of Algorithm 1, or Step 10 of Algorithm 3). We do this because we wish our placement to be timing-aware. Since the placement obtained by solving the QP respects the timing requirements of the circuit, we wish to deviate from such an ideal solution as little as possible.

Algorithm 4 Standard Cell Row Legalization

```

1: input: a placement solution from QP
2: Sort cells by their y-axis
3: Place cells into nearest rows with overflow into adjacent rows
4: for all rows  $R$  do
5:   Solve LP for  $R$  to minimize perturbation from QP
6: while not done do
7:   Sort cells in decreasing order of perturbation
8:   for all cells  $c$  do
9:     Move  $c$  to minimize perturbation
10:  for all rows  $R$  do
11:    Solve LP for  $R$  to minimize perturbation

```

Algorithm 4 outlines our legalization technique. We first find an initial legal placement solution by putting cells into the nearest rows. Cells are spilled into adjacent rows wherever row capacities are exceeded. Then, for each row, a linear program is formulated and solved to obtain the placement of each cell in the row. The cost represents the sum of the displacements of each cell from its ideal location (as given by the QP solution), while constraints are added to forbid overlap of adjacent cells within the row.

Once the initial legalized solution is found, we then proceed to make individual cell moves to improve the solution greedily. The cells are sorted in order of descending perturbation from the QP solution; this helps ensure that cells which stand the most to gain are moved first. For each cell, we determine a legal nonoverlapping placement location which minimizes the perturbation from the QP solution, and move the cell to that location. After all cells are moved in this fashion, we compact all rows using the same linear programming technique used to obtain the initial solution. This process is iterative; empirically we find only a small fixed number of iterations is required before most cells find a stable location.

5 Experiments

We ran our design flow on a set of thirteen industrial benchmark circuits as well as fourteen synchronous designs freely available for academic use, including the largest designs from the ISCAS89 benchmark suite. The academic circuits were technology mapped using an industrial synthesis tool into an arbitrary library from the industrial benchmarks.

The industrial libraries provided with the designs used interpolated lookup-table based models to characterize the cells. Both capacitive load and slew rate dependencies were incorporated in our

timing model. The design technology files gave the electrical characterization for the wires; in all cases, we assumed the use of metal layer 3 for routing. We used the half-perimeter bounding box metric as our estimate of the wirelength, noting that our algorithms are actually independent of the wireload estimation technique used, unlike other works, e.g. [10].

Currently, our placement tool can only handle single-row cells, so for the purpose of our experiments, it was necessary to convert larger circuit elements to single-row instances. Double-row cells were given a different aspect ratio, keeping the same area. Large macros were given an arbitrary size so as to fit in a single row. I/O pads were assigned randomly around the die perimeter.

Limitations in our timing analysis tool required some design changes to be made. Transparent latches were treated as ordinary registers, and combinational cycles were broken arbitrarily. Some hard macros did not have timing information associated with them, so for the purpose of timing analysis hard macros were treated as if they were I/Os for the overall circuit. Some designs used multiple clock domains. As we had no additional information regarding the relative phases and clock frequencies of such, we uniformly regarded the circuits as having only a single clock domain. We note, however, that the techniques described in this paper can be easily extended to multiple clock domains.

Our experimental results are shown in Table 1. The Size column indicates the number of placed instances for the design. The NW MMC column gives the MMC for the design when no wireload is taken into consideration. This is the minimum feasible clock period and serves as a lower bound on the post-placement timing, though any real placement with non-zero wirelengths will be greater. The REG MMC column shows the MMC achieved for a completely placed design using our placement flow with equal weights attached to the wires; effectively, this is a placement tool similar to GORDIAN. The COM MMC column shows the MMC achieved after placement using a combinational slack-based weighting function for the nets.

The SEQ MMC column indicates the MMC achieved after placement using the sequential slack-based weighting for the nets as described in Section 4.1. The percentage figure indicates the *reduction in wire delay* for the SEQ MMC result compared with the COM MMC result. We choose this as a better figure of merit than the absolute reduction in clock period, since no placer can ever hope to reduce the clock period below the no wireload MMC. The Run column indicates the run time for this algorithm, in seconds.

The CYCLE MMC column indicates the MMC achieved after placement using the cycle constraint technique described in Section 4.2, again with the percentage indicating reduction in wire delay compared to the combinational-weighted technique, and the Run column indicating the run time in seconds.

We also compare our tool against Capo, a leading-edge placer which focuses on wirelength minimization [14]. As the two placers have very different objectives, we certainly do not expect either one to be competitive in the other's problem domain. However, this comparison quantifies the benefit of using a sequential flexibility-aware placer, rather than choosing an placer which is best-suited for another task. The CAPO MMC column in Table 1 shows the MMC obtained after placement using Capo, the Run column indicates the run time for Capo in seconds, and the CYvsCA column indicates the percentage improvement in wire delay of our cycle constraint-based technique compared to the placement from Capo.

We show significant improvement in achievable clock period through application of our algorithm. For the industrial bench-

Design	Size	NW		REG		COM		SEQ			CYCLE			CAPO	CYvs CA%
		MMC	MMC	MMC	MMC	MMC	%	Run	MMC	%	Run	MMC	Run		
simon	1112	1.17	1.43	1.31	1.30	7.1	3	1.30	7.1	29	1.41	6	45.8		
s13207	2179	0.87	1.63	1.28	1.29	-2.4	7	1.29	-2.4	34	1.73	8	51.2		
s15850	2496	1.62	2.52	2.32	2.25	10.0	8	2.17	21.4	106	2.36	9	25.7		
dsip	3653	0.74	1.09	1.17	1.06	25.6	9	1.06	25.6	80	1.12	25	15.8		
diffeq2	4463	1.55	2.04	1.89	1.79	29.4	15	1.79	29.4	28	2.01	27	47.8		
tseng	4579	1.55	1.81	1.82	1.75	25.9	15	1.75	25.9	29	1.75	27	0.0		
bigkey2	6117	0.67	1.25	1.29	1.29	0.0	24	1.28	1.6	287	1.29	36	1.6		
s38584	6903	0.74	2.38	2.35	2.18	10.6	34	2.17	11.2	85	2.33	35	10.1		
s38417	7544	0.74	1.06	1.03	1.13	-34.5	43	1.09	-20.7	265	1.05	35	-12.9		
s35932	8871	0.62	2.08	1.86	1.87	-0.8	47	1.87	-0.8	113	2.00	32	9.4		
80386	9144	2.89	4.26	3.96	3.81	14.0	73	3.81	14.0	152	4.19	74	29.2		
viper	10212	2.51	3.44	3.51	3.30	21.0	66	3.27	24.0	135	3.24	69	-4.1		
elliptic3	10591	1.34	1.70	1.69	1.65	11.4	58	1.62	20.0	127	1.69	84	20.0		
clma	25157	4.16	6.59	6.88	6.50	14.0	241	6.50	14.0	309	6.19	132	-15.3		
Average % Improvement, Academic Designs								9.4			12.2			16.0	
ind01	6038	3.55	4.30	3.97	3.80	40.5	39	3.80	40.5	45	4.12	24	56.1		
ind02	12800	3.36	11.19	7.04	7.72	-18.5	119	7.72	-18.5	212	8.75	56	19.1		
ind03	13633	13.55	19.19	18.55	15.55	60.0	129	15.33	64.4	145	15.94	83	25.5		
ind04	26503	1.06	1.56	1.51	1.42	20.0	469	1.42	20.0	549	1.45	192	7.7		
ind05	27518	5.32	11.04	11.23	10.49	12.5	332	10.51	12.2	786	9.55	140	-22.7		
ind06	41077	3.75	5.17	4.75	4.55	20.0	964	4.65	10.0	3014	4.95	295	25.0		
ind07	52751	5.67	6.04	5.79	5.80	-8.3	1523	5.74	41.7	1689	5.88	402	66.7		
ind08	56861	1.37	1.79	1.64	1.57	25.9	1438	1.56	29.6	2671	1.74	365	48.6		
ind09	101884	7.65	12.95	11.45	10.82	16.6	7032	10.82	16.6	9670	11.47	999	17.0		
ind10	116921	4.20	4.79	4.45	4.39	24.0	8489	4.39	24.0	9237	4.65	896	57.8		
ind11	137536	4.27	11.54	8.57	8.53	0.9	8119	8.53	0.9	10875	8.73	1087	4.5		
ind12	140729	7.88	10.53	9.66	9.53	7.3	8826	9.53	7.3	9272	10.09	1133	25.3		
ind13	152066	5.08	5.43	5.31	5.18	56.5	8422	5.18	56.5	10153	5.24	1695	37.5		
Average % Improvement, Industrial Designs								19.8			23.5			28.3	

Table 1: Experimental Results

marks, we achieved an overall improvement in wire delay of 23.5% over a combinational slack-weighted placement technique, and 28.3% improvement over the results of Capo.

Note that overall the benefits of using the cycle constraints, compared to simply using the sequential slack-weighting heuristic, was lower for the academic benchmarks than for the industrial circuits. Our observation is that, for smaller circuits, adding cycle constraints tends to draw the cells close together, causing significant cell overlap in the QP solution. Such heavy overlap makes legalization more difficult, and all of the performance gains made from drawing the critical cycles close together are lost when legalization spreads the overlapping critical cells apart. We have added some simple heuristics to halt the addition of cycle constraints whenever sufficient overlap is seen. However, more work needs to be done in this area.

We note that such excessive cell overlap tends to happen more often with small designs rather than large ones. We conjecture that large designs tend to have more I/O pins on their periphery, which gives rise to more spreading of cells in the QP solution. One may therefore wish to avoid the use of cycle constraints, and only utilize the sequential slack weighting heuristic for small designs.

The wirelength-optimizing version of our tool was 10.7% worse in total wirelength compared to Capo, a typical result for an analytic placer. With the addition of cycle constraints, an 18.8% increase was measured over the wirelength-optimizing implementation. One certainly expects that a final layout which meets the timing constraints of the design will have longer wirelength than a layout which is done purely with minimization of wirelength as a goal. The key feature is that even with this wirelength penalty, the timing still improves. We also note that our tool currently does very little to control the total wirelength of the final placed design. As there

are many nets in the design which are not critical, there is much opportunity for us to further reduce wirelength, especially during row legalization. Recall our legalization procedure seeks only to minimize the perturbation of the QP solution. For non-critical sections of the design, it makes sense to ignore the QP solution and focus on the traditional approach of wirelength minimization instead. Additionally, more modern partitioning techniques, such as those found in Capo, can replace our existing partitioning algorithm we use, especially at the coarsest levels of partitioning where the information provided by the QP solution is limited.

Run times are measured in CPU seconds on a 2GHz Pentium 4 processor. Although it may seem that our run times are significantly worse than the excellent Capo tool, one must note that our design flow includes timing analysis and additional physical constraints for performance optimization, which are completely lacking in Capo.

6 Conclusions

Retiming and clock skew scheduling offer significant opportunities for improving design performance, but current physical design tools do not yet exploit these techniques to their fullest potential. By optimizing the placement of the most sequentially critical components, we have demonstrated that it is possible to significantly improve the post-sequential optimization timing.

Another benefit from sequential-driven placement is that the aggressive reduction of interconnect length in the most sequentially critical cycles will result in their spatial localization. This greatly simplifies the complexity of the distribution problem for clocks with multiple skews. Table 2 suggests that most registers will not even require any skewing. In a traditional placement, the relative locations of the most critical combinational paths are unconstrained

Clock Offset	Fraction of Registers
20% or more	0.04
10% to 20%	0.01
5% to 10%	0.01
2% to 5%	0.01
-2% to 2%	0.90
-2% to -5%	0.00
-5% to -10%	0.00
-10% to -20%	0.01
-20% or less	0.03

Table 2: Clock skews necessary to implement MMC times in Table 1 as a percentage of T_{period} (academic designs)

and can be uniformly distributed across the chip, thereby necessitating the need for a clock network with tightly controlled skew across the die. Since the registers with zero or near-zero slack will be grouped, the effort to distribute of multiple accurate clock domains can be concentrated on this region.

Much can be done to improve our current tool. As noted before, addressing total wirelength is an important consideration and we have already formulated several ways to approach this challenge. The weighting of nets by their criticality is admittedly somewhat ad hoc, so we may use the ideas of [21] to provide a stronger mathematical basis for our weighting function. Implementation improvements will allow us to run on larger designs and to present more datapoints by which to judge our work; runtime will also be improved. Future work includes questions about running the algorithm on partitioned designs, addressing implementation limitations of retiming and clock skew scheduling, and extending the sequentially-aware timing model to other aspects of the synthesis flow.

7 Acknowledgments

The authors would like to thank Robert K. Brayton and Christoph Albrecht for helpful discussion and advice about critical cycles and sequential slack. This work was funded in part by the MARCO Focus Center for Circuit & System Solutions (C2S2, www.c2s2.org), under contract 2003-CT-888.

References

- [1] J. Cong and S. K. Lim, "Physical planning with retiming," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, CA), pp. 1–7, November 2000.
- [2] J. M. Kleinhans, G. Sigl, and F. M. Johannes, "GORDIAN: A global optimization / rectangle dissection method for cell placement," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 506–509, November 1988.
- [3] C. Leiserson and J. Saxe, "Optimizing synchronous systems," *Journal of VLSI and Computer Systems*, vol. 1, pp. 41–67, January 1983.
- [4] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.
- [5] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, pp. 945–951, July 1990.
- [6] S. Held, B. Korte, J. Maßberg, M. Ringe, and J. Vygen, "Clock scheduling and clocktree construction for high-performance ASICs," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, California), pp. 232–239, November 2003.
- [7] I. S. Kourtev and E. G. Friedman, "Synthesis of clock tree topologies to implement nonzero clock skew schedule," in *IEE Proceedings on Circuits, Devices, Systems*, vol. 146, pp. 321–326, December 1999.
- [8] J. G. Xi and W. W.-M. Dai, "Useful-skew clock routing with gate sizing for low power design," *J. VLSI Signal Process. Syst.*, vol. 16, no. 2-3, pp. 163–179, 1997.
- [9] A. K. K. Ravindran and E. Sentovich, "Multi-domain clock skew scheduling," in *Proceedings of the 21th International Conference on Computer Aided Design*, ACM, 2003.
- [10] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance-driven placement algorithm," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 825–839, November 1992.
- [11] J. Cong and X. Yuan, "Multilevel global placement with retiming," in *Proceedings of the 40th ACM/IEEE Design Automation Conference*, pp. 208–213, 2003.
- [12] C.-Y. Yeh and M. Marek-Sadowska, "Delay budgeting in sequential circuit with applications on FPGA placement," in *Proceedings of the 40th ACM/IEEE Design Automation Conference*, pp. 202–207, 2003.
- [13] D. P. Singh and S. D. Brown, "Integrated retiming and placement for field programmable gate arrays," in *Proceedings of the ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pp. 67–76, 2002.
- [14] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements?," in *ACM/IEEE Design Automation Conference*, pp. 477–482, 2000.
- [15] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," in *Proceedings of the IFAC Conference on System Structure and Control*, July 1998.
- [16] A. Dasdan, S. S. Irani, and R. K. Gupta, "An experimental study of minimum mean cycle algorithms," Tech. Rep. UCI-ICS 98-32, University of Illinois at Urbana-Champaign, 1998.
- [17] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proceedings of the 34th Annual Design Automation Conference*, pp. 526–529, June 1997.
- [18] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design & Test Of Computers*, vol. 5, pp. 44–56, Dec. 1988.
- [19] P. M. Pardalos and M. G. Resende, eds., *Handbook of Applied Optimization*. Oxford University Press, 2002.
- [20] E. Golshtein and N. Tretyakov, *Modified Lagrangians and Monotone Maps in Optimization*. John Wiley and Sons, 1996.
- [21] R. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," in *Design Automation Conference*, pp. 620–625, 1991.