SPIRE: A Retiming-Based Physical-Synthesis Transformation System

David A. Papa♯‡ Smita Krishnaswamy† Igor L. Markov‡
iamyou@eecs.umich.edu skrishn@us.ibm.com imarkov@eecs.umich.edu
♯IBM Austin Research Lab, 11501 Burnet Rd., Austin, TX 78758
†IBM T. J. Watson Research Ctr., 1101 Kitchawan Rd., Yorktown Heights, NY 10598
‡University of Michigan, EECS Department, 2260 Hayward St., Ann Arbor, MI 48109-2121

***Abstract*—The impact of physical synthesis on design performance is increasing as process technology scales. Current physical synthesis flows generally perform a series of individual netlist transformations based on local timing conditions. However, such optimizations lack sufficient perspective or scope to achieve timing closure in many cases. To address these issues, we develop an integrated transformation system that performs multiple optimizations simultaneously on larger design partitions than existing approaches. Our system, SPIRE, combines physically-aware register retiming, along with a novel form of cloning and register placement. SPIRE also incorporates a placement-dependent static timing analyzer (STA) with a delay model that accounts for buffering and is suitable for physical synthesis. Empirical results on 45nm microprocessor designs show 8% improvement in worst-case slack and 69% improvement in total negative slack *after* an industrial physical synthesis flow was already completed.**

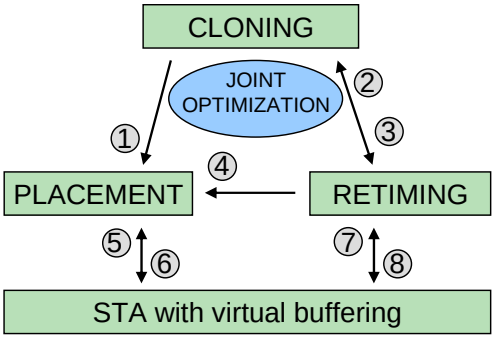
I. Introduction

A relatively recent addition to EDA, physical synthesis arose when wire delays began to significantly impact circuit performance. The physical synthesis process begins by computing a tentative cell placement and proceeds to restructure timing-critical paths. Traditional physical-synthesis flows in the industry [1], [16] apply a series of local, mostly greedy transformations such as inserting individual buffers on particular nets, or relocating individual gates in the limited context of their neighboring gates. Several iterations of such transformations may be required for timing closure [1], [16]. However, growing reliance on physical synthesis motivates the development of transformations that are more powerful in two specific ways.

- **Greater optimization scope:** the ability to effect larger changes in the circuit in terms of simultaneously moving or altering several objects in order to achieve timing closure.
- **Larger optimization window size:** the ability to consider temporal and spatial constraints from partitions of a design.

Increasing the optimization scope and window sizes can help avoid local minima in the solution space that trap individual, local transformations. Additionally, this circumvents the *ordering problem* of individual transforms, since different sequences affect results.

We facilitate more powerful optimizations through retiming. Unlike traditional gate- and net-centric timing optimizations that aim to satisfy given stage-timing constraints, retiming can optimize the constraints themselves to better fit a given netlist. Therefore, we propose a **S**ystem for **P**hysically-aware **I**ncremental **R**etiming and **E**nhancements, or *SPIRE*, that performs register-retiming with accurate delay models, buffering, placement, and logic cloning to seamlessly integrate retiming into physical synthesis. Key features of SPIRE are:



1:	CLONING changes the netlist and influences PLACEMENT
2:	RETIMING helps select combinational gates for CLONING
3:	CLONING creates new opportunities for RETIMING (see Fig. 2)
4:	RETIMING relocates netlist registers, causing new PLACEMENT
5:	PLACEMENT changes interconnect delays used in STA
6:	Register PLACEMENT after retiming is performed based on STA
7:	RETIMING relocates netlist registers, changing paths in STA
8:	STA computes min slack — the optimization goal for RETIMING

Fig. 1. Interactions in SPIRE's joint optimization.

- Multiple degrees of freedom to optimize the circuit, including *gate placement*, *register retiming*, and *gate cloning*.
- A mixed-integer linear programming (MILP) framework for joint optimization that emphasizes synergies between point optimizations as shown in Figure 1.
- An embedding of placement-dependent STA computations with virtual buffering into the MILP, which allows for efficient and accurate consideration of timing constraints from large design partitions.

SPIRE allows for placement, retiming, and cloning to simultaneously optimize a circuit, as shown in Figure 1. In physical synthesis, such a joint optimization problem is often considered intractable. Instead, one chains individual optimizations with limited scope. However, as shown in Figure 2, such *separation of concerns* overlooks opportunities for joint optimization. Therefore, we propose a powerful transformation that is computationally expensive, but can be applied to sizable circuit *windows*. Window sizes can be selected subject to runtime constraints imposed on the system. Our experimental results in Section IV, in fact, show that SPIRE can handle window sizes of thousands of gates by efficiently encoding the problem as an MILP with linearly many constraints in the size of the circuit.

Retiming methods based on [9] enforce timing constraints by requiring a register on every path whose delay exceeds a threshold. However, such methods require computationally-expensive path enumeration within the linear programming formulation. We avoid path enumeration by enforcing linearly many conditional STA-like constraints which determine optimal retiming and placement. Further, different choices for retiming, cloning and gate relocation perturb only a small set of local constraints directly (those affecting nearby edges). Aside from the system as a whole, we highlight the following contributions of this work:

- A method for retiming with an accurate STA-like embedded delay computation model.
- A novel gate-cloning technique to enable retiming.
- A simultaneous retiming and re-placement technique.

The remainder of this paper is organized as follows. Section II reviews background and notation. Section III presents our maximum-slack retiming formulation that incorporates STA, placement, and cloning. In Section IV, our methods are validated on a 45nm high-performance microprocessor against leading-edge physical synthesis tools. Section V outlines additional optimizations that can further increase the scope of SPIRE. Conclusions are drawn in Section VI.

II. Background, Notation and Objectives

In this section, we provide the necessary background in static timing analysis and period-constrained retiming.

A. Static Timing Analysis with Buffered Wires

SPIRE depends on the ability to encode timing constraints efficiently, and in such a way that they can be easily adjusted to accommodate changes resulting from circuit optimizations. Here, we describe the basics of the static timing analysis model we use, and delineate its assumptions.

In static timing analysis, a *timing graph* $G = (V, E)$ is extracted from a logic circuit [13]. Each vertex $v \in V$ is a timing point, and corresponds to an input or output pin of a gate (or a global input or output pin). A pair of vertices, $u, v \in V$, are connected by a directed edge $e(u, v) \in E$ if there is a timing relationship (i.e., a connection) between the pins u and v. This connection can occur within a gate, i.e., between an input pin and an output pin, or it can correspond to a wire connecting two gates. Each edge has an associated delay $\mathrm{delay}(u, v)$ indicating the delay between u and v.

To determine the worst path in the circuit, a topological traversal is performed on the graph beginning at the sources. The *actual arrival time* $\mathrm{AAT}(v)$ at a timing point v in the circuit is the latest arrival time of any of its predecessors after considering delay:

$$\mathrm{AAT}(v) = \max_{\{e(u,v)\}} (\mathrm{AAT}(u) + \mathrm{delay}(u, v)) \quad (1)$$

The *required arrival time* $\mathrm{RAT}(u)$ at a timing point u in the circuit is computed in a similar fashion, traversing backwards from the primary outputs of the circuit:

$$\mathrm{RAT}(u) = \min_{\{e(u,v)\}} (\mathrm{RAT}(v) - \mathrm{delay}(u, v)) \quad (2)$$

The timing graph is topologically traversed twice to find these values, after which the *slack* $\mathcal{S}(v)$ is found at every point v:

$$\mathcal{S}(v) = \mathrm{RAT}(v) - \mathrm{AAT}(v)$$

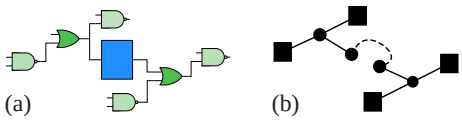


Fig. 3. A circuit (a) and its timing graph (b). Square objects have fixed AATs or RATs. STA is performed only on circular movable objects.

Delay models. Static timing analysis relies on compact models of gate and net delay, e.g., a look-up table to represent gate delays in terms of its inputs. Today, buffering is heavily used in physical synthesis to reduce wire delay and improve timing. Therefore, it is important to estimate buffered wire delay in accurate interconnect delay models. We model the delay along an ideally buffered net of length L as in [2], [12]:

$$\mathrm{delay}(L) = L(R_b C + R C_b + \sqrt{2 R_b C_b R C}) \quad (3)$$

Here, R_b and C_b are the intrinsic resistance and input capacitance of buffers. R and C are unit wire resistance and capacitance, respectively. Empirical results in [2], [12] indicate a 97% correlation between the results of this linear model and an industrial timing analysis tool.

In practice, this model can be reduced to a linear equation in terms of the length of wire with a technology-dependent constant. The routed wirelength between two points can be approximated by the *half-perimeter wirelength* (HPWL). We denote the HPWL between two points u and v as $\mathrm{HPWL}(u, v)$. Suppose that the net connecting (u, v) is bounded by the x and y coordinates, U_x, U_y, L_x and L_y, from above and below. Then

$$\mathrm{HPWL}(u, v) = (U_x - L_x) + (U_y - L_y) \quad (4)$$

In SPIRE, we approximate the delay between two points using an empirically determined, technology-dependent parameter τ. In order to calculate the best value for τ, we buffer a long net n and calculate

$$\tau = \mathrm{delay}(n)/\mathrm{HPWL}(n) \quad (5)$$

Then we calculate the delay between two points u and v as

$$\mathrm{delay}(u, v) = \tau \mathrm{HPWL}(u, v) \quad (6)$$

To compute the initial conditions for SPIRE, the RAT and AAT of all fixed timing points are generated by an STA engine using very accurate delay models and a set of timing assertions created by designers [10], [13]. SPIRE considers the timing of register's input pin fixed and uses an STA engine to determine its RAT value. Similarly, the AAT is fixed on output pin of a register. When calculating these values the STA engine includes considerations of setup and hold time, intrinsic gate delay and clock skew.

The timing metrics that we optimize include the minimum slack of all vertices $(\mathcal{M})$, the total negative slack in the circuit $(\mathcal{T})$, and the total slack below a threshold (Θ_T), computed as shown below. Note that $\mathcal{T} = \Theta_0$.

$$\mathcal{M} = \min_u \mathcal{S}(u) \quad (7)$$

$$\mathcal{T} = \sum_u \min(0, \mathcal{S}(u)) \quad (8)$$

$$\Theta_T = \sum_u \min(0, \mathcal{S}(u) - T) \quad (9)$$

In SPIRE, registers are allowed to move, while combinational gates remain *fixed* in place; this limitation is not inherent,

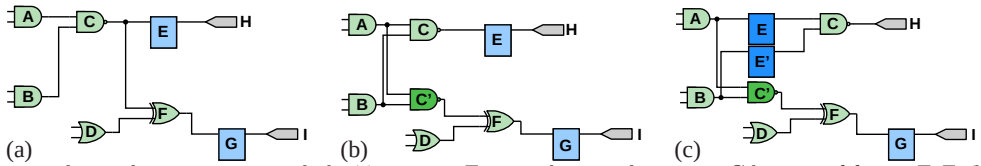


Fig. 2. Retiming and gate cloning to improve slack: (a) Register E cannot be moved past gate C because of fanout E-F. (b) If the NAND gate C is cloned, creating a new gate C' to drive its two sinks, it is possible to retime the top register without changing the logic function. (c) The final result with register E retimed.

as discussed in Section V. After gate cloning (Section III-D), the cloned gates can be physically relocated. For efficiency, we restrict our timing graph edges to those representing (1) each connection between the movable gates, and (2) each connection between a movable gate and a fixed gate. For the subcircuit in Figure 3(a), the resultant timing graph is shown in Figure 3(b).

B. Register Retiming

The original linear programming formulations for minimum-period and minimum-area retiming were developed by Leiserson and Saxe [9]. In their framework, a circuit is represented by a *retiming graph* $G(V, E)$, where each vertex $v \in V$ represents a combinational gate, and each edge $(u, v) \in E$ represents a connection between a driver u and sink v. An edge is labeled by a weight $w(u, v)$, indicating the number of registers (flip-flops) between u and v. The objective of minimum-area retiming is to determine labels $r(v)$ for each vertex v, denoting the number of registers that are moved from the outputs to the inputs of v, that minimize the sum of edge weights. The weight of an edge after retiming is given by:

$$w_r(u, v) = w(u, v) - r(u) + r(v) \tag{10}$$

Therefore, the total number of registers in the retimed circuit can be minimized in terms of the following expression.

$$\sum_{(u,v)\in E} w(u, v) - r(u) + r(v) \tag{11}$$

Additionally, retiming labels have to meet *legality* constraints, $w(u, v) \geq r(u) - r(v)$ for each edge, to enforce the fact that edges cannot have negative weights. A linear program for the minimum-area retiming problem is given in Figure 4. Leiserson and Saxe [9] observe that this problem is the dual of a min-cost network flow problem and can therefore be solved in polynomial time.

$$\begin{aligned}&\texttt{Minimize}\\&\quad \textstyle\sum_{(u,v)\in E} w(u, v) - r(u) + r(v)\\&\texttt{subject to}\\&\quad \forall (u, v) \in E, r(u) - r(v) \leq w(u, v)\end{aligned}$$

Fig. 4. An LP for minimum-area retiming.

As shown in Figure 5, the period can be constrained in this formulation by requiring weight ≥ 1 on every path between two vertices with delay exceeding target period P. However, this formulation requires $\Theta(|V|^2)$ constraints in the form of matrix D that stores the delay of the longest path between the vertices (u, v) in $D(u, v)$, and matrix W that stores the weight of that path. Then, a binary search is performed to determine the minimum achievable clock period. The feasibility of each period according to the legality constraints is checked using the Bellman-Ford algorithm [9].

$$\begin{aligned}&\texttt{Minimize}\\&\quad \textstyle\sum_{(u,v)\in E} w(u, v) - r(u) + r(v)\\&\texttt{subject to}\\&\quad \forall (u, v) \in E, r(u) - r(v) \leq w(u, v)\\&\quad \forall (u, v) \in E | D(u, v) > P, r(u) - r(v) \leq W(u, v) - 1\end{aligned}$$

Fig. 5. An LP for min-area, period-constrained retiming.

Prior work in retiming also includes the ASTRA [14] algorithm, which is a faster approach. It relates the problem of clock skew optimization at each flip-flop to a retiming solution for minimum-period retiming, and uses the Bellman Ford algorithm to derive the longest path. Recently, the authors of [17] used program derivation to automatically generate an algorithm for min-period retiming which iteratively shortens the longest clock periods. Retiming was also explored for slack budgeting and power minimization for FPGAs [5].

Challenges in min-period retiming. Algorithms based on techniques from [9] enforce timing constraints by requiring registers on gate-to-gate paths that exceed a length threshold. This involves computationally expensive enumeration of such paths. Therefore, in our approach we avoid path enumeration by using slack, rather than period as a metric. Slack constraints are linear in the size of the circuit and all path delays are implicitly encoded through the AAT and RAT constraints.

Other retiming algorithms use network-flow based approaches which are difficult to extend to a multi-objective optimization [14]. Using interconnect delays instead of lengths has been a challenge, as wires can be dynamically re-buffered when their lengths change [15]. Unlike much of past literature, we use a buffered delay model to account for this.

Inherent limitations of retiming are associated with multi-fanout branches. To move a register backward through a gate, all fanout branches of the gate must include (or share) a register, and all these registers must be retimed at once. This constraint ensures that the number of registers on any PI-to-PO path stays constant during retiming. Therefore, fanouts can be a bottleneck for retiming. In order to alleviate this problem, we clone gates within the retiming formulation so as to provide additional backward-movement opportunities for registers (see Figure 2).

III. Joint Optimization for Physical Synthesis

This section introduces the SPIRE system which combines several optimizations used individually in the past literature. As shown in Figure 6, combining retiming and placement is better than applying them individually. In this example, only the combined approach closes timing. The main difficulty in combining placement, cloning and retiming is their inter-dependence—optimal locations and cloned configurations depend on the timing constraints which are altered by retiming.

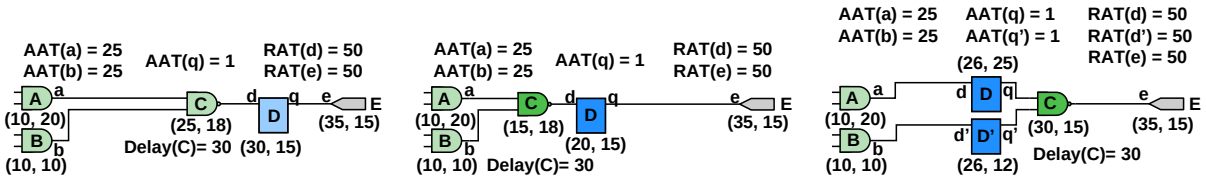


Fig. 6. Advantages of performance-driven retiming with simultaneous re-placement. In the original circuit (a), the timing path feeding the input of the register has negative slack. Moving the gate and register in (b) improves the slack, but movement alone does not allow the path to meet timing constraints. Only by retiming and movement can all timing constraints be met in (c).

A. Embedding the STA Backplane into ILP

In order to incorporate STA into SPIRE, we first encode the RAT and AAT variable computations into an MILP, with constraints corresponding to Equations 1 and 2, both of which are linear. Then, alternative constraints are introduced to analyze each timing arc, for the case where a register is between the source and sink of the arc. Figure 7 shows an LP simply for computing the worst-case slack. For circuit C with gates $G = \{u_1, u_2 \ldots u_n\}$, and registers $R = \{l_1, l_2, \ldots l_m\}$, the variables in this program are:

- AAT and RAT for each $u \in G$, denoted A_u, and R_u.
- $\mathcal{M}$ for the minimum slack.

In other words, for a gate u driven by $i_1, i_2, \ldots i_S$ the constraints to enforce A_u are shown below. Here $1 \leq j \leq S$:

$$A_u \geq A_{i_j} + \tau \text{HPWL}(i_j, u) + D_u \tag{12}$$

Since A_u must actually be equal to one of the values in Equation 12, it is added to the objective function so that it can be minimized. The constraints guarantee that it will be greater than any path's delay. Adding it to the objective guarantees that it will be no more than the greatest path delay. Similarly for R_u, supposing that u drives gates $o_1, o_2, \ldots o_T$, then the corresponding constraints are of the form for $1 \leq k \leq T$:

$$R_u \leq R_{o_k} - \tau \text{HPWL}(g, o_k) - D_u \tag{13}$$

We add $-\text{RAT}(u)$ to the objective function since this variable is maximized rather than minimized. The AAT and RAT of registers (and other end points like primary input and output pins) are simply set according to initial values obtained form the reference timing model. The term $-\mathcal{M}$ is added to the minimization objective. The total slack $\mathcal{T}$ can also easily be computed from the MILP and added as an objective. In practice, we minimize both. However, for brevity, we drop $\mathcal{T}$ from the MILP formulations for the remainder of the paper. Note that the number of constraints in this formulation is proportional to the number of 2-pin arcs in the circuit and not the number of paths. Further, *the number of constraints in which each gate and 2-pin connection appears is limited*, which is key to incorporating retiming, placement and cloning.

B. Max-Slack Retiming

Retiming is the most powerful optimization within SPIRE because it can effect drastic changes on the timing constraints. For instance, moving one register past a gate can allow *cycle stealing* on the order of gate delays along all paths that cross the register. In order to utilize the STA constraints described in the previous section, we develop a maximum slack formulation. The key idea in maximum-slack retiming is that there are two versions of the AAT and RAT computations on each vertex depending upon whether the vertex drives/is driven by a register. The constraints that are actually enforced are determined by the retiming. Therefore, the retiming program maximizes the worst-case slack.

Objective
Minimize : $-\mathcal{M}$
$+\forall(\mathrm{u})(\mathrm{A_u} - \mathrm{R_u})$

subject to
$\forall \mathrm{u} \;\; \mathcal{M} \leq \mathcal{S}(\mathrm{u})$
$\forall \mathrm{u} \forall(\text{fanins f of u})\mathrm{A_u} \geq \mathrm{A_f} + \tau\text{HPWL}(\mathrm{f}, \mathrm{u}) + \mathrm{D_u}$
$\forall \mathrm{u} \forall(\text{fanouts f of u})\mathrm{R_u} \leq \mathrm{R_f} - \tau\text{HPWL}(\mathrm{u}, \mathrm{g}) - \mathrm{D_u}$
$\forall \text{register r}, \mathrm{R_r} \geq \text{clock\_period}$
$\forall \text{register r}, \mathrm{A_r} \leq 0$

Fig. 7. Finding minimum slack using LP.

Figure 8 shows the MILP that combines the STA constraints with retiming. During retiming, we only know the contents of the *retiming graph* (not the timing graph), because any edge in the retiming graph can include a newly retimed register. Therefore, STA constraints change depending on the retiming variable values. However, there are only two possibilities for each retiming arc: either the arc contains a register after retiming, or it does not (and combinations of arcs are implicitly considered). This situation is modeled through IF-THEN logic based on the retimed weight of the edge. If the weight is greater than zero, then the wirelengths involved in RAT and AAT computations change to incorporate the newly retimed register. For brevity of presentation, we temporarily assume that the new register l will be placed at the *center of gravity (COG)* of the neighboring gates of l. Thus, the net connecting u to l has length $\text{HPWL}(u, \text{COG}(l))$ and the net connecting l to v has length $\text{HPWL}(\text{COG}(l), v)$. In the next section, we eliminate this simplification and consider the static timing analysis of nearby gates when calculating slack-optimal register locations.

$$\begin{aligned} &\textbf{if}(\mathbf{w_r}(\mathbf{u}, \mathbf{v}) == \mathbf{0}) \\ &\quad R_u \leq R_v - \tau\text{HPWL}(u, v) - D_u \\ &\quad A_v \geq A_u + \tau\text{HPWL}(u, v) + D_v \\ &\textbf{if}(\mathbf{w_r}(\mathbf{u}, \mathbf{v}) \geq \mathbf{1}) \\ &\quad R_u \leq R_l - \tau\text{HPWL}(u, \text{COG}(l)) - D_u \\ &\quad A_v \geq A_l + \tau\text{HPWL}(\text{COG}(l), v) + D_v \end{aligned} \tag{14}$$

This IF-THEN logic is incorporated into a linear program using the *big-M* formulation. Under this formulation, a constraint $v < k$ takes the form $v < k + Mv_I$, where M is a large constant. If $v_I == 0$, the constraint reduces to the original, if $v_I \neq 0$ then the constraint simply becomes a bound on the variable v, i.e., $v < Mv_I$. Alternatively, IF-THEN logic can be modeled using *indicators*—binary variables that turn constraints on and off.[1] In our program, we define an indicator $\text{hasReg}(u, v)$ as follows:

[1]Indicators are supported by the MILP solver CPLEX 12.1.

Objective

Minimize : $-\mathcal{M} + \sum_{(u,v)\in E}(K)\ \ w_r(u,v)$

subject to

$\forall(\mathrm{u},\mathrm{v}), \mathrm{r(u)} - \mathrm{r(v)} \leq \mathrm{w(u,v)}$

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{if(!hasReg(u,v))}$

$\quad \mathrm{R_u} \leq \mathrm{R_v} - \tau\mathrm{HPWL(u,v)} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{if(!hasReg(u,v))}$

$\quad \mathrm{A_v} \geq \mathrm{A_u} + \tau\mathrm{HPWL(u,v)} + \mathrm{D_v}$

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{if(hasReg(u,v))}$

$\quad \mathrm{R_u} \leq \mathrm{R_l} - \tau\mathrm{HPWL(u,COG(l))} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{if(hasReg(u,v))}$

$\quad \mathrm{A_v} \geq \mathrm{A_l} + \tau\mathrm{HPWL(COG(l),v)} + \mathrm{D_v}$

$\forall \mathrm{u} \in \mathrm{V},\ \ \mathcal{M} \leq \mathcal{S}(\mathrm{u})$

Fig. 8. Max-slack retiming with STA embedded.

Objective :

Maximize $\mathcal{L}$

subject to

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}) \in \mathrm{E_l},\ \ \mathrm{U_x^e} \geq \alpha_\mathrm{x}^\mathrm{u},\ \ \mathrm{U_y^e} \geq \alpha_\mathrm{y}^\mathrm{u}$

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}) \in \mathrm{E_l},\ \ \mathrm{L_x^e} \leq \alpha_\mathrm{x}^\mathrm{u},\ \ \mathrm{L_y^e} \leq \alpha_\mathrm{y}^\mathrm{u}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}) \in \mathrm{E_l},\ \ \mathrm{U_x^f} \geq \alpha_\mathrm{x}^\mathrm{v},\ \ \mathrm{U_y^f} \geq \alpha_\mathrm{y}^\mathrm{v}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}) \in \mathrm{E_l},\ \ \mathrm{L_x^f} \leq \alpha_\mathrm{x}^\mathrm{v},\ \ \mathrm{L_y^f} \leq \alpha_\mathrm{y}^\mathrm{v}$

$\forall \mathrm{e} \in \mathrm{E_l}, \mathrm{L_x^e} \leq \beta_\mathrm{x}^\mathrm{l} \leq \mathrm{U_x^e}$

$\forall \mathrm{f} \in \mathrm{E_l}, \mathrm{L_y^f} \leq \beta_\mathrm{y}^\mathrm{l} \leq \mathrm{U_y^f}$

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}) \in \mathrm{E_l}, \mathrm{R_u} \leq \mathrm{R_l} - \tau(\mathrm{U_x^e} - \mathrm{L_x^e} + \mathrm{U_y^e} - \mathrm{L_y^e}) - \mathrm{D_u}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}) \in \mathrm{E_l}, \mathrm{A_v} \geq \mathrm{A_l} + \tau(\mathrm{U_x^f} - \mathrm{L_x^f} + \mathrm{U_y^f} - \mathrm{L_y^f}) + \mathrm{D_v}$

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}) \in \mathrm{E_l},\ \ \mathcal{L} \leq \mathrm{R_u} - \mathrm{A_u} - \mathrm{D_u}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}) \in \mathrm{E_l},\ \ \mathcal{L} \leq \mathrm{R_v} - \mathrm{A_v} - \mathrm{D_v}$

Fig. 9. Optimal register location relative to adjacent gates.

$$\begin{array}{l} \mathbf{if}(\mathbf{w_r(u,v)} > \mathbf{0})\ \ \mathrm{hasReg}(u,v) = 1 \\ \mathbf{if}(\mathbf{w_r(u,v)} \leq \mathbf{0})\ \ \mathrm{hasReg}(u,v) = 0 \end{array} \tag{15}$$

This variable can be set in a variety of ways. One way is to use the constraint $\mathrm{hasReg}(u,v) \leq w_r(u,v)$ and maximize it. If $w_r(u,v) == 0$ then $\mathrm{hasReg}(u,v) = 0$. If $w_r(u,v) \geq 1$ then, since $\mathrm{hasReg}(u,v)$ is maximized, it is set to 1. However, maximization can sometimes conflict with the objective, therefore we use the following constraints instead:

$$\begin{array}{l} \mathrm{hasReg}(u,v) \leq w_r(u,v) \\ \mathrm{if}(\mathrm{hasReg}(u,v) == 0)\ \ w_r(u,v) = 0 \end{array} \tag{16}$$

The second constraint uses the $hasReg$ variable as an indicator. Together, these two constraints require that $\mathrm{hasReg} = 0$, *if and only if* $w_r(u,v) = 0$. For brevity, we omit the setting of this variable from our formulations. As we will see in Section III-D, maximization of real and integer variables can also fail when the objective has conflicting terms. Our formulation uses the constraints below to maximize general variables (without adding terms to the objective). We constrain the variable $C = \max(A,B)$ as follows:[2]

$$C \geq A, \quad C \geq B, \quad (C == A)||(C == B) \tag{17}$$

The $\min$ function is evaluated similarly. In Figure 8, the slack, RAT, and AAT variables are real values while the retiming variables have to be integer. We utilize a constant weighting factor K to reconcile area with slack. The constant K can be adjusted based on the available area.

Note that the formulation in Figure 8 does not require the derivation of the W or D matrices that were described in Section II. Instead, timing calculations are performed within the MILP. Thus, the number of constraints is only $O(|E|)$ for a retiming graph with edge set E.

C. Register Placement

Registers have special significance in a timing graph because their inputs are in a different clock cycle than their outputs. This facilitates *time borrowing* — the ability to shift delay from one timing path to another by decreasing the delay on inputs paths at the cost of increased delay on output paths, and vice versa. By physically relocating registers, the interconnect delay around registers can be allocated to either the input or output paths. In this section, we describe a formulation that integrates register placement with the retiming described in the previous section. Register locations alter STA constraints by changing interconnect length, and therefore, delays. On each edge with a register, SPIRE chooses the physical location that results in the best possible slack. The placement also interacts with retiming, which optimizies the STA constraints based on register locations for each edge.

In order to perform this integration, we utilize the same type of case-logic as in the previous section. First we modify constraints so that AATs and RATs on edges with registers are calculated with respect to the placement. Register sharing along adjacent edges further complicates the formulation. However, we utilize the formulation from [12], to refine the placement of the shared register based on related timing. The retiming variables are, as in the previous section, optimized to activate the most favorable STA constraints. This interplay between retiming, placement, and STA is shown in Figure 1.

We first describe an LP formulation for local register relocation based on a simplified form of the LP in [12]. We then incorporate it into our retiming formulation.

Suppose register l can be incrementally placed to improve slack while leaving all other gates fixed. We define a timing graph $G_l = (V_l, E_l)$ that consists of vertices and edges that are adjacent to l. V_l contains the driver u, and sinks v, of l. The edge set E_l contains the timing arcs that are adjacent to l. The LP formulation computes the variables β_x^l and β_y^l, the optimal x- and y-coordinates of l. The variables in this LP are as follows:

- α_x^v, α_y^v: fixed x- and y-coordinates of vertices $v \in V_l$.
- $U_x^e, U_y^e, L_x^e, L_y^e$: upper and lower bounds for the location of nets $e \in E_l$. These upper and lower bounds determine the HPWL of the particular net described by edge e as follows: $\mathrm{HPWL}(e) = (U_x^e - L_x^e + U_y^e - L_y^e)$. As the location of the register changes, these net boundaries also change, and, in turn, change the HPWL.
- R_u, A_u: the AAT and RATs of vertices in V_l.
- $\mathcal{L}$: the local worst-case slack (of the worst pin in V_l).

The MILP to determine optimal register placement is shown in Figure 9. This program sets the values of β_x^l and β_y^l such that $\mathcal{L}$ is maximized. Here, A_u of any vertex $u \in V_l$ that *drives* register l is fixed. Similarly R_v for any vertex v that is *driven by* l is also fixed. The only independent variables are

[2]The Logic-OR can be implemented using intermediary variables δ_A, δ_B and indicator variables I_A, I_B with the following constraints: $\delta_A = C - A,\ \ \delta_B = C - B,\ \ I_A \leq \delta_A,\ \ \mathrm{if}(I_A == 0)\ \delta_A = 0,\ I_b \leq \delta_B,\ 6)\ \mathrm{if}(I_B == 0)\ \delta_B = 0, I_A + I_B \leq 1.$

Objective

Minimize : $-\mathcal{M} + \sum_{(u,v)\in E}(K)\ \ w_r(u,v)$

subject to

$\forall(\mathrm{u},\mathrm{v}), \mathrm{r(u)} - \mathrm{r(v)} \le \mathrm{w(u,v)}$

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}) \in \mathrm{E_l},\ \ \mathcal{L} \le \mathrm{R_u} - \mathrm{A_u} - \mathrm{D_u}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}) \in \mathrm{E_l},\ \ \mathcal{L} \le \mathrm{R_v} - \mathrm{A_v} - \mathrm{D_v}$

if(!hasReg($\mathbf{u},\mathbf{v}$)) :

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{R_u} \le \mathrm{R_v} - \tau\mathrm{HPWL(u,v)} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),\ \ \mathrm{A_v} \ge \mathrm{A_u} + \tau\mathrm{HPWL(u,v)} + \mathrm{D_v}$

Let l be register on $(\mathbf{u},\mathbf{v})$

if(hasReg($\mathbf{u},\mathbf{v}$)):

$\mathrm{e} = (\mathrm{u},\mathrm{l}),\ \ \mathrm{U_x^e} \ge \alpha_\mathrm{x}^\mathrm{u},\ \ \mathrm{U_y^e} \ge \alpha_\mathrm{y}^\mathrm{u}$

$\mathrm{e} = (\mathrm{u},\mathrm{l}),\ \ \mathrm{L_x^e} \le \alpha_\mathrm{x}^\mathrm{u},\ \ \mathrm{L_y^e} \le \alpha_\mathrm{y}^\mathrm{u}$

$\mathrm{f} = (\mathrm{l},\mathrm{v}),\ \ \mathrm{U_x^f} \ge \alpha_\mathrm{x}^\mathrm{v},\ \ \mathrm{U_y^f} \ge \alpha_\mathrm{y}^\mathrm{v}$

$\mathrm{f} = (\mathrm{l},\mathrm{v}),\ \ \mathrm{L_x^f} \le \alpha_\mathrm{x}^\mathrm{v},\ \ \mathrm{L_y^f} \le \alpha_\mathrm{y}^\mathrm{v}$

$\forall \mathrm{e} = (\mathrm{u},\mathrm{l}), \mathrm{L_x^e} \le \beta_\mathrm{x}^\mathrm{l} \le \mathrm{U_x^e}$

$\forall \mathrm{f} = (\mathrm{l},\mathrm{v}), \mathrm{L_y^f} \le \beta_\mathrm{y}^\mathrm{l} \le \mathrm{U_y^f}$

$\mathrm{e} = (\mathrm{u},\mathrm{l}), \mathrm{L_x^e} \le \beta_\mathrm{x}^\mathrm{l} \le \mathrm{U_x^e}$

$\mathrm{e} = (\mathrm{u},\mathrm{l}), \mathrm{L_y^e} \le \beta_\mathrm{y}^\mathrm{l} \le \mathrm{U_y^e}$

$\mathrm{f} = (\mathrm{l},\mathrm{v}), \mathrm{L_x^f} \le \beta_\mathrm{x}^\mathrm{l} \le \mathrm{U_x^f}$

$\mathrm{f} = (\mathrm{l},\mathrm{v}), \mathrm{L_y^f} \le \beta_\mathrm{y}^\mathrm{l} \le \mathrm{U_y^f}$

$\mathrm{R_u} \le \mathrm{R_l} - \tau(\mathrm{U_x^f} - \mathrm{L_x^f} + \mathrm{U_y^f} - \mathrm{L_y^f}) - \mathrm{D_u}$

$\mathrm{A_v} \ge \mathrm{A_l} + \tau(\mathrm{U_x^e} - \mathrm{L_x^e} + \mathrm{U_y^e} - \mathrm{L_y^e}) + \mathrm{D_v}$

Fig. 10. Max-slack retiming with relocation of registers.

β_x^l and β_y^l which determine the U and L variables. These, in turn, determine A_v, R_u for all vertices.

The program in Figure 9 is modified in Figure 10 to simultaneously incorporate retiming and placement, and no longer fixes the neighboring RAT and AAT variables. In this figure, each edge (u, v) on which a register appears constrains the placement of the register in question. It is assumed that all edges starting at u, i.e., of the form (u, v), such that $\mathrm{hasReg}(u, v) = 1$ share the same registers. The register is placed in a location which minimizes the slack of neighboring gates. Since the slacks of neighboring gates in turn affect those of *their* neighboring gates, and so forth, a ripple effect ensues. Therefore, the register is actually placed in an optimal location with respect to the entire circuit. The key here is to enforce a small set of local constraints for each edge that interact to find global optima.

D. Cloning to Increase the Scope of Retiming

A key insight in our work is that *opportunities for backward register movement are often limited by fanout branches in combinational circuits.* As illustrated in Figure 2, retiming movement is blocked when fanouts of a gate do not share registers. We increase these opportunities by cloning fanout branches such that registers can move beyond the cloned gate. We achieve this by relaxing legality constraints to allow extra registers to move backwards. In addition, the fanouts of any cloned vertex are divided such that the STA on some of the edges is computed with respect to the cloned, rather than original vertex.

The legality constraints in retiming ensure that no edge has negative weight. With cloning, edges can indeed have negative weight due to registers being retimed backwards through a cloned gate. However, forward retiming of registers still follows traditional legality rules.

Suppose vertex u has fanouts $O = \{o_1, o_2, \ldots o_T\}$ and

Objective

Minimize : $-\mathcal{M} +$

$\sum_{(\mathrm{u,v})\in \mathrm{E}}(\mathrm{K})\mathrm{RegCt(u,v)} + \sum(\mathrm{u})\mathrm{IsCloned(u)}$

subject to

$\forall \mathrm{u}, \forall$ fanins i of u, $\mathrm{minPush(u)} \le \mathrm{w(i,u)} - \mathrm{r(i)}$

$\forall \mathrm{u}, \forall$ fanouts o of u, $\mathrm{maxPull(u)} \ge \mathrm{w(u,o)} + \mathrm{r(o)}$

$\forall \mathrm{u},$ if$(\mathrm{r(u)} > 0)\mathrm{maxPull(u)} \ge \mathrm{r(u)}$

$\forall \mathrm{u},$ if$(\mathrm{r(u)} < 0)\mathrm{minPush(u)} \le -\mathrm{r(u)}$

$\forall(\mathrm{u},\mathrm{v}),$ if$(\mathrm{w_r(u,v)} > 0)$

$\mathrm{RegCt(u,v)} = \mathrm{w_r(u,v)}, \mathrm{CloneCt} = 0$

$\forall(\mathrm{u},\mathrm{v}),$ if(!isClone(u) && !hasReg(u,v)) :

$\mathrm{R_u} \le \mathrm{R_v} - \tau\mathrm{HPWL(u,v)} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),$ if(!isClone(u) && hasReg(u,v)) :

$\mathrm{R_u} \le \mathrm{R_l} - \tau\mathrm{HPWL(u,COG(l))} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),$ if(isClone && hasClone(u,v)) :

$\mathrm{R_{clone(u)}} \le \mathrm{R_v} - \tau\mathrm{HPWL(COG(clone(u)),v)} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),$ if(isClone(u) && !hasClone(u,v)) :

$\mathrm{R_u} \le \mathrm{R_v} - \tau\mathrm{HPWL(u,v)} - \mathrm{D_u}$

$\forall(\mathrm{u},\mathrm{v}),$ if(!isClone(v) && !hasReg(u,v)) :

$\mathrm{A_v} \ge \mathrm{A_u} + \tau\mathrm{HPWL(u,v)} + \mathrm{D_v}$

$\forall(\mathrm{u},\mathrm{v}),$ if(!IsClone(v) && hasReg(u,v)) :

$\mathrm{A_v} \ge \mathrm{A_l} + \tau\mathrm{HPWL(COG(l),v)} + \mathrm{D_v}$

$\forall(\mathrm{u},\mathrm{v}),$ if(isClone(v)) :

$\mathrm{A_v} \ge \mathrm{A_u} + \tau\mathrm{HPWL(u,v)} + \mathrm{D_v}$

$\mathrm{A_{clone(v)}} \ge \mathrm{A_v} + \tau\mathrm{HPWL(u,v)} + \mathrm{D_v}$

$\forall \mathbf{u},$ **if**(isClone($\mathbf{u}$))

$\mathcal{M} \le R_{clone(u)} - A_{clone(u)} - D_{clone(u)}$

$\forall \mathrm{u}, \mathcal{M} \le \mathrm{R_u} - \mathrm{A_u} - \mathrm{D_u}$

Fig. 11. Gate cloning in max-slack retiming.

fanins $I = \{i_1, i_2, \ldots i_m\}$. We represent this situation by imposing two constraints on the retiming variable r(u) for a vertex u: one which is enforced when $r(u)$ is positive, and one which is enforced when $r(u)$ is negative. If $r(u)$ is positive (i.e., the retiming is backward), then the maximum number of registers that are allowed to pass backwards is the greatest number of registers that appear on any fanout branch of u. If $r(u)$ is positive, then the constraint is the same as before:

$$\begin{array}{l} \mathrm{maxPull}(u) = \max_{o\in O}(w(u,o) + r(o)) \\ \mathrm{minPush}(u) = \min_{i\in I}(w(i,u) - r(i)) \\ \mathbf{if}(r(u) > 0)\ \ r(u) < \mathrm{maxPull}(u) \\ \mathbf{if}(r(u) < 0)\ \ \mathrm{minPush}(u) \ge -r(u) \end{array} \tag{18}$$

Together, these two constraints can completely replace the general legality constraints. The presence of registers is indicated by a positive weight on an edge. Negative weights indicate that the driver of the edge was cloned. The original driver is connected to the retimed register on the (neighboring) edge(s) with non-negative weight, and the cloned driver drives the remaining sinks (as identified by edges with negative weight). We use the additional variable $\mathrm{hasClone}(u, v)$ which is set to 1 if and only if the register count on edge (u, v) is negative. These variables are set in a similar way as hasReg. Recall that constraints can be triggered by logical conditions through indicator variables or big-M formulations.

The MILP incorporating cloning is shown in Figure 11. For clarity, we illustrate cloning incorporated into the basic STA-based program with COG-based placements. In practice, we simultaneously place and clone registers and gates.

The slack is computed slightly differently in the pres-

ence of clones. New variables in Figure 11 include indicator variables $i\text{sCloned}(u), A_{clone(u)}, R_{clone(u)}$ for each vertex v. The variable $\text{isCloned}(u) = 1$ if $\text{hasClone}(u, v) = 1$ for one of the edges of the form (u, v). The computation of $A_{\text{clone}(u)}, R_{\text{clone}(u)}$ is performed as follows:

$\mathbf{if}(\mathbf{w_r}(\mathbf{i}, \mathbf{u}) - \mathbf{r}(\mathbf{i}) > \mathbf{0})$
$A_{\text{clone}(u)} \geq A_i + \tau\text{HPWL}(i, \text{COG}(l)) + D_u$
$\mathbf{if}(\mathbf{w_r}(\mathbf{u}, \mathbf{i}) - \mathbf{r}(\mathbf{i}) \leq \mathbf{0})$
$A_{\text{clone}(u)} \geq A_i + \tau\text{HPWL}(i, u) + D_u$
$\mathbf{if}(\mathbf{w_r}(\mathbf{u}, \mathbf{i}) - \mathbf{r}(\mathbf{i}) \leq \mathbf{0})$
$R_{\text{clone}(u)} \leq R_i - \tau\text{HPWL}(\text{COG}(clone(i)), i) - D_u$

For the new RAT variable, we assume that a vertex driven by a clone has no registers on the connecting edge. As illustrated in Figure 11, the main differences in slack computation include 1) the additional edge $(u, clone(v))$ for every edge (u, v) where v is cloned, 2) the use of the clone's AAT, $A_{clone(u)}$, when computing the AAT of vertices v where (u, v) has a clone. We minimize the number of registers and clones in the retimed circuit using two variables isCloned and RegCt, which is computed as follows:

$$\mathbf{if}(\mathbf{w_r}(\mathbf{u}, \mathbf{v}) > \mathbf{0}) \quad \text{RegCt}(u, v) = w_r(u, v) \tag{19}$$

IV. Empirical Validation

We integrate our optimizations into an industrial physical synthesis flow. Our benchmarks are the largest functional units of a 45nm high-performance microprocessor design. We operate on these benchmarks after logic synthesis, timing-driven synthesis, timing-driven placement, electrical correction, and critical path optimization (through buffering and gate sizing) are completed [3]. We use an industrial timing analysis tool to obtain initial conditions for AATs and RATs throughout the circuit [7]. Our experiments were conducted on an 8-core system with 2.8 GHz AMD Opteron 854 CPUs and 80 GB of memory. Our MILPs were solved with ILOG CPLEX 12.1 configured to use up to 8 cores in parallel.

Table I shows a 7.7% improvement (on average) in worst-case slack ($\mathcal{M}$) and a 69% improvement in *total negative slack* ($\mathcal{T}$) when retiming with simultaneous placement. The slack improvements are reported in terms of the clock period $\mathcal{P} = 174ps$. $\mathcal{T}$ is computed as shown in Equation 8 with threshold of $T = 0$. Percentage improvement in min-slack $\mathcal{M}$ is computed as follows:

$$\%\mathcal{M} = \frac{\mathcal{M}_{\text{new}} - \mathcal{M}_{\text{old}}}{\mathcal{P}} * 100\% \tag{20}$$

We note that the slack numbers are reported with respect to *buffered* wire delay. Past literature reports *unbuffered* wire delay, where dramatic improvement in slack may be misleading due to the need for subsequent buffering. In this experiment, the MILP for retiming with placement was given initial solution seeds from the max-slack MILP retiming shown in Figure 8. This helped CPLEX to calculate MILP solutions quickly. The entire optimization sequence took < 41s on each benchmark. Note that our joint optimization was performed *after* several iterations of placement, rebuffering, and gate sizing, yet significantly improved slack.

Table II evaluates the impact of cloning during retiming. In this experiment, we measure the *total thresholded slack* (Θ_T), as defined in Equation 9, with the threshold $T = 100ps$. The threshold value represents the desired amount of guard-banding (protection) against process variations and NBTI, which can degrade timing. Empirical results indicate that cloning can improve the Θ_T of the circuit by up to 57% over just retiming and placement. Thus, even when opportunities for cloning on the critical path are limited, the remainder of the circuit can be improved for increased resilience.

Unlike previous localized transformations, SPIRE scales to design partitions with over 1000 cells as shown in the #std cells column in Table I. SPIRE can process larger circuits by partitioning the design into windows of appropriate size. SPIRE can also be applied in overlapping windows.

V. Extensions

SPIRE's key advantage over existing physical synthesis transformations is the synergistic use of several types of optimizations. Our MILPs are more costly than existing transformations but also more powerful since they can be applied to larger windows than many of the localized transformations used in the industry today [11], [12]. This flexibility of SPIRE allows one to change the size and scope of optimization and offers rich trade-off opportunities. However, increasing optimization strength will likely change the trade-off between runtime and optimization-window size. Additional optimizations can be integrated into SPIRE.

- To relocate combinational gates, create variables for the x- and y- location for each gate and write the delay equations as in Section III-C in terms of those variables.
- To incorporate gate sizing in SPIRE, one must model nonlinear timing characteristics of individual gates or standard cells. This can be accomplished by precomputing the response to a set of discrete sizes (from the library) and selecting them using conditional constraints. If a particular gate size is selected, a corresponding gate delay will be used in the STA, as specified by a conditional constraint.
- Similarly, threshold voltage (V_{th}) assignment is modeled by selecting gate delays with Boolean variables. As lowering V_{th} improves speed at the cost of power, the number of low-V_{th} assignments must be upper-bounded.
- Common placement constraints including region constraints and obstacles can be represented in SPIRE. Region constraints are modeled with linear bounds on the x- and y-coordinates of each gate. To avoid obstacles, the placement region is divided into allowable regions that hug the obstacles. A disjunctive (OR-type) constraint is then added to require placement in one of the allowed regions. Routing congestion can also be represented as an obstacle using this mechanism to prevent any movable objects from being added in congested regions.

By integrating several optimizations and applying them to windows with thousands of objects, SPIRE offers a unique physical synthesis optimization that lies between local optimization of individual objects (which is typical of current tools) and global optimization of the entire design.

VI. Conclusions and Future Work

State-of-the-art physical synthesis methodologies tend to perform a series of local transformations to achieve a target clock period [3]. However, the difficulty of timing closure in high-performance designs calls for *netlist transformations*

Design	#std. cells	Initial $\mathcal{M}$, ps	Initial Regs	Initial $\mathcal{T}$, ps	Retiming+Placement $\mathcal{M}$, ps	Retiming+Placement Regs	Retiming+Placement $\mathcal{T}$, ps	Retiming+Placement Time, s	Overhead % cells	Improvements % $\mathcal{M}$	Improvements % $\mathcal{T}$
azure1	536	3.42	41	0.00	10.14	49	0.00	1.19	0.00	3.87	0.00
azure2	1097	-2.53	79	-15.17	**2.95**	155	**0.00**	4.46	**6.93**	3.15	**100.00**
azure3	1032	-16.22	97	-212.69	-6.49	108	-37.95	0.4	1.07	5.59	82.16
azure4	1125	-2.30	79	-2.30	**3.82**	96	**0.00**	7.66	**1.51**	3.52	**100.00**
azure5	1140	-13.18	89	-114.54	**9.39**	161	**0.00**	40.71	**6.32**	12.97	**100.00**
azure6	1156	-10.49	83	-91.39	**7.14**	149	**0.00**	10.80	**5.71**	10.13	**100.00**
azure7	1198	-29.84	80	-3399.92	-17.02	145	-259.67	20.73	5.43	7.37	92.36
azure8	2578	-38.47	209	-391.03	-28.64	287	-265.68	24.87	3.03	5.65	32.06
azure9	2911	2.56	290	0.00	23.31	318	0.00	7.12	0.96	11.92	0.00
average									3.66	7.73	68.87

TABLE I

Minimum slack ($\mathcal{M}$) AND *total negative slack* ($\mathcal{T}$) IMPROVEMENT DURING SIMULTANEOUS RETIMING+PLACEMENT ON MACROS OF A 45NM MICROPROCESSOR (SEE EQNS. 7-8). MAXIMAL $\mathcal{T}$ IMPROVEMENT (100%) IS REACHED WHEN DESIGN CLOSES ON TIMING. THESE CASES ARE INDICATED IN BOLD. %$\mathcal{M}$ IS COMPUTED AS DESCRIBED IN EQUATION 20 WITH $\mathcal{P} = 174ps$.

Design	#std. cells	Initial Regs	Initial Θ_T, ps	Retiming+Placement Regs	Retiming+Placement Θ_T, ps	Retiming+Cloning+Placement Regs	Retiming+Cloning+Placement Θ_T, ps	Retiming+Cloning+Placement Time, s	Overhead % cells	Improved % Θ_T
azure1	536	41	-4521.87	47	-2989.53	47	-2989.53	6.28	0.00	0.00
azure2	1097	79	-15597.31	153	-4537.57	153	-4537.57	7201.14	0.00	0.00
azure3	1032	97	-15515.34	105	-14333.89	110	-12739.10	2252.07	0.48	11.13
azure4	1125	79	-24206.70	81	-22226.57	83	-21762.75	3727.78	0.18	2.09
azure5	1140	89	-35296.55	148	-18881.61	537	-11333.49	7202.15	34.12	39.98
azure6	1156	83	-32183.65	148	-27566.43	588	-11956.50	237.10	**38.06**	**56.63**
azure7	1198	80	-46265.55	122	-33419.14	620	-17643.49	3741.82	41.57	47.21
azure8	2578	209	-39253.82	296	-26272.53	657	-15117.06	7201.70	14.00	42.46
azure9	2911	290	-13134.72	317	-9539.07	522	-4096.63	3905.28	**7.04**	**57.05**
average									15.05	28.51

TABLE II

Total thresholded slack (Θ_T) IMPROVEMENT THROUGH SIMULTANEOUS RETIMING, CLONING AND PLACEMENT (SEE EQN. 9). CLONING ALSO IMPROVED $\mathcal{M}$ ON AZURE6 BY 3.5%, WHILE ON THE REMAINING TESTCASES THE MOST-CRITICAL PATHS WERE NOT AFFECTED.

that can effect more powerful changes in the circuit. To address these issues, we presented SPIRE, an MILP-based physical synthesis optimization in which dynamic netlist transformations including retiming, cloning, and placement, can be performed simultaneously with respect to an embedded static timing analysis program. We demonstrated that isolated transformations, such as retiming, often run into obstacles that can only be resolved by other transformations, such as gate cloning. Empirical results show that SPIRE is able to significantly improve the worst-case and total slack in functional units of a 45nm high-performance microprocessor after an industrial physical synthesis flow that consists of several individual optimizations is performed.

SPIRE has been developed as an optimization for microprocessor methodologies. Our future work involves extending SPIRE to ASIC and SoC designs which are significantly larger than microprocessor design partitions. For this purpose, we propose automated partitioning to divide the design into windows of manageable size. Figure 12 shows a preliminary experiment incorporating the hMETIS partitioning software into SPIRE [4]. From the plot, we observe that using smaller windows sacrifices some solution quality, but additional partitioning produces smaller instances that can be solved faster.

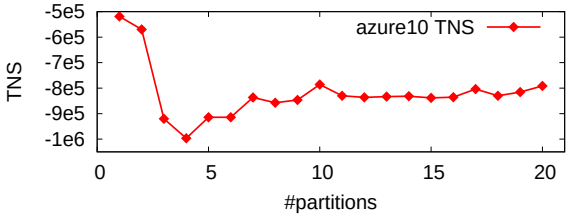


Fig. 12. An illustration of SPIREs effect on $\mathcal{T}$ (TNS) versus the number of approximately equal-size partitions of azure10 generated by the hMETIS partitioner [4].

REFERENCES

[1] C. J. Alpert, C. Chu, and P. G. Villarrubia, "The Coming of Age of Physical Synthesis," *ICCAD* 07, pp. 246-249.

[2] C. J. Alpert et al., "Accurate Estimation of Global Buffer Delay Within a Floorplan," *TCAD* 25(6), 2006, pp. 1140-1146.

[3] C. J. Alpert et al., "Techniques for Fast Physical Synthesis," *Proc. IEEE* 95(3), 2007, pp. 573-599.

[4] hMETIS: http://www-users.cs.umn.edu/~karypis/metis/hmetis/

[5] Y. Hu et al., "Simultaneous Time Slack Budgeting and Retiming for Dual-Vdd FPGA Power Reduction,"*DAC* 06, pp. 478-483.

[6] A. Hurst, P. Chong, A. Kuehlmann, "Physical placement driven by sequential timing analysis," *ICCAD* 04, p.379-386.

[7] J. A. G. Jess et al.,"Statistical Timing for Parametric Yield Prediction of Digital Integrated Circuits," *TCAD* 25(11), 2006, pp. 2376-2392.

[8] K. N. Lalgudi, M. Papaefthymou "Retiming Edge-triggered Circuits under General Delay Models," *ICCAD* 97, p.1393-1408.

[9] C. E. Leiserson, J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, 6, 1991, pp. 5-35.

[10] R. Nair, C. Berman, P. Hauge, E. Yoffa, "Generation of Performance Constraints for Layout," *TCAD* 8(8), 1989, pp. 860-874.

[11] M. Moffit, D. A. Papa, Z. Li, C. J. Alpert, "Path Smoothing via Discrete Optimization," *DAC* 08, pp. 724-729.

[12] D. A. Papa et al., "RUMBLE: An Incremental, Timing-driven, Physical-synthesis Optimization Algorithm," *TCAD* 27(12), 2008, pp. 2156-2168.

[13] S. S. Sapatnekar, *Timing*, Springer-Verlag, New York, 2004.

[14] S. S. Sapatnekar, R. B. Deokar, "Utilizing the Retiming Skew Equivalence in a Practical Algorithm for Retiming Large Circuits," *TCAD* 15(10), 1996, pp. 1237-1248.

[15] P. Saxena, B. Halpin, "Modeling Repeaters Explicitly Within Analytical Placement," *DAC* 04, pp. 699-704.

[16] L. Trevillyan et al.,"An Integrated Environment for Technology Closure of Deep-submicron IC Designs," *IEEE Design & Test* 21(1), 2004, pp.14-22.

[17] H. Zhou, "Deriving a New Efficient Algorithm for Min-period Retiming," *ASP-DAC* 09, pp. 990-993.