Minimizing Implementation Costs with End-to-End Retiming

Aaron P. Hurst Alan Mishchenko Robert Brayton
University of California, Berkeley
Berkeley, CA
{ahurst, alanmi, brayton}@eecs.berkeley.edu

***Abstract* - We introduce end-to-end retiming, a paradigm for efficiently evaluating a varied set of global retiming possibilities. The result is a continuous curve of retiming solutions stretching between the extremes of minimum delay and minimum register count. Any single retiming solution may be infeasible or unacceptably difficult to implement, but with an array of choices, a solution with the minimum implementation cost can be selected. Because the contour of the tradeoff between performance and cost is highly dependent upon circuit topology, it can not be easily predicted beforehand. We describe two applications for which end-to-end retiming is well suited: combined retiming/sizing and simultaneous retiming/clock skew scheduling. The latter combination is particularly interesting, as the costs of pursuing retiming and skewing depend on the topology in often complementary ways; the combination of the two is shown to be much more efficient than either in isolation. We examine the area and dynamic clock power consumption of a set of industry-supplied benchmarks and optimize the circuits to meet an aggressive timing target. End-to-end retiming meets the target with an average reduction of over 131% in the additional area required and 79% in the additional dynamic clock power consumed over the best of either retiming or skewing alone. End-to-end retiming provides a solution approach that is scalable, applicable to different cost objectives, and not limited in its timing model.**

I. INTRODUCTION

If the sequential elements in a design can be repositioned in either structure or time, the timing is no longer limited by the single longest combinational path, but by the maximum average delay around a loop. There is frequently a significant difference between the two; in a current set of industrial and academic benchmarks [1], the difference is up to 35%, even after synthesis and mapping with an industrial tool.

We concentrate on two techniques for sequential optimization: retiming [13] and skew scheduling [8]. Retiming relocates the structural position of the registers in a design, and skew scheduling inserts intentional delays into the clock distribution network to move the temporal position of the registers. Despite differing means of implementation, it has been recognized that these are similar means to the same end.

However, the potential of global sequential optimization remains largely unrealized. Both clock skewing and retiming are present in several commercial design tools, but their role is typically confined to small incremental resynthesis late in the flow, often as an aid in physical design closure. The scope of the allowed change is local and limited. Global optimization is specifically avoided because of its unpredictability in difficulty of implementation. Large improvements in performance require equally large changes in structure or timing of the design. These can be problematic.

Also, additional complexity is introduced into those parts of the design process that are already challenging. Retiming complicates testing and verification by altering the number of registers and their state encoding. In most cases, the automatic verification of a design that has been significantly transformed by retiming is currently not available. Clock skewing, with its specific and individual requirements on the arrival times of the clock signals at each register, imposes difficult challenges for clock network design. While small delay adjustments are possible by inserting buffers on the clock inputs, large skews require impractically large amounts of hardware and/or redesigning the upper levels of the network. The further adoption of sequential optimization is contingent on the minimization of the implementation cost.

While the algorithms for pursuing optimal solutions are well understood, strategies for backing off from extreme solutions to feasible intermediates are less developed. An example is delay-constrained minimum-area retiming. Even if this technique were computationally tractable for large designs, it gives no information about the shape of the cost curve or the quality of nearby alternatives. The designer is left with little to no information about how to balance the extent of retiming with other means of meeting the design specifications.

To this end, we introduce *end-to-end retiming*. At one extreme lies minimum-register retiming and at the other minimum-delay retiming, and a continuous curve of retiming solutions is generated between these two. The entire approximate performance/cost curve of retiming is efficiently enumerated. Also, the algorithm is scalable to large designs.

This approach can be used to not only explore the performance/cost curve of retiming, but also the joint cost

of similar and substitutable synthesis techniques under a constraint. One particularly interesting pairing is simultaneous retiming and skew scheduling. This application is motivated by the observation that the cost of both sequential optimization techniques have a strong dependence on the circuit topology—but in different and often complementary ways. Within a single design, there are critical elements where performance can be improved more efficiently through retiming and others where skewing is more suitable. When the two are used in combination, a performance improvement can be obtained with less implementation cost than either in isolation, avoiding extreme solutions of either and addressing the aforementioned problems. To demonstrate this, we examine the area and dynamic power consumption of the sequential elements in a design and show significant reductions in the requirements for either.

The contributions of this work can be divided into two components: 1) end-to-end retiming, a general paradigm for efficiently exploring and minimizing the isolated or joint implementation cost curve of retiming and 2) the application of simultaneous skewing/retiming to minimize the cost of implementation.

II. BACKGROUND

This paper focuses on methods for restructuring sequential circuits at the netlist level, and in particular, retiming and skew scheduling. Retiming is described in detail in the following subsections, and clock skew scheduling is elaborated in Section 5.2. Both transformations leave the combinational logic untouched, preserve output behavior, and do not unduly complicate timing analysis. Other techniques for balancing computation across sequential boundaries include: architectural changes to state encoding, scheduling, or output latency, as well as lower-level timing tricks that exploit wave pipelining [11] and/or latch transparency [19]. However, these methods do not share the aforementioned characteristics, and while applicable to some designs, are often unsuitable for general logic and physical synthesis flows.

A. Retiming

Let a retiming graph $G = \langle V, E \rangle$ consist of a set of vertices V corresponding to the delay elements in a circuit, each with delay $d(v)$, and a set of edges $E \subseteq V \times V$. Hereafter, an individual edge e is interchangeably referred to by either name or by its endpoints (e.g. $e = u \rightarrow v$). Each edge may contain zero or more initial registers $w_i(e)$, and let $w_r(u \rightarrow v) = w_i(u \rightarrow v) - r(u) + r(v)$ be the final number of registers on each edge after retiming. Let T be the target clock period.

The goal of retiming is to a find a lag function $r(v) \in \mathbb{Z}$ and a set of valid sequential arrival times $R(v) \in \Re \geq d(v)$ for the retimed circuit that satisfies the following constraints: timing propagation (Equation 1), non-negative register count (Equation 2), and correct setup timing (Equation 3). Note that our definition of $R(v)$ is based on the post-retiming circuit and varies from that of [13] but is obtainable through simple substitution. The contents of Section 5.4 motivate this change.

$$R(v) - R(u) \geq d(e) - w_r(e)T \qquad \forall e=(u \rightarrow v) \qquad (1)$$

$$r(u) - r(v) \leq w_i(u \rightarrow v) \qquad \forall (u \rightarrow v) \qquad (2)$$

$$R(v) \leq T \qquad \forall v \qquad (3)$$

If the goal is *minimum-delay* retiming, the minimum feasible period T can be discovered through binary search. In addition to finding any feasible solution, an objective function may be introduced. For example, in *delay-constrained minimum-area* retiming, the objective is to minimize the total number of registers while meeting some target period T_{targ}. This results in what is often a significantly more difficult problem.

It should be noted that this basic formulation typically requires several extensions to be meaningful in a realistic circuit: fan-out sharing and a model of the dependence of delay on changes in load (described in [13][15]). These extensions add further complexity to the problem and may decrease the computational tractability to an impractically small circuit size.

B. Incremental Delay Retiming

The limited scalability and incomplete model of the analytic formulation of retiming has motivated the development of alternatives. Recent research has focused on incremental retiming, a means of improving the worst-case delay of the circuit solution by iteratively relocating registers through a series of incremental moves.

If the optimal minimum-delay retiming is required, Zhou [20] proposes an elegant incremental algorithm for finding the exact minimum-delay solution, even with non-uniform gate delays. The drawback of this approach is its simplistic timing model.

Alternatively, if exploring the exact minimum-delay retiming is not critical, Singh et. al [18] describe a heuristic that produces near-optimal results but allows a wide variety of design constraints to be included in the problem. Because the decision-making is not premised on a simplified timing model, the timing information can be as accurate as needed, even including wire delays and other physical information. The flexibility in including constraints is particularly powerful. The authors concentrate on excluding solutions that violate physical constraints, but moves that lead to the blow-up of any implementation cost can be similarly blocked or delayed.

III. END-TO-END RETIMING

We describe the new technique of end-to-end retiming. It warrants such a label because it visits:

1. *both the min-delay and min-register retiming, and*
2. *a continuous set of solutions between them*

In contrast to retiming to a single solution with a specific goal (e.g. minimum-delay, delay-constrained minimum-area), end-to-end retiming explores an entire spectrum of performance possibilities. Other than the endpoints, no guarantee is made that any single point is exactly optimal in either register count or delay; in general, the path that is generated will be suboptimal to varying degrees.

The motivation behind this approach is the value in having a complete retiming performance/cost curve available. This presents the designer with the information he needs to make an informed choice about how to appropriately back off from extreme solutions and choose a suitable degree of global retiming. A heuristic solution chosen with knowledge of its alternatives is often more valuable than an optimal solution generated by blindly choosing a single design point.

The overall algorithm is described in Algorithm 1 and illustrated graphically in Figure 1. Starting from the initial design with period T_{init}, incremental delay retiming is applied until the minimum-delay solution is reached. After each incremental retiming that improves the delay, a new point along the curve is generated. The cost of this solution is evaluated and the best solution(s) retained for later use. The cost can be flexible, e.g. the total cost of implementing clock skewing to achieve an overall performance goal, T_{target}, plus the cost of generating the given retiming.

The process of incremental retiming for delay is the primary engine for transforming the solution in end-to-end retiming. The incremental movement allows not only a continuous set of snapshots, but also the ability to customize how much computational effort to spend, how optimal the solution must be, and what other constraints can be taken into account in allowing an individual move. We intentionally do not specify an exact recipe for choosing incremental moves and instead refer the reader to the general solutions of [18] and [20] and their own specific needs; the only requirement is that each register moves by no more than one gate delay per iteration.

Algorithm 1: End-to-End Retiming

```
 1 let T_init = period()
 2 let direction = forward
 3 while(improvement) {
 4   if (direction == forward)
 5     incremental_forward();
 6     if (no improvement) direction = backward;
 7   if (direction == backward)
 8     incremental_backward();
 9   evaluate_solution();
10 }
11 retime_minregister();
12 direction = forward;
13 while (period() < T_init) {
14   if (direction == forward)
15     incremental_forward();
16     if (no improvement) direction = backward;
17   else
18     incremental_backward();
19   evaluate_solution();
20 }
```

Once the first application of incremental retiming for delay has reached its limit at the minimum-delay solution, the set of points with better performance than the initial design has been fully explored; however, this is only half of the space. Next, minimum-register retiming [10] is applied to generate a solution that has the exact minimum number of registers of any retiming. The minimum-register retiming algorithm is based on iterative binary maximum-flow computations and is extremely fast, even for the largest circuits. Because it is also canonical in the number of registers, the result is not dependent upon the particular retiming supplied as an input.

A second phase of incremental retiming is then applied until a period at least as small as that of the original circuit has been recovered, at which point the algorithm terminates. The netlist at exit does not generally correspond to the original, and no guarantee can be made about the relative numbers of registers or the total cost. Empirically, after retiming across the entire performance axis, a heuristic incremental method is typically not able to reproduce the quality of the original and exits with a slight increase in register count. This motivates the exploration in two segments, starting first from the initial netlist, as shown in Figure 1.

The smoothness of the resulting curve follows from the restriction on the incremental moves to be one gate at a time. The distance between two adjacent points on the retiming curve can thus be guaranteed to be with a certain delay granularity *g*. If the delay model is load-independent,

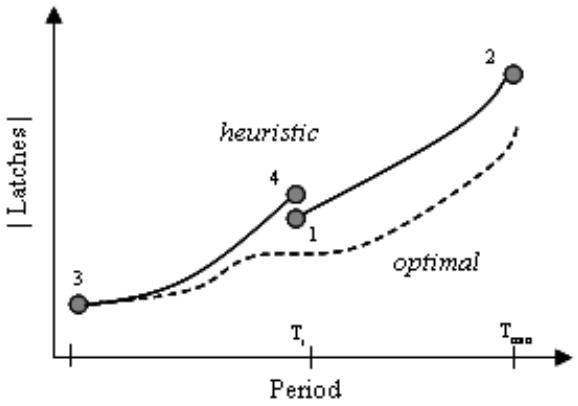

Figure 1: The exploration of the retiming space proceeds from (1) the initial design with period T_{init} to (2) the heuristic min-delay retiming to (3) the exact min-register retiming to (4) the retiming period is > T_{init}. The dashed line suggests the curve of the optimal min-area solutions.

$g = \max_{\forall v} d(v)$, the largest gate delay. If the delay model is load dependent, that is, $d(v) = d_{\text{intrinsic}}(v) + C_{load}\, d_{load}(v)$, the bound becomes

$$g = \max_{\forall v} d_{\text{intrinisic}}(v) + \max_{\forall p=(u\to w)} \sum_{v\in p} d_{load}(v)\Delta C_{load} \ .$$

In almost all cases, the total change in load capacitance along any path is very small; however, if necessary, the maximum change in capacitive load can be fixed by limiting the number of registers than can be retimed in any iteration.

IV. Combined Retiming / Sizing

End-to-end retiming can also be used to minimize the joint cost of two substitutable synthesis techniques under a delay constraint. A preliminary example is the combined application of retiming and gate sizing, for which few good solutions currently exist. When registers are retimed to different locations, the capacitive loadings on the nets from which a register was removed and added are altered. There exist extensions to optimal retiming to account for these load changes and their effect on load-dependent delays, but this assumes that the driving strength of all gates is will remain fixed. If gate sizing is applied after retiming, the changes in load may lead to a different selection of drive strengths and violate this assumption. This is not predictable from within the retiming algorithm, and its solution may be unrealistic. As a result, current commercial tools typically only allow a conservative set of local retiming moves and expend significant computation to evaluate the consequences on gate sizing.

The end-to-end retiming paradigm presents a means of quickly exploring a complete global set of retiming/sizing solutions. Given a performance target, T_{target}, the gate sizes in a circuit can be increased to meet this requirement at the cost of c_{size} additional power or area. As end-to-end retiming progresses and each individual retiming point is generated with period T and area or power cost c_{ret}, the performance shortfall of $T_{target} - T$ is met with sizing (if possible) and the total cost $c_{tot} = c_{ret} + c_{size}$ calculated. The netlist(s) with the minimum total cost are retained.

The gate resizing needs to be performed after each retiming move, but because of the relatively local propagation of gate sizing changes [6] and the single-gate register movements, the operation can be incremental and restricted to the local vicinity of the register moves. The entire curve can be quickly traversed.

Experimental results are presented in Section 6.

V. Combined Retiming / Skewing

The simultaneous application of retiming and skewing presents a particularly interesting application. While both retiming and skewing are a means to the same end—relocating the sequential elements to balance uneven combinational delays—they offer very different modes of implementation. The two have been studied together with the objective of improving performance by relaxing hold constraints [14]. Instead, we examine how they can be used to minimize implementation cost.

A. Motivation: Circuit Topology

Our motivation for simultaneously skewing and retiming the registers in a design is based on the observation that *the costs of retiming and skewing depend on the structure of the circuit in different ways.* For some registers, it is cheaper to meet a performance constraint by moving the register structurally (with retiming), and for others, it is cheaper to move it temporally (with skewing). Figure 2 illustrates an example. The cost of moving a cut of registers with skewing is proportional to the structural width of the initial cut and the temporal distance moved, while the cost of retiming is related to the difference in the structural width of the final and initial cuts.

Because the cost of a retiming movement can be negative when registers are shared, it is possible to reduce the cost of a set of registers with retiming below their cost in the original design. This allows relaxing the performance of the original design with a $T_{target} > T_{init}$ to recover registers. Even with an aggressive performance constraint, it may still be beneficial to introduce timing violations with retiming and then correct for them with intentional skew—if the cost of the additional skew buffers is outweighed by the reduction in registers.

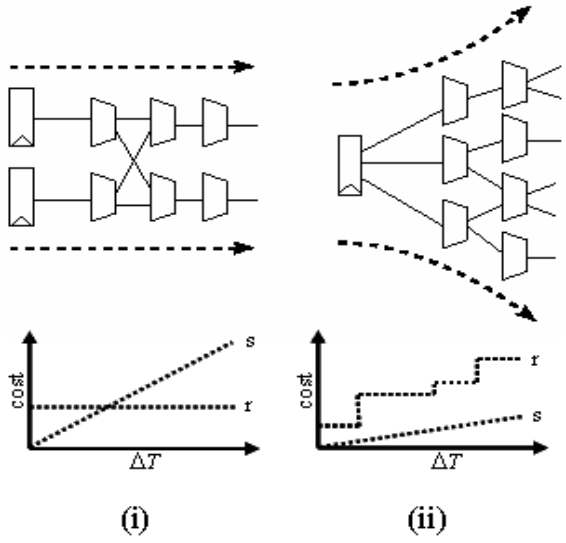

Figure 2: The different cost relationships between circuit structure and moving the latch boundary forward are illustrated for retiming (labeled "r") and skewing (labeled "s"). In Figure (i), a circuit with a pipeline-like structure and confined fan-out can be retimed forward for free. In Figure (ii), the transitive fan-out width grows outward from the latch boundary; in this case, retiming is expensive but the smaller original width of the cut makes skewing cheap.

B. Skew Scheduling

Clock skew scheduling is typically formulated in terms of the inverted retiming graph, where the set of vertices V corresponds to the registers in the design, and the set of edges to the delay elements [8]. Enumerating the registers is convenient, because one independent variable (i.e. its skew) can be created for each.

We depart from this traditional formulation and instead introduce one that is compatible with the retiming. Given the formulation in Section 2.1, the problem can be modified to the skew scheduling problem by 1) assuming all retiming lags $r(v) = 0$ and 2) removing the constraint of Equation 3.

The resulting solution will comprise a set of feasible sequential arrival times $R(v)$, and while there is not a one-to-one correspondence between these variables and the registers in the design, the mapping is trivial. Given the sequential arrival time $R(u)$ at the output of node u, the necessary skew S on each of the $k = 1 \ldots w_r(u)$ registers ordered topologically on edge $(u \rightarrow v)$ is:

$$S(u \rightarrow v, k) = R(u) - kT \tag{4}$$

Because all of the integer variables are fixed to zero, the resulting problem is a linear program. Furthermore, determining the minimum feasible T is an instance of the maximum mean cycle problem, for which several efficient algorithms exist [7]. Experience indicates that Howard's algorithm [4] is both fast and scalable.

In general, there are many possible skew schedules to meet a target period. Analogously to delay-constrained minimum-area retiming, a schedule can be chosen that minimizes the amount of total skew by adding an appropriate optimization objective. However, compared to retiming, this is much less difficult; the minimum-cost schedule can be generated on the graph using the Bellman-Ford algorithm [5].

We assume that skews are implemented at each register's clock input with a string of delay elements that produce the required relative skew. The periodicity of the clock can be used to reduce the required delay to the fractional component of a clock cycle, but large fractional delay values still produce problematic consequences for congestion, power, or timing variation. Alternative strategies for implementing skews are proposed in [9][12], but these are not as widespread.

C. Cost Functions

When retiming is considered in isolation, the essence of its implementation cost can be captured by the change in the number of registers. Likewise, in the skew scheduling, the total amount of skew required provides a rough estimate of the quality of a schedule. For cross and combined comparisons, consider a linear relationship of the form of Equation 5. The constants a_v^{retime} and a_e^{skew} weight the individual costs of the number of registers w_r and the total amount of skew c_s required on each edge e.

$$\min \text{ cost} = \sum_{\forall e} a_v^{retime} w_r(e) + \sum_{\forall e} a_e^{skew} c_s(e) \tag{5}$$

This formulation lends itself to two design metrics of particular interest: area and the dynamic clock power consumption, described by Equations 6 and 7, respectively. Both cost functions are dependent upon the relative areas or input capacitances of the two elements. The quantity d_{buf} describes the delay that can be realized with one skew buffer.

$$A = A_{latch} \sum_{\forall e} w_r(e) + \frac{A_{buf}}{d_{buf}} \sum_{\forall e} c_s(e) \tag{6}$$

$$P = V^2 C_{latch}^{in} \sum_{\forall e} w_r(e) + \frac{V^2 C_{buf}^{in}}{d_{buf}} \sum_{\forall e} c_s(e) \tag{7}$$

It is certainly possible to evaluate these quantities more precisely (e.g. to discretize the number of skew buffers), add other design variables of interest, penalize large changes, or to create hybrid metrics. There is no restriction in the general end-to-end retiming flow that the cost function be linear or even have a closed analytic form; design choices can be evaluated in any manner. These two linear examples are valuable only because they can be solved optimally and compared against the quality of the result from heuristic end-to-end retiming.

D. Exact Formulation

The problems of retiming and skewing can be combined into a single mixed-integer linear program to minimize the cost when simultaneously employing both. In this combined problem, because the number and location of registers will vary with the retiming, the skew component of the total cost is not a straightforward quantity. Instead, we describe a set of variables to capture the total sum of skews as defined in Equation 4 along any given edge, $c_s(e) = \sum_{k=1}^{w_r(e)} S(e,k)$. This is a piecewise linear function of $R(u)$ and $w_r(u \rightarrow v)$. First, consider the case where there is either zero or one register present on the edge. If zero, the skew cost should also be zero, and if one, the cost should be $S(e,1)$. This is realized in the linear constraint of Equation 8, where β is arbitrary and larger than any $R(v)$.

$$c_s(u \rightarrow v) \geq \beta \left[w_r(u \rightarrow v) - 1 \right] + R(u) - T \tag{8}$$

In our experience, there seems to be little to no loss in optimality by leaving the restriction that w_r is at most one, but for completeness, it is possible to relax this constraint by introducing a set of M ordered binary indicator variables $w^j(e)$ to represent $w_r(e)$ via the constraints of

Equation 9 and 10. The general expression of c_s that allows up to an arbitrary maximum M registers to be retimed along each edge is Equation 11.

$$\sum_{j=1}^{M} w^j(e) = w_r(e) \tag{9}$$

$$0 \le w^M(e) \le \ldots \le w^2(e) \le w^1(e) \le 1 \tag{10}$$

$$c_s(u \to v) \ge \sum_{j=1}^{M} \beta\left[w^j(u \to v) - 1\right] + R(u) - jT \tag{11}$$

The result of this linear program is the optimal combination of retiming and skewing to minimize either area or dynamic power. While complete, this formulation for minimum cost is of little practical use: it is computationally intractable for all but the smallest circuits. A better approach is needed.

E. Using End-to-End Retiming

We utilize end-to-end retiming to minimize the cost of using retiming and skewing simultaneously to meet a target performance T_{target}. The version of the algorithm for minimizing the joint solution under a delay constraint requires only the added step of computing the marginal skew required to meet (if possible) the constraint for each netlist. At each point along the retiming curve, there exists a valid retiming with a known period $T_{current}$ and some associated cost. If $T_{current} < T_{target}$, the circuit has already met the target timing constraint; otherwise, additional skewing is required. The skew component of the cost can be very quickly computed by efficiently finding a minimum-cost skew schedule with Howard's algorithm, as described in Section 4.2. The netlist(s) with the best total cost are saved.

VI. EXPERIMENTAL RESULTS

End-to-end retiming is applied to a set of industry-supplied and academic designs. All benchmarks were first pre-optimized using the ABC logic synthesis package [3]. The timing data was extracted using a full table-based slew and load-aware timing analysis, and this model was used in an incremental min-delay retiming algorithm similar to [18]. The maximum mean cycle times were measured and used as the target performances. The following experiments were conducted on a set of 64-bit 2.33GHz Pentium Xeon computers.

A. Combined Retiming and Sizing

As described in Section 4, the area of simultaneously skewing and retiming a set of designs was minimized using end-to-end retiming. Sizing was performed greedily, and it was assumed that each gate was available in 4 discrete drive strengths. In the interest of space, a condensed version of the results is presented in Table 1 (additional information about these designs can be found in Table 3.) In the experiment, the total cost of meeting the target delay is the sum of 1) the increase in the of the combinational logic due to sizing 2) the area of the registers, with or without retiming. Two alternatives are compared. The column "Sizing" describes the area cost of meeting the delay constraint with *only* gate sizing. The column "End-to-End" describes the cost after applying end-to-end retiming to choose a good combined solution. End-to-end timing results in a net 4.9% decrease in this area cost for these benchmarks.

Table 1: Combined Retiming / Sizing

		Area (Unit Squares)		
Name	**| Cells |**	Sizing	End-to-End	% Improv.
nut_000	1852	1.44E+04	1.35E+04	6.3%
nut_003	2358	1.07E+04	9.53E+03	10.9%
oc_ata_vhd_3	4977	2.12E+04	2.04E+04	3.8%
oc_hdlc	3017	1.52E+04	1.40E+04	7.9%
oc_minirisc	2540	1.03E+04	9.56E+03	7.2%
oc_mips	17490	4.78E+04	4.57E+04	4.4%
oc_oc8051	12239	2.70E+04	2.66E+04	1.5%
oc_pci	13873	4.85E+04	4.70E+04	3.1%
oc_vga_lcd	12199	3.96E+04	3.85E+04	2.8%

B. Combined Retiming and Skewing

First, the combined retiming/skewing obtained from end-to-end retiming is compared against the optimum solutions obtained from the exact formulation to illustrate the tradeoff between optimality and runtime. Because of the limitations of the exact method, only the smallest of the ISCAS benchmarks, solvable as a MILP within an hour of runtime, were used. The results of four of the largest designs are presented in Table 2. In half of these cases, the optimal minimum-cost solution was found with heuristic end-to-end retiming, though the average runtime was over two orders of magnitude times faster. Our runtime was dominated by static timing analysis.

Next, end-to-end retiming was applied to a set of larger benchmarks [2]. Our technique was used to minimize both the dynamic power consumption of the clock endpoints (Equation 6) and the total area of the required registers and buffers (Equation 7). The power, area, and skew buffer delay values were taken from the GSC 0.13µm standard cell library provided with [1].

Table 2: Exact v. Heuristic Min-Cost for Retiming/Skewing

	Exact Min-Cost		End-End Min-Cost	
Name	Area	Runtime, s	Area	Runtime, s
s349	2.80 e3	49.0	3.00e3	0.07
s526n	3.70 e3	34.0	4.14e3	0.03
s1196	2.92 e3	1.5	2.92e3	0.04
s1423	1.25 e4	3.8	1.25e4	0.14

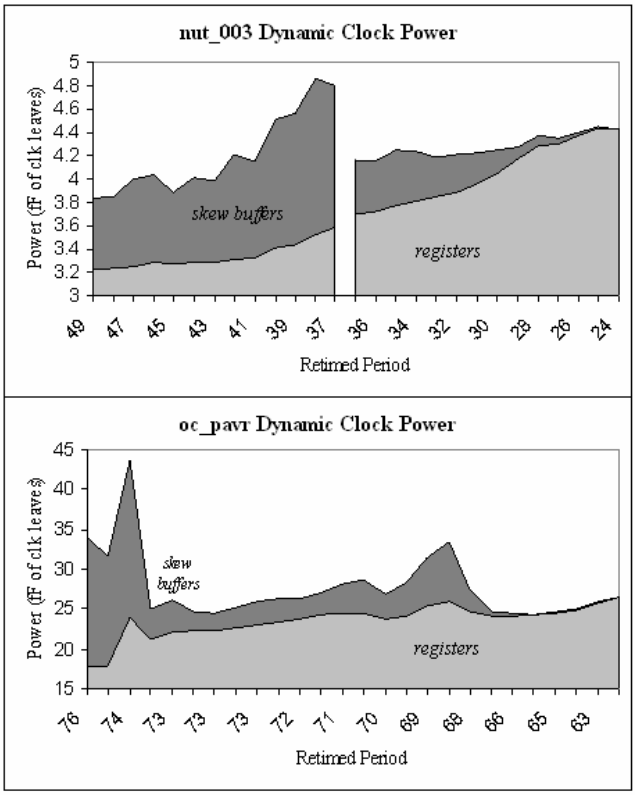

Figure 3. At each retimed period point we show in gray, the dynamic power consumed by the clock inputs for the registers required for retiming to that point. The additional dynamic power required to meet a performance target (in both cases, the minimum clock period) by clock skewing is added on top of this (shown in black). The performance retiming points are generated by end-to-end retiming for the designs *nut_003* and *oc_pavr*. In the upper graph, the break corresponds to the initial retiming solution: the right half is generated by incremental retiming from the original design while the left half is obtained from incrementally retiming from the min-register solution. The lower graph shows a special case where the initial solution (period 76) is already register minimal but to skew to the optimum performance (period 61) is costly. The minimum solution appears to be to retime to period 73 and then to use skewing to obtain period 61.

The full set of results is presented in Table 3. The column labeled T_{targ}/T_{init} indicates the aggressiveness of the target period as a fraction of the original. The sequential elements in the circuit were optimized to meet this period using three different methods: only skewing, only retiming, and a combined application of retiming and skewing computed with our algorithm. In the few cases where retiming alone was not able to meet the target (due to the discrete delays of the gates), the difference was corrected with a small amount of skew and included in that cost. Early mode timing was not an issue in these designs.

On average, meeting the performance target with combined retiming/skewing required over 131% less additional area and 79% less additional dynamic power than the best of either technique in isolation. In many cases (highlighted in bold in Table 3), the faster period was met using less area and/or power than the slower original design (resulting in the >100% reduction in average additional area). Even if the total of all skew buffers and registers (which are necessary for functionality) is considered, the reduction in area is 7.8% and dynamic clock power consumption by 10.7% over either technique in isolation.

Figure 3 illustrates the cost curve generated by end-to-end retiming in more detail for two benchmarks. The two regions represent the relative contributions of skew buffers and registers to the total dynamic power consumed on the leaves of the clock tree. The circuit *nut_003* is a case where the minimum-cost solution lies in the portion of the retiming curve revealed by minimum-register retiming. The aggressive target timing (T_{targ}=24.0) can be met with the minimum dynamic power consumption by first retiming to a slower period (T=49.0) and then recovering the performance with clock skew. The increase in additional skew buffers is outweighed by the decrease in the number of registers.

The balance between retiming and skewing in the minimum cost solution was highly design dependent. In some of the benchmarks, the result consisted of either maximally retiming or maximally skewing (as is the case in *oc_pavr* and *nut_003*, respectively, in Figure 3). However, in general, the minimum-cost solution utilized instances of both.

VII. Conclusions

The technique efficiently explores a smooth set of retiming solutions between the minimum-register and minimum-delay solutions. Because the cost of retiming is unpredictable *a priori*, end-to-end retiming allows an informed decision about the best performance/cost solution. An example application is to minimize the joint cost of retiming and skewing to meet a performance target. By using this technique, the total sequential area can be reduced by 7.8% over the best solution of either in isolation; the total dynamic power consumption of the clock tree endpoints can be reduced by 10.7%.

VIII. References

[1] C. Albrecht, IWLS 2005 Benchmarks, http://www.iwls.org/iwls2005/benchmarks.html.

[2] Altera Corp., Quartus II University Interface Program, www.altera.com/education/univ/research/unv-quip.html

Table 3: Area and Power Results of Combined Optimization with End-to-End Retiming

			Area (unit squares)					Power (clock pF)					Runtime
Name	**|Cells|**	**T_{targ}/T**	Orig	Sk-Only	Ret-Only	End-End	%Imprv	Orig	Sk-Only	Ret-Only	End-End	%Imprv	seconds
mux32_16bit	2438	0.37	8.64e+4	8.88e+4	1.14e+5	8.68e+4	2.3%	7.7	8.3	10.5	8.3	0.0%	2.8
mux64_16bit	4876	0.36	1.70e+5	1.73e+5	2.58e+5	**1.64e+5**	5.2%	15.1	15.9	23.2	15.2	4.4%	9.3
mux8_128bit	5012	0.38	1.87e+5	2.02e+5	2.72e+5	**1.68e+5**	16.8%	16.6	20.1	24.6	**15.2**	24.4%	10.4
mux8_64bit	2516	0.38	9.38e+4	1.01e+5	1.36e+5	**8.55e+4**	15.3%	8.3	10.1	12.3	**7.9**	22.3%	2.7
nut_000	1852	0.46	5.28e+4	6.43e+4	6.44e+4	6.24e+4	3.0%	4.7	7.5	5.7	5.6	2.6%	2.3
nut_001	3799	0.49	7.84e+4	1.29e+5	1.15e+5	1.03e+5	10.4%	7.0	19.2	13.8	10.8	21.7%	16.7
nut_002	1197	0.41	2.76e+4	3.14e+4	3.66e+4	2.90e+4	7.6%	2.4	3.4	3.3	3.1	4.6%	1.2
nut_003	2358	0.65	4.29e+4	4.36e+4	4.98e+4	**3.88e+4**	11.0%	3.8	4.2	4.4	**3.8**	7.9%	2.6
oc_ata_ocidec2	5001	0.93	4.91e+4	9.64e+4	5.30e+4	**4.89e+4**	7.7%	4.4	4.7	4.7	4.5	4.0%	5.1
oc_ata_v_3	4977	0.93	2.54e+4	9.64e+4	3.06e+4	2.82e+4	7.8%	2.3	3.0	2.8	2.7	4.6%	5.8
oc_cordir_p2r	9231	0.71	1.17e+5	3.14e+5	2.90e+5	2.53e+5	12.8%	10.4	57.7	38.6	22.0	43.0%	149.8
oc_hdlc	3017	0.93	6.90e+4	6.89e+4	7.00e+4	**6.11e+4**	11.3%	6.1	6.1	6.2	**5.5**	10.1%	16.3
oc_minirisc	2540	0.92	4.68e+4	4.72e+4	4.72e+4	4.72e+4	0.0%	4.2	4.3	4.3	**4.2**	0.7%	3.9
oc_mips	17490	0.95	2.04e+5	3.02e+5	2.87e+5	2.30e+5	19.9%	18.1	41.7	25.5	20.8	18.4%	380.9
oc_oc8051	12239	0.95	1.22e+5	1.22e+5	1.24e+5	**1.21e+5**	0.8%	10.9	10.9	11.1	**10.7**	1.8%	164.9
oc_pavr	15958	0.82	2.00e+5	2.68e+5	2.98e+5	2.55e+5	4.9%	17.7	34.1	26.5	23.3	12.1%	292.0
oc_pci	13873	0.55	2.19e+5	2.29e+5	2.38e+5	2.24e+5	2.2%	19.5	21.8	21.2	20.3	4.2%	90.1
oc_vga_lcd	12199	0.71	1.80e+5	1.81e+5	2.04e+5	**1.76e+5**	2.8%	16.0	16.4	18.1	**15.9**	3.0%	293.4
oc_wb_dma	17246	0.87	2.88e+5	2.96e+5	3.08e+5	2.92e+5	1.4%	25.6	27.7	27.4	26.1	4.7%	614.7
os_blowfish	14642	0.53	1.44e+5	2.09e+5	1.79e+5	**1.43e+5**	20.1%	12.8	28.3	20.2	14.0	30.7%	165.9
radar12	57463	0.51	6.28e+5	7.05e+5	6.54e+5	6.53e+5	0.2%	55.8	74.8	59.5	59.4	0.2%	2337.0
AVERAGE							**7.8%**					**10.7%**	

[3] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 61225. http://www.eecs.berkeley.edu/~alanmi/abc/

[4] J. Cochet-Terrasson, G. Cohen, S. Gaubert, .M. McGettrick, and J.-P. Quadrat, “Numerical computation of spectral elements in max-plus algebra”, In Proc. IFAC Conf. on Syst. Structure and Control, 1998.

[5] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. The MIT Press, 2d ed., 2001.

[6] O. Coudert, “Gate sizing for constrained delay/power/area optimization,” IEEE Trans. on VLSI, pp. 465-472, Dec 1997.

[7] A. Dasdan, S. Irani, and R. K. Gupta, “Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems”, Proceedings of the 36th DAC, June 1999, pp. 37-42.

[8] J. P. Fishburn, “Clock skew optimization,” IEEE Trans. on Computing, vol. 39(7), pp. 945–951, July 1990.

[9] S. Held, B. Korte, J. Maβberg, M. Ringe, and J. Vygen. “Clock scheduling and clock tree construction for high performance ASICs”. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2003, pp. 756 - 761.

[10] A. Hurst, A. Mishchenko, and R. K. Brayton, “Minimum-register retiming via binary maximum-flow,” ERL Technical Report, UC Berkeley, December 2006.

[11] D. Joy and M. Ciesielski, "Clock period minimization with wave pipelining", IEEE Transactions on Computer-Aided Design, vol. 12(4), pp. 461-472, Apr. 1993.

[12] I. S. Kourtev and E. G. Friedman. “Simultaneous clock scheduling and buffered clock tree synthesis”. Proc. of the IEEE Inl. Symp. an Circuits and Systems, 1997, pp. 1812-1815.

[13] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," Algorithmica, Vol. 6(1), pp. 5-35, 1991.

[14] X. Liu, M. C. Papaefthymiou, and E. G. Friedman, “Maximizing performance by retiming and clock skew scheduling”, Proc. of the 36th DAC, 1999, pp. 231-236.

[15] X. Liu, M. Papaefthymiou, and E.G. Friedman, “Retiming and clock scheduling for digital circuit optimization,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 184-203, Feb 2002.

[16] N. Maheshwari , and S. Sapatnekar, “Efficient minarea retiming of large level-clocked circuits”, Proceedings of DATE 1998, pp. 840-845.

[17] S. S. Sapatnekar and R. B. Deokar, "Utilizing the retiming skew equivalence in a practical algorithm for retiming large circuits," IEEE Transactions on Computer-Aided Design, vol. 15(10), pp. 1237-1248, Oct. 1996.

[18] D. Singh, V. Manohararajah, and S. D. Brown, "Incremental retiming for FPGA physical synthesis," Proceedings of the 42nd DAC, pp. 433-438, 2005.

[19] B. Taskin, and I. Kourtev, “Performance optimization of single-phase level-sensitive circuits using time borrowing and non-zero clock skew”, In Proceedings of the 8th Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, 2002.

[20] H. Zhou, “Deriving a new efficient algorithm for min-period retiming”, Proceedings of ASPDAC, 2005, pp. 990-993.