Retiming Pulsed-Latch Circuits with Regulating Pulse Width

Seungwhun Paik, *Student Member, IEEE,* Seonggwan Lee, and Youngsoo Shin, *Senior Member, IEEE*

***Abstract*—A pulsed-latch is an ideal sequencing element for high-performance application-specific integrated circuit designs due to its simple timing model and reduced sequencing overhead. The possibility of time-borrowing while a latch is transparent is deliberately ignored in pulsed-latch circuits to simplify the timing model. However, using more than one pulse width allows another form of time-borrowing, which preserves the simple timing model. The associated problem of allocating pulse widths is called pulse width allocation (PWA); and we combine it with retiming to achieve a shorter clock period in pulsed-latch circuits than can be obtained by retiming or PWA alone, with less requirement for extra latches than standard retiming. An exact solution can be obtained by an integer linear programming, but this is restricted to small circuits. We therefore introduce a practical heuristic, which performs clock skew scheduling to find the minimum clock period and then brings the clock skew as nearly back to zero as possible by converting it to combined retiming and PWA. Experiments with 45 nm technology circuits suggest that the heuristic algorithm achieves a clock period that is close to minimum in most circuits, with an average 23% reduction compared to initial circuit, at an average cost of a 16% increase in the number of latches. On the same benchmarks, standard retiming achieves a 20% reduction in clock period with a 29% increase in the number of latches. The cost in extra area and energy is reasonable.**

***Index Terms*—Pulsed-latch, retiming, sequential circuit, time-borrowing.**

I. Introduction

THERE are two widely used sequencing elements: edge-triggered flip-flops and level-sensitive latches. Flip-flops are mostly used in application-specific integrated circuit (ASIC) designs due to their simplicity of the timing model. The amount of time available to a combinational block that lies between two flip-flops is fixed, as shown in Fig. 1(a). This constrains timing uncertainties within a combinational block, which is important in the synthesis and optimization of ASIC designs. Latches are frequently used in high-performance custom designs because they have a smaller sequencing overhead than flip-flops. However, their timing model is more complicated. As shown in Fig. 1(b), the amount of time available to a combinational block between two latches varies a lot; this variation can be almost as large as one and a half clock periods, or about as small as half a clock period. In addition, data has to be held for a longer period of time, increasing the likely number of hold-time violations.

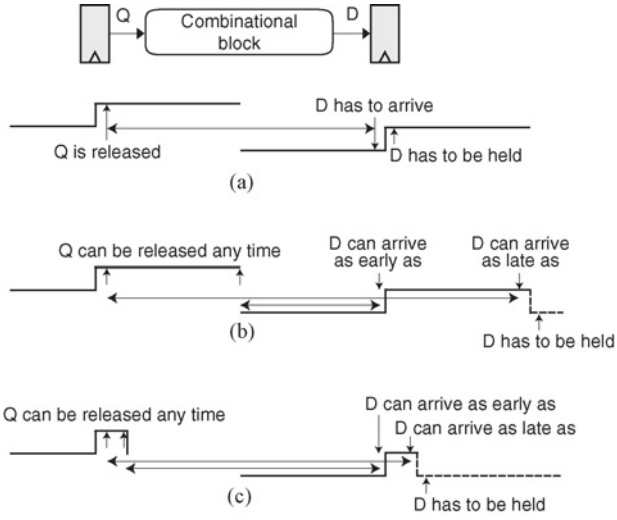

Fig. 1. Comparison of timing models. (a) Edge-triggered flip-flop. (b) Level-sensitive latch. (c) Pulsed-latch.

A pulsed-latch is a latch driven by a brief clock pulse. As shown in Fig. 1(c), the amount of time available to a combinational block is still variable, but the amount of the variation is significantly less than it is in latch circuits. The scope for hold-time violations is also reduced. This makes a pulsed-latch an ideal sequencing element for high-performance [2] or low-power ASIC designs [3], as well as for traditional high-performance microprocessors [4]–[10].

A. Design of Pulsed-Latch Circuits

In pulsed-latch circuits, a normal clock, with a 50% duty ratio, is delivered from a clock source to multiple pulse generators; we refer to this as global clock distribution. Each pulse generator [5], [9] then delivers a pulse to more than one latch, which we call local clock distribution. Since pulses can easily be distorted, local clock distribution has to be carefully designed in pulsed-latch circuits. In particular, a pulse generator and its latches have to be physically close to preserve the pulse shape, and this is a constraint on placement [11].

Manuscript received February 10, 2011; accepted February 20, 2011. Date of current version July 20, 2011. This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund), no. KRF-2008-331-D00406. A preliminary version of this paper [1] was presented at the International Conference on Computer-Aided Design, San Jose, CA, November 2–5, 2009. This paper was recommended by Associate Editor S. Nowick.

S. Paik and Y. Shin are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: swpaik@dtlab.kaist.ac.kr; youngsoo@ee.kaist.ac.kr).

S. Lee is with LG Electronics, Seoul 150-721, Korea (e-mail: seonggwan.lee@lge.com).

Digital Object Identifier 10.1109/TCAD.2011.2126932

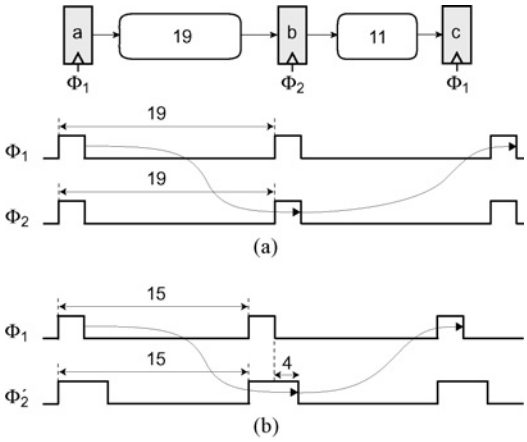

Fig. 2. (a) Same pulse clock applied to all latches. (b) Wider pulse clock applied to *b*.

Even though a pulse is very short, a small amount of time-borrowing, depending on the pulse width, is still possible. In the design of ASICs, however, this possibility is ignored to simplify the timing model. Each combinational block between latches is allowed the interval between two sequential falling (or rising) edges of the pulse, and this period must include the sequencing overhead of the latches. In consequence, the same timing model can be used for flip-flop and pulsed-latch circuits.

Even though the time-borrowing inherently available within a latch is deliberately ignored in this scheme, another form of time-borrowing is possible by using different pulse widths. This is illustrated in Fig. 2. Suppose three latches *a*, *b*, and *c* are driven by exactly the same pulse clock [Fig. 2(a)]. If the maximum combinational delay between *a* and *b* is 19 time units and that between *b* and *c* is 11, then the clock period has to be 19 time units (if the sequencing overhead is ignored). However, if *b* is driven by a new pulse clock, ϕ_2', which is 4 time units wider than the original, then the clock period can be reduced to 15 [Fig. 2(b)]. The combinational block between *a* and *b* is now allocated the interval of a clock period plus 4, which is equivalent to borrowing 4 time units from the adjacent combinational block between *b* and *c*. Note that this comes at the risk of more hold-time violations. We can generalize this problem to allocating pulse widths [2] from a list of widths determined by a library of pulse generators, such that the clock period is minimized. We call this problem pulse width allocation (PWA).

B. Motivation

PWA provides a new handle on optimizing the performance of pulsed-latch circuits; but the extent of time-borrowing is necessarily limited due to the increasing risk of hold-time violations and the limited number of different pulse generators that can be included in a practical design: these limitations are assessed in a quantitative way in Section VI. However, the benefits of PWA can also be exploited by combining it with other sequential optimization techniques such as clock skew scheduling [12] or retiming [13]. Clock skew scheduling has its own limitation; even a small amount of skew is very difficult to realize reliably in today's circuits [14], [15] due to the growth of within-die process variations. For example, we took a 4-level buffered clock-tree and implemented 76 ps of skew in a 0.99 V, 45 nm industrial technology. Monte Carlo simulation showed a standard deviation of 15 ps, which is 20% of the target skew.

In this paper, we consider how PWA can be combined with retiming to optimize the performance of pulsed-latch circuits. These two techniques complement each other: retiming achieves what PWA could do if we could accept more hold-time violations; while PWA achieves what retiming could do if we could accept a large increase in the number of latches. This increase is the main practical limitation of retiming: it is commonly observed that retiming can double or triple the number of sequencing elements [16].

C. Main Contributions

The main contributions of this paper are as follows:

1) formulating a new problem in which retiming and PWA are combined to minimize the clock period of pulsed-latch circuits;
2) an integer linear programming (ILP) approach to obtaining an optimal solution;
3) a heuristic algorithm which accepts the maximum number of extra latches that can be tolerated as a parameter;
4) experiments to assess the heuristic algorithm in terms of clock period, circuit area, and energy dissipation using 45 nm industrial technology, as well as comparing it with ILP.

The remainder of this paper is organized as follows. We first review related works in Section II. In Section III, we briefly overview existing methods of retiming to minimize the clock period, which we then extend to the problem of combined retiming and PWA. We first solve this problem by formulating it as an ILP in Section IV. We go on to propose a heuristic algorithm in Section V, which relies on the equivalence of clock skew and retiming. We initially perform clock skew scheduling to minimize the clock period; then the skew is gradually brought to zero by converting it to retiming and PWA. Experimental results are reported in Section VI, and we summarize this paper and look at potential future work in Section VII.

II. Related Work

Several retiming works are reviewed in this section to put our work in perspective. The equivalence of retiming and clock skew has been well known since the introduction of clock skew scheduling [12]. It is employed to implement a fast retiming algorithm [16], which first performs clock skew scheduling minimizing the clock period and then gradually converts the skews into retiming. A similar approach is adopted in our heuristic for mixed retiming and PWA. Algorithm derivation was used as an another approach to implement a fast retiming algorithm [17], [18].

Retiming often causes unacceptable increase in the number of flip-flops (or latches). It can be alleviated by a combined

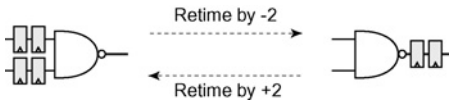

Fig. 3. Retiming operations.

application of retiming and clock skew scheduling. Several approaches have been proposed in this direction; some of them consider the problem under both setup-time and hold-time constraints [19], [20] and some others under setup-time constraints [21] alone.

Latch-based circuits with non-overlapping two-phase clocking can benefit from time-borrowing, thus have a potential to achieve smaller clock period than flip-flop-based circuits; the benefit comes at the cost of more complex timing analysis. Retiming of latch-based circuits with no feedback, i.e., pipelined circuits, is investigated and a polynomial-time algorithm is presented [22]. Clock tuning is an approach to adjust the duty cycle of non-overlapping two-phase clock; it can be combined with retiming to achieve a minimum clock period in $O(V^{11})$ complexity [23], where V is the number of combinational gates. Clock tuning is similar to PWA but there are several differences: clock tuning is a continuous optimization while PWA is a discrete one; two phases have to be assigned alternately in two adjacent latches in clock tuning while more than two pulses can be assigned without any constraint in PWA. The comparison of clock tuning and PWA from a minimum clock period standpoint is an open problem.

In the optimization methods mentioned above, hold-time violations are often ignored [16], [21] and are fixed during post-optimization process by inserting delay buffers to the paths that are violated. This approach usually achieves smaller clock period than the approaches that consider both constraints, but at the cost of extra buffers.

III. Problem Formulation

We will briefly review existing methods of retiming [24] to minimize the clock period in sequential circuits. We then formulate the corresponding retiming problem for pulsed-latch circuits, in which we consider both retiming and PWA.

A. Retiming

A sequential circuit can be modeled as a directed graph $G = (V, E, d, l)$, where $v \in V$ represents a combinational gate with propagation delay $d(v)$, and $e_{uv} \in E$ models a connection from u to v, on which the number of latches (or flip-flops in general) is denoted by $l(e_{uv})$. A retiming is the assignment of an integer number to each vertex: thus, for instance, retiming a NAND gate by 2 implies the movement of two latches from its output to each of its input, and retiming by -2 involves two latches being relocated in the opposite direction, as illustrated in Fig. 3.

The number of latches on e_{uv} after retiming, denoted by $l_r(e_{uv})$, is given by

$$l_r(e_{uv}) = l(e_{uv}) + r(v) - r(u) \tag{1}$$

where $r(v)$ is a retiming of v, which is the number of latches that are moved from its output to each of its inputs. Of course, the number of latches after retiming cannot be negative, and so $l_r(e_{uv}) \geq 0$, which implies that a *legal* retiming must meet the following condition:

$$r(u) - r(v) \leq l(e_{uv}) \qquad \forall e_{uv} \in E. \tag{2}$$

The problem of finding a retiming that minimizes the clock period can be posed as follows.

Problem 1 Given a sequential circuit $G = (V, E, d, l)$, the retiming problem is to find a legal retiming $r : V \rightarrow Z$ such that the clock period $\phi = \max_{p:\, l_r(p)=0} d(p)$ is minimized.

A path through $v_1, v_2, \ldots, v_n$ is denoted by p; if the total number of latches on edges within the path after retiming, denoted by $l_r(p)$, is 0, then p is a combinational path; it can readily be shown that $l_r(p) = l(p) + r(v_n) - r(v_1)$. A path delay $d(p)$ is defined by

$$d(p) = \sum_{i=1}^{n} d(v_i). \tag{3}$$

The clock period ϕ is therefore the maximum $d(p)$ for all combinational paths p.

For a particular clock period ϕ, a retiming has to satisfy the following condition for correct timing:

$$r(u) - r(v) \leq L(u, v) - 1 \qquad \forall u, v :\ D(u, v) > \phi \tag{4}$$

where $L(u, v)$ indicates the minimum number of latches on the paths from u to v, and $D(u, v)$ denotes the maximum path delay when there are $L(u, v)$ latches;[1] thus $D(u, v)$ is the tightest timing requirement between u and v. A system of difference constraints (2) and (4) can be represented by a constraint graph, in which the nodes correspond to the unknown values of each $r(u)$ and $r(v)$, and an edge from each $r(u)$ to each $r(v)$ is assigned a weight $l(e_{uv})$ or $L(u, v) - 1$. A solution does not exist (ϕ is unsatisfiable) if there is any negative weighted cycle, otherwise the values of every $r(u)$ and $r(v)$ can be determined. An alternative algorithm with lower time-complexity exists [24], which does not require the computation of $D(u, v)$ and $L(u, v)$. To solve Problem 1, the process of setting up the difference constraints, deriving the corresponding constraint graph, and checking for negative weighted cycles is repeated for a value of ϕ. This procedure is repeated within a binary search for the minimum value of ϕ.

B. Retiming Pulsed-Latch Circuits with PWA

For retiming pulsed-latch circuits with pulse width allocation (which we will call retiming pulsed-latches for brevity), we refine the graph model of a sequential circuit described in Section III-A by adding the pulse width at each edge. A sequential circuit is now modeled as a directed graph $G = (V, E, d, l, w)$, in which a value of w is assigned to each edge $e \in E$ to indicate the pulse widths supplied to each of

[1] If the path delay from u to v is larger than ϕ, then that path has to be a non-combinational path; thus we require at least one latch on the path after retiming, so that $L(u, v) + r(v) - r(u) \geq 1$, which yields (4).

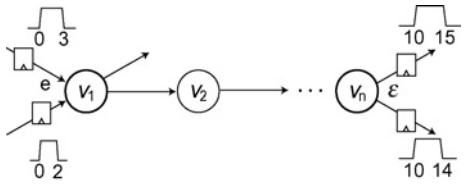

Fig. 4. Path delay with time-borrowing.

the latches on it. A library of pulse generators defines the set of available pulse widths $\mathcal{W} = \{W_0, W_1, \ldots, W_n\}$, where n is the total number of pulse generators and $W_0 = 0$ is introduced to allow us to assign a zero pulse width to an edge on which there are no latches; thus $l(e) = 0$ implies $w(e) = W_0$. We now state the problem of retiming pulsed-latches.

Problem 2 Given a pulsed-latch circuit $G = (V, E, d, l, w)$, the problem of retiming pulsed-latches is to find a legal retiming $r : V \to Z$ and PWA $w : E \to \mathcal{W}$ such that the clock period $\phi = \max_{p:\, l_r(p)=0} d(p)$ is minimized.

The path delay (3) must be refined to account for the amount of time-borrowing due to the difference in pulse widths (recall Fig. 2)

$$d(p) = \sum_{i=1}^{n} d(v_i) - \left(\min_{\epsilon \in out(v_n)} w(\epsilon) - \max_{e \in in(v_1)} w(e) \right) \tag{5}$$

where $out(v_n)$ is the set of outgoing edges of v_n, and $in(v_1)$ is the set of incoming edges of v_1. Fig. 4 shows an example, in which there are two incoming edges of v_1, each of which has a latch; if there is no clock skew, then time 3 is the latest at which the data is available to v_1 (we ignore the setup time and the data-to-Q delay of the latch for simplicity.). We assume that v_n has two outgoing edges; thus time 14 is the latest by which the data from v_n must arrive so that it can be safely captured by the latches. The amount of time borrowed by path p: $v_1, v_2, \ldots, v_n$ is $4-3 = 1$ for a clock period of 10; therefore $\sum d(v_i)$ can be as large as 11. The amount of time-borrowing has to be subtracted from $\sum d(v_i)$ in (5) when we use the path delay $d(p)$ to determine the clock period.

1) *Considerations of Delay Accuracy:* In the retiming formulations of Sections III-A and III-B, we assumed that the gate delay $d(v)$ is a constant. In practice, however, a retiming alters the load capacitance of the fan-in gates of a latch before and after it is relocated; $d(v)$ for the corresponding gates must therefore be adjusted accordingly. Furthermore, the sequencing overhead of the latch (consisting of the setup time t_{su} and the data-to-Q delay t_{dq}) has to be included in the formulation.

These problems have been addressed in the variable register delay model [25], [26], which forms part of our scheme. If a latch is relocated to e_{uv} (en edge between u and v) by a retiming, then the delay of each combinational path that is led by v is increased by t_{dq}, and the delay of each combinational path that sinks at u is increased by $t_{su} + \delta(u)$. The change in delay at u, due to the change in its load capacitance, is now $\delta(u)$, since its load capacitance is now the input capacitance of the latch but not the input capacitance of v. Note that $\delta(u)$ can be either positive or negative.

The expression for path delay (5) must be refined once again to take account of these changes

$$d(p) = \sum_{i=1}^{n} d(v_i) - \left(\min_{\epsilon \in out(v_n)} w(\epsilon) - \max_{e \in in(v_1)} w(e) \right) + t_{dq} + t_{su} + \delta(v_n). \tag{6}$$

In this formulation, $d(v)$ represents the propagation delay through v when it fans out to combinational gates alone. It should be noted that t_{su} and especially t_{dq} are not constant in practice, meaning that their values have to be considered for each edge of $out(v_n)$ and $in(v_1)$, respectively.

IV. ILP for Retiming Pulsed-Latches

Problem 2 can be solved by integer linear programming (ILP). The unknowns are the variables $r(v)$ at all the vertices and $w(e)$ on all the edges; the objective is to minimize ϕ, which we can consider as an integer variable without loss of generality if we use a time unit that is small enough (e.g., pico second).

A. Constraints

The ILP formulation is subject to the legal retiming, pulse width, and timing constraints.

1) *Legal Retiming Constraint:* The number of latches on e_{uv} after retiming, $l_r(e_{uv})$, has to be non-negative, which is expressed by (2), which we shall repeat here for convenience

$$r(u) - r(v) \le l(e_{uv}) \qquad \forall e_{uv} \in E. \tag{7}$$

2) *Pulse Width Constraint:* If there are one or more latches on an edge e after retiming, those latches must have non-zero pulse widths

$$w(e) > 0 \quad \text{if} \quad l_r(e) > 0. \tag{8}$$

Otherwise, the pulse width has to be zero

$$w(e) = 0 \quad \text{if} \quad l_r(e) = 0. \tag{9}$$

To linearize (8) and (9), we introduce a decision variable $\Omega_i(e) \in \{0, 1\}$ for each possible pulse width $i = 0, 1, \ldots, n$

$$\Omega_i(e) = \begin{cases} 1, & \text{if } W_i \text{ is assigned to } e \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\sum_{i=0}^{n} \Omega_i(e) = 1$ always holds; note also that $\sum_{i=1}^{n} \Omega_i(e) = 0$ when $w(e) = 0$, and $\sum_{i=1}^{n} \Omega_i(e) = 1$ when $w(e) > 0$ (thus $\Omega_0(e) = 1$ when $W_0 = 0$ is assigned to $w(e)$). We further note that $l_r(e) \in \{0, 1, \ldots, M\}$, where M is the maximum number of latches that can be placed on e; the value of M can be obtained by taking the maximum number of latches in any cycle of G or in any path from an input to an output of the circuit: the numbers of latches in all these cycles or paths are unchanged by retiming.

We can now rewrite (8) as follows:

$$1 \le l_r(e) \le M \quad \text{if and only if} \quad \sum_{i=1}^{n} \Omega_i(e) = 1 \tag{10}$$

provided that $l_r(e) \in \{0, 1, \ldots, M\}$ and $\sum_{i=1}^{n} \Omega_i(e) \in \{0, 1\}$. Note that this also implies (9). The pairs of $l_r(e)$ and

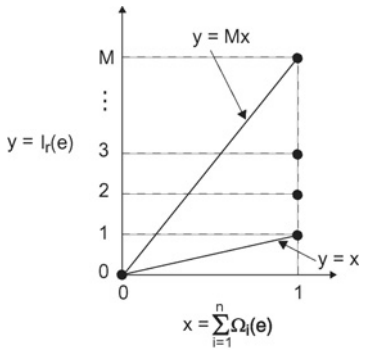

Fig. 5. Points that satisfy the legal retiming constraint (11).

$\sum_{i=1}^{n} \Omega_i(e)$ that satisfy (10) are shown diagrammatically in Fig. 5, from which we can see that all the points lie between or on the two lines $y = x$ and $y = Mx$. Therefore we can express the pulse width constraint in this final linear form

$$\sum_{i=1}^{n} \Omega_i(e) \le l_r(e) \le M \sum_{i=1}^{n} \Omega_i(e). \tag{11}$$

Note that (7) is subsumed in this constraint: if this constraint is satisfied, then $l_r(e)$ is non-negative and $\sum_{i=1}^{n} \Omega_i(e)$ is also non-negative. Therefore, the legal retiming constraint and pulse width constraint are unified into this linear inequality.

3) *Timing Constraint:* The delay of each combinational path between a pair of latches, expressed by (5), has to be no greater than the clock period ϕ

$$\begin{aligned} \text{if } & L(u, v) + r(v) - r(u) = 0 \ \wedge \ l_r(e) \neq 0 \ \wedge \ l_r(\epsilon) \neq 0 \\ & \text{then } D(u, v) + w(e) - w(\epsilon) - \phi \le 0 \end{aligned} \tag{12}$$

for all pairs of e and ϵ, where $e \in in(u)$ and $\epsilon \in out(v)$ (see Fig. 4). Note that, in Fig. 4, the delay in the combinational path that does not occur between latches, say $v_1, \ldots, v_{n-1}$, can be larger than the clock period, even though the path delay $d(p)$ between v_1 and v_n does not exceed the clock period; this is because of time-borrowing, which is accounted for in $d(p)$ but not in the path between v_1 and v_{n-1}. Therefore, (12) only needs to be checked strictly in paths between two latches.

Condition (12) is equivalent to

$$\begin{aligned} L(u, v) + r(v) \quad & -r(u) \ge 1 \ \vee \ l_r(e) = 0 \ \vee \ l_r(\epsilon) = 0 \\ & \vee \ D(u, v) + w(e) - w(\epsilon) - \phi \le 0. \end{aligned} \tag{13}$$

Note that $L(u, v)+r(v)-r(u)$ is the minimum number of latches on the paths from u to v after retiming, which we denote as $L_r(u, v)$; $D(u, v)+w(e)-w(\epsilon)-\phi$ corresponds to the maximum delay on the paths accounting for time-borrowing [recall (5)] subtracted by the clock period; this is the extent by which the timing violation exceeds beyond the clock period, and we denote it as $V(e, \epsilon)$. If there is no latch, the pulse width is zero, i.e., $l_r(e) = 0$ if and only if $\Omega_0(e) = 1$. Condition (13) can now be re-written as

$$L_r(u, v) \ge 1 \ \vee \ \Omega_0(e) = 1 \ \vee \ \Omega_0(\epsilon) = 1 \ \vee \ V(e, \epsilon) \le 0 \tag{14}$$

which is equivalent to

$$L_r(u, v) + \Omega_0(e) + \Omega_0(\epsilon) \ge 1 \ \vee \ V(e, \epsilon) \le 0 \tag{15}$$

provided that the retiming is legal, so that $L_r(u, v) \ge 0$ (Ω_0 is either 0 or 1).

We can convert (15) to a linear form, using the upper bound on $V(e, \epsilon)$, which is given by

$$\begin{aligned} V_{ub} \quad & = \max_{\forall u,v} D(u, v) + \max_{i \in \mathcal{W}} i - \min_{i \in \mathcal{W}} i - \min \phi \\ & = \max_{\forall u,v} D(u, v) + W_n - \phi_{lb} \end{aligned} \tag{16}$$

where ϕ_{lb} is a lower bound on the clock period, which can be obtained from the maximum delay from any latch to itself, or the minimum delay between all latch pairs. Clearly V_{ub} is a positive constant.

We now consider (15) in the two separate cases of $V(e, \epsilon) \le 0$ and $V(e, \epsilon) > 0$. If $V(e, \epsilon) \le 0$, then $L_r(u, v)+\Omega_0(e)+\Omega_0(\epsilon)$ is allowed to take any value, provided that it is non-negative, so as to guarantee legal retiming; thus $L_r(u, v)+\Omega_0(e)+\Omega_0(\epsilon) \ge 0$ and this is equivalent to

$$L_r(u, v) + \Omega_0(e) + \Omega_0(\epsilon) \ge \frac{V(e, \epsilon)}{V_{ub}} \tag{17}$$

because the right-hand side satisfies $-1 < V(e, \epsilon)/V_{ub} \le 0$ and $L_r(u, v)+\Omega_0(e)+\Omega_0(\epsilon)$ is an integer value. If $V(e, \epsilon) > 0$, then we must satisfy $L_r(u, v) + \Omega_0(e) + \Omega_0(\epsilon) \ge 1$, but this is equivalent to (17) because its right-hand side satisfies $0 < V(e, \epsilon)/V_{ub} \le 1$. Hence, (15) is equivalent to (17), which is a linear form of the constraint. Note that this formulation can easily be extended to take account of accurate delay information by using the path delay of (6) instead of (5).

a) *Hold-time constraint:* Constraint (12) ensures that setup-time constraint is satisfied at all latches. The hold-time constraint can be formulated in a similar way if we want to handle it as an integral part of ILP formulation. The delay of each combinational path between a pair of latches has to be no smaller than zero

$$\begin{aligned} \text{if } & L(u, v) + r(v) - r(u) = 0 \ \wedge \ l_r(e) \neq 0 \ \wedge \ l_r(\epsilon) \neq 0 \\ & \text{then } D'(u, v) - w(\epsilon) \ge 0 \end{aligned} \tag{18}$$

for all pairs of e and ϵ, where $D'(u, v)$ denotes the minimum path delay from u to v. Only $w(\epsilon)$ is considered when checking the delay condition for the hold-time constraint. The earliest time at which data can depart from a launching latch is the rising edge of the clock and therefore $w(e)$, which is the pulse width of the launching latch, does not affect short paths [2]. Following the similar procedure to derive (17) from (12), the linear from of (18) is given by

$$L_r(u, v) + \Omega_0(e) + \Omega_0(\epsilon) \ge \frac{V'(e, \epsilon)}{V'_{ub}} \tag{19}$$

where $V'(e, \epsilon) = -D'(u, v) + w(\epsilon)$ and $V'_{ub} = W_n - \min_{\forall u,v} D'(u, v)$.

B. ILP Formulation

The ILP form of Problem 2 can now be summarized by the objective function

$$\text{Minimize} \quad \phi \tag{20}$$

with the legal-retiming and pulse-width constraint given by a unified expression

$$\sum_{i=1}^{n} \Omega_i(e) \le l(e_{uv}) + r(v) - r(u) \le M \sum_{i=1}^{n} \Omega_i(e) \quad \forall e_{uv} \in E \tag{21}$$

and the timing constraint

$$L(u, v) + r(v) - r(u) + \Omega_0(e) + \Omega_0(\epsilon) \ge \frac{V(e, \epsilon)}{V_{ub}} \quad \forall e, \epsilon \in E : e \in in(u), \epsilon \in out(v). \tag{22}$$

The hold-time constraint (19) can be formulated similar to (22).

Retiming often generates a large number of latches. To avoid this, we can minimize the number of extra latches along with the clock period, which leads to the new objective function

$$\text{Minimize} \quad \alpha\phi + \sum_{\forall e_{uv} \in E} (r(v) - r(u)) \tag{23}$$

where α is a weighting factor. We typically choose a very large value for α so that the clock period is not sacrificed to reduce the number of latches. If there is more than one solution with the same clock period, then we want the solution that requires fewer number of latches; otherwise we always take the solution with the shorter clock period.

V. Heuristic Algorithm

The ILP approach to the solution of Problem 2 that we described in the previous section can only be applied to circuits of very small size due to the computation time that it requires; we will illustrate this experimentally in Section VI-C. We have therefore developed a heuristic algorithm, which consists of the following three steps.

1) Perform clock skew scheduling (CSS) minimizing the clock period.
2) Gradually bring clock skew as close to zero as possible by converting it to retiming and PWA.
3) Remove any remaining clock skew, fix any hold-time violations, and determine the final clock period.

The foundation of this algorithm is the equivalence of retiming and clock skew [12], [16], [19], [21]: if a certain clock period is achievable by retiming, the same clock period can be achieved by scheduling the clock skew. We therefore begin by performing CSS to minimize the clock period first; this is a continuous optimization and thus is easier than combined retiming and PWA. We then try to achieve the same effect by converting skew to retiming and PWA, which is a discrete optimization.

A. CSS to Minimize the Clock Period

Fig. 6 shows the pseudocode for the first step in our algorithm. *CSS_Optimize* seeks the minimum clock period by iteratively checking (by calling the function *CSS* in L3) whether a particular clock period ϕ can be achieved by clock skew scheduling. This is done by a binary search (L1 to L5), starting from the initial values of the upper and lower bounds on the clock period, ϕ_{ub} and ϕ_{lb}. This requires $\log_2(\phi_{ub}-\phi_{lb})/\epsilon$ iterations, where ϵ is a suitable increment of clock period, such as 1 ps. The initial value of ϕ_{ub} can be set to the maximum delay between all latch pairs; and either the maximum delay from any latch to itself or the minimum delay between all latch pairs can serve as an initial value for ϕ_{lb}.

Algorithm *CSS_Optimize* $(\phi_{ub}, \phi_{lb}, \epsilon)$
L1 **while** $(\phi_{ub} - \phi_{lb}) > \epsilon$ **do**
L2 $\phi \leftarrow (\phi_{ub} + \phi_{lb})/2$
L3 **if** *CSS* (ϕ) = success **then**
L4 $\phi_{ub} \leftarrow \phi$
L5 **else** $\phi_{lb} \leftarrow \phi$
L6 $\phi_{min} \leftarrow \phi$

Function *CSS* (ϕ)
L7 $s_i \leftarrow 0$, for all latches i
L8 $C(i,j)$: $s_j \ge s_i + (t_{dq} + D_{ij} - \phi), \forall i \rightsquigarrow j$
L9 **while** $\exists_{i,j}, C(i,j)$ is not satisfied **do**
L10 $s_j \leftarrow s_i + (t_{dq} + D_{ij} - \phi)$
L11 **if** $s_j > \phi$ **then return** fail
L12 **return** success

Fig. 6. Clock skew scheduling to minimize the clock period: *CSS_Optimize* finds the skew s_i at each latch i that corresponds to the minimum clock period $\phi_{\min}$.

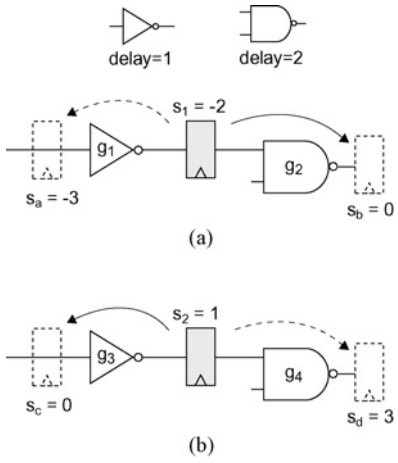

Fig. 7. Conversion of skew to retiming. (a) Only forward retiming can reduce the magnitude of negative skew. (b) Only backward retiming can reduce the magnitude of positive skew.

For a particular value of ϕ, the function *CSS* searches for an assignment of skew that satisfies all the timing constraints. In the implementation of this function, we consider only the setup-time constraints between all pairs of latches i and j of the form

$$s_j \ge s_i + \left(t_{dq} + D_{ij} - \phi\right) \tag{24}$$

where s_i denotes an unknown clock skew and D_{ij} corresponds to the maximum delay of the combinational block between i and j. Extra buffers are subsequently introduced (see Section V-C) into logic paths that violate hold-time constraints; this approach is common to many other methods of clock skew scheduling [16], [27], [28].

All skews are initially set to 0 (L7). We take any unsatisfied constraint $C(i, j)$ between the latches i and j (L9). We set s_j to $s_i + (t_{dq} + D_{ij} - \phi)$ (L10) so that $C(i, j)$ becomes satisfied: in effect we are forcing the left-hand and right-hand sides of

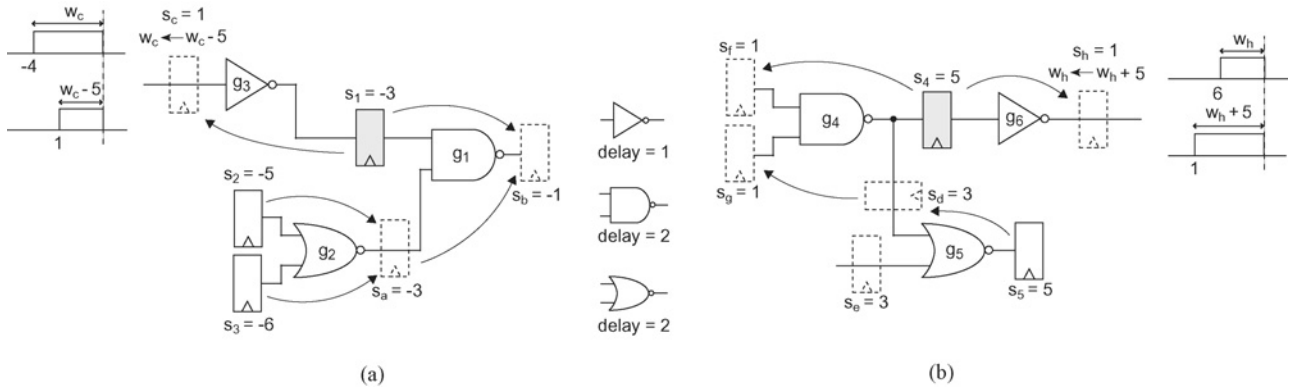

Fig. 8. (a) Forward retiming reduces $|s_1| = 3$ to $|s_b| = 1$, backward retiming while decreasing the pulse width by 5 also achieves $|s_c| = 1$. (b) Backward retiming reduces $|s_4| = 5$ to $|s_f| = |s_g| = 1$, forward retiming while increasing the pulse width by 5 achieves $|s_h| = 1$.

$C(i, j)$ to be equal. If this s_j is larger than ϕ, then the function fails (L11); otherwise we repeat the process (L9 to L11) until there are no more unsatisfied setup-time constraints. Note that this problem is equivalent to the longest-path problem, which can be solved by the Bellman–Ford algorithm. Each vertex corresponds to a latch, and an edge from i to j has a weight of $t_{dq} + D_{ij} - \phi$. Therefore, the successive iterations of L9 can be considered as a "relaxation" approach to the solution of Bellman's equations. Each $C(i, j)$ is checked $m - 1$ times in the worst case, where m is the number of latches.

There are at most $m(m + 1)/2$ unsatisfied setup-time constraints, and each constraint is checked up to $m-1$ times. Thus the function *CSS* is bounded by $O(m^3)$, and the worst-case running-time of *CSS_Optimize* is $O(m^3 \log_2(\phi_{ub} - \phi_{lb})/\epsilon)$. In practice, $\log_2(\phi_{ub} - \phi_{lb})/\epsilon$ can be considered as a constant; it was smaller than 16 for all the benchmark circuits we tested. The clock period may be as high as a few nano seconds (corresponding to hundreds of MHz) in contemporary designs, and a few pico seconds can be used as a value for ϵ, independent of the circuit under consideration. The practical bound on *CSS_Optimize* is thus also $O(m^3)$.

B. Converting Skew to Retiming and PWA

1) *Overview:* The reduction of the magnitude of skew by retiming is illustrated in Fig. 7. When the skew is negative ($s_1 = -2$) as shown in Fig. 7(a), retiming from the input of g_2 to its output (assuming that the other input of g_2 also has a latch), called forward retiming, reduces the magnitude of the skew (from -2 to 0). Retiming from the output of g_1 to its input, called backward retiming, increases the magnitude of the skew, and therefore no help in reducing the magnitude of negative skew. When the skew is positive, as shown in Fig. 7(b), the converse is true and only backward retiming reduces the skew. Therefore, if we convert skew to retiming, the direction of retiming is mandated by the sign of the skew [16].

If we convert skew to retiming while adjusting the pulse-width of the latch (i.e., retiming and PWA), then both forward and backward retiming are possible, independent of the sign of the skew, which provides more flexibility in conversion. This is illustrated in Fig. 8. We are trying to reduce the magnitude of a negative skew $s_1 = -3$, as shown in Fig. 8(a). This can be achieved by forward retiming, using the method already illustrated in Fig. 7(a), which involves relocating latch 1 from the input of g_1 to its output. Note that this causes additional forward retiming of latches 2 and 3, since there is no latch at the other input to g_1. We then compute s_b, the new skew after retiming, which is equal to $\max(s_1 + d_{AZ}, s_a + d_{BZ})$, where d_{AZ} and d_{BZ} correspond to the pin-to-pin delay of g_1 (they are assumed to be the same for simplicity). If $|s_b| < |s_1|$, which is the case in the example, we may perform forward retiming; otherwise the latches remain in their original positions. If we perform backward retiming instead, and relocate latch 1 from the output of g_3 to its input, then the skew becomes more negative (from -3 to -4). However, if we decrease the pulse width at latch c, for instance by 5, then the rising edge of the pulse now arrives 5 time units later, while the arrival time of the falling edge remains unchanged, as shown in the figure. This is equivalent to increasing the skew of c by 5; so the new s_c is $-4 + 5 = 1$.

In Fig. 8(b), we are trying to reduce the magnitude of a positive skew $s_4 = 5$. We perform backward retiming of latch 4, which requires the further backward retiming shown in the figure, since g_4 has to have latches at all its fanout nodes. The new skew s_f is equal to $\min(s_4, s_d) - d_{AZ}$, and s_g can be computed similarly; we may perform backward retiming if both $|s_f|$ and $|s_g|$ are smaller than $|s_4|$, which is the case in the example. If we perform forward retiming instead, the skew becomes more positive (from 5 to 6); but the skew can nevertheless be reduced if we increase the pulse width as shown in the figure.

We try both forward and backward retiming (with a potential change of pulse width) on each latch which is a candidate for retiming. Note that retiming in both directions is not always possible. In Fig. 8(a), if the pulse at latch 1 is already of minimum width, it is useless to perform backward retiming because it only increases the magnitude of the skew; similarly, in Fig. 8(b), forward retiming is not performed if the pulse at latch 4 is at its widest. If both forward and backward retiming are possible, then we use whichever yields a new skew of smaller magnitude, unless the increase in the number of latches exceeds a given limit. If both retimings produce the same reduction in skew, then we use whichever yields fewer latches.

Algorithm *Convert_Skew* (ΔL_{max})
L1 **for** all latch i **do**
L2 **if** $s_i > 0$ **then**
L3 $w_i \leftarrow \text{argmin}_{W_j \in \{W_1,\ldots,W_n\}} |s_i - (W_j - W_1)|$
L4 $s_i \leftarrow s_i - (w_i - W_1)$
L5 **else** $w_i \leftarrow W_1$
L6 **repeat**
L7 Find $i \rightsquigarrow j$ that maximizes $\phi' = D_{ij} - (w_j - w_i)$
L8 **if** $(\phi' = \phi_{min}) \vee (s_i{=}0 \wedge s_j{=}0)$ **then**
L9 **return**
L10 **else** **if** $|s_i| > |s_j|$ **then** $\gamma \leftarrow i$ **else** $\gamma \leftarrow j$
L11 **if** $(s_\gamma < 0) \wedge (w_\gamma = W_1)$ **then**
L12 *Forward_Retime*(γ); $s_f \leftarrow$ *Update_Skew*(ϕ')
L13 $s_b \leftarrow \infty$; *Undo_Retime*
L14 **else if** $(s_\gamma > 0) \wedge (w_\gamma = W_n)$ **then**
L15 *Backward_Retime*(γ); $s_b \leftarrow$ *Update_Skew*(ϕ')
L16 $s_f \leftarrow \infty$; *Undo_Retime*
L17 **else**
L18 *Forward_Retime*(γ); $s_f \leftarrow$ *Update_Skew*(ϕ')
L19 *Undo_Retime*; *Backward_Retime*(γ)
L20 $s_b \leftarrow$ *Update_Skew*(ϕ'); *Undo_Retime*
L21 **if** $(\Delta L_f < \Delta L_{max}) \wedge (|s_f| < \{|s_\gamma|, |s_b|\})$ **then**
L22 *Forward_Retime*(γ)
L23 **else if** $(\Delta L_b < \Delta L_{max}) \wedge (|s_b| < \{|s_\gamma|, |s_f|\})$ **then**
L24 *Backward_Retime*(γ)
L25 **else if** $(\{\Delta L_f, \Delta L_b\} < \Delta L_{max}) \wedge (|s_f|{=}|s_b|{<}|s_\gamma|)$ **then**
L26 Execute retiming causing smaller increase of latches
L27 Drop γ from further consideration

Fig. 9. Overall algorithm to reduce the magnitude of skew by converting it to retiming and PWA where possible.

2) *Algorithm:* The overall algorithm *Convert_Skew*, which converts skew to retiming and PWA if possible, is shown in Fig. 9. Its input parameter $\Delta L_{\max}$ is the maximum number of extra latches that are allowed.

The initial pulse width is set (L1 to L5) to reduce the magnitude of skew, which is set by *CSS_Optimize*, as far as possible before we start converting the skew to retiming and PWA. Since the skew values assigned by *CSS_Optimize* are mostly positive,[2] latches with negative skew are simply assigned the minimum pulse width W_1 (L5). The magnitude of a positive skew can be reduced by assigning a pulse width larger than W_1. This is illustrated in Fig. 10. Since we regard the falling edge of pulse as the timing reference (see Fig. 2), while the skew is measured at the rising edge, assigning $W_j \neq W_1$ to w_i causes s_i to be reduced by $W_j - W_1$. Therefore, we choose $W_j \in \mathcal{W}$, as a value of w_i, to minimize $|s_i|$ (L3).

For each iteration of the skew conversion process (L6 to L27), we consider either a launching or capturing latch of a timing-critical path as a candidate. An alternative approach might be to select the latch with the skew of greatest magnitude, but priority has to be given to the latches involved in determining the clock period. Since skew will eventually

[2]In *CSS_Optimize*, only positive skews are assigned to latches; circuit inputs and outputs, which are not associated with latches, are also assigned positive skews, even though their skews are typically small. After initial skew assignment, the skews of circuit inputs and outputs are brought to zero, which makes the skew of some latches negative.

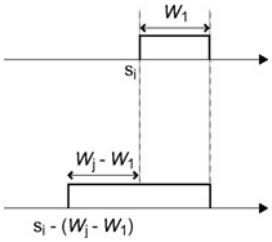

Fig. 10. Using $W_j \neq W_1$ causes s_i to be reduced by $W_j - W_1$.

be reduced to zero when we obtain the final clock period in Section V-C, the critical path must be determined on the basis of the delay of the combinational path subtracted by the time-borrowing [L7, see (5)], but without the clock skew. If that critical path already has the minimum clock period $\phi_{\min}$ (L8), which is set by *CSS_Optimize*, the algorithm simply terminates. If the skews at launching and capturing latches are both zero (L8), then the algorithm also terminates, since ϕ' is the final clock period. Otherwise we select the latch (γ) with larger amount of skew as the candidate for skew conversion (L10).

To reduce the magnitude of the skew at latch γ, we consider three different cases. If the skew is negative and γ has the minimum pulse width W_1 (L11), then forward retiming is the only possibility [see Fig. 8(a)]. The value of skew corresponding to forward retiming, s_f, is set accordingly; and the value for backward retiming, s_b, is set to infinity. As we have discussed in Section III-B1, a retiming move alters the load capacitance of some gates, and this has to be accounted in obtaining an accurate value for the new skew; this adjustment is made by the function *Update_Skew*, illustrated in Fig. 11. In the second case, the skew is positive and γ has its maximum pulse width W_n (L14). Only backward retiming is possible [see Fig. 8(b)]: s_b is set accordingly and s_f is set to infinity. Otherwise, which is the third case, we perform both forward and backward retiming, with PWA if possible, and we again perform *Update_Skew* to obtain s_f and s_b values for the respective retimings (L17 to L20).

If the aggregate number of extra latches after forward retiming, denoted by ΔL_f, does not exceed $\Delta L_{\max}$, and $|s_f|$ is smaller than both $|s_\gamma|$ (meaning that the magnitude of the skew has been reduced) and $|s_b|$, then forward retiming is performed (L21 to L22); backward retiming is performed if the condition is met (L23 to L24). If the magnitude of the skew is the same ($|s_f| = |s_b|$), then we perform the retiming that involves fewer extra latches (L25 to L26).

3) *Update_Skew to Account for Accurate Delay:* If the gate delay does not change during a retiming move, then the new skew value is updated by simply adding (or subtracting) the delay of the gate across which a latch has been relocated. Determining the actual change in delay, however, is more complicated, due to the change in load capacitance, as we have discussed in Section III-B1. This adjustment is performed by the function *Update_Skew* shown in Fig. 11.

This routine uses incremental clock skew scheduling to assign a new skew and pulse width to the latches that have been relocated or newly created (L1). The skews and pulse widths are first initialized (L2 to L3). We then update the path delay D_{ij} if either latches i or j were relocated during the

Function *Update_Skew*(ϕ')
L1 $\mathcal{L} \leftarrow$ a set of latches relocated due to a retiming
L2 **for** all latch $i \in \mathcal{L}$ **do**
L3 $s_i \leftarrow 0$; $w_i \leftarrow W_1$
L4 **for** all path $i \rightsquigarrow j$, where $i \in \mathcal{L} \vee j \in \mathcal{L}$ **do**
L5 Update D_{ij}
L6 $C(i,j) : s_j \geq s_i + w_i - w_j + (t_{dq} + D_{ij} - \phi_{min})$
L7 **while** $\exists_{i,j}, C(i,j)$ is not satisfied **do**
L8 **if** $j \in \mathcal{L}$ **then**
L9 $s_j \leftarrow s_i + w_i - w_j + (t_{dq} + D_{ij} - \phi_{min})$
L10 $w_j \leftarrow \text{argmin}_{W_k \in \{W_1,\ldots,W_n\}} |s_j - (W_k - W_1)|$
L11 $s_j \leftarrow s_j - (w_j - W_1)$
L12 **else**
L13 **if** $C(i,j)$ already uses ϕ' **then return** ∞
L14 **else** replace ϕ_{min} with ϕ' in $C(i,j)$
L15 **if** $s_j + w_j - W_1 > \phi_{min}$ **then return** ∞
L16 **return** $\max_{i \in \mathcal{L}} |s_i|$

Fig. 11. *Update_Skew* function.

retiming move (L5). The constraint $C(i, j)$ is set up for a clock period of $\phi_{\min}$ (L6), because we want to assign values of s_i, s_j, w_i, and w_j which satisfy $\phi_{\min}$ with the modified value of D_{ij}. The remainder of this function is similar to the function *CSS*, as we can see in Fig. 6. We take any unsatisfied constraint $C(i, j)$ (L7) and set s_j to the right-hand side of $C(i, j)$ if j is a relocated latch (L8 to L9); we then choose a W_k from $\mathcal{W}$ that minimizes $|s_j|$ as a value of w_j, and then set s_j accordingly (L10 to L11), which is similar to the initial skew adjustment of Fig. 9.

If latch j has not been relocated (L12), then we cannot assign a new skew to j that satisfies $\phi_{\min}$. In this case, we see whether ϕ', which is the delay of the timing-critical path during a particular skew conversion (L7 in Fig. 9), can be satisfied instead of $\phi_{\min}$ (L14). If ϕ' is also not satisfied, then the function fails (L13), and the corresponding retiming move is canceled. Otherwise, the skew with the largest magnitude becomes the new skew after retiming.

C. Determining the Final Clock Period

Convert_Skew tries to reduce the magnitude of the skew in the latches that lie on timing-critical paths. The skew of many latches that are off the timing-critical paths are unaffected by *Convert_Skew* (except for L1 to L5, which initially reduce all positive skews). These unchanged skews can safely be forced to zero without affecting the clock period.

If we force the skews of latches on timing-critical paths to zero, however, the clock period may increase. This is because the time-borrowing of $s_j - s_i$ for the combinational block between i and j is no longer available. Therefore the clock period is now solely determined by the maximum combinational delay while we take account of the time-borrowing caused by differences in pulse widths as follows:

$$\phi_{0-\text{skew}} = \max_{\forall_{i \rightsquigarrow j}} \left[t_{dq} + D_{ij} - \left(w_j - w_i \right) \right]. \tag{25}$$

Once $\phi_{0-\text{skew}}$ has been determined, a circuit is checked for hold-time violations. Any violations are fixed by inserting delay buffers [29] by iteratively selecting paths that violate hold-time constraint, and then inserting delay buffers in edges until the violation is fixed or the insertions cause setup-time constraints to be violated in other paths that share the same edge. Note that buffer insertion may increase the clock period due to the finite number of buffer sizes available in a library, and the mismatch between the rise and fall delays of buffers.

TABLE I
BENCHMARK CIRCUITS

Name	# Gates	# Latches	ϕ_{ini} (ps)	$\phi_{\min}$ (ps)
s1423	727	74	1838	1670
s9234	1546	135	820	718
s15850	4164	569	1159	940
s35932	11 437	1728	732	513
b04	804	66	787	433
b07	491	49	834	515
b08	225	21	656	328
b11	681	30	822	655
can_btl	452	30	645	588
i2c	2396	129	977	759
ps2	2830	185	845	646
t48	1819	79	721	606
t400	3186	176	1078	781
usbc	2573	402	684	506

VI. EXPERIMENTAL RESULTS

We carried out experiments on a set of sequential circuits taken from the ISCAS and ITC benchmarks, together with circuits extracted from several open cores [30], including a CAN-protocol controller (can_btl), a communication controller (i2c), a keyboard interface unit (ps2), two microprocessor units (t48 and t400), and a USB controller (usbc). The test circuits are listed in Table I, in which the second and third columns are the numbers of combinational gates and latches after synthesizing each circuit with a commercial logic synthesis tool [31]. Industrial 45 nm technology was used for the synthesis and for all the other experimental procedures. After performing STA, the initial clock period ϕ_{ini}, shown in the fourth column of Table I, was set to the maximum latch-to-latch delay. We assumed that all the latches have the same pulse width. The arrival times of all the circuit inputs were set to zero, and the required arrival times of all the circuit outputs were set to the clock period. The minimum clock period $\phi_{\min}$ returned by *CSS_Optimize* of Fig. 6 is shown in the last column of Table I, and represents the minimum clock period that can be achieved.

A set of five pulse generators was designed, with pulse widths of 130, 190, 250, 310, and 370 ps. The design considerations relating to these pulse widths will be discussed in Section VI-B. Various methods of implementing a pulse generator have been proposed [5], [9], [32]; we used the one shown in Fig. 12 because of its low power consumption. The pulse width is modulated by adjusting the delay of the inverter, as shown in the figure; clock gating is possible through EN signal that drives an nMOS transistor N1. The pulse generator was designed to drive up to 10 latches with a slew constraint of 60 ps, which was found to be the upper bound at which

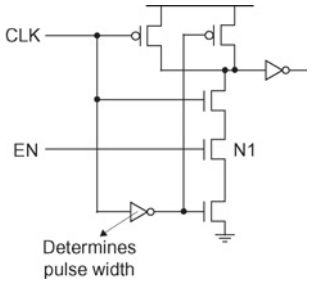

Fig. 12. Pulse generator [9].

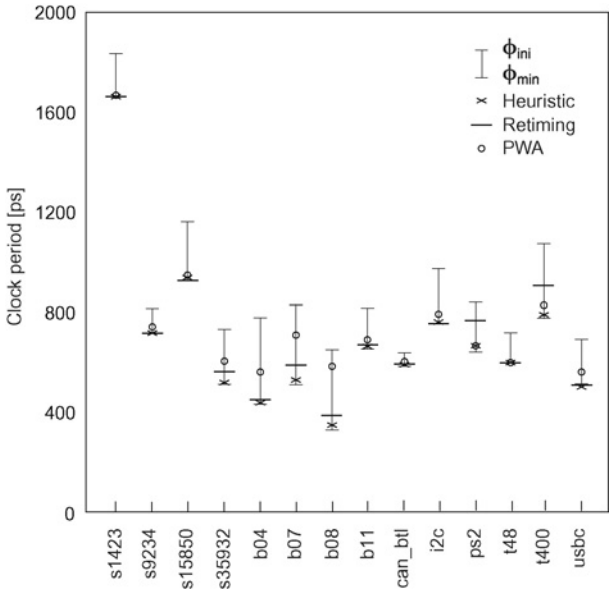

Fig. 13. Comparing the clock period from heuristic algorithm (with $\Delta L_{\max}$ set to ∞) with that from standard retiming and PWA; they are also compared to the initial clock period ϕ_{ini} and the minimum clock period $\phi_{\min}$.

the safe latching of data was ensured. It occupies about 1.2 to 1.6× the area of latch, depending on the width of pulse that it generates.

A. Effectiveness of Heuristic Algorithm in Determining the Clock Period

The heuristic algorithm described in Section V was implemented in SIS [33]. Fig. 13 compares the clock period that it returns (without setting $\Delta L_{\max}$, i.e., $\Delta L_{\max} = \infty$) with a clock period obtained by a standard retiming, and another by PWA (using only time-borrowing); also shown are ϕ_{ini} and $\phi_{\min}$ from Table I, which together represent the extent of the potential for optimization.

The heuristic algorithm achieved the minimum (or close to minimum) clock period for all the circuits that we tested; at the cost of an average 16% increase in the number of latches. Standard retiming produced clock periods that are far from minimum for many of the circuits (namely, s35932, b07, b08, ps2, and t400); furthermore, the number of latches increased by 29% on average. This clearly shows the benefit of combining retiming with PWA. The clock period from PWA alone is larger than that from retiming, except for circuits ps2 and t400, although PWA does not increase the number of

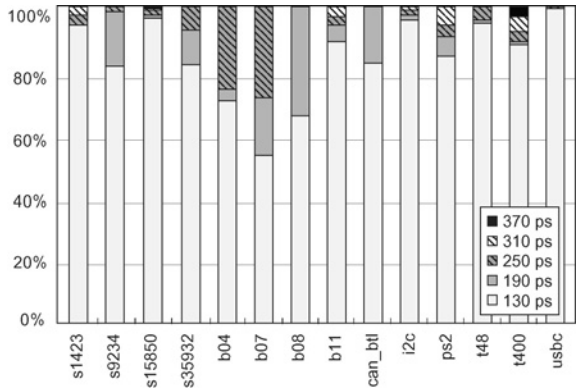

Fig. 14. Pulse generators required by the heuristic algorithm.

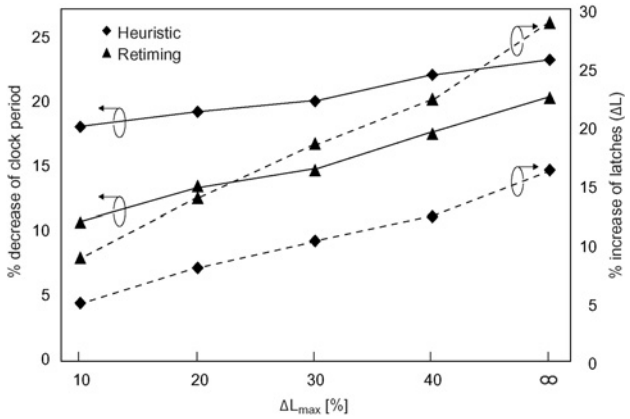

Fig. 15. Heuristic algorithm and retiming with varying $\Delta L_{\max}$: % change of clock period and % increase of latches, compared to the initial circuit, averaged over the circuits in Table I.

latches; but it does require extra buffers to fix more hold-time violations. This demonstrates the limited capability of PWA alone to reduce the clock period. Compared to ϕ_{ini}, the clock period was reduced by 23%, 20%, and 15% on average by using the heuristic algorithm, retiming, and PWA, respectively. Fig. 14 shows the proportion of each type of pulse generator required by the heuristic algorithm. Many circuits need only two or three pulse widths, which indicates that a few different types of pulse generator, together with retiming, can yield a shorter clock period than either retiming or PWA alone.

We tested the effect of varying $\Delta L_{\max}$, the maximum allowable number of extra latches, on the heuristic algorithm, with the result shown in Fig. 15. We also plotted results from our heuristic algorithm without PWA, which represents standard method of retiming (except that the number of extra latches is forced to be less than $\Delta L_{\max}$). The heuristic requires fewer extra latches than retiming. Both forward and backward retiming (combined with PWA where possible), as shown in Fig. 8, are tried by our algorithm. The direction of retiming which has more effect on the magnitude of the skew and requires fewer extra latches is selected; whereas in standard retiming, the direction of a retiming move is simply determined by the sign of the skew. The heuristic algorithm achieves a greater reduction of the clock period than retiming for the same $\Delta L_{\max}$, especially as $\Delta L_{\max}$ becomes smaller.

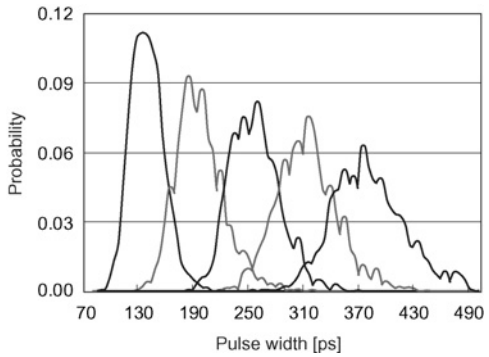

Fig. 16. Variation of pulse widths obtained by Monte Carlo simulation.

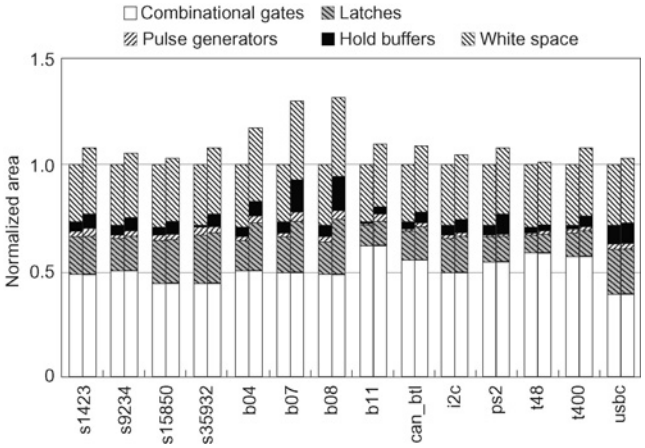

Fig. 17. Areas of circuits initially (left-hand bars) and after performing the heuristic algorithm (right-hand bars).

1) *Proactive Approach to Hold-Time Violations:* The heuristic algorithm in Section V considers only setup-time constraints; hold-time constraints are checked and violations are fixed after the algorithm, not as an integral part. The alternative approach is also possible by including hold-time constraints in the function *CSS* shown in Fig. 6, and by forcing minimum pulse width to latches that are likely to violate hold-time constraints, which we also implemented; the remaining hold-time violations, which now become smaller in numbers, are again fixed by inserting delay buffers.

For the designs obtained by the modified algorithm, the area of extra buffers is reduced by 14.4% on average (see Fig. 17 for the proportion buffer area in total circuit area) while the clock period increases by 5.5% compared to the heuristic algorithm of Section V. Clearly, the choice of algorithm relies on the trade-off between buffer area and clock period.

2) *Variation of Pulse Width:* Integrated circuits are always susceptible to process, voltage, and temperature (PVT) variations. A pulse generator is no exception. Fig. 16 shows the variation of pulse width of five pulse generators obtained by Monte Carlo simulation in 45 nm technology; V_{dd} variation was applied with 0.11 V as 3σ and 1.1 V as a mean and temperature was varied with 50 °C and 75 °C as 3σ and a mean, respectively.

Simply assuming $\pm 3\sigma$ of pulse width does not represent worst or best case, because the difference of pulse width of launching and capturing latches, not pulse width itself, determines the amount of time-borrowing. This can be addressed in two different ways: including extra timing margin (in clock period) to absorb any risk of timing violations due to pulse width variation, or directly taking account of variations during design stage.

We took each circuit generated by the heuristic algorithm; a variation of pulse width shown in Fig. 16 was applied to each corresponding pulse generator and the probability of circuit satisfying its timing constraints (called timing yield) was measured [34]; clock period was then increased until timing yield exceeds 90%. The increment of clock period, which represents timing margin, ranged from 9 to 23 ps, which roughly correspond to 0.25 to 0.60 FO4 delay, respectively. This is not significant overhead, given that a typical timing margin to accommodate PVT variations throughout a whole circuit is 7 or 8 FO4 delay [35].

There has been an effort for PWA under variation of pulse width [34]. The PWA under variation of all circuit elements, not under pulse width variation alone, is seemingly a difficult problem and may be pursued as a future research.

B. Effects on Circuit Area

Fig. 17 shows the area of each circuit after performing the heuristic algorithm (with $\Delta L_{\max}$ set to ∞), as a proportion of its initial area (the area of circuit that uses only minimum pulse width W_1 without any retiming). The height of each bar corresponds to the layout area after performing placement [11], which ensures that the connection between each latch and pulse generator is short enough for safe delivery of pulse. The density constraint was set to 70%; the white space, layout area that is not occupied by any cells, is also specified in Fig. 17.

In both the original and refined circuits, buffers were inserted to fix any hold-time violations, using the procedure explained in Section V-C. Clearly, more buffers are required after performing the heuristic algorithm due to the use of wider pulses. This is especially true for circuits b04, b07, and b08, which also exhibit the largest increase in area, due to their more extensive use of wider pulses (see Fig. 14). Three circuits, b04, b07, and b08, also require many more latches. This is due to the large difference between ϕ_{ini} and $\phi_{\min}$, which can be seen in Fig. 13; the large reduction in clock period that was achieved required heavy use of retiming even with time-borrowing by PWA. The heuristic algorithm increased the overall area of each circuit by 10.1% on average, compared to the original circuits: this is the cost of the reduced clock period.

Fig. 17 suggests that using wider pulses to exploit time-borrowing comes at a cost in increased hold-time violations, which requires more extra buffers to fix. Conversely, if only a restricted list of pulse widths is available, more latches will be needed since we must rely more on retiming. Fig. 18 shows the area (before placement) of two example circuits after performing the heuristic algorithm, for varying numbers of available pulse width. This supports our conjecture that the requirement for latches tends to decrease and the requirement for buffers tends to increase as more pulse widths become

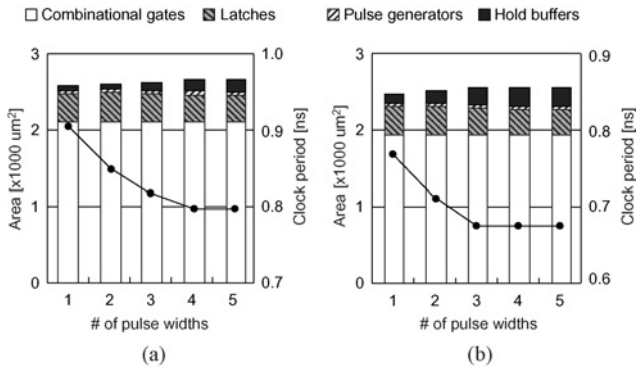

Fig. 18. Areas of circuits after the heuristic algorithm (with $\Delta L_{\max} = \infty$) for different numbers of available pulse widths (e.g., 3 on the horizontal axis means using 130, 190, and 250 ps) together with the clock-periods that were achieved. (a) t400. (b) ps2.

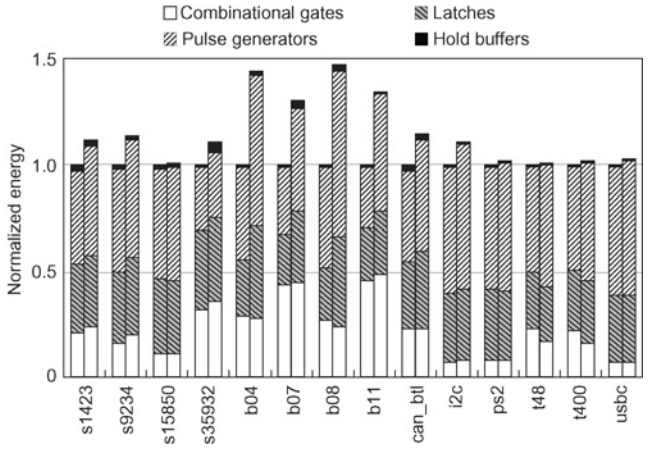

Fig. 19. Energy dissipation of circuits initially (left-hand bars) and after performing the heuristic algorithm (right-hand bars).

TABLE II
ILP VERSUS HEURISTIC: CLOCK PERIOD (ϕ), EXTRA LATCHES (ΔL), AND COMPUTATION TIME

Benchmark				ILP			Heuristic	
Name	# Gates	# Latches	ϕ_{ini} (ps)	ϕ (ps)	ΔL	Time (s)	ϕ (ps)	ΔL
s349	196	15	545	422	12	2741	453	6
s382	179	21	385	332	16	5768	343	0
s400	296	21	374	324	17	7807	345	4
s510	308	6	428	405	1	2209	405	1
b01	59	5	353	320	13	122	332	0
b02	34	4	307	221	2	21	243	0
b06	69	8	336	286	5	117	288	3
ac97	79	10	382	299	3	166	322	0

available. The clock frequencies that are achieved are also shown in the figure. Note that ps2 uses only four pulse widths even when it is allowed all five (see Fig. 14), which explains why the results shown in Fig. 18 are identical for four and five available pulse widths.

C. Comparison of ILP and Heuristic Algorithm

The ILP approach described in Section IV was formulated in SIS, and then submitted to an ILP solver [36]. A cost function (23) that includes both the clock period and the number of extra latches was used for the ILP. Results from ILP and heuristic algorithm (with $\Delta L_{\max} = \infty$), in which only the setup-time constraints are formulated for the ILP for fair comparison, are compared in Table II for several very small example circuits.

Clearly, the ILP approach is limited by its computation time, even for these small circuits; but this never becomes an issue for the heuristic algorithm in any of the example circuits, which took less than a second for all examples in Table II, including those from Table I.

ILP achieves a smaller clock period at all examples of Table II, as we must expect. Note that the order of selection of the candidate latch γ in our heuristic algorithm (see L10 of Fig. 9) does not affect the final clock-period if $\Delta L_{\max} = \infty$, because only its skew magnitude is reduced without affecting any other latches. However, the initial skew that is assigned by *CSS_Optimize*, which is subsequently adjusted by *Convert_Skew* (see L1 to L5 of Fig. 9), affects the extent to which the clock period can be reduced. This can be understood from a simple example: assume that $i \quad j$ is a timing-critical path, $s_i = 0$ and $s_j = 2$, and a gate with a delay of 2 is a fan-in of i and a gate with a delay of 4 is a fan-in of j; s_j cannot be reduced in this case; but if both skews are increased by 2 ($s_i = 2$ and $s_j = 4$), then both of them can be made 0 by performing backward retiming. This suggests that there is a room for improvement in the way that the heuristic algorithm adjusts the initial clock skew, so that the skews of latches involved in the timing-critical paths are brought as close to zero as possible; this is left for future work.

The heuristic algorithm requires fewer extra latches. This can be understood by looking at the number of retiming moves that the two algorithms attempt: ILP tries 14.5 times more retiming moves on average. More retiming moves is likely to increase the requirement for extra latches, because it is then more likely that a move causing a large increase in the number of latches will be selected, even though it brings in only a little reduction in the final clock period.

D. Energy Dissipation

Fig. 19 compares the average energy dissipation (during one clock cycle) of each circuit before and after performing the heuristic algorithm (with $\Delta L_{\max} = \infty$). The energy dissipation was obtained by simulating each circuit with a fast transistor-level simulator [37] while applying 100 randomly generated vectors to the inputs. Both the dynamic and static components of energy dissipation are included in the results.

Pulse generators and latches dissipate most energy since their switching activity is 1.0, while that of combinational gates is between 0.1 and 0.2. The situation may become different, however, in data-path-dominated circuits or in clock-gated circuits, where the proportion of energy dissipation from combinational gates will increase. The energy dissipation of pulse generators is substantial considering the small proportion of area that they occupy (see Fig. 17). The design of the pulse generator that we used consumes about 16 μW, a latch consumes 0.8–4.6 μW (depending on whether its data

input switches), and a 2-input NAND gate consumes 1.3 μW. Compared to designs based on flip-flops, however, pulsed-latch circuits are economical: 10 D flip-flops and 1 leaf-stage buffer, together, consume about 114 μW; 1 pulse generator and 10 latches that it drives consume about 62 μW, which represents a 45% saving in power consumption.

The circuit produced by the heuristic algorithm used an average of 16% more energy. This is due to the increased number of latches resulting from retiming, and the concomitant increase in the number of pulse generators.

VII. Conclusion

We proposed the use of more than one pulse width as a way to improve the performance of pulsed-latch circuits. This has been combined with retiming to achieve even better performance with fewer extra latches than standard retiming. This approach leads to a combined retiming and PWA problem, which we have formulated and solved using ILP for very small circuits, and a fast heuristic algorithm for circuits of realistic size. The heuristic algorithm has been assessed, using a number of circuits implemented in 45 nm technology, in terms of clock period, circuit area, and energy dissipation.

The main drawback in using multiple pulse widths is the increasing risk of hold-time violations. We fixed these violations by introducing extra buffers, which can be a substantial overhead in some circuits. Alternative approaches would be to use flip-flops, where hold-time violations are very likely to happen, or to re-synthesize combinational circuits to reduce the number of hold-time violations followed by inserting buffers to fix any remaining violations; these possibilities are left for future investigation. The pulse generators required by our approach consume a significant amount of power, but not enough to outweigh the advantage of pulse-latch circuits in terms of power consumption. Nevertheless, a new type of pulse generator that consumes less power would benefit the applicability of pulsed-latch circuits to low-power applications.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions, and Professor Taewhan Kim of Seoul National University, Seoul, Korea, for helpful discussion on the complexity of retiming and PWA problem.

References

[1] S. Lee, S. Paik, and Y. Shin, "Retiming and time borrowing: Optimizing high-performance pulsed-latch-based circuits," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2009, pp. 375–380.

[2] H. Lee, S. Paik, and Y. Shin, "Pulse width allocation with clock skew scheduling for optimizing pulsed latch-based sequential circuits," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2008, pp. 224–229.

[3] S. Shibatani and A. Li, "Pulse-latch approach reduces dynamic power," *EE Times*, Jul. 2006.

[4] H. Partovi, R. Burd, U. Salim, F. Weber, L. DiGregorio, and D. Draper, "Flow-through latch and edge-triggered flip-flop hybrid elements," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 1996, pp. 138–139.

[5] S. Kozu, M. Daito, Y. Sugiyama, H. Suzuki, H. Morita, M. Nomura, K. Nadehara, S. Ishibuchi, M. Tokuda, Y. Inoue, T. Nakayama, H. Harigai, and Y. Yano, "A 100 MHz 0.4 W RISC processor with 200 MHz multiply-adder, using pulse-register technique," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 1996, pp. 140–141.

[6] A. Scherer, M. Golden, N. Juffa, S. Meler, S. Oberman, H. Partovi, and F. Weber, "An out-of-order three-way superscalar multimedia floating-point unit," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 1999, pp. 94–95.

[7] L. Clark, E. Hoffman, J. Miller, M. Biyani, L. Liao, S. Strazdus, M. Morrow, K. Velarde, and M. Yarch, "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1599–1608, Nov. 2001.

[8] N. Kurd, J. Barkarullah, R. Dizon, T. Fletcher, and P. Madland, "A multigigahertz clocking scheme for the Pentium 4 microprocessor," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1647–1653, Nov. 2001.

[9] S. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski, "The implementation of the Itanium 2 microprocessor," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1448–1460, Nov. 2002.

[10] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama, "A 1.3-GHz fifth-generation SPARC64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 38, no. 11, pp. 1896–1905, Nov. 2003.

[11] Y. Chuang, S. Kim, Y. Shin, and Y. Chang, "Pulsed-latch-aware placement for timing-integrity optimization," in *Proc. Design Autom. Conf.*, Jun. 2010, pp. 280–285.

[12] J. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[13] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. CalTech Conf. VLSI*, Mar. 1983, pp. 23–36.

[14] K. Carrig, "Chip clocking effect on performance for IBM's SA-27E ASIC technology," *IBM Micronews*, vol. 6, no. 3, pp. 12–16, 2000.

[15] S. Held, B. Korte, J. Maßberg, M. Ringe, and J. Vygen, "Clock scheduling and clocktree construction for high performance ASICs," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2003, pp. 232–239.

[16] S. Sapatnekar and R. Deokar, "Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits," *IEEE Trans. Comput.-Aided Design*, vol. 15, no. 10, pp. 1237–1248, Oct. 1996.

[17] C. Lin and H. Zhou, "An efficient retiming algorithm under setup and hold constraints," in *Proc. Design Autom. Conf.*, Jul. 2006, pp. 945–950.

[18] H. Zhou, "A new efficient retiming algorithm derived by formal manipulation," *ACM Trans. Design Autom. Electron. Syst.*, vol. 13, no. 1, p. 7, 2008.

[19] H.-G. Martin, "Retiming by combination of relocation and clock delay adjustment," in *Proc. Eur. Design Autom. Conf.*, Sep. 1993, pp. 384–389.

[20] X. Liu, M. C. Papaefthymiou, and E. G. Friedman, "Retiming and clock scheduling for digital circuit optimization," *IEEE Trans. Comput.-Aided Design*, vol. 21, no. 2, pp. 184–203, Feb. 2002.

[21] L. Chao and E. Sha, "Retiming and clock skew for synchronous systems," in *Proc. Int. Symp. Circuits Syst.*, 1994, pp. 283–286.

[22] A. Ishii and M. Papaefthymiou, "Efficient pipelining of level-clocked circuits with min-max propagation delays," in *Proc. ACM Int. Workshop Timing Issues Specification Synthesis Digital Syst.*, Nov. 1995, pp. 39–51.

[23] A. Ishii, C. Leiserson, and M. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry," *J. ACM*, vol. 44, no. 1, pp. 148–199, Jan. 1997.

[24] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, nos. 1–6, pp. 5–35, Jun. 1991.

[25] K. Lalgudi and M. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling," in *Proc. Design Autom. Conf.*, Jun. 1995, pp. 304–309.

[26] T. Soyata, E. Friedman, and J. Mulligan, Jr., "Incorporating interconnect, register, and clock distribution delays into the retiming process," *IEEE Trans. Comput.-Aided Design*, vol. 16, no. 1, pp. 105–120, Jan. 1997.

[27] Y. Kohira and A. Takahashi, "Clock period minimization method of semi-synchronous circuits by delay insertion," in *Proc. Asia-Pacific Conf. Circuits Syst.*, Dec. 2004, pp. 533–536.

[28] C. Lin and H. Zhou, "Clock skew scheduling with delay padding for prescribed skew domains," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2007, pp. 541–546.

[29] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 1993, pp. 156–161.

[30] OpenCores [Online]. Available: http://www.opencores.org
[31] *Design Compiler User Guide*, Synopsys, Mountain View, CA, Sep. 2008.
[32] A. Venkatraman, R. Garg, and S. P. Khatri, "A robust, fast pulsed flip-flop design," in *Proc. Great Lakes Symp. VLSI*, May 2008, pp. 119–122.
[33] E. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Univ. California, Berkeley, Tech. Rep. UCB/ERL M92/41, May 1992.
[34] S. Paik, L. Yu, and Y. Shin, "Statistical time borrowing for pulsed-latch circuit designs," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2010, pp. 675–680.
[35] T. Baumann, D. Schmitt-Landsiedel, and C. Pacha, "Architectural assessment of design techniques to improve speed and robustness in embedded microprocessors," in *Proc. Design Autom. Conf.*, Jul. 2009, pp. 947–950.
[36] Zuse Institute Berlin. *Solving Constraint Integer Programs (SCIP)* [Online]. Available: http://scip.zib.de
[37] *NanoSim User Guide*, Synopsys, Mountain View, CA, Sep. 2008.

Seungwhun Paik (S'07) received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2006. He is currently working toward the Ph.D. degree from the Department of Electrical Engineering, KAIST.

His current research interests include computer-aided design for pulsed-latch application-specific integrated circuit (ASIC) design, low-power design, high-level synthesis, and structured ASIC.

Seonggwan Lee received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 2008 and 2010, respectively.

He is currently with LG Electronics, Seoul, Korea.

Youngsoo Shin (M'00–SM'05) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Seoul, Korea.

From 2000 to 2001, he was a Research Associate with the University of Tokyo, Tokyo, Japan, and from 2001 to 2004 was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He joined the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2004, where he is currently an Associate Professor. His current research interests include areas of computer-aided design with emphasis on low-power design and design tools, high-level synthesis, sequential synthesis, and structured ASIC.

Dr. Shin received several awards, including the Best Paper Award at the 2005 International Symposium on Quality Electronic Design, the 2002 IP Excellence Award from Japan, and the Excellence in Teaching Award from KAIST in 2008. He has been a member of the technical program committees and organizing committees of many technical conferences, including DAC, ICCAD, ISLPED, ASP-DAC, CASES, ISVLSI, and ISCAS. He is a member of the ACM SIGDA Low Power Technical Committee and is serving as an Associate Editor of ACM TODAES.