Statistical Time Borrowing for Pulsed-Latch Circuit Designs

Seungwhun Paik, Lee-eun Yu, and Youngsoo Shin
Department of Electrical Engineering, KAIST
Daejeon 305-701, Korea

***Abstract*— Pulsed-latch inherits the advantage of latch in less sequencing overhead while taking the advantage of flip-flop in its convenience during timing analysis. Even though this advantage comes from the fact that pulsed-latch uses a short pulse, it is still capable of a small amount of time borrowing. A problem of allocating pulse width (out of a few predefined widths), where each width is modeled by a random variable, is formulated for minimizing the clock period of pulsed-latch circuits; this is equivalent to assigning a random variable that represents the amount of time borrowed by the combinational block between each latch pair. A statistical approach is important in this problem because assuming +3σ of all pulse widths does not represent the worst case. An allocation algorithm called SPWA as well as an algorithm to compute timing yield is proposed. In experiments with 45-nm technology, compared to the case of no time borrowing, the clock period was reduced by 12.2% and 11.7% on average when the yield constraint Y_c is 0.85 and 0.95, respectively; this is compared to the deterministic counterpart called DPWA, which reduced the clock period by 7.6% and 7.3%. More importantly, DPWA failed to satisfy the yield constraints in four (out of eleven) circuits while the yield constraints were always satisfied in SPWA.**

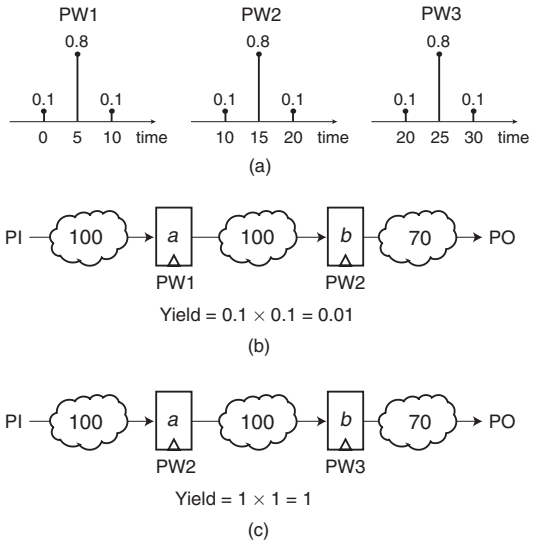

Fig. 1. (a) Pulse width distributions, (b) the allocation of pulses with a lower yield, and (c) the alternative allocation with a higher yield.

I. Introduction

A pulsed-latch is a latch driven by a short pulse. Since the length of time when an input data can be captured is very short, it can be approximated by an edge-triggered flip-flop. Therefore, any combinational block between two pulsed-latches can be assumed to have a clock cycle to compute, which simplifies timing analysis as in flip-flop circuits. Moreover, pulsed-latch circuits retain the advantage of latches in less sequencing overhead, less clock load, and smaller area. These come at a cost of extra pulse generators, which receive a normal clock, generate a clock pulse, and deliver it to local latches that are physically close. The cost, however, is typically small because a number of latches share a single pulse generator. Pulsed-latch has been widely used in custom designs, in particular in microprocessors [1], [2], but their application to ASIC designs has been limited due to lack of tool support [3].

Even though the advantage of pulsed-latch comes from a short pulse, as long as the pulse is wider than its setup time, it is still capable of a small amount of time borrowing [4, p.398], which can be used to tolerate clock skew or reduce the clock period for higher performance. More design freedom is offered if pulse generators of a few different pulse widths are available; this has been combined with clock skew scheduling to reduce the clock period of pulsed-latch circuits [5].

The pulse width, however, is susceptible to process variation as within-die (WID) variation takes increasing proportion of total variations, e.g. 35% in 130-nm but 60% in 70-nm technology [6]. In our experiment with 45-nm technology, a pulse generator with mean pulse width of 130 ps exhibits 20 ps of +3σ. In process corner-based design, we usually assume a +3σ delay as a worst case under the presence of WID variations. This, however, is not the case in pulse width because the amount of time borrowed by a combinational block between two latches is determined by the difference of pulse widths they use, not by the pulse widths themselves. For example, assume that we have three pulses in Fig. 1(a). We want to determine a clock period of a circuit shown in Fig. 1(b) and assign pulses to latches a and b. Let the arrival time of primary input (PI) be 0 and the required arrival time at primary output (PO) be the same as the clock period. If we consider +3σ of pulses (PW1 = 10, PW2 = 20, and PW3 = 30), the clock period of 90 can be obtained by using PW1 for a and PW2 for b, provided that there is no sequencing overhead. However, it can be readily verified that the clock period of 90 is feasible only when PW1 is 10 and PW2 is 20, implying the yield of 0.01. Another assignment where we use PW2 for a and PW3 for b shown in Fig. 1(c) also guarantees a clock period of 90 at +3σ point; the yield in this assignment, however, can be shown to be 1.0. Considering pulses at +3σ point alone does not give any preference to Fig. 1(c).

We formulate a problem of allocating pulse width (out of a few predefined widths), where each width is modeled by a random variable rather than by a +3σ value, with the objective

978-1-4244-5767-0/10/$26.00 2010 IEEE

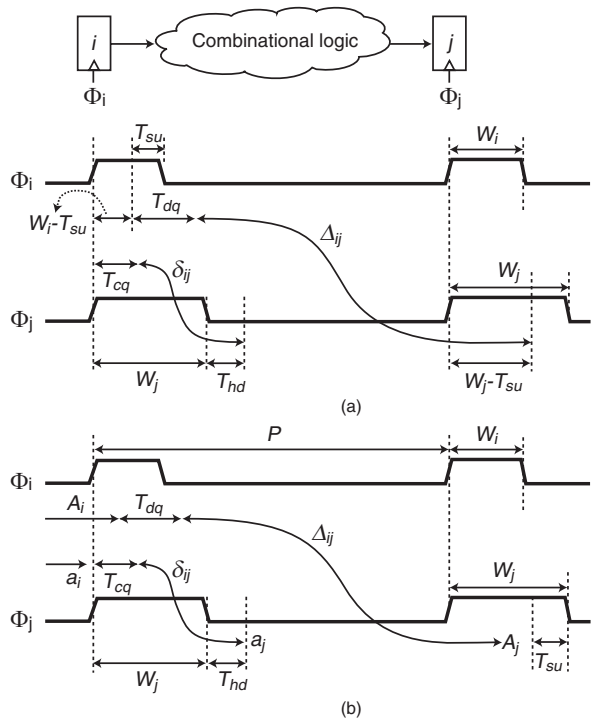

Fig. 2. (a) A pulsed-latch is approximated as a flip-flop and (b) it follows a latch timing model.

of minimizing the clock period of pulsed-latch circuits (Section III-A). The key ingredient in this approach is to estimate the timing yield of a circuit with a particular configuration of pulse widths, which we approach by introducing two constraint graphs (Section III-B). The allocation algorithm is addressed in Section IV-A, which is refined by considering the correlation of pulses that are driven by the same pulse generator in Section IV-B. The deterministic version of the algorithm is also developed (Section II-A) as a reference of comparison, which is used in the experiments (Section V) based on commercial 45-nm technology.

II. Preliminaries

A. Deterministic Pulse Width Allocation

Given a set of pulse widths $\mathcal{W}$, offered by a variety of pulse generators that produce different pulse widths, a deterministic pulse width allocation is to assign a pulse width $W_i \in \mathcal{W}$ to each pulsed-latch i such that the clock period P is minimized. This is subject to setup and hold time constraints. Under the approximation of a pulsed-latch as a flip-flop, the problem of pulse width allocation and clock skew scheduling can be solved by iterative relaxation under setup time constraint [5]. The approach can be readily extended to both setup and hold time constraints when we assign a pulse width alone. In Fig. 2(a), each combinational block between latches i and j, denoted by $i \rightsquigarrow j$, is assumed to use a period of time from $W_i - T_{su}$ to $P + W_j - T_{su}$, i.e. a length of time $P + W_j - W_i$, where $W_j - W_i$ corresponds to the time borrowed by $i \rightsquigarrow j$.

Algorithm DPWA (P)

L1 Sort a list of pulse widths $\mathcal{W}$ in order of increasing width
L2 $W_i \leftarrow \min_{w \in \mathcal{W}} w$
L3 $S(i,j)$: $W_j \geq W_i + \left(T_{dq} + \Delta_{ij} - P\right), \forall_{i \rightsquigarrow j}$
L4 $H(i,j)$: $W_j \leq T_{cq} + \delta_{ij} - T_{hd}, \forall_{i \rightsquigarrow j}$
L5 **while** $\exists_{i,j}, S(i,j)$ is not satisfied **do**
L6 $r_h \leftarrow W_i + \left(T_{dq} + \Delta_{ij} - P\right)$
L7 Select next wider W_j from $\mathcal{W}$ until $W_j \geq r_h$
L8 **if** such $W_j \in \mathcal{W}$ does not exist **then return** fail
L9 **if** $\exists_{i,j}, H(i,j)$ is not satisfied **then return** fail
L10 **else return** success

Fig. 3. Pseudo-code of DPWA algorithm.

The setup time constraint, therefore, is

$$T_{dq} + \Delta_{ij} \leq P + W_j - W_i, \tag{1}$$

where T_{dq} is the data-to-Q delay and Δ_{ij} is the maximum delay of $i \rightsquigarrow j$. For a hold time constraint, data is assumed to depart at the rising edge of clock, thus

$$T_{cq} + \delta_{ij} \geq W_j + T_{hd}, \tag{2}$$

where T_{cq} is the clock-to-Q delay, δ_{ij} is the minimum delay of $i \rightsquigarrow j$, and T_{hd} is the hold time.

To minimize the clock period P, we use a binary search. We take a median clock period P_m in the range (0, P_{crit}), where P_{crit} is the critical path delay when all latches have the same pulse width and represents the maximum clock period P can take. We then call DPWA shown in Fig. 3 to check whether P_m is feasible, i.e. whether there exists an assignment of pulse width such that (1) and (2) are satisfied for all combinational blocks. If P_m is feasible, we repeat the process with new range (0, P_m); otherwise the process continues with (P_m, P_{crit}).

The procedure DPWA shown in Fig. 3 is based on the Bellman-Ford algorithm, which is proven to be optimal [7] in a sense that P is feasible if and only if DPWA returns in success. The algorithm iteratively checks each unsatisfied setup time constraint $S(i,j)$ (L5); it then fixes the right-hand side of $S(i,j)$ (L6) and increases the left-hand side, which is W_j, until $S(i,j)$ becomes satisfied (L7). If such W_j does not exist (L8) or a hold time constraint $H(i,j)$ is violated (L9), the algorithm terminates in fail; otherwise it continues with the next unsatisfied setup time constraint.

B. Latch Timing Constraints

Timing constraints (1) and (2) are sufficient but not necessary conditions for pulsed-latch circuits to work. This allows a simple iterative algorithm shown in Fig. 3 to be developed to detect whether a feasible time borrowing exists for a given clock period. This approximation is based on the motivation of pulsed-latch, i.e. it can be treated like a flip-flop. However, in the statistical time borrowing, where each W_i is modeled by a random variable, the approximation model in (1) and (2) makes the solution more difficult to be found rather than to make it easier, which will be made clear in the following sections. Hence, we resort to the latch timing formulation [8],

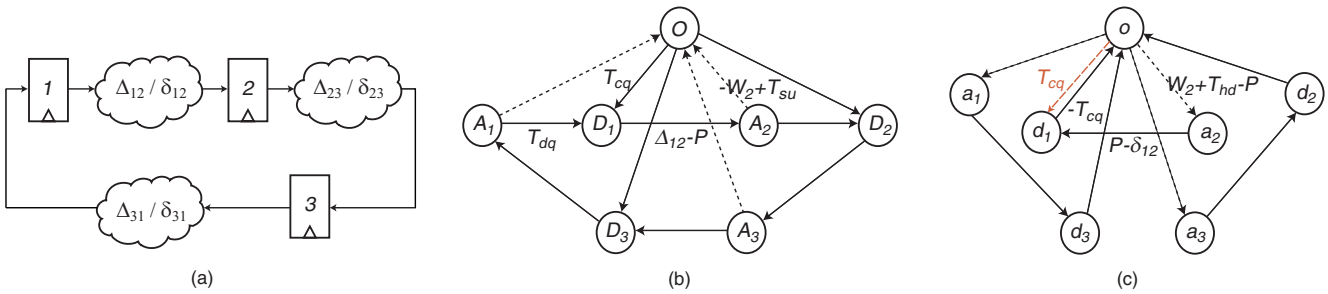

Fig. 4. (a) An example sequential circuit, (b) its setup time constraint graph (G_S), and (c) hold time constraint graph (G_H).

which first defines the latest (earliest) data arrival time A_j (a_j) and the latest (earliest) data departure time D_j (d_j) at latch j,

$$A_j = \max_{\forall_{i \rightsquigarrow j}} (D_i + \Delta_{ij} - P), \quad (3)$$

$$D_j = \max (A_j + T_{dq},\ T_{cq}), \quad (4)$$

$$a_j = \min_{\forall_{i \rightsquigarrow j}} (d_i + \delta_{ij} - P), \quad (5)$$

$$d_j \simeq T_{cq}. \quad (6)$$

The value of a_j is negative in practice, i.e. the earliest data typically arrives at a capturing latch before the rising edge of clock, which allows the approximation in (6). The variables in (3)–(6) let the setup and hold time constraints to be checked at each latch j one by one rather than at each latch pair:

$$S_j: \quad A_j \leq W_j - T_{su}, \quad (7)$$

$$H_j: \quad a_j \geq W_j + T_{hd} - P. \quad (8)$$

Note that (7) and (8) are now sufficient and necessary conditions, as opposed to (1) and (2), which are only sufficient.

III. Problem Formulation and Yield Estimation

A. Problem Formulation

In deterministic pulse width allocation in Section II-A, $\mathcal{W}$ contains a list of constants. In statistical counterpart, which is a main focus of this paper, each pulse width in $\mathcal{W}$ is modeled as a random variable (RV). We are also given a constraint on the probability that a circuit satisfies (7) and (8) at all latches, i.e. the timing yield. To state a problem in a formal fashion,

Problem 1 Given a netlist of a pulsed-latch circuit and a set of RVs $\mathcal{W}$, the SPWA optimization problem is to allocate a pulse width $W_i \in \mathcal{W}$ for each i, such that the clock period is minimized while the probability that (7) and (8) are satisfied at all latches is not less than the yield constraint.

The algorithm to solve Problem 1 is addressed in Section IV-A, which relies on yield estimation of a circuit with a particular configuration of pulse widths.

B. Yield Estimation

1) Constraint Graphs: The timing yield of a pulsed-latch circuit is the probability that (7) and (8) are satisfied at every latch i:

$$Y = Prob\left[\bigwedge_{\forall i} (S_i \wedge H_i)\right]. \quad (9)$$

Primary outputs are treated as latches whose W_i, T_{su}, and T_{hd} are all zero.

To estimate Y, we introduce two constraint graphs, one for S_i, called a setup time constraint graph G_S, and the other for H_i, called a hold time constraint graph G_H. The yield Y then can be obtained by computing the probability that G_S and G_H do not have a positive cycle [9]. In G_S, the vertices consist of A_js and D_js plus a dummy node O with value of 0; similarly the vertices of G_H consist of a_js and d_js plus a dummy node o with value of 0. To construct G_S, we transform (3) into a set of inequalities:

$$D_i + (\Delta_{ij} - P) \leq A_j, \forall_{i \rightsquigarrow j} \quad (10)$$

Each inequality of (10) corresponds to an edge (D_i, A_j) with a weight of $\Delta_{ij} - P$ on it. We also transform (4) into a couple of inequalities:

$$A_j + T_{dq} \leq D_j, \quad (11)$$

$$T_{cq} \leq D_j, \quad (12)$$

where the former is modeled by an edge (A_j, D_j) with a weight of T_{dq}, and the latter by (O, D_j) with T_{cq} as a weight. We finally take (7) and convert it to an edge (A_j, O) with $-W_j + T_{su}$ as a weight. The G_H can be built similarly: (5) is modeled by a set of edges (a_j, d_i) with weight of $-\delta_{ij} + P$; (6) is modeled by (d_i, o) with $-T_{cq}$ as a weight and (o, d_i) with T_{cq} as a weight; (8) is modeled by (o, a_j) with a weight of $W_j + T_{hd} - P$.

Example 1 Consider an example circuit shown in Fig. 4(a). The edges of G_S corresponding to $1 \rightsquigarrow 2$ are highlighted in Fig. 4(b) with their weights; $D_1 + \Delta_{12} - P \leq A_2$ is modeled by (D_1, A_2), $A_1 + T_{dq} \leq D_1$ by (A_1, D_1), $T_{cq} \leq D_1$ by (O, D_1), and $A_2 \leq W_2 - T_{su}$ by (A_2, O). Similarly, the edges of G_H corresponding to $1 \rightsquigarrow 2$ are highlighted in Fig. 4(c). It should be noted that having both edges (o, d_1) and (d_1, o) in G_H is redundant since the cycle consisting of the two edges has a zero weight and thus does not affect the yield. The edge (o, d_1) can always be removed while (d_1, o) cannot because the latter is included in other cycles as readily can be checked in Fig. 4(c).

2) Yield Estimation: In G_S and G_H, all the values in edge weights are constants except W_is, which are RVs. If W_is are not involved in any cycles, the yield is either 0.0 (positive cycles exist) or 1.0 (no positive cycles); otherwise, we discover

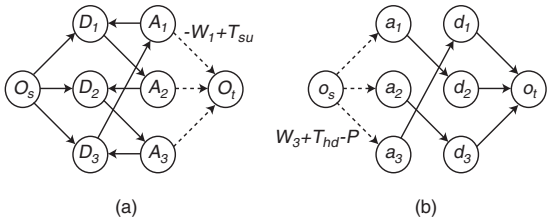

Fig. 5. G_S and G_H in Fig. 4 are transformed into (a) G'_S by splitting O into two vertices O_s and O_t and (b) G'_H by splitting o into o_s and o_t.

each cycle that involves W_is, compute the probability of that cycle being negative, and aggregate all the probabilities to compute the yield Y.

Fortunately, only incoming edges of O and outgoing edges of o, which are shown as dotted arrows in Fig. 4(b) and (c), involve W_is in their weights. This allows us to model O as two vertices: O_s having all outgoing edges of O and O_t having all incoming edges, as shown in Fig. 5(a). The same is performed in o as shown in Fig. 5(b). Note that, in new constraint graph G'_S (G'_H), the path from O_s to O_t (from o_s to o_t) corresponds to the cycle that involves W_i in the original graph G_S (G_H); the cycle in G'_S corresponds to the cycle that does not involve W_i in G_S; there is no cycle in G'_H since all the cycles in G_H contain o. Therefore, we first try to find positive cycles (via Bellman-Ford [10]) of G'_S. If there are any, we return in fail, i.e. the clock period does not satisfy timing constraints or, equivalently, yield is zero for the given allocation of pulse width; otherwise we proceed to find each path from O_s to O_t.

The path from O_s to O_t consists of the longest path from O_s to A_i, whose path weight is denoted by $d(O_s, A_i)$, and an edge from A_i to O_t (see Fig. 5(a)). This implies that we want to compute the probability that

$$d(O_s, A_i) - W_i + T_{su} \leq 0 \tag{13}$$

is satisfied so that the cycle corresponding to this path becomes negative. Similarly, the path from o_s to o_t consists of an edge from o_s to a_i and then the longest path from a_i to o_t, whose path weight is denoted by $d(a_i, o_t)$. Thus, we want to have

$$W_i + T_{hd} - P + d(a_i, o_t) \leq 0 \tag{14}$$

Combining (13) and (14) yields

$$S_i \wedge H_i : \ d(O_s, A_i) + T_{su} \leq W_i \leq -d(a_i, o_t) - T_{hd} + P. \tag{15}$$

For a given pdf of W_i, (15) can be readily computed, which represents the probability that both S_i and H_i are satisfied at latch i. Once (15) is computed for all latches, they are multiplied to yield Y, provided that W_is are independent, i.e. each latch is driven by its own pulse generator. This is refined in Section IV-B when more than one latches share the same pulse generator.

IV. Statistical Pulse Width Allocation

In order to solve Problem 1, we perform a binary search similar to Section II-A. Specifically, we select a median clock period P_m in the range (0, P_{crit}). We check whether there exists any allocation of pulse widths such that corresponding yield, computed by the process discussed in Section III-B.2, is no less than yield constraint Y_c, which is performed by a routine SPWA shown in Fig. 6. If we succeed, we repeat with new range (0, P_m); otherwise (P_m, P_{crit}) is used in the next iteration.

Algorithm SPWA(P, Y_c)

L1 Sort a list of pulse widths $\mathcal{W}$ in order of increasing width
L2 $W_i \leftarrow \min_{w \in \mathcal{W}} w$
L3 Generate G'_S and G'_H
L4 **if** there exists a positive cycle in $G'_S \backslash \{(A_i, O_t), \forall i\}$ **then**
  return fail
L5 $Y \leftarrow \prod Y_i = \prod Prob[S_i \wedge H_i]$
L6 **if** $Y \geq Y_c$ **then return** success
L7 **for** each latch i **do**
L8 **while** $W_i < \max_{w \in \mathcal{W}} w$ **and** $Y_i < Y_c$ **do**
L9 Select next wider $W_i \in \mathcal{W}$ and update Y_i and Y
L10 **if** $Y_i < Y_c$ **then return** fail
L11 **if** $Y \geq Y_c$ **then return** success
L12 **else** *Refine_with_clustering*(P, Y_c)

Fig. 6. Pseudo-code of SPWA algorithm.

A. Algorithm

We initially assign the minimum pulse width to all latches (L2) and estimate yield Y following the procedure of Section III-B.2, which corresponds to L3–L5. If $Y \geq Y_c$, we simply return in success (L6); otherwise we try to find another configuration of pulse widths (L7–L10). This is done by selecting each latch i (L7) and trying next wider pulse width until its probability to satisfy (15), denoted by Y_i, becomes larger than Y_c (L8 and L9). If such pulse width does not exist, we return in fail (L10) because Y, which is the product of Y_is where $0 \leq Y_i \leq 1$, is also smaller than Y_c in such a case.

If $Y \geq Y_c$ after we modify the pulse width of some latches, we return in success (L11). Even if $Y < Y_c$, there is still a chance to increase Y (L12). This is because the latches of the same pulse width will be grouped in clusters, so that each cluster is driven by a single pulse generator. This may increase Y, because the latches in the same cluster will now share the same RV of pulse width. Thus, when $Y < Y_c$ even though all Y_is is no less than Y_c, we call a routine $Refine\_with\_clustering$ (L12), which takes account of such clustering to improve yield.

B. Refinement with Clustering

Let latches i and j be driven by the same pulse width, i.e. pdfs of W_i and W_j are the same as shown in Fig. 7. The computation of Y_i is based on (15), i.e. $Prob[a \leq W_i \leq b]$ for some constants a and b; similarly, Y_j is the probability that $c \leq W_j \leq d$ is satisfied. When W_i and W_j are independent variables, $Y_i Y_j$ is simply a numerical product of Y_i and Y_j. When they are the same RVs because i and j share the same pulse generator, however, $Y_i Y_j = Prob[\max(a, c) \leq W_i \leq$

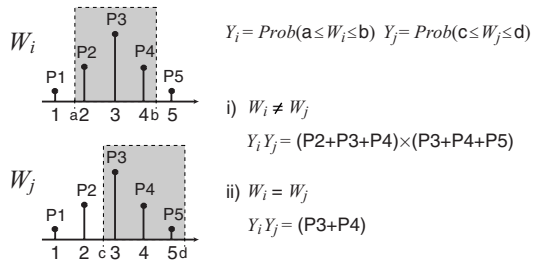

Fig. 7. Computation of aggregate yield for Y_i and Y_j.

Algorithm *Refine_with_clustering* (P, Y_c)

L1 Group latches of the same pulse width: $C_j \leftarrow \{i | W_i = \mathcal{W}_j\}$
L2 $Y \leftarrow \prod Y(C_j/m)$
L3 **if** $Y \geq Y_c$ **then return** success
L4 **while** $Y < Y_c$ **and** $\exists_i$, Y_i increases with next wider W_i **do**
L5 **for** each C_j in order of increasing $\mathcal{W}_j$ **do**
L6 **for** each latch $i \in C_j$ in order of increasing Y_i **do**
L7 **if** $\exists_{k>j}$, $Y(C_j/m) \cdot Y(C_k/m)$
$\leq Y(\{C_j \setminus \{i\}\}/m) \cdot Y(\{C_k \cup \{i\}\}/m)$ **then**
L8 Assign $\mathcal{W}_k$ to W_i
L9 Remove i from C_j and add to C_k
L10 Update Y_i, $Y(C_j/m)$, $Y(C_k/m)$, and Y
L11 **if** $Y \geq Y_c$ **then return** success
L12 **else return** fail

Fig. 8. Pseudo-code of *Refine_with_clustering*.

$\min(b, d)$ as shown in Fig. 7. Therefore, grouping latches of the same pulse width has a chance to increase the yield.

Fig. 8 shows the algorithm. We group all the latches with the same pulse width into a cluster C_j (L1). Each cluster is then partitioned into sub-clusters of m latches, denoted by C_j/m where m is the maximum number of latches each pulse generator can drive. The yield of each C_j/m, denoted by $Y(C_j/m)$, is computed and aggregated for a new yield Y (L2). The computation of $Y(C_j/m)$ is done by

$$Y(C_j/m) = \prod_k Prob\left[L_k \leq \mathcal{W}_j \leq U_k\right],$$

where $\mathcal{W}_j$ is the j-th width in $\mathcal{W}$,

$$L_k = \max_i \{d(O_s, A_i) + T_{su}\},\ i \in C_j^k$$

$$U_k = \min_i \{-d(a_i, o_t) - T_{hd} + P\},\ i \in C_j^k,$$

where C_j^k indicates a k-th cluster of C_j. If $Y \geq Y_c$, we return in success; otherwise we iterate each latch i in order of increasing pulse width and increasing Y_i (L5–6) and assign the wider width to the latch if it improves Y considering subsequent changes in the yield of clusters (L7–10).

V. Experimental Results

A. Experimental Setting

We experimented on a set of sequential circuits taken from the ISCAS benchmarks and open cores [11]. The second and third columns of Table I report the number of combinational gates and the number of pulsed-latches after each circuit was synthesized with SIS [12]. A gate library (for SIS) consisting of 78 gates was built based on a commercial 45-nm technology. The netlist was then submitted to the pulse width allocation algorithm; both DPWA (Section II-A) and SPWA (Section IV-A) algorithms were implemented in SIS.

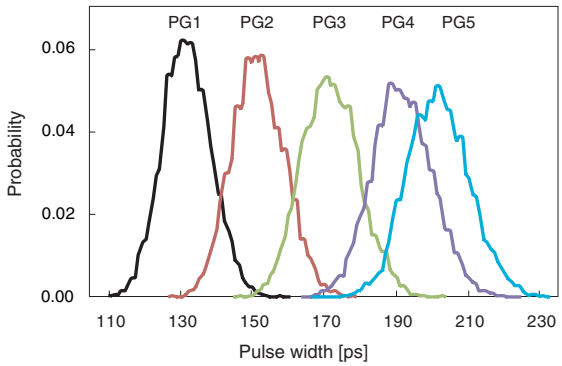

Fig. 9. Pulse width distributions of five PGs obtained by the Monte Carlo simulation.

B. Pulse Generators

We built a set of five pulse generators (PGs) for the experiment. Each PG consists of three inverters and one AND gate, where the delay through cascaded inverters determines the pulse width. The minimum pulse width (PG1) was set to 130 ps, which is the minimum value for safe capturing of data suggested by the technology we used; the remaining pulse widths (PG2∼PG5) were determined empirically. Each PG was submitted to the Monte Carlo simulation 10,000 times using SPICE to derive its pdf, which is reported in Fig. 9. Each pdf was sampled at 15 points; the final discrete pdf was then used for the experiment.

C. Clock Period and Yield

The fourth column of Table I reports the initial clock period, where all latches receive the same (130 ps) pulse width, i.e. no time borrowing is allowed. The remaining columns compare the clock period and the yield obtained by SPWA and DPWA under two different sample yield constraints (0.85 and 0.95). For DPWA, we first run the algorithm shown in Fig. 3; the pulse width used in this case corresponds to the mean of each pdf shown in Fig. 9. We then compute the yield Y assuming that each latch follows corresponding pdf of Fig. 9. If $Y \geq Y_c$, we decrease the clock period P until Y becomes smaller than Y_c; otherwise we increase P. The clock period P_{dpwa}, thus obtained, is reported in column 7 when Y_c = 0.85 and in column 11 when Y_c = 0.95; the entries with – correspond to the case when Y_c can never be satisfied due to hold time violations; P_{dpwa} was smaller than P_{ini} by 7.6% and 7.3% on average when Y_c was satisfied. By using SPWA, however, the clock period was even more reduced; by 12.2% when Y_c = 0.85 and by 11.7% when Y_c = 0.95. More importantly, the yield constraints were satisfied with all circuits while four (out of eleven) circuits failed in deterministic counterpart.

To see how the minimum clock period changes for various yield constraints, we took two benchmark circuits. Fig. 10(a)

TABLE I

COMPARISON OF THE CLOCK PERIOD AND THE YIELD OBTAINED BY SPWA AND DPWA UNDER THE YIELD CONSTRAINT (Y_c)

Benchmark				Yield constraint (Y_c) = 0.85				Yield constraint (Y_c) = 0.95			
Name	# Gates	# PLs	P_{ini} (ps)	P_{spwa} (ps)	Y_{spwa}	P_{dpwa} (ps)	Y_{dpwa}	P_{spwa} (ps)	Y_{spwa}	P_{dpwa} (ps)	Y_{dpwa}
s1423	628	74	1467	1381	0.85	1416	0.92	1401	0.98	1421	0.99
s9234	1201	135	682	637	0.87	655	0.91	636	0.97	656	0.96
s13207	3054	490	867	801	0.92	833	0.95	801	0.98	838	0.99
s15850	3860	515	1163	998	0.85	–	0.49	1003	0.96	–	0.49
s38584	12260	1424	962	890	0.85	933	0.89	895	0.96	935	0.98
irda_fir	278	37	464	390	0.91	–	0.35	395	0.98	–	0.35
i2c_master	338	49	628	530	0.90	530	0.90	531	0.97	531	0.97
mc	886	90	387	314	0.85	–	0.49	320	0.96	–	0.49
wb_dma	2004	198	675	631	0.93	648	0.93	631	0.98	648	0.98
t400	2240	176	915	822	0.87	–	0.04	827	0.96	–	0.04
usbc	2328	402	639	476	0.91	516	0.88	481	0.98	521	0.99
Average			1.00	0.88		0.92		0.88		0.93	

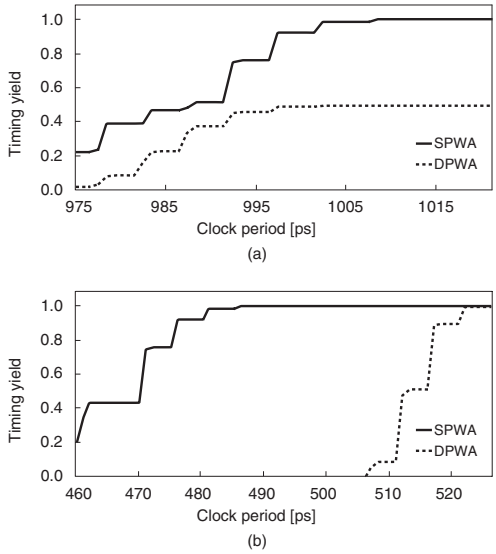

Fig. 10. Timing yield of (a) `s15850` and (b) `usbc` for various clock periods.

shows the result for `s15850`; the yield obtained by DPWA cannot meet Y_c beyond 0.5 because it assigns many large pulse widths to allow time borrowing to the critical paths, which increases the probability of hold time violations. SPWA avoids such allocation by anticipating the probability of hold time violations. Fig. 10(b) is the result for `usbc`; DPWA assigns only PG1s and thus does not benefit from time borrowing because the flip-flop timing model in (1) limits the use of large pulse widths when critical paths are abutted via a latch. Although the trend of two graphs is different, it clearly shows that the minimum clock period obtained by SPWA is always smaller than that by DPWA.

VI. CONCLUSION

Because the pulse widths at $+3\sigma$ point does not represent the worst case for the time borrowing, we have presented the statistical pulse width allocation algorithm, which models pulse widths as RVs, to minimize the clock period of pulsed-latch circuits under the yield constraint. The latch clustering has been combined to improve the yield by considering the correlation of delivered pulses. Experiments on benchmark circuits with 45-nm technology show that, under the same yield constraint, the clock period of SPWA is always smaller than that of DPWA. Moreover, SPWA always satisfies the yield constraint while DPWA fails for some circuits.

In this work, the delay of combinational paths (Δ_{ij} and δ_{ij}) was assumed as a constant. Our work can be extended to consider the variations in Δ_{ij} and δ_{ij}, which is left for a future work.

VII. ACKNOWLEDGEMENT

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST), F01-2007-000-10141-0.

REFERENCES

[1] L. T. Clark et al., "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1599–1608, Nov. 2001.
[2] S. D. Naffziger et al., "The implementation of the Itanium 2 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1448–1460, Nov. 2002.
[3] S. Shibatani and A. Li, "Pulse-latch approach reduces dynamic power," July 2006, EE Times.
[4] N. Weste and D. Harris, Eds., *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison Wesley, 2005.
[5] 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. on Computer Aided Design*, Nov. 2008, pp. 224–229.
[6] P. S. Zuchowski et al., "Process and environmental variation impacts on asic timing," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2004, pp. 336–342.
[7] D. P. Singh and S. D. Brown, "Constrained clock shifting for field programmable gate arrays," in *Proc. Int. Symp. on Field-Programmable Gate Arrays*, Feb. 2002, pp. 121–126.
[8] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," in *Proc. Design Automation Conf.*, 1990, pp. 111–117.
[9] R. Chen and H. Zhou, "Clock schedule verification under process variations," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2004, pp. 619–625.
[10] J. Kleinberg and E. Tardos, Eds., *Algorithm Design*, Addison Wesley, 2006.
[11] "Opencores," http://www.opencores.org/.
[12] E. Sentovich et al., "SIS: a system for sequential circuit synthesis," May 1992, Tech. Rep. UCB/ERL M92/41.