Analysis and Optimization of Variable-Latency Designs in the Presence of Timing Variability

Chang-Lin Tsai, Chao-Wei Cheng, Ning-Chi Huang and Kai-Chiang Wu
Department of Computer Science
National Chiao Tung University, Hsinchu, Taiwan
{cltsai0812, cwcheng8229, nchuang, kcw}@cs.nctu.edu.tw

***Abstract*—Circuit performance has been the key design constraint for over a decade. Variable-latency design (VLD) paradigm was proposed for optimizing the overall performance in terms of throughput. In addition, process variations and aging effects manifest themselves as gate delay shifts, and in turn cause variability of circuit timing (timing variability). Required for dealing with the impact of timing variability better, detailed evaluation and analysis of circuit timing for VLD are actually not straightforward. In this paper, we present a systematic methodology for analyzing a VLD circuit, and identifying critical 1-cycle and 2-cycle paths/gates. Based on the criticality analysis, a gate sizing framework using particle swarm optimization (PSO) is proposed. Our objective is, in a less pessimistic fashion, making constructed VLD circuits better (less vulnerable to timing variability). The proposed framework is experimentally verified to be runtime-efficient and able to provide promising results. On average, an extra timing margin of 11% can be obtained without lengthening the clock period, and only 4% area overhead is introduced.**

I. Introduction

Circuit performance has been the key design constraint for over a decade. Traditional performance enhancement techniques attempt to minimize the longest path delay which determines the clock period. However, the longest path in many designs may be a false path, or may be activated infrequently. The strategy of minimizing such worst-case scenario usually leads to unnecessary pessimism. More recently, the *variable-latency design* (VLD) paradigm was presented [1]-[5]. The main idea of VLD is to explore timing speculation by allowing two clock cycles for paths with longer delays, while allowing one cycle for others (with shorter delays). Therefore, the clock period can be smaller than the longest path delay, and as long as those paths with delays greater than the clock period are not activated frequently, the overall performance in terms of throughput can be significantly enhanced.

On the other hand, the design and manufacturing of semiconductor devices have experienced dramatic innovations, at the cost of downgrading reliability of nanoscale integrated circuits (ICs) and system-on-chips (SOCs). The 2013 ITRS [6] projects that the reliability of sub-100nm technology nodes can reach a noteworthy order of 10^3 FITs (failures in 10^9 hours). Some of the major challenges driving reliability-aware IC/SOC design methodologies encompass process variations [7][8] and aging effects [9][10]. These effects manifest themselves as gate delay shifts, and in turn cause variability of circuit timing (timing variability). To deal with the impact of timing variability, an intuitive approach is incorporating extra timing margins during the design stage, which is effective but may result in overly pessimistic design.

Required for dealing with the impact better, detailed evaluation and analysis of circuit timing for VLD are actually ***not*** straightforward.

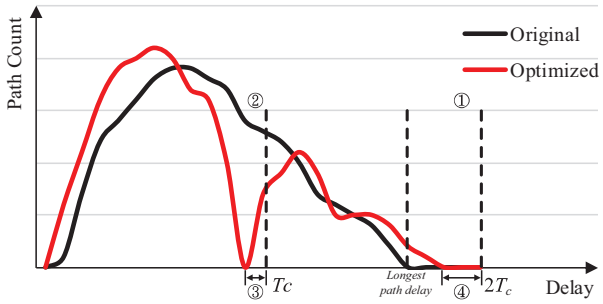


Fig. 1. Path delay distribution

In a VLD circuit, paths are classified into two categories: 1-cycle paths (i.e., paths that are shorter and allowed/assigned 1 clock cycle for computation), and 2-cycle paths (i.e., paths that are longer and allowed/assigned 2 clock cycles). The topologically long paths, which are most likely 2-cycle paths in VLD, are not the only candidates to be optimized. The black line in Fig. 1 shows the path delay distribution of a circuit. To synthesize the circuit into a VLD, the clock period (T_c) is typically set to 60-80% of the longest path delay in order for a preferable tradeoff between performance gain and area penalty [5]. In this circumstance, 2-cycle paths are usually less critical than 1-cycle paths due to larger timing margins (i.e., interval ①) with respect to "2-cycle" time span, meaning that 1-cycle paths are the major candidates for optimization. Also as the black line shown in Fig. 1, no matter what clock period is used for VLD synthesis, there are usually ***many*** 1-cycle paths with delays close to the clock period. Unlike 2-cycle paths, these 1-cycle paths have very limited or even no timing margins (i.e., interval ②, actually none) with respect to "1-cycle" time span, making the resulting VLD highly vulnerable to timing variability.

Related work on VLD mainly focuses on, given a logic circuit, constructing its area-efficient VLD for performance optimization. Among various existing studies, [1]-[5] target general logics while [11]-[13] target arithmetic logics. The authors of [14][15] proposed to consider aging and variation, respectively, when constructing the VLD of a circuit. All of these studies aim at optimizing circuit performance, and the aforementioned issues (i.e., the criticality of 1-cycle paths in a constructed VLD circuit and its vulnerability to timing variability) are not ever addressed explicitly.

However, identifying critical 1-cycle paths is not as simple as identifying critical 2-cycle paths. In this paper, we present a systematic methodology for analyzing a VLD circuit, and identifying critical 1-cycle ***and*** 2-cycle paths. Based on the criticality analysis, a gate sizing framework using particle swarm optimization (PSO) is proposed, where non-critical 2-cycle paths can be exploited (slowed down) for optimizing (speeding up) critical 1-cycle paths. The red line in Fig. 1 depicts the expected path delay distribution of our optimized VLD. The distribution becomes bimodal: 1-cycle paths (forming the

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

left peak) and 2-cycle paths (forming the right peak) both have sufficient timing margins/slacks (i.e., intervals ③ and ④, respectively) to tolerate a certain degree of timing variability. Note that the proposed framework differs from existing work on constructing good VLD circuits. Instead our objective is *making constructed VLD circuits better (less vulnerable to timing variability), in a less pessimistic fashion* compared to simply lengthening clock period. The contributions and advantages of our work are twofold:

- This is the first work dealing with the impact of timing variability given a constructed VLD circuit. Prior work basically synthesizes an original logic circuit (non-VLD, or fixed-latency design) to its VLD counterpart for performance optimization, and the resulting VLD may be extremely vulnerable to timing variability as described earlier. We explicitly address the problem by accurately identifying critical 1-cycle and 2-cycle paths and manipulating them for robustness/tolerance to timing variability.
- The proposed optimization framework based on gate sizing is formulated into a particle swarm optimization (PSO) problem. Our PSO formulation is experimentally verified to be runtime-efficient and able to provide promising results. On average, an extra timing margin of 11% can be obtained without lengthening the clock period, and only 4% area overhead is introduced.

II. Timing Criticality Analysis

Static Timing Analysis (STA) is a method of checking timing requirements inside a design by calculating the propagation delay along each path. It also identifies timing-critical paths and analyzes the timing criticality of each gate. Unlike a fixed-latency circuit, the critical paths in a VLD circuit may not only be those topologically long paths in the circuit, but also those whose delays are smaller than (and close to) the clock period. Thus, traditional STA may fail to identify the correct set of critical paths in VLD and lead to failure during optimization. We propose Timing Criticality Analysis (TCA) for VLD, which aims to accurately analyze the timing of VLD and find out all critical paths/gates in VLD. In the sequel, we briefly introduce traditional STA, on which our critical analysis engine is based.

A. Traditional Static Timing Analysis (STA)

STA calculates the propagation delay along each path. There are two modes of timing: late mode and early mode. In the late mode, the arrival time of a gate g (denoted by $AT_L(g)$) indicates the *latest* time a signal can change at the output of g, determined by the *longest* path from a primary input to g. The required time of a gate g (denoted by $RT_L(g)$) indicates the *latest* time a signal is allowed to change at the output of g, determined by the *longest* path from g to a primary output. Arrival times propagate in a topological order from primary inputs to primary outputs, while required times propagate from primary outputs to primary inputs.

$$Slack_L(g) = RT_L(g) - AT_L(g) \quad (1)$$

The late-mode slack of a gate g (denoted by $Slack_L(g)$) is the difference between its required time and arrival time. It quantifies the amount by which a signal can be delayed at g and not increase the length of the longest path through the network. Typically, a negative $Slack(g)$ indicates that g is a critical gate.

In the early mode, the arrival time of a gate g (denoted by $AT_E(g)$) indicates the *earliest* time a signal can change at the output of g, determined by the *shortest* path from a primary input to g. The required time of a gate g (denoted by $RT_E(g)$) indicates the *earliest* time a signal is allowed to change at the output of g, determined by the *shortest* path from g to a primary output.

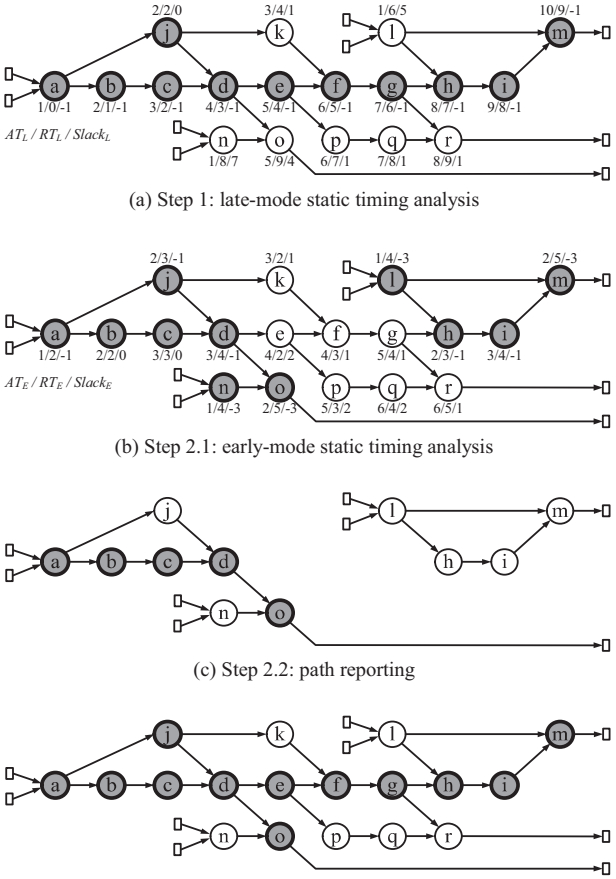


Fig. 2. Example of Timing Criticality Analysis for VLD

The early-mode slack of a gate g (denoted by $Slack_E(g)$) is the difference between its arrival time and required time. It quantifies the amount by which a signal can be sped up at g and not decrease the length of the shortest path in the network.

$$Slack_E(g) = AT_E(g) - RT_E(g) \quad (2)$$

B. Timing Criticality Analysis (TCA) for VLD

Definition 1 (*1-cycle critical paths/gates*): A path is defined as a 1-cycle path if and only if its delay is smaller or equal to clock period, T_c; in other words, it requires no more than one clock cycle to complete an operation. We define those gates on a 1-cycle path as 1-cycle gates. Further, a 1-cycle path is called critical if its delay falls within the closed interval $[T_c \times (1\text{-}Bound), T_c]$ where *Bound* is the range of delay for defining critical paths. We define those gates on a 1-cycle critical path as 1-cycle critical gates.

Definition 2 (*2-cycle critical paths/gates*): A path is defined as a 2-cycle path if and only if its delay is larger than T_c; in other words, it requires more than one clock cycle to complete an operation. We define those gates on a 2-cycle path as 2-cycle gates. Further, a 2-cycle path is called critical if its delay falls within the interval $[2T_c \times (1\text{-}Bound), 2T_c]$. We define those gates on a 2-cycle critical path as 2-cycle critical gates.

Traditional STA is able to find out all critical gates in a fixed-latency circuit. However, when considering VLD, it may fail because traditional STA only recognizes the critical gates on topologically long paths, which are 2-cycle critical gates, while missing 1-cycle critical gates. In the sequel, we propose to extend the concept of traditional STA, such that both 1-cycle critical gates and 2-cycle critical gates can be identified. More specifically, the slack of a gate characterizes the gate in terms of timing criticality. The criticality of a gate is related to the magnitude of its slack: the smaller slack a gate has, the more critical the gate is. Our method of timing criticality analysis for VLD consists of two steps: the first step identifies 2-cycle critical paths/gates, while the second step identifies 1-cycle critical paths/gates.

Step 1 (*late-mode static timing analysis*): Step 1 marks all 2-cycle critical gates by using late-mode STA. Let the required times of primary outputs be $2T_c$. Then, if $Slack_L(g) = 1$, it means that gate g can be delayed by 1 without increasing the delay of the longest path through the network and thus having to make $2T_c$ larger. On the other hand, if $Slack_L(g) = -1$, it means that gate g needs to be sped up by 1 so as to ensure no timing violation without making $2T_c$ larger.

To realize Step 1, we set an appropriate required times for primary outputs. The delay of a 2-cycle critical path is within $[2T_c \times (1\text{-}Bound), 2T_c]$. Thus, the required times of primary outputs is set to $2T_c \times (1\text{-}Bound)$. See Fig. 2(a) as an example. Consider it as a VLD circuit and all gate delays are assumed to be 1. Let $T_c = 5$ and $2T_c = 10$, $Bound = 0.1$. Therefore, the paths with delays $\geq 2T_c \times (1\text{-}Bound) = 9$ are 2-cycle critical paths. We set the required times of primary outputs to 9, then perform late-mode STA, and finally, check $Slack_L(g)$ of each gate. For each gate with $Slack_L(g) \leq 0$, mark it as a 2-cycle critical gate, and in Fig. 2(a) a set of gates {a, b, c, d, e, f, g, h, i, j, m} are marked.

Step 2.1 (*early-mode static timing analysis*): Step 2.1 aims to mark all 1-cycle gates, no matter whether it is critical or not. Let the required times of primary outputs be T_c. Then, if $Slack_E(g) = 1$, it means that gate g can be sped by 1 without decreasing the delay of the shortest path in the network. That is, the shortest path passing through gate g has a delay of T_c+1. On the other hand, if $Slack_E(g) = -1$, it means the shortest path passing through gate g has a delay of T_c-1. That is, there exists (at least) a 1-cycle path with a delay of T_c-1 passing through this specific gate.

To realize Step 2.1, we set an appropriate required times for primary outputs. The delay of every 1-cycle path must be smaller than or equal to T_c. Thus, the required times of primary outputs is set to T_c. Consider the same example in Fig. 2(b). The paths with delay $\leq T_c = 5$ are 1-cycle paths. We set the required times of primary outputs to 5, then perform early-mode STA, and finally, check $Slack_E(g)$ of each gate. For each gate with $Slack_E(g) \leq 0$, mark it as a 1-cycle gate, and in Fig. 2(b) a set of gates {a, b, c, d, h, i, j, l, m, n, o} are marked. All of these gates lie on one (or more) 1-cycle path.

Step 2.2 (*path reporting*): To mark 1-cycle "critical" gates, we employ path reporting. This method uses a min-heap to store partial paths; each element in the heap records the current partial path, its length, and its slack (i.e., the slack of the last gate on the path). The key of this heap is the slack: the smaller slack the current partial path has, the more critical it is. Initially, push partial paths with only primary inputs into the heap. After that, iteratively pop the top partial path (with minimum slack) out of the heap until the heap becomes empty or the top partial path meets the termination condition. If the popped path is an incomplete path (i.e., its last gate not a primary output), we generate new partial path(s) by extending the path from its last gate by one level downstream, and then push the extended path(s) into the heap. Otherwise, if the popped path ends with a primary output and its length falls within $[T_c \times (1\text{-}Bound), T_c]$, it will be reported and all gates on the reported path will be marked as 1-cycle "critical" gates. If the popped path is a complete path but its length is less than $T_c \times (1\text{-}Bound)$, it means that there does not exist unreported critical path. Hence, the procedure of path reporting finishes.

A gate is a 1-cycle critical gate only if it is a 1-cycle gate. As shown in Fig. 2(b), we have extracted all of the 1-cycle gates. Therefore, path reporting only needs to be applied on the sub-network(s) consisting of 1-cycle gates, which significantly reduces the search space and makes our path reporting runtime-efficient. It is possible that a 2-cycle path is included in the sub-network of 1-cycle gates. To further prune the search space during path reporting, we can discard a partial path whose length exceeds T_c, because it is no longer a 1-cycle path, nor 1-cycle critical path to be reported. The result of applying path reporting on the reduced, sub-network(s) of 1-cycle gates is shown in Fig. 2(c). A 1-cycle path with delay $\geq T_c \times (1\text{-}Bound) = 4.5$ is reported as a 1-cycle critical path, and a set of gates {a, b, c, d, o} on the path are marked as 1-cycle critical gates. As shown in Fig. 2(d), which indicates whether a gate is a 1-cycle critical gate (i.e., {o}), or a 2-cycle critical gate (i.e., {e, f, g, h, i, j, m}), or both (i.e., {a, b, c, d}), or neither (i.e., {k, l, n, p, q, r}). This piece of information is the key guideline for optimizing a VLD circuit in terms of robustness/tolerance to timing variability. The proposed optimization methodology is presented in the following section.

III. Proposed Methodology based on Particle Swarm Optimization

The goal of our methodology is to achieve a higher degree of tolerance to timing variability, by reducing the delay of a VLD circuit based on the criticality analysis described in Section II. With delay reduced, larger timing margins can be obtained and the lifetime of the VLD circuit is thus improved. Toward this end, the proposed optimization framework based on gate sizing is formulated into a particle swarm optimization (PSO) problem. We first review the concepts of gate sizing and PSO, and then our proposed methodology will be presented.

A. Gate Sizing

1) Gate Sizing Cases

As mentioned above, sizing a gate not only changes its own delay but also changes the delay of its input gates. Thus, we propose "gate sizing cases" based on the effort of Timing Criticality Analysis for VLD, which determines whether a gate should be upsized or not.

Case 1: If a gate is marked as critical gate, it means slowing down this gate will cause timing violation. Though downsizing this gate slows down itself, its input gates get sped up and might speedup the entire path in the end. As a result, gates in Case 1 can be downsized or upsized.

Case 2: If a gate is a non-critical gate and there exists critical gate in its inputs, then upsizing this gate slows down its input gate, leading to violate the timing constraint. Therefore, gates in Case 2 can only be downsized.

Case 3: If a gate is a non-critical gate and there does not exist any critical gate in its inputs, then upsizing or downsizing this gate both may have a better result. As a result, gates in Case 3 can be downsized or upsized.

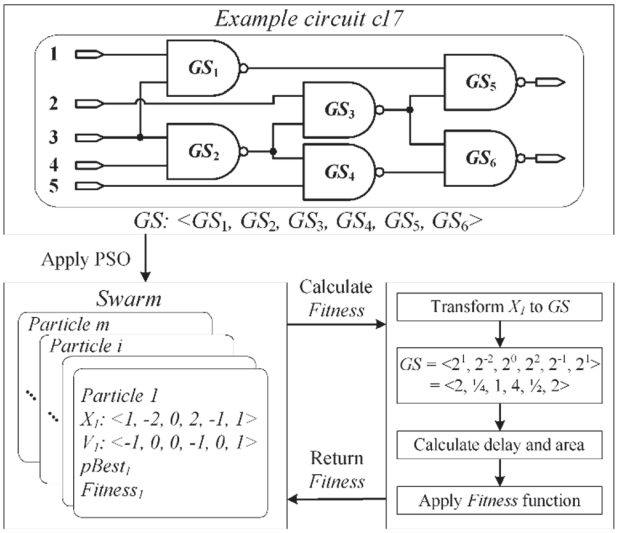


Fig. 3. PSO-based gate sizing implementation

2) Gate Sizing Model & Fitness

When we employ gate sizing to a circuit, each gate in the circuit has an independent size. Therefore, the size of gates in the circuit can be represented as a n-dimension vector:

$$GS: < GS_1, \ldots, GS_n >$$

we define such vector as size vector, GS, where n is the number of gate in the circuit. Size vector strongly influences the delay and area of a circuit. The delay of circuit can be calculated easily with logical effort model and the area is the sum of the size of gates.

To determine the quality of a circuit, we evaluate a fitness value, *Fitness*. For a fixed latency circuit, the *Fitness* is:

$$Fitness(opt) = \alpha\left(1 - \frac{Area(opt)}{Area(ori)}\right) + \beta\left(1 - \frac{Delay(opt)}{Delay(ori)}\right) \quad (3)$$

in this formulation, *ori* is the original circuit, *opt* is optimized circuit. *Delay* indicates the longest path delay of a design. α and β are the weight for area and delay. However, when considering VLD, we must ensure 1-cycle critical paths and 2-cycle critical paths are both optimized. Therefore, the *Fitness* of a VLD is:

$$Fitness(opt) = \alpha\left(1 - \frac{Area(opt)}{Area(ori)}\right) + \beta\left(1 - \frac{Max(Delay_{1-cycle}(opt), Delay_{2-cycle}(opt)/2)}{Delay_{1-cycle}(ori)}\right) \quad (4)$$

where $Delay_{1\text{-}cycle}$ is the longest delay of 1-cycle paths in VLD, $Delay_{2\text{-}cycle}$ is the longest delay of 2-cycle paths. $Delay_{1\text{-}cycle}(ori)$ indicates the clock period of original VLD circuit, T_c. For the optimized VLD, we take the bigger value between $Delay_{1\text{-}cycle}(opt)$ and $(Delay_{2\text{-}cycle}(opt))/2$ as the new T_c. to ensure no timing violation for both 1-cycle paths and 2-cycle paths.

B. Particle Swarm Optimization (PSO)

PSO is a global optimum search algorithm that developed from the simulation of bird flocking behavior [17]. In PSO, each single solution is a particle in the search space, the group of particles is called swarm. Every particle has a velocity which determines the direction for searching optimal solution. Velocity of a particle will be updated during the process of optimization, according to the best solution it has achieved so far and the best solution among all particles. In short, PSO iteratively searches solution by moving particles in the search space. Once a better solution is discovered, it will guide the movement of particles and lead the swarm to optimal solution eventually. A particle i holds the following information:

X_i: particle position, a d-dimension vector $< X_{i,1}, \ldots, X_{i,d} >$.
V_i: particle velocity, a d-dimension vector $<V_{i,1}, \ldots, V_{i,d}>$.
$pBest_i$: the best position that particle i achieved so far.
$gBest$: the global best position, shared by the entire swarm.
$Fitness_i$: a fitness value that determines the quality of X_i.

with the information above, PSO updates the velocity and position to achieve the movement of a particle by following equations:

$$V_i = \omega \times V_i + c_1 r_1 (pBest_i - X_i) + c_2 r_2 (gBest - X_i)$$
$$X_i = X_i + V_i \quad (5)$$

where ω is inertia weight factor. c_1 is cognition learning factor, controls the influence of $pBest_i$. c_2 is social learning factor, controls the influence of $gBest$. r_1 and r_2 are two uniformly distributed random numbers that generated independently. The movement of a particle is influenced by the inertia of particle itself, the best position it achieved so far and the global best position that the entire swarm achieved so far. This make PSO won't easily trap into local optimum and converge to the best solution in a short time.

C. Proposed Methodology

Take an observation on the characters of gate sizing model and particle, we can see there are many similarities. For example, gate sized circuit has a size vector, where particle has a position vector. Another similarity is gate sizing model and particle both hold a *Fitness* that determines their quality. As a result, it is reasonable and efficient to solve gate sizing problem by PSO. [18] proposed a method which used PSO to find the gate widths of two adders. Instead, our proposed method is valid for general circuits. Moreover, we present a precise PSO formulation that solves the gate sizing problem, which has never been proposed.

In our proposed methodology, we regard every particle as a gate sizing circuit. Therefore, given a circuit with n gates, we create a n-dimension position vector for particle i, i.e. X_i: $<X_{i,1}, \ldots, X_{i,n}>$. And the j^{th} component in position vector, $X_{i,j}$, corresponds to the j^{th} component in gate size vector, GS_j (which indicates the size of the j^{th} gate in circuit). Take Fig. 3 as an example, there are total 6 gates in circuit *C17,* so position vector X_i should have dimension $d = 6$.

For simplicity and without loss of generality, we assume there are five different possible sizes for every gate in our gate sizing implementation. Due to the discrete size value used in gate sizing problem, we rescaled the position in PSO with discrete value. With GS_j and $X_{i,j}$ both hold a discrete value, every $X_{i,j}$ can correspond to a GS_j and the position vector can easily transform to a size vector. The domain and transformation of GS_j and $X_{i,j}$ are defined as follows:

$$\text{The } j^{th} \text{ component in GS: } GS_j \in \{¼, ½, 1, 2, 4\},\ 1 \le j \le n$$

$$\text{The } j^{th} \text{ component in } X_i\text{: } X_{i,j} \in \{-2, -1, 0, 1, 2\},\ 1 \le j \le n$$

$$GS_j = 2^{X_{i,j}} \quad (6)$$

Now we present the procedure of particle initialization. After X_i is created, we assign $X_{i,j}$ with a uniformly distributed random value, indicates that particles are uniformly distributed inside the search space. To initialize $pBest_i$ and $gBest$ for particles, *Fitness* of particles must be evaluated. Therefore, we need to transform particles back to

Table 1. Results of timing, area & runtime

Circuit	*Gate #*	*Longest path delay (ns)*	*T_c (ns)*	*Proposed Methodology*			
				Optimized T_c (ns)	*Improvement in timing margin*	*Area overhead*	*Runtime (sec)*
s832	269	0.6809	0.4086	0.3736	8.56%	4.06%	16
s938	333	0.7220	0.4332	0.3930	9.28%	3.38%	26
s1423	559	1.9799	1.1879	1.0018	15.67%	3.17%	696
s1488	599	1.0266	0.6160	0.5649	8.29%	2.28%	33
s1494	604	1.0409	0.6245	0.5826	6.71%	5.25%	38
s3330	1023	0.9837	0.5902	0.5367	9.06%	3.89%	186
s3384	1403	2.1209	1.2725	1.0908	14.28%	3.08%	526
s35932	11878	2.7493	1.6496	1.2613	13.36%	2.01%	698
s38584	12237	4.6158	2.7695	2.3700	14.42%	5.22%	1119
c432	210	0.9744	0.5846	0.5451	6.76%	3.15%	761
c880	334	0.6724	0.4034	0.3842	4.75%	3.11%	124
c7552	2330	2.0351	1.2211	1.0740	12.04%	10.63%	1732
apex3	1891	1.0625	0.6375	0.5752	9.77%	4.09%	122
too_large	529	0.8105	0.4863	0.4399	9.54%	3.51%	57
vda	609	0.8264	0.4959	0.4561	8.03%	2.93%	67
x3	772	0.6396	0.3838	0.3236	15.67%	4.74%	48
x4	362	0.6015	0.3609	0.2860	20.77%	6.00%	22
x6dn	408	0.5930	0.3558	0.3158	11.23%	3.36%	27
x7dn	425	0.5831	0.3499	0.3197	8.63%	1.83%	29
Avg					**10.89%**	**3.98%**	

circuit, that is, X_i back to *GS,* and calculate circuit fitness with the circuit of gate sizing implementation using *GS*. For transformation from X_i to *GS*, each component GS_j is calculated by (6) with the corresponding $X_{i,j}$. After that, calculate the delay and area of *GS* and evaluated the new *Fitness* by (4). The new *Fitness* will then be compared with the *Fitness* of $pBest_i$ and *gBest*. If the new *Fitness* is greater than the *Fitness* of $pBest_i$/*gBest*, update $pBest_i$/*gBest* with current particle position. Take the lower half of Fig. 3 as an example, if X_1 is initialized as $<1, -2, 0, 2, -1, 1>$, then it will be transformed to a $GS = <2^1, 2^{-2}, 2^0, 2^2, 2^{-1}, 2^1> = <2, ¼, 1, 4, ½, 2>$. This *GS* indicates the gate sizing implementation that *Particle 1* currently holds, which *Fitness* evaluation is descripted in the right half side of Fig. 3.

Another modification on PSO is restricting the particle movement with sizing cases that gain from Timing Criticality Analysis for VLD. We update V_i and X_i by (5) as original PSO does. When the position update is finished, we check whether $X_{i,j}$ matches the sizing cases. For example, assume the original design has a position vector X_{ori}. Then, if the gate that corresponds to $X_{i,j}$ can only be downsized, then $X_{i,j}$ must be smaller or equal to $X_{ori,j}$. If a movement violates the sizing cases, then set $X_{i,j}$ to $X_{ori,j}$ and set $V_{i,j}$ to 0. This modification strongly influences the convergence of PSO. If we do not restrict particles' movement by sizing cases, the swarm may take a long time to find a solution. Even worse, the swarm may not find a good enough solution in a limited time.

IV. Experimental Results

In this section, we demonstrate the experimental results of our proposed methodology for analysis and optimization of VLD. We use the benchmark circuits in ISCAS'85, ISCAS'89, LGSynth'91 (including MCNC), and IWLS'05. We synthesized circuits using Berkeley ABC with a TSMC 65nm library.

For the circuits we synthesized, each gate in a circuit is with its nominal size. Such sizing implementation is obviously unrealistic. Therefore, we generated a baseline for each circuit, which has a reasonable gate sizing implementation. More precisely, we first consider every circuit as a fixed-latency design and perform a pre-optimization by PSO with the fitness function of (3). After the baseline circuits are generated, we now consider them as VLDs. To distinguish between one-cycle and two-cycle computations, an additional circuitry called "hold logic" is constructed. The hold logic generates a hold signal that is asserted when a computation requires more than one clock cycle. This is determined by checking whether the computation involves a logic path with delay greater than one clock period, i.e., whether the computation activates such a path. Because our objective is to optimize constructed VLDs, we make the following two assumptions. First, we assume that the hold logic of VLD is already constructed. The hold logic construction was proposed in [2][5]. It can be seen that the delay of hold logic is much lower than the clock period of VLD in most cases. Therefore, we assume the constructed hold logic already have enough timing margin to tolerate timing variability. Then, the clock period of VLD, T_c, needs to be determined. For simplicity and without loss of generality, we set T_c to (0.6×*Longest Path Delay*), and $2T_c$ therefore becomes (1.2×*Longest Path Delay*). There are three reasons for choosing (0.6×*Longest Path Delay*) but not (0.5×*Longest Path Delay*) as the clock period. First, a small clock period may lead to large area overhead to construct hold logic. Second, if the clock period is too small, a large number of paths will be identified as 2-cycle paths, lending to a poor throughput enhancement. Third, if T_c is larger than (0.5×*Longest Path Delay*), and $2Tc$ is larger than (1.0×*Longest Path Delay*), the criticality of 2-cycle paths will be lower. Hence, it is possible to slow down non-critical 2-cycle paths for speeding up 1-cycle critical paths.

A. Results of Timing, Area and Runtime

Table 1 shows the comparison of improvement in timing margin, and area overhead of the baseline and optimized circuits. Column one and two shows the circuit name and the number of gate in the circuit. Column three represents the longest path delay of the baseline circuit, without VLD consideration. Column four shows the clock period of the baseline with VLD consideration, which is T_c = (0.6×*Longest Path Delay*). Column five is the optimized T_c, and columns six and seven show the percentages of improvement in timing margin and area overhead, respectively. The area overhead accounts for the additional area resulting from gate sizing. We assume that the baseline has the same hold logic as the optimized circuit. Column eight is the runtime of the entire methodology, with 1000 iterations of PSO. On average, our methodology has 10.89% improvement in timing margin with only 3.98% area overhead, which implies that an extra timing margin of 10.89% can be obtained without lengthening the clock period. This makes the optimized VLD circuits much less vulnerable to timing variability due to aging, which usually accounts for 10-15% increase in circuit delay.

B. Tolerance to Timing Variability

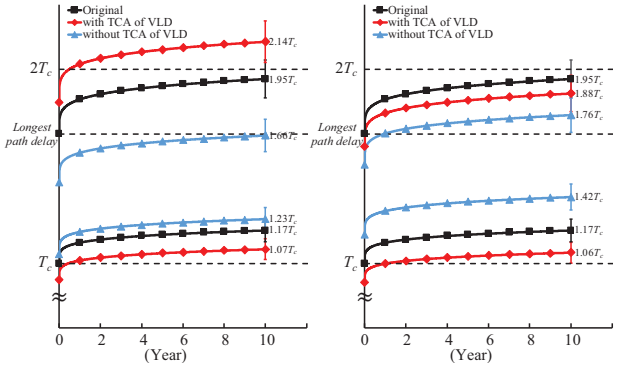
With VLD being optimized, we obtain extra timing margins and the lifetime is thus improved. Fig. 4 shows the various aging behaviors of circuits *s1488* and *apex3* over time. For both circuits, we set their T_c to (0.6×*Longest Path Delay*). The two black lines reveal the aging behaviors of the original VLD, where the upper line indicates 2-cycle critical path delay and the lower line indicates 1-cycle critical path delay. The two red lines reveal the behaviors of the optimized VLD, which considers both 1-cycle and 2-cycle critical paths during optimization. The two blue lines reveal the behaviors of the VLD optimized by our approach which considers only 2-cycle critical paths but not 1-cycle critical paths.

It can be clearly seen that, when considering both 1-cycle and 2-cycle critical paths, 2-cycle paths (less critical) may even be slowed down while 1-cycle paths (more critical) is always sped up. Based on the aging model proposed in [19][20], if 10-year aging is considered, 1-cycle critical path delay degrades to $1.07T_c$ while 2-cycle critical path delay degrades to $2.14T_c$. That is to say, T_c needs to be lengthened to $1.07T_c$ (i.e., by 7%) and thus $2T_c = 2.14T_c$ so as to ensure no timing

violation for 10 years. For the original (non-optimized) VLD, T_c needs to be lengthened to $1.17T_c$ (i.e., by 17%) to ensure no timing violation for 10 years. Although it seems that the 2-cycle critical path delay degrades to 2.14 T_c in our optimization, those paths still can finish computation within 2 clock cycles, which won't affect the correctness of VLD. More precisely, it indicates we sufficiently make use of those non-critical 2-cycle paths to reduce the delay of critical 1-cycle paths. If our optimization methodology is performed with the criticality of 1-cycle paths ignored, 1-cycle critical paths will be slowed down even though 2-cycle paths can be sped up much more significantly. However, this is useless since the clock period should be determined by not only 2-cycle critical paths, but also 1-cycle critical paths. In this case, to ensure no timing violation for 10 years, T_c will need to be lengthened by 23%.

The results of this experiment demonstrate that the VLD circuits being optimized by our framework are more tolerant to timing variability, than the case of the original VLDs or the VLDs being optimized without considering 1-cycle critical paths. This points out the importance of considering criticality of both 1-cycle and 2-cycle paths. Failing to consider 1-cycle critical paths will mislead the optimization process and will finally result in even worse vulnerability to timing variability.

Now take an observation on Fig. 4(a), for the original design (black line), the 1-cycle paths will about to violate timing constraint after 1 year. With performance optimized considering both 1-cycle and 2-cycle critical paths (red line), 1-cycle and 2-cycle path will not easily violate timing constraint, lengthening the lifetime of VLD. Also, it can be seen that the 2-cycle critical path has a greater delay than the longest path delay. This proved that we could slow down non-critical 2-cycle paths for speeding up 1-cycle critical paths. For the optimization without considering 1-cycle critical paths (blue line), we can see that the delay of 2-cycle critical path has extremely reduced. However, the delay of 1-cycle critical path increases and is larger than the 1-cycle critical path delay of original design. This makes the design more easily violate timing constraint, which decreases the lifetime of design. However, unlike Fig. 4(a), the 2-cycle critical path delay of Fig. 4(b) is lower than the longest path delay. This indicates it is possible that the optimization decreases the delay of both 1-cycle critical path and 2-cycle critical path instead of increasing the delay of 2-cycle critical path. Such event happens when large amount of 1-cycle critical gates are passed through by 2-cycle paths.



(a) Circuit *s1488* (b) Circuit *apex3*

Fig. 4. Tolerance to timing variability

V. Conclusion

In this paper, we propose to analyze and optimize a VLD circuit. The criticality of 1-cycle and 2-cycle paths/gates in the VLD and its vulnerability to timing variability are explicitly addressed. Experimental results show that the proposed methodology changes the path delay distribution such that sufficient timing margins can be obtained, which implies a higher degree of tolerance to timing variability. On average, 11% extra timing margin can be obtained with only 4% area overhead. If a lifetime of 10 years is targeted, another 6-7% increases in clock period are needed, which is much less than those by applying pessimistic, naïve optimization.

References

[1] L. Benini *et al.*, "Telescopic units: a new paradigm for performance optimization of VLSI designs," *IEEE TCAD*, vol. 17, no. 3, pp. 220-232, March 1998.

[2] L. Benini *et al.*, "Automatic synthesis of large telescopic units based on near-minimum timed supersetting," *IEEE Trans. on Computers*, vol. 48, no. 8, pp. 769-779, Aug. 1999.

[3] Y.-S. Su *et al.*, "An efficient mechanism for performance optimization of variable-latency designs," in *Proc. of DAC*, pp. 976-981, June 2007.

[4] D. Baneres, J. Cortadella, and M. Kishinevsky, "Variable-latency design by function speculation," in *Proc. of DATE*, pp. 1704-1709, April 2009.

[5] Y.-S. Su *et al.*, "Performance optimization using variable-latency design style," *IEEE TVLSI*, vol. 19, no. 10, pp. 1874-1883, Oct. 2011.

[6] ——, International Technology Roadmap for Semiconductor, 2013.

[7] K. Agarwal and S. Nassif, "Characterizing process variation in nanometer CMOS," in *Proc. of DAC*, pp. 396-399, June 2007.

[8] S. Borkar, "Tackling variability and reliability challenges," *IEEE Design and Test of Computers*, vol. 23, no. 6, p. 520, June 2006.

[9] J. W. McPherson, "Reliability challenges for 45nm and beyond," in *Proc. of DAC*, pp. 176-181, July 2006.

[10] W. Wang *et al.*, "Statistical prediction of circuit aging under process variations," in *Proc. of CICC*, pp. 13-16, Sep. 2008.

[11] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: a new paradigm for arithmetic circuit design," in *Proc. of DATE*, pp. 1250-1255, March 2008.

[12] Y. Chen *et al.*, "Variable-latency adder (VL-Adder) designs for low power and NBTI tolerance," *IEEE TVLSI*, vol. 18, no. 11, pp. 1621-16242, Nov. 2010.

[13] I.-C. Lin, Y.-H. Cho, and Y.-M. Yang, "Aging-aware reliable multiplier design with adaptive hold logic," *IEEE TVLSI*, vol. 23, no. 3, pp. 544-556, March 2015.

[14] S. Gupta and S. S. Sapatnekar, "BTI-aware design using variable latency units," in *Proc. of ASP-DAC*, pp. 775-780, Jan. 2012.

[15] S. Gupta and S. S. Sapatnekar, "Variation-aware variable latency design," in *IEEE TVLSI*, vol. 22, no. 5, pp. 1106-1117, May 2014.

[16] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[17] X. Hu, Y. Shi, and R. Eberhart, "Recent advances in particle swarm," in *Congress on Evolutionary Computation*, pp. 90-97, June 2004.

[18] A. Johari *et. al*, "Logical circuit gate sizing using PSO guided by logical effort - an examination of the 12-stage ripple carry adder circuit," in *Proc. of Int'l Conf. on Computer Applications and Industrial Electronics*, pp. 61-66, Dec. 2011.

[19] W. Wang *et al.*, "The impact of NBTI effect on combinational circuit: modeling, simulation, and analysis," *IEEE TVLSI*, vol. 18, no. 2, pp. 173-183, Feb. 2010.

[20] W. Wang *et al.*, "An efficient method to identify critical gates under circuit aging," in *Proc. of ICCAD*, pp. 735-740, Nov. 2007.