Automated Design of Self-Adjusting Pipelines

Jieyi Long, and Seda Ogrenci Memik

Department of Electrical Engineering and Computer Science

Northwestern University, Evanston, IL 60208

{jlo198, seda} @ eecs.northwestern.edu

ABSTRACT

We propose a self-adjusting pipeline structure to enhance chip performance and robustness considering the effects of process variations. We achieve this by introducing delay sensors to monitor internal timing violations within a pipeline stage and variable clock skew buffers to adjust the timing of the pipeline stage based on the feedback from the delay sensors. Furthermore, we formulate the delay sensor insertion and variable clock skew configuration problem as a stochastic mixed-integer programming problem and propose a simulated-annealing based algorithm to solve it. A comparison between the designs with and without the self-adjusting enhancement reveals that, we are able to improve the average performance of a batch of chips by 9.5%.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault Tolerance

General Terms

Design, Performance, Reliability.

Keywords

Self-Adjusting, Delay Monitoring, Variable Clock Skews.

1. INTRODUCTION

New technologies and the complexity growth of the designs place a large burden on synthesis, simulation, and verification tools. It is becoming increasingly complicated to verify the correctness of execution of a design. As a result, circuits are designed conservatively, i.e., the designers assume the worst-case scenario and optimize the circuits for them. In addition to the complexity of guaranteeing correctness, this approach results in designs with sub-optimal performance. Besides the sheer complexity, another important problem faced by the designers is the intricacies of using smaller manufacturing technologies. As manufacturing technologies are scaled down, it becomes exponentially harder to verify and guarantee the correctness. Each component in the system can possibly affect the timing and operation of another component. Therefore, complete correctness cannot be verified by local simulations only. In addition, simulating components at newer technologies is becoming further challenging due to the emergence of various physical phenomena. Therefore, there is a need for novel methods to satisfy the performance requirements of next-generation high performance circuits.

DAC 2008, June 8–13, 2008, Anaheim, California, USA

In this paper, we present the self-adjusting pipeline architecture and the supporting design automation framework as a mean to overcome this important hurdle. Our approach is to shift the complexity of the simulation and verification to the higher levels by designing self-adjusting hardware, i.e. hardware that adapts to unexpected events and hence, to variation in performance parameters during execution. In this work, we achieve this by introducing a variable clock skew scheme to adjust the timing of pipeline stages dynamically. We show that the design process of these self-adjusting hardware structures can be automated. Particularly, we target the Delay Monitoring and Skew Buffer Insertion problem in order to determine the sensor locations and nominal delay of the skew buffer to maximize the average performance of a batch of chips. We formulate this problem as a stochastic mixed-integer programming problem and propose a simulated-annealing based algorithm for solving the problem.

The novelty of our work lies in the following aspects. Majority of existing statistical yield optimization techniques are static [1-4]. However, we propose a dynamic technique that enables robustness towards process variations during run-time. Through the use of our proposed self-adjusting design it is possible to intervene before the timing variation actually manifests itself as a violation. This is a fundamentally different paradigm than existing dynamic techniques, which operate on the detect, stall, and re-execute principle [5].

We have evaluated the effectiveness of our approach on the pipeline of a high performance DEC Alpha-like microprocessor. Each block has been characterized in terms of the impact of process variations (die-to-die, within die…) on the timing of the critical paths within the blocks. Using these parametric models we have performed the automated delay monitor and variable skew buffer insertion in these blocks. A comparison between the designs with and without the self-adjusting enhancement reveals that, we are able to improve the *batch performance* by 9.5%. Batch performance is the metric corresponding to the average performance of a large set of chips, which is commonly used to assess the yield of a batch after applying speed binning.

The remainder of this paper is organized as follows Section 2 provides an overview of related work. In Section 3, the model of process variations and a brief review of the hardware design of the built-in delay sensor are presented, followed by the detailed discussion of the self-adjusting pipeline architecture. We introduce our systematic framework for designing a self-adjusting pipeline in Section 4. Our experimental evaluation is presented in Section 5. We conclude with a summary of our contributions and findings in Section 6.

2. RELATED WORK

The increasing impact of process and environmental variations on circuit timing has motivated several self-adjusting architectures. Chakraborty et al. considered the problem of guaranteeing the

timing correctness of sequential circuits under on-chip temperature variation. The idea is to insert tuneable delay buffers into the clock tree, which can be adjusted on-the-fly [6, 7]. Long et al. [8] solved the same problem by introducing SACTA, a self-adjusting clock tree architecture leveraging the specially designed skew buffers to achieve adaptability. The skew buffers provide proper clock skews which lengthen or shorten the effective clock cycle time for each pipeline stage depending on the local temperature level.

Ernst et al. [5] proposed a circuit-level timing error detection/correction scheme. Each pipeline register lying on the critical path is coupled with a shadow register which is controlled by a delayed clock. The results stored in the pipeline registers and their corresponding shadow registers are compared and if they do not agree it is denoted as a timing error. In such a case, the pipeline is stalled and the erroneous operation is re-executed. One specific technique employed by this scheme is called *dynamic retiming*. The idea is to create intentional clock skews such that those pipeline stages that repeatedly experience timing errors are assigned longer intervals. The skews can be changed dynamically within a certain range if the execution time of the pipeline stage changes due to environmental fluctuations.

Our scheme is fundamentally different as we introduce the delay sensors to detect the timing violations in internal nodes of the pipeline stages and we can reconfigure the adjustable skew buffers before the current operation is completed. Hence, we do not need to stall the pipeline and re-execute a failed operation.

3. SELF-ADJUSTING PIPELINE ARCHITECTURE

In this section, we will first analyze the impact of process variations on circuit timing, which motivated our self-monitoring/adjusting scheme. Then, we will discuss the delay monitoring elements and the self-adjusting pipeline architecture.

3.1 Impact of Process Variations on Timing

In deep sub-micron technology, circuit parameters such as gate-oxide thickness, channel length, etc. are statistical parameters rather than fixed values. This phenomenon is called process variation and can be categorized into die-to-die (D2D) and with-in-die (WID) variations. D2D variation refers to the variation in process parameters across dies and wafers, while WID variation is the variations on the circuit parameters across different regions in a single die. WID variation is considered to be the dominant factor in the deep-submicron regime.

Since the CMOS gate delay is a function of the above mentioned circuit parameters, process variations have a direct impact on circuit timing. However, analysis shows that different pipeline stages can have different levels of susceptibility to process variations. According to the FMAX model introduced by Bowman et al. [9] the criticality of a pipeline stage is determined by the number of independent *potential* critical paths (N_{cp}) of the stage and the critical path logic depth (L_{cp}). A potential critical path refers to a path having a nominal delay close to the clock cycle time of the circuit. The pipeline stages having larger N_{cp}/L_{cp} ratios tend to be more vulnerable to process variations, and thereby having a higher probability of containing a critical path. For instance, a recent study on the effects of process variations on microprocessors has shown that the L1 cache has almost 60% chances of containing the critical path [10]. Another study also reports that process variations can have an uneven impact on the timing of different pipeline stages of a sequential circuit [1].

This disparity motivated us develop a self-adjusting enhancement customized for the most vulnerable pipeline stage in a given design to monitor the internal timing violations. This customization is realized through the use of an automated design process. In Section 3.2, we first elaborate on how delay monitoring can be performed. Next, we present our proposed self-adjusting pipeline architecture in Section 3.3. Finally, we describe the automated design framework to realize an instance of such a self-adjusting pipeline in Section 4.

3.2 Delay Monitoring Elements

Ghosh et al. [11] proposed a low-overhead built-in delay sensor (BIDS) which is capable of detecting failures at the internal nodes of a circuit. The schematic of the delay sensor is depicted in Figure 1. A sawtooth waveform with the duration of the time period of the reference clock is generated. This signal is connected to a track-and-hold (TAH) circuit whose sampling switch is controlled by the observation node P. Suppose node P is expected to make a transition from "1" to "0". When node P is "1", the switch is on and the output tracks the sawtooth waveform. As P makes a falling transition, the TAH switch is turned off and the voltage of the output capacitor of the TAH holds its value V_{TAH}. The comparator then compares V_{TAH} with V_{REF}, which is a measure of the maximum tolerable delay T_{MAX} of node P. If V_{TAH} is higher than V_{REF}, the output of the comparator becomes "1", indicating a timing violation at node P.

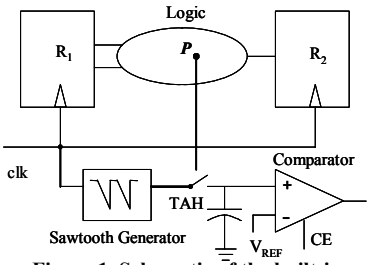

Figure 1. Schematic of the built-in delay sensor

The above described BIDS can be calibrated for process variations [11]. Due to this process-variation-tolerant feature, in the following discussions, we will omit the impact of process variations on the delay sensors.

3.3 Self-Adjusting Pipeline Architecture

Figure 2 shows a portion of a sequential circuit. It consists of two pipeline stages. Assume that the first stage is significantly more likely to contain the critical path of the overall system compared with the subsequent pipeline stage. We refer to the first stage as the *vulnerable stage*. The bold lines denote the potential critical paths. The N_{cp}/L_{cp} ratio of the first stage (which is equal to 3/2) is much larger than that of the second stage (which is equal to 1/4). Figure 3 depicts the enhanced pipeline. First, we insert a set of BIDS to cover all potential critical paths in the first stage. The delay sensors are represented by ⊕ in Figure 3. Second, we add one adjustable skew buffer at the CLK pin of register R_2. This buffer can provide one of two possible skew values at any given time. The structure of the adjustable skew buffer is shown in Figure 4. It consists of a δ-buffer (with nominal delay of μ_δ, and variance σ_δ^2), a MUX and an OR gate. All the outputs of the delay sensors are OR-ed and then used to control the MUX. If an internal timing violation is detected by any sensor the output of

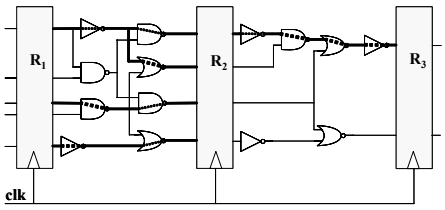

Figure 2. Two consecutive pipeline stages with the first stage being the most vulnerable stage to process variation

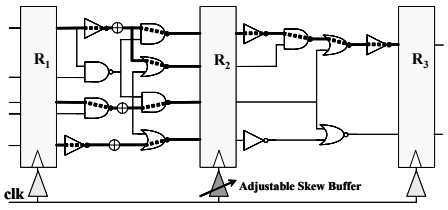

Figure 3. The pipeline enhanced with built-in delay sensors and adjustable skew buffers

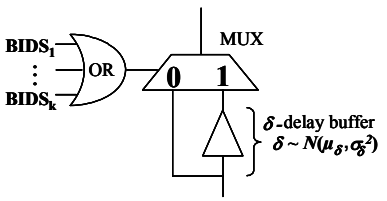

Figure 4. Circuit design of the adjustable skew buffer

the OR gate becomes "1". In that case, the delay of the adjustable skew buffer is equal to the sum of the MUX delay and the delay of the δ-buffer. Otherwise, when the output of the OR gate is "0", indicating that no local timing violation is detected, the delay of the adjustable skew buffer becomes equal to the delay of the MUX only. We also add delay elements whose delay is equal to that of the MUX before the CLK ports of R_1 and R_3 so that in case no internal timing violation is detected, the circuit will operate the same as the original circuit. If an internal timing violation is detected within the vulnerable stage, the effective cycle time of this stage will be increased by the delay of the δ-buffer.

Our scheme is essentially speculating on timing violations based upon monitoring of a set of internal nodes within the vulnerable stage. The detection of a timing violation at an internal node does not necessarily mean that the pipeline stage will fail without reconfiguration of the skew buffer. Hence, our definition of timing violation is probabilistic. Unlike our scheme existing self-adjusting architectures that apply dynamic retiming check timing violation at the end of each pipeline stage. Clearly, they can determine a timing violation with certainty. However, they will not have the opportunity to intervene on time to avoid it. They detect a violation after the fact. On the other hand, the internal delay monitoring elements signal the possibility of timing violation with a certain probability. Similarly, the adjustable skew buffer will have a certain probability of success in avoiding the timing violation, since its own behavior is also subject to parametric variation. Our choice of the locations of the delay monitors and the specifications of the adjustable buffer (nominal delay value of the δ-delay buffer) will determine the effectiveness of the self-adjusting architecture. A given configuration will result in a certain probability of "yield". This is what we are trying to maximize. This necessitates a systematic treatment of the associated optimization problem. We propose an automated design framework to derive the configuration of the self-adjusting architecture for a given pipeline topology and a set of parametric variations for timing. The benefit of our approach is that we are able to intervene for a statistically significant fraction of the internal timing violations and prevent them from turning into actual timing/computation errors at the pipeline stage boundary. Thereby, performance overhead of error detect/stall/re-execute based techniques is avoided.

4. SYSTEMATIC FRAMEWORK FOR DESIGN OF SELF-ADJUSTING PIPELINES

In this section, we propose a systematic framework to design a self-adjusting pipeline. First, for a given pipeline design the impact of timing variation on the latencies pipeline stages and the contribution of each stage to the overall critical path are determined. We make use of the FMAX model [9] described in Section 3.1 to accomplish this. Next, we identify the most vulnerable pipeline stage. This is the stage whose potential critical paths have the highest likelihood to contribute to the overall critical path. If the immediately adjacent pipeline stage has a significantly smaller probability of dictating the critical path of the overall pipeline, then it can be coupled with the vulnerable stage. In fact, our approach can be easily extended to handle two pipeline stages, which are not immediately adjacent. Then, we transform the vulnerable stage into a self-adjusting one. In practice, immediately adjacent pairs of pipeline stages can be found in current microprocessors. We will discuss specific examples in Section 5. This transformation can be applied to multiple pairs of pipeline stages iteratively.

The design of a self-adjusting pipeline stage essentially involves determining the locations of the delay sensors within the vulnerable stage and the nominal delay of the δ-buffer residing within the adjustable skew buffer. Our goal is to maximize the average performance of a batch of chips.

4.1 Problem Formulation

To mitigate the impacts of process variation on a batch of chips, the technique of *speed-binning* is commonly used. After manufacturing, each chip is tested over a spectrum of frequencies until a timing failure is observed. The *yield* of a bin is defined as the percentage of chips that fall into this bin.

Das et al. [10] introduced the *Batch Performance* (BP) metric to evaluate the average performance of a batch of chips. Assuming n bins with frequencies $f_1 < f_2 < \ldots < f_n$, each having yields $y_1, y_2, \ldots, y_n$, the Batch Performance is defined as

$$BP = \sum_{i=1}^{n} f_i \cdot y_i \qquad (1)$$

We use the BP metric as the optimization objective of our problem.

Problem 1 (Automated Delay Sensor Insertion and Clock Skew Buffer Configuration): *Given* 1) a two-stage balanced pipeline comprised of one highly process variations susceptible stage, following by one relatively robust stage, 2) the maximum tolerable delay for each internal node of the pipeline (derived trough statistical timing analysis), *Determine* the delay sensor locations and the nominal delay (μ_δ) of the adjustable skew buffer such that the *BP* metric of the pipeline is maximized.

We model the two-stage pipeline as a directed acyclic graph (DAG) *G*. Figure 5 depicts the DAG for the pipeline shown in Figure 2. The CMOS gates are mapped to vertices and the

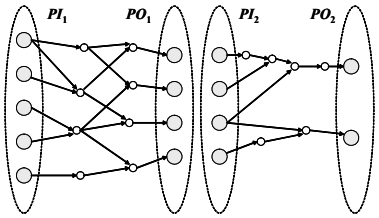

Figure 5. Directed acyclic graph model of the pipeline shown in Figure 2

interconnects between gates are mapped to the directed edges. We use notation V and E to represent the set of vertices and the set of edges, respectively. The registers are mapped to a special set of vertices called the *primary input*/*output* vertices (gray vertices in Figure 5). The sets of primary input/output vertices of the i^{th} stage will be denoted by PI_i/PO_i. Each register between the two stages is split into one primary input and one primary output vertex with no edge between them. We call a path on G a *primary path* if it starts from a primary input vertex and ends at a primary output vertex. Due to process variations, multiple of these primary paths may be potential critical paths. Obviously, to capture all possible timing failures in the first stage, each potential critical path in this stage should be covered by a delay sensor, i.e., there should be a delay sensor located on one edge on this path. On the other hand, to keep the number of sensors minimal, each potential critical path in the first stage should contain no more than one sensor. Also, we do not need to place any sensor in the second stage, as we only focus on applying the enhancement to the most vulnerable stage. These requirements pose some constraints on the sensor locations. To formulate these constraints, we denote the sub-graph of G consisting of all the potential critical paths by G_{pcp}. We then assign a binary decision variable x_i for each vertex v_i of G_{pcp}. The decision variables specify the locations of the delay sensors:

a sensor is located on directed edge (v_i, v_j), iff $x_i - x_j = 1$.

To formulate the requirement that each path contains exactly one sensor, we have the following constraints:

$$\begin{aligned} x_i - x_j \geq 0, \text{ for all } (v_i, v_j) \in E \\ x_p = 1, \text{ for all } v_p \in PI_1 \textbf{ and } x_p = 0, \text{ for all } v_p \in PI_2 \\ x_q = 0, \text{ for all } v_q \in PO_1 \textbf{ and } v_q \in PO_2 \end{aligned} \quad (2)$$

For a given primary path in the first stage, as the decision variables for the primary input and output vertices are 1 and 0, respectively, there should be at least one edge (v_i, v_j) on this path with $x_i - x_j = 1$, indicating that the primary path contains at least one sensor. On the other hand, since we require $x_i - x_j \geq 0$ for all edges along this path, the decision variables of the vertices along the primary path can "toggle" at most once. This means there is at most one sensor on the primary path. Since the decision variables of all the primary input/output vertices in the second stage are 0, none of the primary paths in the second stage contains any delay sensor, as $x_i - x_j = 0$ for each edge (v_i, v_j) in the second stage.

Another set of constraints are concerned with the fact that the reconfiguration of the adjustable skew buffer takes a certain amount of time. In fact, as shown in Figure 4, the output signals of the delay sensors need to propagate through an OR gate (may be an OR tree depending on the implementation) before reaching the control port of the MUX. If we place the delay sensors too close to the primary output vertices (although this would have enabled the most accurate assessment of total delay in the stage), we cannot guarantee that the skew adjustment takes into effect on time. To fulfill this requirement, we do not allow a sensor to be inserted after a vertex if the $(\mu - 3\sigma)$ delay between this vertex and any primary output is less than $\mu_{OR} + 3\sigma_{OR}$. μ_{OR} and σ_{OR}^2 are the mean and variance of the delay of the OR gate, respectively. We perform a preprocessing on G_{pcp} to identify such vertices. We call the set formed by these vertices as the *forbidden vertex set* and denote it by V_F. To ensure that no BIDS will be inserted after any vertex in V_F, we only need to require that $x_f = 0$, for each $v_f \in V_F$.

Now let us analyze the batch performance. Denoting the probability that a chip operates correctly at a frequency below f by $\mathbf{Pr}(f)$, given a frequency bin $[f_k, f_{k+1}]$, its yield is determined by

$$y_k = \mathbf{Pr}(f_{k+1}) - \mathbf{Pr}(f_k).$$

Therefore, the batch performance of the self-adjustable pipeline can be calculated as follows

$$BP = \sum_{k=1}^{n} f_k \cdot \left(\Pr\left(f_{k+1}\right) - \Pr\left(f_k\right)\right) \quad (3)$$

Let us first calculate the probability $\mathbf{Pr}(f)$. We associate each vertex v_i on G_{pcp} with a random variable D_i which describes the accumulative delay at the output of the gate that v_i corresponds to. D_i can be obtained using a statistical timing analysis technique. Although it is generally is hard to determine D_i if we consider the spatial correlation and path reconvergence, it can be approximated by a Gaussian distribution as it is commonly practiced in statistical timing analysis techniques.

Denoting the maximal tolerable delay at v_i by D_i^m, the sensor system will raise an alert if there is a sensor detecting a timing violation. Defining the following function

$$\alpha(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}, \quad (4)$$

A sensor located at edge (v_i, v_j) detects an error if random variable $\alpha((x_i - x_j)(D_i - D_i^m))$ is equal to 1. Therefore, the sensor system raises an alert when the random variable R_1

$$R_1 = \sum_{(v_i, v_j)} \alpha\left(\left(x_i - x_j\right) \cdot \left(D_i - D_i^m\right)\right) \quad (5)$$

is larger than 0. As we have described in Section 3.3, upon the detection of a timing violation, the pipeline automatically generates a clock skew by amount of δ. Due to process variations δ itself is not a fixed value. We model δ as a Gaussian variable with mean value μ_δ and variance σ_δ^2, where the value of σ_δ^2 depends on the technology. Therefore, μ_δ alone is sufficient to represent the distribution of δ. After reconfiguration, the pipeline will operate correctly if the delay of each primary output of the first stage is smaller than $(1/f + \delta)$. Also, the delay of the primary outputs of the second stage should be smaller than $(1/f - \delta)$. Let us introduce a second random variable R_2

$$R_2 = \left(\prod_{v_k \in \mathrm{PO}_1} \alpha\left(-D_k + 1/f + \delta\right)\right) \cdot \left(\prod_{v_k \in \mathrm{PO}_2} \alpha\left(-D_k + 1/f - \delta\right)\right) \quad (6)$$

R_2 will be equal to 1 if and only if for each primary output vertex v_k, $\alpha(-D_k + 1/f - \delta)$ is equal to 1, representing the situation that no primary path actually violates the timing constraint. Therefore, the pipeline meets the timing constraint after reconfiguration if and only if $R_2 = 1$. On the other hand, if the sensor system does not detect any timing error, i.e., if $R_1 = 0$, the pipeline operates correctly if the following random variable R_3 is equal to 1.

$$R_3 = \left(\prod_{v_k \in \mathrm{PO}_1} \alpha\left(-D_k + 1/f\right)\right) \cdot \left(\prod_{v_k \in \mathrm{PO}_2} \alpha\left(-D_k + 1/f\right)\right) \quad (7)$$

Note that $R_1 > 0$ and $R_1 = 0$ are mutually exclusive events. Therefore, the probability that the self-adjusting pipeline operates correctly at a frequency below f is given by

$$\mathbf{Pr}(f) = \mathbf{Pr}(R_1 > 0, R_2 = 1) + \mathbf{Pr}(R_1 = 0, R_3 = 1). \quad (8)$$

The batch performance can then be determined by substituting (8) into (3). Although Expression (3) involves $\{f_i\}$, these frequency values are predetermined. The decision variables that can be adjusted are the sensor locations, given by $\{x_i\}$ subject to the constraint set (2) and the mean of the skew buffer delay μ_δ.

According to the above discussion, Problem 1 can be formulated as a *Stochastic Mixed-Integer Programming* (SMIP) problem:

$$\begin{aligned} \max \quad & BP\left(x_1, x_2, ..., x_{|V|}, \mu_\delta\right) \\ \text{s.t.} \quad & x_i - x_j \geq 0, \text{ for each } (v_i, v_j) \in E \\ & x_p = 1, \text{ for each } v_p \in PI_1 \\ & x_q = 0, \text{ for each } v_q \in PO_1 \\ & x_p = 0, \text{ for each } v_p \in PI_2 \\ & x_q = 0 \text{for each } v_q \in PO_2 \\ & x_f = 0, \text{ for each } v_f \in V_F \\ & x_i \in \{0, 1\}, \text{ for each } v_i \in V \\ & \mu_\delta \in \mathbf{R}^+ \end{aligned} \quad (9)$$

4.2 Simulated-Annealing Based Optimization

Generally speaking, stochastic mixed-integer programming problems are extremely hard [12]. The objective function stated above cannot be expressed analytically. Random variables R_1 and R_2 are correlated as well as R_1 and R_3. This makes the computation of $\mathbf{Pr}(R_1 > 0, R_2 > 0)$ and $\mathbf{Pr}(R_1 = 0, R_3 > 0)$ extremely complicated. Therefore, we use Monte Carlo simulation (on 10,000 randomly generated instances of pipelines) to evaluate the cost function for a given set of $\{x_i\}$ and a given mean skew buffer delay μ_δ. We propose to use a simulated-annealing based technique to solve the stochastic mixed-integer programming problem described in last section.

Some observations help us accelerate the algorithm. First, once we fix the values of the decision variables, since random variable R_3 defined by (8) does not involve δ, $\mathbf{Pr}(R_1 = 0, R_3 = 1)$ in (8) will not vary with μ_δ. Second, as random variable D_k takes only positive value, if δ is larger than $1/f$, R_2 will be 0, which means that $\mathbf{Pr}(R_1 > 0, R_2 = 1) = 0$. Therefore, μ_δ is actually confined within a relatively small range. Due to the difficulty of obtaining the analytical expression of BP, we use Monte Carlo simulation to evaluate BP. Utilizing the fact that the possible range of μ_δ is strictly limited, local search techniques for black-box optimization such as blind random search or adaptive sampling search can be adopted efficiently to find the μ_δ value, which maximizes BP (could be sub-optimal) for a given set of decision variables [13].

Solution Space: We call a decision variable set $X = \{x_1, x_2, \ldots, x_{|V|}\}$ a *legal decision variable set* if it satisfies constraint (2). According to the above discussion, for each legal decision variable set X we can use local search techniques to obtain the most suitable μ_δ for X. Therefore, we only need the simulated-annealing process to set the values of the decision variables.

Initial Solution: An obvious legal initial solution is

$$x_i = \begin{cases} 1, & v_i \in PI_1 \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

Other legal solutions, which can be obtained easily by solving inequality set (2), can also be used.

Solution Perturbation: We define two types of moves to perturb the current solution:

***$M_0(x_j, X)$* (0→1 toggle)**: i) keep the value of x_i for each $i \neq j$; ii) change the value of x_j from 0 to 1 if 1) $x_j = 0$ and $x_i = 1$ for each $(v_i, v_j) \in E$, and 2) $x_j \notin V_F$;

***$M_1(x_i, X)$* (1→0 toggle)**: i) keep the value of x_j for each $j \neq i$; ii) change the value of x_i from 1 to 0 if $x_j = 1$ and $x_j = 0$ for each $(v_i, v_j) \in E$;

Each move transforms a legal decision variable set to another legal set. To efficiently implement the perturbation, we maintain two sets X_0 and X_1. X_0 contains candidates for move M_0 defined as $X_0 = \{x_j \mid x_j = 0 \wedge x_i = 1 \text{ for each } (v_i, v_j) \in E\}$. X_1 is defined similarly: $X_1 = \{x_i \mid x_i = 1 \vee x_j = 0 \text{ for each } (v_i, v_j) \in E\}$.

After a move, we need to update X_0 and X_1. The rules for updating X_0 and X_1 are as follows:

After M_0: $X_0 = X_0 + \{x_k \mid \text{for each } (v_j, v_k) \in E\} - \{x_j\}$, $X_1 = X_1 + \{x_j\}$;

After M_1: $X_0 = X_0 + \{x_i\}$, $X_1 = X_1 + \{x_k \mid \text{for each } (v_k, v_i) \in E\} - \{x_i\}$;

Figure 6 shows the pseudo code of the simulated-annealing based BIDS Insertion and Clock Skew Configuration algorithm. To determine the initial temperature, we perform a sequence of random moves and calculate Δ_{BP}^{avg}, the average cost changes for all downhill moves. Then, the initial temperature is chosen such that $\exp(\Delta_{BP}^{avg} / T) = P$, where P represents the initial probability of accepting downhill moves and is very close to 1. At each temperature, a number of trials are attempted until either we make N downhill moves, or the total number of moves exceeds $2N$, where N is an increasing function of $|V|$, the number of vertices. When we exit from the inner loop, the temperature is reduced by a

```
Algorithm SA_BIDS_CSC_Insertion {
  T = Δ_BP^avg / ln P; // initial temperature
  X = {x_i | i = 1, 2, …, |V|}, x_i = 1 iff v_i ∈ PI_1; // initial solution
  X_best = X;
  X_0 = Φ; // initial candidates for M_0
  X_1 = {x_j | v_i ∈ PI_1 for each (v_i, v_j) ∈ E}; // Initial candidates for M_1
  Do {
    MT = downhill = reject = 0;
    Do {
      Randomly select x_i from X_0 ∪ X_1;
      X_new = M_0(x_i, X) if x_i ∈ X_0, M_1(x_i, X) otherwise;
      MT = MT + 1;
      Local search for μ_δ that maximizes BP(X_new, μ_δ);
      Δ_BP = BP(X_new, μ_δ) − BP(X, μ_δ);
      If ((Δ_BP > 0) or (random < exp(Δ_BP / T)) {
        If (Δ_BP < 0) downhill = downhill + 1;
        X = X_new; // accept the new solution
        Update X_0 and X_1; // accept the new solution
        If (BP(X, μ_δ) > BP(X_best, μ_δ)) X_best = X;
      } Else {
        reject = reject + 1;
      }
    } While ((downhill < N) and (MT < 2N));
    T = λT;
  } While ((reject / MT < 0.95) and (T > ε));
}
```

Figure 6. Pseudo code of the simulated-annealing based algorithm

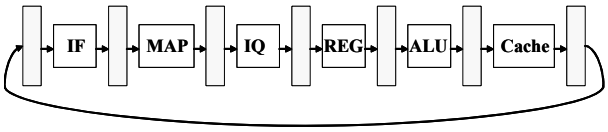

Figure 7. A DEC Alpha-like 6-stage pipeline

fixed ratio λ, which is set to 0.85 in our implementation. The entire algorithm terminates when the number of accepted moves becomes too small ($\leq$ 5% of total number of moves made), or when the temperature becomes too low.

5. EXPERIMENTAL RESULTS

We have tested our scheme on a DEC Alpha-like 6-stage pipeline. The pipeline is depicted in Figure 7. It is a 4-way processor with 64KB Level 1 instruction and data caches each. The pipeline latencies have been balanced such that the nominal latency of each stage is 400ps (corresponding to 2.5GHz clock frequency).

It has been reported that at 45nm technology, L1 data cache has the highest probability (58.9%) of containing a critical path in a pipeline similar to our target 6-stage pipeline [10]. Furthermore, this probability is significantly lower (less than 2%) for the immediately adjacent Instruction Fetch stage [10]. Therefore, we evaluated the effectiveness of our proposed solution by applying it onto these two stages. We generated the DAG model of the L1 data cache and the Instruction Fetch unit. The cache is a special structure which contains analog circuitry and memory cells. Obviously, the BIDS is only capable of monitoring digital waveforms. Therefore, the sensors cannot be located arbitrarily in the block. We have isolated the digital portion comprised of input inverters, decoder, and output selection logic. We created a DAG representation of this portion. Then, we performed one pass of static timing analysis on this model to identify the potential critical paths and construct the graph model G_{pcp} that consists of these paths. Our simulated-annealing based optimization algorithm is then applied on G_{pcp} to determine μ_δ and the sensor insertion points, maximizing the batch performance.

We assume six frequency bins: B_1 = [2.00, 2.05], B_2 = [2.05, 2.10], B_3 = [2.10, 2.15], B_4 = [2.15, 2.20], B_5 = [2.20, 2.25], B_6 = [2.25, 2.30], where the unit is GHz. Figure 8 shows the results for speed-binning where the *x*-axis represents the frequency bins and the *y*-axis represents the yields for each bin. The light and dark bars show the yields of the original and enhanced self-adjusting pipelines across the frequency bins, respectively. It can be seen that the average frequency shifts right (i.e., towards a higher value) after we enhance the pipeline with a set of BIDS and the adjustable skew buffers. We calculate the batch performance based on the speed-binning results. The batch performance value for the self-adjusting pipeline is 2.178GHz, while the same value for the original pipeline is 1.989GHz, indicating a 9.5% improvement. This result denotes the yield improvement for two specific stages. However, the overall yield distribution for the entire microprocessor is expected to be similar. The main reason for that is the fact that the L1 cache dictates the critical path of the overall microprocessor pipeline 58.9% of the time. Finally, note that our systematic framework can be applied to any given sequential circuit. For different processor architectures and technology and process parameters other pipeline stages can be targeted.

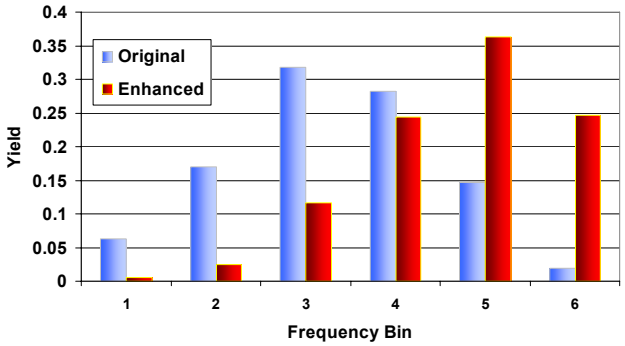

Figure 8. Speed-binning results for the original and enhanced pipeline with self-adjusting L1 cache stage

6. CONCLUSIONS

In this paper, we have proposed a self-adjusting pipeline architecture to enhance system performance and reliability. We employ built-in delay sensors to monitor the internal timing violations and variable skew buffers to adjust circuit timing. We further propose a systematic framework to automatically determine the delay sensor placement and the skew buffer configuration. Experimental results on a microprocessor pipeline reveal that we can enhance the batch performance by 9.5%.

7. ACKNOWLEDGMENTS

This work was partially supported by NSF under NSF Grant # CNS-0546305 and 0830-350-J205.

8. REFERENCES

1. Datta, A., et al. *Statiscal Modeling of Pipeline Delay and Design of Pipeline under Process Variation to Enhance Yield in sub-100nm Technologies*. in *Design, Automation and Test in Europe*. 2005.

2. Davoodi, A. and A. Srivastava. *Variability Driven Gate Sizing for Binning Yield Optimization*. in *Design Automation Conference*. 2006.

3. Orshansky, M., S. Nassif, and D. Boning, *Design for Manufacturability and Statistical Design*. 2007: Springer US.

4. Sinha, D., N. Shenoy, and H. Zhou, *Statistical Timing Yield Optimization by Gate Sizing.* IEEE Trans. on Very Large Scale Integrated (VLSI) Systems, 2006. **14**(10): p. 1140-1146.

5.Ernst, D., et al. *Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation*. in *Int. Sump. on Microarchitecture*. 2003.

6.Chakraborty, A. and K. Duraisami. *Dynamic Thermal Clock Skew Compensation using Tunable Delay Buffers*. in *Int. Symp. on Low Power Electronic Design*. 2006.

7.Duraisami, K., et al. *Design Exploration of a Thermal Management Unit for Dynamic Control of Temperature-Induced Clock Skew*. in *Int. Symp. on Circuits and Systems*. 2007.

8.Long, J., et al. *A Self-Adjusting Clock Tree Architecture to Cope with Temperature Variations*. in *Int. Conf. on Computer-Aided Design*. 2007.

9.Bowman, K.A., S.G. Duvall, and J.D. Meindl, *Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration.* IEEE Journal of Solid-State Circuits, 2002. **37**(2).

10.Das, A., et al. *Mitigating the Effects of Process Variations: Architectural Approaches for Improving Batch Performances*. in *Workshop on Architectural Support for Gigascale Integration*. 2007.

11.Ghosh, S., et al., *A Novel Delay Fault Testing Methodology using Low-Overhead Built-In Delay Sensor.* IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2007. **25**(12).

12.Ntaimo, L. and M.W. Tanner, *Computations with Disjunctive Cuts for Two-Stage Stochastic Mixed 0-1 Integer Programming* 2007, Stochastic Programming E-Print Series.

13.Kargupta, H. and D.E. Goldberg. *SEARCH, Blackbox Optimization, and Sample Complexity*. in *Foundation of Genetic Algorithm*. 1997.