Stochastic Error Rate Estimation for Adaptive Speed Control with Field Delay Testing

Shoichi Iizuka Masafumi Mizuno Dan Kuroda Masanori Hashimoto Takao Onoye
Dept. Information Systems Engineering, Osaka University
{iizuka.shoichi, hasimoto}@ist.osaka-u.ac.jp

ABSTRACT

This paper proposes a stochastic framework for error rate estimation that models adaptive speed control as a continuous-time Markov process and derives its transition rates using developed similarity database. The proposed framework is implemented for adaptive speed control systems based on timing error prediction and scan-test. Experimental results show that the proposed framework enabled 12 orders of magnitude faster MTTF estimation than ordinary logic simulation. The accuracy of MTTF estimation under random delay fluctuation is clarified through a comparison with logic simulation. The proposed estimation can contribute to design and validation of adaptive speed control systems with field delay testing.

1. INTRODUCTION

Device miniaturization due to technology scaling has made parametric performance variation more and more significant. Lower supply voltage makes circuits sensitive to environmental fluctuation, especially to supply voltage. Furthermore, aging effects, such as NBTI (negative bias temperature instability), HCI (hot carrier injection) and TDDB (time dependent dielectric breakdown), cause unexpected timing failures in field. A traditional approach to avoid timing failures due to above manufacturing variability, environmental fluctuation and aging is giving a timing margin in design time and production test. Such an overdesign must involve power, area and cost overhead, and/or performance loss. Nowadays, the performance loss easily spoils the advantage of technology scaling.

To overcome this problem, adaptive performance control in field is studied. Dynamic voltage and frequency scaling and body biasing are popular ways to control performance. Traditionally, such performance control is carried out so that no timing errors would happen in all the paths in a circuit. For example, performance tuning for aging-induced error mitigation [1] and concurrent timing testing [2] are presented. On the other hand, voltage over-scaling, which accepts rare potential timing errors, is actively studied to exploit the low activation probability of critical paths for aggressive power reduction [3–6]. The path activation probability heavily depends on the running program, and in some cases a dramatic power reduction can be achieved.

To implement adaptive performance control, we need to regularly evaluate the performance to check whether the current circuit performance is necessary and sufficient for given performance specifications. For most of digital systems, timing specification is the primary one, and it needs to be verified with online or offline delay testing. Online testing is concurrently carried out along ordinary functional operations using functional input vector patterns [3–6]. The online testing is classified into two groups; error detection [3, 5, 6] and error prediction [4, 5]. Error detection is accompanied with error recovery mechanism, whereas error prediction tells whether timing errors will start to occur soon. On the other hand, in offline delay testing, which here includes pseudo online testing that is carried out during idle time, CUT (circuit under test) is disconnected with surrounding circuits and functional and/or structural tests are performed with test patterns prepared beforehand [2].

A fundamental problem of adaptive speed control is that the possibility of timing error occurrence cannot be completely reduced to zero, since, for example, a sudden delay increase larger than expectation can induce a timing error without error detection or before error prediction. Similarly, offline delay testing may miss the error because delay testing is carried out with a certain time interval. Researchers working for any types of adaptive speed control claim that by tuning some design parameters the possibility of timing error occurrence can be reduced to almost zero and the mean time to failure (MTTF) over years can be easily attained with some overhead. For example, delay testing should be more frequently carried out, or earlier error prediction should be enforced.

However, it is challenging to quantitatively estimate such a long MTTF and extremely low probability of error occurrence. A naive simulation is totally impractical since one year operation of a processor, for example, includes 3×10^{16} cycles, and to get 10-k samples, 3×10^{20} cycles must be simulated. With a logic simulator processing 3×10^{3} cycles per second, it takes 3×10^{9} years, and hence another approach instead of naive simulation is indispensable.

This paper presents a stochastic estimation framework aiming at analyzing MTTF of adaptively performance-controlled circuits. The proposed framework models the adaptive speed control under dynamic delay variation as a continuous-time Markov process, and stochastically estimates MTTF. Given a matrix of transition rates between states, the MTTF can be calculated via matrix computations and its calculation time is independent of how long MTTFs are and how rarely the timing error happens, which is an excellent property for evaluating a long-MTTF circuit operation. To construct the transition rate matrix, we developed a similarity database and a direct derivation method of the matrix using the database. Thanks to this development, the proposed framework computes MTTF 10^{12} times faster than a logic simulator in a test case.

The rest of this paper is organized as follows. Section 2 models the adaptive speed control as a continuous-time Markov process, and derives a closed-form MTTF expression. Section 3 presents a similarity database for efficiently obtaining the rate of state transition. Section 4 describes the implementation of the proposed stochastic framework for two adaptive speed control systems. Experimental results are shown in Section 5, and conclusions are given in Section 6.

978-1-4799-1071-7/13/$31.00

2. CONTINUOUS-TIME MARKOV PROCESS MODELING OF ADAPTIVE SPEED CONTROL

2.1 Overview and State Assignment

We first model adaptive speed control under dynamic delay variation as a continuous-time Markov process. Markov process is a stochastic process having a Markov property that the next state is determined by only the current state and is independent of the previous states. Especially, continuous-time Markov process is a special Markov process whose time parameter is continuous [7][8].

We assign states as follows. The circuit delay temporally fluctuates due to unintentional temperature change, power supply noise and aging. By sensing such temporal delay fluctuation with online/offline delay testing, the performance of the circuit under adaptive speed control is intentionally tuned by supply voltage scaling and/or body biasing. We define states in Markov process such that each state is associated with a pair of unintentional delay variation and levels of intentional speed control. We often prepare several discrete values for supply voltage scaling and body biasing. On the other hand, the unintentional delay variation is continuous in nature, but for the model simplicity, we discretize the unintentional delay variation into several representative values. We call these states as normal states. On the other hand, we add one more failure state meaning that a timing error happened in the past.

Figure 1 illustrates an example of state assignment and a series of state transitions falling into the failure state. In this example, the circuit starts to operate at speed control level of 0 with 0ps delay fluctuation. Then, both the speed control level and delay fluctuation are varying dynamically. At a certain time, a timing error happens at speed control level of 0 with 30ps delay fluctuation, and the state falls into the failure state.

In a continuous-time Markov process, transition rate of going from state i to state j, $q_{i,j}$, which will be formally defined in the next section, is the key parameter that characterizes the process behavior. Given a matrix of the transition rates, we can obtain closed-form expressions of state probability as a function of time t. This means that once the matrix of transition rates is given, the MTTF computation can be carried out with a constant time, and the computation time is independent of the timing error rate and MTTF of the circuit under evaluation. Note that the above computation is applicable to any types of adaptive speed control, since the state assignment explained above is independent of the implementation of adaptive speed control.

As a related work, it should be noted that [4, 9] used discrete-time Markov chain for estimating MTTF. However, this work was tailored for adaptive speed control with error prediction and other adaptive speed control systems cannot be analyzed. In addition, the conventional method aimed to estimate MTTF under a specific static operating condition and hence dynamic variations cannot be considered in MTTF estimation. On the other hand, the proposed framework in this paper estimates MTTF for various types of adaptive speed control systems under dynamically fluctuating operating condition. Due to this difference, each speed level was selected as a state in [4, 9], which is different from that shown in Fig. 1. Furthermore, [4, 9] required path delay distributions and their path activation probabilities, which is a significant bottleneck in terms of analyzable circuit size, and thereby only adders were analyzed in [4, 9]. In contrast, in this paper, a processor on which various programs are running is analyzed by exploiting the similarity database developed in this work.

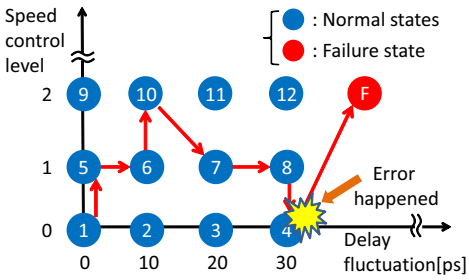

Figure 1: An Example of State Assignment and Transition.

2.2 Deriving Closed-Form State Probability Expressions from Transition Rate Matrix

This section derives closed-form state probability expressions. The state transition probability $p_{i,j}(s,t)$ is defined as a probability that a system being in state i at time s will stay in state j at time t.

$$p_{i,j}(s,t) = P(X(t) = j | X(s) = i). \tag{1}$$

In case of a stationary Markov process, $p_{i,j}(s,t)$ can be simply expressed as $p_{i,j}(t)$.

The transition rate of leaving state i, q_i, is defined as

$$q_i = \lim_{h \to 0+} \frac{1 - p_{i,i}(h)}{h} = -\frac{dp_{i,i}(0)}{dt}, \tag{2}$$

where as $h \to 0+$, $p_{i,i}(h) \to 1$. When the number of state is finite, $q_i < \infty$ holds. The transition rate of going from state i to state j $(i \neq j)$ is defined as

$$q_{i,j} = \lim_{h \to 0+} \frac{p_{i,j}(h)}{h} = \frac{dp_{i,j}(0)}{dt}, \tag{3}$$

where $q_{i,j} < \infty$ always holds. Q-matrix, which consists of $-q_i$ and $q_{i,j}$, is expressed by

$$\mathbf{Q} = \begin{bmatrix} -q_1 & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & -q_2 & \cdots & q_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N,1} & q_{N,2} & \cdots & -q_N \end{bmatrix}, \tag{4}$$

where N is the number of states and $\sum_{j \neq i} q_{i,j}(t) = -q_i$ holds.

Once Q-matrix is given, the state transition probability as a function of time t can be analytically derived by solving a Kolmogorov forward differential equation below [7].

$$\frac{dp_{ij}(t)}{dt} = -q_j p_{ij}(t) + \sum_{\nu \neq j} p_{i\nu} q_{\nu j}. \tag{5}$$

Let us introduce a way to solve this differential equation using matrix computation. Letting λ_i denote the i-th eigenvector of Q-matrix and $\mathbf{u}_i$ denote its corresponding eigenvector, we define the matrices below.

$$\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_N], \tag{6}$$

$$\mathbf{\Lambda}(t) = \begin{pmatrix} e^{\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 t} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & e^{\lambda_N t} \end{pmatrix}. \tag{7}$$

Using $\mathbf{U}$ and $\mathbf{\Lambda}$(t), the matrix of state transition probability is

expressed by

$$\mathbf{P}(t) = \begin{bmatrix} p_{1,1}(t) & p_{1,2}(t) & \cdots & p_{1,N}(t) \\ p_{2,1}(t) & p_{2,2}(t) & \cdots & p_{2,N}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p_{N,1}(t) & p_{N,2}(t) & \cdots & p_{N,N}(t) \end{bmatrix} = \mathbf{U}\mathbf{\Lambda}(t)\mathbf{U}^{-1}. \quad (8)$$

By specifying the initial state ($init$), the state probability being at state i at time t, $p_{init,i}(t)$ can be computed using the analytic expression in Eq. (8).

Once we obtain $\mathbf{P}(t)$, MTTF can be calculated.

$$MTTF = \int_0^\infty t \cdot \frac{dp_{init,fail}(t)}{dt} dt, \quad (9)$$

where $p_{init,fail}(t)$ is the state transition probability from the initial state to failure state. In addition, since we know the probabilities of being at each state, other performance metrics, such as power dissipation, can be computed using information of average power dissipation at each state.

3. SIMILARITY DATABASE

Section 2 explained that once the transition rate of going from state i to state j, $q_{i,j}$, becomes available, we can obtain the exact closed-form probability expression of timing failure as a function of time. This section presents a basic idea of similarity database tailored for $q_{i,j}$ extraction, which is applicable to any types of adaptive speed control systems. The detailed implementation for a particular system will be discussed in Section 4.

The basic idea of similarity database is simulating the circuit behavior by referring representative similar operations which were simulated beforehand and of which results are stored in the similarity database. The similarity database includes various circuit operations at available speed levels in the range of possible delay fluctuation. The similarity of circuit operation could be defined using various clues, for example, usage of arithmetic units, memory access patterns, program execution times, program types and so on in addition to current speed level and delay fluctuation. In embedded systems, the number of programs to be executed is limited and the program itself is a major factor to define similarity. This similarity database is constructed for each circuit and technology.

In the database design, we need to determine the time unit for each database record. Now focusing on a long time-term hour-to-year operation, a clock cycle is obviously too short as the time unit of the record. The time for one program execution could be a reasonable time unit, since the program is a major factor to define the similarity. This paper adopts the time for one program execution as the time unit of the similarity database. Note that the difference in the execution times for each program is taken into account in $q_{i,j}$ computation of Section 4.3. Instead, we can choose one function execution or one instruction execution as the time unit for each database record if more appropriate. Besides, we need to pay attention to the possible amount of delay fluctuation within the time unit. If the delay fluctuation within the time unit would be much larger than the interval of delay fluctuation in the database records, the dynamic delay fluctuation cannot be well reproduced as a stochastic process. In this case, the time unit needs to be shortened so that the amount of delay fluctuation within the time unit becomes shorter than the interval of delay fluctuation in the database.

Then, given a set of keys, for example, the current speed level, the current delay fluctuation and the program to execute, the similarity database quickly returns some features influential to the behavior of the adaptive speed control. The features include such as occurrence of timing error, error prediction results, results of path delay tests, and so on, in addition to ordinary performance metrics, such as power consumption. We need to select features for each system of adaptive speed control, and will illustrate two examples in Section 4.2.

Using this similarity database, we know the occurrence of events that correspond to the transition from state i to state j, such as speed level transition, during a program execution. If the execution frequency of each program and a model of dynamic delay fluctuation are given, we can calculate $q_{i,j}$ by referring the similarity database. This algorithm will be shown in Section 4.3.

4. IMPLEMENTING PROPOSED FRAMEWORK FOR PARTICULAR ADAPTIVE SPEED CONTROL SYSTEMS

This section explains two implementations of the proposed stochastic framework taking on-line testing based adaptation and off-line testing based adaptation as examples. While the concept of the stochastic framework is applicable to any designs, adaptive speed controls and delay variation models, the implementation details depend on them.

4.1 Assumed Adaptive Speed Control

This subsection explains two adaptive speed control systems that we use for experiments in Section 5.

4.1.1 On-line Test Based Adaptation using TEP-FF

Figure 2 shows a circuit that adaptively controls the speed and power dissipation using a warning signal generated by a timing-error predictive (TEP) FF [4], and the timing error rate of this run-time adaptive speed control is analyzed in this paper as one of applications of the proposed estimation framework. The TEP-FF consists of a normal flip-flop, a delay buffer and a comparator (XOR gate). When the timing margin is gradually decreasing, a timing error occurs at the TEP-FF before the main FF captures a wrong value due to the delay buffer, which enables us to know that the timing margin of the main FF is not large enough. A warning signal is generated to predict the timing errors, and it is monitored during a specified period. Note that timing errors are predicted, not detected, which is a distinct difference from Razor [3]. Once a warning signal is observed, the circuit is controlled to speed up, in other words, the circuit delay is reduced by voltage scaling and/or body biasing. Note that clock frequency is fixed throughout this paper. If no warning signals are observed during the monitoring period, the circuit is slowed down for power reduction. This proactive speed control overcomes the variation of the timing margin which is different chip by chip and varies depending on operating condition and aging.

Even when the TEP-FF is well configured to generate the warning signal, the error occurrence cannot be reduced to zero. This is because when critical paths are not activated for a long time in the circuit operation, the circuit might be slowed down excessively. If a critical path is activated in this condition, a timing error happens.

To reduce the error occurrence, we can tune the following design parameters; the number of TEP-FFs, locations where TEP-FFs should be inserted, delay time of the delay buffer in each TEP-FF, monitoring period and fineness of the speed control [4, 9].

4.1.2 Off-line Test Based Adaptation using Scan-Test

We next explain an adaptive speed control system that repeatedly performs scan-based delay test in idle times of the circuit. While the circuit is idle, test-patterns that were prepared beforehand and stored in an internal or external memory are loaded and

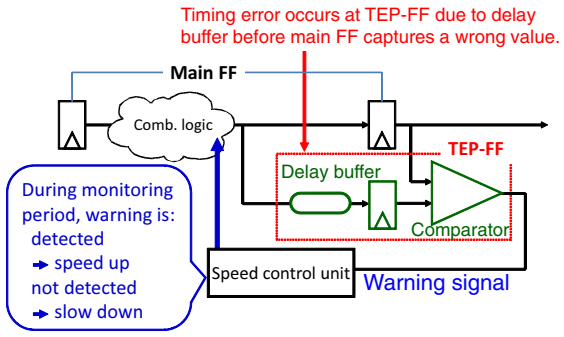

Figure 2: Run-time Adaptive Speed Control with TEP-FF.

it is checked if the circuit includes timing-violating paths or not. When a timing-violating path is detected, the minimum speed level that includes no timing-violating paths is selected for the operation in the following. Otherwise, the speed level is decremented. This scan-based test has higher freedom of applicable test-patterns, and hence accurate error detection, in other words, lower missing rate of timing-violating paths can be expected.

Here, there are two strategies for scan-test execution. One strategy forces the circuit to be idle with a fixed time interval, which can guarantee the time interval between the delay tests. This strategy is helpful to make the timing error rate predictable in addition to mitigating the error rate. A drawback is the performance degradation due to the testing, and in some real-time systems, this strategy could be difficult to adopt. The other strategy is to perform off-line tests only in true idle time. While the performance degradation does not arise, the test interval is less predictable and consequently the error rate tends to be higher. We in this paper assume the first strategy having the fixed time interval of delay testing.

In this off-line test based adaptation, the test interval is a key parameter to determine the rate of timing error occurrence. If it is set to be long, the delay fluctuation in the duration of successive delay tests is likely to be large enough to cause timing violations. For mitigating the timing errors, the test interval should be short. On the other hand, frequent tests induce performance overhead and degrade the system throughput. To reduce the error occurrence while coping with the overhead, we need to carefully tune the test interval.

4.2 Similarity Database Implementation

We first explain what features should be considered for the adaptive speed control with error prediction. In this system, the error prediction result determines the levels of speed control, and hence it should be included in the similarity database as features. Of course, it should be included whether a timing error occurred or not. Other possible features are execution time needed and power dissipation.

Figure 3 exemplifies the database records. The keys include program, input data for program, speed level and the amount of delay fluctuation. The data to output are the existence of timing errors, the power dissipation, the execution time of the program, and the time at which a warning signal for timing error arises.

Besides, the adaptive speed control with timing error prediction has some key design parameters; where TEP-FFs should be inserted, how long delay elements in TEP-FFs should be, etc., as listed in the previous section. The similarity database should cover those design parameters to make it possible to explore the design space of the adaptive speed control system. This means that the database should know which TEP-FF gives a warning signal with various lengths of delay element. With this information, we do not

Key				Data					
Program	Data	Speed Level	Delay	Error Time	Exec Time	Power	Error Prediction (Each Buffer Delay)		Time
P1	D1	L1	0	0	10000	1	0	…	0
			10	0	10000	1	0	…	100
			⋮	⋮					
		L2	0	0	10000	0.8	0	…	100
			10	5000	10000	0.8	2500	…	1000
			⋮	⋮					
		⋮	⋮	⋮					
	D2	L1	0	6000	12000	1	0	…	5000
			10	5000	12000	1	3000	…	1000
			⋮	⋮					
		L2	0	0	12000	0.8	0	…	0
			10	0	12000	0.8	0	…	6000
			⋮	⋮					
		⋮		⋮					
⋮									

Figure 3: An illustration of similarity database for adaptive speed control with TEP-FF.

have to reconstruct the similarity database for different configurations of TEP-FFs. This information can be extracted with a single run of logic simulation below.

We prepare a gate-level net-list with a SDF (standard delay format) file that includes gate delay information calculated depending on the selected speed level and gate delay fluctuation. In this net-list, all the FFs are accompanied with several TEP-FFs with different delay elements. We also prepare input vectors depending on the program and data to process. In addition, to know whether timing error occurs or not, RTL description is prepared. The net-list above and RTL description are simulated simultaneously and the FF values stored are compared at every cycle. By simulating this with a logic simulator, features needed for simulating the adaptive speed control with timing error prediction can be obtained. For constructing the similarity database, we repeatedly execute logic simulations with each program, input data, speed level and delay, and store each execution time, error prediction time and other needed information in the database.

For the similarity database for off-line test based adaptation, we need to prepare another database which tells us whether the path delay test detects timing-violating paths at each speed level with each amount of delay fluctuation. With this test database and the database similar to that for on-line test based adaptation, we can immediately reproduce the circuit behavior both for the program execution and scan-test.

4.3 Computation of State Transition Rate

This subsection explains how to construct Q-matrix using the similarity database for the adaptive speed control systems explained in Section 4.1.

State transitions occur in three cases; (1) a timing error arises, (2) the speed level changes and (3) the amount of delay fluctuation varies. We thus need to compute the probabilities for these three cases. The computation of Q-matrix consists of two steps. The first step computes the preliminary state transition rates for each record in the database, i.e. for a program executed at a speed level with an amount of delay fluctuation. On the other hand, the probability of each record being executed in the actual circuit behavior is different depending on, for example, the processor usage. The second step calculates the overall state transition rates taking into account the processor usage and the dynamic delay fluctuation.

4.3.1 On-line Test Based Adaptation using TEP-FF

Algorithm 1 Step 1 of Q-matrix computation for k-th record whose key is (Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$). (on-line test based adaptation using TEP-FF).

1: (ErrorTime, ExecTime, Power, ErrorPredictionTime) ← DB(Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$)
2: **if** ErrorTime $\neq 0$ **then**
3: $q'_{State_k,fail}[k] \leftarrow 1/\text{ErrorTime}$
4: **else if** ErrorPredictionTime $\neq 0$ **then**
5: $q'_{State_k,\mathbf{dest}(\text{SpeedLevel}_k++,State_k)}[k] \leftarrow 1/\text{ErrorPredictionTime}$
6: **else**
7: $q'_{State_k,\mathbf{dest}(\text{SpeedLevel}_k--,State_k)}[k] \leftarrow 1/\text{MonitorTime}$
8: **end if**

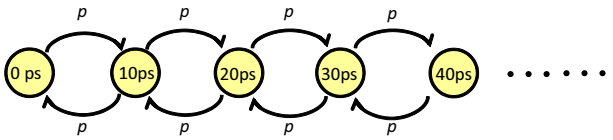

Figure 4: A Stochastic Delay Fluctuation Model.

The first step calculates the preliminary rate of state transition for each similarity database record ($q'_{i,j}[k]$). Algorithm 1 describes the computation procedure for the k-th record whose key is (Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$). The function DB returns the data consisting of ErrorTime, ExecTime, Power and ErrorPrediction-Time. Here, these variables in Algorithms 1-4 correspond to key and data in Fig. 3. MonitorTime denotes the length of the monitoring period. The function dest(A, B) returns the state to which the current state of B moves when an event of A occurs. These time variables are represented by the numbers of clock cycles. Besides, at line 3, 1/ErrorTime is calculated. Please recall that the rate of state transition is defined as Eq. (3). Due to line 2, during the time of ErrorTime(=h), a timing error happens, which means $p_{i,j}$(ErrorTime)=1. Thus, 1/ErrorTime is calculated. Similar computations are executed at lines 5 and 7.

The second step computes $q_{i,j}$ in Q-matrix by Algorithm 2. **S** represents the universe set of states, and $\mathbf{E}_i$ is a subset of **E**, where **E** is the universe set of database records. $\mathbf{E}_i$ includes the records whose key matches with SpeedLevel$_i$ and Delay$_i$. In order to consider the probability of each program execution and the difference in execution time, $weight$ is computed, where the function Prob(Program, Data) returns the probability of which Program with Data is executed. For all the elements of $\mathbf{E}_i$, i.e. for all the sets of Program and Data, the weighted average of state transition rate is calculated (line 5 to 11).

The state transition due to dynamic delay fluctuation is considered at lines 13 and 14. Here, a stochastic fluctuation model expressed as a Markov chain in Fig. 4, which will be used in experiments later, is assumed. With the probability of p, the amount of delay fluctuation changes. It should be noted that in Algorithm 2, the state transitions in terms of speed level and delay fluctuation are exclusively considered for simplifying the explanation. The state transitions due to speed level and delay fluctuation rarely arise simultaneously, and hence Algorithm 2 works in most cases. On the other hand, the computation taking into account the simultaneous transitions is possible without any technical difficulties while its explanation is omitted in the paper.

4.3.2 Off-line Test Based Adaptation using Scan-Test

Algorithm 2 Step 2 of Q-matrix computation (on-line test based adaptation using TEP-FF).

1: **for** $i \in \mathbf{S}$ **do**
2: **for** $j \in \mathbf{S}(i \neq j)$ **do**
3: $qa \leftarrow 0$
4: $qb \leftarrow 0$
5: **for** $k \in \mathbf{E_i}$ **do**
6: (ErrorTime, ExecTime, Power, ErrorPredictionTime) ← DB(Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$)
7: $weight \leftarrow \text{ExecTime} \times \text{Prob(Program}_k\text{, Data}_k\text{)}$
8: $qa \leftarrow qa + weight \times q'_{i,j}[k]$
9: $qb \leftarrow qb + weight$
10: **end for**
11: $q_{i,j} \leftarrow qa/qb$
12: **end for**
13: $q_{i,\mathbf{dest}(\text{Delay}_i++,i)} \leftarrow p/\text{ExecTime}$
14: $q_{i,\mathbf{dest}(\text{Delay}_i--,i)} \leftarrow p/\text{ExecTime}$
15: **end for**

Algorithm 3 Step 1 of Q-matrix computation for k-th record whose key is (Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$). (off-line test based adaptation using scan-test).

1: (ErrorTime, ExecTime, Power) ← DB$_1$(Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$)
2: **if** ErrorTime $\neq 0$ **then**
3: $q'_{State_k,fail}[k] \leftarrow 1/\text{ErrorTime}$
4: **end if**

The first step for the off-line test based adaptation using scan-test is described in Algorithm 3. Algorithm 3 is close to Algorithm 1, and computations related to error prediction are removed.

Algorithm 4 describes the second step. The procedure (line 1 to 14) is the same with that of Algorithm 2. At line 15, the database which stores scan-test results is referred, where it is assumed that the same scan-test is performed independent of speed level and delay fluctuation for simplicity in this paper, while various test patterns can be stored in the database. If a timing violation is detected (line 16), the minimum speed level at which no timing violations happen is searched by the function FindSpeedLevel (line 17). The transition rate to the state corresponding to this minimum speed level is computed at line 18, where TestInterval is the time interval between scan-tests. If no timing violations are detected, the speed level is decremented and its corresponding computation is performed at line 20.

5. EXPERIMENTS

This section shows experimental results. We first demonstrate that the proposed estimation framework can estimate MTTF of the adaptive speed control with timing error prediction 12 orders of magnitude faster than the conventional logic simulation. We next discuss the estimation accuracy. Finally some analysis examples are presented.

5.1 Experimental Setup

5.1.1 Target Circuit

In this work, the adaptive speed control is applied to MIPS R3000 microprocessor. R3000 is a 32-bit RISC microprocessor and implemented with five pipeline stages. The processor was designed such that RTL hardware description was synthesized by a commercial logic synthesizer with a 65nm industrial standard cell

Algorithm 4 Step 2 of Q-matrix computation (off-line test based adaptation using scan-test).

for $i \in \mathbf{S}$ **do**
  for $j \in \mathbf{S}(i \neq j)$ **do**
    $qa \leftarrow 0$
    $qb \leftarrow 0$
    for $k \in \mathbf{E_i}$ **do**
      (ErrorTime, ExecTime, Power) $\leftarrow$ DB_1(Program$_k$, Data$_k$, SpeedLevel$_k$, Delay$_k$)
      $weight \leftarrow$ ExecTime$\times$ Prob(Program$_k$, Data$_k$)
      $qa \leftarrow qa + weight \times q'_{i,j}[k]$
      $qb \leftarrow qb + weight$
    end for
    $q_{i,j} \leftarrow qa/qb$
  end for
  $q_{i,\mathbf{dest}(\mathrm{Delay}_i++,i)} \leftarrow p/$ExecTime
  $q_{i,\mathbf{dest}(\mathrm{Delay}_i--,i)} \leftarrow p/$ExecTime
  (TimingViolation) $\leftarrow$ DB_2(SpeedLevel$_k$, Delay$_k$)
  if TimingViolation = yes **then**
    $vlv \leftarrow \mathrm{FindSpeedLevel}(\mathrm{Delay}_i)$
    $q_{i,\mathbf{dest}(vlv,i)} \leftarrow 1/$ TestInterval
  else
    $q_{i,\mathbf{dest}(SpeedLevel_i--,i)} \leftarrow 1/$TestInterval
  end if
end for

library. The number of standard cells is 6,813. The maximum clock frequency at 1.2V and 25°C is 147MHz, which corresponds to the critical path of 6.8ns.

5.1.2 Similarity Database Construction

In constructing similarity database, we attached TEP-FFs having six different delays (100ps, 300ps, 1ns, 2ns, 3ns, 4ns) in parallel to all the FFs in R3000, and extracted the warning signals and presence of timing error as features from the logic simulation results. Here, these seven delay values are just an example, and the number of delay values can be increased without time overhead in terms of similarity database construction, as explained in Section 4.2. We selected four benchmark programs (CRC32, SHA1, Dijkstra and Quicksort) from MIBenchmark [11] and 30 sets of input data for each program. The database quality, such as the number of input data sets, is thought to affect the estimation accuracy, and the impact of input data will be experimentally evaluated later. The database for scan-test was constructed using patterns for path delay tests generated by a commercial ATPG tool [12]. Launch on capture (LoC) scheme is adopted. Ten speed levels, i.e. ten supply voltages (1.2V, 1.1V, 1.0V, 0.90V, 0.85V, 0.80V, 0.75V, 0.70V, 0.65 and 0.60V) were prepared. For simplicity, all the cells have the same delay variation at each supply voltage, while any technical limitation is not given by the proposed framework. As for dynamic delay variation per gate due to, such as, environmental fluctuation and aging, 0 to 260 ps delay increases with 10 ps step were evaluated. With this database setup, the maximum number of states that can be analyzed with the proposed framework is 10 (speed levels) $\times$ 27 (delay fluctuation) + 1 (failure) = 271, and it was adopted for the experiments.

In the experiments, the programs to execute on the processor are limited to four mentioned above. Thus, we select a record in the similarity database which matches the same program, the same speed level and the same delay variation and has the processing data with the minimum hamming distance (CRC32, SHA1), with similar network depth (Dijkstra), or with the similar initial sort quality (Quicksort) to the data to process now.

Table 1: Comparison of Simulation Throughput.

Simulation methods	Throughput [cycle/s]	Throughput Relative
Ordinary logic	1.5×10^3	1
Similarity-based	1.9×10^8	1.3×10^5

5.1.3 Implementation

The proposed estimation framework is implemented with MATLAB, C++ and ruby script language. The matrix computation in Section 2.2 is carried out by MATLAB. The similarity-based simulator, which will be explained later, is implemented with C++, and some text processing is executed with ruby.

5.2 CPU Time Evaluation

We here evaluate the speed-up of the proposed similarity-based computation over ordinary logic simulation. The adaptive speed control system based on TEP-FF is analyzed in this subsection.

For a comparison, we also implemented a simulator so-called "similarity-based simulator" in addition to the similarity-based stochastic computation explained in Section 4.3. The similarity-based simulator simulates a time series circuit behavior by successively referring the similarity database. We can obtain a TTF (time to failure) by simulating the circuit behavior until a timing error happens, and then we repeat this TTF evaluation in Monte Carlo manner to obtain the statistics of TTF, such as MTTF. Table 1 lists the simulation throughputs of ordinary logic and similarity-based simulations. These simulations were carried out on a computer with CentOS5, Intel Xeon X5680 processor and 96GB memory. We can see that the similarity-based simulator achieved 10^5 times higher simulation throughput. However, this throughput improvement is not large enough to estimate long MTTFs, which will be shown in the next experiment. While the similarity-based simulator can reduce estimation time of MTTF from the logic simulation, the estimation time is still proportional to the MTTF.

We next evaluated the CPU times needed to estimate MTTF by three methods; ordinary logic simulation, similarity-based simulation, and the proposed stochastic framework. Table 2 lists the CPU time of the proposed method and the estimated CPU times of ordinary logic and similarity-based simulations. As an example, we evaluated the MTTF in case that p in the delay fluctuation model of Fig. 4 is set to 2.5×10^{-7}, which corresponds to the average state occupation time of 1,000 seconds. Delay value of TEP-FF was set to 1ns, all TEP-FFs were enabled, and the monitoring time was 100,000 cycles (0.68 ms). Under this setting, the MTTF is estimated to be 8.13 $\times 10^{12}$ clock cycles, which corresponds to 75 hours. Besides, such a long MTTF can be evaluated by neither ordinary logic simulation nor similarity-based simulation, which is the motivation of this work. We therefore estimated the necessary CPU times by MTTF [cycles] $\times$ 10,000 [runs] $\div$ throughput [cycle/second] in Table 1 for these two simulations.

Table 2 clearly shows that the MTTF estimation by ordinary logic and similarity-based simulations is absolutely infeasible. On the other hand, the proposed method estimated MTTF in 37 seconds. The speedups of the proposed method over ordinary logic simulation and similarity-based simulation only are 12 orders of magnitude and 7 orders of magnitude, respectively. Remind that the CPU times of the ordinary logic simulation and the similarity-based simulation is roughly proportional to TTF, whereas the CPU time of the proposed stochastic framework is independent of TTF. Therefore, the speedup becomes more significant as the MTTF of

Table 2: Comparison in CPU time for MTTF Evaluation (Only a single processor was used).

Methods	CPU time	
	Actual	Relative
Ordinary logic sim.	1.7×10^6 years †	1.4×10^{12}
Similarity-based sim.	13 years †	1.1×10^7
Proposed overall (Matrix computation) (Transition rate computation)	37 seconds (32 seconds) (5 seconds)	1

†: estimated from the simulation throughputs in Table 1.

the target system becomes longer.

We have to mention the time needed to construct the similarity database. This similarity database needs to be constructed once even though design parameters of the adaptive speed control are varied. We therefore this construction time is not included in Table 2. Besides, 2.6 days were necessary for the database construction using eight processors in this setup.

5.3 Accuracy Evaluation

The MTTF is estimated from Q-matrix, and the Q-matrix comes from the records of the similarity database. Then, the size of similarity database affects the accuracy of MTTF estimation. In addition, the proposed probability computation explained in Section 4.3 stochastically handles the elapse of the monitoring time in on-line test based adaptation and that of the test interval in off-line test based one. This stochastic treatment might cause an estimation error.

We assessed the estimation error due to these two factors through a comparison with logic simulation. We used adaptive speed control with TEP-FFs in which the delay time of the delay buffer was 100ps and the monitor time was 100k cycles. We estimated MTTFs in two cases; p values in the stochastic delay fluctuation model in Fig. 4 were 0.5 (Case 1) and 0.2 (Case 2), respectively. The numbers of TTF evaluations using logic simulation for Case 1 and Case 2 are 107 and 217. The size of similarity database is the same as previously explained (4 programs and 30 input data per each program). In logic simulation, on the other hand, a number of input data which are not included in the database are given. The MTTFs estimated by the proposed method and logic simulation are shown in Table 3. We can see that the estimated MTTFs are very close in both Case 1 and Case 2, and the differences are only 1.9% and 2.9%. Due to the extremely long CPU time of logic simulation,

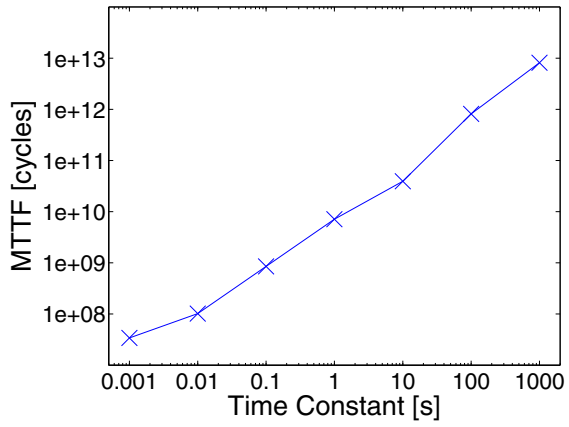

Figure 5: MTTF versus Time Constant of Delay Fluctuation.

Table 3: Accuracy comparison.

Method	MTTF [cycles]	
	Case 1	Case 2
Logic sim	2.10×10^6	2.40×10^6
Proposed	2.14×10^6	2.33×10^6

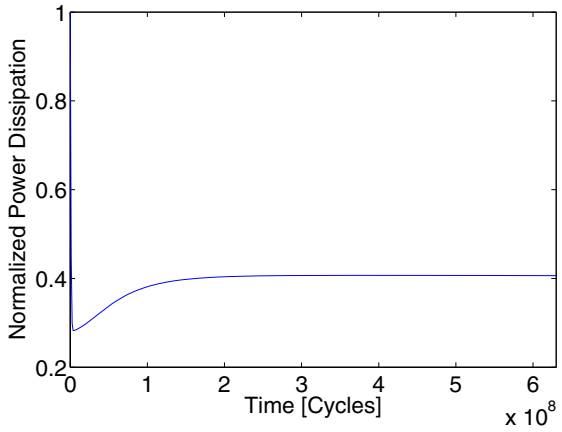

Figure 6: Temporal Variation of Average Power Dissipation.

only two cases are presented here, but this result could be one of evidences validating the proposed stochastic approach for MTTF estimation.

5.4 Analysis Example

Now, we can quickly estimate MTTFs of adaptive speed control systems of which operation parameters are changed. This subsection shows some examples illustrating the dependence of MTTF on the operation parameters.

Figure 5 plots MTTF when p in the delay random fluctuation model of Fig. 4 is varied from 0.25 to 2.5×10^{-7}, which corresponds to the average state occupation time of 0.001 to 1000 second. This average state occupation time is here called as time constant of delay fluctuation and selected as the horizontal axis of Fig. 5. We can see that MTTFs vary depending on the time constant, and slower delay fluctuation makes MTTFs longer. This is reasonable, since less frequent delay increase makes the error probability lower.

Figure 6 illustrates the temporal variation of average power dissipation. The time constant of delay fluctuation was 0.01 second. The initial state was the state at 1.2V with 0ps delay increase, and then the power dissipation decreased first. Then the voltage was once over-scaled, and then the power got saturated to be constant.

Figure 7 shows MTTF when the delay time of the delay buffer in each TEP-FF is varied from 100ps to 4ns. We can see that MTTF becomes significantly short when the delay is less than 300ps. On the other hand, when the delay is over 3000ps, MTTF does not improve. This result suggests that the delay shorter than 300ps cannot well predict the timing errors and the MTTF of 10×10^9 cycles cannot be obtained only adjusting the delay buffer. Figure 8 plots MTTF when the monitoring period of adaptive speed control is swept from 100k to 100M cycles. As the monitoring period becomes longer, MTTF also gets longer.

It should be noted that these analyses can be executed without reconstructing the similarity database. By using this property, we can explore and design an adaptive speed control system satisfying

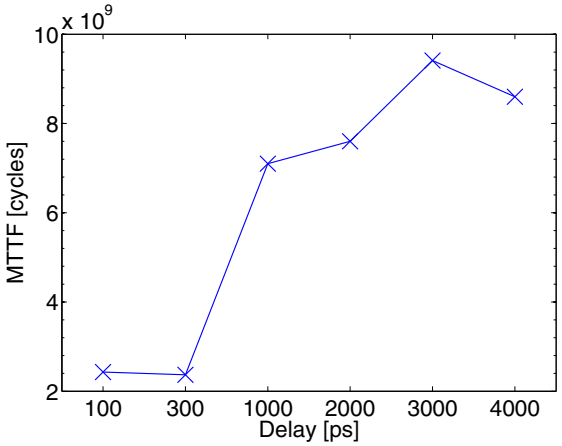

Figure 7: MTTF versus Delay of Delay Buffer in TEP-FF.

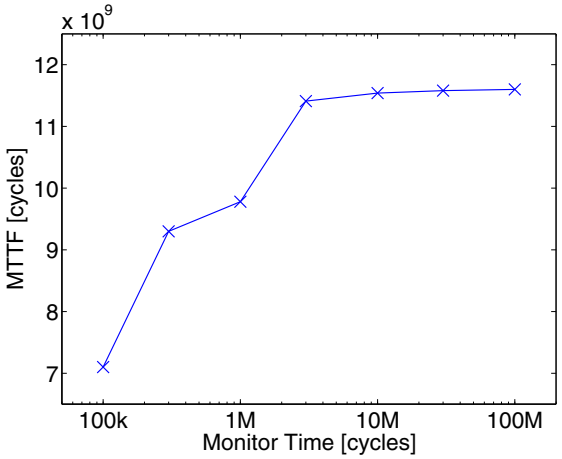

Figure 8: MTTF versus Monitoring Period.

given specifications.

Finally, we demonstrate MTTF for the adaptive speed control based on off-line scan-test. Figure 9 plots MTTF when the interval of scan-test is changed from 1k to 100k cycles. We can see that more frequent scan-test contributes to longer MTTF and this becomes significant in the case that the scan-test interval is over 10k cycles. The proposed framework quantitatively tells us the MTTF tendency, which is helpful for system design and validation.

6. CONCLUSION

In this work, we proposed a stochastic MTTF estimation framework for adaptive speed control system. The modeling of adaptive speed control as a continuous-time Markov process enabled MTTF-independent computation time for MTTF estimation. To acquire entries of Q-matrix rapidly, we devised a similarity database that refers to features of a similar simulation result stored in the database. Thanks to the stochastic modeling and the similarity database, it becomes possible to estimate long MTTF in 37 seconds with a single CPU, which is 12 orders of magnitude faster than ordinary logic simulation. This enablement of long MTTF evaluation contributes to quantitatively designing and validating adaptive speed control system with field delay testing.

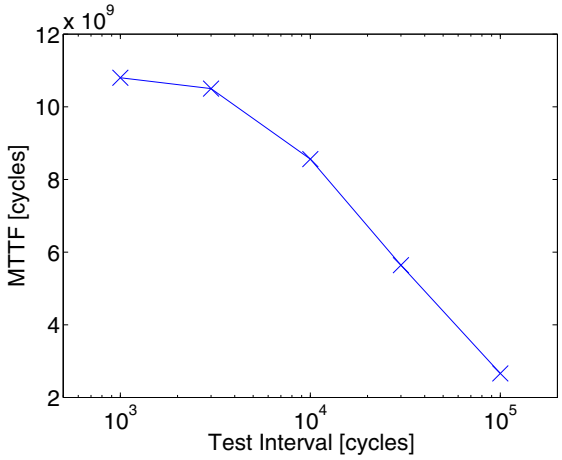

Figure 9: MTTF versus Scan-test Interval.

Acknowledgements

This work was supported by Semiconductor Technology Academic Research Center (STARC) and the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

7. REFERENCES

[1] M. Agarwal, B. C. Paul, Z. Ming, and S. Mitra, "Circuit Failure Prediction and Its Application to Transistor Aging," in *Proc. VTS*, pp.277–286, 2007.

[2] Y. Li, S. Makar, and S. Mitra, "CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns," in *Proc. DATE*, pp.885–890, 2008.

[3] S. Das, et.al., "A self-tuning DVS processor using delay-error detection and correction," *IEEE JSSC*, vol.41, pp.792–804, Apr. 2006.

[4] H. Fuketa, M. Hashimoto, Y. Mitsuyama, and T. Onoye, "Adaptive Performance Compensation with In-Situ Timing Error Predictive Sensors for Subthreshold Circuits," *IEEE TVLSI*, vol. 20, no. 2, pp. 333–343, Feb. 2012.

[5] K. A. Bowman, et.al., J. W. Tschanz, S. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De, "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance," *IEEE JSSC*, Vol. 46 , No. 1, pp.194 –208, Jan. 2011.

[6] D. Blaauw, et.al., "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," in *ISSCC Dig.*, pp.400–401, 2008.

[7] A. Papoulis and S. U. Pillai, "Probability, Random Variables and Stochastic Process, Fourth Edition," McGraw-Hill Higher Education, 2002.

[8] J. R. Norris, "Markov Chains," Cambridge University Press, 1997.

[9] H. Fuketa, M. Hashimoto, Y. Mitsuyama, and T. Onoye, "Trade-Off Analysis between Timing Error Rate and Power Dissipation for Adaptive Speed Control with Timing Error Prediction," *IEICE Trans. Fundamentals*, vol. E92-A, no. 12, pp. 3094–3102, Dec. 2009.

[10] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K.S. Kim., "Robust system design with built-in soft-error resilience," Computer, Feb. 2005.

[11] M. R. Guthaus, et.al., "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Workshop on Workload Characterization*, 2001.

[12] Synopsys Inc:, TetraMAX ® ATPG User Guide, September. 2009.