CLIM: A Cross-Level Workload-Aware Timing Error Prediction Model for Functional Units

Xun Jiao, Abbas Rahimi, Yu Jiang, Jianguo Wang, Hamed Fatemi, Jose Pineda de Gyvez, *Fellow, IEEE*, and Rajesh K. Gupta, *Fellow, IEEE*

Abstract—Timing errors that are caused by the timing violations of sensitized circuit paths, have emerged as an important threat to the reliability of synchronous digital circuits. To protect circuits from these timing errors, designers typically use a conservative timing margin, which leads to operational inefficiency. Existing adaptive approaches reduce such conservative margins by predicting the timing errors in advance and adjusting the timing margin adaptively. However, these error prediction approaches overlook the impact of input workload (i.e., operands) on path sensitization, thereby resulting in a loss of accuracy. The diversity of input operands leads to complex path sensitization behaviors, making them hard to represent in timing error modeling. In this paper, we propose **CLIM**, a cross-level workload-aware timing error prediction model for functional units (FUs). **CLIM** predicts whether there are timing errors in FU at two levels: bit-level and value-level. At the bit level or value level, **CLIM** predicts each output bit or entire output value as one of two classes: {*timing correct*, *timing erroneous*} as a function of input workload and clock period, respectively. We apply supervised learning methods to construct **CLIM**, by using input operands, computation history and circuit toggling as input features, as well as outputs' timing classes as labels. These training data are collected from gate-level simulations (GLS) of post place-and-route designs in TSMC 45nm process. We evaluate **CLIM** prediction accuracy for various FUs and compare it with baseline models. On average, **CLIM** exhibits 95 percent prediction accuracy at value-level, 97 percent at bit-level, and executes at a rate 173X faster than GLS. We utilize **CLIM** to analyze the value-level and bit-level reliability of FUs under random and real-world application workloads. At value-level, **CLIM**-based reliability estimation is within 2.8 percent deviation on average of detailed GLS ground truth. At bit-level, we introduce the concept of *bit-level reliability specification* of error-tolerant applications and compare this with the **CLIM**-based bit-level reliability estimation. By comparison, **CLIM** will classify the application quality into two classes: {*acceptable*, *non-acceptable*}. On average, 97 percent application quality classification is consistent with GLS ground truth.

Index Terms—Error-tolerant design, timing error modeling, approximate computing, hardware reliability, path sensitization, input workload

1 Introduction

With the continuous scaling of CMOS technology, microelectronic integrated circuits are even more susceptible to *timing errors* caused by timing violations of sensitized paths, making them a notable threat to reliability. To protect circuits from timing errors, designers typically use conservative timing margins acting as guardbands, computed from a multi-corner worst-case analysis at design time through static timing analysis (STA). While safe, worst-case paths are never or rarely exercised, resulting in loss of performance. Increasing variability caused by process, voltage, temperature and aging (PVTA) in advanced processes further exacerbates this problem.

Attempting to reduce such performance loss, *better-than-worst-case* (BTWC) approaches have been explored. These approaches reduce timing margins by scaling frequency, and use recovery schemes to correct the timing errors caused by frequency overscaling [5], [10], [14], [16]. Although effective, such techniques could incur silicon overhead for online monitoring. Furthermore, these approaches incur performance penalties when correcting timing errors.

To avoid such overhead, a less intrusive adaptive approach has been proposed to predict and prevent timing errors by adaptively changing the clock period. Instruction-level models identify critical instructions by measuring their maximum delay and use this information to guide runtime adaptation [11], [34], [41]. Rahimi et al. proposed a timing error rate prediction model for functional units based on hardware PVTA variation information [30]. However, these works all assume a worst-case scenario for path sensitization that overlooks the effect of input operands, leading to pessimistic modeling. Actually, the same instruction or FU could exhibit a different delay under different input operands, resulting in different timing error rates (TERs) [41]. Unfortunately, due to the extremely large input space, incorporating input operands into timing error modeling becomes very difficult, if not impossible.

- *X. Jiao, J. Wang, and R. K. Gupta are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093. E-mail: {xujiao, csjgwang, gupta}@cs.ucsd.edu.*
- *A. Rahimi is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94704. E-mail: abbas@eecs.berkeley.edu.*
- *Y. Jiang is with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: jy1989@illinois.edu.*
- *H. Fatemi and J. Pineda are with NXP Semiconductors, Eindhoven 5656AE, The Netherlands. E-mail: {hamed.fatemi, jose.pineda.de.gyvez}@nxp.com.*

Manuscript received 18 Feb. 2017; revised 26 Sept. 2017; accepted 10 Oct. 2017. Date of publication 13 Dec. 2017; date of current version 16 May 2018. (Corresponding author: Yu Jiang.)
Recommended for acceptance by J. Henkel.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2783333

As the efforts to reduce conservative timing margins become more aggressive, designers are opening up to continuing operations even in the presence of timing errors. As of now, *approximate computing* has allowed occasional errors in system as long as it delivers an acceptable output quality [7], [9], [15], [27]. The output quality of approximate computing is hard to guarantee because it usually depends on the input data. Rely [7], is a language for expressing approximate computation that allows developers to define a *reliability specification*, which identifies the minimum required probability with which a program must produce an exact result. Chisel [27], further enhances the capabilities of Rely by providing combined *reliability* and/or *accuracy* specification. The accuracy specification determines a maximum acceptable difference between the approximate and exact result values, while the reliability specification specifies the probability that a computation will produce an acceptably accurate result. The former specification can be guaranteed through *unequal error protection* methods [3], or by carefully partitioning the computation through reliable or unreliable media [9]. However, meeting the latter specification is a challenge for automatic model generation, since the model must provide reliable information about the possibility of an error occurrence under different workload conditions, i.e., accurate *error prediction*. Prediction of timing errors is a difficult problem because the space of instructions and operands is large. Our attempt to raise the abstraction level at which this characterization and prediction takes place to a microarchitectural level faces the following challenges:

Challenge 1. Dynamic path sensitization could be potentially affected by various parameters, such as operand values, instruction types, and computation history. These become more complex as we move up the level of abstraction in an attempt to identify useful 'features' from the input parameter space for effective timing error models.

Challenge 2. There might be numerous failed circuit paths in the design, and the timing errors might be caused by any one of them. It is unclear how these features will determine what paths to sensitize and and therefore how they will induce timing violations. We have no prior knowledge of the circuit and in general, under cryptographic assumptions Probably Approximately Correct (PAC) learning of Boolean circuits is difficult, even under uniform distribution over the inputs [25].

Proposed Approach. Therefore, to overcome these challenges and provide an accurate error prediction model, based on our previous study in [21], we propose **CLIM**, a cross-level supervised learning-based model to predict timing errors for a given input workload, clock period and FU type. The key idea of **CLIM** is to establish a prediction model that can best explore the relationship from input features to sensitized circuit paths by learning the existing patterns and their corresponding output classes. For a given input data and clock period, **CLIM** predicts output data to be one of two predefined classes–{*timing correct, timing erroneous*} at two levels: bit-level and value-level.

First, we measure the timing errors at each cycle to generate output class labels using GLS of post-layout designs in TSMC 45nm technology. We also perform a trial-and-error process to extract useful features from input data. Second, we apply supervised learning methods to construct and train **CLIM** for four FUs: (INT_ADD, FP_ADD, INT_MUL, FP_MUL) at two levels with extracted input features and output class labels. Third, we evaluate the prediction accuracy of **CLIM** by comparing its predicted results with GLS-based ground truth.

Contribution. This paper makes the following contributions:

1) We present a detailed bit-level and value-level timing error behavior characterization using standard ASIC flow and gate-level simulation. We show that different input operands lead to different error behaviors, and from those conclusions we extract useful 'features' from input operands to train the model. We apply *random forest tree* on the training data to develop **CLIM**, an input workload-aware learning model to predict bit-level and value-level timing errors. To our best knowledge, this is the first cross-level timing error model of FUs considering the effect of input workload.
2) We evaluate the performance of **CLIM** at two granularities under various datasets and circuit parameters such as circuit structures and clock periods. **CLIM** demonstrates average prediction accuracy of 95 and 97 percent at value-level and bit-level respectively, exceeding baseline models.
3) We quantify the degree of error tolerance of arithmetic operations in error-tolerant applications by deriving their bit-level reliability specifications. By comparing such bit-level reliability specifications with **CLIM**-predicted bit-level reliability, we predict output quality of such applications into two classes: {*acceptable, non-acceptable*}. This prediction is on average 97 percent consistent with GLS-based classification. We also utilize **CLIM** to analyze the value-level reliability of FUs, which exhibits deviation within 2.8 percent on average of detailed GLS ground truth. We demonstrate the efficiency of **CLIM** by comparing it to the execution speed of GLS.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 formulates the modeling problem and defines useful terms. Section 4 describes the process of constructing **CLIM**, including timing error extraction, input feature extraction and application of supervised learning methods. Section 5 evaluates **CLIM** performance and describes its utilization at value-level and bit-level. Section 6 discusses the potential limitations of **CLIM**. We conclude our work in Section 7.

2 Related Work

Various techniques have been proposed for tolerating timing errors and delivering an acceptable output. There are mainly three angles in solving this problem: correcting errors, predicting errors and accepting errors.

Correcting Errors: Various hardware methods have been proposed to detect and correct errors [4], [14], [36], [37]. A shadow flip-flop was used in [14] to detect and correct any timing errors induced by speculated voltage scaling. Such shadow flip-flop approaches were also used in error-detection sequential circuit (EDS) [4] to double the sample and compare the signal at different timing. A tunable

replica circuit was deployed at each pipe-stage to monitor timing errors in a less intrusive way. VARIUS [36] proposes a microarchitecture-aware model for the process variation-induced timing errors. To address dynamic variations, a thermal-aware technique scales the voltage/frequency to track temperature fluctuations [18]; an event-guided method takes proper architectural actions to avoid voltage variations [17]. However, the overhead with such intrusive error detection and correction techniques can be large, e.g., 18 percent [28] and 21 percent [16] overheads in area, and 8 percent [8] power overhead.

Predicting Errors: To avoid such penalties, less-intrusive techniques predict timing errors in advance and then adaptively change timing guardbands to improve performance while preventing errors.

Several works predict timing errors at instruction-level [31], [34], [41]. They claim that some instructions, which they call critical instructions, are more prone to timing errors. They rely on large-scale GLS and determine critical instructions by monitoring their timing violation when the clock period is decreased. These instruction-level schemes have two main drawbacks. First, they characterize instruction-level errors based on GLS with limited benchmarks. The input data from these benchmarks may only sensitize a limited number of critical paths, resulting in an underestimate of instruction-level criticality. On the other hand, critical instructions are characterized based on a worst-case scenario of GLS, but in reality, the instruction might not face timing error because its input operand is not sensitizing the critical path. This results in pessimistic modeling. Second, the large amount of GLS is time-consuming and not scalable and may not cover all path sensitization behaviors. Thus, these two issues motivate us to develop a timing-efficient workload-aware model that does not need exhaustive simulation and considers dynamic path sensitization variation under various input workload.

There do exist some works that use machine learning to predict timing errors [22], [30], [38], [42]. A linear discriminant classifier predicts the timing error rate of FUs by obtaining PVTA variation information and then adjust timing guardbands accordingly [30]. However, it overlooks the effect of input operands by predicting errors purely based on PVTA parameters. A logistic regression (LR)-based method is used to predict bit-level timing errors in [22]. However, we have shown LR is not a better choice compared to RFC. Furthermore, its coarse-grained CPRs sometimes could miss interesting behaviors: it uses up to 15 percent CPR but as shown in Fig. 3 some FUs do not have timing errors until 40 percent. And it does not consider circuit topology as an input feature in developing the model. B-Hive, predicts bit-level timing errors for voltage-scaled FUs [38]. This work is similar to ours with one difference: it predicts bit-level error based on voltage. The researchers claim that incorporating input workload in model development provides only a negligible increase in accuracy. Nonetheless, we found that incorporating input operands into error modeling is important, hence the timing error behavior of FUs, as shown in Fig. 2. Our supervised learning model can be aligned with such approaches to provide an accurate error model for hardware execution under various operating conditions.

Accepting Errors: Although such adaptive methods are effective in improving performance, these methods strive to achieve exact instruction execution and are not useful for the accuracy specification with a wide range of requirements. Recently, the research community has embraced approximate computing, which utilizes intrinsic error tolerance at the application level to allow occasional error occurrences in a system as long as the output quality is still acceptable by users. Such applications include multimedia applications, machine learning applications and emerging fields such as recognition, mining, and synthesis (RMS). Hardware-level approximation techniques relax the design constraints (computing device or memory) by tuning approximate knobs. An accuracy-configurable integer adder offers two operating modes: exact and approximate [23]. During the exact operating mode, error detection and correction must be applied, while in the approximate mode the errors can be ignored and left out uncorrected. Similarly, floating-point units can dynamically switch between exact and approximate modes [33]. Its approximate mode ignores the timing errors on the less significant N bits of the fraction part where N is a reprogrammable memory-mapped register. Another technique is proposed for timing error acceptance to improve the quality-energy tradeoff for DCT/IDCT components [19].

Our Work: In this work, we combine *predicting errors* and *accepting errors*. For error prediction, we differ from previous work in that: 1) we predict a timing error at two granularities by using a *RFC* method rather than relying on an exhaustive simulation to characterize the timing errors; 2) we consider the effects of input workload and circuit topology on path sensitization, hence the timing errors; 3) we combine the error prediction with error acceptance in approximate computing to estimate the application quality under various configurations.

3 Problem Formulation

Problem Formulation: We follow the procedure of representing the timing errors of a circuit as a function of circuit parameters and input workload. More specifically, we abstract a circuit as a mapping from an input space $\mathcal{I}$ consisting of p circuit parameters (e.g., the circuit structure, and clock speed) and m input bits, to create an input I. Suppose the function implemented by an ideal circuit, without timing errors is ϕ_i and the function of the real physical circuit is ϕ_r, which includes the effect of timing errors. The output value in error is $\psi(I) = \phi_i(I) \oplus \phi_r(I)$, where $\oplus$ is the XOR operator. Our goal is to learn (an approximation of) ψ given a range of inputs and circuit parameters.

However, in general we do not know the structure of the ψ function – it is not even clear a-priori if the structure of ψ is similar to the structure of the circuit function ϕ. We thus propose evaluating a sequence of *non-parametric* classification methods to classify the inputs and thereby map them into different outputs as shown in Section 4.2.3.

Definition: We define $x[t]$ as the input operands vector, $y[t]$ as the GLS output and $y\_gold[t]$ as the pure-RTL simulation output value, all at cycle t. Note that $y[t]$ may contain timing errors while $y\_gold[t]$ is always clean. We denote $y_i[t]$ and $y\_gold_i[t]$ as ith bit position of the GLS and RTL simulation output respectively, where $i = 1, 2, \ldots N$ and N is the

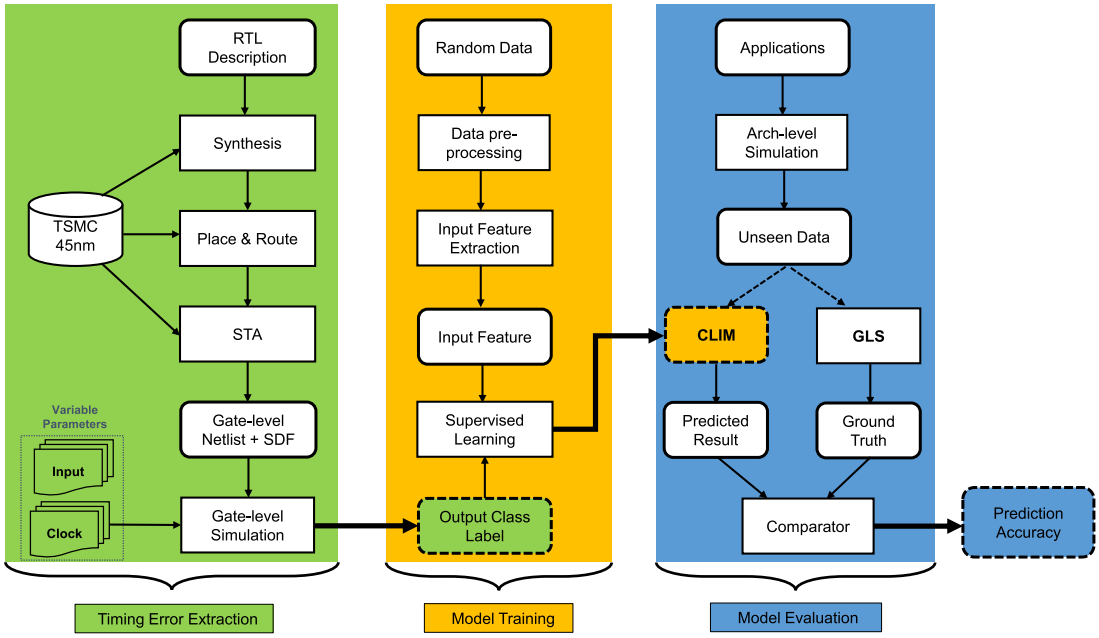


Fig. 1. **CLIM** model overview with three key stages: a) Timing Error Extraction to examine the timing errors under different input workload and clock periods, and to generate output timing class labels; b) Model Training to apply random forest classification (RFC) to construct **CLIM** with extracted input features from preprocessed data and output timing class labels; c) Model Evaluation to evaluate **CLIM** prediction accuracy by comparing its predicted results with GLS-generated ground truth, under different benchmarks datasets.

number of output bits. We define the two classes for output value: C_e representing *timing erroneous* and C_c representing *timing correct*, and we define the class of $y[t]$ and $y_i[t]$ as $C[t]$ and $C_i[t]$ respectively. At cycle t, if $y[t] = y\_gold[t]$, then $C[t]$ is marked as class C_c. If mismatched, then $C[t]$ is marked as class C_e. The same principle applies to bit-level to determine the bit-level timing class. Our goal is to predict the output class $C[t]$ ($C_i[t]$) at cycle t as a function of input workload, clock period and FU type, denoted as follows:

$$C[t] = f(t_{clk}, FU_{type}, x[t], x[t-1], x[t-2], \ldots, x[1]) \quad (1)$$

$$C_i[t] = f(t_{clk}, FU_{type}, x[t], x[t-1], x[t-2], \ldots, x[1]), \quad (2)$$

where t_{clk} is the clock period, FU_{type} is FU type, $x[t]$, $x[t-1], \ldots x[1]$ are the input workloads at cycle $t, t-1, \ldots 1$. The reason for putting the entire input stimuli history is that we do not know whether previous input workload would set a circuit state and thereby have an effect on the timing error behavior of current cycle t. In instruction-level models [11], [34], [41], the effects of input workload are not considered. However, per the conclusion in [38], timing errors would be better modeled by including a deep history. Therefore, we later investigate the features from input data which affect the output timing error behaviors, as shown in Section 4.2.2. In summary, this becomes a binary classification problem: for a given input data and circuit parameters at cycle $t, t-1, \ldots 1$, **CLIM** predicts the output $C[t]$ ($C_i[t]$) to be one of two classes: C_c or C_e.

4 CLIM Model

CLIM *Model:* It is composed of three phases as shown in Fig. 1: *Timing Error Extraction*, *Model Training* and *Model Evaluation*. a) The *Timing Error Extraction* phase implements the standard ASIC flow and uses GLS to generate timing class: C_c if matched, otherwise C_e. b) In the *Model Training* phase, we preprocess the training data and extract useful features from them, which will then be incorporated into modeling. We then apply RFC method to construct the model with the input features and output timing class labels generated from last phase. c) In the *Model Evaluation* phase, we use **CLIM** to predict the timing class of the FU output value and then compare the predicted class with GLS ground truth to compute prediction accuracy. More details about the three phases are illustrated as follows.

4.1 Timing Error Extraction

We use both 32-bit integer and single-precision floating point units (FPUs) as our experimental platforms: INT_ADD, INT_MUL, FP_ADD, FP_MUL, implemented in VHDL. FPUs are fully compatible with the IEEE-754 standard and can provide more complex structures compared to their integer counterparts. We change the data types and circuit structures to better evaluate the robustness of our model. We extract the value-level and bit-level timing errors through *Timing Error Extraction* module as illustrated in Fig. 1, which is divided into several steps.

We use FloPoCo [13] to generate the synthesizable VHDL codes of combinational circuits. We put wrappers at input and output ports to have better timing notations. We then use *Synopsys Design Compiler* to synthesize the VHDL codes and use *Synopsys IC Compiler* to generate post place-and-route netlist in TSMC 45nm technology. Next, we use *Synopsys PrimeTime* to perform static timing analysis, generating a Standard Delay Format (SDF) file. Then, we vary clock periods to simulate the netlist with *Mentor Graphics Modelsim* to do SDF back-annotation gate-level simulation to generate output data $y[t]$. The input stimuli of simulation x[t], comes from two sources: Python-written random data generator

and the application input data profiled using *Multi2Sim* [39], a cycle-accurate CPU-GPU heterogeneous architectural simulator. At cycle t, the input stimuli vector $x[t]$ is applied to **GLS** to generate output $y[t]$ ($y_i[t]$) and compare with pure-RTL simulation result $y\_gold[t]$ ($y\_gold_i[t]$) to derive timing errors as shown in Section 3.

4.2 Model Training

4.2.1 Data Preprocessing

To collect the training input data, we generate the random input data as stimuli for simulations. For a 32-bit bit vector, we randomly set each bit independently to produce the training data. But note that for test input data, which might come from application profiling, its format could be in decimal format. We need to preprocess such input data to convert it into the correct format, for example, 0.5 should be converted to 00111111000000000000000000000000 if the FU is of IEEE-754 single-precision format. The reason for doing this is that the FU accepts 32-bit input vectors and each bit value could affect the dynamic path sensitization, hence the final timing class. The decimal value cannot precisely represent the impact of each bit location. Therefore, in our model training, we use each bit value to compose input features rather than the decimal value alone.

As a matter of methodology, we remove the repetitive pair of $\{x[t-1], x[t]\}$ in the dataset because the same pair of current and preceding input leads to same timing class (as shown next). We also exclude an ambiguous case where the preceding input $x[t-1]$ is the same with current input $x[t]$, because even if a timing violation occurs at cycle t, the output could still appear to be correct. We note that these two situations are unlikely especially with randomly chosen 32-bit operands.

4.2.2 Feature Extraction

From the processed training input data, we need to find out the useful input features that determine the output timing class. Empirically, the current cycle input workload $x[t]$ directly affects the dynamic path sensitization at cycle t, hence the final output timing class. However, it is not clear whether the preceding input has impact on the current cycle path sensitization and timing behavior. To explore the effect of history input workload, we use a trial-and-error process, which iteratively varies the preceding input while fixing the current input workload. We set the experiment as follows:

- Case 1: We fix the current input $x[t]$ and randomly vary the preceding cycle input $x[t-1]$, where we set cycle $t = 10, 30, 50, 70, ...$ We use this to evaluate the effects of immediately preceding input.
- Case 2: We fix both the current input $x[t]$ and the immediately preceding input $x[t-1]$, while randomly varying the preceding input of immediately preceding input $x[t-2]$, where we set t as above. We use this to evaluate the effects of the deeper history.

We use 100K cycles for simulation and use different clock periods. At value-level, in Case 1, we found the timing class $C[t]$ varies irregularly. More specifically, by comparing every two examined neighboring outputs, e.g., $c[30]$ and $c[50]$, we found 44 percent of neighboring pairs exhibit different timing classes. In Case 2, we found all output timing classes $C[t]$ exhibit exactly the same behaviors, i.e., all C_c or C_e. At the bit-level, we examine the hamming distance between every two examined neighboring timing class outputs, where each output is a 32-bit vector of $C_i[t]$, where $i = 0, 1, \ldots 31$. The hamming distance between two vectors is defined as the number of mismatched bit positions, e.g., 10,001 and 10,000 has a hamming distance of 1. In case 1, we can see most pairs have a positive hamming distance, indicating that the resulting output timing classes are different. In case 2, the neighboring hamming distance is always 0, which means the bit-level timing class output is exactly the same for every bit position.

This key observation shows that only the preceding and current cycle input vectors $x[t-1]$, $x[t]$ are accountable for timing errors in the current cycle t. For a combinational logic placed between sequential elements, it is natural that the preceding input workload sets a state for the circuit, and then the current input toggles nets based on the current state. Thus, the path sensitization depends on both the current circuit state and current circuit input. However, since most previous works do not consider input operands as features for timing error modeling [30], [41] and some work points out that including a deeper history would increase the accuracy [38], we investigate the effects of input operands and history. This key observation locates the source factors that determine the dynamic path sensitization and motivates an workload history-aware modeling approach.

On the other hand, we explore circuit parameters that can reflect or partially reflect the timing violation behaviors. One parameter that can be used is timing class output. At the value-level, the circuit output timing errors occur if and only if at least one output bit location faces a timing violation. The timing violation of a particular output bit occurs only when there is at least one sensitized circuit path ending at that bit facing violation. A sensitized path would have all of its nodes toggled [6]. Hence, the end point, i.e., the output bit, should also be toggled. Thus, we also take the final output value into our modeling as part of the input feature. In summary, by composing aforementioned features, our final input features are $\{x[t-1], x[t], y\_gold[t-1], y\_gold[t]\}$. At bit-level, the same principle applies and leads to the final input features are $\{x[t-1], x[t], y\_gold_i[t-1], y\_gold_i[t]\}$.

4.2.3 Training Process

Since the model has two levels, we also need to train the model at two-levels respectively. At the value-level, we set $\{x[t-1], x[t], y\_gold[t-1], y\_gold[t]\}$ as the input feature and C_t as output class labels; At the bit-level, we set $\{x[t-1], x[t], y\_gold_i[t-1], y\_gold_i[t]\}$ as the input feature and $C_i[t]$ as output class labels. Therefore, for a given circuit with *K-bit* output, a set of *K+1* binary classifiers is developed. *Model Training* stage in Fig. 1 illustrates the process of constructing the model. First, we apply 500K random data points as training input data. We extract the input feature through *Feature Extraction* module and output labels through *Timing Error Extraction* stage. We then apply and evaluate several supervised learning methods on these training data to train **CLIM**.

While certain positive learnability results exist for specific classes of circuits [26], they do not cover the circuits we consider here. In contrast to these aforementioned methods (which essentially learn a model of the circuit under consideration), we focus on learning when a circuit does not work as desired, i.e., the circuit contains timing errors. Capturing the timing errors will require learning a binary classifier. Thus, we evaluate four supervised learning methods for their increased sophistication and practical use: k-nearest neighbor (**k-NN**), support vector machine (**SVM**), logistic regression (**LR**) and random forest tree (**RFC**) classifiers [2]. These learning methods are very popular in classifying various kinds of tasks and we want to see whether they fit for the timing error classification tasks. By comparing them we can also conclude why we choose a particular method. The machine learning module is provided by Scikit learning module [29] in Python, and we use the default configurations for the classifiers.

We evaluate **k-NN** because it provides useful theoretical properties [12] and has limited parameters to train. Given an input vector x, k-NN classifier predicts a timing error if the majority of the k nearest neighbors of x in the dataset $\mathcal{D}$ has timing errors. However, in our case, K-NN finds its nearest neighbors based on hamming distance, which actually overlooks the situations wherein different bit positions would have disparity of significance on path sensitization. Thus, we would expect the k-NN model perform badly. In addition to this, k-NN classifiers typically have sub-par generalization performance (i.e., performance on new data) when available labeled data is limited, which could potentially lead to appropriate feature normalization and scaling issues.

To address these problems, we evaluate **LR** and **SVM** because they can learn weights w on each bit position, which considers the disparity of significance of different bit positions.

In **LR**, we learn weights to compare the logic functions that perform well on the training data $\mathcal{D}$. In particular, for an input x we predict 1, or the timing error, if the ratio of $\frac{F(x)}{1-F(x)} >= 1$ where $F(x)$ is given by

$$F(x) = \frac{1}{1 + e^{-w \cdot x}}. \tag{3}$$

In **SVM**, given labels y_i for the N training data points x_i, SVM learn w based on the following large margin optimization problem:

$$\min_{w,\eta,b} \frac{1}{2}||w||^2 + C\sum_i \eta_i \tag{4}$$

$$\text{s.t. } y_i(w \cdot x_i + b) \geq 1 - \eta_i, \tag{5}$$

where w is weights and b is offset, which jointly determine a separating hyperplane. Essentially, weights are learned that maximize the margin (η_i) by which examples are classified correctly. Typically, input examples are mapped to a higher dimensional kernel space (we use the popular Gaussian Radial Basis Function (RBF) kernel in our experiments).

LR and *SVM* can learn the disparity of significance of different bit positions. However, one potential limitation is that, they put a fixed weight on each bit position. It is unclear whether each bit position contributes linearly to the final timing error, and the contribution of each bit position might be changed along with the change of other bit values. Think about an "AND" gate – if one input is zero, then the final result will always remain the same regardless of the value of the other input.

To address this problem, we propose to use *RFC*. *RFC* is an ensemble-learning method that constructs multiple decision trees at training time and uses their averaging to improve accuracy and control overfitting. Decision trees are a non-parametric supervised learning method that aims to establish a tree-like model by learning decision rules from training data. As a white box model, the decision rules are based on Boolean logic; thus it is easy to understand and interpret. However, decision trees can easily create overly complex trees and become very deep by learning many irregular patterns with a large variance. This will lead to the notorious overfitting problem, which cannot generalize the data well. RFC alleviates this problem by constructing multiple decision trees. In our scenario, *RFC* can predict the timing errors based on the decision rules it learned from the data patterns. This method emphasizes the disparity of different bit positions and also considers the interaction between the input bits. Although it may lose the opportunity to learn some "irregular" patterns, overall it reduces overfitting and boosts performance.

$$S = \begin{bmatrix} f_{1A} & f_{1B} & f_{1C} & C[1] \\ f_{2A} & f_{2B} & f_{2C} & C[2] \\ \vdots & \vdots & \vdots & \vdots \\ f_{dA} & f_{dB} & f_{dC} & C[d] \end{bmatrix} \tag{6}$$

$$S_1 = \begin{bmatrix} f_{5A} & f_{5B} & f_{5C} & C[5] \\ f_{10A} & f_{10B} & f_{10C} & C[10] \\ \vdots & \vdots & \vdots & \vdots \\ f_{100A} & f_{100B} & f_{100C} & C[100] \end{bmatrix} \tag{7}$$

$$S_2 = \begin{bmatrix} f_{15A} & f_{15B} & f_{15C} & C[15] \\ f_{20A} & f_{20B} & f_{20C} & C[20] \\ \vdots & \vdots & \vdots & \vdots \\ f_{200A} & f_{200B} & f_{200C} & C[200] \end{bmatrix} \tag{8}$$

$$S_M = \begin{bmatrix} f_{3A} & f_{3B} & f_{3C} & C[3] \\ f_{40A} & f_{40B} & f_{40C} & C[40] \\ \vdots & \vdots & \vdots & \vdots \\ f_{400A} & f_{400B} & f_{400C} & C[400] \end{bmatrix}. \tag{9}$$

We use Equation (6) to (9) to illustrate the process of creating a random forest classifier. Equation (6) is the original training dataset, where we have d input samples, each of which is composed of 3 features, that lead to a particular class C. We split the entire training data into M independent sub-sample datasets, $S_1, S_2, \ldots, S_M$. Then, we use M decision tree classifiers to fit all sub-sample datasets. Hence, M decision trees are developed. Finally, each decision tree predicts the class and we use the majority vote of all M votes as the final prediction result. In the model construction, we need to

TABLE 1
Prediction Accuracy, Training Time, and Testing Time of Four Learning Methods

method	Accuracy	Training Time	Testing Time
LR	85%	42.8 s	0.21 s
KNN	87%	4224 s	849 s
SVM	92%	18600 s	1968 s
RFC	93%	94.74 s	0.26 s

tune several important parameters such as number of trees in the forest, the depth of trees, and the number of features to test at each node. Increasing these parameters could possibly improve the prediction accuracy but incurs more computational overhead. Thus, we use the default settings recommended by Scikit learning module [29].

Table 1 presents the prediction accuracy, training and testing time of four methods using 100K random training data and 10K random test data under a computer configuration of 2-core Intel(R) Xeon(R) CPU E5504@2.00 GHz and 50 GB memory. More specifically according to the Table, LR is fastest because of its relatively easy computation process, which assigns weight to each bit position. However, it achieves the lowest accuracy because the contribution of each bit position is not identical to the final output. Although SVM achieves good accuracy, compared with the other three classifiers its long running time impedes its use. Comparing to the other three baseline classifiers, we can emphasize why RFC is the choice because it can interpret the difference at each bit position (compared with KNN) as well as interactions among bits (compare with SVM and LR). Finally, we choose RFC due to its high accuracy, fast computing time and superior interpretability. Note that the training process is a one-shot activity, so the testing time is more important for model usage.

4.3 Model Evaluation

We evaluate the model performance by comparing with GLS under various FUs, clock periods, and datasets.

4.3.1 Evaluation Metrics

Prediction Accuracy: Prediction accuracy is an intuitive measurement of how accurate the predictions are. We define mean bit-level prediction accuracy (MBPA) and mean value-level prediction accuracy (MVPA) as follows:

$$MBPA[clk] = \frac{\sum_{bit\ i}\left(\frac{\sum_{cycle\ t}|C^{(pred)}_{clk,i,t} == C^{(real)}_{clk,i,t}|}{\#cycles}\right)}{\#bit\_positions} \tag{10}$$

$$MVPA[clk] = \frac{\sum_{cycle\ t}|C^{(pred)}_{clk,t} == C^{(real)}_{clk,t}|}{\#cycles}, \tag{11}$$

where $C^{(pred)}_{clk,i,t}$ and $C^{(real)}_{clk,i,t}$ are the predicted and real timing classes (1 for timing-erroneous and 0 for timing-correct) for bit position i at a given clock period clk and cycle t. $C^{(pred)}_{clk,t}$ and $C^{(real)}_{clk,t}$ are the predicted and real timing classes (1 for timing-erroneous and 0 for timing-correct) for the entire value at a given clock period clk and cycle t. Its best value is 0 and worst value is 1.

4.3.2 Comparison Methods

We compare **CLIM** against following baseline methods, which can help us evaluate the true performance of our model:

- *rand* [35]: This model is adopted from [35]. We call it *rand* model because it predicts the timing class with random guessing without considering the effects of input operands.
- *fixed* [11], [34], [41]: This model is adopted from [11], [34], [41]. We call it *fixed* model because it always predicts a fixed timing class based on the pre-characterized information, i.e., it predicts C_c (C_e) when the clock period does (not) meet the measured maximum instruction-level timing delay. At the bit-level, it always predicts the particular timing class that has more instances in training data. For example, if in the training data more data are timing correct than erroneous, then this model always predicts *timing correct*. Note that this model can lead to high prediction accuracy if the dataset is heavily biased, e.g., 99 percent of the output data is C_c. Then its prediction accuracy is 99 percent by always predicting *timing correct*.

5 Experimental Results

In this section, we first describe our experimental setup. Second, we characterize hardware timing behaviors. Third, we evaluate **CLIM** performance at both the bit-level and the value-level. Lastly, we examine **CLIM** efficiency.

5.1 Experimental Setup

To provide a decent amount of timing errors, we set the experimental clock period for each FU for which their value-level timing error rates (TERs) reach 10, 20, and 30 percent under random data approximately, where TER is calculated as $\#erroneous\_cycles/\#total\_cycles$. From this point on, we refer the *clock period reduction* (CPR) pair, which leads to such three TERs as *{CPR1, CPR2, CPR3}*. Note that such CPR pair values are different for each FU.

While such CPRs could be derived through a trial-and-error GLS, it is very time-consuming since we need to iterate clock periods until the target timing error rates is met. This process could take numerous GLS, especially considering we have four FUs and three CPRs. Therefore, we derive such clock periods through the characterization of dynamic delays of all simulation cycles. We know a timing error occur if the clock period is less than the dynamic delay at a cycle; therefore we only need to sort all the dynamic delays and find the top 10, 20, and 30 percent dynamic delay as the *{CPR1, CPR2, CPR3}*. First, we extract all the dynamic delays; we parse the value change dump (VCD) file, which is generated by GLS at a relatively slow clock period to make sure there is no timing violation. The VCD file records the toggled endpoints of each circuit path at each cycle. Second, for each clock cycle, we use the last toggle event time of the input pin of all sequential elements (flip-flop, registers, etc.) to subtract the last positive clock edge arrival time to get the maximum delay at that cycle. For example, at cycle N the positive clock edge occurs at time t, and the very last toggled event at the data input pin of all sequential elements occurs at time t', then the dynamic delay at this cycle is $t' - t$. Third, we sort

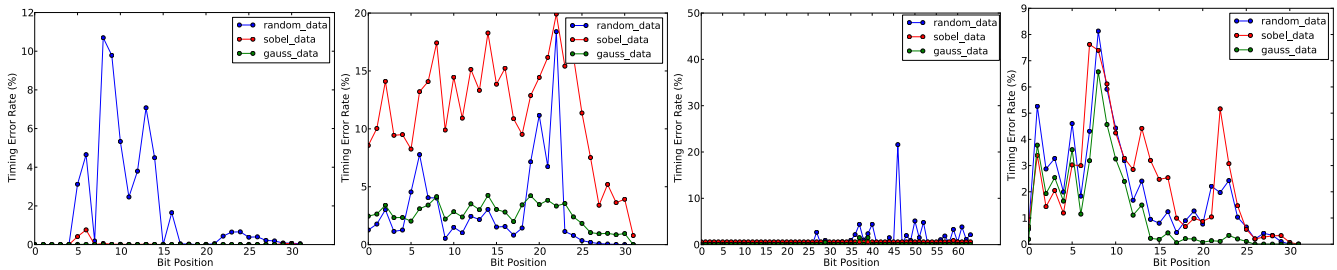


Fig. 2. Bit-level timing error rate (%) under different input datasets.

all the dynamic delays in a descending order, and locate the delay at the top 10, 20, and 30 percent position.

We use three datasets to evaluate and utilize the model: random data, Sobel filter and Gaussian filter. The two image processing applications are adopted from AMD APP SDK [1]. The openCL code of these applications are simulated by Multi2Sim to profile input data. The images are adopted from Caltech-UCSD Birds 200 vision dataset [40].

5.2 Hardware Characterization

Timing errors are caused by the violations of circuit timing specification where the sensitized path delay is larger than the clock period. Thus, the key to modeling timing errors is to model the path sensitization behavior. We present a case study that demonstrates the effect of input operands on timing errors. We utilize the *Timing Error Extraction* module described in Section 4.1 to characterize the timing errors of different FUs.

5.2.1 Bit-Level

We depict bit-level timing errors at *CPR3* under different input datasets as illustrated in Fig. 2, where we observe several important facts.

First, under the same input dataset, different bit positions exhibit different timing error rates. This is because different output bits lie on different paths with different delays. Second, under a different input dataset, the same bit positions exhibit different timing error rates. For example, in Fig. 2c, some bit positions under the *sobel* and *gauss* datasets exhibit a nearly zero timing error rate while those same bits under random dataset exhibit up to a 20 percent timing error rate. This is because different input data exercise different paths towards an output bit, thus causing different delays. Third, some bit positions might exhibit similar timing error rates under different datasets. For example, in Fig. 2d, some bit positions exhibit a similar timing error rate under three datasets. From this observation, we infer that the path sensitization behavior in FP_MUL is relatively similar under these three input datasets, thus resulting in similar timing error rates. In summary, these observations of input data impact on timing error behavior has motivated us to develop an workload-aware model.

5.2.2 Value-Level

In Fig. 3 we present three different CPRs of four FUs that would lead to 10, 20, and 30 percent TERs, where we can observe several important facts. First, for INT_ADD, 10 percent TER is caused by 43 percent CPR, meaning that 43 percent timing margin is used to protect 10 percent timing violations. This suggests a large timing margin has been used for worst-case scenarios. Second, TER increases rapidly after that: TER increases from 10 to 30 percent while CPR only increases from 43 to 45 percent. This suggests that many paths of similar lengths are sensitized in this delay range, which this is consistent with the timing wall phenomenon [24]. When we compare FP_ADD timing characteristics with INT_ADD, we found there is a difference and a similarity. The difference is that, for FP_ADD, the same level of timing error rate is caused by a lower CPR, indicating that FP_ADD is more susceptible to clock period reductions. The similarity is that, the TER also rapidly increases after that point: TER increases from 10 to 30 percent while CPR only increases from 21 to 23 percent. This is also consistent with the timing wall phenomenon. Both designs suggest that there is a large timing margin used to protect worst-case timing violations (10 percent) and emphasizes the need for accurate timing error model.

5.2.3 Failed Paths

We also compute the number of paths with negative slack under such CPRs for INT_ADD. As illustrated in Fig. 5, the number of failed paths increases with the CPR. We note that for every CPR point, there are more than 6K failed paths. This means that once any path in this set fails, the whole design faces timing violation. This corresponds to challenge 2 in Section 1, where multiple path failures can lead to timing violations; however we need to learn whether any member of these failed paths will be sensitized. For the FP_ADD, even for the

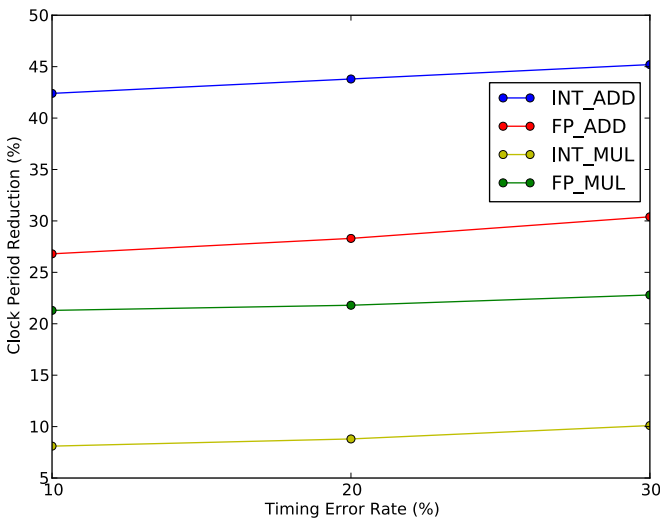


Fig. 3. Value-level timing error rate (%) versus clock period reduction (CPR).

TABLE 2
Bit-Level CLIM on INT_ADD for Timing Error Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	96.7%	92.8%	50.0%	96.1%	92.3%	49.9%	95.6%	88.7%	50.0%
sobel	99.8%	99.9%	49.9%	99.9%	99.9%	49.9%	99.9%	99.8%	50.0%
gauss	99.9%	99.9%	49.9%	99.9%	99.9%	49.9%	99.9%	99.9%	49.9%

slightest TER at 10 percent, we observed more than 30K failed paths. **CLIM** needs to predict the timing violation even if there is only one path failure, which makes it an extremely difficult task to learn such path sensitization behaviors.

5.3 Bit-Level CLIM

We first evaluate the bit-level model of **CLIM** on four FUs and compare with baseline models.

Tables 2 and 3 present the MBPA of **CLIM** for INT_ADD and FP_ADD, where we can observe several facts. For INT_ADD, **CLIM** exhibits prediction accuracy between 95.6-99.9 percent across three datasets and CPRs. Meanwhile, **fixed** can deliver prediction accuracy between 88.7-99.9 percent and *rand* almost always achieves 50 percent accuracy. More specifically, *fixed* only achieves 99.9 percent accuracy when the input dataset is *sobel* or *gauss*. These two datasets are heavily biased with almost zero TERs, according to Fig. 2a. For *rand* dataset, which is more representative, **CLIM** achieves 96.2 percent accuracy while *fixed* achieves 91.3 percent accuracy on average. For FP_ADD, **CLIM** exhibits prediction accuracy between 93.5-98.7 percent across three datasets and CPRs. Meanwhile, *fixed* can deliver prediction accuracy between 87.9-97.5 percent and *rand* almost always achieves 50 percent accuracy. On average, **CLIM** achieves 96.3 percent accuracy and *fixed* achieves 92.1 percent accuracy. In summary, under mild TERs, *fixed* classifier can almost always achieve decent accuracy, perhaps leading one to doubt whether it is necessary and worthwhile to develop **CLIM**. In fact, *fixed* classifier has no ability to identify any positive output because it always predicts outputs to be C_e. This will severely hurt system reliability as it assumes no error when an error could occur. Thus, we further compare these models on more FUs by using them to predict the output quality of approximate computing applications as presented in Tables 4 and 5. Before getting to the result, we first introduce bit-level reliability specification of approximate computing applications.

Bit-level Specification. The error-tolerant applications used in the approximate computing field exhibit enhanced error resilience at the application-level when multiple valid output values are permitted. Instead of a single output value, the output value is associated within an application-specific quality metric, such as peak signal-to-noise ratio (PSNR).

TABLE 3
Bit-Level CLIM on FP_ADD for Timing Error Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	97.6%	91.1%	49.8%	95.5%	88.6%	50.1%	94.8%	87.9%	50.0%
sobel	96.3%	93.4%	49.9%	94.4%	89.4%	50.0%	93.5%	88.6%	49.9%
gauss	98.7%	97.5%	50.0%	98.1%	96.7%	49.9%	98.1%	96.2%	50.0%

TABLE 4
Bit-Level CLIM on INT_MUL for Application Quality Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
sobel	100%	100%	3.1%	100%	100%	3.1%	100%	100%	3.1%
gauss	100%	100%	4.6%	100%	100%	4.6%	98.4%	95.3%	4.6%

Therefore, if execution is not numerically precise, the application can still appear to execute correctly from the users' perspective. We focus on error-tolerant applications mainly from the image processing domain, including Sobel filter and Gaussian filter. In image processing applications, a PSNR larger than 30dB is generally considered as acceptable to users [33]. As illustrated in Fig. 4, it is hard to tell the difference between exact output and approximate output.

We quantify the degree of error tolerance of arithmetic operations in these applications by defining the notion of bit-level reliability specification. Similar with Rely [7] described in Section 1, it defines the minimum required probability with which the arithmetic operation output bit must be correct so that the application can deliver an acceptable output. For example, if we say reliability specification of 20th bit of FP_MUL operation is 70 percent, it means if the reliability of this bit is lower than 70 percent, the application output quality is not acceptable.

We compute the reliability specification for each bit of interested arithmetic operations through an iterative fault injection process as shown in Fig. 6. First, we flip the one output bit of our interested operation (e.g., INT_MUL) with an initial probability that is small enough so that the application output quality is acceptable. This fault injection is done using our-modified version of Multi2Sim [39] simulator. Second, we check the output quality of the resulted application using Matlab. Third, if the output quality is acceptable, we increase the bit flip probability and repeat step 1 and 2 until the output quality is not acceptable, then we use the last acceptable probability as the threshold probability. After these steps, we calculate the reliability specification as $1 - threhold\_probability$. We repeat such fault injection processes for every bit position across multiple arithmetic operations and error-tolerant applications.

Quality Estimation. We then use **CLIM** to predict the error-tolerant application quality into two classes: {*acceptable*, *non-acceptable*} with the following process. First, we obtain the bit-level reliability specification of each bit position. Second, we use **CLIM** to predict the bit-level TER of each bit position, and then use $1 - TER$ to derive bit-level reliability. We then compare the predicted reliability with reliability specification. If the predicted reliability is greater than the specification, then **CLIM** will predict the application quality is acceptable; otherwise it is unacceptable. For example, if

TABLE 5
Bit-Level CLIM on FP_MUL for Application Quality Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
sobel	100%	68.7%	87.5%	100%	68.7%	87.5%	96.8%	68.7%	87.5%
gauss	96.8%	75.0%	78.1%	93.7%	75.0%	78.1%	93.7%	75.0%	78.1%

(a) Original Input (b) Exact Output (c) Approximate Output

Fig. 4. (a) Original input image. (b) Error-free exact Sobel filter output with PSNR = inf. (c) Error-injected approximate Sobel filter with PSNR = 30dB.

the predicted reliability for 20th bit of FP_MUL is greater than 70 percent, then **CLIM** will predict the application quality is acceptable. Third, we use GLS to compute the ground-truth reliability for each bit position. Then we use such reliability to determine whether the application quality is acceptable by comparing it with reliability specification, as with the second step. Finally, GLS will produce a ground truth result on whether an application quality is acceptable or not. Fourth, we then compare the prediction result of **CLIM** with GLS ground truth and compute the prediction accuracy across all the bit positions. We repeat the same process for *fixed* and *rand* classifier.

Tables 4 and 5 compare the accuracy of the three models. For INT_MUL, both **CLIM** and *fixed* achieve high prediction accuracy because according to Fig. 2c, *sobel* and *gauss* have almost zero TERs. Thus, the real reliability is close to 100 percent which matches the *fixed* classification. The *rand* achieves low accuracy because its predicted reliability is close to 50 percent while the real reliability is close to 100 percent. For most bit positions, the bit-level specification is between 50 and 100 percent, thus *rand* has a different prediction than ground truth, resulting in low accuracy. For FP_MUL, **CLIM** achieves accuracy between 93.7-100 percent while *fixed* and *rand* achieves 68.7-75.0 percent and 78.1-87.5 percent respectively. The low accuracy of *fixed* is due to the fact that most bit positions of FP_MUL have non-zero TERs. For example, for bit position 21 under *sobel* dataset whose bit-level specification is 99 percent, its ground-truth reliability is 96.7 percent as computed by GLS, making the quality non-acceptable. Since **CLIM**-predicted reliability is 95.5 percent, *fixed*-predicted reliability is 100 percent and *rand*-predicted reliability is 50 percent, both **CLIM** and *rand* correctly predict *non-acceptable* while *fixed* predicts *acceptable*, leading to a misprediction. In summary, **CLIM** demonstrates robustness across FUs and datasets regardless of whether it is biased, while *fixed* achieves low accuracy due to its inability to identify erroneous instances.

5.4 Value-Level CLIM

Tables 6 and 7 present the MVPA of **CLIM** for INT_ADD and FP_ADD. For INT_ADD, **CLIM** exhibits average prediction accuracy at 97.4 percent across three datasets and CPRs. Meanwhile, *fixed* delivers average prediction accuracy at 6.8 percent and *rand* almost always achieves 50 percent accuracy. For FP_ADD, **CLIM** exhibits average prediction

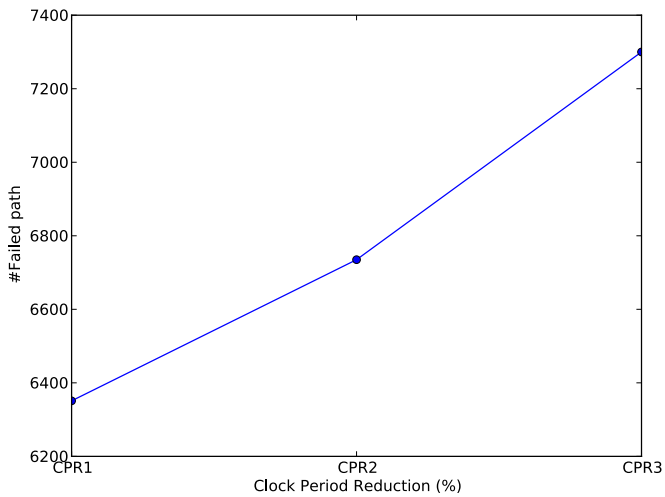


Fig. 5. The number of failed path of INT_ADD under different clock period reductions.

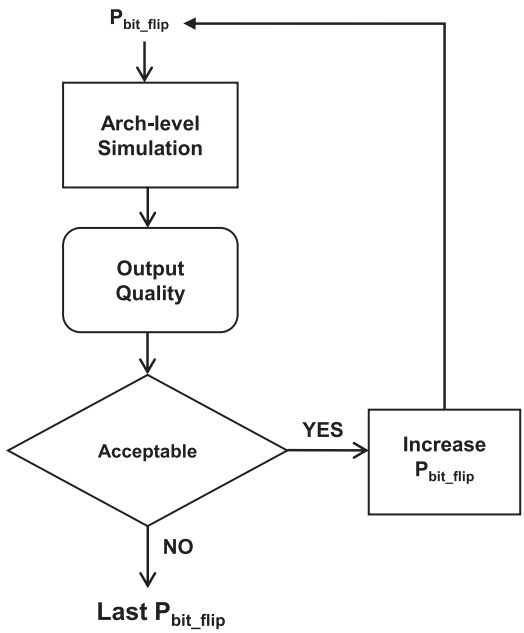


Fig. 6. Derive bit-level reliability specification for error-tolerant applications through fault injection. P_{bit_flip} is a bit flip probability.

TABLE 6
value-Level CLIM on INT_ADD for Timing Error Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	96%	9.9%	49.7%	93.5%	19.0%	50.0%	91.2%	29.6%	49.8%
sobel	99.3%	0.7%	49.9%	99.0%	0.8%	49.8%	98.4%	1%	50.0%
gauss	99.9%	0.1%	50.0%	99.9%	0.1%	50.0%	99.0%	0.1%	49.9%

accuracy at 93.6 percent across three datasets and CPRs. Meanwhile, *fixed* delivers average prediction accuracy at 28.5 percent and *rand* almost always achieves 50 percent accuracy. The low accuracy of *fixed* classifier is due to the fact that at instruction-level, it always predicts C_e for all cycles because examined clock periods are all smaller than instruction-level timing delay. It only considers the worst-case scenario to set its instruction-level timing delay. Since *fixed* always predicts C_e when the examined clock period is smaller than instruction-level delay, its predicted value-level reliability is always close to 0. This will severely deviate from the ground truth reliability when the TER is mild. Thus, we further compare these models on more FUs by utilizing them to predict the reliability as presented in Tables 8 and 9.

Before getting to the result, we introduce the evaluation metric on assessing the accuracy of reliability predictions: absolute error, as follows

$$AE = |reli_{pred} - reli_{gls}|, \tag{12}$$

where $reli_{pred}$ is the predicted reliability while $reli_{gls}$ is the ground truth reliability derived by **GLS**. This metric defines the difference between the predicted value and the "true" value, so a smaller value means a better performance.

Tables 8 and 9 compare the accuracy of three models. For INT_MUL, **CLIM** achieves AE between 0.5-4.7 percent while *fixed* and *rand* achieve 65.1-91.6 percent and 15.0-49.9 percent respectively. For FP_MUL, **CLIM** achieves AE between 0.4-6.2 percent while *fixed* and *rand* achieve 69.9-89.9 percent and 19.9-39.9 percent respectively. The low accuracy of *fixed* is due to the fact that at the three CPRs, the TERs are approximately 10, 20, and 30 percent respectively and *fixed* always predicts 0 reliability, leading to a huge difference. This indicates that only considering the worst-case instruction-level delay to predict timing errors could lead to a huge deviation from the real scenario, which might be even worse than a random guess. Meanwhile, **CLIM** demonstrates its robustness with average AE at 2.8 percent.

5.5 CLIM Efficiency

We compare the **CLIM** speed with GLS. On average across all datasets and FUs, **CLIM** computes 173X faster than GLS. The more complex the circuit structure, the slower speed for simulation. But this might not apply to **CLIM**, because it processes input data according to its own rule, which might not scale up with the complexity of the circuit structure. For previous instruction-level models [41], the authors claim that the **GLS** is very time-consuming and becomes a bottleneck for research purpose. Thus, **CLIM** provides a faster alternative way to examine reliability without performing time-consuming conventional **GLS**.

TABLE 7
Value-Level CLIM on FP_ADD for Timing Error Prediction

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	95.6%	9.7%	50.2%	93.2%	20.1%	50.1%	92.3%	67.0%	49.8%
sobel	95.3%	33.8%	49.9%	88.8%	39.9%	50.1%	92.2%	48.8%	49.9%
gauss	97.1%	9.6%	49.9%	94.2%	11.9%	50.0%	93.3%	15.6%	49.8%

TABLE 8
Value-Level CLIM on INT_MUL for Reliability Prediction Using AE

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	0.9%	89.9%	49.9%	0.6%	79.4%	29.4%	0.5%	70.2%	20.3%
sobel	2.4%	91.6%	41.6%	1.8%	84.4%	34.4%	4.6%	69.2%	19.2%
gauss	0.6%	89.8%	39.8%	3.1%	81.6%	31.5%	4.7%	65.1%	15.0%

6 Discussion

Variability Consideration: This paper mainly focuses on modeling timing error based on dynamic path sensitization behaviors caused by input operands. Therefore, it does not consider hardware variability effects such as PVTA variation on timing errors. Our previous work [30] did establish a timing error model by considering hardware variability, which is orthogonal to our approach by considering the input stimuli. Therefore, these two approaches can be combined to provide a more holistic model.

Potential Usage: The machine learning approaches proposed in this paper can also be used to predict the timing errors for a different implementation of circuits, such as approximate adders [20]. On the other hand, the model could be utilized online to guide dynamic frequency scaling (DFS) with an efficient physical implementation. Recently, a voltage-droop induced delay prediction model has been implemented using SVM to guide online DFS [42], whose hardware overhead is 1.5 percent for today's processor design. We expect the overhead of **CLIM** is less than such a model, since by comparison Table 1 shows that SVM computing time is more than 7000X of RFC model.

Potential Limitation: The main limitation of such a learning-driven method is that it only works for arithmetic functional units. It is unclear whether it can work for other micro-architecture parts such as memory. This is because the advantage of machine learning is that it can learn the path sensitization based on input data pattern, which is the main factor that determine timing errors. But for memory, there is not a clear clue as to the source factors of its timing errors.

Potential Improvement: In fact, despite the fact that the deep neural networks recently achieve very high classification accuracy in various classification tasks, their implementations

TABLE 9
Value-Level CLIM on FP_MUL for Reliability Prediction Using AE

datasets	CPR1			CPR2			CPR3		
	CLIM	fixed	rand	CLIM	fixed	rand	CLIM	fixed	rand
random	3.3%	89.9%	39.9%	3.9%	79.9%	29.9%	3.3%	70.5%	20.5%
sobel	0.4%	89.7%	39.7%	4.0%	80.0%	30.0%	0.5%	69.9%	19.9%
gauss	4.2%	89.9%	39.9%	5.8%	79.5%	29.5%	6.2%	70.9%	20.9%

require massive amount of hardware resources that limits their usage for on-chip monitoring. Alternative brain-inspired learning method such as hyperdimensional computing [32] enables fast and one-shot learning with low cost binary components, and exhibits reliable operations under extreme low signal-to-noise ratio conditions hence it can be efficiently implemented for on-chip online usage.

7 Conclusion and Future Work

CLIM is a supervised learning-based model to predict timing errors of functional units at two granularities: the bit-level and the value-level. It considers the impact of input operands on dynamic path sensitization (and hence timing errors). We perform gate-level simulation on a post-layout netlist to extract timing errors and useful 'features' from input data and circuit activity. We then apply a random forest classification method to construct the model with extracted input features and output labels. We considered input workload, computation history, and circuit toggling as input features to construct **CLIM**. For a given input data and circuit parameter, **CLIM** predicts the output to be one of two classes: {*timing correct*, *timing erroneous*}. On average across several FUs and CPRs, its bit-level and value-level prediction accuracy are 97 and 95 percent respectively. We utilize **CLIM** in estimating error-tolerant application output quality, achieving an average of 97 percent accuracy. **CLIM**-based reliability estimation is within 2.8 percent deviation on average of detailed gate-level simulation.

Our ongoing work seeks to improve the efficiency of model building by using efficient and more advanced learning methods.

Acknowledgments

The authors would like to thank the support from the NSF Expedition in Computing grant CCF-1029783 and NXP Semiconductors.

References

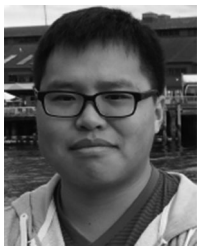
[1] AMD app SDK v2.5. (2011). [Online]. Available: http://www.amd.com/stream
[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.
[3] S. Borade, B. Nakiboğlu, and L. Zheng, "Unequal error protection: An information-theoretic perspective," *IEEE Trans. Inform. Theory*, vol. 55 no. 12, pp. 5511–5539, Dec. 2009.
[4] K. Bowman, et al., "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 44 no. 1, pp. 49–63, Jan. 2009.
[5] K. Bowman, et al., "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 46 no. 1, pp. 194–208, Jan. 2011.
[6] M. Bushnell and V. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Berlin, Germany: Springer Science & Business Media, 2004, vol. 17.
[7] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," *ACM SIGPLAN Notices*, vol. 48, pp. 33–52, 2013.
[8] K. Chae, S. Mukhopadhyay, C.-H. Lee, and J. Laskar, "A dynamic timing control technique utilizing time borrowing and clock stretching," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2010, pp. 1–4.
[9] H. Cho, L. Leem, and S. Mitra, "ERSA: Error resilient system architecture for probabilistic applications," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31 no. 4, pp. 546–558, Apr. 2012.
[10] M. R. Choudhury, V. Chandra, R. C Aitken, and K. Mohanram, "Time-borrowing circuit designs and hardware prototyping for timing error resilience," *IEEE Trans. Comput.*, vol. 63 no. 2, pp. 497–509, Feb. 2014.
[11] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *Proc. Des., Autom. Test Eur. Conf. Exhibition*, 2015, pp. 381–386.
[12] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. 13 no. 1, pp. 21–27, Jan. 1967.
[13] F. De Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Des. Test Comput.*, vol. 28, no. 4, pp. 18–27, Jul.–Aug. 2011.
[14] D. Ernst, et al., "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, 2003, pp. 7–18.
[15] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 449–460.
[16] M. Fojtik, et al., "Bubble razor: An architecture-independent approach to timing-error detection and correction," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2012, pp. 488–490.
[17] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wei, and D. M. Brooks, "An event-guided approach to reducing voltage noise in processors," in *Proc. Conf. Des. Autom. Test Eur.*, 2009, pp. 160–165.
[18] F. Hameed, M. A. A. Faruque, and J. Henkel, "Dynamic thermal management in 3D multi-core architecture through run-time adaptation," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2011, pp. 1–6.
[19] K. He, A. Gerstlauer, and M. Orshansky, "Circuit-level timing-error acceptance for design of energy-efficient DCT/IDCT-based systems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23 no. 6, pp. 961–974, Jun. 2013.
[20] X. Jiao, V. Camus, M. Cacciotti, Y. Jiang, C. Enz, and R. K. Gupta, "Combining structural and timing errors in overclocked inexact speculative adders," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2017, pp. 482–487.
[21] X. Jiao, Y. Jiang, A. Rahimi, and R. K. Gupta, "SLoT: A supervised learning model to predict dynamic timing errors of functional units," in *Proc. Des., Autom. Test Eur. Conf. Exhibition, 2017*, pp. 1183–1188.
[22] X. Jiao, et al., "Supervised learning based model for predicting variability-induced timing errors," in *Proc. 13th Int. New Circuits Syst. Conf.*, 2015, pp. 1–4.
[23] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 820–825.
[24] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. 15th Asia South Pacific Des. Autom. Conf.*, 2010, pp. 825–831.
[25] M. Kharitonov, "Cryptographic hardness of distribution-specific learning," in *Proc. 25th Annu. ACM Symp. Theory Comput.*, 1993, pp. 372–381.
[26] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, fourier transform, and learnability," *J. ACM*, vol. 40 no. 3, pp. 607–620, 1993.
[27] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels," in *Proc. ACM Int. Conf. Object Oriented Program. Syst. Languages Appl.*, 2014, pp. 309–328.
[28] P. Ndai, et al., "Trifecta: A nonspeculative scheme to exploit common, data-dependent subcritical paths," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 18 no. 1, pp. 53–65, Jan. 2010.
[29] F. Pedregosa, et al.,"Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[30] A. Rahimi, L. Benini, and R. K. Gupta, "Hierarchically focused guardbanding: An adaptive approach to mitigate PCT variations and aging," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2013, pp. 1695–1700.
[31] A. Rahimi, L. Benini, and R. K. Gupta, "Application-adaptive guardbanding to mitigate static and dynamic variability," *IEEE Trans. Comput.*, vol. 63 no. 9, pp. 2160–2173, Sep. 2014.
[32] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proc. Int. Symp. Low Power Electronics Des.*, 2016, pp. 64–69.
[33] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini, "A variability-aware OpenMP environment for efficient execution of accuracy-configurable computation on shared-FPU processor clusters," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, 2013, Art. no. 35.

[34] S. Roy and K. Chakraborty, "Predicting timing violations through instruction-level path sensitization analysis," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 1074–1081.
[35] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, pp. 164–174, 2011.
[36] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Trans. Semicond. Manuf.*, vol. 21 no. 1, pp. 3–13, Feb. 2008.
[37] Y. Tamir and M. Tremblay, "High-performance fault-tolerant VLSI systems using micro rollback," *IEEE Trans. Comput.*, vol. 39 no. 4, pp. 548–554, Apr. 1990.
[38] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy, "b-HiVE: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, Art. no. 105.
[39] R. Ubal, B. Janscikitg, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.*, 2012, pp. 335–344.
[40] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD birds 200," California Institute of Technology, 2010.
[41] J. Xin and R. Joseph, "Identifying and predicting timing-critical instructions to boost timing speculation," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 128–139.
[42] F. Ye, F. Firouzi, Y. Yang, K. Chakrabarty, and M. B. Tahoori, "On-chip droop-induced circuit delay prediction based on support-vector machines," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35 no. 4, pp. 665–678, Apr. 2016.

Xun Jiao received the dual bachelor's degree from the Beijing University of Posts and Telecommunications, China and the Queen Mary University of London, United Kingdom, in 2013. He is currently working toward the PhD degree with the University of California, San Diego. His research interests include error-tolerant computing and machine learning.

Abbas Rahimi received the BS degree in computer engineering from the University of Tehran, Tehran, Iran, the MS and PhD degrees in computer science and engineering from the University of California San Diego, La Jolla, CA, in 2010 and 2015. He is currently working toward the postdoctoral degree in the Department of Electrical Engineering and Computer Sciences, the University of California Berkeley, Berkeley, CA. He is a member of the Berkeley Wireless Research Center and collaborating with UC Berkeley's Redwood Center for Theoretical Neuroscience. His research interests include brain-inspired computing, approximate computing, massively parallel integrated architectures, embedded systems and software with an emphasis on improving energy efficiency and robustness. His doctoral dissertation has been selected to receive the 2015 Outstanding Dissertation Award in the area of "New Directions in Embedded System Design and Embedded Software" from the European Design and Automation Association (*EDAA*). He has also received the Best Paper at BICT, 2017, and the Best Paper Candidate at DAC, 2013.

Yu Jiang received the BS degree in software engineering from the Beijing University of post and telecommunication, beijing, China, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2010 and 2015. He is currently working toward the Postdoc researcher degree in the department of computer science of University of Illinois at Urbana-Champaign, IL. His current research interests include domain specific modeling, formal computation model, formal verification and their applications in embedded systems, safety analysis and assurance of cyber-physical system.



Jianguo Wang received the bachelor's degree from Zhengzhou University, China, and the Mphil degree in computer science from The Hong Kong Polytechnic University, in 2009 and 2012. He is currently working toward the PhD degree with the University of California, San Diego. His research interests include data management system and new computing hardware.

Hamed Fatemi received the BSc and MSc degrees from the Electrical and Computer Engineering Department of the University of Tehran, Tehran, Iran, and KNT, and the PhD degree in computer architecture from the Eindhoven University of Technology, Eindhoven, The Netherlands in 1998, 2001, and 2007, respectively. He is an Innovation lead / Department manager at NXP Semiconductors. His research interests include the areas of low-power design, multi-processors, heterogeneous and reconfigurable systems, and variability tolerance design. Fatemi has authored and co-authored more than 25 US patents, scientific publications and presentations

Jose Pineda de Gyvez is a fellow at NXP Semiconductors where he coordinates R&D efforts on low power design technologies. His industrial responsibilities are positioned in the interface between design and technology. He also holds the professorship Resilient Nanoelectronics (parttime) in the Department of Electrical Engineering, the Eindhoven University of Technology, The Netherlands. This professorship fills a gap between industry and academia by bringing industrial knowledge into classrooms, and open innovation into NXP. He was a faculty member in the Department of Electrical Engineering, Texas A&M University. He has been associate editor of several the IEEE Transactions and is often involved in program and steering committees of international symposiums. He is also a member of the editorial board of the *Journal of Low Power Electronics*. He has more than 150 publications in the fields of low power IC design, analog signal processing, and design for manufacturability and test. He is (co)-author of four books, and has more than 20 US granted patents. He is a fellow of the IEEE.

Rajesh K. Gupta received the BTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, the MS degree in electrical engineering and computer science from the University of California, Berkeley, and the PhD degree in electrical engineering from Stanford University, California, in 1984, 1986, and 1994. He is a professor of computer science and engineering with the University of California, San Diego (UCSD), La Jolla, and holds the Qualcomm endowed chair. His research interests span topics in embedded and cyber-physical systems with a focus on energy efficiency from algorithms, devices to systems that scale from IC chips, and data centers to built environments such as commercial buildings. He currently leads NSF project MetroInsight with the goal to organize and use city-scale sensing data for improved services. His past contributions include SystemC modeling and SPARK parallelizing high-level synthesis, both of which have been incorporated into industrial practice. Earlier, he led NSF Expeditions on Variability, and DARPA-sponsored efforts under the Data Intensive Systems (DIS) and Circuit Realization at Faster Timescales (CRAFT) programs. He and his students have received a best demonstration paper award at ACM BuildSys'16, best paper award at IEEE/ACM DCOSS08 and a best demonstration award at IEEE/ACM IPSN/SPOTS05. He currently holds INRIA International Chair at the French international research institute in Rennes, Bretagne Atlantique. He is a fellow of the IEEE and the ACM.