SLoT: A Supervised Learning Model to Predict Dynamic Timing Errors of Functional Units

Xun Jiao[‡], Yu Jiang[§], Abbas Rahimi[*], and Rajesh K. Gupta[‡]
[‡]Department of Computer Science and Engineering, UC San Diego, La Jolla, CA, USA
[§] School of Software, Tsinghua University, Beijing, China
[*]Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, CA, USA
{xujiao, gupta}@cs.ucsd.edu, jy1989@mail.tsinghua.edu.cn, abbas@eecs.berkeley.edu

***Abstract*—Dynamic timing errors (DTEs), that are caused by the timing violations of sensitized critical timing paths, have emerged as an important threat to the reliability of digital circuits. Existing approaches model the DTEs without considering the impact of input operands on dynamic path sensitization, resulting in loss of accuracy. The diversity of input operands leads to complex path sensitization behaviors, making it hard to represent in DTE modeling.**

In this paper, we propose `SLoT`, a supervised learning model to predict the output of functional units (FUs) to be one of two timing classes: {*timing correct*, *timing erroneous*} as a function of input operands and clock period. We apply random forest classification (RFC) method to construct `SLoT`, by using input operands, computation history and circuit toggling as input features and outputs' timing classes as labels. The outputs' timing classes are measured using gate-level simulation (GLS) of a post place-and-route design in TSMC 45nm process. For evaluation, we apply `SLoT` to several FUs and on average 95% predictions are consistent with GLS, which is 6.3X higher compared to the existing instruction-level model. `SLoT`-based reliability analysis of FUs under different datasets can achieve 0.7-4.8% average difference compared with GLS-based analysis, and execute more than 20X faster than GLS.

I. Introduction

With the continuous scaling of CMOS technology, timing violations-induced DTEs have emerged as an important threat to reliability. Increasing variability caused by process, voltage, temperature and aging (PVTA) in advanced processes further exacerbates this problem. Conventional synchronous design typically applies conservative clock period based on static timing analysis (STA) to ensure operations without timing errors. This results in unnecessary loss of performance because the DTEs might not occur even if the clock period does not meet STA constraints because the critical path is not always sensitized. Therefore, to improve performance, adaptive techniques have been proposed to predict DTEs in advance and adjust operating frequency to prevent such DTEs.

One typical approach is to predict the DTEs based on instruction type [4] [14] [17]. It characterizes the maximum timing delay of each instruction type from a set of selected representative benchmarks. The instruction-level model will predict DTEs if the maximum instruction-level timing delay is beyond clock period. However, these instruction-level models predict DTEs based on the maximum measured timing delay, a worst-case scenario of sensitized timing delay that overlooks the effect of input operands on dynamic path sensitization behaviors, leading to a pessimistic modeling. Actually, the instruction-level timing delay does not only rely on the instruction type, but also its input operands, because the dynamic path sensitization is directly affected by input operands. This is seen in that one instruction might exhibit different DTE behaviors when they execute different input operands [17]. Prediction of DTEs is a difficult problem for the space of dynamic timing characterization, instructions and operands is large. Our attempt to raise the abstraction level at which this characterization and prediction takes place to microarchitectural level faces following challenges:

Challenge 1: dynamic path sensitization is potentially affected by a large number of parameters, from operand values, instruction types and data dependencies. 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 DTE models

Challenge 2: there might be numerous failed timing paths in the design, and the DTE might be caused by any one of them. It is unclear how these features will determine what paths to sensitize and inducing 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 hard [16] even under uniform distribution over the inputs [11].

Proposed approach: To overcome these challenges, we propose **`SLoT`**, a supervised learning model, to predict the DTEs of FUs based on the input operands (workload) and clock period. The key idea of **`SLoT`** is to establish a prediction model that can best explore the relationship from input features to sensitized critical paths by learning the existing patterns and their corresponding output classes. For a given input data and clock period reduction (CPR) from safe clock period, **`SLoT`** predicts output data to be one of two predefined classes– {*timing correct, timing erroneous*}.

First, we measure the DTEs at each cycle to generate output class labels using GLS of post-layout design 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 **`SLoT`** for four FUs: (INT_ADD, FP_ADD, INT_MUL, FP_MUL) with extracted input features and output class labels. Third, we evaluate the prediction accuracy of **`SLoT`** by comparing its predicted results with GLS-based ground truth.

Contributions: This paper makes the following contributions:

- We build a DTEs extraction module using standard ASIC design flow and post-layout GLS to analyze the sources of DTEs and the effect of input operands. Based on which, we can extract the useful input features to train the model.
- We propose a methodology to construct automatic models for DTEs prediction using supervised learning methods by taking input operands, clock period and circuit activity into account.
- We demonstrate the performance of **SLoT** in its average prediction accuracy at 95% across several FUs and CPRs, which is 6.3X higher than the instruction-level model. Furthermore, we demonstrate the usage of **SLoT** in reliability analysis of FUs under the input data profiled from real-world applications, which achieves 0.7-4.8% average difference compared with GLS-based reliability analysis, and executes 20X faster than GLS.

II. Problem Formulation

Problem Formulation: We follow the procedure of representing the DTE 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 DTEs. The output value in error are $\psi(I) = \phi_i(I) \oplus \phi_r(I)$, 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 to evaluate a sequence of *non-parametric* classification methods to classify the inputs to map into different outputs as shown in Section III-B.

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 define the two classes for output value: C_e representing *timing erroneous* and C_c representing *timing correct*, and we define the class of $y[t]$ as $C[t]$. 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. Our goal is to learn the output class $C[t]$ at cycle t as a function of input workload, clock period and FU type, denoted as follows:

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

where t_{clk} is the clock period, FU_{type} is FU type, $x[t]$, $x[t-1]$, ... $x[1]$ are the input workload at cycle $t, t-1, ...1$. The reason of putting previous (history) inputs is that we do not know whether previous input workload will have an effect on the DTE behavior of current cycle t. Therefore, we need to investigate the features from input data which affect the output DTE behaviors, as shown in Section III-B. In summary, this becomes a binary classification problem: for a given input data and circuit parameters at cycle $t, t-1, ...1$, **SLoT** predicts the output $C[t]$ to be one of two classes: C_c or C_e.

III. SLoT Model

SLoT Model: It is comprised of three phases as shown in Fig. 1: *DTEs Extraction, Model Training* and *Model Evaluation*. a) The *DTEs Extraction* phase implements the standard ASIC flow and uses GLS to generate output values $y[t]$ under given input data and circuit parameters. Then the output value is compared with clean output data generated from pure-RTL simulation to determine the timing class: C_c if matched, otherwise C_e. Such output class labels will be used in the next *Model Training* phase. 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, **SLoT** predicts the timing class of the FU output value for a given test input data and circuit parameters, and then compare the predicted result with GLS ground truth to compute prediction accuracy. More details about the three phases are illustrated as follows.

A. DTEs Extraction

We use integer FUs as well as floating point FUs which can provide more complex circuit structure compared to their integer counterpart. We change the data types and circuit structures to better evaluate the robustness of our model. We use FloPoCo [5] to generate the synthesizable VHDL codes of FUs with wrapper at input and output ports. 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 Standard Delay Format (SDF) file. We select the operating voltage to be 0.85V and temperature to be 50°C. 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 $y[t]$. Finally, we compare the simulated output data $y[t]$ with pure-RTL simulation output data $y_{gold}[t]$ to generate $C[t]$, where $t = 1, 2, 3, ...N$.

The input stimuli of simulation comes from two sources: Python-written random data generator and the application input data profiled using *Multi2Sim* [15], a cycle-accurate CPU-GPU heterogeneous architectural simulator. In every cycle, we provide input stimuli operands to the GLS and generate output $y[t]$ at cycle t, which will then be classified into either C_c or C_e based on definitions in Section II. These generated output timing class labels will then be used in *Model Training* phase described in Section III-B. We perform this experiment under various circuit parameters and input data.

B. Model Training

Data Preprocessing: We preprocess the random input data to convert it into correct format, for example, 0.5 should be converted to 00111111000000000000000000000000 if the FU is of IEEE-754 single-precision format. The reason of 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

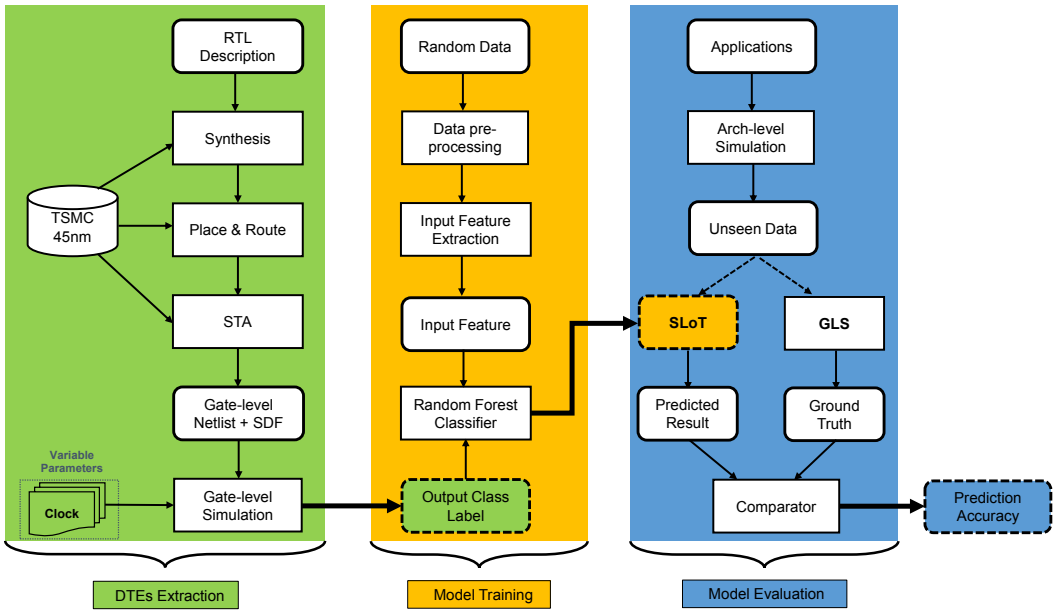

Fig. 1. **SLoT** model overview with three key stages: a) DTEs Extraction to examine the DTEs under different input workloads and CPRs, and to generate output timing class labels to be used in next stage; b) Model Training to apply random forest classification (RFC) to construct **SLoT** with extract input features from preprocessed data and output timing class labels generated from last stage; c) Model Evaluation to evaluate **SLoT** prediction accuracy by comparing its predicted results with GLS-generated ground truth, under different benchmarks datasets.

model training, we take each bit value to compose input features rather than the decimal value alone.

To save training effort, we remove the repetitive pair of $\{x[t-1], x[t]\}$ in the training input data, 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 timing violation occurs at cycle t, the output could still appear to be correct.

Input Feature Extraction: Then, from the processed training input data, we need to find out the useful input features that determines 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 will have 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 vary 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 effect 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 like above. We use this to evaluate the effect of deeper history.

We use 100K cycles for simulation and use different CPRs. In Case 1, we found the timing class $C[t]$ varies irregularly. More specifically, by comparing every two neighboring output, e.g., $y[30]$ and $y[50]$, we found 44% neighboring pairs exhibits different timing classes. In Case 2, we found all output timing class $C[t]$ exhibits exactly the same behaviors, i.e., all C_c or C_e. This 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. This is reasonable that the preceding input workload set a state for the circuit, and then the current input toggles nets based on the current state. Therefore, path sensitization or nets toggling activity depends on both the current circuit state and current circuit input. Therefore, we find out the sources that determines the dynamic path sensitization behaviors and consider that as model input features.

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 output vectors. The circuit output timing errors occur if and only if at least one output bit location faces timing violation. The timing violation of a particular bit occurs only when there is at least one sensitized timing path ended in that bit faces violation. The necessary condition for a path being sensitized is that all nodes along that path are toggled [3]. Hence, if the end point, i.e., the output bit, is not toggled, the path is not sensitized. In other words, only the toggled output bit could cause a timing violation to sensitized path. Thus, we also take the final output value into our modeling as part of input feature. Note that, we take the pure-RTL clean output value $\{y_{gold}[t-1], y_{gold}[t]\}$ as input features. In summary, by composing aforementioned features, our final input features are $\{x[t-1], x[t], y_{gold}[t-1], y_{gold}[t]\}$.

Random Forest Classifier(RFC): We evaluate several commonly used supervised learning classification methods: k-nearest neighbor (k-NN), support vector machine (SVM), logistic regression (LR), and random forest classifiers (RFC) for their increased sophistication and practical use, from Scikit-learn package [12]. Table.I presents the prediction accuracy,

training and testing time of four methods using 100K random training data and 10K random test data. LR is fastest because of its relatively easy computation process by assigning weight to each bit position. However, it achieves lowest accuracy because the contribution of each bit position is not linear to final output and might change if other bits changed. KNN makes predictions based on the distance between input vectors, which overlooks the different significance (effects) of different bit positions on final output. Although SVM achieves good accuracy but its long running time impedes its use. Finally, we choose RFC due to its high accuracy, fast computing time and superior interpretability.

TABLE I
PREDICTION ACCURACY, TRAINING AND TESTING TIME OF FOUR LEARNING METHODS.

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

RFC is an ensemble method that fits a number of decision tree (DT) classifiers and use averaging to reduce the overfitting problem of DT classifier. In this case, each bit position contributes to the final output timing behavior differently, and could affect each other. This situation is fit for using decision tree because a decision rule could be formed by learning the joint contribution of different bit positions.

Training Process: As discussed above, we set $\{x[t-1], x[t], y_{gold}[t-1], y_{gold}[t]\}$ as input feature and $C[t]$ as output class labels, where $x[t]$, $y_{gold}[t]$, $x[t-1]$, and $y_{gold}[t-1]$ are input and output binary vectors at cycle t and $t-1$, and $C[t] \in \{C_e, C_r\}$ is output timing class provided by *DTEs Extraction* phase. We then apply RFC to 1M random training input data and corresponding output timing class labels to construct a binary classification model.

C. Model Evaluation

For a given input workload and circuit parameter, our model will predict the corresponding output timing class to indicate whether there is a timing violation. We evaluate the model performance using *prediction accuracy*, and compare it with baseline models.

1) Evaluation Metric: We derive prediction accuracy by comparing the **SLoT** predicted result with GLS results:

$$prediction\_accuracy = \frac{\#matched\_cycles}{\#total\_cycles} \quad (2)$$

where $\#total\_cycles$ is the number of total simulation cycles, and $\#matched\_cycles$ is the number of cycles at which predicted result matched GLS result, i.e., both results are either C_c or C_e.

2) Comparison Methods: We compare **SLoT** against following baseline methods which can help us evaluate the true performance of our model:

- **INST-level:** this model is consistent with the instruction-level model used in [4] [14] where the model will predict C_e when the clock period does not meet the measured maximum instruction-level timing delay, otherwise C_c.
- **SLoT-NH:** this model is trained similarly with **SLoT** except it does not consider preceding cycle input history as input features, i.e., it only considers $\{x[t], y_{gold}[t]\}$ as input features. Since it only consider current cycle output, it does not consider output toggling as well.
- **SLoT-NT:** this model is trained similarly with **SLoT** except it does not consider circuit output toggling as input features, i.e., it only considers $\{x[t-1], x[t]\}$ as input features.

IV. EXPERIMENTAL RESULTS

In this section, we characterize FUs timing behavior under five different CPRs. Then, we present the prediction accuracy of **SLoT** model and compare with the baseline models. Finally, we use **SLoT** to analyze the reliability of FUs under three benchmark datasets and compare with GLS.

A. Hardware Characterization

We use the *DTEs Extraction* described in Section III-A to investigate the timing characteristics of four FUs under the 1M random input stimuli. Fig. 2 presents five different CPRs that would lead to the timing error rates (TERs) of FUs at 5%, 10%, 15%, 20% and 25% respectively, where TER is calculated as $\#erroneous\_cycles/\#total\_cycles$. From now on, we refer the CPR pairs which lead to such five TERs as {*CPR1, CPR2,...CPR5*}. Note that such CPR pairs are different for each FUs. We observe several important facts from Fig. 2.

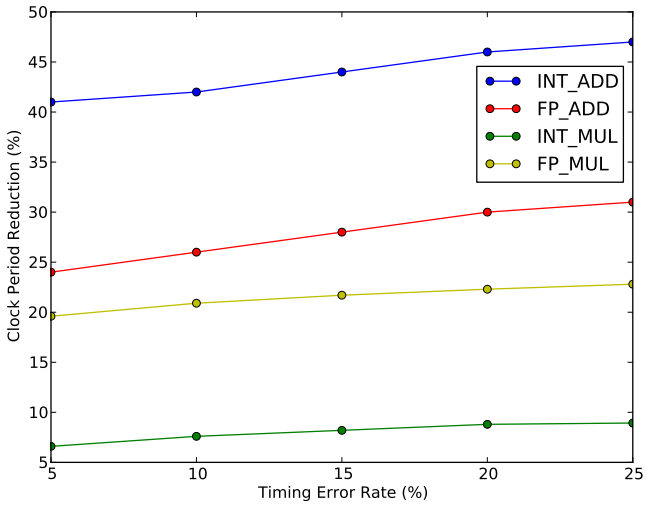

Fig. 2. The clock period reduction (CPR) and corresponding timing error rate (TER) under 1M random data.

First, for INT_ADD, 5% TER is caused by 41% CPR, meaning that 41% timing margin is used to protect 5% timing violations. This suggests a large timing margin has been used for worst-case scenarios. Second, TER increases rapidly after then: TER increases from 5% to 20% when CPR only increases 5%: from 41% to 46%. This suggests that there are many paths of similar lengths are sensitized in this delay range and this is consistent with the timing wall phenomenon [10]. When we compare FP_ADD timing characteristics with INT_ADD, we found there is difference and similarity. The

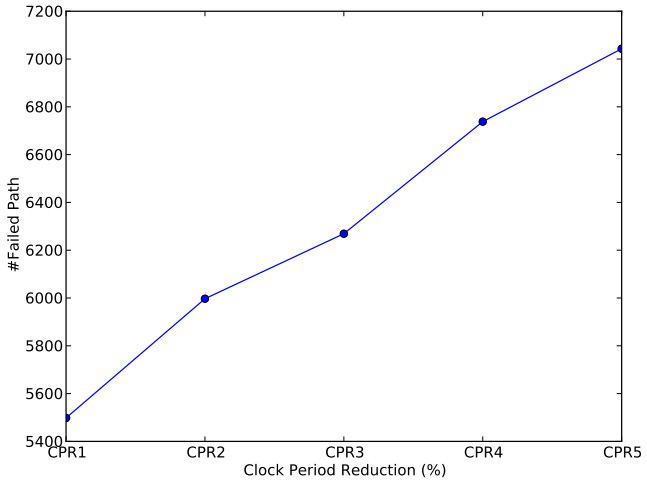

Fig. 3. The number of failed path of INT_ADD under different clock period reductions.

difference is that, for FP_ADD, 5% TER is caused by 24% CPR, meaning that the timing margin 'wasted' to protect the worst-case scenarios are less than that of INT_ADD. The similarity is that, the TER also rapidly increases after then: TER increases from 5% to 20% when CPR only increases 6%: from 24% to 30%. This is also consistent with the timing wall phenomenon. Both designs suggest that there are a large timing margin used to protect worst-case timing violation (5%) and emphasizes the need for accurate timing error model.

We also compute the number of paths with negative slack under such CPRs for INT_ADD. As illustrated in Fig.3, the number of failed paths increases with the CPR. We note that, for every CPR point, there are more than 5K failed paths. This means that, once any path in this set fails, then the whole design faces timing violation. This corresponds to the challenge 2 in the Section I where multiple timing paths failure can lead to a timing violation but we need to learn whether any member of these failed paths will be sensitized. For the FP_ADD, even for the slightest TER at 5%, we observed more than 30K failed paths. **SLoT** needs to predict the timing violation given that even if there is only one path failure, which makes it a extremely difficult task to learn such path sensitization behaviors.

B. Model Accuracy

We run our experiments at the $\{CPR1, CPR2,...CPR5\}$. Table. II and Table. III presents the prediction accuracy of **SLoT** for INT_ADD and FP_ADD under random data, where we can see **SLoT** exhibits the prediction accuracy ranging between 91-99% and achieves average prediction accuracy at 95%. INST-level model achieves on average 15% prediction accuracy: it 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, that overlooks the effect of input operands on dynamic path sensitization hence dynamic timing delay. The **SLoT**-NH model achieves average prediction accuracy at 85% while **SLoT**-NT can achieve 90% on average. We found **SLoT**-NH achieves same accuracy as a naive classifier whose predictions are always C_c. This suggests that without considering history input workload, the classifiers does not learn anything useful. **SLoT**-NT achieves good accuracy, meaning that the effect of circuit toggling activity is not as important as history input workload. Thus, compared to these baseline classifiers, **SLoT** achieves 6.3X, 12% and 5% higher accuracy respectively.

TABLE II
PREDICTION ACCURACY FOR INT_ADD UNDER RANDOM DATA.

CPR	CPR1	CPR2	CPR3	CPR4	CPR5
INST-level	5%	10%	15%	20%	25%
SLoT-NH	95%	90%	85%	80%	75%
SLoT-NT	96%	92%	88%	85%	83%
SLoT	99%	96%	94%	92%	91%

TABLE III
PREDICTION ACCURACY FOR FP_ADD UNDER RANDOM DATA.

CPR	CPR1	CPR2	CPR3	CPR4	CPR5
INST-level	5%	10%	15%	20%	25%
SLoT-NH	95%	90%	85%	80%	75%
SLoT-NT	97%	96%	95%	93%	93%
SLoT	97%	96%	95%	93%	94%

C. Reliability Analysis

To further evaluate the robustness of **SLoT**, we vary the input datasets and use two new FU types: INT_MUL and FP_MUL. We present a case study that uses **SLoT** to analyze the reliability of INT_MUL and FP_MUL under three different input datasets: random and two image processing applications selected from AMD APP SDK v2.5 [1]: Gaussian filter and Sobel filter. We profile the input operands of INT_MUL and FP_MUL using *Multi2Sim* with images from Caltech-UCSD Birds 200 vision dataset [16]. Using these input datasets, we analyze the reliability of the two FUs using two ways: GLS and **SLoT**. The GLS-based analysis is used as ground truth. Table.IV and Table.V present the reliability analysis based on **SLoT** and GLS, where we observe several important facts. First, on average across all datasets, **SLoT**-based analysis is within 0.7-4.8% of detailed GLS and the average difference is around 2.5%. **SLoT**-based reliability analysis remains small difference compared with ground truth when circuit types changed from INT_MUL to a more complex structure FP_MUL, which demonstrate the robustness of such model across different circuit structures. Second, across all datasets, **SLoT**-based analysis computes 21-4600X faster than GLS-based analysis. The more complex of the circuit structure, the slower speed is for simulation. But this might not apply to **SLoT**, because it process input data according to its own rule which might not scale up with the complexity of circuit structure. For previous instruction-level models [17], the authors claim that the GLS is very time-consuming, that becomes a bottleneck for research purpose. Thus, **SLoT** provides a faster alternative way to examine reliability without performing time-consuming conventional GLS.

V. RELATED WORK

Better-than-worst-case (BTWC) design methods have been explored to overscale frequency while ensuring circuit reli-

TABLE IV
SLoT-BASED RELIABILITY RESULT AND GLS-BASED RELIABILITY RESULT FOR INT_MUL UNDER FIVE CPRS.

Dataset	CPR1		CPR2		CPR3		CPR4		CPR5		Ave diff%	Computing time	
	SLoT	GLS	**SLoT**	GLS	**SLoT**	GLS	**SLoT**	GLS	**SLoT**	GLS		SLoT	GLS
random	94.2%	94.8%	89.0%	89.9%	84.1%	84.8%	78.8%	79.4%	73.7%	74.2%	0.7%	6.87s	213s
sobel	94.0%	92.8%	94.0%	91.6%	87.4%	85.0%	86.2%	84.4%	77.8%	74.2%	2.3%	0.01s	46s
gauss	95.1%	94.1%	90.4%	89.8%	85.2%	82.8%	84.7%	81.6%	73.8%	69.1%	2.4%	0.08s	47s

TABLE V
SLoT-BASED RELIABILITY RESULT AND GLS-BASED RELIABILITY RESULT FOR FP_MUL UNDER FIVE CPRS.

Dataset	CPR1		CPR2		CPR3		CPR4		CPR5		Ave diff%	Computing time	
	SLoT	GLS	**SLoT**	GLS	**SLoT**	GLS	**SLoT**	GLS	**SLoT**	GLS		SLoT	GLS
random	92.6%	94.9%	86.6%	89.9%	81.1%	84.9%	76.0%	79.9%	70.9%	74.5%	3.3%	3.9s	190s
sobel	93.8%	94.7%	90.1%	89.7%	88.0%	84.8%	84.0%	80.0%	74.7%	75.0%	1.8%	27.5s	579s
gauss	91.0%	94.9%	85.7%	89.9%	80.9%	84.7%	73.7%	79.5%	68.6%	74.8%	4.8%	3.8s	159s

ability [2]. It uses correction schemes to recover the DTEs induced by frequency overscaling. A shadow flip-flop was used in [6] to detect and correct any timing errors induced by speculated voltage scaling. However, it could introduce hardware overheads and performance penalty by using online monitoring and correction schemes.

A less-intrusive way is to model the DTEs and adaptively change the clock frequency to prevent DTEs' occurrence. This mechanism ensures error-free operation in a proactive manner. A bit-level timing error rate prediction model for floating point units has been proposed considering the impact of workload, voltage and temperature on timing errors[9]. However, the scalability problem prevents its use because it need to develop a binary classifier for each bit position. Instruction-level models identifies critical instructions based on their vulnerability to timing errors and measured maximum delay [17] [4] and then perform an instruction-based dynamic frequency scaling to improve performance. They consider a worst-case scenario to set instruction-level timing delay. In a similar vein, a model is developed for critical instruction sequence, which considers the worst-case delay incurred by a sequence of instructions [13]. Machine learning has been used to predict the reliability of embedded system [8] [7]. In this work, we use machine learning to predict reliability of digital circuits.

In summary, our model is inspired by these early works but with major differences. **SLoT** looks into the effects of input operands on DTEs and incorporate them to the modeling. **SLoT** can be used to assist the modeling at higher levels by provide an accurate error model at lower circuit-level, to enable a more robust and fine-grained optimization.

VI. CONCLUSIONS

SLoT is a supervised learning-based model to predict dynamic timing errors of functional units. It considers the impact of input operands on dynamic path sensitization and hence timing errors, which is overlooked by previous models. 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 random forest classification method to construct the model with extracted input features and output labels. For a given input data and circuit parameter, **SLoT** predicts the output to be one of two classes: $\{$*timing correct, timing erroneous*$\}$. On average across several FUs and CPRs, its prediction accuracy is 95%. **SLoT**-based reliability estimation is within 0.7-4.8% of detailed gate-level simulation.

VII. ACKNOWLEDGMENTS

The authors would like to thank Dr. Hamed Fatemi and Dr. Jose Pineda de Gyvez from NXP Semiconductors for valuable discussions. This material is based upon the work supported by the National Science Foundations Variability Expedition in Computing under Award No. 1029783.

REFERENCES

[1] Amd app sdk v2.5. [online]. available: http://www.amd.com/stream.
[2] Keith Bowman et al. A 45 nm resilient microprocessor core for dynamic variation tolerance. *JSSC*, 2011.
[3] Hari Cherupalli et al. Graph-based dynamic analysis: Efficient characterization of dynamic timing and activity distributions. In *ICCAD*, 2015.
[4] Jeremy Constantin et al. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *DATE*. IEEE, 2015.
[5] Florent De Dinechin et al. Designing custom arithmetic data paths with flopoco. *IEEE Design & Test of Computers*, (4):18–27, 2011.
[6] Dan Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO-36.*, 2003.
[7] Yu Jiang et al. Bayesian-network-based reliability analysis of plc systems. *IEEE transactions on industrial electronics*, 2013.
[8] Yu Jiang et al. System reliability calculation based on the run-time analysis of ladder program. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 2013.
[9] Xun Jiao et al. Supervised learning based model for predicting variability-induced timing errors. In *Proc. of NEWCAS*. IEEE, 2015.
[10] Andrew B Kahng et al. Slack redistribution for graceful degradation under voltage overscaling. In *ASP-DAC*. IEEE, 2010.
[11] Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *STOC*. ACM, 1993.
[12] Fabian Pedregosa et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 2011.
[13] Abbas Rahimi et al. Application-adaptive guardbanding to mitigate static and dynamic variability. *Computers, IEEE Transactions on*, 2014.
[14] Sanghamitra Roy et al. Predicting timing violations through instruction-level path sensitization analysis. In *DAC*. ACM, 2012.
[15] Rafael Ubal et al. Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *Proc. of PACT*, Sep. 2012.
[16] Peter Welinder et al. Caltech-ucsd birds 200. 2010.
[17] Jing Xin et al. Identifying and predicting timing-critical instructions to boost timing speculation. In *MICRO*, 2011.