SoC Speed Binning Using Machine Learning and On-Chip Slack Sensors

Mehdi Sadi, *Student Member, IEEE*, Sukeshwar Kannan, LeRoy Winemberg, and Mark Tehranipoor, *Senior Member, IEEE*

***Abstract*—Speed binning of system-on-chips (SoCs) using conventional $F_{\max}$ test requires application of complex functional test patterns. Functional workload-based speed binning techniques incur high test-cost in terms of long test-time and complexity in functional test generation, and require high-end automatic test equipment. In this paper, we propose a novel speed binning flow that uses path timing slacks, extracted with robust digital embedded sensor IPs, of selected critical/near-critical paths. We apply machine learning techniques to model a predictor considering the extracted slacks and the $F_{\max}$ values from a set of randomly tested die during wafer sort. The trained predictor is used to obtain the $F_{\max}$ for the remaining chips. The proposed flow has been demonstrated in an SoC benchmark circuit at 28 nm technology. For sufficient number of training samples, $F_{\max}$ is correctly predicted for 99% of the prediction samples.**

***Index Terms*—$F_{\max}$ test, machine learning, slack sensor, speed-binning.**

I. Introduction

With aggressive technology scaling the transistor density per unit chip area has increased significantly over the past decade. This paved the way for many-core processors and highly integrated system-on-chips (SoCs). The increased variability in transistor parameters and workload-dependent fluctuations in operating conditions have made harnessing the full benefits of scaling a challenging task [1]. Variations can be broadly categorized into the following classes.

1) One-time static process variations due to manufacturing imperfections that cause transistor and interconnect parameters to drift from their designed values.
2) Run-time dynamic variations—power supply noise and temperature fluctuations—resulting shifts in operating conditions.

Manuscript received February 18, 2016; revised May 24, 2016; accepted August 13, 2016. Date of publication August 25, 2016; date of current version April 19, 2017. This work was supported in part by the National Science Foundation under Grant CCF-1565404, and in part by the Semiconductor Research Corporation under Award 2646. A preliminary version of this paper appeared in Proc. of ACM Great Lakes Symposium on VLSI (GLSVLSI) 2015 [20]. This paper was recommended by Associate Editor S. Reda.

M. Sadi and M. Tehranipoor are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: mehdi.sadi@ufl.edu; tehranipoor@ece.ufl.edu).

S. Kannan is with GLOBALFOUNDRIES, Malta, NY 12020 USA (e-mail: sukeshwar.kannan@globalfoundries.com).

L. Winemberg is with NXP Semiconductors, Austin, TX 78735 USA (e-mail: leroy.winemberg@nxp.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2602806

In terms of spatial locations, the variation can be segmented into die-to-die and within-die components. Variations impact transistor length, width, threshold voltage, oxide thickness, etc., all of which directly contribute to path delay variations. As a result, a certain path may show significant discrepancy in the delay between pre- and post-fabrication stages [2], [3] and the actual speed limiting paths might be masked in the simulation phase [4], [5].

As a result of path delay variations, the maximum operating frequency of a chip (e.g., microprocessors, DSPs, microcontrollers, and application specified integrated circuits) or the $F_{\max}$ varies from chip-to-chip and wafer-to-wafer. In order to sustain an acceptable yield rate without compromising performance, at the production test phase chips are binned according to the maximum functional or operating frequency. Chips in the higher bin are faster and sold at a higher profit. To accurately identify the $F_{\max}$ it is necessary to execute functional workloads or benchmarks and excite all the possible critical paths at increasingly higher clock frequencies until any of the capture flip-flops fail [6]. Execution of functional workload consumes a significant portion of tester time and memory. Moreover, this flow is required to be repeated at multiple frequencies to identify the actual $F_{\max}$. Additionally, an expensive automated test equipment (ATE) is required that is capable of running the test workloads at the comparatively higher $F_{\max}$ frequency range. In [7], experimental data on test time for server processors were presented to corroborate the fact that a major portion of total test time is associated with speed binning. Although functional $F_{\max}$ testing is the most appropriate method to identify the chip operating frequency or $F_{\max}$, the high cost and memory requirement of the ATE and the long test time associated with workload execution at multiple frequencies have motivated researchers to investigate alternative low-cost and faster techniques to identify chip $F_{\max}$ [8]–[11], [14]–[20].

A. Related Work

Structural test patterns—generally transition delay fault (TDF) and path delay fault (PDF) patterns—are widely used to test the integrity of circuit paths against defects. Since structural patterns are less expensive to generate and offer much higher fault coverage, there have been significant amount of research to utilize structural test results in estimating actual chip $F_{\max}$ [8]–[10]. Cory *et al.* [8] studied the correlation between the structural and functional $F_{\max}$ using sample chips, where a linear relation was established.

Brand *et al.* [9] demonstrated that structural $F_{\max}$ based on complex PDF patterns exhibited correlation to the actual functional $F_{\max}$. In [10] and [11], data learning techniques were used to build an $F_{\max}$ predictor from structural test results. However, in order to apply structural delay fault test patterns, the path under test must be testable. To make all the paths testable by structural test patterns, extra test points may be required to be inserted which incur area overhead [12]. Also structural test's correlation (i.e., direct linear correlation) to the functional-test $F_{\max}$ cannot be guaranteed for advanced technology nodes because of process variations [11], [19].

Researchers proposed the concept of built-in speed-grading [14] and built-in delay-binning [15] where the built-in self-test (BIST) hardware and on-chip programmable clock generator were used to apply BIST patterns at different frequencies by the chip itself, thus eliminating the need for expensive ATE and also reducing the test time. However, because of the pseudo-random nature of BIST patterns, these patterns may not excite the critical/near-critical paths [13]. As a result, the accuracy of this method in correlating to the chip $F_{\max}$ cannot be proven [19].

In [16] and [17], on-chip circuits were used to measure the propagation delay of critical paths and then the longest measured delay dictated the chip $F_{\max}$. Although direct measurement of critical path delays would eliminate the costly procedure of repeated application of the functional test at different frequencies, the large number of possible critical paths in modern SoC make it almost impossible to include that many path delay monitoring circuitry. Also, the preselected critical paths may not remain critical in each fabricated chip due to the increasing levels of process variations [3].

In [18] and [19], $F_{\max}$ obtained from functional test and data collected from on-chip ring oscillator (RO)-based process monitors were utilized in training machine learning predictors. Next, for rest of the samples, the RO data were fed to the trained predictors to obtain the $F_{\max}$. The reported speed-binning accuracy ranged from 90% to 93% in [18] and 87% to 93% in [19] based on silicon data collected from chips fabricated in 55 and 28 nm technology, respectively.

For chip timing data extraction, in [21], a systematic method to synthesize multiple design dependent RO (DDRO) circuits using standard cell gates is presented. Because of the design dependent aspect of the proposed RO, and the use of critical path cluster information in the design, the DDRO offers significant accuracy improvement over conventional inverter-based RO in delay estimation [21]. Since DDRO is a replica type monitor, it offers lower area overhead and design cost compared to the *in-situ* monitors. However, as discussed in [21], replica monitors such as DDRO can only capture global variations; whereas *in-situ* monitors are better suited for capturing local variations in addition to the global variations as they operate on actual circuit paths. Since our research goal requires the capture of timing slacks directly from a set of selected critical paths, instead of replica monitors we designed *in-situ* monitors that can record the slack values. In [22], an *in-situ* technique called SlackProbe was proposed that probes the intermediate nodes of critical paths to detect any impending timing failure within a monitoring window. SlackProbe was designed to detect possible timing failures—without recording the actual slack value—on the critical paths for the purpose of activating remedial countermeasures, whereas the goal of our designed slack sensor is to measure and record the timing slack of critical/near-critical paths for $F_{\max}$ estimation.

B. Our Contributions and Novelties

One of the primary objectives of this paper is to reduce the test time and cost associated with speed binning. With the aim of test cost and time reduction, we evaluate the application of machine learning-based predictive methodology in speed binning using extracted path slack. In our proposed flow, instead of testing testable paths with patterns which also require clock sweeping, we take a completely different approach—we measure the timing slack of selected number of possibly critical/near-critical paths with embedded sensors to estimate what $F_{\max}$ is. Contrary to [18] and [19], where standalone ROs are placed randomly at multiple locations inside the chip, our approach methodically places timing slack sensors on actual circuit paths spread across the chip layout that are potentially critical or near-critical paths. In addition to capturing slacks from actual critical/near-critical paths—which strongly influence the $F_{\max}$, the multiple number of embedded slack sensors also act as process monitors. In the experimental work of [9] that analyzed data from large number of test chips, it was reported that the use of process monitor sensors, such as ROs, substantially improved the $F_{\max}$ prediction accuracy. The use of sensors may slightly increase the design effort, but significantly reduces the $F_{\max}$ test cost and time as explained in the later sections. The design effort is not significant as we insert our slack sensors on selected number of critical/near-critical paths inside the gate-level netlist and then use the conventional physical design flow. The selected paths are not necessarily the exact critical paths, making the flow more realistic to modern SoC designs at lower technology nodes. The use of sensors also allow measuring $F_{\max}$ in the field during functional mode and in-field BIST mode in addition to the manufacturing test mode.

In summary, our contributions and novelties in this paper are as follows.

1) A novel $F_{\max}$ prediction framework is presented to eliminate the long test time and cost associated with functional $F_{\max}$ testing. For the development of accurate machine learning predictors, the timing slacks extracted from actual critical/near-critical paths—instead of data from path replica or RO monitors—are used as features. To identify the most accurate predictor for this framework, we tested five different machine learning techniques.
2) For the purpose of *in-situ* timing slack data extraction, a novel timing slack sensor IP has been designed that can monitor and record the timing slack of critical/near-critical capture flip-flops.
3) In order to ensure that the features of the machine learning tools—the path slacks—capture spatial variation and workload profile, a novel layout and gate-overlap aware algorithm has been proposed for the selection of the optimum set of critical/near-critical flip-flops. The algorithm ensures that the selected capture flip-flops (and the corresponding sensors) are physically spread across the

layout. Moreover, to minimize the extra design efforts, a netlist-level automated sensor insertion technique has been developed to insert the sensor IPs in the synthesized gate-level netlist of the SoC.

The rest of this paper is organized as follows. Section II describes our proposed $F_{\max}$ framework. The architecture of slack extraction sensor IP is presented in Section III. Sensor insertion algorithm is provided in Section IV. Different machine learning classification techniques are discussed in Section V. Simulation results and analysis are presented in Section VI, followed by the conclusions in Section VII.

II. Proposed Speed Binning Flow

The chip $F_{\max}$ is limited by the path delay of the most critical path. Because of process variations, the exact critical path cannot be identified at the prefabrication stage [5]. As a result any of the critical or near-critical paths—identified from the static timing analysis (STA) or statistical STA—can be the speed-limiting most critical path for a certain chip depending on the chip-to-chip process variations. Without loss of generality, path slack, and $F_{\max}$ distributions are correlated [8], [9]. As a result, if the impact of process variations on path slack distribution is known, using the slack's correlation with $F_{\max}$, one would be able to predict the distribution of $F_{\max}$ with process variations at the production test stage. In order for this approach to be practical, we would require a simpler method to extract path slack *in-situ* from fabricated chips. Toward this goal, we have designed an all digital sensor IP which is connected to a capture flip-flop, and records the worst-case slack of all the paths that terminate at that capture flip-flop. The use of slack sensors eliminates the frequency sweep step associated with speed binning. Structural test patterns are used to excite those critical/near-critical paths monitored by the slack sensors attached to the respective path-ending capture flip-flops. Using modern ATPG tools (e.g., Synopsys Tetramax) those specific paths or capture flip-flops can be easily excited by PDF or TDF patterns, without the requirement of executing exhaustive functional patterns. Application of these test patterns and corresponding response extraction can be accomplished by the ATE tool through the chip's JTAG ports.

Before proceeding with the proposed speed binning flow, it is necessary to make sure all the chip samples are free from all possible manufacturing defects such as stuck-at fault, path/TDF, etc. [13]. This is to ensure that no defective chip escapes from the initial stage of chip test [13]. Highly unreliable trained predictor may result if gross timing defects are not screened out in advance. Also predicted $F_{\max}$ response from defective samples will be erroneous.

Our proposed $F_{\max}$ prediction methodology is depicted in Fig. 1. The essence of this approach is to train a machine learning classifier with sufficient training data obtained from chips under test and training (CTT) [Fig. 1(a)] and later use this trained predictor to estimate $F_{\max}$ for new chips under test (CUT) [Fig. 1(b)]. The required number of CTTs [marked as black dots on the wafer in Fig. 1(a)] n depends on the characteristics of the data and generally $n << k$, where k is the number of remaining CUTs [marked as red dots on the wafer in Fig. 1(b)]. Let us assume $\{S_{i1} \dots S_{im}\}$ represents the slacks from m sensors of the ith chip. The matrix X_{tr} contains

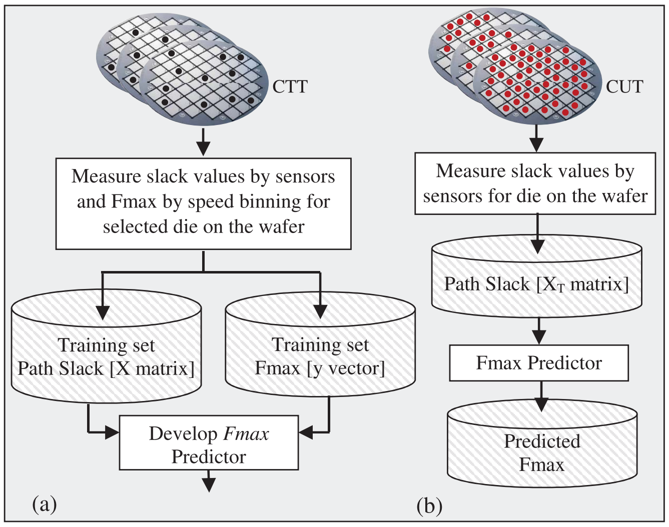
CTT
Measure slack values by sensors and Fmax by speed binning for selected die on the wafer
Training set Path Slack [X matrix]
Training set Fmax [y vector]
Develop Fmax Predictor
(a)
CUT
Measure slack values by sensors for die on the wafer
Path Slack [X_T matrix]
Fmax Predictor
Predicted Fmax
(b)

Fig. 1. (a) $F_{\max}$ predictor development from CTTs. (b) $F_{\max}$ prediction of CUTs using the developed predictor.

slack data of m sensors from each of the n different training samples. The column vector y_{tr} contains the $F_{\max}$ of each of the n training chips. The slack data collected from k prediction samples in matrix X_{pr} are fed to the trained classifier to obtain the predicted $F_{\max}$ for these samples

$$X_{\text{tr}} = \begin{bmatrix} S_{11} & \cdots & S_{1m} \\ \vdots & \ddots & \vdots \\ S_{n1} & \cdots & S_{nm} \end{bmatrix} y_{\text{tr}} = \begin{bmatrix} F_1 \\ \vdots \\ F_n \end{bmatrix}$$
$$X_{\text{pr}} = \begin{bmatrix} S_{11} & \cdots & S_{1m} \\ \vdots & \ddots & \vdots \\ S_{k1} & \cdots & S_{nm} \end{bmatrix}.$$

During training phase both functional $F_{\max}$ test and sensor data extraction is done. The $F_{\max}$ from functional test and the extracted sensor responses are fed to the machine learning software running at the ATE's computer and a trained predictor is obtained. In the training process, if there are N discrete $F_{\max}$ classes in the training data or N bins, an N-level classifier is trained with the training data. Sensor data extraction involves three steps: 1) activation of the sensor monitored paths by shifting in the appropriate test patterns through JTAG port; 2) shifting out the sensor responses through JTAG port; and 3) storage of the responses in a log file for feeding into the ATE's computer.

After the machine learning tools are trained with sufficient number of CTTs, for CUTs, the expensive procedure of functional $F_{\max}$ testing is omitted and only slacks are measured using the embedded sensors. At prediction phase, the three steps of the above mentioned sensor data processing are completed followed by step 4) execution of the trained machine learning program in ATE's computer to get the predicted $F_{\max}$. The $F_{\max}$ prediction time can be obtained by summing the times spent in each of the above four steps involved in the prediction phase. Silicon results presented in [19]—which used randomly placed ROs as sensors and Bayesian regression for machine learning—reported 2780× reduction in $F_{\max}$ test

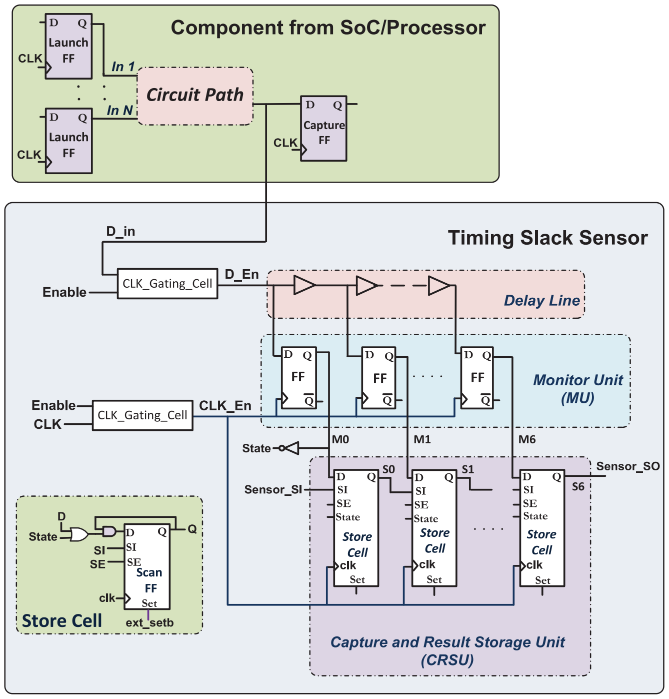

Fig. 2. Slack sensor architecture for rising transition.

time for a four bin search space over the conventional functional $F_{\max}$ testing for ARM-CA9 SoCs fabricated in 28 nm technology.

III. Path Slack Sensor

An important component of the flow presented in Fig. 1 is a low-cost embedded sensor to accurately measure slack of the selected number of critical/near-critical paths. The circuit diagram of the used slack sensor IP is shown in Fig. 2. When activated, the sensor monitors and records the worst-case timing slack at the capture flip-flop for all the paths terminating at that particular capture flip-flop. The sensor probes path ending capture flip-flop's D port through a minimum size clock gating cell ensuring that small amount of load capacitance is added to the main circuit path. In Fig. 2, the architecture of rising transition detector sensor is shown. To make a falling transition detection sensor the $M0$–$M6$ bits are simply required to be connected to the $\bar{Q}$ outputs (instead of Q) of the flip-flops of the monitor unit (MU). The different modules of the sensor and their functions are briefly described below.

A. Clock Gating Unit

The sensor IP is clock gated with the *Enable* input to turn it on only during the timing slack measurement interval as shown in Fig. 2. The gated clock signal CLK_*En* controls all the local flip-flops in the sensor. To preserve the relative timing relationship between the actual clock (CLK) and input data (*D_in*), the clock gating cell was placed with both the incoming data (*D_in*) and clock (CLK) inputs. As a result CLK_*En* and *D_En* were synchronized similarly to their non clock-gated counterparts. The clock gating mechanism also minimizes power dissipation during the sensor OFF stage by cutting the dynamic power dissipation.

B. Delay Line

The delay line consists of a chain of unit size buffers from the standard cell library. The buffers quantize the total slack according to the propagation delay of the used buffer. The number of required buffer stages in the delay line depends on the desired slack detection range and the delay of a unit size buffer. Use of the unit size buffer from the standard cell library ensures that the slack detection resolution is the optimum.

C. Monitor Unit

A flip-flop is attached at the end of each buffer to capture the states in bits $M0$–$M6$ at the active clock edge. The combination of delay line and the flip-flops in the MU convert the timing slack into a corresponding digital data. The first bit, $M0$, of MU is inverted to get the *State* signal. This *State* signal indicates if the desired (rising or falling, depending on sensor configuration) transition is occurring in the incoming data *D_in*.

D. Capture and Result Storage Unit

The capture and result storage unit (CRSU) is constructed of storage cells connected in a scan chain. Each storage cell consists of an OR-AND gate and a scan flip-flop as shown in Fig. 2. Each flip-flop in the scan chain samples the corresponding MU flip-flop's output. The AND gate connected to the scan flip-flop inside the storage cell makes it sticky, in the sense that if the flip-flop ever records a zero, it will hold on to the zero until reset or it is scan-loaded with a logic one. The *State* signal makes the OR gate transparent to the outputs of MU flip-flops only for the desired transitions. At the initialization stage all the scan flip-flops are set to logic one. During the slack evaluation phase, *SE* is set to zero and the flip-flops inside CRSU record the worst-case slack observed at the monitored capture flip-flop. The sensor data extraction process is initiated by setting *SE* pin to logic high which connects the flip-flops in a scan chain clocked by the main clock and finally the stored slack data is extracted through the *Senor_SO* pin. For multiple number of sensors, their CRSU are connected by a scan chain for data extraction. The meta-stability issue may occur in at most one of the flip-flops of the MU block if the output from the buffer of the delay line arrives late in the flip-flop's setup time window. Any such meta-stability is automatically resolved in our architecture as we are using two flip-flops in series—forming a synchronizer circuit [23]—with the output of each buffer in the delay line. Since flip-flops in both MU and CRSU are operated by the same clock, the values that were latched by MU flip-flops are sampled again in the next clock cycle by the CRSU flip-flops. This latency of one clock cycle between successive sampling is utilized to resolve the meta-stability in the output of MU flip-flops, if any. As a result, even if there occurs a meta-stability in the MU block, the flip-flops of the CRSU will practically be meta-stability resolved because of the high mean time between failure of a synchronizer circuit [23] and the designed sticky feature of the CRSU flip-flops.

In addition to its application in speed binning, the sensor IP can be also used for in-field reliability observation. When

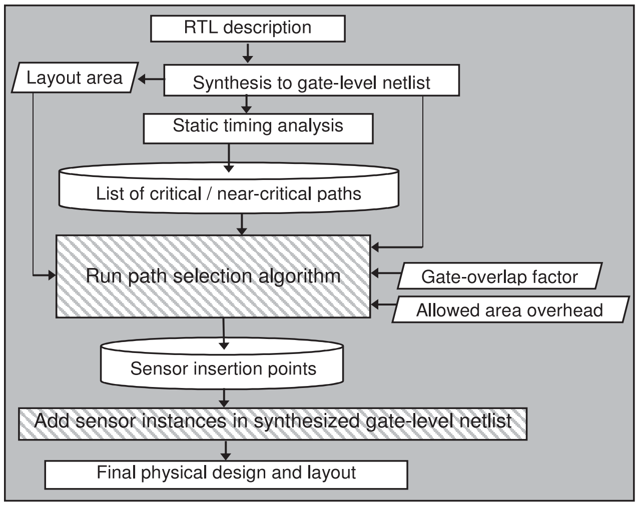
RTL description
Layout area
Synthesis to gate-level netlist
Static timing analysis
List of critical / near-critical paths
Run path selection algorithm
Gate-overlap factor
Allowed area overhead
Sensor insertion points
Add sensor instances in synthesized gate-level netlist
Final physical design and layout

Fig. 3. Sensor insertion flow.

necessary, instances of this sensor embedded into the SoC can be activated to monitor and record the delay degradation on the critical/near-critical paths because of noise, aging, and other wear-out. For this paper, we only analyze the application of the sensor IP in speed binning at $time_0$.

IV. Sensor Insertion Flow

In this section, we develop a gate-netlist level and layout-aware algorithm to select path-ending capture flip-flops to be monitored by sensors. The sensors act as features in our machine learning tool. The candidate capture flip-flops for sensor placement should be selected based on two main criteria: 1) the sensors target the critical/near-critical paths and 2) the sensors or features of the machine learning tool should be a good representative of the trend of the modeling data to incorporate the chip-to-chip PVT variations [25]. Our proposed sensor insertion flow along with the capture flip-flop selection algorithm, shown in Fig. 3, addresses these criteria. For the first condition, it is not practical to place sensors at the end of each of the critical/near-critical paths, as the number of such paths are extensive in modern SoCs. Hence, the number of sensors to place are decided by the area-overhead budget. For the second criterion, the proposed algorithm ensures that the sensors cover a wide range of diverse or distinct paths and those paths are spatially distributed across the layout. Spatial distribution of the sensors allow monitoring PVT variation effects on the timing critical paths. The flow begins with the synthesis of hardware from RTL to the gate-level netlist as depicted in Fig. 3. At this stage, based on the standard cell library used in synthesis and the gate-level netlist, an estimate of the layout area is obtained from the synthesis CAD tool. After that, STA is performed on the synthesized netlist, and critical/near-critical paths (considering both single and multicycle paths) are sorted in order of their respective path slacks. Next the capture flip-flop selection algorithm presented in Algorithm 1 is executed. The algorithm is composed of three main procedures as described in the following.

The first procedure *netlist parser* (lines 1–11) takes as input the gate-level netlist of the design, list of logical modules

Algorithm 1 Proposed Capture Flip-Flop Selection Algorithm

```
1: procedure NETLIST PARSER
2: Input: Gate-level netlist of the main design
3: Input: Number of logical modules in design netlist, $N_{module}$
4: Input: Static timing analysis report with capture flip-flops and slack data
5: Input: Cut-off timing slack, $S_{cut}$
6: Output: For each logical module $k$, a list of unique capture flip-flops sorted in
   ascending order of slack, $D_k = \{d_{1k}, d_{2k}, d_{3k}, \ldots d_{nk}\}$
7:     $C_{FF} \leftarrow$ set of all unique capture flip-flops in design netlist with slack below $S_{cut}$
8:     for $k$ =1 to number of logical modules $N_{module}$ do
9:         $D_k \leftarrow$list of capture flip-flops ($D_k \subset C_{FF}$) in logical module $k$
10:     end for
11: end procedure
12:
13: procedure GATE-OVERLAP AWARE FLIP-FLOP SORTER
14: Input: Logical module based capture flip-flop list $D_k$ from previous procedure
15: Input: Gate-overlap threshold, $\delta_{OV}$
16: Input: Number of logical modules in design netlist, $N_{module}$
17: Output: Gate-overlap aware list of capture flip-flops (sorted in ascending order of
    slack) for each logical module $k$, $L_k$
18:
19:     for $k$ =1 to number of logical modules $N_{module}$ do
20:         for $i$ =1 to number of flip-flops in $D_k$ do
21:             $G_{ik} \leftarrow$list of on-path (critical/near-critical) gates for flip-flop $d_{ik} \in D_k$
22:         end for
23:     end for
24:     Initialization: $T_k = \{G_{1k}\}$ for $k$=1 to $N_{module}$
25:     Initialization: $L_k = \{d_{1k}\}$ for $k$=1 to $N_{module}$
26:     for $k$ =1 to number of logical modules $N_{module}$ do
27:         for $i$ =2 to number of flip-flops in $D_k$ do
28:             $\delta$ = percentage gate-overlap between $T_k$ and $G_{ik}$
29:             if ($\delta < \delta_{OV}$) then
30:                 $T_k = T_k \cup G_{ik}$
31:                 $L_k = L_k \cup d_{ik}$
32:             end if
33:         end for
34:     end for
35: end procedure
36:
37: procedure IDENTIFICATION OF SENSOR INSERTION POINTS
38: Input: Area of the sensor module, $Area_{Sensor}$
39: Input: Estimated total area of the main design, $Area_{main.design}$
40: Input: Area-overhead budget, $Area_{Overhead}$
41: Input: Optimized list of unique capture flip-flops in each logical module $k$, $L_k$
42: Output: Set of target capture flip-flops for sensor insertion in each module $k$, $FF_k$
43:     $Total_{Sensor} = (Area_{main.design} * Area_{Overhead})/Area_{Sensor}$
44:     for $k$ =1 to number of logical modules $N_{module}$ do
45:         $N_k$ = number of capture flip-flops in list $L_k$
46:     end for
47:     $\sum N_k$=total number of capture flip-flops in all of the $k$ modules
48:     for $k$ =1 to number of logical modules $N_{module}$ do
49:         $P_k = \frac{N_k}{\sum N_k}$; module $k$'s contribution to the total number of candidate flip-
            flops
50:         $S_k = P_k * Total_{Sensor}$; Allotted number of sensors for logical module $k$
51:         $FF_k \leftarrow$ select top $S_k$ capture flip-flops from list $L_k$
52:     end for
53: end procedure
```

inside the design, timing slack data for each capture flip-flop obtained from STA, and a cut-off slack margin. This procedure outputs the sorted list (D_k) of capture flip-flops grouped according to the logical modules where those belong to. The logical module and slack-based sorting of capture flip-flops is accomplished by looping over all of the capture flip-flop instances above the target cut-off slack, followed by parsing of the gate-level netlist to identify the flip-flop's location in the logical hierarchy and putting it in the appropriate list as shown in lines 8–10.

The second procedure *gate-overlap aware flip-flop sorter* (lines 13–35) outputs a reduced list L_k of capture flip-flops obtained by narrowing down the selection to include paths with diverse or distinct logic gates. The distinctness of the selected capture flip-flops is achieved by reducing the number of overlapping logic gates among the critical/near-critical paths terminating at those capture flip-flops. The inputs to this procedure are the flip-flop list D_k from the previous procedure, a target gate-overlap threshold δ_{OV} and the list of logical modules. In the first stage of this procedure, the lists G_{ik}—of all on-path logic gates for the critical/near-critical paths terminating at each capture flip-flop i of each logical module k—are generated as shown in lines 19–23. Next, for each hierarchical logical module k, we create two arrays T_k and L_k to hold the on-path logic gates and capture flip-flops, respectively. We initialize L_k with the first capture flip-flop d_{1k} from the list D_k and T_k with the corresponding on-path logical gates as demonstrated in lines 24 and 25. Next, for each logical module, we iterate over the rest of the capture flip-flops and include a capture flip-flop in the list L_k if the gate-overlap threshold between this particular flip-flop's on-path logic gates and the existing gates in the list T_k is less than the preselected threshold δ_{OV} (lines 26–34).

The final procedure *Identification of Sensor Insertion Points* generates the list FF_k of target capture flip-flops in module k for sensor insertion. The inputs to this procedure are the area of the main design, area of each sensor, area-overhead budget and the optimized flip-flop list L_k from the previous procedure. In line 43, the total number of available sensors is calculated from the area-overhead budget. Next, in lines 44–46, N_k number of flip-flops in list L_k are identified for each module k. In lines 49 and 50, available sensor quota S_k for each logical module k is calculated. This quota for a module is decided by the module's share in the cumulative number of total capture flip-flops. Lastly, in line 51, the final capture flip-flop list FF_k for each module k is reported by selecting top S_k flip-flops from the list L_k for sensor insertion.

After finalizing the sensor insertion points or the capture flip-flops, the sensor instances are added inside the synthesized netlist of the SoC at those identified insertion points using custom scripts and the physical design is completed.

V. Machine Learning

In developing our speed binning model, we explored five major multiclass machine learning classifiers [33]: 1) multinomial logistic regression; 2) multiclass support vector machine; 3) bootstrap aggregation (bagging) decision tree; 4) random forest decision tree; and 5) adaptive boosting (AdaBoost). Brief descriptions on each of these techniques are given below.

A. Multinomial Logistic Regression

Multinomial logistic regression is a supervised machine learning technique, where a linear combination of the observed features are used to predict the class label [26]. In this regression model the log-odds of the outcomes are modeled as a linear combination of the predictor variables. Contrary to linear regression, where the dependent variable or the outcome is continuous, in logistic regression it is categorical, i.e., discrete.

The training set of m training samples is represented in the form $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$. The input data sample $x^{(i)} \in \mathbb{R}^N$, where N is the total number of features. For K possible classes, the class labels $y^{(i)} \in \{1, 2, \ldots, K\}$. The weight parameters, corresponding to the class labels, are $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \mathbb{R}^N$. By concatenating the columns $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ an N-by-K matrix $\theta = [\theta^{(1)} \ \ \theta^{(2)} \ \ \ldots \ \ \theta^{(K)}]$ is formed [26].

During supervised training stage, using the training data the matrix θ is obtained for which the following cost function $J(\theta)$ is minimum. Iterative optimization algorithm is used for obtaining the solution [26]

$$J(\theta) = -\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp\left(\theta^{(k)^T} x^{(i)}\right)}{\sum_{j=1}^{K} \exp\left(\theta^{(j)^T} x^{(i)}\right)}.$$

During prediction, for any test input x, a hypothesis is required to estimate the probability that $P(y = k|x)$ for each $k \in \{1, 2, \ldots K\}$. This hypothesis outputs a vector of length K representing the probabilities. The hypothesis $h_\theta(x)$ is [26]

$$\begin{aligned} h_\theta(x) &= \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} \\ &= \frac{1}{\sum_{j=1}^{K} \exp\left(\theta^{(j)^T} x\right)} \begin{bmatrix} \exp\left(\theta^{(1)^T} x\right) \\ \exp\left(\theta^{(2)^T} x\right) \\ \vdots \\ \exp\left(\theta^{(K)^T} x\right) \end{bmatrix}. \end{aligned}$$

In $h_\theta(x)$ the class label for which the probability is maximum is taken as the predicted class for the test data x.

B. Multiclass Support Vector Machine

The support vector machine (SVM) classifier is inherently a binary classifier that distinguishes between two classes [27]. To apply SVM technique to multiclass classification, the multiclass problem is first reduced to multiple binary classification problems that can be solved separately [28], [29]. For a multiclass classification scenario with K number of classes, one-against-one or all-pairs comparison would require $\binom{K}{2}$ or $K(K-1)/2$ binary classifiers. The $\binom{K}{2}$ hypotheses that are generated by this process are next combined to get the final predicted class [28]. To combine multiple binary SVMs for multiclass classification problems, the error correcting output code (ECOC) multiclass model has been proposed to achieve a very high prediction accuracy [28].

The ECOC model reduces the problem of classification with three or more classes to a set of binary SVM classifiers by using a coding matrix M_C, where the elements of M_C are from the set $\{-1, 0, +1\}$ [29]. For the all-pairs approach and K number of classes, the coding matrix M_C has K rows and $\binom{K}{2}$ columns. Each row of the coding matrix corresponds to a distinct class. The SVM of each column of M_C corresponds to a distinct pair (r_p, r_q), where (r_p, r_q) is one of the $\binom{K}{2}$ possible pairs. In the column corresponding to the SVM for the pair (r_p, r_q), $+1$ (to indicate positive class) is assigned for the row r_p, -1 (to indicate negative class) is assigned for row r_q and zeros (ignore) are assigned for all other rows [29].

During training phase, all the $\binom{K}{2}$ binary SVMs—corresponding to the columns of M_C—are trained using conventional SVM methods [27]–[29].

After training is complete, for a new observation X all the trained $N = \binom{K}{2}$ binary SVMs are executed. The vector of predictions by the individual SVMs are, $f(X) =$

Algorithm 2 Bagging Machine Learning Classification

```
1: procedure TRAINING PHASE
2: Input: Training data set $Z = \{X_{Tr}, y_{Tr}\}$; Matrix $X_{Tr}$ is of the form $[x_1; x_2; \ldots; x_m]^T$ where each $x_i$ is a row vector ($i$=1 to $m$) and $y_{Tr}$ is a $m$ element column vector consisting of class labels for each row of matrix $X_{Tr}$
3: Input: Number of classifiers to train, $N$
4: Output: A set of trained classifiers $T = \{D_1, D_2, D_3, \ldots D_N\}$
5:     Initialization: $T = \{\}$
6:     for k =1 to number of classifiers $N$ do
7:         $S_k \leftarrow$generated bootstrap sample with replacement from training data set $Z$
8:         $D_k \leftarrow$built and trained decision-tree classifier using $S_k$ as the training data
9:         $T = T \cup D_k$
10:    end for
11: end procedure
12:
13: procedure PREDICTION PHASE
14: Input: A set of trained classifiers, $T = \{D_1, D_2, D_3, \ldots D_N\}$
15: Input: New data for classification, $x_C$
16: Output: Predicted class or label, $y_C$, corresponding to the input data $x_C$
17:
18:    for k =1 to number of classifiers $N$ do
19:        $r_k \leftarrow$ response obtained from execution of trained classifier $D_k \in T$ on $x_C$
20:    end for
21:    $y_C \leftarrow$ Majority vote $\{r_1, r_2, \ldots r_k\}$
22: end procedure
```

$[f_1(X), \ldots, f_N(X)]$. As the final predicted class label, $\hat{y}$ is chosen such that

$$\hat{y} = \underset{r \in Y}{\operatorname{argmin}} \sum_{s=1}^{N} L(M_C(r,s) f_s(X))$$

where L is the loss function, Y is the set of K class labels [29].

C. Bootstrap Aggregating

Bagging is an ensemble meta-algorithm that improves the stability and accuracy of machine learning classifiers [30]. The essence of bagging is to fit many large decision trees to bootstrap-resampled versions of the training data, and classify the final label by majority vote. Bagging averages a given procedure over many samples, to reduce its variance. As bagging algorithm performs averaging on bootstrap samples, the error from variance can be suppressed even for the regression or classification procedures that are not very stable [30]. In situations for which the predictor has a relatively large variance, bagging can appreciably reduce the mean squared prediction error, provided that the learning sample is sufficiently large. The bagging machine learning flow [30] is presented in Algorithm 2. In the training phase (lines 1–11), the inputs to the algorithm are the training samples along with their respective class labels and the targeted number of classifiers to train. The m by f matrix X_{Tr} consists of m training samples with f number of features. The corresponding class labels are included in the column vector y_{Tr}. The output T is a set of trained classifiers on the bootstrap samples. As depicted in lines 6–10, on each iteration a bootstrap sample S_k is drawn from the training data set, a decision-tree classifier D_k is obtained with this sample and included in the classifier ensemble T. In the subsequent prediction phase (lines 13–22), the trained ensemble T and new data for classification x_C are fed, and the predicted class label y_C is obtained. Each classifier D_k in the ensemble T reports a predicted class label r_k as shown in line 19. Finally in line 21, a majority vote is taken on the individually predicted labels and the winner is the final class label y_C.

Algorithm 3 AdaBoost.M2 Machine Learning Classification

```
1: procedure TRAINING PHASE
2: Input: Set of unique class labels $Y$, where number of class labels is $L$
3: Input: Training data set $Z = \{X_{Tr}, y_{Tr}\}$; Matrix $X_{Tr}$ is of the form $[x_1; x_2; \ldots; x_m]^T$ where each $x_i$ is a row vector ($i$=1 to $m$) and $y_{Tr}$ consists of class labels for each row of matrix $X_{Tr}$. $y_{Tr}$ is of the form $[y_1\ y_2\ \ldots\ y_m]^T$ where $y_i \in Y$ ($i$=1 to $m$)
4: Input: Weak learning algorithm WeakLearn
5: Input: Number of iterations or boosting rounds, $N$
6: Output: The final classifier $h_{fin}$
7:     Let $B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}$, so length of B, $|B| = m(L-1)$
8:     Initialization: $D_1(i, y) = \frac{1}{|B|} = \frac{1}{m(L-1)}$ for $(i, y) \in B$
9:     for k =1 to number of iterations $N$ do
10:        $h_k \leftarrow$hypothesis returned by WeakLearn called with mislabel distribution $D_k$
11:        $\varepsilon_k = \frac{1}{2} * \sum_{(i,y)\in B} [D_k(i, y)] * [1 - h_k(x_i, y_i) + h_k(x_i, y)]$ = pseudo-loss of $h_k$
12:        $\beta_k = \varepsilon_k / (1 - \varepsilon_k)$
13:        $D_{k+1}(i, y) = \frac{D_k(i,y)}{Z_k} * \beta_k^{0.5*[1+h_k(x_i,y_i)-h_k(x_i,y)]}$; $Z_k$ is normalization constant
14:    end for
15:
           $$h_{fin}(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{k=1}^{k=N} log(\frac{1}{\beta_k}) * h_k(x, y)$$
16: end procedure
17:
18: procedure PREDICTION PHASE
19: Input: The final classifier $h_{fin}$
20: Input: New data for classification, $x_C$
21: Output: Predicted class or label, $y_C$, corresponding to the input data $x_C$
22:
           $$y_C = h_{fin}(x_C) = \underset{y \in Y}{\operatorname{argmax}} \sum_{k=1}^{k=N} log(\frac{1}{\beta_k}) * h_k(x_C, y)$$
23: end procedure
```

D. Random Forest

The random forest algorithm strives to improve the efficacy of the bootstrap aggregating technique by decorrelating the trees [31]. The random forest technique is similar to the bagging flow presented in Algorithm 2, with the only exception that at each tree split a random sample of q features are drawn, and only those features are considered for splitting in the decision tree. Generally $q = \sqrt{f}$, where f is the total number of features. The overall effect is that the variance is reduced when we average the trees. In other words, random forest curbs the over-fitting of the training data set [30], [31].

E. Adaptive Boosting

AdaBoost is a machine learning meta-algorithm [32], where the training flow starts with the execution of the first classifier supplied with equally weighted training data. Based on the accuracy of the first classifier, weights on misclassified data are increased for the second classifier. This procedure of giving emphasis on misclassified data is repeated until all the classifiers are trained. The AdaBoost.M2 algorithm [32], presented in Algorithm 3, takes as input the set of unique class labels Y, the training data set (X_{Tr}, y_{Tr}), a choice of a basic classifier called *WeakLearn*, the number of iterations N, and outputs the final optimized classifier h_{fin}. The mislabel weight distribution function D_1 is initialized to a uniform distribution. Next the algorithm calls the basic classifier *WeakLearn*—generally a tree classifier—repeatedly in a series of iterations. On iteration k, the booster provides *WeakLearn* with a mislabel distribution D_k over the training data set. As shown in line 10, the response of *WeakLearn* is a classifier or hypothesis h_k, which is able to correctly classify a portion of the training set that has higher probability with respect to the distribution D_k. The weak learner's goal is to find a hypothesis which minimizes the training error ε_k (line 11) with respect to the distribution D_k that was provided to the weak learner. To indicate the degree of credibility, each weak hypothesis h_k outputs a vector with values in the range of 0–1 corresponding to the class labels, where the objects with values close to 0 or 1 are considered to be implausible or plausible, respectively. In line 11 of Algorithm 3, if $h_k(x_i, y_i) = 1$ and $h_k(x_i, y) = 0$, then h_k has predicted that x_i's class label is y_i, not y, which is the correct prediction. On the other hand, if $h_k(x_i, y_i) = 0$ and $h_k(x_i, y) = 1$, then h_k has incorrectly made the opposite prediction. $h_k(x_i, y_i) = h_k(x_i, y)$ implies random guess [32]. The quality of a trained hypothesis h_k is judged with the pseudo-loss factor ϵ_k. Pseudo-loss is used to instruct the algorithm to concentrate on the labels that are hardest to differentiate. Pseudo-loss ϵ_k is computed with respect to a distribution D_k over the set of all pairs of examples and incorrect labels as shown in line 11. In line 13, mislabel weight distribution for next iteration is updated according to the training error, and this enables the boosting algorithm to force the *WeakLearn* to focus not only on harder-to-classify samples, but more particularly, on the incorrect labels that are hardest to differentiate. Finally, after N iterations, the booster combines the weak hypotheses $h_1, h_2, \ldots, h_k$ into a single final hypothesis h_{fin} as reported in line 15. Later in the prediction stage, new test data x_C is supplied to the trained classifier h_{fin} to get the predicted class label y_C as shown in line 22 of Algorithm 3.

VI. Simulation Results and Analysis

The first step in extracting the path slacks for speed binning using embedded sensors, is to design the sensor IP with appropriate slack detection range. The detection resolution is limited by the delay of a unit buffer. The nominal resolution at typical-typical corner for our selected 28 nm standard cell library is 20 ps [38]. The expected resolution at 14 and 22 nm nodes are 10 and 6 ps, respectively, considering minimum size buffers at nominal conditions [39]. The resolution also sets the maximum round-off error limit of slack measurement. The slack detection range is set to 15% of the nominal clock period of the SoC in which the sensors would be embedded. With a nominal clock period of 833 ps (based on path-delay analysis of possible critical paths) for our benchmark circuit, at the sensor nominal resolution of 20 ps for the selected standard cell library, 7 buffers are required to cover a 15% slack margin—which is about 140 ps.

After deciding the number of required buffers in the delay line, the slack sensor IP was written in gate-level Verilog and synthesized with design compiler using Synopsys 28 nm standard cell library [38]. During creation of the layout of the sensor soft macro, the buffers were placed in the same row adjacent to one another to ensure delay consistency among the buffers. The flip-flops inside a sensor were also placed adjacently in a row. Each sensor occupied a layout area of 164 μm^2 and dissipated 26 μW power while activated. A post layout SPICE netlist was extracted and simulated with HSPICE [38]. Timing diagrams from post-layout SPICE simulations are given in Fig. 4, where the timing slack simulated

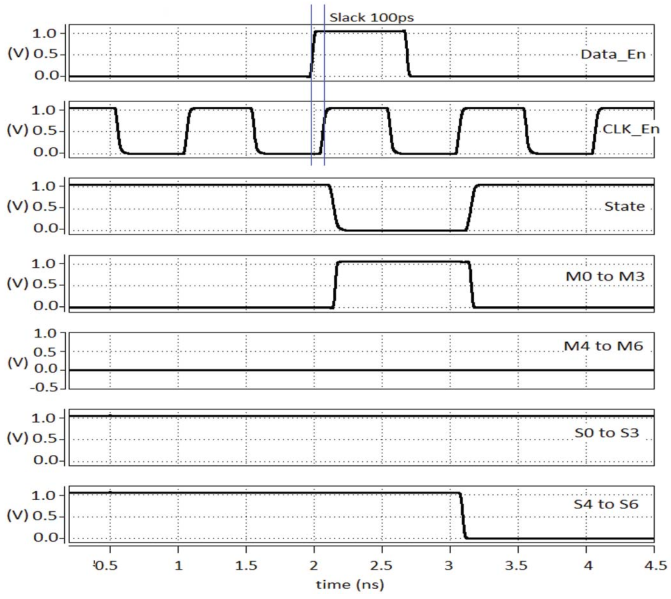

Fig. 4. Wave-shapes of sensor response from SPICE simulation.

is 100 ps. The active zero *State* signal transitions to zero after the rising clock edge at 2 ns. The flip-flops of MU record this slack data into bits $M0$–$M6$ as shown in the respective wave-shapes in Fig. 4. Wave-shapes $S0$–$S6$ capture the worst-case slack information for the case of a rising transition of the *Data_En* signal. In Fig. 4, the timing slack corresponding to the clock edge at 2 ns is recorded in storage bits $S0$–$S6$ after a latency of one clock cycle at the clock edge at 3 ns to avoid meta-stability as discussed in Section III.

The sensor calibration results for 28 nm standard cell library are given in Table I, where column 1 reports the slack obtained by subtracting the flip-flop setup time from the delay between the data arrival time and the active clock edge. Column 2 shows the effects of temperature variations on the sensor responses at nominal VDD. Since only seven buffers were used in the delay line, it was observed—for the selected 28 nm library—that the sensor responses were invariant of any shift in temperature within 20 °C–80 °C, the typical temperature in IC test environment. A similar calibration table can be obtained for other temperatures. We assume that the chip temperature would be available from the typical thermal sensors used in SoCs [36]. As mentioned in Section III, for a rising transition detection, the sensors are initialized with a sensor reading of 1111111, before the path slack measurements commence. At the upper detection limit the sensor code is 1111111 and as the slack decreases, the sensor code changes from 1111111 to 1111110, then to 1111100 and eventually to the lower detection limit at 1000000, where each zero represents slack reduction by an amount equal to the delay of a buffer. Hence, the sensor reading of 1111111 indicates slack is at least 140 ps and 0000000 interprets that the monitored slack was negative. Other states not mentioned in the calibration table indicate that the current slack is beyond the detection range of the sensor. Since the speed of the buffers used in the delay line is sensitive to power supply noise and voltage droop, we analyzed the power supply variation sensitivity of the sensor IP and tabulated the results in Table I. The effect of voltage

TABLE I
SENSOR CALIBRATION RESULTS FOR 28 nm STANDARD CELL LIBRARY

Slack (ps)	Nominal VDD 1.05 V	95% VDD 0.9975 V	90% VDD 0.9450 V
	20°C to 80°C	20°C to 80°C	20°C to 80°C
140	1 1 1 1 1 1 1	1 1 1 1 1 1 0	1 1 1 1 1 0 0
120	1 1 1 1 1 1 0	1 1 1 1 1 1 0	1 1 1 1 0 0 0
100	1 1 1 1 1 0 0	1 1 1 1 1 0 0	1 1 1 0 0 0 0
80	1 1 1 1 0 0 0	1 1 1 1 0 0 0	1 1 1 0 0 0 0
60	1 1 1 0 0 0 0	1 1 1 0 0 0 0	1 1 0 0 0 0 0
40	1 1 0 0 0 0 0	1 1 0 0 0 0 0	1 1 0 0 0 0 0
20	1 0 0 0 0 0 0	1 0 0 0 0 0 0	0 0 0 0 0 0 0

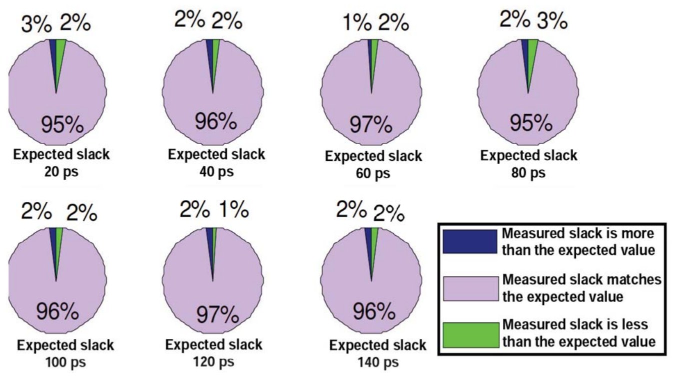

Fig. 5. Monte Carlo process variation simulation results on sensor IP at nominal VDD.

TABLE II
LAYOUT STATISTICS AND SENSOR AREA OVERHEAD FOR FGU

No. of Cells	No. of Flip-flops	Estimated Layout Area without sensor (std. cell)	Each Sensor Area	Allotted Sensors for 2% (std. cell) or 0.39% (overall) Area Over-head
58802	7514	204304 μm^2	164 μm^2	25

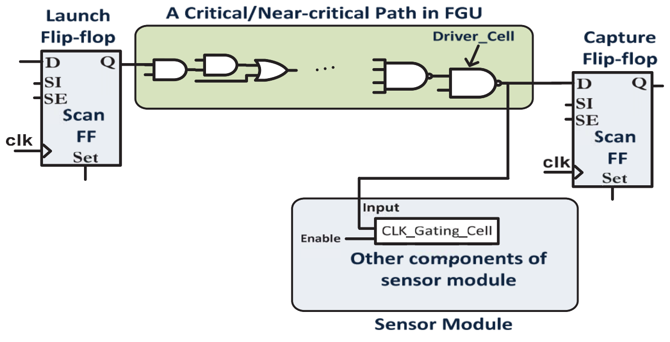

Fig. 6. Sensor loading effect on path delay.

TABLE III
SENSOR LOADING EFFECT ON PATH DELAY

Capacitance on D pin of flip-flop (for cell strength X1)	0.461 fF
Input pin capacitance of clock gating cell (for cell strength X2)	0.668 fF
Driver_Cell's load drive capability (for cell strength X1)	4 fF
Delay of a critical path before sensor insertion (SPICE simulation)	762 ps
Delay of a critical path after sensor insertion (SPICE simulation)	769 ps

droop—slowing down of the sensor as shown for the 90% VDD case in column 3—can be easily decoupled from slack data with the aid of power supply noise sensors [37].

To assess the impact of process variations on the sensitivity of the sensor, we conducted a 150 sample Monte Carlo simulation on the sensor IP. The selected parameters to vary from nominal were transistor length (L), width (W), and threshold voltage (V_{th}) each by 15%, and oxide thickness (t_{ox}) by 4%. All simulations were done within the statistical range of 3 standard deviations (3σ). The results are depicted in the pie charts of Fig. 5, where it can be observed that in all the cases about 95% of the responses matched the expected calibration results of Table I, in the rest of the cases the sensor response was either slower or faster by a unit buffer delay. These results imply that worst-case observation error is limited to one buffer delay.

Next, we implemented our proposed sensor insertion flow and the capture flip-flop selection algorithm described in Section IV for speed binning. We selected the floating point and graphics unit (FGU) circuit from the OpenSPARCT2 SoC's SPARC core [40]. The circuits were synthesized to gate-level netlist with 28 nm standard cell library using design compiler [38]. For the FGU circuit, from an initial post synthesis timing analysis with PrimeTime [38], the list of all the timing paths sorted in the descending order of nominal delay were obtained. The layout area of FGU was estimated from the synthesis tool design compiler [38]. The statistics from the sensor insertion flow are shown in Table II. For an area-overhead budget of 2%, the corresponding number of capture flip-flops to be monitored by sensors were around 25 as reported in column 5 in Table II. Here, the area-overhead of 2% is a conservative estimate as we have considered only the standard cell area. If we also include the area of the register file macro in calculating the overhead, the overall area-overhead is only 0.39%.

Fig. 6 shows the diagram of an actual critical/near-critical path from FGU, which was used to analyze the sensor's loading impact on the critical/near-critical path's delay. In Table III, the pin capacitance values and cell's load drive capacity factor are reported from the NLDM/CCS liberty [38] library file and design manual, respectively, available with the 28 nm standard cell library. We also performed a detailed SPICE simulation on the path with and without the sensor. As shown in rows 4 and 5 of Table III, because of the sensor insertion the path delay increased to 769 ps from 762 ps, which is less than 1% increase.

Column 1 in Table IV reports the different logical modules of the FGU circuit and column 2 reports the number of unique capture flip-flops in each logical module. The slack cutoff range for identifying critical/near-critical capture flip-flops was set as 15% of the nominal clock period. While selecting the critical/near-critical paths to monitor with sensors, we analyzed if the path's output node was rare. A rare node is defined as a node where the switching activity is very low. To obtain the switching activity, it is required to run dedicated test programs that represent the actual workload, and then monitor the switching profile at the node of interest. Since, for FGU circuit module we did not have dedicated workload programs, we have used the Sandia controllability/observability analysis program (SCOAP)-based technique [34] to identify the switching profile at a node. This technique is based on the observation that the probability of a node being rare is directly correlated to

TABLE IV
SENSOR DISTRIBUTION IN FGU OF OPENSPARCT2

Logical Module Name	No. of Unique Capture Flip-flops	Flip-flops with worst slack below cut-off	Flip-flops with worst slack below cut-off After gate-overlap Analysis		No. of Alotted Slack Sensors
			Number	Percent of Total	
FAC	495	6	5	1.6	1
FAD	693	6	5	1.9	1
FDC	76	6	4	1.6	1
FDD	1040	213	97	38.5	8
FEC	84	5	4	1.6	1
FGD	1102	41	23	9.1	2
FIC	65	5	4	1.6	1
FPC	699	13	7	2.8	1
FPE	97	6	4	1.6	1
FPF	731	52	16	6.4	1
FPY	1931	157	83	32.9	7

the node's controllability [34]. A node with very low controllability implies the logic transition on that node is a scarce event. The SCOAP [35] technique can calculate "controllability to 0" and "controllability to 1" for each node in the circuit, where higher values indicate lower controllability. For our switching analysis, we take the ratio, $R_T = \textit{minimum}$ (controllability to 0, controllability to 1)/*maximum* (controllability to 0, controllability to 1). $0 < R_T \leq 1$; and the closer the value of R_T to 0, the lower is the probability of a switching event on that node. We used the SCOAP function available with Synopsys Tetramax tool [38] to evaluate the R_T parameter at the output node (which connects to the D pin of capture flip-flop) of each critical/near-critical path. For the critical/near-critical paths of FGU circuit, the R_T value ranged from 0.44 to 0.95. Since these values are not low enough to indicate significantly low switching activity node, we did not eliminate any critical/near-critical path because of lower probability of switching profile. The results are also intuitive because many gates are connected in-series in a long path, as a result output node of a critical/near-critical path is hardly rare (i.e., low switching event).

To identify the critical/near-critical paths of FGU, and to sort them according to the logical modules where they terminate, the STA was performed in three steps: 1) the STA was run on the full FGU module, and the list of critical/near-critical capture flip-flops obtained; 2) those capture flip-flops (along with their corresponding paths) were grouped according to their logical module location; and 3) the STA was again run individually for each logical module with only the path group of that module. The results of STA are shown in columns 3 and 4 of Table IV. The gate-overlap aware flip-flop sorting algorithm (Algorithm 1 in Section IV) was executed with a gate-overlap factor of 30%. The flip-flop reduction results form gate-overlap analysis are given in column 4 in Table IV. Each logical module's percentage share in the candidate flip-flop list are given in column 5. According to each module's respective share, the slack sensors were allotted into those logical modules. An effort was made to include at least one sensor in each logical module. After finalizing the candidate capture flip-flops for sensor insertion following the proposed flow, the sensor instance was added to those points inside the

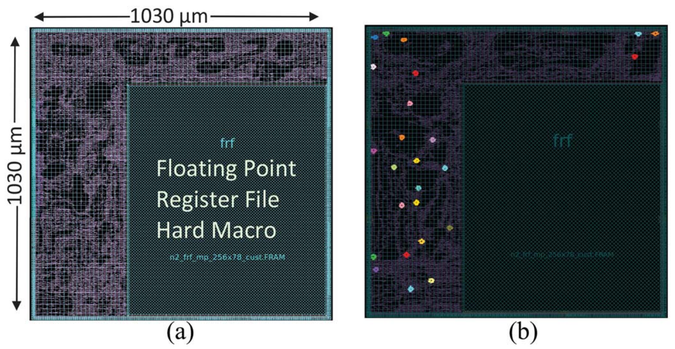
1030 µm
1030 µm
frf
Floating Point Register File Hard Macro
n2_frf_mp_256x78_cust.FRAM
frf
(a)
(b)

Fig. 7. (a) Layout of FGU with embedded sensor network. (b) Cells of sensor modules are highlighted.

TABLE V
PROCESS VARIATION PROFILE FOR MONTE CARLO SIMULATIONS

Process Variation Profile	Transistor Parameter Varied by Percent (%)			
	Channel Length (L)	Channel Width (W)	Threshold Voltage (V_{th})	Gate-oxide Thickness (t_{ox})
Case 1	10	10	10	3
Case 2	15	15	15	4

synthesized gate-level netlist of the FGU. The different steps of our proposed sensor insertion methodology were completed by using Synopsys CAD tools [38], and custom scripts written in Perl, Shell, and Tcl. Finally, the full physical design of the FGU, with embedded sensor network, was completed with IC compiler [38]. Fig. 7(a) shows layout of FGU with 25 embedded sensors. From Fig. 7(b), it can be observed that the sensor blocks are distributed across the layout as those were placed inside the different logical modules of the main netlist according to our proposed flow.

The detailed transistor level circuit netlist of the 25 sensors along with their corresponding monitored paths were extracted from the layout using Synopsys tools [38]. Also, the transistor level netlist of the top 10% critical/near-critical paths—identified from post layout timing analysis—were extracted for $F_{\max}$ identification. Any of these top 10% paths can be the most critical path, deciding the $F_{\max}$. While identifying the possible critical paths, both single cycle and multicycle paths were considered. For multicycle paths, the slacks were normalized according to the cycle count. Because of post-fabrication process variations, it is expected that the delays of these identified critical paths will vary from chip-to-chip. To simulate the effect of post-fabrication process variations, we performed a 300 point Monte Carlo simulation on the extracted critical paths as well as the sensor-path pairs using HSPICE [38]. For Monte Carlo simulation two process variation scenarios were considered as reported in Table V. For case 1, the selected parameters to vary from nominal were transistor length (L), width (W), and threshold voltage (V_{th}) each by 10%, and oxide thickness (t_{ox}) by 3%. For case 2, these parameters were varied by 15% for L, W, V_{th}, and 4% for t_{ox}. All simulations were done within the statistical range of 3 standard deviations (3σ). The simulation time was approximately 25 h on a system with an Intel Xeon processor with 8 cores and 96 GB RAM. The path slacks were extracted from the simulations of the sensor-path pairs by exciting them with appropriate stimulus functions

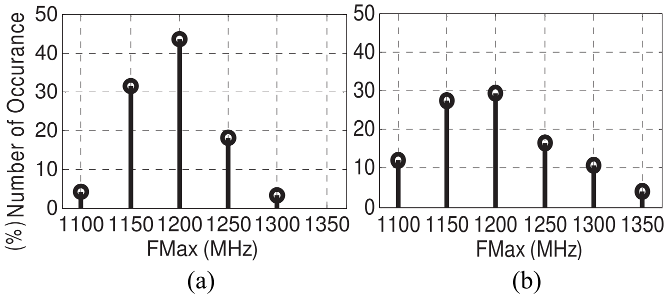

Fig. 8. $F_{\max}$ distribution in Monte Carlo samples in (a) case 1 and (b) case 2.

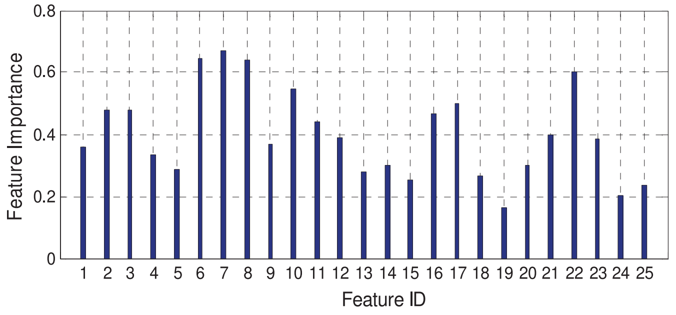

Fig. 9. Importance of each feature (sensor) in developing the random forest predictor.

generated by Synopsys tools [38]. From detailed SPICE simulations of all of the top 10% paths of the circuit, the longest path delay was identified. After adding 10% margin with this delay as a guard-band against aging and noise, the minimum clock cycle and the $F_{\max}$ was estimated for each sample. In order to quantize the continuous $F_{\max}$ values we chose 50 MHz as the grid size or bin width. The distribution of $F_{\max}$ in Monte Carlo samples for the two variation scenarios are shown in Fig. 8. For comparatively lower process variation scenario of case 1 [Fig. 8(a)], the $F_{\max}$ distribution spreads from 1100 to 1300 MHz with a std. deviation of 46 MHz. On the other hand, for comparatively higher process variation scenario of case 2 [Fig. 8(b)], $F_{\max}$ distribution ranges from 1100 to 1350 MHz with std. deviation of 62 MHz. Comparison of these two figures reveal the degree of $F_{\max}$ spread with the magnitude of process variations.

The 300 data samples obtained from Monte Carlo simulations were partitioned into a training set and a validation set. The validation data set contained 100 samples and three cases of training set—consisting of 100, 150, and 200 samples, respectively—were considered. Each of the 25 slacks obtained by the sensors act as a feature in our machine learning-based speed binning flow. The machine learning capabilities of MATLAB [41] were used to implement and train the five classifiers—ECOC SVM, logistic regression, bagging, random forest, and AdaBoost.M2—described in Section V. The relative importance of the 25 features in training the random forest predictor is shown in Fig. 9. After the classifiers were all trained with the training data set, the 100 sample validation data set were applied to the trained classifiers and the corresponding predicted $F_{\max}$ were obtained. These predicted $F_{\max}$ values were later compared with actual $F_{\max}$ to identify the misbinning rate. In Fig. 10, prediction mismatch for each of the 100 validation samples are shown for the different machine learning algorithms and variation profiles for 100 sample training data set. Fig. 10(a)–(e) is for the variation profile of case 1 and Fig. 10(f)–(j) is for variation profile of case 2. It can be observed that for the comparatively lower process variation of case 1, 96%, 98%, 90%, 98%, and 84% prediction accuracy are obtained for bagging, random forest, AdaBoost.M2, ECOC SVM, and logistic regression, respectively. On the other hand, for case 2, the prediction accuracy deteriorated to 91%, 93%, 89%, 94%, and 78%, respectively. In all cases except logistic regression the worst-case misprediction is 50 MHz or one bin. The degraded prediction accuracy for case 2 can be explained with the $F_{\max}$ distribution in the training data set as shown in Fig. 8, where case 2 has 6 bins compared to 5 for case 1.

The effect of number of training samples on prediction accuracy is shown in Fig. 11. For the variation profile of case 1, as we increased the training sample size, the prediction accuracy improved significantly reaching 99% for random forest, ECOC SVM, and Bagging for training set sizes of 150 and beyond. For case 2 [Fig. 11(b)], 200 training samples were required to achieve 99% accuracy for random forest, ECOC SVM, and bagging. From a comparative analysis of the results from five different machine learning techniques in Figs. 10 and 11, it is observed that random forest and ECOC multiclass SVM offer the best prediction accuracy. On the other hand, linear logistic regression performed poorly; indicating a weak direct linear correlation between sensor responses and functional $F_{\max}$. The lower accuracy of AdaBoost.M2 can be explained by the fact that this algorithm emphasizes on the hard to predict samples, and as a result might suffer from over-emphasizing if a small number of samples are included in a particular class in the training data set. This is indeed the case as is evident in the $F_{\max}$ distribution plots in Fig. 8, where only 5% and 4% samples are in the 1100 and 1300 MHz bins, respectively, for case 1 [Fig. 8(a)]. For case 2 [Fig. 8(b)], only 5% samples are in the 1350 MHz bin. The ranking of the observed accuracy of these 5 machine learning techniques in developing a $F_{\max}$ predictor is consistent with the findings of [33].

In the scenario that the chips are sold with the $F_{\max}$ labels predicted by the trained classifier, there is a chance that even for a classifier that was trained with sufficient number of training samples, 1% of the predicted chips might exhibit mispredicted $F_{\max}$. The worst-case misprediction is one bin faster or slower than the actual $F_{\max}$. For a faster chip binned in the slower bin, the customer will not complain. However, a slower chip shipped as a faster one may cause customer return. Depending on the tradeoff between benefits gained—from test time reduction and tester cost savings from $F_{\max}$ prediction—and the cost that will be incurred from dealing with the customer returns of the worst-case 1% shipped samples, the chip vendor can decide if they will sell the chips with the predicted $F_{\max}$ labels or perform a second stage of expedited functional $F_{\max}$ testing at the predicted $F_{\max}$ frequency to eliminate customer returns. If the chip passes this functional test, the correct $F_{\max}$ was predicted. If it fails, then the functional $F_{\max}$ test is done again at the frequency of the immediate lower bin, where a match is expected. In this way

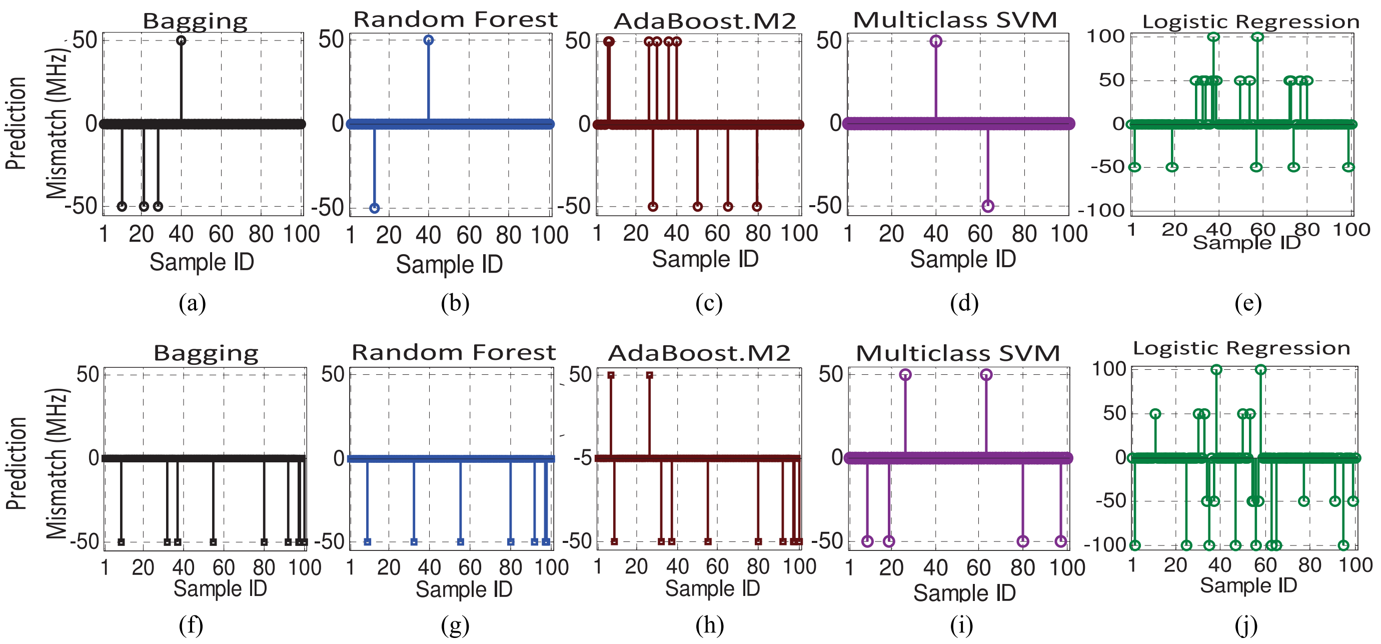

Fig. 10. Prediction results for each of the 100 validation samples for 100 sample training data. Case 1—(a) bagging, (b) random forest, (c) AdaBoost.M2, (d) ECOC SVM, and (e) logistic regression. Case 2—(f) bagging, (g) random forest, (h) AdaBoost.M2, (i) ECOC SVM, and (j) logistic regression.

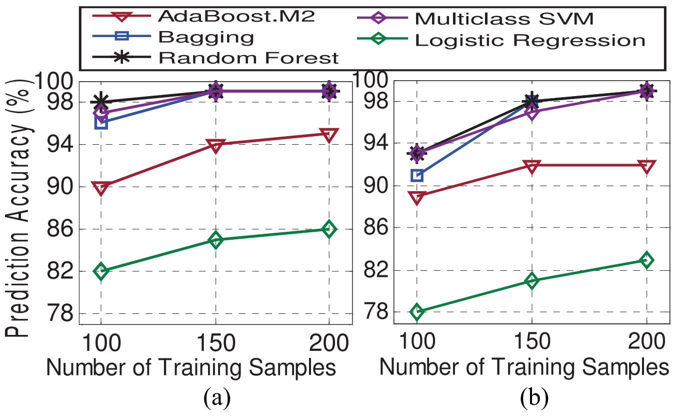

Fig. 11. Improvement of prediction accuracy with increasing number of training samples for variation profile of (a) case 1 and (b) case 2.

there are at most two functional tests that will give the correct $F_{\max}$ for all chips shipped to the customer, as opposed to the test with frequency sweep performed over the possible $F_{\max}$ labels in the conventional flow.

VII. Conclusion

We have proposed an innovative speed-binning flow that utilizes machine learning and the path slacks extracted with on-chip sensors. A novel layout-aware and gate-netlist level sensor insertion algorithm places the sensors uniformly in the layout across the critical/near-critical capture flip-flops. The sensor-extracted path slacks are used as features in modeling and training machine learning software running in the ATE. For a sufficient number of training samples, the worst-case mismatch between predicted and actual $F_{\max}$ is one bin and occurs for 1% of the predicted samples. The proposed framework has the potential to eliminate the high cost associated with conventional functional test-based speed-binning flows.

References

[1] S. S. Sapatnekar, "Overcoming variations in nanometer-scale technologies," *IEEE J. Emerg. Sel. Topic Circuits Syst.*, vol. 1, no. 1, pp. 5–18, Mar. 2011.

[2] P. Das and S. K. Gupta, "Extending pre-silicon delay models for post-silicon tasks: Validation, diagnosis, delay testing, and speed binning," in *Proc. 31st IEEE VLSI Test Symp. (VTS)*, Berkeley, CA, USA, 2013, pp. 1–6.

[3] J. Zeng, R. Guo, W.-T. Cheng, M. A. Mateja, and J. Wang, "Scan-based speed-path debug for a microprocessor," *IEEE Des. Test Comput.*, vol. 29, no. 4, pp. 92–99, Aug. 2012.

[4] E. J. Jang, A. Gattiker, S. Nassif, and J. A. Abraham, "Efficient and product-representative timing model validation," in *Proc. 29th IEEE VLSI Test Symp. (VTS)*, 2011, pp. 90–95.

[5] J. Zeng, J. Wang, C.-Y. Chen, M. Mateja, and L.-C. Wang, "On evaluating speed path detection of structural tests," in *Proc. 11th Int. Symp. Qual. Electron. Design (ISQED)*, San Jose, CA, USA, 2010, pp. 570–576.

[6] C. K. H. Suresh, E. Yilmaz, S. Ozev, and O. Sinanoglu, "Adaptive reduction of the frequency search space for multi-vdd digital circuits," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2013, pp. 292–295.

[7] E. Dimaandal and M. Padilla, "Test-time reduction methodology: Innovative ways to reduce test time for server products," in *Proc. 15th IEEE Electron. Packag. Technol. Conf.*, Singapore, 2013, pp. 718–722.

[8] B. D. Cory, R. Kapur, and B. Underwood, "Speed binning with path delay test in 150-nm technology," *IEEE Des. Test Comput.*, vol. 20, no. 5, pp. 41–45, Sep./Oct. 2003.

[9] K. A. Brand, S. Mitra, E. Volkerink, and E. J. McCluskey, "Speed clustering of integrated circuits," in *Proc. Int. Test Conf. (ITC)*, 2004, pp. 1128–1137.

[10] J. Chen, J. Zeng, L.-C. Wang, J. Rearick, and M. Mateja, "Selecting the most relevant structural Fmax for system Fmax correlation," in *Proc. 28th IEEE VLSI Test Symp. (VTS)*, Santa Cruz, CA, USA, 2010, pp. 99–104.

[11] J. Chen *et al.*, "Data learning techniques and methodology for Fmax prediction," in *Proc. Int. Test Conf. (ITC)*, Austin, TX, USA, 2009, pp. 1–10.

[12] J. Rearick, "Too much delay fault coverage is a bad thing," in *Proc. Int. Test Conf. (ITC)*, Baltimore, MD, USA, 2001, pp. 624–633.

[13] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. New York, NY, USA: Springer, 2006.

[14] H.-J. Hsu, C.-C. Tu, and S.-Y. Huang, "Built-in speed grading with a process-tolerant ADPLL," in *Proc. 16th Asian Test Symp. (ATS)*, Beijing, China, 2007, pp. 384–392.

[15] C.-I. Chung, J.-S. Jhou, C.-H. Cheng, and S.-Y. Li, "Functional built-in delay binning and calibration mechanism for on-chip at-speed self test," in *Proc. 18th Asian Test Symp. (ATS)*, Taichung, Taiwan, 2009, pp. 163–168.

[16] A. Raychowdhury, S. Ghosh, and K. Roy, "A novel on-chip delay measurement hardware for efficient speed-binning," in *Proc. On-Line Testing Symp. (IOLTS)*, 2005, pp. 287–292.

[17] X. Wang, M. Tehranipoor, S. George, D. Tran, and L. Winemberg, "Design and analysis of a delay sensor applicable to process/environmental variations and aging measurements," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1405–1418, Aug. 2012.

[18] Q. Shi, M. Tehranipoor, X. Wang, and L. Winemberg, "On-chip sensor selection for effective speed-binning," in *Proc. 57th IEEE Int. Midwest Symp. Circuits Syst.*, College Station, TX, USA, Aug. 2014, pp. 1073–1076.

[19] S.-P. Mu, M. C.-T. Chao, S.-H. Chen, and Y.-M. Wang, "Statistical framework and built-in self-speed-binning system for speed binning using on-chip ring oscillators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1675–1687, May 2016.

[20] M. Sadi, M. Tehranipoor, X. Wang, and L. Winemberg, "Speed binning using machine learning and on-chip slack sensors," in *Proc. 25th Ed. ACM Great Lakes Symp. VLSI (GLSVLSI)*, Pittsburgh, PA, USA, May 2015, pp. 155–160.

[21] T.-B. Chan, P. Gupta, A. B. Kahng, and L. Lai, "Synthesis and analysis of design-dependent ring oscillator (DDRO) performance monitors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2117–2130, Oct. 2014.

[22] L. Lai, V. Chandra, R. C. Aitken, and P. Gupta, "SlackProbe: A flexible and efficient in situ timing slack monitoring methodology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 8, pp. 1168–1179, Aug. 2014.

[23] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Des. Test Comput.*, vol. 28, no. 5, pp. 23–35, Sep./Oct. 2011.

[24] M. Sadi *et al.*, "BIST-RM: BIST-assisted reliability management of SoCs using on-chip clock sweeping and machine learning," in *Proc. IEEE Int. Test Conf. (ITC)*, 2016, pp. 1–10.

[25] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[26] (May 2016). [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

[27] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," Ph.D. dissertation, Dept. Comput. Sci., Nat. Taiwan Univ., Taipei, Taiwan, 2003.

[28] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artif. Intell. Res.*, vol. 2, no. 1, pp. 263–286, 1995.

[29] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," *J. Mach. Learn. Res.*, vol. 1, pp. 113–141, Dec. 2001.

[30] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[32] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, Bari, Italy, 1996, pp. 148–156.

[33] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, 2014.

[34] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. New York, NY, USA: Springer, 2011.

[35] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia controllability/observability analysis program," in *Proc. 17th Conf. Design Autom.*, 1980, pp. 190–196.

[36] S. Paek *et al.*, "Hybrid temperature sensor network for area-efficient on-chip thermal map sensing," *IEEE J. Solid-State Circuits*, vol. 50, no. 2, pp. 610–618, Feb. 2015.

[37] M. Sadi and M. Tehranipoor, "Design of a network of digital sensor macros for extracting power supply noise profile in SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1702–1714, May 2016.

[38] (Nov. 2015). [Online]. Available: http://www.synopsys.com/

[39] (Nov. 2015). [Online]. Available: http://ptm.asu.edu/

[40] (Nov. 2014). [Online]. Available: http://www.oracle.com/

[41] (Dec. 2015). http://www.mathworks.com/products/matlab/

Mehdi Sadi (S'12) received the B.S. degree in electrical engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2009, and the M.S. degree in electrical engineering from the University of California at Riverside, Riverside, CA, USA, in 2011. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL, USA.

His Ph.D. studies are funded by Semiconductor Research Corporation and NSF. He was a Research Intern with GLOBALFOUNDRIES, Malta, NY, USA and NXP, Austin, TX, USA. His current research interests include digital very large scale integration design, design for test, computer architecture, and circuit design.

Sukeshwar Kannan received the B.E. degree in electrical and electronics engineering from Visvesvaraya Technological University, Belgaum, India, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from the University of Alabama, Tuscaloosa, AL, USA, in 2010 and 2013, respectively.

He is a Principal Engineer with the Technology and Integration for the Advanced Silicon Packaging Development Group, GLOBALFOUNDRIES U.S. Inc., Malta, NY, USA. His current research interests include 3-D IC characterization, reliability, design-for-test, manufacturing development, analog mixed-signal, high-voltage, and RF semiconductor test.

LeRoy Winemberg received the B.S.E.E. degree from the University of Notre Dame, Notre Dame, IN, USA, and the M.S.E.E. degree with the University of California at Berkeley, Berkeley, CA, USA.

He is the DFT Manager with NXP's Automotive Microcontroller and Processor, BU's Austin Research and Development Group focusing on zero defect test and methodology. He was the DFT Manager for both Freescale's Automotive Microcontroller Products Group and TI's ASIC Division. He has also held various positions with Agilent, Santa Clara, CA, USA, HP, Palo Alto, CA, USA, Actel, Aliso Viejo, CA, USA, and LSI Logic, San Jose, CA, USA.

Mark Tehranipoor (S'02–M'04–SM'07) received the Ph.D. degree from the University of Texas at Dallas, Richardson, TX, USA, in 2004.

He is currently the Intel Charles E. Young Professor with Cybersecurity, University of Florida, Gainesville, FL, USA. He has published over 300 journal articles and refereed conference papers and has given over 150 invited talks and keynote addresses. He has published six books and eleven book chapters. His current research interests include hardware security and trust, supply chain security, and very large scale integration (VLSI) design, test, and reliability.

Dr. Tehranipoor was a recipient of several best paper awards as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 MURI Award. He serves on the program committee of over a dozen of leading conferences and workshops. He served as the Program Chair of the 2007 IEEE Defect-Based Testing Workshop, the Program Chair of the 2008 IEEE Defect and Data Driven Testing (D3T) Workshop, the Co-Program Chair of the 2008 International Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS), the General Chair for D3T-2009 and DFTS-2009, and the Vice-General Chair for NATW-2011. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) and served as the HOST-2008 and HOST-2009 General Chair. He is currently serving as an Associate Editor for JETTA, JOLPE, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and ACM TODAES. He served as the Founding Director for CHASE and CSI Centers, University of Connecticut. He is a Golden Core Member of IEEE, and a member of ACM and ACM SIGDA.