A Deep Learning Methodology to Proliferate Golden Signoff Timing

Seung-Soo Han[+], Andrew B. Kahng[†‡], Siddhartha Nath[†] and Ashok S. Vydyanathan[‡]
[†]CSE and [‡]ECE Departments, University of California at San Diego, USA
[+]Department of Information and Communication Engineering, Myongji University, Korea
shan@mju.ac.kr, {abk, sinath, avydyana}@ucsd.edu

***Abstract*—Signoff timing analysis remains a critical element in the IC design flow. Multiple signoff corners, libraries, design methodologies, and implementation flows make timing closure very complex at advanced technology nodes. Design teams often wish to ensure that one tool's timing reports are neither optimistic nor pessimistic with respect to another tool's reports. The resulting "correlation" problem is highly complex because tools contain millions of lines of black-box and legacy code, licenses prevent any reverse-engineering of algorithms, and the nature of the problem is seemingly "unbounded" across possible designs, timing paths, and electrical parameters.**

In this work, we apply a "big-data" approach to the timer correlation problem. We develop a machine learning-based tool, G*olden* T*imer* e*X*t*ension* (GTX), to correct divergence in flip-flop setup time, cell arc delay, wire delay, stage delay, and path slack at timing endpoints between timers. We propose a methodology to apply GTX to two arbitrary timers, and we evaluate scalability of GTX across multiple designs and foundry technologies / libraries, both with and without signal integrity analysis. Our experimental results show reduction in divergence between timing tools from 139.3ps to 21.1ps (i.e., 6.6×) in endpoint slack, and from 117ps to 23.8ps (4.9× reduction) in stage delay. We further demonstrate the incremental application of our methods so that models can be adapted to any outlier discrepancies when new designs are taped out in the same technology / library. Last, we demonstrate that GTX can also correlate timing reports between signoff and design implementation tools.

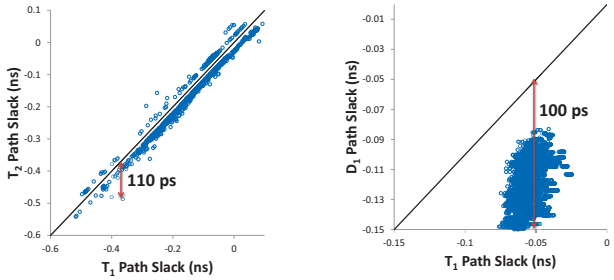
I. Introduction

Accurate timing closure is a critical step in signoff flows of all semiconductor companies [10] and can consume up to 60% of design time [6]. Multiple static timing analysis (STA) tools exist today and different companies adopt different tools as "golden" or the best-in-class STA tool depending on their requirements and product quality standards. According to the analyst firm Gary Smith EDA [17], EDA vendors such as Synopsys [36], Cadence [22], Atrenta [21], CLK Design Automation [25], Incentia Design Systems [27] and Mentor Graphics [31] provide STA and signal integrity analysis tools for use in IC design. These tools typically have high license fees and long runtimes, and they invariably diverge in their timing reports – even though each is well-calibrated to the latest commercial circuit simulators and "qualified" for signoff at leading foundries. Owing to cost and budget constraints, design teams may have limited or no access to a particular "golden" timing tool, but may be interested in comparing the divergence in timing reports between the timing tool they use and that golden tool. The ability to correlate with another (golden) timing tool helps design teams understand if they have overdesign or underdesign, i.e., when their timing tool's reports are respectively pessimistic or optimistic compared to the golden tool's reports. Another use model may be to estimate, based on the timing reports of design implementation tools, how far the implementation is from signoff after each optimization loop (timing-driven placement, congestion-aware routing, leakage reduction, etc.).

We use "gt1-gt2" (that is, "golden tool 1 to golden tool 2") to refer to the problem of correlating two signoff timing tools. We estimate the timing reports of one tool based on the reports of another tool. The correlation problem is extremely complex because:

- tools can suffer from the complexity of millions of lines of black-box code;
- tools can diverge from published user documentation [8], and maintain implementation "errors" for legacy reasons;
- discrepancies between tools change with releases [18] (typically 2× per year for mature tools from major EDA providers);
- tool licenses explicitly prohibit benchmarking and reverse-engineering of internal algorithms; and
- the correlation problem is seemingly "unbounded", as the space of possible timing paths, slew times, multiple-input switching events, coupling effects on delay, etc. is essentially infinite.

978-3-9815370-2-4/DATE14/©2014 EDAA



(a) T_1–T_2. (b) T_1–D_1.

Fig. 1. Path slack discrepancies.

The cost of leaving the gt1-gt2 problem unsolved grows as embedded processor cores reach 3GHz frequencies in 20nm and 16/14nm designs: miscorrelations of >100ps in timing slack correspond to discrepancies of multiple (3 – 4) logic stages at these advanced technology nodes and can strongly impact power and/or area tradeoffs [2] [3] [6]. Figures 1(a) and 1(b) respectively show examples of 110ps and 100ps timing miscorrelations between two leading commercial signoff timing tools T_1 and T_2, as well as between T_1 and a commercial design implementation tool D_1. According to industry experts, reasons for miscorrelation include the use of multiple engines within tools for optimal accuracy and runtime as well as the effects of net length and long waveform tail [16] [19] [20]. Our premise is that the gt1-gt2 problem, while extremely complex, is still treatable as a finite problem that is amenable to big-data mindsets as has been recently seen in highly challenging applications such as natural language processing [26] [35]. Specifically, we identify appropriate modeling parameters and develop a tool, GTX (Golden Timer eXtension), using well-known machine learning techniques[1] to correct[2] setup time, cell delay, wire delay, stage delay, and path slack divergence between tools. Our methodology is properly considered to be deep learning-based because the models in GTX are hierarchical, e.g., the output of the cell and wire delay models are input to the stage delay model [12]. Our modeling goals for each model are to (1) minimize the sum of squared errors, and (2) minimize the maximum range of errors. We achieve:

- Correlation of path slack at timing endpoints[3] between two tools within a range of <30ps for designs implemented in 28FDSOI and 45GS foundry libraries[4] using NLDM delay tables;
- Strong correlation results independent of whether signal integrity (SI) and on-chip variation (OCV) are enabled or disabled (non-SI, non-OCV); and
- Scalability and portability of GTX to design projects in new foundry libraries.

Our main contributions are summarized as follows.

- We develop GTX by identifying appropriate modeling parameters, and by exploiting big-data mindsets and machine learning techniques to correct timing divergence between tools. To the best of our knowledge, our work is the first to attempt timing correlation with a big-data approach.
- Our models to correlate path slack between timing tools are accurate across multiple technology nodes and designs. In non-SI

[1]Detailed descriptions of the machine learning techniques used in this work can be found in [7].

[2]GTX uses the timing reports of T_1 to generate timing values that reduce divergence from T_2. Of course, GTX can also perform the reverse, i.e., use timing reports of T_2 to reduce divergence from T_1.

[3]We refer to path slacks at timing endpoints as, simply, path slacks.

[4]Throughout our paper, we refer to 45GS as 45nm, and to 28FDSOI as 28nm.

mode, our models reduce the range of divergence in path slack between tools from 32.5ps to 5.9ps (i.e., 5.5× reduction) at 28nm. In SI mode, our models reduce the range from 139.3ps to 21.1ps (i.e., 6.6× reduction) at 45nm. We demonstrate that our method applies to small as well as relatively large (*leon3mp*) designs.
- We demonstrate that GTX can reduce the number of outliers (from 407 to 26, i.e., 16× reduction, in the example we study) by incrementally modifying models when new designs are added.
- GTX can be applied to multiple designs, implementation flows, and technology nodes. We demonstrate the generality of GTX with two use cases – correlating two signoff tools, and correlating one signoff tool with a design implementation tool.

In the remainder of this paper, Section II surveys related work. Section III describes our modeling parameters and methodology for developing machine learning-based models for GTX. Section IV describes circuits used to generate training, validation and test sets used to develop models, and the design of experiments used to validate GTX. We also report results for multiple tools at multiple foundry nodes. Section V outlines future work and concludes this paper.

II. Related Work

Prior works that quantify miscorrelations between signoff STA tools or propose methodologies to minimize tool divergence are limited. Kahng et al. [8] develop an internal incremental STA tool by using least-squares regression to model wire delay. They then use offset-based correlation with a signoff timing tool to minimize divergence in path slack estimates of their incremental STA tool, relative to the signoff tool. Their models are developed using the ISPD-2013 [28] gate-sizing contest library, and do not include any models for stage or cell delays, or for flip-flop setup times.

To model effects of temporal and spatial manufacturing variations on path delay, Ganapathy et al. [5] use multivariate regression. They report estimation errors to be within 5% of SPICE simulations. Tetelbaum [14] uses root-sum-square (RSS) of variations in stage delay and a weighted function of the worst case sum of variations in stage delay to estimate total path delay; path delay estimation errors of less than 5% are reported. Sinha et al. [13] propose use of RSS for delay variation in their announcement of the TAU-2013 contest to speed up timing analysis by using multicores and parallel computing techniques.

In correlating STA tools, Mishra et al. [10] recalculate clock uncertainties based on miscorrelation between two tools and apply the updated uncertainty values to achieve better timing correlation between the tools. They do not empirically demonstrate the accuracy or efficiency of their approach, either in terms of runtime or in terms of the number of iterations taken to achieve acceptable correlation between the tools. Rakheja et al. [11] demonstrate that timing reports from design implementation tools, such as Synopsys *IC Compiler* [38], and signoff STA tools, such as Cadence *Encounter Timing System* [23], can differ. They propose a manual and iterative approach to fix paths for which the tools have large divergence in timing estimates. For SOCs, manual fixes are infeasible and automated approaches are required.

Motassadeq [9] quantifies differences in output slew between Synopsys *HSPICE* [37] and *PrimeTime* [39] for *Nonlinear Delay Model* (NLDM) and *Composite Current Source* (CCS) [24] delay models, but does not propose a methodology to reduce the divergence in slew estimates in tool reports between CCS and NLDM models.

III. Methodology

We now describe our methodology to develop flip-flop setup time, cell, wire, and stage delay and path slack models for GTX. We describe parameters used in the models, and then the machine learning methodology used to develop these models.

A. Parameter Selection

Signoff timing tools typically differ in path slack due to discrepancies in cell, wire and stage delays. Further, tools differ in their calculations of rise/fall delays across each input-to-output pin arc of cells. Figures 2 – 5 illustrate these discrepancies between two leading commercial signoff timing tools T_1 and T_2. Figure 5 in particular highlights the discrepancies between tools across a single MUX21 cell.

Path slack is calculated from the required setup time at the capture flip-flop of the path and from stage delays; these in turn are calculated from cell and wire delays in each stage. Figures 2 and 3 show that one tool (T_1) can be optimistic in cell delay reports and pessimistic in wire delay reports as compared to the other tool (T_2). There is a "canceling" effect for stage delays [8]. However, the "canceling" effect does not eliminate stage delay discrepancies between tools, as illustrated in Figure 4.

Table I lists all parameters used in our models. Note that cell and wire delays include incremental values for SI mode analysis.

TABLE I
Parameters reported by each tool in both SI and non-SI modes.

Parameter	Meaning	Mode
C_{eff}	Effective load capacitance	SI, non-SI
C_{coup}	Total coupling capacitances	SI
C_w	Wire ground capacitance	SI, non-SI
R_w	Wire resistance	SI, non-SI
$d_{tr,c,i}$	Cell input slew	SI, non-SI
$d_{tr,c,o}$	Cell output slew	SI, non-SI
d_c	Cell delay	SI, non-SI
d_w	Wire delay	SI, non-SI
d_{stg}	Total stage delay	SI, non-SI
$d_{su,\mathrm{ff}}$	Flip-flop setup time	SI, non-SI
$d_{slk,p}$	Path slack	SI, non-SI

B. Modeling Flow for GTX Models

To minimize divergence and achieve close correlation between signoff timing tools, we use a big-data approach and machine learning models. We do not reverse-engineer tools as licenses prohibit us from doing so; reverse-engineering can also become intractable because each tool implements millions of lines of legacy and black-box code. Instead, we develop machine learning-based models for GTX to correct the divergence in setup time, cell, wire, and stage delays and apply these models to fit path slack between two STA tools. In the following, we use the latest versions of two widely used commercial signoff tools, and show how reports from a tool T_2 can be used to develop models that estimate a tool T_1's reports. Our methodology is applicable to any pair of signoff or design implementation tools that can perform STA.

In non-SI mode, divergence between tools is typically smaller for wire delays than for cell delays, so we develop only a cell delay model.[5] In SI mode, however, wire delay divergence between tools can be significant due to differences in handling of crosstalk effects, so we model both cell and wire delays. Therefore, in both non-SI and SI modes we develop three (path slack, setup, and cell delay) models; additionally, in SI mode, we develop wire and stage delay models. Figure 6 shows the hierarchy of the five models in GTX and why we refer to our methodology as "deep". We use hierarchical rather than flat modeling for improved correlation and decreased range of divergence. Combining individual models of cell delay, wire delay, and setup time in an additive manner to estimate path slack can result in errors being added up as well. For example, Kahng et al. [8] use additive wire delay models that result in large divergence in path slack. Therefore, they invoke the golden timer at regular intervals to correct the path slack. Hierarchical modeling prevents errors being added linearly by applying an additional layer of modeling that provides a better fit to timing estimates. In the following discussion, $T_1(\cdot)$ and $T_2(\cdot)$ refer to values of parameter $(\cdot)$ respectively reported by T_1 and T_2.

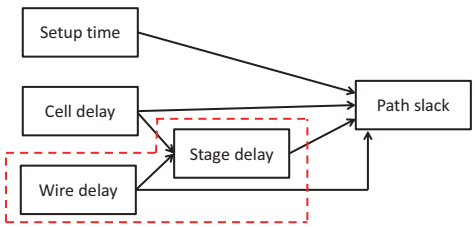


Fig. 6. Hierarchical GTX models. The models within the dotted lines are used only in SI mode.

Setup time. Our experiments in 28FDSOI indicate that flip-flop setup time reports between timing tools can diverge by up to 17.5ps. To reduce the divergence between tools, we model setup time as

$$\widehat{T_1}(d_{su,\mathrm{ff}}) = f\left(T_2(d_{su,\mathrm{ff}}, d_{tr,c,i})\right) \quad (1)$$

[5]Our experimental results indicate that by introducing wire delay models in non-SI mode, GTX results do not change significantly.

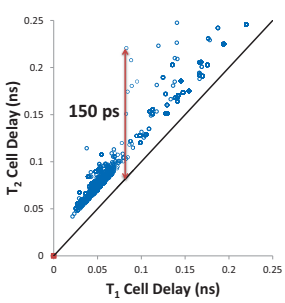


Fig. 2. Cell delay discrepancy.

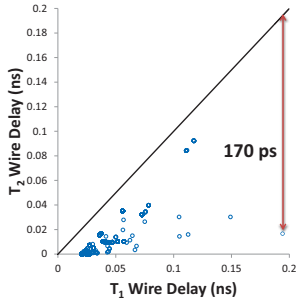


Fig. 3. Wire delay discrepancy.

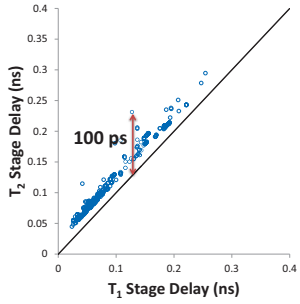


Fig. 4. Stage delay discrepancy.

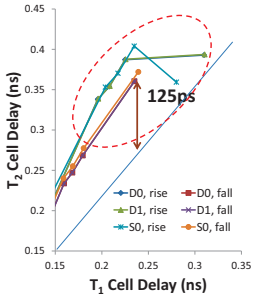


Fig. 5. Pin arc, rise/fall discrepancy.

where $\widehat{T_1}(d_{su,\mathrm{ff}})$ is the predicted T_1 setup time, $T_2(d_{tr,c,i})$ refer to the T_2-reported input slews at the D and clock pins of the capture flip-flop, and $f(\cdot)$ is the modeling function. These parameters correspond to those used to index the NLDM setup time tables in foundry libraries.

Cell delay. Our 28FDSOI studies also indicate that tools can differ in reported cell delays by >300ps (under extreme load and slew conditions).[6] Furthermore, the delay divergence between tools can vary across different input-to-output pin arcs, especially in complex cells such as *AOI* and *MUX*. In addition, tool reporting for rise and fall delays can diverge significantly. Figure 5 illustrates these divergences for rise and fall delays of *D0, D1 and S0* pins of a 2:1 *MUX*. With these considerations, we develop rise and fall delay models of each input-to-output pin arc of each cell in the design as

$$\widehat{T_1}(d_c) = f\left(T_2(d_c), \mathrm{LUT}(d_c)\right) \tag{2}$$

where $\widehat{T_1}(d_c)$ is the predicted T_1 cell delay, and $\mathrm{LUT}(d_c)$ is the cell delay determined using linear interpolation of NLDM delay lookup tables (LUTs) of a given cell [8]. The inputs for LUT interpolation are $T_2(C_{eff})$ and $T_2(d_{tr,c,i}) + \Delta Slew$, where $\Delta Slew$ is the upstream slew correction between the tools. We use $d_{tr,c,i}$ and C_{eff} because the NLDM delay tables in the foundry libraries are indexed by these. We use $\Delta Slew$ to correct upstream slew differences between the tools because our experiments indicate that certain tools always propagate the worst slew in path-based analysis mode. We model $\Delta Slew$ as

$$\Delta Slew = (\alpha(LUT(d_{tr,c,o}) + \beta) - T_2(d_{tr,c,o}) \tag{3}$$

where $LUT(d_{tr,c,o})$ is the output slew of the upstream cell calculated using linear interpolation between the library LUTs based on the T_2-reported $d_{tr,c,i}$ and C_{eff}. α and β are regression coefficients determined by fitting $T_2(d_{tr,c,o})$ of the upstream cell to $T_1(d_{tr,c,o})$.

Wire delay. We model wire delay, using a similar set of parameters as in [8], as

$$\widehat{T_1}(d_w) = f\left(T_2(d_w, d_{tr,c,o}), R_w \cdot \{C_w, C_{eff}, C_{coup}\}\right) \tag{4}$$

where $\widehat{T_1}(d_w)$ is the predicted T_1 wire delay and the parameters $R_w \cdot \{C_w, C_{eff}, C_{coup}\}$ represent delay due to different capacitances.

Stage delay. We model stage delay, using a similar set of parameters as in [13] and [14], as

$$\widehat{T_1}(d_{stg}) = f\left(T_2(d_{stg}), \widehat{T_1}(d_w, d_c)\right) \tag{5}$$

where $\widehat{T_1}(d_{stg})$ is the predicted T_1 stage delay.

Path slack. We develop two path slack models for non-SI and SI modes. The models are different because in SI mode, wire and stage delay models are required to correct large discrepancies in path slack as described above. Our path slack model in non-SI mode is

$$\widehat{T_1}(d_{slk,p})_{\tilde{SI}} = f\left(T_2(d_{slk,p}, \frac{\sigma}{\mu}(d_w)), \frac{\sigma}{\mu}(\widehat{T_1}(d_c, d_{su,\mathrm{ff}}))\right) \tag{6}$$

where $\widehat{T_1}(d_{slk,p})_{\tilde{SI}}$ is the predicted T_1 path slack in non-SI mode and $\frac{\sigma}{\mu}(\cdot)$ is the *coefficient of variation* of the parameter $(\cdot)$. Our path slack model in SI mode is

$$\widehat{T_1}(d_{slk,p})_{SI} = f\left(T_2(d_{slk,p}), \frac{\sigma}{\mu}(\widehat{T_1}(d_w, d_c, d_{stg}, d_{su,\mathrm{ff}}))\right) \tag{7}$$

where $\widehat{T_1}(d_{slk,p})_{SI}$ is the predicted T_1 path slack in SI mode.

Besides coefficient of variation, we also try two other normalization techniques, *standard score* [7] and *variance-to-mean* ratio [7]. We experimentally observe that coefficient of variation and standard score give similar results because they determine the contribution of each wire, cell, or stage delay to the overall delay of all wires, cells, or stages in a path. Variance-to-mean ratio, on the other hand, cannot determine the contribution of an individual (wire, cell, or stage) delay to the corresponding total delay in a given path; hence, it is less accurate.

Incremental modeling. Large product organizations often tape out multiple designs in the same technology. A new design can, conceivably, use cells and/or wiring configurations that are "out of scope" for the current fitted models. Such "new" cells/wires can introduce divergence in timing reports.[7] To mitigate these divergences, we propose an *incremental modeling* flow as follows.

- Step 1. Add any observations that result in divergence in timing of more than a threshold value (e.g., 10ps) to the existing training sets of each of the GTX models.
- Step 2. Re-train GTX models with the training sets from Step 1.
- Step 3. Test the updated models on all data points from the new design.

IV. Validation and Results

We now present validation of GTX and results of our experiments. First, we describe our design of experiments, including descriptions of designs used and our flow to collect training, validation and testing data for modeling. Second, we conduct four experiments to assess and measure performance of GTX. We use two leading (foundry-qualified) signoff timing tools T_1 and T_2, and a leading design implementation tool D_1, in our experiments. All tool versions are 2013 releases.

- **Experiment 1.** Correlate tools T_1 and T_2 in non-SI mode.
- **Experiment 2.** Correlate tools T_1 and T_2 in SI mode.
- **Experiment 3.** Correlate tools T_1 and D_1 in SI mode.
- **Experiment 4.** Validate the incremental modeling flow on a new design with many outliers.

A. Design of Experiments

We use real-world designs as well as artificial circuits in our experiments. Real-world designs include the *leon3mp* multicore processor from Aeroflex Gaisler AB [29], and *aes_cipher_top*, *wb_dma_top* and *jpeg_encoder* from Opencores [32]. We generate artificial training circuits to finely control various aspects of a timing path to verify robustness of our methodology.[8] We synthesize all designs with 45nm bulk triple-Vt and 28nm FDSOI dual-Vt foundry libraries. We perform hierarchical synthesis at 45nm and flat synthesis at 28nm to demonstrate the scalability of GTX across different flows and foundry technologies. We generate verilog netlists, Synopsys

[6] Simulations with HSPICE [37] indicate that T_1 is accurate to within 0.02ps of HSPICE results, whereas T_2 diverges more substantially from HSPICE.

[7] If new cells are not introduced in a design, incremental modeling is not required for GTX.

[8] We observe that synthesis and implementation tools tend to construct designs that occupy the middle region of delay tables. We create artificial training circuits to define the extreme ranges of timing discrepancies so as to create robust and scalable models.

Design Constraints (SDC) [1], and Standard Parasitic Exchange Format (SPEF) [40] files as inputs to timing tools.

Real-world designs. Table II shows the post-layout number of standard-cell instances for each design implemented in 45nm and 28nm foundry libraries. At 45nm, we use less strict constraints on timing, maximum fanouts, and transition, and we restrict tools from using cell sizes $X0$, $X1$, and $\geq X20$.[9] However, at 28nm we allow the tools to use all cells from the library, and apply tight timing constraints but relaxed maximum fanout and transition constraints.[10]

TABLE II
NUMBER OF INSTANCES IN REAL-WORLD DESIGN.

	# Instances (clock period in ns)	
Testcase	45nm	28nm
aes_cipher_top	18818 (1.0)	16688 (0.8)
wb_dma_top	3641 (0.5)	2349 (0.5)
jpeg_encoder	46702 (1.25)	53641 (0.67)
leon3mp	–	750854 (1.2)

Artificial training circuits. We develop generators using custom *Tcl* scripts to finely control various aspects of a timing path as listed below.

- Path – #stages and #fanouts.
- Cell – input slews, types, sizes, and Vt flavors.
- Wire – parasitics (R_w, C_w, C_c), #segments, and aggressors.

CPU time needed to generate the verilog netlist, SDC, and SPEF files is ∼6s (independent of the number of fanouts and stages) on an Intel Xeon E5-2640 2.5GHz server. The size of each of these files is ∼4KB for a circuit with one stage and a fanout of one. The size of SPEF files can potentially be large (e.g., 232KB for a circuit with 60 stages and four fanouts in each stage) because we do not implement name mapping.

Each training circuit consists of a chain of *driver* and *driven* cells and flip-flops at the beginning (launch) and the end (capture) to create a *constrained path*. Optionally, cells can be added to achieve multiple fanouts from each driver. Pins that are not on the constrained path are connected to dummy flip-flops and/or ports to ensure that there are no floating pins. An example of a circuit with two *stages* without SI aggressors is shown in Figure 7. The *constrained path* is from *f1/Q* to *f2/D*, through instances *u1* and *u2*. To generate a training circuit with multiple stages, the "repeated unit" in Figure 7 is replicated between the launch and the capture flip-flops[11]. Figure 8 illustrates a circuit with one SI aggressor and coupling capacitances.

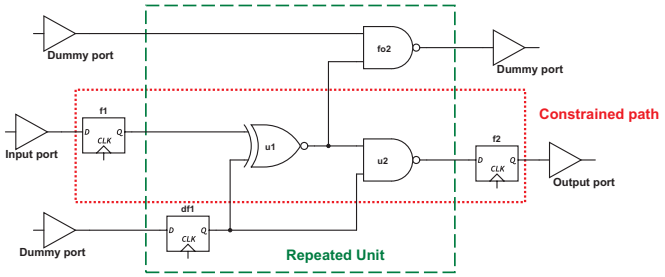


Fig. 7. Example of a non-SI training circuit.

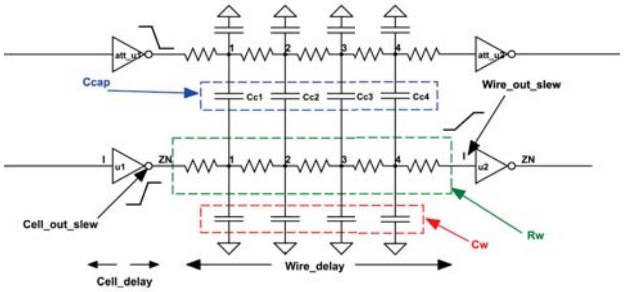


Fig. 8. Example of an SI training circuit.

[9]We observe that these cell sizes are known for being problematic in designs; some designers commonly use similar restrictions.

[10]At 45nm, the maximum fanout constraint is set to 20 and the maximum transition time is set to one-sixth of the clock period. At 28nm these values are respectively 40 and one-eighth of the clock period.

[11]The "repeated unit" contains a dummy flip-flop which is inserted to ensure valid operation of gates, and is not part of the *constrained path*.

B. Data Collection for Modeling

We generate training, validation, and test datasets in the following way. First, we obtain verilog netlists, SDC, and SPEF files with coupling capacitances for our designs. Second, we use 2013-released versions of two commercial signoff timing tools, commonly adopted as golden tools by design teams, to perform path-based timing analysis of the top 10K worst paths in both SI and non-SI modes.[12] For Experiment 3, we use a commercial design implementation tool D_1. Last, to compare tools in a fair manner, we ensure that options and global flags for both tools are set to produce similar reports as follows:

- Timing reports. Each tool reports all parameters from Table I.
- Path timing calculation. Each tool performs path-based analysis, i.e., slews are propagated only along "paths-of-interest".
- SI and OCV analyses. SI- and OCV-aware analysis modes are enabled, and glitch analysis is disabled.[13]
- Parasitic information. In SI mode, each tool uses coupling parasitic information for timing analysis.

Detailed cell characterization for the cell delay model. We perform a one-time detailed characterization of each input-to-output pin arc of each cell in a design because our experiments indicate that cell delay requires very detailed modeling to minimize the range of errors.[14] We create a single-stage artificial training circuit for the cell, annotate multiple input slews and capacitances spanning the entire NLDM delay tables in the foundry libraries used by the design, and obtain rise and fall delays for each combination of slews and capacitances and for all rise and fall input transitions. Similar characterization is performed for flip-flops as well. Table III shows sample resource utilization for cell characterization for a design implemented with 28nm foundry libraries. File size refers to the file with training, validation and test datasets for each cell.

TABLE III
RESOURCE UTILIZATION FOR CELL CHARACTERIZATION AT 28NM.

Cell	#arcs	#data points	Time (min)	File size (KB)
INV	1	140	20	20
NAND2	2	280	55	36
MUX21	3	560	95	68
AOI13	4	560	95	68

We characterize a total of 397 cells at 28nm and 305 cells at 45nm libraries; these contain a total of 1870 input-to-output pin arcs[15]. The characterization time for these cells is 116h per core (a one-time overhead of just under 5 days) on an eight-core Intel Xeon E5-1410 2.8GHz server. Table III shows resource utilization for cell characterization at 28nm. *MUX21* and *AOI13* cells have the same runtime and number of training data points because NLDM table sizes vary between these cells, and we use more values of input slews and capacitances from the NLDM tables of *MUX21* than of *AOI13*.

Modeling techniques. To develop models, we use training data points from artificial circuits and validation data points from real-world designs. To test the models, we use a separate set of data points from our real-world designs. Table IV shows the sizes of the training, validation and test sets for each experiment. Extremely large sizes of our training and test sets reflect our "big-data" approach whereby models are derived using ≥200K data points for cell, wire, and stage delays. Thereafter, we may apply our incremental modeling flow for new designs in the same technology/library.

We apply both linear and nonlinear machine learning techniques (least-square regression (LSQR), artificial neural networks (ANN) [7], support vector machines regression (SVMR) [4] with radial basis function kernel, and random forests (RF) [7]) to all GTX models. For each model, we choose the technique that best minimizes both mean squared error (MSE) and the range of errors, i.e., the difference between maximum and minimum errors. We observe that LSQR and

[12]We use custom *Tcl* scripts to ensure that the same 10K paths are analyzed by respective tools as we generate our training sets.

[13]Our experiments indicate that in both OCV and non-OCV modes the divergence in clock-to-Q delay and setup times vary by less than 5ps, and delays for other cells vary by less than 1ps. Therefore, in the following we report results in OCV mode only.

[14]When signoff involves multiple corners, cell delays need to be characterized for each corner, and the corner-specific timing model must be used.

[15]We characterize only those cells used in our designs. If necessary, an entire library can be characterized.

ANN are not as effective as RF and SVMR in minimizing the range of errors. ANN is effective in modeling setup time and cell delays, SVMR is effective in modeling wire and stage delays, and RF is effective in modeling path slack. We use the built-in *Matlab vR2013a* [30] toolbox for ANN, *LIBSVM* implementation of SVMR in *Matlab* [4], and an open-source *Matlab* implementation of RF [34].[16] Once models are developed, the time to test a model depends on the size of the test dataset. In our experiments, runtime is ∼3.23s for 30K path slack data points in the test set. Figure 9 shows our complete modeling flow for GTX. Note that, by default, model development is a one-time effort. New designs may require incremental modeling to reduce the number of outliers.

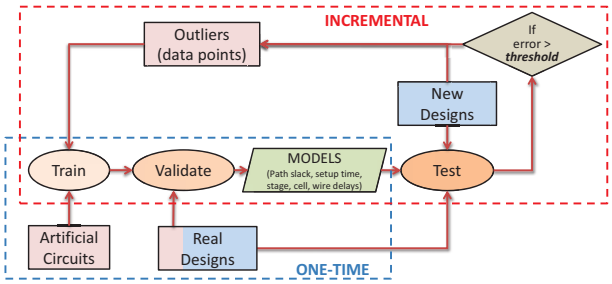


Fig. 9. Our modeling flow.

TABLE IV
TRAINING, VALIDATION AND TEST DATASET SIZES

Experiment #	Module	Training	Validation	Testing
1	Path slack	22680	6480	33240
	Setup time	21798	6228	33114
	Cell delay	354320	15520	326760
2	Path slack	17270	7664	34066
	Setup time	28830	8236	34120
	Cell delay	323804	9875	315776
	Wire delay	304108	39788	143941
	Stage delay	323880	39872	184560
3	Path slack	21770	1440	35790
	Setup time	21540	1120	35340
	Cell delay	320118	11346	332613
	Wire delay	215506	9980	156774
	Stage delay	211736	10553	139327
4	Path slack	17554	5166	32616
	Setup time	28840	8237	34989
	Cell delay	341042	29972	100387
	Wire delay	344086	29900	100520
	Stage delay	341708	29926	98895

C. Results for Experiments

We validate GTX with the four experiments described in Section IV.[17] All experiments are performed on an Intel Xeon E5-2640 2.5GHz server and all reported runtimes are for this platform.

Results for Experiment 1. We correlate timing between T_1 and T_2 in non-SI mode. Figures 10(a) and 10(b) show the timing divergence between tools before and after fitting. The total runtime is 38min.[18] For ANN, we use up to five hidden layers to model cell delay and two hidden layers to model setup time. We constrain RF to 200 trees and 5000 observations per leaf node. Our models reduce the range of divergence in path slack from 32.5ps to 5.9ps (i.e., 5.5× reduction) at 28nm, and from 18.8ps to 7.1ps (i.e., 2.6× reduction) at 45nm.

[16]ANN uses hidden layers as a modeling parameter. We sweep the number of hidden layers from one to ten and choose the value that achieves minimum MSE and range of errors. RF uses multiple classification trees and applies different models to a set of observations at a leaf node of each tree [7]. We sweep the number of trees from 50 to 500 in steps of 50, and the number of observations per leaf node ranging from 1000 to 10000 in steps of 1000. For each experiment, we report the number of trees and the number of observations per leaf node that minimizes MSE and the range of errors.

[17]We ensure that identical input files (Liberty, netlist, SDC and SPEF) are provided to both tools, such that slack miscorrelation is due to delay and timing calculation only. Thus, in Experiment 3 we do not use, e.g., Cadence *Ostrich* [33] to perform parasitic correlation with golden SPEF from Synopsys *StarRC* [36], in which case the design implementation tool's (D_1) parasitic estimation may be another source of miscorrelation.

[18]The reported runtimes for experiments do not include cell characterization time, which is separately discussed in Section IV-B.

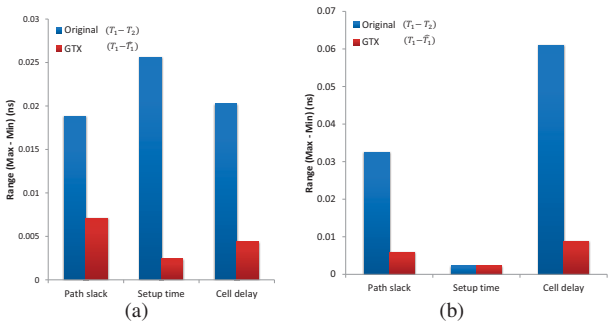


Fig. 10. Experiment 1 results at (a) 45nm and (b) 28nm.

Results for Experiment 2. We correlate timing between T_1 and T_2 in SI mode. Figures 11(a) and 11(b) show the divergence between tools before and after fitting. The total runtime is 116min. For ANN, we use up to seven hidden layers to model cell delays and two hidden layers for setup time. We constrain RF to 400 trees and 2000 observations per leaf node as we observe that this selection minimizes the range of errors. Our models reduce the range of divergence in path slack from 89.2ps to 22.3ps (i.e., 4× reduction) at 28nm and from 139.3ps to 89.2ps (i.e., 6.6× reduction) at 45nm. The stage delay model in GTX improves accuracy even when path slack diverges by >130ps.

To confirm the robustness of our approach, we also conduct the inverse experiment, i.e., where we use timing reports of T_1 to estimate timing reports of T_2. The error metrics are comparable to those shown in Figures 11(a) and 11(b). Figures 12(a) and 12(b) depict five stages from a 28-stage path (Path #2197) from *jpeg_encoder*, with cell delays, wire delays and path slack reported by T_1 and T_2, and their respective fitted values $\hat{T}_1$ and $\hat{T}_2$ from GTX. The fitted values are within 8ps of the tool-reported values.

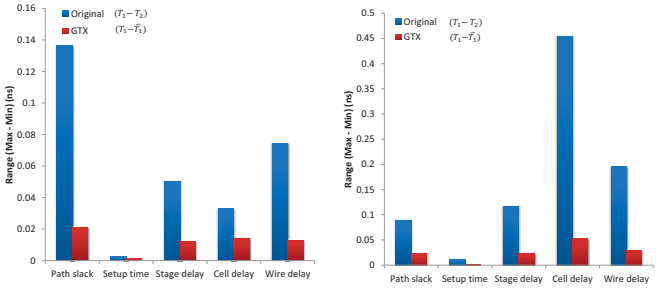


Fig. 11. Experiment 2 results at (a) 45nm and (b) 28nm.

Results for Experiment 3. We correlate timing between T_1 and a leading design implementation tool D_1 in SI mode at 28nm. Figure 13 shows the divergence between tools before and after fitting. The total runtime is 104min. For ANN, we use up to seven hidden layers to model cell delay and five hidden layers for setup time. We constrain RF to 450 trees and 4000 observations per leaf node. Our models reduce the range of divergence in path slack from 162.8ps to 23.1ps (i.e., 7× reduction).

Results for Experiment 4. We incrementally refine our models for a new design with many outliers while correlating timing parameters. A new design, 5× *jpeg_encoder*, is derived from the original *jpeg_encoder* design [32]. We create a new top module that instantiates the original *jpeg_encoder* module five times to obtain 5× *jpeg_encoder*. The new design is implemented at 28nm and has ∼300K cell instances in the post-layout netlist. We use a tighter timing constraint for this design than with *jpeg_encoder*, which results in different cells and timing paths being used. Change in top-level routing across each *jpeg_encoder* block also changes wire delay due to crosstalk effects. Therefore, 5× *jpeg_encoder* requires modification of the models derived for *jpeg_encoder*. Figure 14 shows the divergence between tools before and after incremental fitting for path slack, cell, wire and stage delays. The total runtime is 87min. For ANN, we use up to seven hidden layers to model cell and stage delays and two hidden layers for setup time. We constrain RF to 400 trees and 2000 observations per leaf node. We do not report setup time because the divergence is <3ps. The total runtime is 177min. In the context of a

Instance	(Cell)	Dir	Delay(T_1)	Delay(T_2)	Delay($\widehat{T_1}$)	
FE_CN_C274/A1	(NAND2X7)	IN	0.0447	0.0062	0.0452	r
FE_CN_C274/ZN	(NAND2X7)	OUT	0.0565	0.1076	0.0545	f
FE_CN_C277/A	(BUFFX8)	IN	0.0110	0.0044	0.0082	r
FE_CN_C277/Z	(BUFFX8)	OUT	0.0272	0.0664	0.0266	r
FE_CN_C281/A	(INVX8)	IN	0.0057	0.0023	0.0051	f
FE_CN_C281/ZN	(INVX8)	OUT	0.0215	0.0264	0.0213	r
FE_CN_C286/A2	(XOR2X4)	IN	0.0070	0.0066	0.0072	f
FE_CN_C286/Z	(XOR2X4)	OUT	0.0332	0.0581	0.0352	f
FE_CN_C294/A1	(OAI22X4)	IN	0.0825	0.0225	0.0837	r
FE_CN_C294/ZN	(OAI22X4)	OUT	0.0677	0.0781	0.0598	f
slack (VIOLATED)			-0.339	-0.588	-0.342	

(a) T_2 fitted to T_1

Instance	(Cell)	Dir	Delay(T_1)	Delay(T_2)	Delay($\widehat{T_2}$)	
FE_CN_C274/A1	(NAND2X7)	IN	0.0447	0.0062	0.0065	r
FE_CN_C274/ZN	(NAND2X7)	OUT	0.0565	0.1076	0.1063	f
FE_CN_C277/A	(BUFFX8)	IN	0.0110	0.0044	0.0050	r
FE_CN_C277/Z	(BUFFX8)	OUT	0.0272	0.0664	0.0631	r
FE_CN_C281/A	(INVX8)	IN	0.0057	0.0023	0.0026	f
FE_CN_C281/ZN	(INVX8)	OUT	0.0215	0.0264	0.0260	r
FE_CN_C286/A2	(XOR2X4)	IN	0.0070	0.0066	0.0057	f
FE_CN_C286/Z	(XOR2X4)	OUT	0.0332	0.0581	0.0588	f
FE_CN_C294/A1	(OAI22X4)	IN	0.0825	0.0225	0.0231	r
FE_CN_C294/ZN	(OAI22X4)	OUT	0.0677	0.0781	0.0794	f
slack (VIOLATED)			-0.339	-0.588	-0.582	

(b) T_1 fitted to T_2

Fig. 12. Five sample stages from a 28-stage path in *jpeg_encoder* at 28nm showing cell delay (OUT), wire delay (IN) and path slack reported by T_1 and T_2. The respective fitted values after using GTX are (a) Delay($\widehat{T_1}$) and (b) Delay($\widehat{T_2}$) when T_1 or T_2 is the respective fitted tool. All values are in ns.

new chip design project, this overhead of several hours is negligible. Our models reduce the range of divergence in path slack from 89.2ps to 36ps (2.5×), and the number of outliers from 407 to 26 (i.e., 16× reduction).

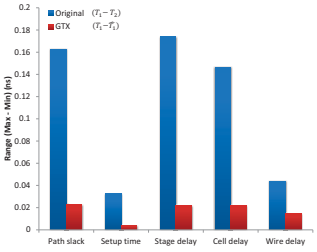


Fig. 13. Expt 3 results at 28nm.

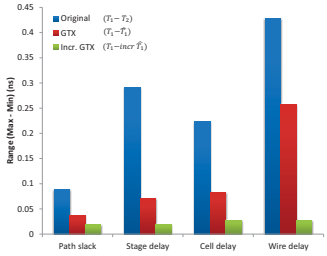


Fig. 14. Expt 4 results at 28nm.

V. Conclusions

Improvements to timing signoff methodologies can significantly reduce the number of iterations in the IC design flow. Design teams often want to correlate one signoff tool's timing reports with those of another tool to reduce pessimism and/or optimism. We describe a new tool, GTX, that embodies a big-data approach for the correlation problem using a hierarchy of models. We apply machine learning to develop models for path slack, setup time, stage, cell, and wire delays and can "correct" endpoint path slack divergence between two signoff timers from 89.2ps to 22.3ps (i.e., 4× reduction) at 28nm, and from 139.3ps to 21.1ps (i.e., 6.6× reduction) at 45nm with SI and OCV analysis enabled. GTX can also be applied to improve timing correlation between an implementation and a signoff tool; our experiments show 7× reduction of path slack divergence from 162.8ps to 23.1ps. We show that GTX scales to multiple foundry nodes and libraries, and that incremental modeling in GTX provides the capability to adapt to new designs in a given technology. Our ongoing work seeks three improvements: (i) expand GTX to CCS [24] models and statistical variation-aware analysis [15]; (ii) develop methodologies to characterize libraries for "ideal" delay and power per unit length; and (iii) develop a methodology to integrate GTX into arbitrary production timing closure flows so as to reduce the amounts of iterations, turnaround time and overdesign needed to achieve final timing signoff.

Acknowledgments

We thank Tom Spyrou, Dr. Cho Moon, Roger Embree, and Dr. Puneet Sharma for their early feedback on our project. We thank Gary Smith of Gary Smith EDA for his listing of timing analysis tool providers. We also thank CMP and STMicroelectronics for access to the 28nm FDSOI design kit.

References

[1] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.
[2] S. Bansal and R. Goering, "Making 20nm Design Challenges Manageable", http://www.chipdesignmag.com/pdfs/chip_design_special_DAC_issue_2012.pdf
[3] T.-B. Chan, A. B. Kahng, J. Li and S. Nath, "Optimization of Overdrive Signoff", *Proc. ASP-DAC*, 2013, pp. 344-349.
[4] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines", *ACM Trans. on Intelligent Systems and Technology* 3(2) (2011), pp. 27:1-27:27.
[5] S. Ganapathy, R. Canal, A. Gonzalez and A. Rubio, "Circuit Propagation Delay Estimation Through Multivariate Regression-Based Modeling Under Spatio-Temporal Variability", *Proc. DATE*, 2010, pp. 417-422.
[6] R. Goering, "What's Needed to "Fix" Timing Signoff?", *DAC Panel*, 2013.
[7] T. Hastie, R. Tibshirani and J. J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.
[8] A. B. Kahng, S. Kang, H. Lee, S. Nath and J. Wadhwani, "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. SLIP*, 2013.
[9] T. El Motassadeq, "CCS vs NLDM Comparison Based on a Complete Automated Correlation Flow Between PrimeTime and HSPICE", *Proc. Saudi International Electronics, Communications and Photonics Conference*, 2011, pp. 1-5.
[10] A. Mishra, J. Kumar and U. Singhal, *Resolving Timing Miscorrelation Using Timing Uncertainties.* http://www.edn.com/design/integrated-circuit-design/4390721/Resolving-timing-miscorrelation-using-timing-uncertainties
[11] S. Rakheja and N. S. Krishna, *Establishing Timing Correlation Between Tools.* http://www.edn.com/design/integrated-circuit-design/4313674/Establishing-timing-correlation-between-tools
[12] R. Salakhutdinov, J. B. Tenenbaum and A. Torralba, "Learning with Hierarchical-Deep Models", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 35(8) (2013), pp. 1958-1971.
[13] D. Sinha, L. G. e Silva, J. Wang, S. Raghunathan, D. Netrabile and A. Shebaita, "TAU 2013 Variation Aware Timing Analysis Contest", *Proc. ISPD*, 2013, pp. 171-178.
[14] A. Tetelbaum, "Method of Estimating a Total Path Delay in an Integrated Circuit Design with Stochastically Weighted Conservatism", *U.S. Patent* No. 7,213,223, 2007.
[15] V. Veetil, K. Chopra, D. Blaauw and D. Sylvester, "Fast Statistical Static Timing Analysis Using Smart Monte Carlo Techniques", *IEEE Trans. on CAD* 30(6) (2011), pp. 852-856.
[16] C. Moon, Synopsys Inc., *personal communication*, July 2013.
[17] G. Smith, Gary Smith EDA, *personal communication*, September 2013.
[18] P. Sharma, Freescale Inc., *personal communication*, July 2013.
[19] R. Embree, *personal communication*, July 2013.
[20] T. Spyrou, Altera Corporation, *personal communication*, July 2013.
[21] *Atrenta Inc.* http://www.atrenta.com
[22] *Cadence Design Systems.* http://www.cadence.com
[23] *Cadence Encounter Timing System.* http://www.cadence.com/products/di/ets/pages/default.aspx
[24] *CCS.* http://www.opensourceliberty.org/ccspaper/ccs_bgr.pdf
[25] *CLK Design Automation Inc.* http://www.clkda.com
[26] *Google Translate.* http://translate.google.com
[27] *Incentia Design Systems Inc.* http://www.incentia.com
[28] *Discrete Gate Sizing Contest.* http://www.ispd.cc/contests/13/ispd2013_contest.html
[29] *Leon3 Multicore Processor.* http://www.gaisler.com/index.php/products/processors/leon3
[30] *MATLAB.* http://www.mathworks.com
[31] *Mentor Graphics Inc.* http://www.mentor.com
[32] *OpenCores.* http://opencores.org/projects
[33] *Ostrich.* http://www.cadence.com/community/blogs/di/archive/2008/10/15/an-interview-with-global-timing-debug-architect-thad-mccraken.aspx
[34] *Random Forest.* https://code.google.com/randomforest-matlab
[35] *Apple Siri.* http://www.apple.com/ios/siri
[36] *Synopsys Inc.* http://www.synopsys.com
[37] *Synopsys HSPICE User Guide.* http://www.synopsys.com/tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx
[38] *Synopsys IC Compiler User Guide.* http://www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/ICCompiler.aspx
[39] *Synopsys PrimeTime User Guide.* http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx
[40] *SPEF.* http://www.edaboard.com/thread37705.html