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@mjnu.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, Golden Timer eXtension (GTX), to correct divergence in flip-flop setup time, delay, stage delay, and path slack at timing endpoints between tools. 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 number of 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], Incenta 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 overdesigned or underdesigned, 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.

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 mismiscorrelations between two leading commercial signoff timing tools T1 and T2, as well as between T1 and a commercial design implementation tool D1. 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 techniques1 to correct2 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 endpoints3 between two tools within a range of <30ps for designs implemented in 28FDSDOI and 45GS foundry libraries4 using NLD and 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

1Detailed descriptions of the machine learning techniques used in this work can be found in [7].
2GTX uses the timing reports of T1 to generate timing values that reduce divergence from T2. Of course, GTX can also perform the reverse, i.e., use timing reports of T2 to reduce divergence from T1.
3We refer to path slacks at timing endpoints as, simply, path slacks.
4Throughout our paper, we refer to 45GS as 45nm, and to 28FDSDOI 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 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. Tettelbaum [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 Synopsis HSPICE [37] and PrimeTime [39] for Nonlinear Delay Model (NLDMA) 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 NLDMA 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 2–3 illustrate these discrepancies between two leading commercial signoff timing tools. Therefore, to reduce the divergence between tools, we model setup time as

\[ T_{\text{setup}}(d_{\text{in}}, d_{\text{out}}) = f(T_1 + T_2, d_{\text{in}}, d_{\text{out}}, d_{\text{delay}}) \]  

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

Setup time. Our experiments in 28FDOSOI 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

\[ T_{\text{setup}}(d_{\text{in}}, d_{\text{out}}) = f(T_1 + T_2, d_{\text{in}}, d_{\text{out}}, d_{\text{delay}}) \]  

2Our experimental results indicate that by introducing wire delay models in non-SI mode, GTX results do not change significantly.
where $\hat{T}_1(d_{u,t})$ is the predicted $T_1$ setup time, $T_2(d_{r,c})$ refer to the $T_2$-reported input slew 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). 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

$$\hat{T}_1(d_e) = f(T_2(d_e), \text{LUT}(d_e))$$  \hspace{1cm} (2)

where $\hat{T}_1(d_e)$ is the predicted $T_1$ cell delay, and LUT($d_e$) 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 $T_2(d_{r,c})$ and $\Delta$Slew, where $\Delta$Slew is the upstream slew correction between the tools. We use $d_{r,c}$ 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

$$\Delta\text{Slew} = (\alpha(\text{LUT}(d_{r,c,o})+\beta) - T_2(d_{r,c,o})$$  \hspace{1cm} (3)

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

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

$$\hat{T}_1(d_w) = f(T_2(d_w, d_{r,c,o}), R_w \cdot \{C_w, C_{eff}, C_{coup}\})$$  \hspace{1cm} (4)

where $\hat{T}_1(d_w)$ 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

$$\hat{T}_1(d_{a,g}) = f(T_2(d_{a,g}), \hat{T}_1(d_w, d_c))$$  \hspace{1cm} (5)

where $\hat{T}_1(d_{a,g})$ 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

$$\hat{T}_1(d_{t,k},p)_{sl} = f(T_2(d_{t,k,p}), \frac{\sigma}{\mu} (\hat{T}_1(d_w, d_{a,g}, d_{u,t})))$$  \hspace{1cm} (6)

where $\hat{T}_1(d_{t,k},p)_{sl}$ 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

$$\hat{T}_1(d_{t,k},p)_{SI} = f(T_2(d_{t,k,p}), \frac{\sigma}{\mu} (\hat{T}_1(d_w, d_{a}, d_{a,g}, d_{u,t})))$$  \hspace{1cm} (7)

where $\hat{T}_1(d_{t,k},p)_{SI}$ 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. 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. We synthesize all designs used and our flow to define the extreme ranges of timing discrepancies so as to create robust and scalable models.

1If new cells are not introduced in a design, incremental modeling is not required for GTX.

2We 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] 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 ≥ X20. 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

<table>
<thead>
<tr>
<th>Design</th>
<th># Instances (clock period in ns)</th>
<th>SDC</th>
<th>SPEF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>494</td>
<td>2349</td>
<td>2349</td>
</tr>
<tr>
<td><strong>area_capint_top</strong></td>
<td>1800.1 (1.0)</td>
<td>10898 (0.8)</td>
<td>10898</td>
</tr>
<tr>
<td><strong>wb_fifo_top</strong></td>
<td>3641 (0.3)</td>
<td>2349 (0.5)</td>
<td>2349</td>
</tr>
<tr>
<td><strong>jpeg_encoder</strong></td>
<td>46702 (1.25)</td>
<td>59641 (0.67)</td>
<td>59641</td>
</tr>
<tr>
<td><strong>iwiltsim</strong></td>
<td>950854 (1.2)</td>
<td>950854 (1.2)</td>
<td>950854</td>
</tr>
</tbody>
</table>

**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 (Rv, Cv, Cc), #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-flops11. Figure 8 illustrates a circuit with one SI aggressor and coupling capacitances.

![Fig. 7. Example of a non-SI training circuit.](image)

![Fig. 8. Example of an SI training circuit.](image)

**Table III** RESOURCE UTILIZATION FOR CELL CHARACTERIZATION AT 28NM.

<table>
<thead>
<tr>
<th>Cell</th>
<th># Inst</th>
<th>Data points</th>
<th>Time (min)</th>
<th>File size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>1</td>
<td>140</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>NAND2</td>
<td>2</td>
<td>280</td>
<td>55</td>
<td>36</td>
</tr>
<tr>
<td>MUX21</td>
<td>5</td>
<td>500</td>
<td>95</td>
<td>68</td>
</tr>
<tr>
<td>AOI13</td>
<td>4</td>
<td>500</td>
<td>95</td>
<td>68</td>
</tr>
</tbody>
</table>

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 arcs12. 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

10We 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 r2013a [30] toolbox for ANN, LIBSVM implementation of SVMR in Matlab [4], and an open-source Matlab implementation of RF [34]. 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.

C. Results for Experiments

We validate GTX with the four experiments described in Section IV. 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 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., \( 4 \times \) reduction) at 28nm and from 139.3ps to 89.2ps (i.e., \( 6.6 \times \) reduction) at 45nm. The stage delay model in GTX improves accuracy even when path slack diverges by \( >100ps \).

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 5ps of the tool-reported values.

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. Our models reduce the range of divergence in path slack from 89.2ps to 22.3ps (i.e., \( 4 \times \) reduction) at 28nm and from 139.3ps to 89.2ps (i.e., \( 6.6 \times \) reduction) at 45nm.

Results for Experiment 3. We correlate timing between \( T_1 \) and a leading design implementation tool \( D_j \) 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., \( 7 \times \) reduction).

Results for Experiment 4. We incrementally refine our models for a new design with many outliers while correlating timing parameters. A new design, \( 5 \times \) 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 \( 5 \times \) jpeg_encoder. The new design is implemented at 28nm and has \( \sim 300K \) 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, \( 5 \times \) 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
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 × reduction). The respective fitted values after using GTX are (a) Delay(\hat{T}_1) and (b) Delay(\hat{T}_2) when T_1 or T_2 is the respective fitted tool. All values are in ns.

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 of another tool to reduce pessimism and/or optimism. We describe a method to develop models for path slack, setup time, stage, cell, and wire delays 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 CCC [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

[34] Synopsys Inc. http://www.synopsys.com
[38] SPEF. http://www.edaonline.com/thread37705.html