Quantifying Error: Extending Static Timing Analysis with Probabilistic Transitions

Kevin E. Murray*, Andrea Suardi†, Vaughn Betz*, George Constantinides†
* Electrical and Computer Engineering, University of Toronto
† Electrical and Electronic Engineering, Imperial College London
{kmurray,vaughn}@eecg.utoronto.ca {a.suardi,g.constantinides}@imperial.ac.uk

***Abstract*—Timing analysis is a cornerstone of the digital design process. Statistical Static Timing Analysis was introduced to reduce pessimism by modelling device delay variations. However it ignores circuit logic, which may cause some timing paths to never or only rarely be sensitized. We introduce a general timing analysis approach and tool to calculate the probability that individual timing paths are sensitized, enabling the calculation of bounding delay distributions over all input combinations. We show the connection to the well-known #SAT problem and present approaches to improve scalability, achieving average results 46 to 32% less pessimistic than Static Timing Analysis while running 14.6 to 44.0 times faster than Monte-Carlo timing simulation.**

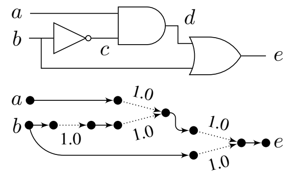

Fig. 1: Example circuit and timing graph with annotated delays.

TABLE I: PATH-DELAY DISTRIBUTION.

Active Path	Delay	Probability
$b \to c \to d \to e$	3.0	0.125
$a \to d \to e$	2.0	0.1875
$b \to e$	1.0	0.3125
None (constant output)	0.0	0.375

I. Introduction

Timing analysis, determining how quickly a synchronous circuit can operate reliably, is a cornerstone of digital design used to guide optimization and verify performance and correctness. Static Timing Analysis (STA) [1] is the conventional approach to performing timing analysis. However increasing variation due to smaller process technology causes the worst case delay to deviate significantly from the average case, and can make conventional STA very pessimistic [2], [3]. Statistical Static Timing Analysis (SSTA) reduces this pessimism by modelling device delay variation, allowing designers to sacrifice a small amount of device coverage (and resulting yield) for considerable performance improvement [3]. However, like STA, SSTA calculates the worst case across all input combinations.

We propose a complementary approach which quantifies the stochastic variation across *inputs* rather than devices, with the aim of sacrificing a small amount of input combination coverage to achieve considerable performance improvement on the remaining combinations.

This is motivated by applications amenable to approximate computing, where small errors may be acceptable (e.g. decoding lossy video), correctable, or where the input data is inherently noisy (e.g. sensor readings) and extreme accuracy is unwarranted [4]. By running devices beyond their strictly robust operating regimes, it is hoped that better trade-offs between power, area and performance can be achieved. For instance [5] studied the impact of 'overclocking' arithmetic operators beyond their 'maximum' safe operating frequencies for improved performance, and inspired changes to the architecture of arithmetic components [6].[1]

However designing such systems is challenging as their behaviour is difficult to analyze. This is particularly true at the circuit-level where conventional tools like STA and SSTA assume worst-case switching behaviour to ensure coverage of all input combinations. In [5] timing analysis was performed by hand, a method which is time consuming, error prone and not scalable. To the best of our knowledge there has been no generic approach to the timing analysis of this kind of design.

[1]In practice the entire design may not be error tolerant (e.g. control signals). In an overclocking context such signals must have sufficient slack to operate reliably at the overclocked frequency.

We aim to address this design capability gap by developing a generalization of STA. Instead of assuming worst-case switching behaviour to generate a longest path delay bound as in STA, we determine a bounding delay distribution over all input combinations.[2] Automating this process bridges the gap between the physical and logic design domains, opening new avenues for co-design and optimization.

To simplify matters we consider the case where the set of inputs at cycles i and $i+1$ are independent, identically uniformly distributed and statistically stationary. All these restrictions can be lifted by appropriate pre-processing without changing the approach presented in this paper.

Our key contributions include:

- a new timing analysis formulation to determine bounding delay distributions across input combinations,
- reduction of this analysis to #SAT,
- techniques to improve scalability on real circuits, and
- experimental comparison with Monte-Carlo simulation.

Section II discusses background and related work. Sections III and IV present our formulation and implementation. Sections V and VI describe the experimental methodology and results. Section VII concludes and outlines future work.

II. Background & Related Work

The conventional approach for performing timing analysis is STA [1]. STA performs a pessimistic analysis by considering only the topological structure of the circuit, pessimistically assuming that all signals switch every cycle. This topological structure is stored as a timing graph:

[2]This differs from conventional SSTA, where delay distributions are derived from device and interconnect delay variations.

Definition 1 (Timing Graph)
A directed graph where nodes represent the pins of circuit elements, edges represent timing dependencies and edge weights correspond to delays.

The circuit's primary inputs and state element outputs (e.g. Flip-Flop q pins) become *timing sources*: nodes with no fan-in. We denote the number of timing sources by I. Conversely, the primary outputs and state element inputs (e.g. Flip-Flop d pins) become *timing endpoints*: nodes with no fan-out. An example timing graph is shown in Fig. 1. The timing sources are inputs a and b ($I = 2$), and the output e is the single timing endpoint.

We can now define a timing path:

Definition 2 (Timing Path)
A path in the timing graph between a timing source and timing endpoint.

STA calculates the delay of the longest (or critical) timing paths, by calculating the latest (worse-case) *arrival time* of signals at each node in the graph. In Fig. 1, the path $b \to c \to d \to e$ is the critical timing path, with a delay of 3.0 units.

Conventional STA always performs a robust analysis (never underestimating delay), but can be quite pessimistic in practice. There are two primary sources of pessimism: the use of worst-case delays and assuming worst-case switching behaviour.

SSTA [3] has been developed to address the pessimism introduced by worse-casing delay values, which becomes particularly problematic in the face of increasing device and interconnect variation. SSTA directly models the statistical variation of device and interconnect delays, calculating delay distributions rather than the fixed worst-case delays used by conventional STA. Directly modelling delay variations reduces pessimism since worst-case delay combinations (which are unlikely to occur) can be ignored.[3]

Both STA and SSTA assume worst-case switching behaviour: assuming all signals switch every clock cycle. This is not true in real operation. The most obvious cases are 'false paths', timing paths which are impossible to exercise in practice.

There has been some previous work investigating these issues. The problem of false paths is discussed in detail in [7], which presents algorithms for detecting near-critical false paths. In [8], toggle rate information (similar to vector-less power estimation) is used to adjust the results of SSTA to reduce pessimism. However only the average toggling behaviour (i.e. across many cycles) is considered and the toggling of individual timing paths can not be distinguished. The problem of re-convergent paths which cause correlations is also not addressed.

III. Extended Static Timing Analysis Formulation

To present the Extended Static Timing Analysis (ESTA) formulation we begin by defining some terminology.

Definition 3 (Path Sensitization Probability)
The probability of a timing path experiencing a transition during a single clock cycle.

By associating a delay with each path sensitization we can build path-delay distributions:

[3]In practice, the designer chooses an acceptable level of timing yield for their design, accepting the failure of some devices.

Definition 4 (Path-delay Distribution)
A set of paths, delays, and their associated sensitization probabilities.

Table I shows the path-delay distribution for the circuit in Fig. 1, assuming uniform input transition probability. We observe that the longest path is active only 12.5% of the time, much less than the other paths. Interestingly no timing paths are active 37.5% of the time since the output (e) remains constant.

A path-delay distribution is different from the delay distribution produced by SSTA. Under SSTA the delays are distributed according to a statistical delay model (i.e. due to device and interconnect delay variation), while in ESTA delays are distributed according to the probability of individual timing paths being activated.[4]

We can now define the ESTA problem which we focus on for the remainder of this paper:

Definition 5 (Extended Static Timing Analysis Problem)
Determine the path-delay distributions at all timing endpoints.

A. Using #SAT to Calculate Probabilities

To calculate a path-delay distribution we need to determine the delays and sensitization probabilities of different timing paths. The delay of a path can be calculated by traversing the timing graph and adding up the annotated delays, but calculating path sensitization probabilities is more involved.

We first define an activation function:

Definition 6 (Activation Function)
A Boolean function which evaluates to true whenever a path is sensitized by a transition (which could be a glitch or static value).

Given an activation function f with support size $|f|$ (i.e. the number of variables f depends on) we can calculate its sensitization probability as:

$$p = \#SAT(f)/2^{|f|} \tag{1}$$

where $\#SAT(f)$ represents the number of satisfying assignments to f, and $2^{|f|}$ represents the total number of possible assignments.

#SAT is a well established problem in theoretical computer science closely related to Boolean satisfiability (SAT) [9]. Where SAT seeks to find *a* satisfying assignment to a Boolean function, #SAT seeks the *number* of satisfying assignments. There are a number of algorithms for solving #SAT which are more efficient than naively enumerating the satisfying assignments with SAT [9].

Provided we can build appropriate activation functions for the paths under analysis we can use Eq. (1) to calculate their sensitization probabilities. A path with zero sensitization probability is by definition a false path.

B. Transition Model

To analyze a circuit we need to model the different signal transitions which can occur. While different models are possible, we model four types of transitions: R (rising), F (falling), H (high), and L (low), where H and L correspond to signals

[4]While we use a deterministic delay model for each timing edge in this work, there is no limitation preventing the use of a statistical delay model.

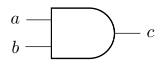

Fig. 2: AND gate

TABLE II: AND GATE TRANSITIONS.

a	b	c	a	b	c	a	b	c	a	b	c
R	R	R	F	R	F	H	R	R	L	R	L
R	F	F	F	F	F	H	F	F	L	F	L
R	H	R	F	H	F	H	H	H	L	H	L
R	L	L	F	L	L	H	L	L	L	L	L

which remain static and do not change. As an example, consider the AND gate in Fig. 2. For this simple circuit we can enumerate the possible output transitions, as shown in Table II.

Temporary glitches are modelled by their final transition. For example a low signal which temporarily glitches high before returning to low would be modelled as a F transition.

C. Combining Activation Functions

We can now define a timing tag, which intuitively corresponds to the delay of a transition along a particular path:

Definition 7 (Timing Tag)
A tuple $(\tau, \nu, f) \in \mathbb{Q} \times \{R, F, H, L\} \times (\mathbb{B}^{2I} \rightarrow \mathbb{B})$, corresponding to a path and transition combination. τ is the arrival time, ν the signal transition, and f the activation function.

For example, a tag $(15, R, x_1 \wedge \overline{x_2})$ corresponds to a rising transition with an arrival time of 15 units, which occurs only when the Boolean function $x_1 \wedge \overline{x_2}$ evaluates true. Since f is a general Boolean function encoding all the scenarios where the timing tag applies, false and re-convergent paths can be accounted for by constructing f appropriately.

Consider the AND gate from Fig. 2. If we have two timing tags t_a and t_b arriving at the gate inputs and a gate delay of δ_{AND} we can construct the corresponding output tag t_c as:

$$\begin{aligned} t_c = (&\delta_{AND} + max(t_a.\tau, t_b.\tau), \\ &AND(t_a.\nu, t_b.\nu), \\ &t_a.f \wedge t_b.f) \end{aligned} \tag{2}$$

where $\delta_{AND} + max(t_a.\tau, t_b.\tau)$ is the latest arrival time of a transition at the output, $AND(t_a.\nu, t_b.\nu)$ is the resulting transition (e.g. determined from Table II), and $t_a.f \wedge t_b.f$ is the logical conjunction (AND) of the input activation functions.

More generally for a K-input gate with delay δ_{gate} implementing the logic function $h(x_1, x_2, \ldots, x_K)$ and incoming tags $t_1, t_2, \ldots, t_K$ the output tag t_{gate} can be defined as:

$$\begin{aligned} t_{gate} = (&\delta_{gate} + max(t_1.\tau, t_2.\tau, \ldots, t_K.\tau), \\ &H(t_1.\nu, t_2.\nu, \ldots, t_K.\nu), \\ &t_1.f \wedge t_2.f \wedge \ldots \wedge t_K.f) \end{aligned} \tag{3}$$

where $H(\nu_1, \nu_2, \ldots, \nu_K)$ is the transition function derived from the logic function $h()$[5]. Eq. (3) produces an STA-like delay estimate which is a safely pessimistic upper-bound, ensuring no paths will be underestimated. The activation function is specified as the conjunction of all the incoming tag activation functions, since all the tags must arrive to generate the corresponding arrival time and output transition.

D. Condition Functions

Eq. (3) describes how to propagate timing tags through a gate (or wire[6]), allowing us to construct timing tags – including their associated activation functions – by walking through the timing graph. However we still require some base activation functions at timing sources, and a method to specify their transition probabilities.

We can accomplish this by defining a set of Boolean *conditioning functions* at each timing source. For the simplest case of uniform probability we can define the following conditioning functions:

$$\begin{aligned} f_R(x, x') &= \overline{x} \wedge x' \quad & f_F(x, x') &= x \wedge \overline{x'} \\ f_H(x, x') &= x \wedge x' \quad & f_L(x, x') &= \overline{x} \wedge \overline{x'} \end{aligned} \tag{4}$$

where intuitively x and x' represent the current and next state of the source. Each function in Eq. (4) corresponds to a particular transition occurring on the source.

Note each condition function in Eq. (4) is satisfied 25% of the time (e.g. $\frac{\#SAT(f_R)}{2^{|f_R|}} = \frac{1}{4}$), assuming uniformly random x and x'. This yields a uniform probability for each transition. In general arbitrary conditioning functions can be used, allowing for non-uniform probabilities and correlations between sources.

By combining Eq. (3) with condition functions such as those in Eq. (4) we can propagate timing tags from timing sources to timing endpoints. The probabilities of the tags at all timing endpoints can then be calculated using Eq. (1) to construct the path-delay distributions – completing the ESTA analysis.

IV. ESTA IMPLEMENTATION

We have developed a tool to perform ESTA. The tool is written in C++ and uses Binary Decision Diagrams (BDDs) [10] (via the CUDD library [11]) to represent the netlist logic and timing tag activation functions. BDDs allow easy manipulation of Boolean functions and enable #SAT to be solved efficiently once the BDD is constructed [9].

A. Calculating Timing Tags

The basic procedure to calculate a node's output timing tags is shown in Algorithm 1. Provided with the set of tags arriving at each of the K inputs, we enumerate the Cartesian product of the input tag sets (line 3) to consider all possible cases of input transitions and arrival times. Lines 4-6 evaluate a specific set of $CaseTags$ (one tag per input) according to Eq. (3). For each case the resulting tag is recorded (line 7), and the full set of output tags returned (line 8) for use by downstream nodes.

Algorithm 1: ESTA Node Traversal

```
Require: $In^{(1)}_{tags}, ..., In^{(K)}_{tags}$ sets of tags on each input, $\delta$ input
    to output delay, $h$ node logic function
1: function TRAVERSENODE($In^{(1)}_{tags}, ..., In^{(K)}_{tags}, \delta, h$)
2:     $Out_{tags} \leftarrow \varnothing$
3:     for each $CaseTags \in In^{(1)}_{tags} \times ... \times In^{(K)}_{tags}$ do
4:         $\tau \leftarrow \delta + \text{MAX}(CaseTags[0].\tau, ..., CaseTags[K].\tau)$
5:         $\nu \leftarrow H(CaseTags[0].\nu, ..., CaseTages[K].\nu)$
6:         $f \leftarrow CaseTags[0].f \wedge ... \wedge CaseTags[K].f$
7:         $Out_{tags}$.APPEND($(\tau, \nu, f)$)
8:     return $Out_{tags}$
```

[5] $H()$ can be determined by evaluating $h()$ twice; first at the initial and then at the final values of the input transitions (e.g. 0 then 1 for a R transition).

[6] Wires can be treated as single-input 'gates' implementing logical identity.

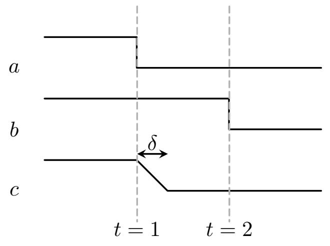

Fig. 3: Input filtering example. The arrival time at c is $1 + \delta$, but STA or a naive ESTA implementation will report $2 + \delta$.

After all nodes in the timing graph have been processed the tags at all endpoints can be evaluated with #SAT to build the path-delay distributions.

B. Input Filtering

Consider the timing diagram in Fig. 3 for the AND gate from Fig. 2. Initially, both inputs (a, b) are high, producing a high output (c). At $t = 1$ input a falls, producing a falling transition on the output c with some delay. The later transition on input b at $t = 2$ produces no change in the output c, since input a remained low controlling the output. In this case input a can be said to 'filter' transitions on input b.

While Algorithm 1 handles these cases correctly, it does so in an unnecessarily pessimistic manner; always using the latest arrival time, even if the associated transition would be filtered and have no effect. To counteract this, as each input arrives we restrict the node logic function to the input's post-transition value. We can then use Boolean difference to identify subsequently arriving input transitions which do not affect the output. Such input tags are ignored during arrival time calculation, removing the unnecessary pessimism.

C. Tag Merging

In the worst case Algorithm 1 can produce $O(\ell^K)$ output tags where ℓ is the maximum number of tags across all K inputs. While the number of tags produced is often smaller in practice it can still grow large – particularly since the output tags of a node become the inputs to subsequent nodes.

To counteract this rapid growth, we can merge tags together. Suppose we have the tags $t_1, t_2, \ldots, t_n$ with the same transition ν. We can produce a new tag t_{merged} which approximates the original tags:

$$\begin{aligned} t_{merged} = (&max(t_1.\tau, t_2.\tau, \ldots, t_n.\tau), \\ &\nu, \\ &t_1.f \vee t_2.f \vee \ldots \vee t_n.f) \end{aligned} \tag{5}$$

We ensure a safely pessimistic approximation by using the maximum arrival time, and only merging tags with the same transition. The activation function of the merged tag is the logical disjunction (OR) of the original tags, since any of the tags could produce this bounding transition.

Note the approximation is exact if the original tags have the same delay. Our implementation always merges such tags.

D. Run-Time/Accuracy Trade-Offs

Although precise tag merging helps, for larger circuits the number of tags (and hence run-time) can grow prohibitively large. Accordingly we have also developed several different techniques to trade-off accuracy for reduced run-time. They all rely on Eq. (5) to safely merge tags.

Fixed Delay Binning: Merges output tags within a delay bin of size d. For instance at $d = 100$ tags with delays 75, 80, 110, 120, 155 would be reduced to 80 and 155.

Adaptive Binning: Merges input tags to limit the size of the Cartesian product evaluated at a node to at most m, by iteratively re-binning the input tags at larger bin-sizes.

Percentile Binning: Merges output tags which can not generate s-percentile critical paths. For example, $s = 0.05$ would ensure no merging occurred on the top 5% of critical paths, while all other tags would be merged.

In practice these various techniques can be used in combination.

E. Computational Complexity

The computational complexity of our ESTA implementation is dominated by two components: evaluating tags, and constructing BDDs. The complexity of evaluating a Cartesian product of tag sets (excluding BDD construction) is $O(L^K)$, where L is the maximum number of tags on any node input. This must be done across all n nodes in the timing graph, taking $O(nL^K)$ time. Constructing BDDs in the worst case takes $O(2^{n_{vars}})$ time, where n_{vars} is the number of Boolean variables. For the conditioning functions in Eq. (4), $n_{var} = 2I$. The resulting complexity of ESTA is $O(nL^K + 4^I)$.

For typical circuits K is bounded by a small constant and L can be controlled with the methods in Section IV-D. As a result run-time is typically dominated by BDD construction. However in practice decreasing L also reduces BDD construction time, since fewer BDDs covering more of the input space (which are typically simpler to encode) are required.

While ESTA's worst-case complexity is similar to exhaustive simulation, $O(n4^I)$, ESTA's typical complexity is far lower in practice. In particular, constructing BDDs to evaluate #SAT is far more efficient than enumerating a vast input space.[7]

V. Evaluation Methodology

To evaluate our ESTA implementation we compare it against post-place-and-route timing simulation performed with Mentor Graphics Modelsim SE 10.4c. The evaluation flow used is shown in Fig. 4. We take in a circuit netlist which is mapped onto a 40nm 6-input Look-Up-Table (6-LUT) based FPGA using VPR [12] to generate an SDF file with both logic and routing delays. The SDF file is then used to annotate identical delays in both Modelsim and ESTA. To avoid unrealistic glitch filtering Modelsim was run using the transport delay model.

A. Monte-Carlo Simulation

While exhaustive simulation is useful for verification it quickly becomes impractical, since the number of cases to be simulated grows as $O(4^I)$. To enable the evaluation of larger benchmarks, and provide a more realistic run-time comparison to ESTA, we also developed a Monte-Carlo (MC) based simulation framework for calculating path-delay distributions.

It is important to distinguish between the strength of guarantees that MC and ESTA provide. ESTA guarantees it will

[7]For example, exhaustive simulation of `clma`, requires evaluating $> 10^{249}$ sets of input transitions; *counting* the satisfying assignments is far faster.

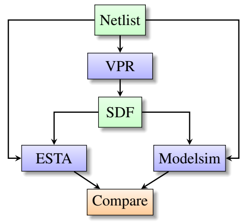

Fig. 4: Evaluation Flow.

TABLE III: NORMALIZED CRITICAL PATH DELAYS WITH FALSE PATHS.

	STA	MC	ESTA	
s298	1.000	0.981	0.981	
clma	1.000	0.959	0.984	†
frisc	1.000	0.965	0.999	*
elliptic	1.000	0.974		

* ESTA upper-bound, †MC optimistic

always produce safely pessimistic upper bounds of path-delay distributions. MC can not provide any such guarantees.

In the MC framework we uniformly generate random sets of input transitions to sample the input space. This sampling procedure was run for 48 hours on each benchmark. All quality comparisons are based on the more accurate 48-hour sample, while run-times are determined by finding the smallest sample size which meets some convergence criteria.

Since it is impractical to exhaustively simulate large circuits we determine convergence based on sample statistics. We define convergence based on the max-delay probability, as this is the delay region of interest when considering timing errors. Intuitively we define convergence as when we are confident the sample *probability* of the maximum-delay varies by less than 5% across multiple samples. More formally:

Definition 8 (MC Max-Delay Convergence) *Let $\hat{p}$ be the sample probability of activating maximum delay paths. Given a sample with max-delay probability confidence interval $[LB, UB]$ at 0.99 confidence, we say the sample has converged if* $\frac{UB-LB}{\hat{p}} < 0.05$.

The Monte-Carlo run-times reported include only the simulation run-time, and exclude the large amount of post-processing required to extract useful transition and delay information.

B. Benchmarks

We evaluate ESTA on the 20 largest MCNC benchmarks [13] which are listed in Table IV. Any state elements (e.g. flip-flops) were replaced with primary inputs and outputs.

C. Metrics

To compare the Quality of Results (QoR) between STA, ESTA and MC we used the Earth Mover's Distance (EMD) metric [14], commonly used to compare image histograms. EMD corresponds to the minimal amount of 'work' required to transform one discrete distribution into another.

To account for different critical path delays across benchmarks we normalize EMD by the EMD between MC and STA. The resulting normalized EMD $\in [0, 1]$ describes how closely the MC distribution is approximated. A value of 1 corresponds to an STA-like analysis (worst-case switching behaviour), and a value of 0 corresponds to the MC distribution (true switching behaviour). For each benchmark we report the run-time for determining the maximum delay across all outputs and the normalized EMD of the resulting path-delay distribution.

VI. RESULTS

Using the experimental methodology from Section V and the implementation from Section IV we perform several experiments to investigate the characteristics of ESTA. Unless otherwise noted ESTA was run with $d = 100$ps. Results were collected on a Xeon E5-2643v3 machine with 256GB of memory.

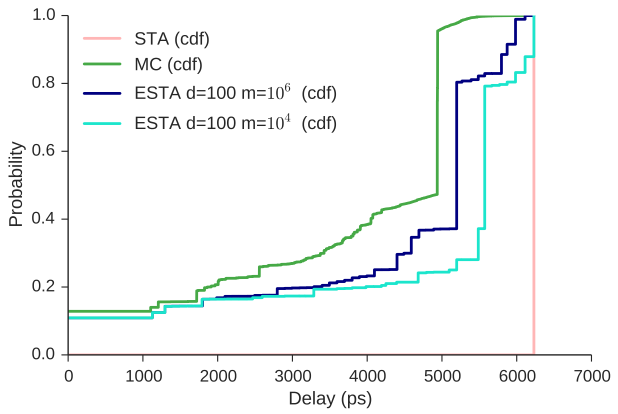

Fig. 5: Maximum delay CDF plots on the `spla` benchmark.

A. Verification

To verify the correctness of our ESTA tool we exhaustively compared with Modelsim on a set of small micro-benchmarks including simple logic circuits, ripple-carry adders and array multipliers. We verified for all possible input transitions, that ESTA agreed with simulation's output transitions and produced an upper-bound of the simulation delay.

B. Maximum Delay Estimation

By running both MC, STA, and ESTA (with percentile binning) we can investigate the impact of false-paths. While most circuits in the MCNC20 benchmarks produced the same critical path delay in all tools, the existence of false paths caused divergence on the benchmarks in Table III. ESTA was able to identify the true critical path delay on the `s298` and `clma` benchmarks, and confirm false paths exist on `frisc`.[8]

Notably on `clma` MC reported an unsafe (optimistic) delay – indicating the worst-case path was never sampled. This illustrates the utility of ESTA's strong guarantees; it never underestimates delay.

C. ESTA and MC Comparison

Fig. 5 plots the maximum path-delay Cumulative Distribution Functions (CDFs) for `spla`. STA, which assumes worst-case switching behaviour, produces a single maximum delay estimate of 6230ps (the critical path delay) for all cases ($p = 1$). In contrast MC, which directly simulates switching behaviour, produces a path-delay distribution, showing 13% of input transitions cause no delay (i.e. don't affect the output), and only 4% of input transitions produce delays > 5000ps. ESTA is always safely conservative compared to MC and less pessimistic than STA, with its CDF always falling between MC and STA. The form of ESTA's CDF follows the shape of MC's CDF, with larger m producing a more accurate result.

Table IV quantitatively compares ESTA and MC accuracy. First considering the MC max-delay probability ($\hat{p}$), we observe that benchmarks with relatively large max-delay probability tend to converge, while those with smaller probability ($\hat{p} \leq$

[8]ESTA exceeded memory limits on `frisc` due to the large number of nearly critical paths, and exceeded 48 hours run-time on `elliptic`.

TABLE IV: QUALITY AND RUN-TIME ON THE MCNC20 BENCHMARKS.

			Prob.	QoR (norm. EMD)		Run-time (minutes)		
	I	LUTs	MC $\hat{p}$	ESTA m=10^4	ESTA m=10^6	MC	ESTA m=10^4	ESTA m=10^6
ex5p	8	740	$3.9 \cdot 10^{-3}$	0.73	0.37	151.7	0.5	4.6
apex4	9	970	$5.9 \cdot 10^{-3}$	0.85	0.74	118.1	1.5	8.7
ex1010	10	3093	$2.7 \cdot 10^{-3}$	0.96	0.85	922.1	5.8	40.4
s298	11	1301		0.87	0.79		5.7	59.4
misex3	14	1158	$1.5 \cdot 10^{-3}$	0.66	0.46	781.3	5.2	54.9
alu4	14	1173	$2.5 \cdot 10^{-3}$	0.83	0.46	473.6	2.4	58.9
spla	16	3005		0.40	0.25		6.3	207.1
pdc	16	3627		0.56	0.31		11.3	118.4
apex2	39	1478	$6.9 \cdot 10^{-4}$	0.33		2068.2	87.1	
seq	41	1325		0.34			57.9	
des	256	554	$1.8 \cdot 10^{-2}$			60.7		
clma	415	6239		0.92			319.2	
tseng	436	798						
diffeq	440	871						
dsip	460	880	$3.5 \cdot 10^{-2}$			49.8		
bigkey	494	883	$1.2 \cdot 10^{-2}$			150.6		
frisc	905	3028						
elliptic	1252	2135						
s38584.1	1332	4486	$3.2 \cdot 10^{-3}$			2829.4		
s38417	1545	3465						

Blank entries exceeded 48 hours run-time.

10^{-4}) tend not to. Intuitively, it is difficult to determine if a small $\hat{p}$ is caused by an inherently rare path, or by insufficient sample size. This causes MC to converge slowly.

Now considering ESTA m=10^4 QoR, we see ESTA's analysis falls between STA and MC, with normalized EMD ranging between 0.96 (nearly STA-like) and 0.33 (more MC-like), with an average of 0.68 across the benchmarks which completed. Increasing m to 10^6 reduces the average normalized EMD by 28% to 0.53 on the common benchmarks which completed.

The QoR gap between ESTA and MC is derived from three factors. First, binning (Section IV-D) introduces additional pessimism since the true distribution is approximated with fewer timing tags. Second, Modelsim performs more aggressive input filtering than ESTA (Section IV-B), since it optimistically assumes signals are stable before they transition.[9] Third, Modelsim optimistically treats simultaneous transitions at a gate input with no logical effect (e.g. simultaneous R/F in Table II) as producing no output transition. To remain safely pessimistic ESTA models these with an appropriate R/F transition.

The run-time performance of ESTA and MC are also shown in Table IV. Looking at MC, it converges on only 10 of the 20 benchmarks within 48 hours, and fails to converge even on benchmarks with few inputs (e.g. `s298`). ESTA m=10^4 completed 11 of 20 benchmarks and shows more stable run-time, completing all of the benchmarks with 41 or fewer inputs and also the largest benchmark (in terms of logic) `clma`. For those benchmarks which completed under both MC and ESTA, ESTA m=10^4 and m=10^6 completed 44.0$\times$ and 14.6$\times$ faster than MC respectively. For all benchmarks with > 415 inputs ESTA run-time exceeded 48 hours during BDD construction. These results show that ESTA can be used to quickly calculate bounding path-delay distributions on circuits with a moderate number of inputs. For example, enabling automated analysis of the (12 input) overclocking designs considered in [5].

VII. CONCLUSION & FUTURE WORK

In conclusion we have presented ESTA, a new timing analysis method which accounts for non-worst-case switching behaviour, calculating safe bounding path-delay distributions over all input combinations. We showed how the sensitization probability of timing paths can be calculated using #SAT, allowing path-delay distributions to be constructed. We presented a BDD-based implementation, including approaches to improve accuracy and scalability. Finally we performed an experimental comparison of ESTA and Monte-Carlo based timing simulation, showing ESTA can on average run 14.6 to 44.0$\times$ faster while achieving results within 53 to 68% of Monte-Carlo, a 46 to 32% reduction in pessimism compared to STA.

There are a variety of directions for future work. The key algorithmic challenge for ESTA is scalability. The main run-time bottleneck of our ESTA implementation is solving #SAT. One avenue for investigation is the use of CNF-based #SAT solvers instead of BDDs. Another approach is to give up some of ESTA's guarantees for improved scalability by approximating the solution to #SAT [9]. Improved run-time quality trade-offs (Section IV-D), particularly those that actively consider the impact on quality would also improve results. For instance percentile binning, which analyzes only the most critical paths, may prove a more scalable approach.

There are also open questions driven by the application of ESTA. While it is possible to model arbitrary correlations and probabilities in ESTA it is not clear how best to construct condition functions to model the switching behaviour of state elements like flip-flops. It would also be useful to combine both ESTA and SSTA (i.e. a statistical delay model) to determine the path-delay distribution while considering device-level delay variation. Finally automated ESTA enables larger scale investigation of how different circuit implementations impact the path-delay distribution, which has interesting applications to approximate computing techniques such as overclocking.

[9] ESTA can not do the same as it calculates upper bounds on arrival times. Performing combined min/max delay analysis would improve ESTA's filtering.

REFERENCES

[1] S. Sapatnekar, "Static timing analysis," in *EDA for IC implementation, circuit design, and process technology*. CRC press, 2006, ch. 6.

[2] S. R. Nassif, "Modeling and forecasting of manufacturing variations," in *Int. Workshop on Statistical Metrology*, 2000, pp. 2–10.

[3] D. Blaauw *et al.*, "Statistical timing analysis: From basic principles to state of the art," *IEEE TCAD*, vol. 27, no. 4, pp. 589–607, 2008.

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *European Test Symp.*, 2013.

[5] K. Shi *et al.*, "Accuracy-performance tradeoffs on an fpga through overclocking," in *IEEE FCCM*, 2013, pp. 29–36.

[6] ——, "Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs," in *DAC*, 2014, pp. 190:1–190:6.

[7] D. H. C. Du *et al.*, "On the general false path problem in timing analysis," in *DAC*, June 1989, pp. 555–560.

[8] B. Liu, "Signal probability based statistical timing analysis," in *DATE*, March 2008, pp. 562–567.

[9] C. P. Gomes *et al.*, "Model counting," in *Handbook of satisfiability*. IOS press, 2009, ch. 20.

[10] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, 1992.

[11] F. Somenzi, "CUDD: CU Decision Diagram package release 2.5.1," University of Colorado at Boulder, 2015.

[12] J. Luu *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM TRETS*, vol. 7, no. 2, pp. 1–30, 2014.

[13] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide 3.0," MCNC, Tech. Rep., 1991.

[14] Y. Rubner *et al.*, "The earth mover's distance as a metric for image retrieval," *Int. J. of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.