Calculated Risks: Quantifying Timing Error Probability with Extended Static Timing Analysis

Kevin E. Murray, Andrea Suardi, Vaughn Betz, George Constantinides

***Abstract*—Timing analysis is a key step in the digital design process. By modelling device delay variations Statistical Static Timing Analysis reduces pessimism compared to traditional Static Timing Analysis. However it ignores the circuit's logic which causes 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 how this analysis is related to the well-known #SAT problem and present approaches to improve scalability, achieving, on average, results 75 to 37% less pessimistic than Static Timing Analysis while running 569 to 16 times faster than Monte-Carlo timing simulation.**

***Index Terms*—Extended Static Timing Analysis (ESTA), #SAT, Binary Decision Diagrams (BDDs), Better than worst case design, Overclocking, Timing Verification**

I. Introduction

TIMING analysis, the process of determining whether a circuit will operate correctly at speed, is a key step in the digital design process. The conventional approach for performing timing analysis is Static Timing Analysis (STA) [1]. However, with smaller manufacturing process technologies the worst case and average case delay can deviate significantly, causing STA to be very pessimistic [2], [3]. Statistical Static Timing Analysis (SSTA) was introduced to reduce this pessimism, by modeling device and interconnect delay variations. This allows designers to sacrifice a small amount of delay coverage (and resulting yield) for considerable performance improvement [3]. However, both STA and SSTA assume the worst case switching behaviour across all input combinations.

We propose a complementary approach which quantifies the stochastic variation across *inputs* rather than across delays, 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 (*e.g.* by a higher level algorithm), or where the input data is inherently noisy (*e.g.* sensor readings) and extreme accuracy is unnecessary [4]. By running systems 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[1] for improved performance, and inspired changes to the architecture of arithmetic components [6].

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 exhaustive coverage of 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.

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]

To simplify matters we initially consider the case where the set of inputs at cycles i and $i+1$ are independent, identically uniformly distributed and statistically stationary. We later describe how these restrictions can be lifted.

An earlier version of this work was published in [7]. We have extended it by: enhancing our formulation to handle non-uniform and correlated inputs, improving scalability with a more efficient analysis mode, exploring the impact of Monte-Carlo convergence criteria and performing a more extensive evaluation and comparison of our approach with Monte-Carlo simulation. Our key contributions include:

- a new timing analysis formulation to determine bounding delay distributions across input combinations,
- methods to model non-uniform and correlated inputs,
- reduction of the 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. Section V describes how non-uniform input probabilities and correlations can be modelled. Sections VI and VII describe the experimental methodology and results. Section VIII concludes and outlines future work.

K. Murray and V. Betz are with the Department of Electrical and Computer Engineering, University of Toronto, Canada: {kmurray,vaughn}@eecg.utoronto.ca

A. Suardi and G. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, United Kingdom: {a.suardi,g.constantinides}@imperial.ac.uk

Manuscript received November 7, 2017; revised January 24, 2018; accepted March 3, 2018.

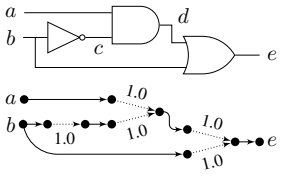

Fig. 1: Example circuit and timing graph with annotated delays.

TABLE I: PATH-DELAY DISTRIBUTION (UNIFORM INPUT TRANSITION PROBABILITY).

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

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:

Definition 1 (Timing Graph)
A directed acyclic graph[3] *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 (worst-case) *arrival time* of signals at each node in the graph (that is, the latest time the signal can become stable). 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.[4]

[1]In practice non-error tolerant signals (*e.g.* control signals) must have sufficient slack to operate reliably at the overclocked frequency.

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

[3]If the timing graph contains cycles it can be transformed into an acyclic graph using standard techniques [8].

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

Both STA and SSTA assume worst-case switching behaviour, by assuming all signals switch every clock cycle. In real operation this assumption does not hold. The most obvious cases are *false paths*, timing paths which are impossible to exercise in practice. An example is shown in Fig. 2

There has been a variety of previous work investigating these issues. In [9] a framework for comparing false path identification criteria is presented. [10] presents algorithms for detecting near-critical false paths. Algebraic Decision Diagrams (ADDs) are used in [11] to identify and eliminate false paths. [12] describes how Automatic Test Pattern Generation techniques can be applied to a transformed version of the circuit to identify false paths. In [13] a circuit's true delay bounds are determined by checking whether critical paths are sensitizable using CNF-based SAT solvers. However, unlike this work, these works consider only false path identification and elimination. They do not consider the probability of (true) paths being activated.

In [14], 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 cannot be distinguished. The problem of re-convergent paths which cause correlations is also not addressed.

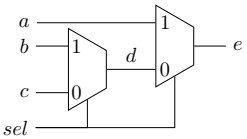

Fig. 2: The path $b \to d \to e$ is a false-path; the common sel means transitions on b can never propagate to e after the other paths have stabilized.

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:

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, where rise, fall, static high, and static low each have 25% probability. We observe that the longest path is active only 12.5% of the time, much less likely than other paths. This occurs when b undergoes a falling transition ($p = 0.25$), and a simultaneously undergoes either a static high or rising transition ($p = 0.5$). 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.[5]

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:

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

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) [15]. 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 [15].

Provided we can build appropriate activation functions for the paths under analysis we can use Eq. (1) to calculate their sensitization probabilities. This generalizes previous work, in that 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 use a transition mode [16] model with four types of transitions: R (rising), F (falling), H (high), and L (low), where H and L correspond to signals which remain static. As an example, consider the AND gate in Fig. 3. For this simple circuit we can enumerate the possible output transitions, as shown in Table II.

Temporary glitches are modelled by the logical transition which occurs and an appropriate delay. For example, Fig. 4 shows a high signal which temporarily glitches before settling to low, which is modelled as a F transition with arrival time $1 + \delta$.

[5]While we use a deterministic delay model for each timing graph edge in this work, it does not preclude the use of a statistical delay model.

Fig. 3: 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

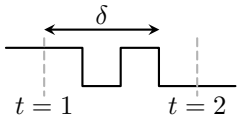

Fig. 4: ESTA models the glitch between $t = 1$ and $t = 2$ as a F transition with arrival time $1 + \delta$.

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 $(\tau, \nu, f) \in \mathbb{R} \times \{R, F, H, L\} \times (\mathbb{B}^Q \to \mathbb{B})$, corresponding to a path and transition combination. τ is the arrival time, ν the signal transition, f the activation function, and Q is the support size of f.*

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 appropriately constructing f.

Consider the AND gate from Fig. 3. 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 $g(x_1, x_2, \ldots, x_K)$ and incoming tags $t_1, t_2, \ldots, t_K$ the output tag t_{gate} can be defined as:

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

where $G(\nu_1, \nu_2, \ldots, \nu_K)$ is the transition function derived from the logic function $g()$[6]. 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 logical conjunction of all the incoming tag activation functions, since all the tags must arrive to generate the corresponding output transition and arrival time.

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

D. Conditioning Functions

Eq. (3) describes how to propagate timing tags through a gate (or wire[7]), 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. Intuitively these functions model the statistical behaviour of the circuit's primary inputs.

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 conditioning 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. This is discussed in more detail in Section V.

By combining Eq. (3) with conditioning 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.[8] The tool is written in C++ and uses Binary Decision Diagrams (BDDs) [17] (via the CUDD library [18]) 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 BDDs are constructed [15].

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) there by ensuring all possible cases of input transitions and arrival times are considered. Lines 4 to 6 evaluate a specific set of $InputTags$ (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.

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.[9]

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

[8]The source code is available from www.github.com/kmurray/esta.

[9]While our implementation always performs a full timing analysis, it could be extended to perform an incremental timing analysis using the same techniques as conventional STA [19], [20].

Algorithm 1 Basic ESTA Node Traversal

Require: $In_{tags}^{(1)}, \ldots, In_{tags}^{(K)}$ sets of tags on each input, δ input to output delay, G node logic function

1: **function** TRAVERSENODE($In_{tags}^{(1)}, \ldots, In_{tags}^{(K)}, \delta, G$)
2: $OutTags \leftarrow \varnothing$
3: **for each** $InputTags \in In_{tags}^{(1)} \times \ldots \times In_{tags}^{(K)}$ **do**
4: $\tau \leftarrow \delta + \text{MAX}(InputTags[1].\tau, \ldots, InputTags[K].\tau)$
5: $\nu \leftarrow G(InputTags[1].\nu, \ldots, InputTages[K].\nu)$
6: $f \leftarrow InputTags[1].f \wedge \ldots \wedge InputTags[K].f$
7: $OutTags.\text{APPEND}((\tau, \nu, f))$
8: **return** $OutTags$

B. Input Filtering

Consider the timing diagram in Fig. 5 for the AND gate from Fig. 3. 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.

It should be noted that this approach maintains the monotone speed-up property[10] necessary for a robust analysis [21]. While input filtering is performed in a delay dependent manner it is safe since:

- Decreasing the delay of an input transition can only *increase* the amount of filtering performed (since the signal would be stable earlier).
- Increased filtering can only *decrease* the delay of a resulting output transition (if an additional late arriving input transition was filtered).
- Input filtering never prevents a transition from propagating through the circuit.

Together these ensure that the delay of any timing path can never increase if a component delay decreases, and that all timing paths are considered during the analysis – none are 'dropped' in a delay-dependent manner.

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.

[10]Monotone speed-up ensures that decreasing the delay of circuit element(s) cannot increase the critical path delay.

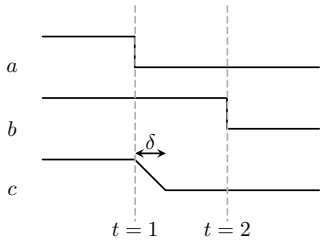

Fig. 5: Input filtering example. The arrival time at c is $1+\delta$, but STA or a naive ESTA implementation will report $2+\delta$.

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:

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

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 arrival time. 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 with the same transition, and delay within the same delay bin. For a delay bin size of d, the delay bins are defined as $[k \cdot d, (k+1) \cdot d)$, $k \in \mathbb{N}$.

As an example, consider a set of five tags with the same transition and delays 75, 80, 110, 120, 155. Performing Fixed Delay Binning with $d = 100$ would produce two tags with delays 80 and 155.

Adaptive Binning: Merges input tags to limit the size of the Cartesian product evaluated at a node to at most m. This is performed by iteratively re-binning the node's input tags at increasing bin-sizes, until the size of the cartesian product drops below m.

Percentile Binning: Merges output tags which cannot generate s^{th}-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. Enhanced Algorithm

Algorithm 2 shows the procedure for traversing nodes including the enhancements from Sections IV-B to IV-D. Lines 3 to 5 calculate the cartesian product of tags, limiting the number of cases evaluated to at most m. To perform input filtering we first sort the K input tags of a single case by ascending order of arrival time (Line 10).[11] We then check whether the transitions associated with each input can affect the output (Line 12). If a transition does affect the output (is not filtered) we 'restrict' the node's logic function (Line 13), so the current input's stable post-transition value will be considered when filtering later arriving inputs. Finally, we bin the resulting tags into bins of size d (Line 17).

Algorithm 2 Enhanced ESTA Node Traversal

Require: $In^{(1)}_{tags}, ..., In^{(K)}_{tags}$ sets of tags on each input, δ input to output delay, G node logic function, m maximum cartesian product size, d delay bin size

1: **function** TRAVERSENODE($In^{(1)}_{tags}, \ldots, In^{(K)}_{tags}, \delta, G, m, d$)
2: $Out_{tags} \leftarrow \varnothing$
3: $AllCases = In^{(1)}_{tags} \times ... \times In^{(K)}_{tags}$ ▷ Cartesian product
4: **if** $|AllCases| > m$ **then** ▷ Limit product size to m
5: $AllCases \leftarrow$ REDUCEPRODUCT($AllCases, m$)
6: **for** $InputTags \in AllCases$ **do**
7: $\tau \leftarrow \varnothing$
8: $\nu \leftarrow G(InputTags[1].\nu, ..., InputTags[K].\nu)$
9: $f \leftarrow$ TRUE
10: SORTASCENDINGARRIVAL($InputTags$)
11: **for** $InputTag \in InputTags$ **do** ▷ Each node input
12: **if not** FILTERED($G, InputTag$) **then**
13: $G \leftarrow$ RESTRICT($G, InputTag$) ▷ Constrain input
14: $\tau \leftarrow$ MAX($\tau, \delta + InputTag.\tau$) ▷ Update delay
15: $f \leftarrow f \wedge InputTag.f$ ▷ Update activation f'n
16: Out_{tags}.APPEND((τ, ν, f))
17: $Out_{tags} \leftarrow$ REDUCETAGS(Out_{tags}, d) ▷ Delay binning
18: **return** Out_{tags}

In our implementation activation function construction (Line 15) is performed lazily. As a result the cost of constructing activation function BDDs is incurred only during the final #SAT evaluations, and not during the graph traversal.

F. Computational Complexity

The computational complexity of our ESTA implementation is dominated by two components: evaluating tags, and performing #SAT.

The complexity of manipulating BDD's is $O(2^{n_{vars}})$, where n_{vars} is the total number of Boolean variables in the BDD. The size of a node's Cartesian tag product is $O(L^K)$, where K is the maximum number of inputs to any node, and L is the maximum number of tags on any node input. Sorting the input tags for filtering takes $O(K \log K)$ time, and manipulating the node's logic function (as a BDD) takes $O(2^K)$ time. The complexity of evaluating a single node is then $O(L^K(2^K + K \log K))$. This must be done across all n nodes in the timing graph, taking $O(nL^K(2^K + K \log K))$ time.

To solve #SAT we must construct BDDs which in the worst case takes $O(2^Q)$ time, where Q is the total number of Boolean

[11] This enforces causality by ensuring that only stable inputs can filter subsequently arriving transitions.

conditioning function variables. The resulting complexity of ESTA is

$$O(nL^K(2^K + K\log K) + 2^Q)\,. \tag{6}$$

In typical circuits K is bounded by a small constant (*e.g.* 6), and the complexity simplifies to

$$O(nL^K + 2^Q)\,. \tag{7}$$

If we define q as the number of conditioning function variables per timing source then $Q = qI$. For the conditioning functions in Eq. (4) $q = 2$ and the complexity becomes

$$O(nL^K + 4^I)\,. \tag{8}$$

L can be controlled using the methods in Section IV-D, and as a result run-time is typically dominated by BDD construction for #SAT. However in practice decreasing L also reduces #SAT run-time, since fewer BDDs covering more of the input space (which are typically simpler to encode) are required.

While the time complexity of ESTA is high, this is fundamental to the problem of analyzing the logic behaviour of a circuit under all possible inputs. For instance, all known general false-path detection algorithms also have exponential time complexity, since false-path identification is reducible to SAT: an NP-complete problem. The ESTA problem (calculating the probability that timing paths are activated) is more general than the false-path identification problem (finding timing paths with zero activation probability), and therefore subject to the same theoretical limits.

It should also be noted that the $O(4^I)$ term represents the worst-case complexity of BDD construction, however it can be more efficient in practice – such as when using techniques like dynamic variable re-ordering [22].

G. Comparison with Exhaustive Simulation

It is informative to compare the complexity of ESTA and Exhaustive Simulation, which takes

$$O(n4^I) \tag{9}$$

time.

An important distinction between Eqs. (8) and (9) is the coefficient of the $O(4^I)$ term. In exhaustive simulation the coefficient is n (circuit size), while in ESTA the coefficient is a constant. This results in a significant difference: exhaustive simulation's complexity grows as the *product* of circuit size and the number of possible input combinations, while ESTA's complexity grows linearly with circuit size and $O(4^I)$.

Intuitively, ESTA achieves this lower complexity since it traverses the timing graph *only once* during its analysis. In contrast, exhaustive simulation must evaluate the whole circuit for each of the 4^I possible sets of input transitions. In addition, constructing BDDs to evaluate #SAT is far more efficient in practice than enumerating a vast input space.[12]

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

H. Per-Output & Max-Delay Modes

Our ESTA tool can be run in two possible operating modes:

Per-Output Mode: Calculates a unique path-delay distribution for each timing endpoint. This is analogous to the maximum arrival time at each timing endpoint reported by STA.

Max-Delay Mode: Calculates a single path-delay distribution corresponding to the maximum delay across all timing endpoints. This is analogous to the Critical Path Delay (CPD) reported by STA. However the maximum path-delay distribution calculated may involve multiple timing paths, since different paths will be maximal for different sets of input transitions. This requires the ESTA tool to track which timing path (tag) is responsible for the maximum delay of each set of input transitions. To reduce this overhead, binning is applied when calculating the maximum delay distribution to reduce the number of unique tags.

It is important to note that the max-delay distribution cannot be calculated from the path-delay distributions produced in per-output mode, since the distributions contain only aggregate information and do not capture which path is maximal for a given set of primary input transitions.

V. Non-Uniform & Correlated Conditioning Functions

In traditional STA a design is typically analyzed with multiple delay models to capture the impact of delay variation at different 'timing corners', and in multiple operational 'modes' to model the behaviour of the design in different use cases [23]. Like traditional STA, ESTA can be performed using different combinations of delay models and operating modes. However, ESTA generalizes STA's concept of a mode to include the transition probability distribution of the primary inputs.

So far we have considered uniform and independent inputs, which are modelled with the conditioning functions in Eq. (4). In general, inputs may have non-uniform probabilities and may not be independent. In practice, the probabilities and correlations could be determined from designer knowledge, simulation, the upstream circuit logic, or derived from word-level statistics [24]. In order to model these statistical characteristics we need to develop different conditioning functions.

Intuitively, conditioning functions can be viewed as a form of pre-processing which transforms a set of uniform and independent Boolean variables (*e.g.* x and x') into some output distributions. As shown in Fig. 6, if unique independent variables are used for the conditioning functions associated with each primary input, then the resulting conditioning functions are independent. However, if some variables are shared across conditioning functions, as in Fig. 7, the resulting conditioning functions may not be independent and could exhibit correlations.

A. Non-uniform Conditioning Functions

As described by Eq. (1), the probability of a particular transition is proportional to the number of satisfying assignments (*i.e.* minterms) of the associated conditioning function. To model

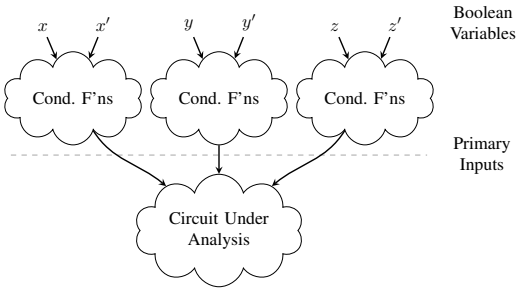

Fig. 6: Independent Condition Functions.

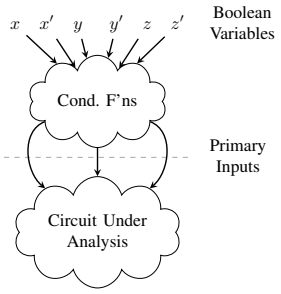

Fig. 7: Non-Independent Condition Functions.

non-uniform input probabilities, we can assign the minterms in proportion to the desired probabilities. However care must be taken to ensure no minterms are shared between conditioning functions associated with the same primary input, as each transition must be independent (*e.g.* R and F cannot occur simultaneously). Furthermore, by adding additional variables to the conditioning functions' support, probabilities can be assigned at finer granularity.

Fig. 8 shows how a set of conditioning functions for a non-uniform primary input can be created. Each transition is assigned to a unique subset of the possible minterms in proportion to the desired probability.

There are many possible conditioning functions which satisfy a set of target probabilities. The algorithm we use to construct non-uniform conditioning functions is shown in Algorithm 3. This approach assigns adjacent minterms to a particular transition. Since the minterms for a particular transition are grouped together, the conditioning function is easier to encode as a BDD.[13] First the conditioning functions are initialized on Lines 2 to 5. Next the transitions are sorted (Line 7), so the highest probability transition is processed first. The number of minterms to assign to each transition is calculated on Line 10, based on transition probability and the number of Boolean variables per input (q). The relevant minterms are then set in the conditioning function on Line 13, and the conditioning functions are returned on Line 15.

B. Correlated Condition Functions

It is also possible to construct conditioning functions which capture correlations between inputs. In such cases the number of minterms shared between two conditioning functions describes their degree of correlation.

[13]Evaluation on 12 of the MCNC20 benchmarks showed this approach reduced ESTA run-time by $1.8\times$ compared to a round-robin allocation, such as shown in Fig. 8.

x_1	x_2	x_3	f
0	0	0	R
0	0	1	F
0	1	0	H
0	1	1	L
1	0	0	R
1	0	1	L
1	1	0	L
1	1	1	L

$$f_R(x_1,x_2,x_3) = (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3})$$
$$f_F(x_1,x_2,x_3) = (\overline{x_1} \wedge \overline{x_2} \wedge x_3)$$
$$f_H(x_1,x_2,x_3) = (\overline{x_1} \wedge x_2 \wedge \overline{x_3})$$
$$f_L(x_1,x_2,x_3) = (\overline{x_1} \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge x_2 \wedge x_3)$$

Fig. 8: Example Truth Table and conditioning functions satisfying $P(f_R) = 0.25$, $P(f_F) = 0.125$, $P(f_H) = 0.125$ and $P(f_L) = 0.5$.

Algorithm 3 Grouped Minterm Allocation

Require: q the number of Boolean variables per primary input, P the desired probabilities of $R/F/H/L$ transition

1: **function** AllocateGroupedMinterms(q, P)
2: $f[R] \leftarrow$ False
3: $f[F] \leftarrow$ False
4: $f[H] \leftarrow$ False
5: $f[L] \leftarrow$ False
6: $Trans \leftarrow (R, F, H, L)$
7: SortDescendingProbability($Trans, P$)
8: $m \leftarrow 0$ ▷ Current minterm number
9: **for** $t \in Trans$ **do**
10: $M \leftarrow P[t] \cdot 2^q$ ▷ Number of minterms to allocate
11: **for** $1 \ldots M$ **do**
12: $f_m \leftarrow$ GetMintermFunction(q, m)
13: $f[t] \leftarrow f[t] \vee f_m$ ▷ Set minterm
14: $m \leftarrow m + 1$
15: **return** $(f[R], f[F], f[H], f[L])$

Consider the following joint probability specification between inputs a and b:

$$\begin{array}{c} \\ b_R \\ b_F \\ b_H \\ b_L \end{array} \begin{array}{c} \begin{array}{cccc} a_R & a_F & a_H & a_L \end{array} \\ \begin{pmatrix} 1/16 & 0 & 0 & 1/8 \\ 1/16 & 0 & 1/8 & 0 \\ 1/16 & 1/8 & 1/4 & 0 \\ 1/16 & 1/8 & 0 & 0 \end{pmatrix} \end{array} \tag{10}$$

where each entry corresponds to the probability of specific transitions occurring simultaneously on a and b.

In Eq. (10) a_R is uncorrelated with the transitions on input b, since it is equally likely to occur with any transition on b. In contrast, a_F occurs only when b is undergoing a H or L transition, a_H occurs only if b is undergoing a F or H transition, and a_L occurs only if b is undergoing a R transition.

A set of conditioning functions which model this behaviour

are:

$$\begin{aligned}
f_{a_R}(x_1,x_2,x_3,x_4) &= (\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_0 \wedge x_2 \wedge x_3) \\
&\vee (\overline{x_0} \wedge x_1 \wedge \overline{x_2} \wedge x_3) \\
f_{a_F}(x_1,x_2,x_3,x_4) &= (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_0} \wedge x_1 \wedge x_2) \\
f_{a_H}(x_1,x_2,x_3,x_4) &= (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_0 \wedge x_1 \wedge \overline{x_2}) \\
&\vee (x_0 \wedge x_1 \wedge \overline{x_3}) \vee (x_0 \wedge \overline{x_2} \wedge \overline{x_3}) \\
&\vee (\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3) \\
f_{a_L}(x_1,x_2,x_3,x_4) &= (x_0 \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3) \\
&\vee (\overline{x_0} \wedge \overline{x_1} \wedge x_2 \wedge x_3) \\
f_{b_R}(x_1,x_2,x_3,x_4) &= (x_0 \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3) \\
&\vee (\overline{x_0} \wedge x_1 \wedge \overline{x_2} \wedge x_3) \\
&\vee (\overline{x_0} \wedge \overline{x_1} \wedge x_2 \wedge x_3) \\
f_{b_F}(x_1,x_2,x_3,x_4) &= (x_0 \wedge x_1 \wedge x_2) \vee (\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3) \\
f_{b_H}(x_1,x_2,x_3,x_4) &= (\overline{x_1} \wedge \overline{x_3}) \vee (\overline{x_2} \wedge \overline{x_3}) \vee (x_0 \wedge x_1 \wedge \overline{x_2}) \\
f_{b_L}(x_1,x_2,x_3,x_4) &= (x_0 \wedge \overline{x_1} \wedge x_2 \wedge x_3) \vee (\overline{x_0} \wedge x_1 \wedge x_2)
\end{aligned} \tag{11}$$

Where, for example:

$$\begin{aligned}
P(a_H \wedge b_F) &= \frac{\#SAT(f_{a_H} \wedge f_{b_F})}{2^4} \\
&= \frac{\#SAT((x_0 \wedge x_1 \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3))}{16} \\
&= \frac{2}{16}
\end{aligned} \tag{12}$$

as desired, since f_{a_H} and f_{b_F} share the two minterms: $x_0 \wedge x_1 \wedge x_2 \wedge \overline{x_3}$ and $\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2} \wedge x_3$.

In general, constructing a set of conditioning functions that satisfy an arbitrary set of joint probabilities for a large number of inputs is challenging. This task can be formulated as a constrained minterm assignment problem, and solved with Integer Linear Programming (ILP). However this approach is computationally expensive, and as a result it is only practical for groups of less than 4 correlated circuit inputs.

VI. 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. 9. We take in a circuit netlist which is mapped onto a 40nm 6-input Look-Up-Table (6-LUT) based FPGA using VPR [25] 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.

We evaluate all tools on the 20 largest MCNC benchmarks [26] which are listed in Table VI. Any state elements (*e.g.* Flip-Flops) were replaced with primary inputs and outputs.[14]

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 $\Theta(4^I)$. 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.

[14]While our ESTA implementation does not perform clock-pessimism removal, the same techniques used in traditional STA could be applied [27], [28].

It is important to distinguish between the strength of guarantees that MC and ESTA provide. ESTA guarantees it will *always* produce safely pessimistic upper bounds of path-delay distributions. MC cannot 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 criterion.

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 to be when we are confident the relative error of the sample maximum-delay *probability* across multiple samples is sufficiently small. 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 α confidence, we say the sample has converged if $\frac{UB-LB}{\hat{p}} < \Delta_{\hat{p}_{rel}}$.

For instance, choosing $\alpha = 0.95$ and $\Delta_{\hat{p}_{rel}} = 0.05$ corresponds to a 95% confidence that the relative error in $\hat{p}$ is less than 5%. MC run-time is dependent on how α and $\Delta_{\hat{p}_{rel}}$ are chosen and their effect is evaluated in Section VII-C.

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

B. Metrics

To compare the Quality of Results (QoR) between STA, ESTA and MC we used the Earth Mover's Distance (EMD) metric [29], 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 (assumed true switching behaviour). For each benchmark we report the run-times for determining the maximum delay across all outputs, and for determining the delay distribution of each output. We also report the mean normalized EMD of the resulting path-delay distributions.

VII. Results

Using the experimental methodology from Section VI 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, $m = 10^4$ and $s = 0.0$, which were found experimentally to offer good run-time/quality trade-offs as shown in Section VII-D. For MC we used $\alpha = 0.95$ and $\Delta_{\hat{p}_{rel}} = 0.05$ unless otherwise noted.

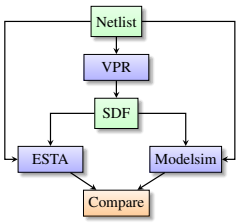

Fig. 9: Evaluation Flow.

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

Results were collected on an Intel Xeon E5-2643v3 machine with 256GB of memory.

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 each possible set of input transitions, that ESTA agreed with simulation on the resulting output transitions and that ESTA's delay estimate was an upper-bound of the simulation delay.

B. Maximum Delay Estimation with False Paths

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`.[15]

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. Monte-Carlo Convergence

The specific criteria chosen for MC convergence (Definition 8) can affect whether MC converges and its associated run-time.

Fig. 10 illustrates the impact of varying these parameters on the run-time of the `apex2` benchmark, a representative example. Increasing the confidence level (α) increases run-time, particularly as the confidence approaches 1.0.[16] Decreasing the allowed relative error in max-delay probability ($\Delta_{\hat{p}_{rel}}$) strongly increases run-time. Reducing the allowed relative max-delay probability variation from 5% to 2.5% causes run-time to more than triple, and exceed 48 hours for confidences beyond 0.99.

Table IV shows the impact, across all benchmarks, of varying $\Delta_{\hat{p}_{rel}}$ and α on MC max-delay convergence. For relatively loose convergence criteria ($\Delta_{\hat{p}_{rel}} = 0.05$ and $\alpha = 0.95$), only 10 of the 20 MCNC benchmarks converge. Increasing the confidence level (α) increases run-time, but decreasing the allowed relative error in $\hat{p}$ has a more substantial impact, causing substantial

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

[16]Higher α causes the size of the calculated confidence interval to increase slowing convergence.

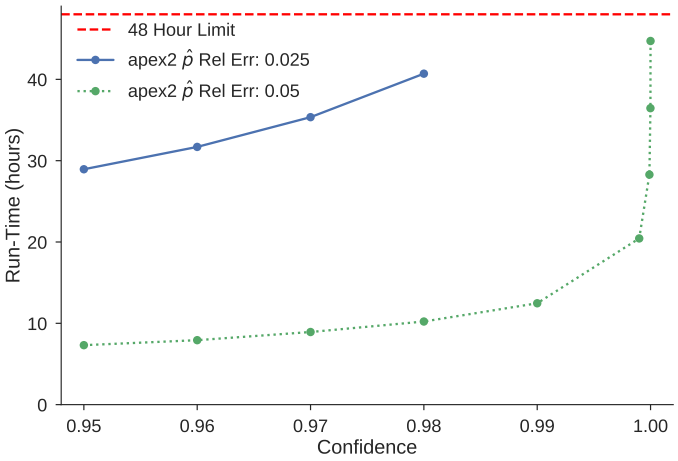

Fig. 10: MC max-delay run-time on `apex2` for various convergence criteria.

TABLE IV: IMPACT OF CONVERGENCE CRITERIA ON MC MAX DELAY RUN-TIME OF THE MCNC20 BENCHMARKS.

$\Delta_{\hat{p}_{rel}}$	α	# Converged Benchmarks	Median MC Run-time (hours)
0.05	0.95	10	1.09
0.05	0.99	10	1.88
0.05	0.999	10	3.07
0.025	0.95	10	4.34
0.025	0.99	8	3.62
0.025	0.999	8	5.92
0.01	0.95	6	11.60
0.01	0.99	5	17.57
0.01	0.999	5	28.71
0.005	0.95	3	21.11
0.005	0.99	2	33.13
0.005	0.999	0	

Blank entries exceeded 48 hours run-time

increases in the median run-time, and reducing the number of converged benchmarks. At $\Delta_{\hat{p}_{rel}} = 0.005$ and $\alpha = 0.999$, MC fails to converge on all benchmarks.

It should be noted that to equal ESTA's strong upper-bound guarantee (analogous to 1.0 confidence) MC effectively reverts to exhaustive simulation. Furthermore, all comparisons we make between MC and ESTA are performed with relatively loose MC convergence criteria ($\Delta_{\hat{p}_{rel}} = 0.05$, $\alpha = 0.95$), and both MC convergence and run-time would be much worse if tighter convergence criteria were used.

D. ESTA Run-time/Accuracy Trade-Offs

Fig. 11 shows run-time/accuracy trade-offs for the binning methods described in Section IV-D.

Increasing the fixed bin size (d) from 0 to 100 at $m = \infty$ speeds-up ESTA by 2.2×, with only a 4% degradation in QoR. Beyond $d = 100$ the QoR degrades significantly for little or no run-time improvement.

Decreasing m from ∞ to 10^6 has limited impact on QoR. Further decreasing m to 10^4 causes some impact, resulting in a 40% QoR degradation for speed-up of 1.5× compared to $m = \infty$ at $d = 10$. At $m = 10^2$ speed-up compared to $m = \infty$ improves further (e.g. a further 1.8× compared to $m = \infty$ at

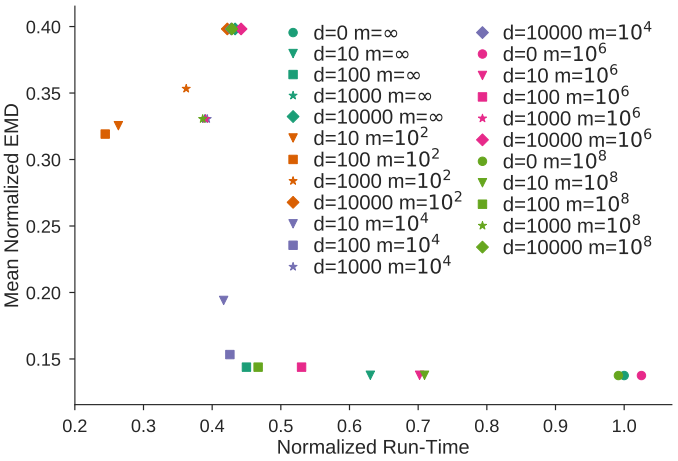

Fig. 11: ESTA run-time/accuracy tradeoffs on the `dsip` benchmark in per-output mode.

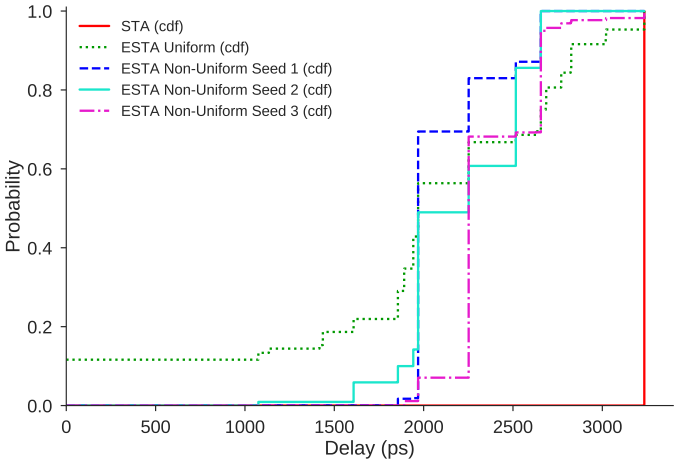

Fig. 12: CDFs for the `pksi_83_d` output of the `dsip` benchmark for various conditioning functions. The 'ESTA Uniform' CDF corresponds to using the conditioning functions from Eq. (4) for each primary input. Each 'ESTA Non-Uniform' CDF correspond to a different set of randomly generated non-uniform conditioning functions used at the primary inputs. ESTA was run with $d = 1$ and $m = \infty$.

$d = 100$), but at the cost of significant QoR degradation (e.g. a further 222% compared to $m = \infty$ at $d = 100$).

In general $d = 100$ and m in the range 10^4 to 10^6 provide good run-time/accuracy trade-offs. d is the most effective, and setting it to a small but non-negligible portion of the circuit delay ensures only timing tags with similar delays are merged together. This reduces run-time with minimal QoR impact. m serves to limit ESTA's worst-case complexity by bounding the $O(L^K)$ term in Eq. (8). Setting m to a large, but finite value ensures designs with large numbers of convergent timing paths remain tractable, at the cost of some additional pessimism.

E. ESTA Non-Uniform Conditioning Functions

As described in Section V, ESTA can model non-uniform and correlated inputs with appropriate conditioning functions.

Fig. 12 illustrates the impact of different conditioning functions on the delay-probability Cumulative Distribution Functions (CDFs) of the `pksi_83_d` output of the `dsip` benchmark. The condition functions for the 'ESTA Non-Uniform' are constructed to satisfy randomly generated target transition probabilities. By changing the random number generator's seed, different target transition probabilities can be created. In this instance the non-uniform conditioning functions result in a sharper delay transition around 2000ps compared to the uniform conditioning functions. Notably, the different input probability distributions can result in significantly different delay behaviour; for instance the 99$^{\text{th}}$-percentile delay for 'ESTA Non-Uniform Seed 2' is 2656ps, while for 'ESTA Uniform' it is 3235ps, 22% higher.

It is also important to consider the impact of conditioning functions on ESTA's run-time. An key factor is how many Boolean variables (q), are used to form the support of the conditioning functions of each input. Larger values of q allow probabilities to be specified at finer granularity, since the smallest probability that can be specified is $\frac{1}{2^q}$.

Table V shows the run-time impact of using the uniform conditioning functions from Eq. (4), or randomly generated non-uniform conditioning functions, for various values of q. Increasing q has a strong effect on run-time, since it is directly proportional to the total number of Boolean variables.

TABLE V: RELATIVE ESTA PER-OUTPUT RUN-TIME FOR VARIOUS CONDITIONING FUNCTIONS ($d = 100$, $m = 10^4$).

Benchmark	Uniform $q = 2$	Non-Uniform $q = 4$	Non-Uniform $q = 6$
ex5p	1.00	1.08	1.73
apex4	1.00	4.33	7.10
ex1010	1.00	1.26	1.87
s298	1.00	2.82	6.60
misex3	1.00	5.58	8.75
alu4	1.00	8.58	14.07
spla	1.00	1.34	7.30
pdc	1.00	2.91	12.27
apex2	1.00		
seq	1.00	41.12	
des	1.00	4.54	18.25
clma	1.00		
tseng	1.00		
diffeq			
dsip	1.00	4.86	10.34
bigkey	1.00	6.23	12.00
frisc			
elliptic			
s38584.1	1.00	12.22	
s38417			
Geomean	1.00	4.37	7.42

Blank entries exceeded 48 hours run-time

Increasing q to 4 and 6 allows probabilities to be specified with 6.25% and 1.56% granularity respectively. However this comes at a cost to run-time, with ESTA taking 3.2× and 7.4× longer on the common benchmarks which completed. Clearly, the smallest value of q which captures the non-uniform input distribution should be used to minimize run-time.[17]

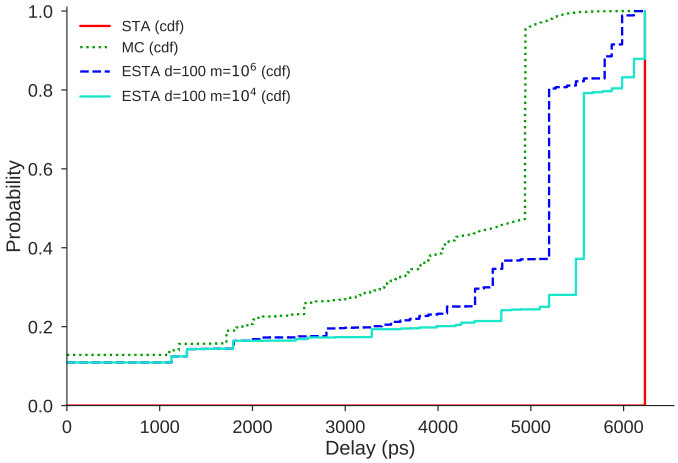

Fig. 13: Maximum delay CDFs on the `spla` benchmark.

F. ESTA and MC Comparison

Fig. 13 plots the maximum path-delay CDFs for the `spla` benchmark. 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 VI quantitatively compares ESTA and MC.

1) Max-Delay Comparison: First, we consider the behaviour of ESTA and MC when evaluated in Max-Delay mode (Section IV-H), where a single path-delay distribution represents the maximum delay over all timing endpoints.

For MC max-delay probability ($\hat{p}$) we observe that benchmarks with relatively large max-delay probability tend to converge, while those with smaller probability ($\hat{p} \leq 10^{-4}$) tend not to converge. A relatively high $\hat{p}$ means max-delay paths are observed relatively frequently, and the confidence interval around $\hat{p}$ shrinks quickly. In contrast, on benchmarks with relatively rare max-delay paths (*e.g.* $\hat{p} \leq 10^{-4}$) MC often fails to converge within 48 hours. Intuitively, it is difficult to determine from a given sample 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 Max-Delay QoR for ESTA m=10^4, 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 geometric mean of 0.63 across the benchmarks which completed. Increasing m to 10^6 (Max-Delay ESTA m=10^6) reduces the geometric mean normalized EMD by 24% to 0.48 on the common benchmarks which completed.

The run-time performance of Max-Delay ESTA and MC are also shown in Table VI. Looking at Max-Delay Run-time for 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`). Max-Delay Run-time for 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 $m = 10^6$ achieved a geometric mean speed-up of $164\times$ and $16\times$ respectively. For all benchmarks with > 415 inputs ESTA in Max-Delay mode exceeded 48 hours run-time during #SAT evaluation.

2) Per-Output Comparison: We can also evaluate ESTA and MC in Per-Output mode (Section IV-H), where a unique path-delay distribution is calculated for each timing endpoint.[18] Since there are multiple path-delay distributions produced we calculate the normalized EMD and report the arithmetic mean across all timing endpoints. For MC convergence we report the time required for the maximum delay of each timing endpoint to converge, according to Definition 8.

Table VI also shows the Per-Output QoR; ESTA produces a more accurate analysis than in the Max-Delay mode: with a geometric mean EMD of 0.28 and 0.25 for $m = 10^4$ and $m = 10^6$ respectively.

Looking at Per-Output Run-time in Table VI we can see that MC converges on only 6 of 20 benchmarks, and total run-time increased $4.3\times$ compared to MC Max-Delay mode for those which converged. In comparison, in Per-Output mode ESTA converges on more benchmarks: 16 and 13 for $m = 10^4$ and $m = 10^6$ respectively. On the common benchmarks which completed, ESTA's geometric mean speed ups over MC were $569\times$ and $109\times$ for $m = 10^4$ and $m = 10^6$ respectively.

3) Discussion: The QoR gap between ESTA and MC is derived from four factors. First, binning (Section IV-D) introduces additional pessimism since the true distribution is approximated with fewer timing tags. Second, Modelsim performs more aggressive transition filtering than ESTA (Section IV-B). Since Modelsim simulates glitching behaviour it can filter transitions when some inputs have not yet stabilized to their final values. In contrast, ESTA does not directly model glitching behaviour, instead calculating upper bounds on the stable arrival times, and only filters based on the stable post-transition signal value. To illustrate this difference consider Fig. 4. ESTA will filter based on the signal state only after $t = 1 + \delta$ (when the signal is guaranteed to be stable and glitch-free), while Modelsim will filter based on the signal state for the full time period (even before the signal has stabilized).[19] 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. Fourth, on some benchmarks MC fails to converge (*e.g.* `s298`), meaning our assumed ground-truth may not be the actually the ground-truth. In such cases, if ESTA's calculated

[17]It should be noted that despite these slow downs ESTA would still outperform MC (Section VII-F).

[18]This more indicative of how an ESTA-like tool would be used to analyze an overclocking-style design, since the acceptable error rates are likely to be endpoint dependent.

[19]ESTA's behaviour ensures it maintains the monotone speed-up property [21]. Modelsim's analysis does not satisfy the monotone speed-up property, since it filters transitions in a delay-dependent manner.

path-delay distributions are closer to the real ground-truth than MC's, the reported normalized EMD may be pessimistic.

It is also interesting to note that MC performs worse (converges on fewer benchmarks and runs slower) when run in Per-Output mode, while ESTA exhibits the inverse behaviour. For MC, Per-Output mode requires that all timing endpoints converge according to Definition 8, while MC Max-Delay mode requires only that the maximum delay converges. Thus Per-Output is a stricter convergence criteria, since it requires potentially non-maximal delay outputs to converge. For ESTA, Per-Output mode is faster since it does not require tracking the covered input transitions, and produces better QoR since it avoids the final stage of tag merging and binning used to compute the maximum distribution over all timing endpoints in Max-Delay mode (Section IV-H).

Overall, these results show that ESTA can be used to quickly calculate bounding path-delay distributions on most of the circuits, including some with 100s or 1000s of primary inputs. This would enable automated analysis of practical design components such as the 12-input overclocked designs considered in [5].

VIII. 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. We also described how non-uniform probabilities and correlations can be modelled in ESTA. Our experimental comparisons of ESTA and Monte-Carlo timing simulation, showed ESTA on average runs 16 to $569\times$ faster while achieving results within 25 to 63% of Monte-Carlo, representing a 75 to 37% reduction in pessimism compared to STA.

ESTA is more general than the previous state of the art for calculating path delay distributions (manual analysis by hand), and is more scalable than Monte-Carlo timing simulation while performing a more robust analysis with stronger correctness guarantees. ESTA's high complexity does limit its worst-case scalability, and analysis of large-scale designs remains intractable, as they do for all known general methods that analyze input-dependent circuit behaviour. However, ESTA is suitable for use on high value sub-circuits where the additional design effort, pessimism reduction and strong correctness guarantees are warranted.

There are a variety of directions for future work. The key algorithmic challenge for ESTA is scalability, with #SAT being the main run-time bottleneck in our implementation.

One potential approach is to simplify the transition model from the 4-state model ('R', 'F', 'H', 'L') used in this work, to a simpler 2-state model (e.g. 'Static', 'Switch'). This would reduce ESTA's complexity to $O(nL^K + 2^I)$, but would likely increase pessimism since filtering (Section IV-B) would become less effective.

There have also been a variety of works proposing accuracy/guarantee trade-offs while approximating the solution to #SAT [15]. These approximations could substantially reduce the time required to perform #SAT provided: some potential for error in the analysis is acceptable, or #SAT can be approximated pessimistically (*e.g.* only ever safely over-estimating activation probabilities). Whether these accuracy/guarantee/complexity trade-offs are worthwhile requires further investigation.

Another avenue for investigation is using CNF-based solvers instead of BDDs for solving #SAT. It would also be interesting to compare ESTA's efficiency at identifying false paths with other dedicated false path detection algorithms.

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, warrants further investigation.

There are also open questions driven by the application of ESTA. While it is possible to model arbitrary correlations in ESTA it is not clear how to automatically construct such conditioning functions in an efficient manner. It is also not clear how best 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, since ESTA enables automated evaluation of the path-delay distribution associated with different circuit implementations, it could be used to drive design optimization. This would require new metrics to guide optimization which combine path delay and path activation probability information. Potential metrics could include a probabilistically weighted slack, or an error-rate slack. Such metrics would allow designers and optimization tools (such as technology mapping or placement) to directly optimize a design's error-rate based on its physical implementation. This would enable new trade-offs between delay, path sensitization probability, and other circuit characteristics which would have interesting applications to approximate computing techniques such as overclocking.

Acknowledgments

This work was supported by an NSERC CGS-D scholarship, the NSERC/Intel Industrial Research Chair in Programmable Silicon, EPSRC (EP/P010040/1, EP/K034448/1), the Royal Academy of Engineering, and Imagination Technologies. This research was also enabled in part by SciNet [30] and Compute Canada [31].

References

[1] S. Sapatnekar, "Static timing analysis," in *EDA for IC implementation, circuit design, and process technology*, L. Lavagno *et al.*, Eds. 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 Trans. Comput.-Aided Design Integ. Circuits. Syst.*, 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, pp. 1–6.

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

Benchmark	I	LUTs	Max-Delay Prob.	Max-Delay QoR (norm. EMD)		Max-Delay Run-time (minutes)			Per-Output QoR (mean norm. EMD)		Per-Output Run-time (minutes)		
			MC $\hat{p}$	ESTA m=10^4	ESTA m=10^6	MC	ESTA m=10^4	ESTA m=10^6	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	0.40	0.33	305.1	0.7	5.6
apex4	9	970	$5.9 \cdot 10^{-3}$	0.85	0.74	118.1	1.5	8.7	0.63	0.48	712.0	1.2	8.7
ex1010	10	3093	$2.7 \cdot 10^{-3}$	0.96	0.85	922.1	5.8	40.4	0.84	0.65		10.1	127.4
s298	11	1301		0.87	0.79		5.7	59.4	0.54	0.45		15.7	118.6
misex3	14	1158	$1.5 \cdot 10^{-3}$	0.66	0.46	781.3	5.2	54.9	0.41	0.34		4.4	68.3
alu4	14	1173	$2.5 \cdot 10^{-3}$	0.83	0.46	473.6	2.4	58.9	0.58	0.44	1966.4	1.8	62.3
spla	16	3005		0.40	0.25		6.3	207.1	0.26	0.19		12.7	108.3
pdc	16	3627		0.56	0.31		11.3	118.4	0.39	0.25		15.5	282.7
apex2	39	1478	$6.9 \cdot 10^{-4}$	0.33		2068.2	87.1		0.31			80.1	
seq	41	1325		0.34			57.9		0.19	0.15		5.7	164.5
des	256	554	$1.8 \cdot 10^{-2}$			60.7			0.27	0.23	1101.8	0.6	2.8
clma	415	6239		0.92			319.2		0.13			129.4	
tseng	436	798							0.15			87.0	
diffeq	440	871											
dsip	460	880	$3.5 \cdot 10^{-2}$			49.8			0.15	0.14	56.1	0.5	0.6
bigkey	494	883	$1.2 \cdot 10^{-2}$			150.6			0.17	0.16	151.2	0.4	0.5
frisc	905	3028											
elliptic	1252	2135											
s38584.1	1332	4486	$3.2 \cdot 10^{-3}$			2829.4			0.09	0.07		26.3	111.8
s38417	1545	3465											
Common Geomean			$3.0 \cdot 10^{-3}$	0.80	0.55	360.8	2.2	22.1	0.31	0.27	398.3	0.7	3.6

'Common Geomean' is the geomean of the subset of benchmarks which completed in MC, ESTA $m = 10^4$ and ESTA $m = 10^6$ in the same mode. Note this means Max-Delay and Per-Output geomeans are not comparable.
Blank entries exceeded 48 hours run-time.

[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] K. E. Murray *et al.*, "Quantifying Error: Extending Static Timing Analysis with Probabilistic Transitions," in *DATE*, 2017, pp. 1486–1491.

[8] Y.-C. Hsu *et al.*, "Finding the longest simple path in cyclic combinational circuits," in *ICCAD*, Oct 1998.

[9] H. C. Chen and D. H. C. Du, "Path sensitization in critical path problem," in *ICCAD Digest of Technical Papers*, Nov 1991, pp. 208–211.

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

[11] R. I. Bahar *et al.*, "Timing analysis of combinational circuits using ADDs," in *Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC*, Feb 1994, pp. 625–629.

[12] P. Ashar and S. Malik, "Functional timing analysis using ATPG," *IEEE Trans. Comput.-Aided Design Integ. Circuits. Syst.*, vol. 14, no. 8, pp. 1025–1030, Aug 1995.

[13] Y.-T. Chung and J.-H. R. Jiang, "Functional Timing Analysis Made Fast and General," *IEEE Trans. Comput.-Aided Design Integ. Circuits. Syst.*, vol. 32, no. 9, pp. 1421–1434, Sep. 2013.

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

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

[16] S. Devadas *et al.*, "Computation of floating mode delay in combinational circuits: theory and algorithms," *IEEE Trans. Comput.-Aided Design Integ. Circuits. Syst.*, vol. 12, no. 12, pp. 1913–1923, Dec 1993.

[17] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[18] F. Somenzi, "CUDD: CU Decision Diagram package release 2.5.1," University of Colorado at Boulder, 2015. [Online]. Available: http://vlsi.colorado.edu/~fabio/CUDD

[19] C. Visweswariah *et al.*, "First-order incremental block-based statistical timing analysis," *IEEE Trans. Comput.-Aided Design Integ. Circuits. Syst.*, vol. 25, no. 10, pp. 2170–2180, Oct 2006.

[20] T.-W. Huang and M. D. F. Wong, "Opentimer: A high-performance timing analysis tool," in *ICCAD*, 2015, pp. 895–902.

[21] P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and Its Implications*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[22] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *ICCAD*, 1993, pp. 42–47.

[23] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs A Practical Approach*. New York, NY, USA: Springer Science + Business Media, 2009.

[24] S. Ramprasad *et al.*, "Analytical estimation of signal transition activity from word-level statistics," *IEEE Trans. Comput.-Aided Design Integ. Circuits. Syst.*, vol. 16, no. 7, pp. 718–733, Jul 1997.

[25] 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.

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

[27] J. Zejda and P. Frain, "General framework for removal of clock network pessimism," in *ICCAD*, Nov 2002, pp. 632–639.

[28] T.-W. Huang *et al.*, "UI-timer: An Ultra-fast Clock Network Pessimism Removal Algorithm," in *ICCAD*, 2014, pp. 758–765.

[29] 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.

[30] C. Loken *et al.*, "SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre," *Journal of Physics: Conference Series*, vol. 256, no. 1, 2010.

[31] www.computecanada.ca.

Kevin E. Murray (S'12) received his BASc in Engineering Science in 2012 and MASc in Electrical and Computer Engineering in 2015, both from the University of Toronto, where he is a PhD candidate in Electrical and Computer Engineering. He has previously been a visiting Research Assistant at Imperial College London, and worked on digital design flows at Advanced Micro Devices (AMD). His research interests include CAD for digital systems focusing on timing analysis, latency insensitive design methods and floorplanning for FPGAs.

Andrea Suardi Andrea Suardi received his M.Sc. in Electronic Engineering from the Politecnico di Milano, Italy in 2006 and he gained his Ph.D. in Electronics and Communication Engineering at the same university in 2010. Currently, he is a research associate at Imperial College London, UK. His research interests are in digital architecture design, in particular timing analysis and high efficiency FPGA based systems for supercomputing and control applications.

Vaughn Betz (S'88–M'91–SM'17) received the B.Sc degree in EE from the University of Manitoba in 1991, the MS degree in ECE from the University of Illinois at Urbana-Champaign in 1993, and the PhD degree in ECE from the University of Toronto in 1998. He co-founded Right Track CAD to develop new FPGA architectures and CAD tools and was its VP of Engineering until its acquisition by Altera in 2000. He was at Altera from 2000 to 2011, ultimately as Senior Director of Software Engineering, and is one of the architects of both the Quartus II CAD system and the Stratix I - V and Cyclone I - V FPGAs. He is a Professor and the NSERC/Intel Industrial Research Chair in Programmable Silicon at the University of Toronto, where his research covers FPGA architecture, CAD, and acceleration of computation using FPGAs.

Dr. Betz has published over 70 technical papers in refereed journals and conferences, 12 of which have received best paper or most significant 20/25 year paper awards. He holds 97 issued US patents.

George Constantinides George A. Constantinides (S96-M01-SM08) received the Ph.D. degree from Imperial College London in 2001. Since 2002, he has been with the faculty at Imperial College London, where he is currently Royal Academy of Engineering / Imagination Technologies Research Chair, Professor of Digital Computation, and Head of the Circuits and Systems research group. He has served as chair of the FPGA, FPL and FPT conferences. He currently serves on several program committees and has published over 150 research papers in peer refereed journals and international conferences. Prof Constantinides is a Senior Member of the IEEE and a Fellow of the British Computer Society.