Accurate Estimation of Dynamic Timing Slacks using Event-Driven Simulation

Dimitrios Garyfallou*†, Ioannis Tsiokanos†, Nestor Evmorfopoulos*,
Georgios Stamoulis*, and Georgios Karakonstantis†
*Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece
†Institute of Electronics, Communications and Information Technology, Queen's University Belfast, UK
Email: *{digaryfa, nestevmo, georges}@e-ce.uth.gr †{itsiokanos01, g.karakonstantis}@qub.ac.uk

***Abstract*—The pessimistic nature of conventional static timing analysis has turned the attention of many studies to the exploitation of the dynamic data-dependent excitation of paths. Such studies may have revealed extensive dynamic timing slacks (DTS), however, they rely on frameworks that inherently make worst-case assumptions and still ignore some data-dependent timing properties. This may cause significant DTS underestimation, leading to unexploited frequency scaling margins and incorrect timing failure estimation. In this paper, we develop a framework based on event-driven timing simulation that identifies the underestimated DTS, and evaluate its gains on various post-place-and-route designs. Experimental results show that our event-driven simulation scheme achieves on average 2.35% and up-to 194.51% DTS improvement over conventional graph-based techniques. When compared to existing frequency scaling schemes, the proposed approach enables us to further increase the clock frequency by up-to 10.42%. We also demonstrate that our approach can reveal that timing failures may be up-to 2.94× less than the ones estimated by existing failure estimation techniques, under potential variation-induced delay increase.**

I. Introduction

The advent of the aggressive technology scaling era has introduced extensive static and dynamic parametric variations, which may result in up-to 50% delay variations [1]. As a result, this trend renders circuits prone to timing failures.

State-of-the-art. Conventionally, manufacturers address such failures by adopting guardbands, which essentially provide sufficient timing margins to account for any variation-induced delay increase [2]. However, such margins are considered to be overly pessimistic, since they are estimated statically based on rare operating conditions. In fact, conventional static timing analysis (STA) [3] may be effective in quickly revealing the most critical paths, but assumes worst-case inputs at each circuit node, thus ignoring the data-dependent excitation of each path [2], [4]. This approach ultimately forces circuits to operate at a much lower frequency or at a higher supply voltage than what they could potentially achieve [1].

To circumvent such overheads, recent schemes try to reveal any dynamic timing slack (DTS) that may exist in activated paths by applying dynamic timing analysis (DTA) [5]–[10]. On one side, many of these works [5], [6], [10] try to exploit the available DTS for upscaling the frequency [5], [10], changing the cycle time [6] or reducing the supply voltage [9]. On the other side, other works exploit DTA in order to estimate timing failure rates while considering the processed data [7], [8].

Common characteristic of the above works is that the applied DTA is based on delay-annotated gate-level simulation tools (e.g. ModelSim [11]). Although such tools account for the paths that are dynamically activated, they still assume fixed, worst-case delays which are estimated by Graph Based Analysis (GBA). In fact, GBA provides a pessimistic delay estimation that is based on STA assumptions [3]. None of the above works evaluate the impact of GBA on DTS estimation. A few recent works [9], [12] focus on improving the runtime of graph-based DTA (GB-DTA) rather than improving the DTA accuracy. This means that existing approaches may be leaving a large amount of timing slacks hidden and unexploited.

Contributions. In this paper, we primarily aim at unveiling the unexploited DTS ignored by all previous approaches based on GB-DTA, by taking into consideration the actual data-dependent path delays. The main contributions of this work can be summarized as follows:

- We develop a framework to unveil DTS that has been underestimated by prior works. To achieve this, we implement a tool that is based on event-driven dynamic timing analysis (ED-DTA), which has been established as the most accurate DTA method [13].
- We compare the DTS calculated by ED-DTA with the DTS estimated by a GB-DTA method which is based on delay-annotated gate-level simulation. Our results indicate that ED-DTA leads to 2.35% on average and up-to 194.51% more DTS, in terms of relative difference, compared to GB-DTA. Considering only the critical activated paths, the average DTS improvement is increased to 11.2%.
- We demonstrate the impact of underestimated DTS on clock frequency and timing failures. First, we measure the point of first failure (PoFF) between ED-DTA and GB-DTA, indicating the frequency improvement. Second, we estimate the timing errors manifested under potential variation-induced worst-case delay increase levels for ED-DTA and GB-DTA.

The rest of the paper is organized as follows. Section II discusses the background and the motivation of our work, while Section III describes the proposed approach and the implemented design flow. Section IV presents the experimental results. Conclusions are drawn in Section V.

II. Background and Motivation

A. Static Timing Analysis

Typically, a digital circuit consists of circuit elements (i.e. gates) connected with interconnects, and can be represented as a graph composed of nodes (input/output ports and gate pins), and edges (i.e. timing arcs) which are connecting the nodes. Each distinctive directed connection of timing arcs, thus nodes, forms a *timing path*. Each of the numerous timing paths require some time to propagate the signal, which essentially defines the delay of each path. This path delay is computed by adding up the delays of all timing arcs included in the path. Note that each timing arc delay is a function of input *slew* which is defined as the amount of time required for the signal to transition from high-to-low or low-to-high. The input *slew* on a timing arc also determines the *slew* on the output node of the arc.

One of the most essential steps in the design of any circuit is the identification of the worst-case critical path and the estimation of its delay, which eventually determines the maximum operational frequency of the circuit. Such a step requires to carry out STA, which is typically done using an early-late split, where each path has an early (lower) bound and a late (upper) bound on its delay to account for various parametric variations [3].

STA propagates the upper and lower slew bounds on the nodes through the timing arcs. Based on that, it calculates the earliest and the latest timing instants that a signal reaches a circuit node. These timings are quantified as earliest and latest *arrival time (at)*, while the limits imposed on a circuit node for proper logic operation are quantified as earliest and latest *required arrival time (rat)*. The difference between the required arrival time and signal arrival time at a circuit node determines the *slack*, which quantifies how well the timing constraints are met. That is, a positive slack means the required time is satisfied, and a negative slack means the required time is in violation. The *rat* and *slack* are defined in the following equations, taking into consideration the setup constraint t_{setup} which denotes the amount of time that the signal should be stable before the active clock edge on every flip-flop (FF) to ensure proper operation:

$$rat^{setup} = rat_D^{late} = at_{CK}^{early} + T_{clk} - t_{setup} \tag{1}$$

$$slack^{setup} = slack_D^{late} = rat_D^{late} - at_D^{late} \tag{2}$$

where T_{clk} is the clock period, and CK and D denote the clock and the data pin of the testing FF, respectively.

To provide a quick estimation of the worst setup timing slack among all paths, which is adequate for determining T_{clk}, GBA is applied which assumes worst-case (upper) arrival time and slew bounds on each node [3]. GBA which is essentially the default STA method used in commercial tools (e.g. PrimeTime) may provide quick estimations but ignores the lower or intermediate arrival times and slew bounds. As a result, GBA may lead to a substantially different slack estimation than the actual one, which depends on input data.

B. Dynamic Timing Analysis

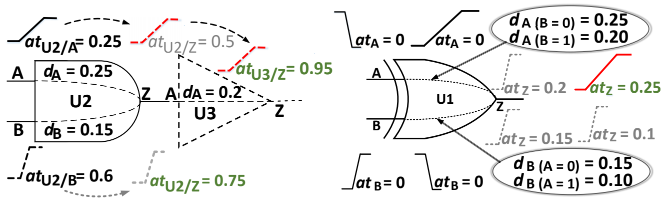
It has been recently estimated [8] that roughly 99% of critical paths are triggered by less than 10% of all operations and the chance to experience the worst-case input conditions and the upper slew bounds assumed by STA is very low [14]. These findings have turned the attention of many studies into the estimation of the so-called Dynamic Timing Slack (DTS) that may exist within any path depending on the dynamically changing processed data [5]–[10]. Such works aim at revealing any unexploited room for occasional frequency up-scaling [5], [6], [10] or voltage down-scaling [9] and for estimating timing error rates in speculative processors [7], [8]. The majority of such studies rely on frameworks that use delay-annotated gate-level simulation tools (e.g. ModelSim [11]). However, the annotated delays inherently impose significant pessimism in slack estimation, since they are extracted (in a form of a Standard Delay Format [SDF] file) from tools (e.g. PrimeTime) that are based on GBA. Therefore, although the developed frameworks can account for the data-dependent path activation, they may still underestimate the available DTS.

To better understand the source of potential slack underestimation, let us take a better look on few shortcomings of GBA along with few examples.

1) Slew merge points. When two $slew$ values arrive at the same node of the graph, GBA propagates forward the worst $slew$ and at. The most common example of a slew merge point is an output pin of a gate where multiple timing arcs terminate. In Fig. 1a, we explain how the $late$ timing information of an AND gate is propagated across the timing graph. In this example, we can see that the worst (max) $slew$ (highlighted in red) and worst (max) at (highlighted in green) on pin U3/A are propagated through different paths. As a result, the $delay$ from U2/B to U3/Z appears to be bigger than real physical delay of the path, since the timing arc delay $d_{U3/A \to U3/Z}$ has been determined by the $slew$ of a different path.

2) Worst-case input state assumptions. Another major source of pessimism in GBA, is the worst-case input state assumptions. In more detail, some gates (e.g. XOR, XNOR, MUX) are state-dependent. This means that a specific transition (rise or fall) on the output pin of a gate can occur for different input transition combinations. Fig. 1b demonstrates the $late$ timing propagation for a XOR gate. In this case, GBA propagates the worst timing information, resulting in even more pessimism in the delay estimation of upstream paths. Note that the worst propagated slew (highlighted in red) at the output pin U1/Z corresponds to the rise transition/slew propagated from U1/A, when U1/B is at logic 0.

To demonstrate the inaccuracy involved in GB-DTA methodologies, let us give an example of DTS estimation for a specific activated path. In Fig. 2, GBA analysis has been performed on a timing graph, where the example gates of Fig. 1 have been combined together. For this initial logic state of the circuit, when the signal on $U1/A$ makes a falling transition $(1 \to 0)$, the path from $U1/A$ to $FF2/D$ is activated. However, recall that the worst slew on $U1/Z$ has been chosen for the rising $(0 \to 1)$ transition of $U1/A$ (see Fig. 1b), which determines the worst-case annotated delays $d_{U2/A \to U2/Z}$ and



(a) Slew merge points (b) Worst-case input state assumptions

Fig. 1: Main sources of pessimism in graph based analysis.

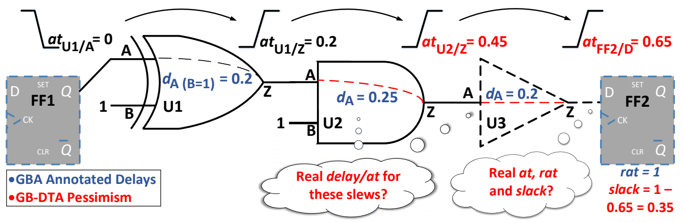


Fig. 2: Graph based analysis pessimism in GB-DTA.

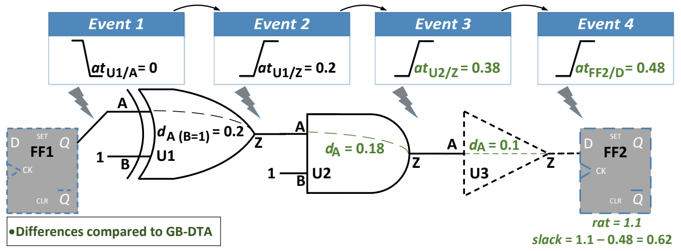


Fig. 3: Accurate DTA using Event-Driven Simulation.

$d_{U3/A \rightarrow U3/Z}$. Thus, $at^{late}_{FF2/D}$ is calculated based on those delays. Note that the $at^{early}_{FF2/CK}$ is assumed to be always equal to the worst-case value calculated in GBA, since the clock path usually consists only of series of inverters/buffers and does not include slew merge points. It is also important that, except from $at^{late}_{FF2/D}$, $rat^{late}_{FF2/D}$ induces additional pessimism in DTS estimation since the t_{setup} value is calculated based on the worst slew propagated on $FF2/D$, during GBA. Assuming $at^{early}_{FF2/CK} = 0$ns, $T_{clk} = 1.2$ns and $t_{setup} = 0.2$ns, GB-DTA estimates the DTS $slack^{late}_{FF2/D} = 1 - 0.65 = 0.35$ns (based on eq. 1, 2), indicating that the slack is overly pessimistic compared to the path's real physical DTS. This indicates that there is a need to reveal the amount of slack that remains unexploited by existing GB-DTA frameworks [5]–[10] and up-to what extent it can affect the point of first failure (PoFF), the dynamic frequency scaling and the failure rate estimation.

III. Proposed Approach

In this section, we present our approach for unveiling DTS that may have been underestimated by GB-DTA methods used in prior works. To this end, we describe the principles and operation of our developed ED-DTA tool and its integration within a state-of-the-art flow used to evaluate DTS.

A. Event-driven DTA (ED-DTA)

Our approach is based on event-driven timing simulation, which is considered as the most accurate DTA method [13]. In the context of ED-DTA, a logic transition at a circuit node is modelled as an event. Therefore, every event is characterized by the logic value, the $slew$ and the at of the corresponding transition. This allows to track every single node excitation that may take place within any timing path of the circuit as opposed to the graph-based methods. In ED-DTA, events are processed in the correct time order by building and traversing a time-sorted event list. During the dynamic simulation, the event with the smallest at is chosen to be propagated forward to its fanout nodes followed by the rest of the events.

Our event-driven approach uses both functional and timing models of the gates. An event arriving at an input pin of a gate causes the functional evaluation of the gate using its functional model. Whenever the logic value on the output pin of a gate changes, a new event is created and has to be placed in the appropriate point in the event list to ensure causality. For this purpose, the gate's timing model is used to calculate the timing characteristics ($slew$, at) for this output node and schedule the generated event, according to its at, for later processing.

The key idea behind the proposed method is that the $slew$ and at of the generated event depend only on the triggering input event. On top of that, the implemented gate timing analysis does not make worst-case assumptions, but takes into account the current logic state of all input-output pins. As described in Section II, correct $slew$ and at propagation along a timing path is crucial since it determines the delay of timing arcs and the $slew$, at, t_{setup}, rat and $slack$ on the endpoint.

To better understand the impact of our approach on the accuracy of DTS estimation, let us take a closer look on the example shown in Fig. 3, where we consider the same setup timing check as shown in Fig. 2. In the case of our approach, when the signal in $U1/A$ makes a falling transition ($1 \rightarrow 0$), an event is created on $U1/A$ (Event 1) which triggers a rising transition ($0 \rightarrow 1$) on $U1/Z$ and the scheduling of the corresponding event (Event 2). In contrast to GBA, the real output $slew$ is calculated and stored on the triggered event to be propagated forward. To this end, when ED-DTA processes the event on $U1/Z$, the real output slew on $U2/Z$ and timing arc delay $d_{U2/A \rightarrow U2/Z}$ are calculated, to schedule the event on $U2/Z$ (Event 3) with its accurate timing characteristics ($slew$ and at). Following the described event propagation procedure, an event reaches on the path endpoint $FF2/D$ (Event 4) at $at_{FF2/D} = 0.48$ns. Assuming that the propagated $slew_{FF2/D}$ (which is smaller than the worst $slew$ propagated by GBA) now results in $t_{setup} = 0.1$ns, and that $at_{FF2/CK} = 0$ns and $T_{clk} = 1.2$ns, ED-DTA estimates the correct DTS $slack_{FF2/D} = 1.1 - 0.48 = 0.62$ns (based on eq. 1, 2), which is significantly bigger than the worst-case value estimated by GB-DTA.

The developed ED-DTA tool operates on the gate-level and represents the circuit as a timing graph. In our implementation, an event is characterized by the node on which the transition occurs, the $slew$, the at and the logic value of the transition. Note that the timing and functional models of the gates are retrieved from the "standard-cell" library which was used to implement the design.

The details of ED-DTA tool are described in Algorithm 1. Our tool takes the gate-level netlist and the signal activity information (VCD) and performs both setup and hold dynamic timing analysis. After creating the timing graph, it reads the VCD file to schedule input-vector events in the event-list and to initialize the circuit to its steady state. During event-driven simulation, the events with the smallest at are propagated through timing arcs to their fanout nodes. If the fanout node is a path endpoint, the tool computes the setup and hold DTS, and an error is reported in case a violation occurs.

B. Realization of the Proposed Approach

The ED-DTA tool described above is integrated within a state-of-the-art Electronic Design Automation (EDA) work-

Algorithm 1 Event-driven DTA (ED-DTA)

```
 1: Read gate-level netlist and create timing graph
 2: Load VCD file and schedule input events in event list
 3: Initialize circuit to its steady-state
 4: while $event\_list\_not\_empty$ do
 5:     Propagate events scheduled on $current\_time$ to
        their fanout nodes
 6:     if fanout node is a path endpoint then
 7:         Calculate DTS and perform setup/hold timing check
 8:     else
 9:         Evaluate logic on output node of fanout gate
10:         if output node logic value has changed then
11:             Calculate output $slew$ and $at$, and schedule
                event in event list
12:         end if
13:     end if
14:     Remove obsolete events from event list
15:     Advance $current\_time$
16: end while
```

flow along with the traditional graph-based DTA and STA methods. The overall workflow is depicted in Fig. 4, where our modifications compared to the conventional EDA flow are highlighted in orange. As can be seen, our workflow consists of a design and an analysis phase.

The first step of the design phase is logic synthesis which is followed by the place and route steps. In this paper, each design is synthesized, placed and routed on the 45nm NanGate standard cell library, using Design Compiler and Innovus, respectively. Note that these steps are performed utilizing optimizations which aim at achieving maximum performance.

The first step of the analysis phase is to verify that the design has met the timing closure. To this end, we developed a graph-based STA (GB-STA) tool based on [15], which provides accurate results compared to commercial tools. Both GB-STA and ED-DTA tools implement the Elmore and Composite Current Source (CCS) timing models for wire and gate delay estimation, respectively. Note that wire delay estimation can be improved by employing a more accurate interconnect timing analysis method [16]. GB-STA tool has also been extended to annotate the worst-case wire and gate delays in an SDF file, which is essential for GB-DTA. For the sake of this analysis, we use post-layout gate-level simulation supported by ModelSim. To enable characterization of DTS, ModelSim outputs a VCD file that contains information about the value changes occurred during simulation. More specifically, in VCD file, we monitor the clock input port, output ports and inputs (data and clock) of all flip-flops. This file is then provided to a custom post-processing tool for DTS estimation. For each path endpoint, this tool identifies the arrival time of the last event in each clock cycle and relates it to the arrival time of the next active clock edge, to estimate the DTS. At the final step of the analysis phase, we compare the DTS measured by ED-DTA with the one estimated by GB-DTA.

To verify that ED-DTA tool implements the functionality of the specification correctly, we compare the logic values on each output port extracted by ED-DTA tool against ModelSim.

IV. Evaluation Results

In this section, we evaluate our methodology for revealing the overly pessimistic estimation of DTS, existing in GB-DTA

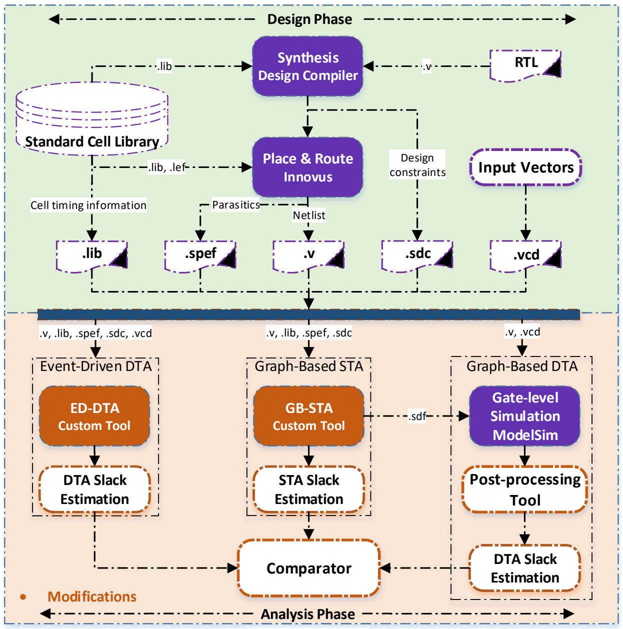


Fig. 4: Workflow of the proposed approach.

methods. First, we compare the DTS estimated by ED-DTA (DTS_{ED}) with the DTS extracted by conventional GB-DTA (DTS_{GB}). Then, we explore potential gains of ED-DTA by exploiting the available DTS. For such an analysis, we use the c6288 and s344 benchmarks from the ISCAS suite [17]; and the fir, brent.kung.32b, kogge.stone.32b, sobel, multiplier.32b and neural.network benchmarks from the AxBench suite [18]. These benchmarks represent a variety of algorithms that covers a wide range of domains, i.e. arithmetic computation, machine learning, signal and image processing. Since DTS depends on the input data, we also extract $100k$ random generated input vectors for each benchmark.

A. Evaluation of DTS

To evaluate the efficacy of our approach, we first estimate DTS_{ED}, DTS_{GB} and the static timing slack of GB-STA (STS_{GB}). Then, we measure the absolute error (AE_{DTA}) between DTS_{ED} and DTS_{GB} for every activated path i, as: $AE_{DTA}(i) = |DTS_{GB}(i) - DTS_{ED}(i)|$. To quantify how big or small AE_{DTA} is relatively to DTS_{GB}, we report the DTS improvement achieved by ED-DTA, which is defined as:
$$DTS_{imp.}(i) = \left|\frac{DTS_{GB}(i) - DTS_{ED}(i)}{DTS_{GB}(i)}\right| \cdot 100 = \frac{AE_{DTA}(i)}{|DTS_{GB}(i)|} \cdot 100$$
where i denotes a specific activated path.

Table I lists these observations along with the size and clock period (T_{clk}) for each benchmark. Note that the average (mean) DTS_{GB} is up-to 4.3× bigger than the average STS_{GB}. According to this table, the average AE_{DTA} ranges up-to 0.049ns, while the maximum (max) AE_{DTA} observed is equal to 0.158ns. As shown, the worst-case assumptions in GB-DTA lead to considerably high inaccuracy. In particular, ED-DTA provides varying degrees of DTS improvement, with the maximum $DTS_{imp.}$ ranging from 13.34% in the case of neural.network to 194.51% for brent.kung.32b. Overall, the proposed ED-DTA reveals on average 2.35% more DTS than GB-DTA. Fig. 7 depicts the distribution of DTS improvement across all activated paths for the considered benchmarks. Note

TABLE I: Static Timing Slacks, Dynamic Timing Slacks and DTS improvement achieved by ED-DTA for various benchmarks

Benchmark	# Gates	T_{clk} (ns)	STS_{GB} (ns)		DTS_{GB} (ns)		DTS_{ED} (ns)		AE_{DTA} (ns)		DTS Improvement ($DTS_{imp.}$)	
			min	mean	min	mean	min	mean	mean	max	mean	max
s344	91	0.34	0.008	0.085	0.042	0.199	0.046	0.203	0.002	0.011	1.19%	20.53%
fir	146	0.7	0.001	0.131	0.014	0.284	0.017	0.289	0.003	0.011	1.11%	27.73%
brent.kung.32b	319	0.38	0.003	0.046	0.003	0.122	0.005	0.127	0.003	0.014	3.15%	194.51%
kogge.stone.32	557	0.34	0.003	0.022	0.003	0.073	0.005	0.079	0.004	0.015	5.41%	86.27%
sobel	564	0.93	0.003	0.249	0.043	0.546	0.064	0.556	0.008	0.036	1.57%	43.55%
c6288	2488	2.94	0.006	0.608	0.615	1.974	0.724	2.024	0.049	0.158	2.81%	20.53%
multiplier.32b	6343	0.96	0.007	0.104	0.043	0.443	0.061	0.455	0.011	0.043	2.43%	38.11%
neural.network	13236	1.14	0.001	0.486	0.137	0.785	0.144	0.794	0.008	0.035	1.11%	13.34%

that the minimum (min) DTS improvement is evaluated to zero across all benchmarks. This can be attributed to the fact that there are activated paths with exactly the same timing behavior and hence identical DTS_{GB} and DTS_{ED}.

Intuitively, input data that activate critical paths increase $DTS_{imp.}$. This can be attributed to the fact that critical paths, which usually consist of more slew merge points and input-state dependent gates compared to less critical paths, are more likely to be affected by the two main sources on pessimism in GB-DTA (see Section II).

B. DTS & Dynamically Activated Critical Paths

To experimentally verify this intuition, we extract the top 5% critical activated paths (5% of the activated paths with the smallest slack). Fig. 5 shows the average and maximum DTS improvement across all benchmarks when only the critical paths are considered. We observe that the average $DTS_{imp.}$ has been increased to 11.2%. Additionally, the maximum $DTS_{imp.}$ of critical paths is equal to the maximum $DTS_{imp.}$ of all activated paths, verifying our intuition that critical paths impose the worst-case inaccuracy in GB-DTA.

C. Dynamic Frequency Scaling & Timing Failures

The underestimated slacks limit the gains achieved by works that exploit DTS [5]–[8], [10]. Such works leverage the available DTS to reduce the clock period up-to the PoFF [5], [6], [10] or estimate the error rates under potential variation-induced delay increase levels [7], [8].

PoFF and the number of timing errors strongly depend on the available DTS (a negative slack means that the path will fail). Fig. 6 shows the slack distributions obtained by ED-DTA and GB-DTA for c6288 and sobel benchmarks. For the sake of simplicity, the horizontal axis shows only the slacks that are less than 50% of T_{clk} (see Table I). In this figure, it can be seen that PoFF varies considerably under GB-DTA and ED-DTA. Particularly, in the case of c6288, based on GB-DTA, PoFF is measured when T_{clk} was reduced to 2.325ns (i.e. DTS_{GB} = 0.615ns). Conversely, ED-DTA under c6288 incurs PoFF at T_{clk} = 2.216ns (i.e. DTS_{ED} = 0.724ns). In the case of sobel, PoFF under GB-DTA and ED-DTA occurs when T_{clk} = 0.887ns (DTS_{GB} = 0.043ns) and T_{clk} = 0.866ns (DTS_{ED} = 0.064ns), respectively. Considering all the examined benchmarks, ED-DTA results in 1.89% on average and up-to 6.22% higher clock frequency than GB-DTA. This improvement corresponds to the minimum achievable clock frequency increase, since PoFF is measured based on the most critical path across all clock cycles of the running application.

Recently, instruction or cycle based frequency scaling schemes have been proposed [5], [6]. To estimate the potential gains that can be achieved by applying ED-DTA to such schemes, we extract the most critical activated path on each cycle which determines PoFF and the achievable frequency for this cycle. For each benchmark, we measure the average (across all cycles) frequency improvement (in terms of relative change over the nominal frequency) achieved by GB-DTA and ED-DTA. Overall, ED-DTA leads to 3.77% on average and up-to 10.42% higher average frequency increase than GB-DTA.

GB-DTA not only leads to pessimistic estimation of DTS, but also exhibits a substantial amount of excited paths with such inaccurate slacks. This finding indicates that there is an increased probability of timing failures in GB-DTA under a potential delay increase. To better illustrate this, let us assume two clock reduction (CR) levels which represent potential variation-induced worst-case delay increase. Specifically, we reduce T_{clk} by 25% (referred to as CR1) and 40% (referred

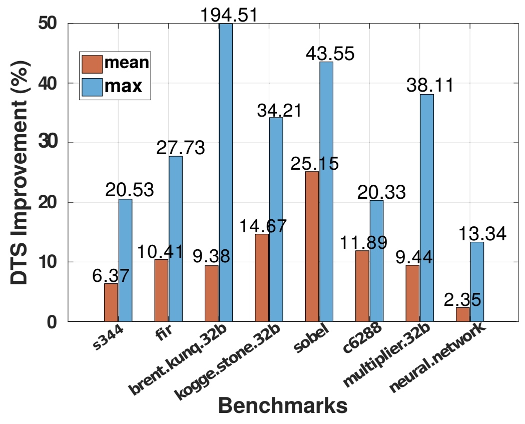
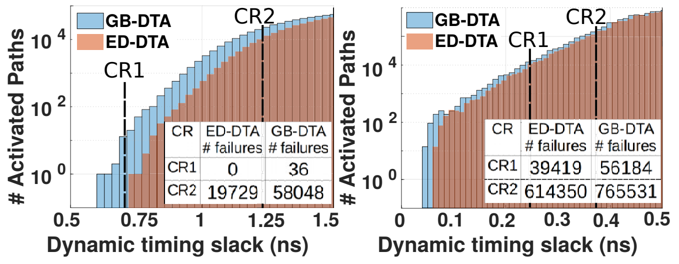


Fig. 5: Mean and maximum DTS improvement achieved by ED-DTA under the top 5% critical activated paths.



CR	ED-DTA # failures	GB-DTA # failures
CR1	0	36
CR2	19729	58048

CR	ED-DTA # failures	GB-DTA # failures
CR1	39419	56184
CR2	614350	765531

(a) c6288 (b) sobel

Fig. 6: DTS distributions obtained by ED-DTA and GB-DTA.

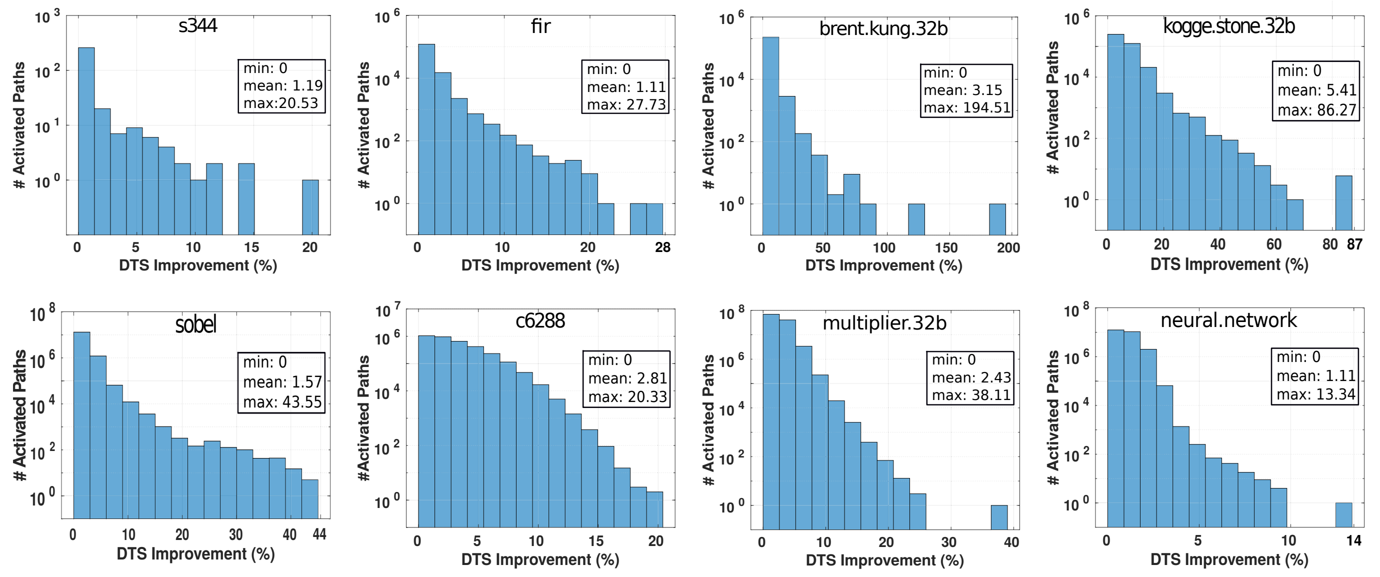


Fig. 7: Distributions of DTS improvement achieved by ED-DTA, relatively to GB-DTA, across all dynamically activated paths.

to as CR2). CR1 and CR2 are consistent with the levels of variation-induced delay increase that have been reported in the literature [1], [4]. As depicted in the bottom right of Fig. 6, under c6288 and CR1, ED-DTA manifests no timing failures, while GB-DTA incurs 36 timing failures. At CR2, GB-DTA results in 2.94× more failures than ED-DTA. Similar results are obtained in the case of sobel, where GB-DTA manifests 1.43× more timing failures compared to ED-DTA at CR2.

D. ED-DTA Runtime

For the performance evaluation of our approach, we use a Linux workstation with an Intel 8-core Xeon CPU running at 2.1GHz. ED-DTA runtimes range from 9.4s for s344 up-to 8019s for multiplier.32b. Note that the proposed ED-DTA tool has been developed for single CPU execution, since the primary purpose of this paper is to unveil the unexploited DTS ignored by GB-DTA methodds, rather than to improve DTA performance. However, ED-DTA runtime can be improved by up-to three orders of magnitude by utilizing Graphics Processing Units (GPUs) to exploit the available parallelism [19].

V. Conclusions

In this paper, we presented a framework to unveil the pessimism in DTS estimation imposed by conventional GB-DTA methods which inherently rely on worst-case assumptions. To achieve this, we developed an accurate ED-DTA tool that considers the actual data-dependent path delays. Experimental results show that our ED-DTA approach unveils 2.35% on average and up-to 194.51% more DTS, compared to GB-DTA. When only the critical activated paths are considered, the average DTS improvement is increased to 11.2%. We also demonstrated that the proposed approach enables further increase of the clock frequency by up-to 10.42% compared to existing frequency scaling schemes, and reveals that timing failures can be up-to 2.94× less than the ones estimated by existing failure estimation techniques.

Acknowledgments

This work is supported by the European Community Horizon 2020 programme under grant no. 688540 (UniServer).

References

[1] P. Gupta *et al*, "Underdesigned and opportunistic computing in presence of hardware variability," *TCAD*, vol. 32, 2013.
[2] G. Karakonstantis *et al.*, "Containing the nanometer "pandora-box": Cross-layer design techniques for variation aware low power systems," *IEEE JETCAS.*, vol. 1, 2011.
[3] J. Bhasker *et al.*, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, New York, USA: Springer, 2009.
[4] D. Bull *et al.*, "A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation," *JSSC*, vol. 46, 2011.
[5] A. Rahimi *et al.*, "Application-adaptive guardbanding to mitigate static and dynamic variability," *Transactions on Computers*, vol. 63, 2014.
[6] J. Constantin *et al.*, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *DATE*, 2015.
[7] X. Jiao *et al.*, "Clim: A cross-level workload-aware timing error prediction model for functional units," *Transactions on Computers*, vol. 67, 2018.
[8] J. Xin *et al.*, "Identifying and predicting timing-critical instructions to boost timing speculation," in *MICRO*, 2011.
[9] H. Cherupalli *et al.*, "Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems," in *ISCA*, 2016.
[10] I. Tsiokanos *et al.*, "Low-power variation-aware cores based on dynamic data-dependent bitwidth truncation," in *DATE*, 2019.
[11] *ModelSim: https://www.mentor.com/products/fv/modelsim/.*
[12] H. Cherupalli *et al.*, "Scalable n-worst algorithms for dynamic timing and activity analysis," in *ICCAD*, 2017.
[13] A. Krstic *et al.*, "Pattern generation for delay testing and dynamic timing analysis considering power-supply noise effects," *TCAD*, vol. 20, 2001.
[14] R. G. Dreslinski *et al.*, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, 2010.
[15] C. Kalonakis *et al.*, "Tktimer: fast & accurate clock network pessimism removal," in *ICCAD*, 2014.
[16] D. Garyfallou *et al.*, "A sparsity-aware MOR methodology for fast and accurate timing analysis of VLSI interconnects," in *SMACD*, 2019.
[17] M. C. Hansen *et al.*, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test*, vol. 16, 1999.
[18] A. Yazdanbakhsh *et al.*, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, 2017.
[19] E. Schneider *et al.*, "Multi-level timing simulation on gpus," in *ASP-DAC*, 2018.