Find the Real Speed Limit: FPGA CAD for Chip-Specific Application Delay Measurement

Ibrahim Ahmed, Shuze Zhao, Olivier Trescases and Vaughn Betz
Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
Email:{ibrahim, szhao, ot, vaughn}@ece.utoronto.ca

***Abstract*—Process variation is increasing with each successive technology node, and it has reached the point where the worst-case timing modelling employed by current FPGA CAD tools is significantly underutilizing the available silicon. Previous studies have proposed exploiting FPGA reconfigurability to reduce this underutilization using techniques such as late binding and dynamic voltage scaling. Most of the proposed solutions require the ability to measure the target application's delay on each configured chip. To accurately measure the delay of an application on a certain chip, we must measure the delay of its speed limiting paths on this specific chip. In this paper, we present a variation-aware CAD tool that automatically generates calibration bitstreams to measure the delay of any input application. Our tool identifies the statistically critical paths of the circuit and optimally selects which paths to test such that it minimizes the chances of reporting an optimistic delay, under a constraint on the number of allowed calibration bitstreams. Experimental results across a suite of benchmarks show that with one calibration bitstream we achieve 16× lower probability of reporting an optimistic delay compared to a greedy approach. With three calibration bitstreams, we reduce the probability of optimism to two chips in a million, approximately 6,000× lower than a greedy approach.**

I. Introduction

Technology scaling has been the cornerstone for the continuous growth of the semiconductor industry; following Moore's law [1], the number of transistors on a chip has been doubling every 2-3 years. However, fully utilizing these devices is getting more challenging, as scaling also increases device variations. As we shrink feature size, it becomes infeasible to fabricate identical elements; fundamental lithography limitations result in variation of key transistor attributes like gate length. Other sources of variation include oxide thickness fluctuations, in-consistent dopant concentrations, and stress variation [2]. One major effect of process variation is that identically designed devices have different delay after fabrication. Computer-aided design (CAD) tools deal with this variability by modelling each device as having the worst process variation and being operated at the worst temperature and voltage IR-drop. While this model guarantees successful operation of the designed circuit, it underutilizes the fabricated silicon. The effects of worst-case modelling can be more severe for FPGAs [3], where the circuit critical paths are not known by the manufacturer. For a modern FPGA, [4] reports a 38.9% average frequency increase compared to the reported CAD tool maximum frequency (F_{max}) across a suite of benchmarks.

Researchers have proposed exploiting FPGA reconfigurability to reduce the impact of CAD tool pessimism. To get back some of the lost speed, [5] and [6] propose generating multiple bitstreams for each application, measuring the actual delay of each bitstream and selecting the fastest bitstream for each chip. Using this strategy, [6] reports a 34.8% improvement in the critical path delay compared to worst-case design. [7] and [8] report up to 54% and 33% power reduction by using delay measurements to find the minimum supply voltage (V_{dd}) that can safely run the desired application on each target chip. While the studies in [5]–[8] differ in their objectives, they all require the ability to measure the delay of any application on each configured chip.

Ahmed et al. proposed using an off-line calibration approach to measure the delay of an application [8]. They extract the top critical paths of the application reported by static timing analysis (STA), select and replicate as many of these paths as possible, and measure their actual on-chip delay. However, the authors of [8] left some questions unanswered: how to choose which paths to test, how to assess the risk of underestimating the application delay in each programmed chip, and how to extend the procedure to allow for multiple calibration bitstreams. To tackle these issues, we enhanced the FRoC CAD tool initially presented in [8]; our enhanced FRoC (iFRoC) connects to a commercial FPGA CAD tool (*Quartus*) and uses the flow shown in Fig. 1 to measure any application's delay. iFRoC utilizes the STA engine of *Quartus* to extract a set (P_{cand}) of potentially speed limiting paths and detailed timing information on each circuit element used by paths in P_{cand}. iFRoC then models process variation, identifies the statistically critical paths and generates one or more calibration designs to measure the delay of these paths. Our novel contributions in this work include:

- Identifying which paths are more important to test in the presence of process variation.
- Proposing a new path-selection algorithm that minimizes the probability of measuring an optimistic application delay[1].
- Quantifying the quality of results from our proposed algorithm across a large suite of benchmarks, with a range of process variation models.

The remainder of this paper is organized as follows: Section II discusses related work and gives a brief overview of the off-line calibration approach presented in [8]. The proposed variation model used in our experiments is presented in Section III. Section IV discusses our novel algorithm to select paths

[1]Under a constraint on the number of calibration bitstreams.

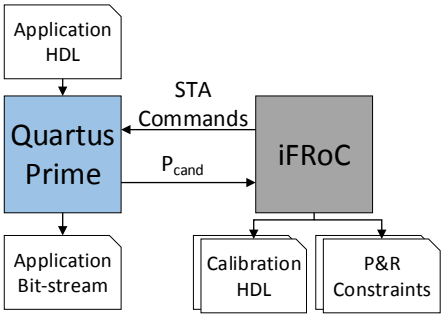

Fig. 1: Proposed flow to measure an application delay.

for testing and Section V presents our experimental set-up and the achieved results. We conclude the paper in Section VI.

II. Related Work and Background

We start by defining the following terms:

Application delay[2]: The minimum clock period that guarantees an error free operation of the application on a specific programmed chip.

STA reported critical path (CP): The path with the lowest slack computed by the standard worst-case timing models.

Actual CP: The path with the measured lowest slack on a specific programmed chip.

$\boldsymbol{P_{cand}}$: The set of top critical paths reported by STA that potentially includes all actual CPs.

Each application has one STA reported CP but could have different actual CPs on different chips or under different operating conditions. To measure the application delay on a certain chip, we need to measure the delay of the actual CP on that chip. Ideally, P_{cand} should include all paths that could limit speed in *any* chip (all actual CPs). We explore how large P_{cand} needs to be in Section V.

A. Related Work

Wong et al. in [9] attempt to measure the delay of combinational circuits by driving the primary inputs (PIs) of the circuit using a test stimuli generator and observing the primary outputs (POs) for failures. To determine the delay of the circuit, the authors incrementally increase the clock frequency until a failure is detected in any of the POs. A major limitation of this approach is that it only works for combinational circuits. Moreover, as the circuit gets more complex, some paths may not be sensitized by finite test stimuli applied to the PIs. In [10], the authors propose using the transition probability (TP) of the circuit's POs to determine its delay. They drive the PIs with a test generator and observe the TP of the POs as they increase the clock frequency. While the approach in [10] is able to test sequential circuits, it does not guarantee that the actual CP of the application is exercised and so the measured delay could be optimistic. Complex sequential circuits usually have buried internal states that are hard to reach by only stimulating the circuit PIs. For example, a circuit with a simple 64-bit counter needs 2^{64} cycles just to reach its final state.

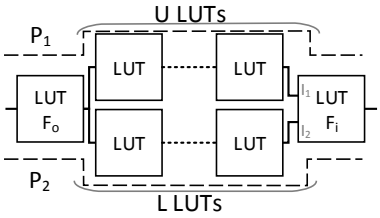

Fig. 2: Re-convergent fan-out example.

The work presented in [11]–[13] targets delay fault testing and so it cannot be directly used for delay measurement. Since the testing scheme in [11] is not capable of testing inverting transitions across a look-up table (LUT), it is not suitable for measuring the exact delay of an application. Similar to [8], the work in [12] extracts the reported top critical paths and tests them to check for delay faults. However, their testing algorithm does not scale well with LUT size and it does not fully utilize the reconfigurability of FPGAs, which results in $\sim$ 1000 bitstreams being required to test the extracted paths of an application.

B. Off-line Calibration

The testing algorithm proposed in [8] takes a set of selected paths (P_{selec}) that can be tested in one bitstream and generates a calibration design that measures the delay of these paths. To create this calibration design, the exact resources (LUTs, registers, wires, etc.) used by the selected paths are replicated and either 1 or 2 control signals are added to each LUT being tested. Since a LUT delay is almost independent of its mask, all LUT masks are changed to guarantee sensitization of all tested paths. The LUT mask[3] used is:

$$F = Fix \bullet (I_1 \oplus I_2 ... \oplus I_{K-2} \oplus Edge) + \overline{Fix} \bullet Edge \quad (1)$$

where Fix and $Edge$ are the control signals that fix the LUT output to a constant value and select the edge transition (inverting or non-inverting) across the LUT, respectively. Fixing the LUT's output is important to guarantee that all off-path inputs of any path being tested are constant. K is the number of LUT inputs (4 to 6 for modern FPGAs) and I_i is the i^{th} LUT input. I_1 to I_{K-2} are the *available* inputs that form the paths in P_{selec}. All LUTs have an $Edge$ control signal, but Fix is only added to a LUT if it is necessary to fix its output. Fix is required when any LUT fan-out has more than 1 input connected to paths in P_{selec}. After replicating the selected paths, they are grouped into test phases, where each test phase includes paths that can be tested simultaneously. Finally, a test controller that provides the input test stimulus and required control signals is generated. To identify the circuit delay, the test controller sets the clock initially to a safe low frequency and increments the frequency until an error is detected.

Ideally, all paths in P_{cand} should be selected for testing in a single calibration bitstream (P_{selec}=P_{cand}). However, as

[2]For single-clock circuits, but iFRoC is not limited to single-clock designs.

[3]See [8] for additional masks to test carry-chains.

detailed in [8], some paths cannot be tested in the same bitstream. The most common reason that prevents testing all paths in the same bitstream is exceeding the available LUT inputs; since extra control signals are added to all tested LUTs, LUTs must have spare inputs that are not used by any tested path. Hence, paths in P_{selec} cannot use all inputs of any LUT.

Re-convergent fan-out also constrains paths in P_{selec}. It occurs when one node (divergence point) branches out to multiple fan-outs and then converges to another node (convergence point). Fig. 2 shows a general example of a re-convergent fan-out connection in which P_1 and P_2 pass through U and L LUTs between the divergence and convergence points. This type of connection usually causes problems in testing [14]. However, since the LUT mask in (1) can fix the output of any LUT, it is able to break most re-convergent fan-outs except one: direct re-convergent fan-out in which there is a direct connection with no LUTs between the divergence and convergence point. For example, if U or L in Fig. 2 is zero, it is impossible to fix the off-path input of P_2 or P_1 at LUT F_i, so we cannot test P_1 and P_2 in one bitstream.

In [8], P_{selec} is generated by extracting the reported top critical paths (P_{cand}) of the application and greedily removing paths to reach a state where all paths can be tested in a single bitstream. In this paper, we present an improved algorithm for selecting which paths to test in a bitstream and we extend our algorithm to optimize for multiple calibration bitstreams.

III. Modelling Variation

Ignoring process variation, we could measure the delay of an application by just measuring the delay of its STA reported CP, which means that P_{cand} and P_{selec} would only include the path with the lowest slack reported by the STA. Unfortunately, process variation cannot be avoided and is worsening with scaling. For this reason, when performing path-based delay testing it is necessary to test many paths with delay close to the STA reported CP delay [8], [13]. However, many of these paths largely overlap and share many resources, and so testing them all may not be necessary to get an accurate application delay. To identify which paths must be tested to safely determine an application's delay, we must build a model for process variation.

While the same FPGA circuit element (LUT, wire, etc.) could have different delays, *Quartus* STA only uses the worst-case delay for each element type. In practice, FPGA manufacturers set the "worst-case delay" statistically such that the probability of an optimistic static timing analysis is very low; they are not guaranteed upper bounds on individual circuit element delays. Hence, we developed a reasonable statistical model, whose "worst-case delay" matches that of *Quartus*. iFRoC builds a normally distributed random variable (RV) to represent each element delay. The mean (μ_i) and standard deviation (σ_i) of the random variable representing the i^{th} circuit element (X_i) are calculated using:

$$\begin{aligned} \sigma_i &= var \times \mu_i \\ Qdelay_i &= \mu_i + yld \times \sigma_i \end{aligned} \tag{2}$$

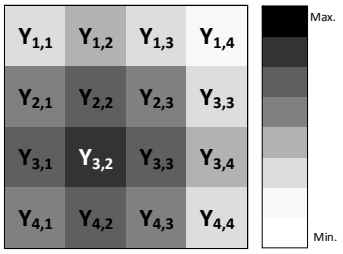

Fig. 3: An FPGA grid model and a color map of the spatial correlation between elements in grid (3,2) and all other elements.

where $Qdelay_i$ is the delay reported by *Quartus* for the i^{th} circuit element, var and yld are variables that represent the amount of process variation and how pessimistic *Quartus* delay models are, respectively.

A. Spatial Correlation

The assumption that the delay of circuit elements are independent can underestimate the delay of a path [15]. In reality, physical parameters that affect the device delay are spatially correlated. For example, gate length variations in neighbouring transistors are usually similar and so their delay variations are correlated.

Previous studies in [16] and [17] have presented two approaches to model spatial correlation: a grid based model with principal-component-analysis (PCA) and a quadtree model. Both approaches model the underlying physical device parameters as normally distributed random variables and compute each circuit element delay based on its sensitivity to the modelled physical parameter. This sensitivity is usually found through SPICE simulations. The work presented in [16] needs measurements to compute the covariance matrix for each physical parameter and the work in [17] has the limitation that neighbours with the same proximity might have different correlations.

Neither of the proposed models can be directly used in our situation as we have no SPICE models for the actual silicon used in our target FPGA chip. Moreover, we already know the "worst-case delay" of each circuit element from *Quartus* which we want our RVs to match. We developed a model that ensures that each X_i has the mean and standard deviation computed by (2) and is spatially correlated to its neighbouring circuit elements. To build our model, we divide the chip into a grid as shown in Fig. 3. Each grid (x, y) has an associated independent normally distributed RV ($Y_{x,y}$) with 0 mean and unit variance. Based on its physical location, we map each X_i to an (x, y) grid and compute it as:

$$\begin{aligned} X_i &= \mu_i + \frac{\sigma_i(\sum_{k=K_s}^{K_e}\sum_{j=J_s}^{J_e}\alpha_{i,j,k}Y_{j,k} + \beta_i\epsilon_i)}{\sqrt{\sum_{k=K_s}^{K_e}\sum_{j=J_s}^{J_e}\alpha_{i,j,k}^2 + \beta_i^2}} \\ K_s &= y - V, K_e = y + V, J_s = x - H, J_e = x + H \end{aligned} \tag{3}$$

where μ_i and σ_i are the mean and standard deviation of X_i calculated from (2). V and H are constants that determine the size of the bounding box that will be used to compute X_i. ϵ_i is an independent normally distributed RV with 0 mean and a unit variance representing the independent random variation of X_i. $\alpha_{i,j,k}$ and β_i are sensitivity constants that can be used to fit our model to measured data.

As we do not have the resources to measure delays across thousands of chips to get actual data to fit our model, we chose values for our sensitivity constants such that we achieve a reasonable spatial correlation model. We set all sensitivity constants to 1 and set the bounding box to be a 3×3 box, so H and V are also set to 1. With these values, Fig. 3 shows a color map of the relative correlation between a LUT in grid (3,2) and LUTs in all other grids. The correlation to LUTs in the same grid is the highest, and it drops as we go to grids that are further away, with minimum correlation to LUTs in grid (1,4). This model has the attractive feature that correlation smoothly increases as the distance between elements decreases, unlike the quadtree model where neighbours across boundaries are less correlated. This is specially important for FPGAs in which carry chains are usually packed in neighbouring grids, so it is important to capture correlation between them.

Building an accurate process variation model and validating it through physical measurements is a cumbersome task. We are building this model to compare the delay of paths against each other and, unlike statistical static timing analysis (SSTA), we are not concerned with the absolute delay of any path (as we get this information from on-chip measurement). For this reason, we can tolerate more inaccuracy in the variation model than SSTA engines. We have also varied the parameters of our delay variation model and found that our results are not very sensitive to the exact parameters chosen except for var; we explore the impact of this parameter in Section V.

IV. Monte Carlo Based Paths Selection

As explained in Section II, testing all paths in P_{cand} in the same calibration bitstream is sometimes not possible, due to testing constraints, therefore we want to identify which paths are more important to be selected for testing. For this purpose, we define *path statistical criticality* as the probability of a path being the actual CP of the application. To obtain each path's statistical criticality, iFRoC: 1) models the part of the application comprising P_{cand} using our proposed process variation model, 2) performs a Monte Carlo (MC) simulation over all possible chip delays by randomly sampling from our delay distribution model and 3) counts the number of MC samples (chips) at which each path was the actual CP. The statistical criticality of each path is then computed by normalising this number to the total number of MC samples and is stored in a vector C. We also define the set of paths with non-zero statistical criticality as the set of important paths (P_{imp}).

A. Integer Linear Programming

Our goal is to minimize the probability of reporting an optimistic application delay; this happens when we miss testing the application's actual CP. Formally, we define the probability of reporting an optimistic application delay (Prob$_{fail}$) as:

$$Prob_{fail} = \sum_{i \in P_{untst}} C_i \tag{4}$$

where P_{untst} is the subset of paths from P_{cand} that are not tested in any calibration bitstream and C_i is the statistical criticality of the i^{th} path.

We define the *path selection* as the task of choosing a legal set of paths ($P_{selec,n}$) to test in the n^{th} calibration bitstream. To minimize (4) under the constraint of using N calibration bitstreams, we formulate the path selection task as an integer linear programming (ILP) problem. As we use more calibration bitstreams, test time and the capacity of the flash memory needed to store these bitstreams increase, so the acceptable number of calibration bitstreams may vary depending on the test time and flash memory budget. Our ILP formulation accepts the required number of calibration bitstreams (N) as an input, and it optimally selects which paths to test in each calibration bitstream such that it minimizes (4) over the N bitstreams. The main variables created for our ILP are:

1) P_i^n : a boolean variable that represents the i^{th} path from P_{cand}. If the i^{th} path is selected to be tested in the n^{th} calibration bitstream, then P_i^n = 1, otherwise P_i^n = 0. N variables are created for each path in P_{cand}.
2) $L_{x,y}^n$: a boolean variable that represents the x^{th} input of LUT y. If this input is connected to any of the tested paths in the n^{th} calibration bitstream then $L_{x,y}^n$ = 1, otherwise $L_{x,y}^n$ = 0. N variables are created for each LUT input used by any path in P_{cand}.

Intuitively, to minimize (4) we should select paths with higher statistical criticality and so we developed the following ILP objective function:

$$\begin{aligned} &\textbf{maximize} \quad \sum_{n=1}^{N} \sum_{i=1}^{|P_{cand}|} \hat{C}_i P_i^n \\ &\hat{C}_i = \begin{cases} C_i \times MC_{smpl}, & \text{if } C_i > 0 \\ \frac{1}{|P_{cand}|}, & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

where MC_{smpl} is the number of MC samples. We use $\hat{C}_i$ to weight paths instead of C_i so that paths that were never the actual CP in any MC sample are still given a small weight. If there are multiple solutions that would give the same Prob$_{fail}$, our objective function chooses the one that tests more paths with 0 statistical criticality. By setting the $\hat{C}_i$ of paths with $C_i = 0$ to $\frac{1}{|P_{cand}|}$, we guarantee that the summation of all the $\hat{C}_i$ of non-statistically critical paths is less than 1 and so they are less important than any path with a non-zero statistical criticality.

The objective function in (5) is subject to multiple sets of constraints. The first set of constraints ensures that we do not

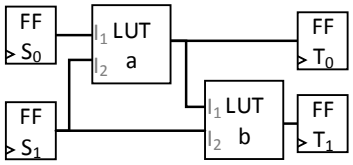

Fig. 4: A circuit with a direc re-convergent fan-out.

TABLE I: Paths from Fig. 4 in P_{cand}.

Path	Elements
P_1	S_1, b, T_1
P_2	S_1, a, b, T_1
P_3	S_0, a, b, T_1
P_4	S_1, a, T_0
P_5	S_0, a, T_0

test the same path in multiple bitstreams, as this would lead to over counting in (5), and is defined as:

$$\sum_{n=1}^{N} P_i^n \leq 1, \quad \forall i \in \{r \in \mathbb{N} : r \leq |P_{cand}|\} \tag{6}$$

The number and time complexity of generating the constraints given in (6) is $O(|P_{cand}| \times N)$.

The second set of constraints relates the P_i^n variables to the $L_{x,y}^n$ variables, by ensuring that P_i^n can only be tested if all LUT inputs used by P_i^n are connected to tested paths in the n^{th} bitstream. For the i^{th} path in the n^{th} bitstream, the constraints are given as:

$$P_i^n \leq L_{x,y}^n, \quad \forall x, y \in LI_i \tag{7}$$

where LI_i is a set of pairs storing each LUT and LUT input used by P_i^n. The number and generation time complexity of the constraints given in (7) is $O(|P_{cand}| \times N \times L_{path})$, where L_{path} is the number of LUTs per path.

We transformed the testing constraints that prevent certain paths from being tested in the same calibration bitstream, discussed in Section II-B, into linear inequalities and added them to the ILP formulation. To avoid exceeding the available LUT inputs, we must ensure that the number of LUT inputs connected to tested paths plus the required control signals is less than the LUT inputs. If we need to fix the output of a LUT then we need to add 2 control signals to this LUT, otherwise only 1 control signal is added. For each LUT y in the n^{th} bitstream, the constraints are:

$$\begin{gathered} \sum_{x=1}^{K} L_{x,y}^n \leq K - 1 - Fix_y^n \\ Fix_y^n \geq L_{q,t}^n, \forall t \in F_{out_y}, \forall q \in \{x \in \mathbb{N} : x \leq K\} \setminus \{r_t\} \end{gathered} \tag{8}$$

where K is the LUT size, F_{out_y} is the set of tested LUTs connected to the output of LUT y and r_t is the input of LUT t that is connected to the output of LUT y. From (8), Fix_y is 1 if LUT y requires 2 control signals and is 0 otherwise. LUT y requires 2 control signals if one of its fan-outs feeds a LUT with more than 1 input connected to tested paths. The number of constraints given in (8) and the time complexity to generate them is $O(K \times F_{out_{LUT}} \times Z \times N)$, where Z is the number of LUTs used by all paths in P_{cand} and $F_{out_{LUT}}$ is the number of fan-outs per LUT.

The last set of constraints deals with the direct re-convergent fan-out case (explained in Section II-B), where paths that form a direct re-convergent fan-out cannot be tested in the same bitstream. While detecting general re-convergent fan-outs has been studied in previous work [14], our problem is unique as we want to detect only the direct re-convergent fan-out in the union of the paths in P_{cand}. Fig. 4 shows an example of a circuit with a direct re-convergent fan-out between P_1 and P_2 at LUT b, and Table I lists the paths in P_{cand} for this circuit and the circuit elements used by each path. To understand how iFRoC detects this direct re-convergent fan-out, we make the following definitions: $P_{x,y}^n$ is the set of boolean variables from the n^{th} calibration bitstream that represents the paths in P_{cand} using LUT y through input x, and $PP_{z,y}^n$ is the set of boolean variables from the n^{th} calibration bitstream that represents the paths in P_{cand} using the LUT or register feeding input z of LUT y. To fix the off-path input (z) at LUT y for paths in $P_{x,y}^n$, we must fix the output of the LUT or register feeding input z of LUT y. Fixing this output means that all paths in $PP_{z,y}^n$ cannot be tested while paths in $P_{x,y}^n$ are being tested. If the same path exists in these two sets, then we cannot test this path as we will not be able to fix its off-path inputs. This only occurs in the presence of a direct re-convergent fan-out. For example in Fig. 4, due to the direct re-convergent fan-out between P_1 and P_2, we get $P_{I_1,b}^n = \{P_2^n, P_3^n\}$ and $PP_{I_2,b}^n = \{P_1^n, P_2^n, P_4^n\}$, so we cannot test P_2 and fix input I_2 of LUT b at the same time, in the n^{th} bitstream. Formally, iFRoC loops over all LUTs and for each LUT y in the n^{th} bitstream it computes $P_{inter_{x,z}}^{y,n}$ as:

$$P_{inter_{x,z}}^{y,n} = P_{x,y}^n \cap PP_{z,y}^n \quad \forall x, z \in \{r \in \mathbb{N} : r \leq K\}, x \neq z \tag{9}$$

If $\exists x \exists z (P_{inter_{x,z}}^{y,n} \neq \emptyset)$ then there is a direct re-convergent fan-out at LUT y between paths in $P_{inter_{x,z}}^{y,n}$ and paths in $P_{z,y}^n$. This means that paths from these two sets are mutually exclusive and cannot be selected in the same $P_{selec,n}$. iFRoC generates the third set of ILP constraints as:

$$P_{inter_{x,z}}^{y,n} \oplus P_{z,y}^n \leq 1, \quad \forall x, z \in \{r \in \mathbb{N} : r \leq K\}, x \neq z \tag{10}$$

where $\oplus$ is the mutual exclusion operator and is defined as:

$$\begin{gathered} G_x \oplus G_y = T_x + T_y \\ T_x \geq G_{x,k} \quad \forall k \in \{r \in \mathbb{N} \ : \ r \leq |G_x|\} \\ T_y \geq G_{y,k} \quad \forall k \in \{r \in \mathbb{N} \ : \ r \leq |G_y|\} \end{gathered} \tag{11}$$

where G_i is a set of boolean ILP variables, $G_{i,j}$ is the j^{th} variable in the G_i set, $|G_i|$ is the size of G_i and T_i is a boolean ILP variable.

In Fig. 4, since $P_{inter_{I_1,I_2}}^{b,n} = \{P_2^n\}$ and $P_{I_2,b}^n = \{P_1^n\}$, the constraint in (10) ensures that iFRoC never selects them both for testing in the same bitstream. The number of constraints given in (10) and the time complexity to generate them is $O(K^2 \times Z \times P_{LUT} \times N)$, where P_{LUT} is the number of paths per LUT.

V. Experimental Results

To evaluate iFRoC, we created a benchmark suite of circuits with varying sizes and characterstics. We used the 20 largest MCNC benchmarks (MCNC20) [18], registered their I/Os, and

TABLE II: Benchmark information.

Circuit	LE utilization	F_{max} (MHz)	$\|P_{cand}\|$	$\|P_{import}\|$
alu4_m	39194(34%)	135.7	802	436
apex2	1054	129.7	590	102
diffeq	888	144.0	884	42
elliptic_l	82459 (72%)	109.7	3415	436
elliptic_m	40592 (35%)	122.8	24766	1275
elliptic	1986	141.9	1865	101
FIR	67958 (59%)	114.9	12832	625
frisc_l	85490 (75%)	107.7	13803	817
frisc_m	38730 (34%)	101.6	2112	97
frisc	2278	114.1	6716	122
spla_m	38971 (34%)	121.5	776	335
stereovision0	12059 (11%)	196.7	251	33
stereovision1	22938 (20%)	172.0	1418	398
stereovision2	35530 (31%)	117.9	32581	368
Xbar	30248 (26%)	125.4	1102	232
Geomean			2596	228.2

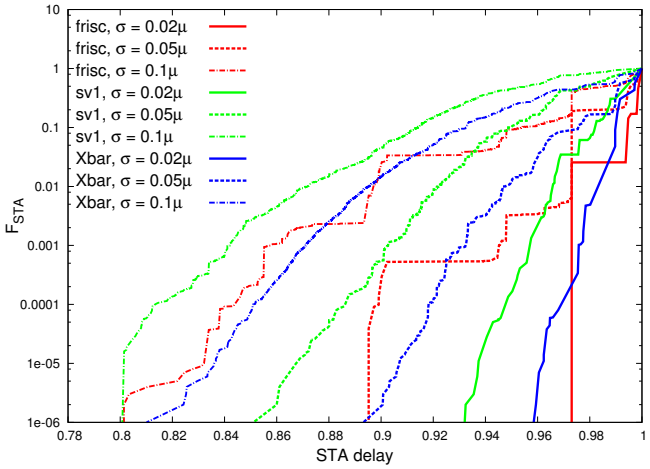

Fig. 5: $F_{sta}(x)$ of multiple benchmarks at different var and using yld = 2.

for each MCNC20 benchmark we created a medium and large sized circuit by replicating it multiple times to fill $\sim$ 35% and 75% of the chip, respectively. We also used three of the VTR [19] benchmarks synthesized without using BRAMs or DSPs since iFRoC does not yet support testing these resources. We also included a dual-channel 51-tap low-pass FIR filter from the *Quartus* IP catalogue and a full crossbar (Xbar) with 16 100-bit-wide ports. We only show a representative subset of the MCNC20 benchmarks as the rest show similar behaviour. The first three columns of Table II show the selected benchmarks along with their F_{max} and logic element (LE) utilization, where *_m and *_l are the medium and large replicated MCNC20 benchmarks.

Our experiments targeted a Cyclone IV EP4CE115F29C7 FPGA manufactured using TSMC 60-nm technology. For process variation, we used our model presented in Section III and divided the chip into a 114×73 grid such that each grid cell contains one logic array block (LAB). iFRoC uses the *Gurobi* software [20] to solve the ILP problem.

A. Candidate Paths Exploration

To determine how many paths should be extracted from the application and added to the set of candidate paths (P_{cand}), we define the random variable X_{sta} as the *Quartus* delay of the actual CP of the application normalized to the STA reported CP delay (delay of the most critical path reported by *Quartus*). The cumulative distribution function (CDF) $F_{STA}(x)$ of X_{STA} represents the probability of paths with normalized STA reported delay less than x becoming the actual CP.

Since we do not know the real values of var and yld in (2), we explored the space by selecting a range from optimistic to pessimistic values for each variable. For var, we used $\{0.01, 0.02, 0.05, 0.1\}$ representing a delay standard deviation from 1% to 10% of the mean. For yld, we used $\{1, 2, 3\}$ indicating that the delay for each circuit element used by *Quartus* varies from $\mu + 1\sigma$ to $\mu + 3\sigma$. Fig. 5 shows $F_{STA}(x)$ for three of our benchmarks for various var when $yld = 2$. For all our benchmarks when $var = 0.01$, $F_{STA}(x)$ is zero except at $x = 1$, which means that the STA reported CP has a statistical criticality of 1 and no other path is ever the actual CP. This implies that if the standard deviation is only 1% of the mean, it is enough to only add the STA reported CP to P_{cand}.

As expected, Fig. 5 shows that as σ increases, the probability of paths with lower STA reported delay becoming the actual CP also increases. To get an F_{STA} of 0.001 for the $frisc$ benchmark, meaning that there is less than 0.1% chance that P_{cand} does not include the actual CP, we should extract paths with delay higher than $\sim$ 97% of the STA reported CP delay if the standard deviation is 2%. However, if the standard deviation is 10%, we should extract paths with delay higher than $\sim$ 85% of the STA reported CP delay to get the same F_{STA}. The work presented in [21] measured an average delay standard deviation of 4% on a 65-nm FPGA and the work in [22] reported a standard deviation of $\sim$ 3% for 45-nm technology. Based on these values we assume that a standard deviation of 5% is a reasonable value and unless otherwise stated, the σ used in the remaining results is 5%.

Although yld has a significant impact on the absolute delay of paths, it has almost no effect on $F_{STA}(x)$. Since $F_{STA}(x)$ is only concerned with the relative delay of paths to identify the actual CP, it is not affected by yld. For this reason, we fix yld to a value of 2 in our remaining experiments. The true value of yld is proprietary information that is only known by the creators of the *Quartus* timing model, but a value of 2 is in-line with common practice. A low value of yld would allow *Quartus* to report higher F_{max} at the expense of lower yield, while a higher value would increase the yield but reduce the reported F_{max}.

Across our suite of benchmarks, extracting paths with delay higher than 90% of the STA reported CP delay guarantees that F_{STA} is at most 5×10^{-4} when σ is 5%. However, our results show that as the delay variation increases we need to extract more paths. Since we are assuming a standard deviation of 5%, we include all paths with delay higher than 90% of the

TABLE III: Prob$_{fail}$ and $|P_{test}|$ when using different number of calibration bitstreams.

	1 bitstream				2 bitstreams				3 bitstreams			
	Greedy		ILP+MC		Greedy		ILP+MC		Greedy		ILP+MC	
Circuit	Prob$_{fail}$	$\|P_{test}\|$	Prob$_{fail}$	$\|P_{test}\|$	Prob$_{fail}$	$\|P_{test}\|$	Prob$_{fail}$	$\|P_{test}\|$	Prob$_{fail}$	$\|P_{test}\|$	Prob$_{fail}$	$\|P_{test}\|$
alu4_m	0.12	667	0.00034	619	0.0027	792	0	782	0	802	0	802
apex2	0.45	249	0.0144	228	0.055	408	0.00019	233	0.041	495	0	488
diffeq	2.9e-05	473	0	456	0	741	0	741	0	834	0	834
elliptic_l	0.0041	1872	0.0018	1812	0.00066	2420	1e-06	2326	0.00017	2681	0	2612
elliptic_m	0.017	14174	0.0044	13502	0.0037	20095	0.00067	18883	0.0005	22876	0	21775
elliptic	0.13	586	0.038	445	0.13	939	0.00023	602	0.093	1190	1.4e-05	883
FIR	0.053	8449	0.017	7801	0.038	10750	0.00047	10111	0.034	11598	1.2e-05	11182
frisc_l	0.025	7771	0.00039	7281	0.015	10350	0	9891	2e-05	11980	0	11642
frisc_m	0.00097	951	2.1e-05	573	1.5e-05	1317	0	1136	1.5e-05	1633	0	1452
frisc	0.0064	2859	6.7e-05	2326	0.0015	4127	0	4050	0.0015	5261	0	5299
spla_m	0.32	653	0.0012	623	0.00014	764	1.1e-05	752	0	776	0	775
stereovision0	3.4e-05	193	3.4e-05	204	0	251	0	251	0	251	0	251
stereovision1	0.143	1148	0.021	1130	0.022	1402	0.0019	1354	0	1418	0	1418
stereovision2	0.176	17796	0.003	16366	0.0035	26678	8.6e-05	23663	2.3e-05	30364	0	29034
Xbar	0.25	884	0	858	0.25	1069	0	1066	0	1100	0	1102
Arithmetic mean	0.11	3915	0.0068	3615	0.034	5474	0.00024	5056	0.011	6217	1.7e-06	5969

STA reported CP delay in P_{cand}.

B. Probability of Failure

The last two columns of Table II show the sizes of P_{cand} and P_{import}, where P_{import} is the set of paths with non-zero statistical criticality, i.e. the paths that determine the application delay in *some* chip. These values show that only a small subset of paths in P_{cand} are important to measure; the geomean size of P_{import} is $\sim 12.5\times$ smaller than $|P_{cand}|$. The reduction is even higher for benchmarks with larger $|P_{cand}|$: $|P_{import}|$ is roughly two orders of magnitude smaller than $|P_{cand}|$ for the circuit with largest $|P_{cand}|$ (*stereovision2*).

To quantify the importance of modelling variation and intelligently selecting which paths to test, we ran each benchmark through the flow in Fig. 1 twice. In the first run (Greedy), iFRoC was set to maximize the number of tested paths without knowledge of path statistical criticality or process variation. For the second run (ILP+MC), iFRoC used the variation model of Section III to compute the statistical criticality of paths by running a 10^6-sample MC simulation and used the proposed ILP formulation to prefer selecting paths with higher statistical criticality for testing. Table III presents a comparison between the Greedy and the ILP+MC run in terms of the Prob$_{fail}$ and the number of tested paths ($|P_{test}|$) when using different numbers of calibration bitstreams. Although, with 1 calibration bitstream, Greedy tests more paths, ILP+MC has a $16\times$ lower Prob$_{fail}$. This shows that while testing as many paths as possible is important, it is not enough to minimize the probability of underestimating the delay of an application. Moreover, Greedy can result in a significantly high Prob$_{fail}$: up to 0.25 and 0.45 for *apex2* and *Xbar* with 1 calibration bitstream, while ILP+MC resulted in 1.4e-2 and 0 for the same benchmarks. Another interesting observation is that for ILP+MC the Prob$_{fail}$ converges to 0 much faster as the number of calibration bitstreams increases. This convergence results in ILP+MC having a four orders of magnitude lower Prob$_{fail}$ with 3 bitstreams compared to Greedy.

Our proposed algorithm (ILP+MC) allows us to achieve significantly low Prob$_{fail}$ with just a few calibration bitstreams: with 3 calibration bitstreams, the average Prob$_{fail}$ is 1.7e-06 across all benchmarks. This means that if we used ILP+MC to measure the application delay across 1 million chips, we will underestimate the delay in only 2 chips. Even with only a single calibration bitstream, ILP+MC achieves a low average Prob$_{fail}$ of 0.68%. The Prob$_{fail}$ shows no trend with circuit size: the highest reported Prob$_{fail}$ is from one of the small benchmarks (*elliptic*) and the lowest Prob$_{fail}$ comes from a medium (*Xbar*) and a small (*diffeq*) sized circuit. The Prob$_{fail}$ depends on how much overlap exists between paths in P_{import}. If paths in P_{import} overlap in a way that prevents them from being tested in the same calibration bitstream, then Prob$_{fail}$ decreases as we cannot select all statistically critical paths for testing in a single calibration bitstream.

C. iFRoC Runtime Analysis

iFRoC runtime is dominated by the MC simulation and so the number of MC samples determines the total runtime. To quantify the effect of the number of MC samples on iFRoC runtime, we ran each benchmark through the flow in Fig. 1 using 10^4 and 10^6 samples for the MC simulation. Table IV shows iFRoC's total runtime with 1 calibration bitstream for each case. This total runtime includes the time to perform the MC simulation; formulate and solve the ILP problem; and generate all required files (HDL and constraints) for the calibration bitstreams. Table IV also presents the MC simulation time alone and as shown the geomean of the MC simulation time is $\sim 97\%$ and 99% of the total runtime for the 10^4- and 10^6-sample MC simulation, respectively. The time required to formulate and solve the ILP problem for most benchmarks is less than 10 seconds and the longest time is ~ 100 seconds for the $stereovision2$ circuit. This shows that our ILP formulation scales well with circuit size and $|P_{cand}|$.

To compare the Prob$_{fail}$ from the 10^4- and 10^6-sample MC simulation, we ran another 10^6-sample MC simulation (with a

TABLE IV: iFRoC runtime with 10^4- and 10^6-sample MC simulation.

	Runtime (sec) 10^4-sample MC sim.		Runtime (sec) 10^6-sample MC sim.	
Circuit	Total	MC	Total	MC
alu4_m	263.5	262.7	25050	25049.6
apex2	102.2	101.7	9900.2	9899.9
diffeq	23.2	22.5	2730.8	2729.92
elliptic_l	339.1	336.4	36986.7	36983.5
elliptic_m	864.7	839.3	99045.6	99003
elliptic	66	64.6	5610.8	5609.3
FIR	734.5	700.2	66764.5	66735.4
frisc_l	600	589.7	57583.1	57571.3
frisc_m	49.6	47.6	8810	8807.3
frisc	112	104	10195.5	10179
spla_m	232	231.1	22295.5	22294.6
stereovision0	16.3	16.1	1546.3	1546
stereovision1	144.9	143.6	13905.2	13904
stereovision2	760.8	657.7	61254.6	61160.8
Xbar	170.7	170	15445.1	15444.4
Geomean	166.5	161.6	16694	16688

different initial seed) and used this MC simulation to evaluate the Prob$_{fail}$ for the runs with 10^4 and 10^6 MC samples. For the MC simulation with 10^6 samples, the Prob$_{fail}$ was 0.68%, while it was 0.78% for the 10^4-sample MC simulation. This slight increase in Prob$_{fail}$ shows that even with 10^4 samples, we are able to identify the more important paths to test and thus achieve a low Prob$_{fail}$ with a geomean total runtime of 166.5 seconds, which is 100× smaller than using 10^6 MC samples.

VI. Conclusion

While the traditional worst-case delay modelling allows safe operation across process corners and operating conditions, it prevents us from utilizing the average device to its full potential. Previous works have measured a significant performance gap between what CAD tools report and the actual performance delivered by an FPGA chip [4], and researchers in [5], [8] have proposed interesting approaches (late binding and DVS) that exploit FPGA reconfigurability to shrink this gap. These approaches require measuring the delay of an application on its target chip. To realize this requirement, we developed iFRoC: a variation-aware CAD tool that is able to accurately measure the application delay on every programmed FPGA chip by creating calibration bitstreams that measure the on-chip delay of selected timing-critical paths from the application. iFRoC extracts timing information about the placed and routed circuit from the commercial *Quartus* CAD tool. iFRoC, then, models the delay of each circuit element as a distribution whose "worst-case delay" matches the *Quartus* reported delay. The variation model used by iFRoC accounts for spatial correlation and is parametrizable to allow it to be easily fitted to measured data. To calculate the statistical criticality of each path, iFRoC performs a MC simulation and identifies the actual CP at each MC sample. Using an ILP formulation iFRoC is able to optimally choose the most important, non-conflicting paths to test such that it minimizes the probability of not testing the actual CP of the application, given a certain number of calibration bitstreams. Experimental results show that across a suite of benchmarks iFRoC achieves an average Prob$_{fail}$ of 6.8e-03 and 1.7e-06 when using 1 and 3 calibration bitstreams, respectively, which is 16× and 6,000× lower than a greedy approach. Accordingly, iFRoC's delay calibration procedure enables DVS and late binding approaches with high reliability and low cost.

Acknowledgment

We would like to thank Mark Bourgeault and Gurvinder Tiwana for the insightful discussions. This work was funded by NSERC, Intel, OCE and SRC.

References

[1] "Progress in digital integrated electronics," in *1975 International Electron Devices Meeting*, vol. 21, 1975, pp. 11–13.
[2] K. Agarwal and S. Nassif, "Characterizing Process Variation in Nanometer CMOS," in *DAC*, June 2007, pp. 396–399.
[3] J. S. J. Wong *et al.*, "Self-Measurement of Combinatorial Circuit Delays in FPGAs," *TRETS*, vol. 2, no. 2, pp. 10:1–10:22, Jun. 2009.
[4] J. M. Levine *et al.*, "Dynamic Voltage & Frequency Scaling with Online Slack Measurement," in *FPGA*, 2014, pp. 65–74.
[5] Y. Matsumoto *et al.*, "Performance and Yield Enhancement of FPGAs with Within-die Variation Using Multiple Configurations," in *FPGA*, 2007, pp. 169–177.
[6] P. Sedcole and P. Y. K. Cheung, "Parametric Yield in FPGAs Due to Within-die Delay Variations: A Quantitative Analysis," in *FPGA*, 2007, pp. 178–187.
[7] C. T. Chow *et al.*, "Dynamic voltage scaling for commercial FPGAs," in *FPT*, Dec 2005, pp. 173–180.
[8] I. Ahmed *et al.*, "Measure twice and cut once: Robust dynamic voltage scaling for FPGAs," in *FPL*, Aug 2016, pp. 1–11.
[9] J. S. J. Wong *et al.*, "Self-Measurement of Combinatorial Circuit Delays in FPGAs," *TRETS*, vol. 2, no. 2, pp. 10:1–10:22, Jun. 2009.
[10] ——, "A transition probability based delay measurement method for arbitrary circuits on FPGAs," in *FPT*, Dec 2008, pp. 105–112.
[11] M. B. Tahoori and S. Mitra, "Application-Dependent Delay Testing of FPGAs," *TCAD*, vol. 26, no. 3, pp. 553–563, March 2007.
[12] P. R. Menon *et al.*, "Design-specific path delay testing in lookup-table-based FPGAs," *TCAD*, vol. 25, no. 5, pp. 867–877, May 2006.
[13] I. G. Harris *et al.*, "BIST-based delay path testing in FPGA architectures," in *ITC*, 2001, pp. 932–938.
[14] S. Xu and E. Edirisuriya, "A new way of detecting reconvergent fanout branch pairs in logic circuits," in *ATS*, Nov 2004, pp. 354–357.
[15] D. Blaauw *et al.*, "Statistical Timing Analysis: From Basic Principles to State of the Art," *TCAD*, vol. 27, no. 4, pp. 589–607, April 2008.
[16] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *TCAD*, vol. 24, no. 9, pp. 1467–1482, Sept 2005.
[17] A. Agarwal *et al.*, "Statistical timing analysis for intra-die process variations with spatial correlations," in *ICCAD*, Nov 2003, pp. 900–907.
[18] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide 3.0," MCNC, Tech. Rep., 01 1991.
[19] J. Luu *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *TRETS*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014.
[20] I. Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2016. [Online]. Available: http://www.gurobi.com
[21] B. Gojman *et al.*, "GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays Using Timing Extraction," *TRETS*, vol. 7, no. 4, pp. 32:1–32:23, Dec. 2014.
[22] P. Sedcole and P. Y. K. Cheung, "Within-die delay variability in 90nm FPGAs and beyond," in *FPT*, Dec 2006, pp. 97–104.