GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays Using Timing Extraction

BENJAMIN GOJMAN, University of Pennsylvania
SIRISHA NALMELA, Juniper Networks
NIKIL MEHTA, California Institute of Technology
NICHOLAS HOWARTH and ANDRÉ DEHON, University of Pennsylvania

Timing Extraction identifies the delay of fine-grained components within an FPGA. From these computed delays, the delay of any path can be calculated. Moreover, a comparison of the fine-grained delays allows a detailed understanding of the amount and type of process variation that exists in the FPGA. To obtain these delays, Timing Extraction measures, using only resources already available in the FPGA, the delay of a small subset of the total paths in the FPGA. We apply Timing Extraction to the Logic Array Block (LAB) on an Altera Cyclone III FPGA to obtain a view of the delay down to near-individual LUT SRAM cell granularity, characterizing components with delays on the order of tens to a few hundred picoseconds with a resolution of ±3.2ps, matching the expected error bounds. This information reveals that the 65nm process used has, on average, random variation of $\sigma/\mu = 4.0\%$ with components having an average maximum spread of 83ps. Timing Extraction also shows that as V_{DD} decreases from 1.2V to 0.9V in a Cyclone IV 60nm FPGA, paths slow down, and variation increases from $\sigma/\mu = 4.3\%$ to $\sigma/\mu = 5.8\%$, a clear indication that lowering V_{DD} magnifies the impact of random variation.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms: Algorithms, Measurement, Reliability

Additional Key Words and Phrases: Component-specific mapping, variation measurment, variation characterization, in-system measurement

ACM Reference Format:
Benjamin Gojman, Sirisha Nalmela, Nikil Mehta, Nicholas Howarth, and André DeHon. 2014. GROK-LAB: Generating real on-chip knowledge for intra-cluster delays using timing extraction. ACM Trans. Reconfig. Technol. Syst. 7, 4, Article 5 (December 2014), 23 pages.
DOI: http://dx.doi.org/10.1145/2597889

1. INTRODUCTION

Circuit variation is quickly becoming one of the biggest problems to overcome if the benefit from Moore's Law scaling is to continue. It is no longer possible to maintain

This work is supported by the National Science Foundation, under grant CCF-0904577.
Authors' addresses: B. Gojman, Computer and Information Systems, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104; email: bgojman@seas.upenn.edu; S. Nalmela, Juniper Networks, 10 Technology Park Drive, Westford, MA 01886; email: siri.nalmela@gmail.com; N. Mehta, Department of Computer Science California Institute of Technology MC 305-16, 1200 E. California Blvd., Pasadena, CA 91125; email: nikil.mehta@gmail.com; N. Howarth and A. DeHon, Electrical and Systems Engineering, University of Pennsylvania, 200 S. 33rd Street, Philadelphia, PA 19104; emails: howarthnj@gmail.com, andre@acm.org.
 1936-7406/2014/12-ART5 $15.00
DOI: http://dx.doi.org/10.1145/2597889

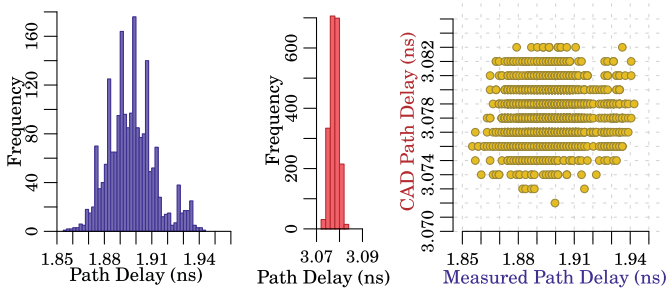


Fig. 1. Path delay of 1,000 nearly identical paths of length 7 LUTs, comparing measured delays to delays reported by the CAD tools for a Cyclone III 65nm FPGA.

an abstraction of identical devices without incurring huge yield losses, performance penalties, and high energy costs. Current techniques such as margining and speed grade binning are used to deal with this problem. However, they will become prohibitively conservative, only offering a limited solution that will not scale as variation increases.

Figure 1 concretely demonstrates the opportunities available to improve on these techniques. We carefully measured 1,000 paths consisting of seven buffers in one logic array block (LAB) of an Altera Cyclone III 65nm FPGA. Figure 1(a) shows a histogram of the results of these measurements. Similarly, Figure 1(b) shows the distribution of delays as computed by the CAD tools for these paths. Immediately we see a large difference between the two means. Techniques such as dynamic voltage scaling (DVS) [Chow et al. 2005] go a long way to reduce this difference. However, of interest here, and what DVS fails to address, is the wide range of the measured distribution—96ps. Ignoring this range forces all resources to operate, at best, accommodating the slowest measured resource. Knowledge of this distribution, on the other hand, provides the possibility of operating faster than the slowest resource. Figure 1(c) demonstrates that there is no correlation between the delays measured and those reported by the CAD tools, indicating that the delays we are measuring are not correlated to any effect modeled by the Altera tools.

FPGAs have the unique advantage over ASICs that they can use more fine-grained and aggressive techniques that carefully choose which resources to use after fabrication in order to mitigate adverse variation effects. Mehta et al. [2012] show that a component-specific mapping solution reduces energy needs by 50% and will be a necessity to extend beneficial scaling as variation increases. This approach requires measurement of the underlying resource delays for the CAD tools to generate a custom mapping perfectly adapted to the variation in the FPGA. In this article, we present Timing Extraction, a methodology that allows the kind of fine-grained measurement of fabricated component delays necessary for Mehta et al. [2012] in an efficient and inexpensive manner, utilizing only resources already available on conventional FPGAs. Although suitable to nearly any FPGA, to practically validate Timing Extraction, we apply it to clusters (LABs) in the Altera Cyclone III and Cyclone IV FPGAs and confirm that the measurements and calculations reflect underlying process variation.

The key challenge in Timing Extraction is that it is not possible to directly measure the characteristics of every LUT or wire in an FPGA. Nonetheless, we show that it is possible to obtain fine-grained delays using an indirect approach to measure, compute, and characterize the variation of small groups of components. Wong et al. [2009] demonstrated the feasibility of measuring path delays without the need for any

dedicated test circuitry by surrounding the path with two registers that are already part of the reconfigurable fabric. Timing Extraction takes advantage of this measurement technique but goes further by demonstrating how to use the measurements to resolve the delays of individual resources.

The measured path is composed of multiple components, the individual delays of which we would like to know. By configuring and measuring a small set of overlapping paths, we can set up a linear system of equations that, when solved, gives the individual delay of each component in the paths [Gojman et al. 2011]. A simple example will give better intuition as to what the technique actually accomplishes. Consider that we measure three paths. Path 1 is composed of components A and B; Path 2, B and C; and Path 3, C and A. Suppose the delays of the paths are 5ps, 4ps, and 3ps, respectively. That leads to the system of equations to follow:

$$\begin{aligned} A + B &= 5\text{ps} \quad \text{Path 1} \\ B + C &= 4\text{ps} \quad \text{Path 2} \\ C + A &= 3\text{ps} \quad \text{Path 3} \end{aligned}$$

Even though we did not measure the delays directly, with little work we can solve for the delay of A, B, and C to be 2ps, 3ps, and 1ps, respectively.

Timing Extraction does exactly this but at a level that allows us to characterize a full FPGA. Formulating the naive problem, where every wire and transistor in the FPGA are represented by a separate variable in the system of equations, invariably leads to an underdetermined system without a unique solution (Section 3.2). However, Timing Extraction judiciously groups components into discrete units of knowledge (DUKs), which, combined with a careful selection of measured paths, guarantee a solution to the delay of each DUK in the system (Section 3.3). With that information, we can predict the delay of any path that could be used when mapping logic to the FPGA.

We begin with a brief review of the required background (Section 2). Section 3 develops the ideas of Timing Extraction by using the logic clusters in the Cyclone III as a case study. We expand Timing Extraction to intra-LUT measurements and begin to show how it is generally applicable in Section 4. Section 5 analyzes the expected measurement error. Results from our measurements are presented in Section 6. While we present concrete details on how to measure the Cyclone III, the general technique can be extended to any modern FPGA; in Section 7, we briefly sketch how to port the ideas and why they are generally applicable. An outline of future work is explored in Section 8, before the conclusion (Section 9).

Novel contributions of this work include the following:

—First identification and demonstration of techniques for determining the delay of individual LUTs and the unique interconnect delay between pairs of LUTs using only on-chip FPGA resources
  —Identification of smallest delay-measurable groups of components
  —Identification of the smallest set of measurements necessary to extract complete fine-grained delay information within a cluster (LAB)
  —Algorithm for calculating component delays from path measurements
—Technique for predicting delay of any path in a cluster (LAB) using component LUT delay measurements
—First set of measurements to fully characterize the delay components within a cluster (LAB) and within LUTs in a commercial FPGA
—Quantification of process variation at a near-LUT-level granularity
—Quantification of increased random variation with voltage scaling
—Characterization of significant contribution from random variation in process variation

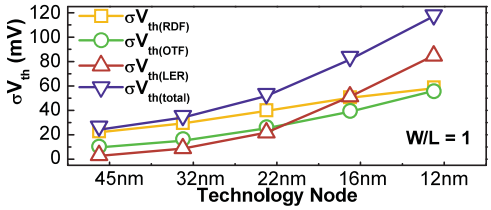


Fig. 2. $\sigma_{V_{th}}$ as a function of technology nodes, based on predictive technology models. This considers the individual effects of random dopant fluctuations (RDFs), line edge roughness (LER), and oxide thickness (OTF) from Ye et al. [2010].

A preliminary version of this work appeared in Gojman et al. [2013], where the basic ideas of Timing Extraction are introduced and applied exclusively to the clusters, measuring individual logic element (LE) delays. This work expands Timing Extraction to measure even smaller circuit structures, intra-LUT delays. At the same time, it paves the way for the general application of Timing Extraction to an arbitrary circuit structure in an FPGA (Section 4). Furthermore, this work formally quantifies the expected error the measurement technique introduces (Section 5) and presents significantly more precise measurements and results (Section 6), done by increased control of the placement and routing of our test circuit (Section 6.1), that are demonstrably within the expected error bound (Section 6.3).

2. BACKGROUND

2.1. Process Variation

Process variation refers to differences between device parameters due to manufacturing. These differences ultimately affect the delay and energy requirements of the device. Correlated variation has historically accounted for the majority of process variation, where the amount a device varies is correlated to some parameter, such as location on the wafer. Consequently, most techniques aim to reduce correlated variation. Binning, for example, mitigates die-to-die variation, while biasing [Lewis et al. 2009] reduces correlated regional variation. In essence, correlated variation provides a model that can be used to reduce process variation. However, as feature sizes continue to shrink, more and smaller transistors fit on one chip, greatly increasing the contribution of random variation to process variation. Unfortunately, unlike correlated, random variation is not easily modeled and mitigated.

Figure 2 shows how the three main contributors to random variation—oxide thickness, line edge roughness, and random dopant fluctuations—lead to a significant increase in variation experienced by V_{th}, the transistor's threshold voltage, as technology scales.

The value of V_{th} has a direct and profound effect on the performance and energy requirements of a transistor. Equations (1) and (2) represent the current through a transistor during the saturation and subthreshold operating points [Rabaey et al. 1999; Hanson et al. 2006]. Although physical parameters such as transistor geometry, W, L, and dopant concentration, η, have a strong stochastic variation component, it is the exponential dependence on V_{th} that brings about the harmful effects of random variation on the current through a transistor:

$$I_{ds,sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right) \tag{1}$$

$$I_{ds,sub} = \frac{W}{L} \eta C_{ox} (n-1) \cdot {v_T}^2 \cdot e^{\frac{V_{gs}-V_{th}}{n \cdot v_T}} \left(1 - e^{\frac{-V_{ds}}{v_T}} \right). \tag{2}$$

In turn, the propagation delay τ_{pd} and leakage energy of the circuit are a function of current (Equations (3) and (4)):

$$\tau_{pd} = C_l \cdot \frac{V_{ds}}{I_{ds}} \tag{3}$$

$$E_{leak} = I_{ds,sub} \cdot V_{ds} \cdot \tau_{cycle}. \tag{4}$$

As such, random physical variation expresses itself in differences in the energy efficiency and delay of a transistor.

Statistical static timing analysis (SSTA) [Srivastava et al. 2005] attempts to model the expected random variation and with it the expected behavior of the FPGA. With this model, the CAD tools can generate a mapping that, statistically speaking, will reduce the effects of random variation. Unfortunately, this solution inherently fails to accommodate every FPGA. Instead of employing this one-size-fits-all solution, Timing Extraction measures and extracts detailed delay information from the FPGA after fabrication. This can then be provided to the CAD flow, which generates a component-specific mapping tailoring the design to the particular FPGA [Mehta et al. 2012].

The delay of a component in the FPGA not only is affected by process variation but also can fluctuate due to environmental and temperature changes [Li et al. 2010] as well as aging effects [Stott et al. 2010]. To ensure that measured delays consistently represent process variation, Timing Extraction requires that measurements be taken in a highly controlled manner. Section 6.1 details the controls employed for our application on the Cyclone FPGA. The consistency of the results presented in Section 6.3 concretely demonstrates that Timing Extraction does measure process variation.

2.2. Altera Cyclone LAB Architecture

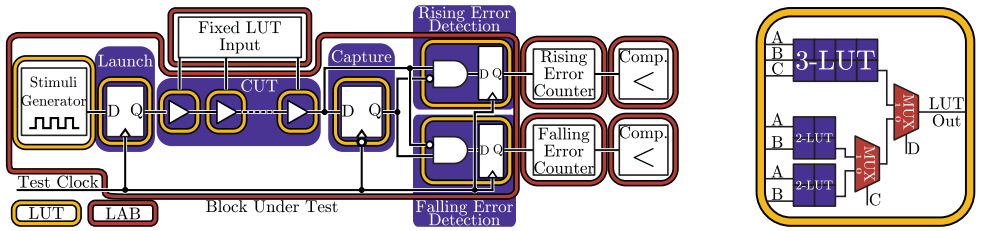
Timing Extraction is a general methodology that provides fine-grained delay measurement of small groups of components within an FPGA. Although it is applicable to any FPGA, to ground the presentation in this article, we focus our application to the LABs of the Altera Cyclone III and Cyclone IV FPGAs.

The LAB in these FPGAs is composed of 16 LEs, each having a 4-LUT and optional register output, a set of 38 routing channels for external inputs, and 16 local routing channels for LE-to-LE communication. Local routing is 50% depopulated where LUT inputs *A* and *B* form a complete input set; that is, every LE can connect to every other LE in the LAB by using either input *A* or *B*, and similarly inputs *C* and *D* form a complete input set (Figure 4). The scope of this article limits delay measurements to the 16 LEs and the 16 local routing channels in the LAB.

2.3. Path-Delay Measurements

We use a launch-capture technique to measure the delay of a path in an FPGA. In this approach, a combinatorial circuit, known as the circuit under test (CUT), is configured between a launch register and a capture register. Starting at an initial frequency and increasing to a maximum frequency, signals are sent from the launch register to the capture register. When a signal fails to reach the capture register within half of a clock cycle, we know that the delay of the path is greater than twice the frequency at which that signal was clocked. This technique has been successfully used to capture the delay of paths on FPGAs for many applications [Smith and Tian 2009; Sedcole et al. 2008; Wong et al. 2009; Majzoobi et al. 2010].

A limitation of this measurement technique, however, is that it cannot measure a path that is faster than twice the highest frequency supported by the FPGA's on-chip PLLs. Twice the frequency comes from the fact that the launch and capture registers are clocked on opposite clock edges. Therefore, any work that exclusively uses this



(a) Components and simplified placement of path-delay measurement circuit

(b) Block diagram for a Cyclone III 4-LUT

Fig. 3. Test setup and Cyclone-III 4-LUT decomposition.

measurement technique will be limited to reporting delays of long paths. To ground this, consider that the maximum frequency for the Cyclone III PLLs used in this work is 402.5MHz. This means that the fastest path we can measure is $\frac{1}{2 \cdot 402.5} = 1.24$ns. Figure 1(a) shows that, on average, a path of length 7 LUTs is measured to take 1.90ns, meaning that, roughly, on average, the delay through 1 LUT is 271ps. Combining this fact with our maximum frequency leads to the conclusion that the smallest path we can measure is five LUTs long. This ignores the expected variation spread. Therefore, to err on the side of caution, we do not measure anything with less than 6 LUTs in a path. Nevertheless, as we will later show, this work reports on delays on the order of 1 LUT by taking delay measurements of long paths and breaking them into smaller parts. Wong et al. [2009] and Tuan et al. [2011] take only a single measurement within each LAB or CLB and make no attempt to characterize within-LAB variation. The most closely related technique used in Culbertson et al. [1997] and Yu et al. [2010] takes the difference between two ring oscillators to extract subcluster delays. However, this approach fails to account for the unique interconnect delay between pairs of LUTs, nor is it able to account for register delays.

Due to the nature of CMOS and FPGA circuit design that uses NMOS pass transistors, there could be a delay difference in a rising transition as compared to a falling transition. In order to separate the falling and rising delays, our CUT is composed of buffers in series. In this way, all elements in a path transition in the same direction, allowing us to separate the rising transition through the path from falling transitions. Figure 3(a) shows a diagram of the path-delay measurement circuit used. A signal with a 50% duty cycle is provided to the launch register. The signal propagates through the CUT and the capture register records its output. Errors are detected by the two error detection circuits, one monitoring rising failures, the other, falling failures. In Gojman et al. [2013], we demonstrate a systematic 10% difference between falling and rising delays. Though we always capture both transitions, due to this systematic difference, we only report rising transitions in this work.

For each LUT that forms the CUT, only one input is used. The remaining inputs are fixed to a chosen binary vector provided by the fixed LUT input block on an adjacent LAB. By explicitly fixing the value of these unused inputs, we control exactly which pair of configuration SRAM cells in the LUT is used, allowing measurement of the intra-LUT delays as explained in Section 4. Figure 3(b) shows the architecture of a Cyclone III 4-LUT [Altera 2005b, 2009]. From this, we expect that the LUT delay will vary systematically depending on which input is controlling, with inputs A and B slower than C, which will be slower than D.

Because of operating variation such as clock jitter, it is not sufficient to observe one failure to declare the delay of a path. Instead, the path is tested at one frequency many times, and two counters, for rising and falling transitions, keep track of how many

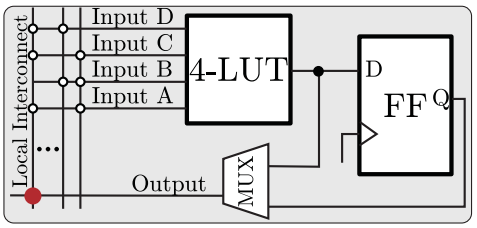


Fig. 4. Block diagram of a Cyclone FPGA LE (4-LUT and register), including local interconnect.

failures occurred at that frequency, for that transition. If at frequency f, the number of failures reaches a percent of the total number of transitions, the delay of that circuit is reported as $\frac{1}{f}$. The transition from no failures to 100% failures is gradual. If we assume that the variation that caused this gradual failure rate is mostly stochastic and has a symmetric probability distribution, then the 50% failure rate provides the most accurate estimate of delay given a small number of samples. We do not use this frequency for regular operation, since at this frequency signals fail timing 50% of the time. Knowing the variance in cycle time, we can then select a suitable operating frequency that keeps timing errors down to an acceptable level.

3. TIMING EXTRACTION

The general idea behind Timing Extraction is easy to understand. It is not possible to measure the delay of every component in an FPGA directly since individual transistors or wires cannot be isolated from their surrounding components. Nevertheless, by measuring the delay of different paths through an FPGA, it is possible to decompose the delays of these paths into their constituents. Essentially, each path consists of a linear sum of the delay of its parts; therefore, we can cast this problem as a linear system of equations where each equation represents a path and equals the measured delay of the path. With enough equations, we can solve for all the unknowns and directly acquire the delays of every component used in these paths. In order for the system of equations to have a unique solution, it is imperative to carefully select what the variables in the equations represent. In this section, we use the Altera Cyclone LAB architecture to ground the development of the general Timing Extraction methodology. We begin by considering what is individually calculable, followed by an analysis of what paths must be measured. This leads to the realization that our initial assessment of what is individually calculable is flawed, which ultimately leads us to the notion of DUKs, allowing for a complete solution.

3.1. Logical Components

It is not possible to measure the delay of a single wire or transistor in the FPGA, even indirectly. To explain, consider the simple representation of the Cyclone LUT in Figure 4. Suppose we want to know the delay of only the highlighted crosspoint in isolation. This is not possible since any path that uses that crosspoint must use the labeled Local Interconnect, Output, and MUX. However, since any path that uses this crosspoint will naturally use the other components, there is no practical reason to measure its delay independent of these components. This gives the notion of a Logical Component or LC Node and the first attempt at defining what the variables in our system of equations represent.

As explained in Section 2.3, measured paths start at a register, go through zero or more buffers, and end at a register. A path in a LAB will begin at a register, go through some number of LUTs, and end at a second register. Figure 5 shows how we

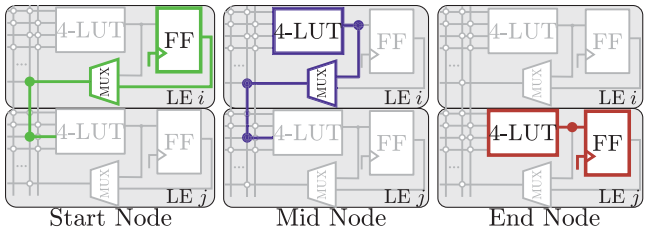


Fig. 5. Highlighted, an example of the components that form each of the three types of LC Nodes in a Cyclone LAB.

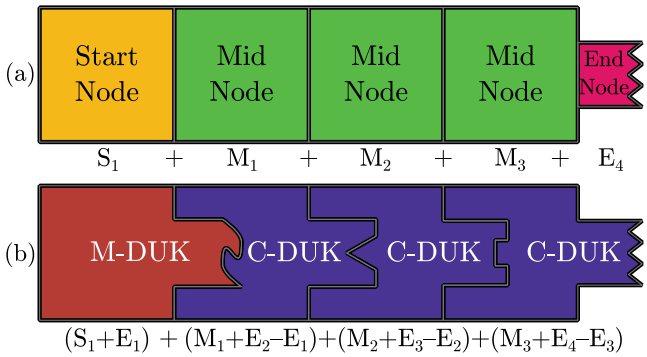


Fig. 6. Equivalence between LC Node basis and DUK basis. To build intuition, the shapes give a geometric interpretation to the delay of each LC Node or DUK. The equations below each figure show this mathematically.

decompose this path into three types of LC Nodes. The path begins at an LC Node whose first component is a register, known as a *Start Node*; goes through zero or more LC Nodes with no registers, *Mid Nodes*; and ends at an *End Node*, an LC Node whose last component is a register.

Figure 6(a) represents a path using groups of Start, Mid, and End Nodes. Thus, we let LC Nodes correspond to variables in our system of equations and represent each measured path delay by a linear sum of the delays of these LC Nodes.

To solve for the delay of all LC Nodes, we must measure at least a number of paths equal to the number of LC Nodes in a LAB. A Start Node and Mid Node start at one LE and end at a second LE. Considering that there are 16 LEs in a LAB and two input sets, AB and CD (Section 2.2), this gives a total of $16 \times 15 \times 2 = 480$ Start and 480 Mid Nodes per LAB. Since End Nodes only use one LE, there are only 16 End Nodes per LAB. In total, there are $480 + 480 + 16 = 976$ LC Nodes in a LAB, which is the minimum number of paths we must measure to solve for their delay.

3.2. Matrix Representation

Once we measure a correct set of 976 paths and solve for the delay of all LC Nodes, it will be possible to reconstruct the delay of any of the approximately 10^{18} paths within a LAB. Therefore, the problem is deciding which 976 paths to measure. To better discuss this solution, we formulate our system of equations as a matrix. A path is represented by a row, while a column describes an LC Node. An entry L_{ij} in the matrix is 1 if LC Node j forms part of path i, and 0 otherwise. Since there are 976 LC Nodes and we need at least 976 paths, our matrix will be at least as large as 976×976. Once the

delays of the paths are measured, we use this matrix and the path delays to solve for all LC Nodes.

Linear algebra tells us that if the rank of the original matrix is equal to the number of LC Nodes, then we can solve for the delay of each LC Node. Otherwise, if it is less than the number of LC Nodes, the system is underdetermined and, in general, contains an infinite number of solutions. Unfortunately, even if we measure the delay of all 10^{18} paths, the rank of the matrix is 960, 16 less than the total number of LC Nodes in a LAB. Section 7 provides some intuition as to why this is the case for any FPGA in which we let LC Nodes represent the variables in the system of equations.

Even though the matrix is rank deficient, it must have a nonempty vector space that accounts for its basis. In turn, this means that there must be a set of linearly independent paths, which, when taken together and measured, allow us to compute the delay of any other measurable path in the circuit. Since the LAB has a matrix with rank 960, we only need to measure a linearly independent set of 960 paths to compute the delay of any path in the LAB. Essentially, instead of using a basis where every path in the matrix is represented by a linear combination of LC Nodes, we use a basis where every path is represented by a linear combination of the 960 paths measured.

Although this approach provides the delay of any path, it does not achieve the desired results for two reasons. First, it is difficult to incorporate these results into conventional routing algorithms when a component-specific route is sought, since routing algorithms [McMurchie and Ebeling 1995] tend to expand routes incrementally and we only have complete path-delay information. Second, the basis does not provide a fine-grained understanding of the variation. The next section addresses these shortcomings by defining a particularly convenient basis that spans the matrix yet provides the fine-grained, incremental variation information desired.

3.3. DUK Basis

Timing Extraction's objective is to provide fine-grained delay information that can then be used to characterize the variation in the FPGA as well as perform a component-specific mapping to the FPGA. We know it is not possible to solve for the delay of every LC Node; however, our solution should allow us to formulate path delays as a linear sum of a small number of components. By definition, an LC Node is the smallest delay we care to measure; however, since we cannot solve for LC Nodes, we consider the next best thing, a basis where the variables represent a small linear combination of LC Nodes. We refer to this small linear combination of LC Nodes as a *DUK*. First, we introduce the vectors that compose the DUK basis; then we show the equivalence between an LC-based and a DUK-based model; and finally, we demonstrate that unlike LC Nodes, we can compute the delay of DUKs.

Instead of having three types of variables that are combined to represent a path, this basis contains two types of DUKs. The delay of a Start Node plus an End Node forms the first DUK (Equation (5)). On its own, this DUK forms a complete measurable path, starting at a register and ending at a second register. Moreover, all paths stem from this DUK; therefore, we refer to it as a Mother DUK, or *M-DUK*. The second DUK is known as a Child DUK, or *C-DUK*. As its name suggests, it follows the Mother DUK and incrementally grows a path. A C-DUK consists of the delay of a Mid Node plus the difference of two End Nodes (Equation (6)):

$$\text{M-DUK} = S_i + E_j \tag{5}$$

$$\text{C-DUK} = M_i + E_j - E_k. \tag{6}$$

Assuming we have their delays, together, these two types of DUKs allow us to compose any measurable path in exactly the same way that LC Nodes did. In general, a

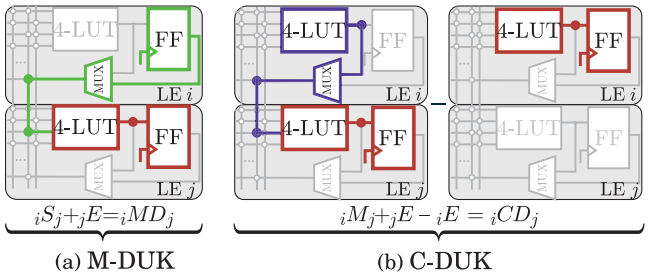


Fig. 7. Highlighted, an example of the LC Nodes that form the two types of DUKs in a Cyclone LAB.

measurable path will be represented by an M-DUK and zero or more C-DUKs. For a path to be measurable, it must start and end at a register. M-DUKs naturally represent such paths. The function of a C-DUK is to replace the End Node and extend the path by adding a Mid Node and a new End Node. Consider the path shown in Figure 6(a) consisting of a Start Node, three Mid Nodes, and an End Node. We can easily represent this path in the DUK basis using one M-DUK and three C-DUKs, as shown in Figure 6(b). Figure 6(b) represents each DUK as a jigsaw piece to give a geometric meaning to the notion that two DUKs must complement each other in order to correctly represent a path. Here, instead of each DUK having a different delay, each DUK has a unique shape. The concave left side of a C-DUK represents the carved-out delay of the subtracted End Node, while the convex right side of a DUK shows the addition of an End Node.

In general, given a path represented by LC Nodes, we can easily re-express it using the DUK basis by replacing the Start Node with an M-DUK containing the same Start Node, and every Mid Node by a C-DUK composed in part by the Mid Node, and subtracting the same End Node that is added to the DUK before it. The last C-DUK must also contain the End Node of the path in question.

3.4. DUKs in Cyclone LAB

Figure 7 shows how DUKs map to LE i and j in a Cyclone LAB. Similar to the Start Node, the M-DUK spans two LEs. Since there are 16 LEs in a LAB and two input sets (Section 2.2), there are $16 \times 15 \times 2 = 480$ M-DUKs. An equal number of C-DUKs exist, since a C-DUK also spans two LEs. Using the 960 DUKs in a LAB, it is possible to represent any path in the LAB originally represented by a set of LC Nodes. Under Figure 7 appear two LC Node equations leading to the corresponding DUKs. A subscript prefix on both the LC Nodes and the DUKs indicate the source LE, and a subscript suffix signals the sink LE. We can establish a one-to-one correspondence between Start Nodes and M-DUKs (Figure 7(a)) by observing that the prefix and suffix on the Start Node match the prefix and suffix of the M-DUK. Essentially, it indicates that if the Start Node begins in LE i and ends in LE j, the M-DUK will as well. A similar bijection exists between Mid Nodes and C-DUKs (Figure 7(b)). The equations in Figure 7 also indicate which End Nodes must be added or subtracted to correctly form the DUK.

These equations and this notation allow us to trivially transform a path based on LC Nodes into one using DUKs. We replace the Start Node with the M-DUK that has the same source and sink LE. Similarly, we replace every Mid Node with the matching C-DUK. The delay contributed by the End Node will already form part of the last DUK. An example will help solidify this transformation.

Consider the path with four LC Nodes:

$$_iS_j + {}_jM_k + {}_kM_l + {}_lE.$$

Applying the transformation algorithm described previously leads to the path

$$ {}_iMD_j + {}_jCD_k + {}_kCD_l. $$

Expanding each DUK to its LC Node representation leads to

$$ \underbrace{{}_iS_j + {}_jE}_{{}_iMD_j} + \underbrace{{}_jM_k + {}_kE - {}_jE}_{{}_jCD_k} + \underbrace{{}_kM_l + {}_lE - {}_kE}_{{}_kCD_l}, $$

which, after simplifying the terms, equals the original LC Node-based path.

It is not a coincidence that the number of DUKs, 960, matches the rank of the matrix formed by paths $\times$ LC Nodes. The previous algorithm shows how a linear combination of DUKs can be used to represent an arbitrary measurable path. This is the definition of a basis for the matrix. Therefore, these DUKs form a basis for the path–LC Node matrix. As such, by obtaining the delay of the 960 DUKs, we can compute the delay of any of the 10^{18} paths in the LAB.

This basis is superior to the one suggested at the end of Section 3.2, where 960 linearly independent paths are selected to form the basis, for several reasons. First, DUKs can be composed incrementally, allowing routing algorithms to easily incorporate this delay information into their path search. Second, DUKs provide a uniformity that the other basis lacks. There is no guarantee that all paths in the other basis will be of the same length or use similar LUT inputs. Therefore, it is not easy to compare delays between and within LABs. DUKs, on the other hand, have two consistent forms, M-DUKs and C-DUKs. We can directly compare one C-DUK using LUT input A to another C-DUK using LUT input A and know that if one is faster, it is due to process variation and not because of differences in what they represent. Finally, DUKs provide very fine-grained delay information, almost on the order of one LE, while the other basis only has delays of paths.

3.5. Obtaining DUK Delays

It should come as no surprise that it is impossible to measure C-DUKs directly, since one term subtracts the delay of an End Node. It is relatively simple, however, to figure out which paths combine to give a C-DUK's delay. Consider C-DUK ${}_iM_j + {}_jE - {}_iE$ from Figure 7(b). To get this delay, we simply measure a path starting with a set of Nodes represented by path prefix π and ending in Nodes ${}_iM_j + {}_jE$ and subtract from it a path starting with the Nodes in π and ending in Node ${}_iE$. This leads to the path equation

$$ (\pi + {}_iM_j + {}_jE) - (\pi + {}_iE) = {}_iM_j + {}_jE - {}_iE. \tag{7} $$

In a sense, this mathematically demonstrates the purpose of a C-DUK, removing the last End Node in a path and replacing it with a new Mid Node and End Node.

Since every M-DUK represents the delay of a Start Node plus an End Node and a path must begin at a Start Node and end at an End Node, our path measurement technique (Section 2.3) should allow us to directly measure the delay of every M-DUK. Unfortunately, as established in Section 2.3, the shortest path we can confidently measure is of length 6, while an M-DUK forms a much smaller path of length 1 LUT and two registers (Figure 7(a)). Therefore, we take an indirect approach to measuring the delay of an M-DUK by measuring three paths and taking a linear combination of these paths.

To compute the delay of M-DUK ${}_iS_j + {}_jE$, we measure one path that begins by a set of nodes represented by π_1 and ends with ${}_lM_j + {}_jE$. Then we measure a second path that begins with ${}_iS_j + {}_jM_k$ and ends with a set of nodes represented by π_2. Finally, we measure a path that is similar to the second path at the beginning and similar to the

first path at the end: $\pi_1 + {}_lM_j + {}_jM_k + \pi_2$. Adding the first two paths and subtracting the third leads to the delay of the M-DUK as shown in the following path equation:

$$(\pi_1 + {}_lM_j + {}_jE) + ({}_iS_j + {}_jM_k + \pi_2) - (\pi_1 + {}_lM_j + {}_jM_k + \pi_2) = {}_iS_j + {}_jE. \quad (8)$$

There exist a few requirements on which nodes may form part of π_1 and π_2. Since the third path uses both π_1 and π_2, we must make sure that each of the 16 LUTs in the LAB is used only once between the Nodes in π_1, π_2 and the two Mid Nodes ${}_lM_j + {}_jM_k$. Also, π_1 and π_2 should not use the LUT i or j. These requirements are easy to satisfy and allow for long paths that we can measure using the limited frequency resources in the Cyclone III and Cyclone IV FPGAs.

All told, we measure two paths for every C-DUK and three for each M-DUK; at worst, this means we must measure $2 \times 480 + 3 \times 480 = 2{,}400$ paths per LAB. Although this is slightly larger than the minimum of 960 given by performing Gaussian Elimination on the path $\times$ LC Node matrix, it is still a small number compared to the total possible paths, and it meets the Timing Extraction goals: fine-grained measurements suitable for direct variation characterization and component-specific routing.

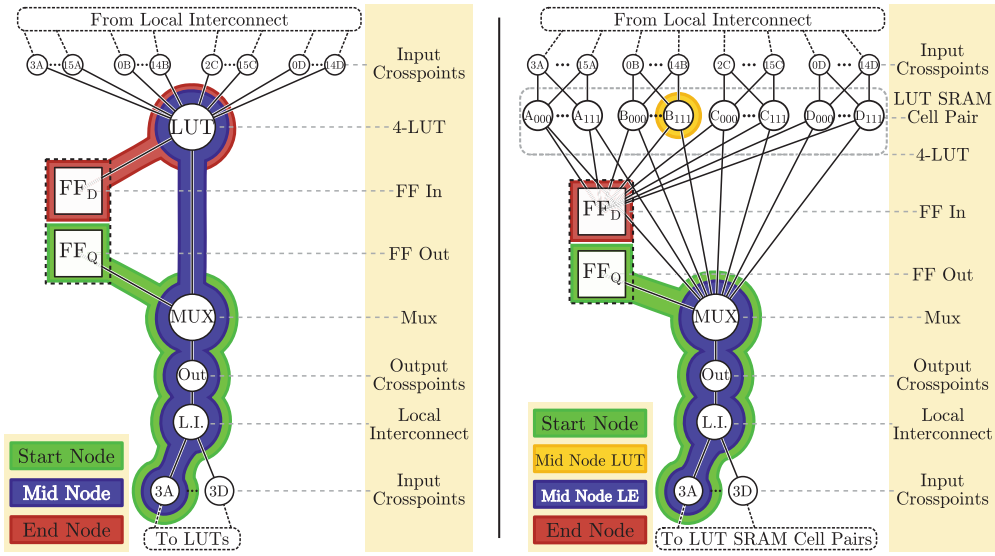
4. INTRA-LUT TIMING EXTRACTION

The previous section introduced the key concepts of Timing Extraction by explaining how to acquire detailed delay measurements for LEs within a LAB. In this section, we apply these ideas to the internal structure of the LUTs within the LEs, including SRAM cells and MUX paths (Figure 3(b)), and use it as an opportunity to demonstrate how Timing Extraction is easily applicable to other circuit structures. To do this, we examine how breaking the LUT into individual memory cells changes the structure of our Logical Components (Section 3.1). This forces us to reconsider which DUKs are necessary to cover any required path within a LAB and introduces a new DUK to complement the existing two and acquire fine-grained intra-LUT delays.

To measure intra-LUT delays, it is useful to represent the LE as a graph of physical components. Figure 8(a) shows this graph representation for the Cyclone LE from Figure 4. To capture the fact that a measured path must start and end at a register, the register is modeled by two nodes in the graph, one for the output, FF_Q, and one for the input, FF_D. Two physical components, i and j, in this graph will be part of the same Logical Component (Section 3.1) if and only if j is downstream from i and i is the only upstream physical component that connects to j. Essentially, a Logical Component starts at a physical component with fanin greater than 1 and ends before the next downstream physical component with fanin greater than 1.

The three Logical Components introduced in Figure 5 immediately result from applying this definition when we compute LC Nodes that explicitly use or avoid the register nodes, as highlighted in Figure 8(a).

When considering intra-LUT delays, it is necessary to modify the graph from Figure 8(a) so that the individual SRAM cells within the LUT are accounted for, but first we must understand how to represent these SRAM cells in the graph. As shown in Figure 3(a), LUTs in a measured path, which form part of the circuit under test (CUT), are logically configured as simple buffers. For a K-LUT, this means that only one out of K inputs is used, and only two SRAM cells in the LUT will be read. For a given used input, which two SRAM cells are read depends on how the unused $K-1$ inputs are configured. For example, in the Cyclone III 4-LUT, when the LUT is programmed as a buffer on input A and the other three inputs are fixed low, we use the SRAM cells addressed by input bit-vectors 0000 and 1000. If, instead, the unused inputs are fixed high, the SRAM cells will be those read by bit-vectors 0111 and 1111. Altogether, there are eight pairs of SRAM cells that can be used to implement a buffer on input A. We can label the eight pairs as A_{000} through A_{111}. A similar labeling for inputs B, C, and



(a) Unifide LUT Representation (b) Intra-LUT Representation

Fig. 8. Graph representation of Figure 4. Highlighted, an example of the components that form each of the types of LC Nodes in a Cyclone LAB.

D leads to 32 pairs of SRAM cells, and each one of these 32 pairs is represented as a node in Figure 8(b), replacing the LUT node in Figure 8(a). Logical components in this new graph are formed in the same way as before; however, the result, as highlighted in Figure 8(b), is four types of logical components, *Start Node*, *Mid Node LUT*, *Mid Node LE*, and *End Node*, instead of the original three.

4.1. Intra-LUT DUKs

A path is now formed by a Start Node followed by zero or more pairs of Mid Node LUT–Mid Node LEs, and terminating with a Mid Node LUT and an End Node. Though the number and composition of the paths is different, it is still the case that we cannot solve for the delay of each LC node since this is a fundamental property of the launch-capture measurement technique. Therefore, we again consider forming DUKs to get the desired information.

The original M-DUK is essentially representing the smallest path that can be formed between two registers. The original C-DUK is a mechanism that extends a path by replacing one LC Node with two, while maintaining the requirement that the path starts and ends at a register. Defining the two DUKs this way allows us to define similar DUKs in this new representation.

The smallest path between two registers is one that begins at a Start Node, goes through one Mid Node LUT, and finishes at an End Node (Equation (9)). To extend a path, we replace the End Node with a Mid Node LE followed by a Mid Node LUT and a new End node (Equation (10)). As before, these two DUKs are all that is required to form any measurable path; however, representing intra-LUT delays greatly increases the total number of DUKs:

$$\text{M-DUK} = {}_iS_j + {}_jM_{xxx}^{LUT} + {}_jE \tag{9}$$

$$\text{C-DUK} = {}_iM_j^{LE} + {}_jM_{xxx}^{LUT} + {}_jE - {}_iE. \tag{10}$$

To count the number of DUKs, we look at what LC Nodes form the DUK and the corresponding connectivity between these Nodes. Considering the LAB in a Cyclone III, a Start Node begins at a register in LE i and ends before a LUT input in LE j. Since there are 16 LEs per LAB and the connections from one LE to another are 50% depopulated, there is a total of $16 \times 15 \times 2 = 480$ Start Nodes. A Start Node connects to eight distinct Mid Node LUTs, one for each pair of SRAM cells used to implement a buffer on a given LUT input. Finally, the Mid Node LUTs connect directly to exactly one LUT register, represented by an End Node. Therefore, altogether, there are $480 \times 8 \times 1 = 3{,}840$ M-DUKs. A similar accounting also leads to 3,840 C-DUKs.

Although still less then the 10^{18} measurable paths in a LAB, 7,680 DUKs is eight times greater than the 960 DUKs accounted for in Section 3.4. This factor of eight comes directly from the fact that now we represent pairs of SRAM cells in the 4-LUT of the Cyclone III. To reduce the total number of DUKs, we introduce a new type, a DUK that swaps one Node for another. Specifically, the Sibling DUK, or *S-DUK*, will swap one Mid Node LUT with another, as represented by Equation (11):

$$\text{S-DUK} = {}_iM^{LUT}_{xxx} - {}_iM^{LUT}_{yyy}. \tag{11}$$

A given Mid Node LUT in some path can be replaced by seven other Mid Node LUTs without changing anything else in the path. Physically, this swap replaces one pair of SRAM cells used to implement a buffer on a particular input of a LUT with another pair using that same input in that same LUT. Since there are four inputs and 16 LEs, there is a total of $4 \times 16 \times 7 = 448$ S-DUKs. Having these S-DUKs allows us to reduce the number of M-DUKs and C-DUKs by a factor of eight since we can measure M-DUKs and C-DUKs fixing unused LUT inputs to one value, and then use S-DUKs if a different value is desired. As such, we end up with a total of $480 + 480 + 448 = 1{,}408$ DUKs per LAB, a much better result than 7,680 before introducing S-DUKs.

4.2. Obtaining DUK Delays

All that remains is to describe which, and how many, paths should be measured to compute the delay of all 1,472 DUKs. For M-DUKs and C-DUKs, we use the same mechanism as in Section 3.5 but adjust it to account for the new LC Node representation. To compute the C-DUK's delay described by Equation (10), we again measure two paths that differ only in their endings but have same prefix π (Equation (12)):

$$\left(\pi + {}_iM^{LE}_j + {}_jM^{LUT}_{000} + {}_jE\right) - (\pi + {}_iE) = {}_iM^{LE}_j + {}_jM^{LUT}_{000} + {}_jE - {}_iE. \tag{12}$$

The M-DUK is computed by measuring three paths where the sum of the first two minus the third gives the desired M-DUK delay as shown in Equation (13):

$$\begin{aligned}&\left(\pi_1 + {}_lM^{LE}_j + {}_jM^{LUT}_{000} + {}_jE\right) + \left({}_iS_j + {}_jM^{LUT}_{000} + {}_jM^{LE}_k + \pi_2\right) \\ &\quad - \left(\pi_1 + {}_lM^{LE}_j + {}_jM^{LUT}_{000} + {}_jM^{LE}_k + \pi_2\right) = {}_iS_j + {}_jM^{LUT}_{000} + {}_jE.\end{aligned} \tag{13}$$

Since S-DUKs are the difference of two Mid Node LUTs, they are computed using two paths that differ in exactly one Mid Node LUT (Equation (14)). However, since there is very little restrictions on what the paths should be, we can reuse any one of the paths from either the C-DUK or M-DUK measurements as the first path in Equation (14):

$$\left(\pi_1 + {}_iM^{LUT}_{000} + \pi_2\right) - \left(\pi_1 + {}_iM^{LUT}_{xxx} + \pi_2\right) = {}_iM^{LUT}_{000} - {}_iM^{LUT}_{xxx}. \tag{14}$$

Therefore, as before, we have that C-DUKs require two paths, M-DUKs three, and, due to path reuse, S-DUKs only need one more. This means $2 \times 480 + 3 \times 480 + 1 \times 448 = 2{,}848$ paths must be measured to compute the delay of all DUKs in one LAB. In other words, by measuring 19% more paths than in Section 3.5, we augment our delay

knowledge with full detailed measurement information about the delay within the individual LUTs. Section 6.2 shows detailed results of these measurements.

5. MEASUREMENT PRECISION

The precision of the delay computed for each DUK is limited by the granularity to which we can adjust the clock used in the launch-capture measurement technique. In this section, we quantify the expected DUK error introduced due to this limited clock granularity. Section 6.3 demonstrates how our empirical results match the analysis presented here. As explained in Section 2.3, measuring the delay of a path requires adjusting the test clock to find the frequency at which the signal first fails to reach the capture register. Assuming the finest granularity by which the frequency can be adjusted is Δ_{clock} seconds, then any measurement made will at worst be Δ_{clock} seconds slower than the actual delay of the path (Equation (15)):

$$\tau_{measured} = \tau_{path} + \varepsilon_{path} \mid \varepsilon_{path} < \Delta_{clock}. \tag{15}$$

To determine the error in DUK delays due to limited clock granularity, we refer to Section 3.5 and Section 4.2, which explain how to compute the delay of a DUK. A C-DUK's delay is the difference of the delay of two paths. Equation (16) expresses the delay and measurement error of a C-DUK in terms of the delay and measurement error of two paths, A and B:

$$\tau_{CDUK} + \varepsilon_{CDUK} = (\tau_A + \varepsilon_A) - (\tau_B + \varepsilon_B). \tag{16}$$

The error of a C-DUK is thus $\varepsilon_A - \varepsilon_B$. Since both ε_A and ε_B are less than Δ_{clock}, the error will be greatest when path A's error is nearly Δ_{clock} and path B's error is nearly zero, or vice versa, as formalized in Equation (17). Since an S-DUK is also the difference of two paths, Equation (17) also characterizes its measurement error:

$$-\Delta_{clock} < \varepsilon_{CDUK} < \Delta_{clock}. \tag{17}$$

A similar analysis applies for the M-DUK. The delay of an M-DUK is derived by the sum of two paths minus a third, as Equation (18) demonstrates:

$$\tau_{MDUK} + \varepsilon_{MDUK} = (\tau_A + \varepsilon_A) + (\tau_B + \varepsilon_B) - (\tau_C + \varepsilon_C). \tag{18}$$

Therefore, the error of an M-DUK is $\varepsilon_A + \varepsilon_B - \varepsilon_C$. Since all three terms are less then Δ_{clock}, we can bound the error by the worst case, when either ε_A and ε_B are nearly Δ_{clock} and ε_C is nearly zero, or the other way around, as captured by Equation (19):

$$-\Delta_{clock} < \varepsilon_{MDUK} < 2 \cdot \Delta_{clock}. \tag{19}$$

To validate that computed DUK delays accurately represent exact DUK delays to within the expected error, it would be necessary to know the exact DUK delays. However, we are unable to get this information. Therefore, to certify the accuracy of computed DUK delays, we must calculate a DUK delay two or more ways and confirm that the results are within expected error.

This is easily done, because there are few requirements set on the paths used to compute DUK delays, since there are nearly no restrictions on the subpath prefixes and suffixes represented by π in Section 3.5 and Section 4.2. Consequently, it is possible to formulate multiple sets of paths by changing these π subpaths yet still compute the same DUK. It is worth noting, however, that since every computed DUK must obey the error bounds of Equations (17) or (19), it is possible that one computation will be on one end of the bound and the other on the other end of the bound. As a result, it is necessary to consider not the error between a computed DUK and the actual DUK delay, as was done earlier, but the maximum error between two computed DUKs. We look to Equation (17) and determine that for a C-DUK or an S-DUK, this error is

$\pm 2 \cdot \Delta_{clock}$. For an M-DUK, Equation (19) leads us to conclude it is $\pm 3 \cdot \Delta_{clock}$. Section 6.3 uses these bounds to certify the accuracy of our measurements.

6. EXPERIMENTAL RESULTS

We applied Timing Extraction both to 18 Arrow BeMicro boards that have a Cyclone III FPGA EP3C16F256C8N [Arrow 2009] and to one Terasic DE0-Nano with a Cyclone IV FPGA EP4CE22F17C6N [Altera 2003], modified to allow control over the FPGA's internal V_{dd}. In this section, we present the main results from our measurement experiments on both boards.

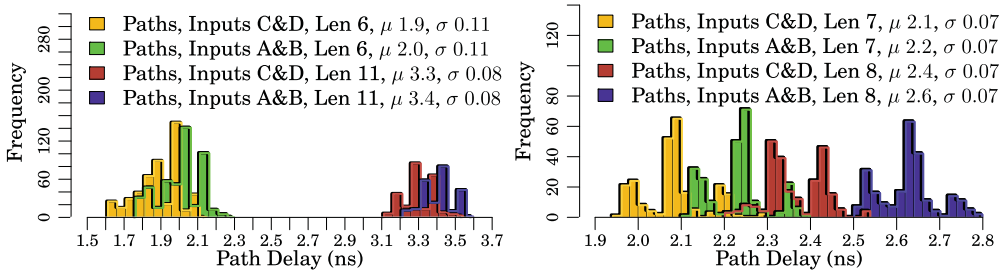
6.1. Methodology

The delay of a path in an FPGA is subject to many sources of variation beyond process variation. These include effects such as CAD tool decisions, local supply voltage IR-drop, crosstalk, and temperature fluctuations. To annul the effects of these variation sources, we perform our measurements in a very structured and systematic way. We divide the FPGA into a control region, where logic required to control the measurement tests is placed on 66 LABs, and a measurement region containing the LABs that will be measured. This keeps the control logic away from the paths under test so that noise effects in the control circuitry will have minimal impact on the measured circuitry. Leveraging the constraints provided by QUIP [Altera 2005a], the placement and routing of all but the LABs being measured are fixed and consistent for all our measurements. This assures us that signal path lengths and compositions are identical across tests and do not directly contribute to the differences in measured delays. QUIP is also used to dictate the placement and routing of the path being measured within a LAB. Moreover, to reduce the overall activity in the FPGA, we do not measure LABs in parallel, but rather measure LABs one at a time. This guarantees that local heating and switching-activity-dependent IR drop do not impact the delay measurements. Furthermore, all measurements are taken in a temperature-controlled room, and we perform our measurement several times to reach a stable internal temperature before recording the final path delay. All these precautions lead to path delays measured in a consistent and precise manner with repeatable results, suggesting that the measurements reveal the underlying process variation and allowing us to compare results between LABs and FPGAs without worry that other variation effects cloud our results.

We use the path measurement technique (Section 2.3) on 18 Cyclone III FPGAs to measure the 2,848 paths per LAB necessary to compute all DUK delays. Each measurement set takes on average 20 minutes per LAB. Due to limitations in the Cyclone III PLLs, for our measurements, we increment the frequency at linear intervals of $\Delta_{clock} = 1.6$ps and, at each frequency, perform 2^{15} path measurements, taking as the delay of the path the frequency that yields a 50% failure rate for that path. With these parameters, we are 99.985% confident that our margin of error is $\pm 1\%$ of 50%. Unless otherwise specified, throughout this section, we present results related to LAB (27,22) of a Cyclone III. Where appropriate, we indicate more general results.

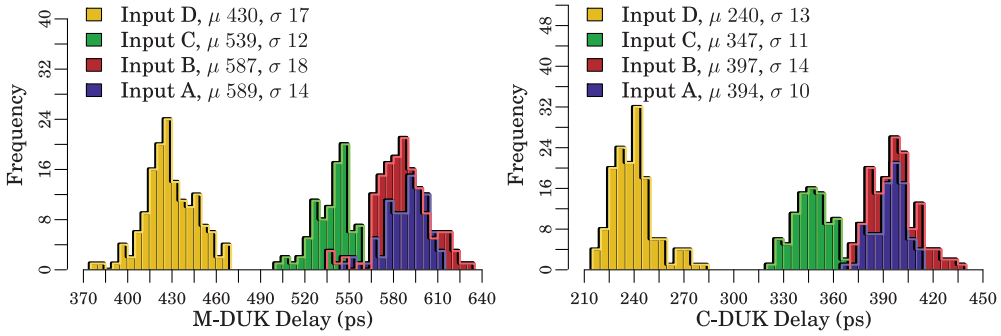
6.2. Extracted Characterization

Figure 9 shows the resulting distribution of the paths measured to compute C-DUKs and M-DUKs in a LAB. We highlight four separate distributions to isolate two sources of known systematic difference, the path length and the LUT inputs used. From these paths, we compute DUK delays; Figure 10 shows these distributions. In this case, the different colors indicate the LUT input used by the DUK. Figure 11 shows the individual delays for each C-DUK over LUT inputs A and B for two different LABs. Note that there is no single delay associated with a LUT; each source–sink pair has a unique delay, both between different DUKs in one LAB and between the same DUK in



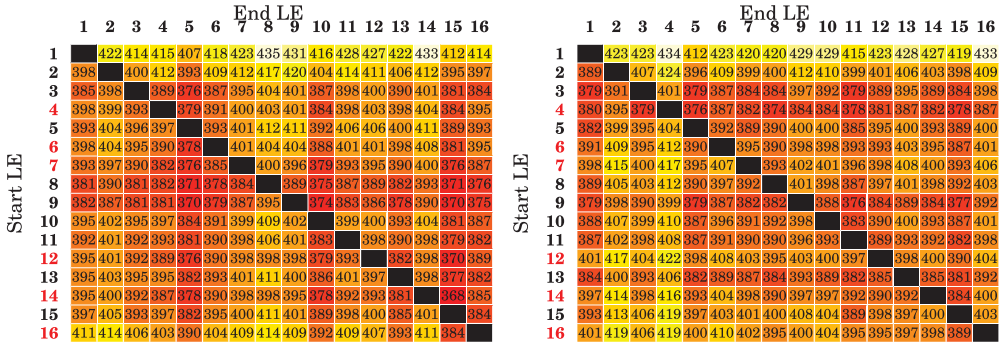
(a) 1,440 paths for M-DUKs (b) 960 paths for C-DUKs

Fig. 9. Path delay distribution for the 2,400 paths required to solve all C-DUKs and M-DUKs, differentiating known systematic variation, Cyclone III LAB (27,22).



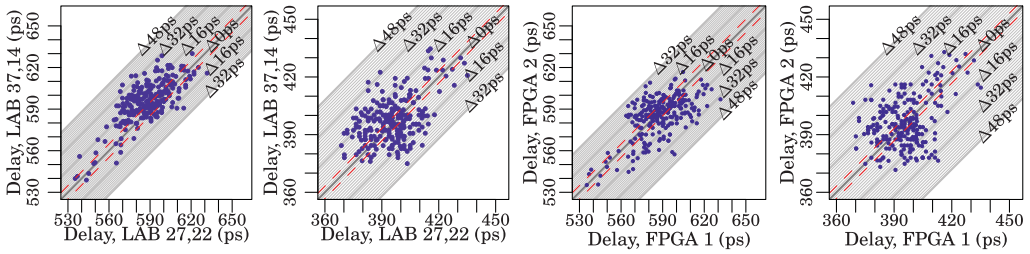
(a) 480 M-DUKs (b) 480 C-DUKs

Fig. 10. C-DUK and M-DUK delay distribution, differentiating known systematic variation, Cyclone III LAB (27,22).



(a) LAB (27,22) (b) LAB (37,14)

Fig. 11. C-DUK delays in picoseconds over LUT inputs *A* and *B*. Rows index start LE of C-DUK; columns index end LE. LUT input *A* is shown by highlighted red row header, *B* otherwise. Two LABs in Cyclone III.



(a) M-DUKs, LAB vs LAB, same FPGA (b) C-DUKs, LAB vs LAB, same FPGA (c) M-DUKs, FPGA vs FPGA, same LAB (d) C-DUKs, FPGA vs FPGA, same LAB

Fig. 12. Correlation between DUKs in two LABs in one FPGA (a & b) and between two FPGAs for the same LAB (27,22) (c & d). Diagonal lines indicate difference between results in terms of $\Delta_{clock} = 1.6$ps. Thicker lines indicate $10 \cdot \Delta_{clock}$. Red dashed lines indicate error boundaries as computed in Section 5. Cyclone III.

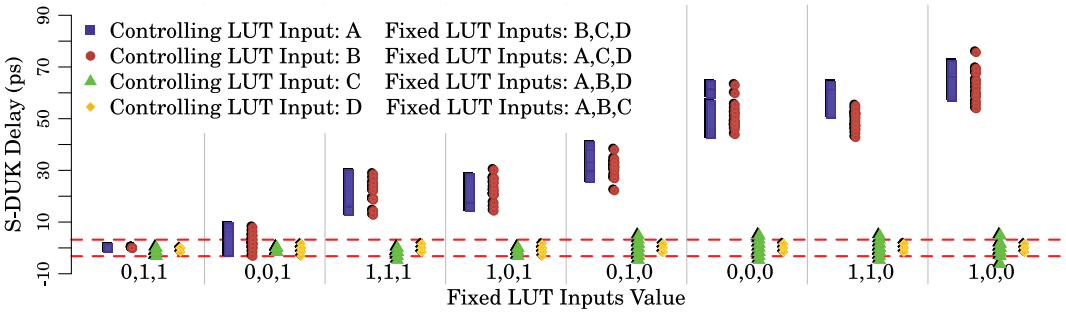


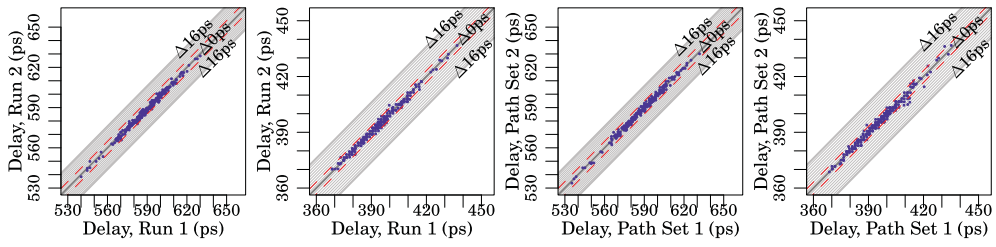
Fig. 13. S-DUK delays. Each scatter plot shows the delay of S-DUKs with the same controlling and fixed LUT inputs. The region between red dashed lines is shown as a reference of the range of the expected error as computed in Section 5. Cyclone III LAB (27,22).

different LABs, demonstrating the importance of accounting for LUT-to-LUT routing. Within a LAB, on average, over all 18 FPGAs, we see a standard deviation of $\sigma/\mu = 3\%$ for M-DUKs and $\sigma/\mu = 5\%$ for C-DUKs.

Figure 12 compares the DUK delay distribution of two LABs in one FPGA and of one LAB in two FPGAs, respectively. The results indicate that the variation is composed of a spatially correlated component, a within-die correlated component, and a random component. If the variation was only correlated, the data points on these graphs would lie on the $\Delta 0ps$ diagonal line. Similarly, if it was all random variation, the data points would resemble Figure 1(c).

For all results presented, when a LUT input is not used to implement the buffer, it is fixed to the vector 0011 for inputs A through D, respectively. S-DUKs allow us to take these results and adjust them for other vectors. Figure 13 shows the S-DUK delays as a scatter plot for each vector, differentiating which input is controlling by the color of the distribution. For example, for the column labeled 0, 1, 1, when A is the controlling input, B, C, and D are fixed at 0, 1, and 1, respectively. When C is the controlling input, A, B, and D are fixed at 0, 1, and 1, respectively.

The main thing to notice is that when inputs A or B are controlling, the value of the other inputs greatly matters, whereas the value of the unused inputs does not matter when C or D are controlling. This is consistent with the expected architecture of the LUT as shown in Figure 3(b). When A or B is controlling, the critical path incurs the full delay of going through the internal LUTs followed by the muxes. Inputs C and D do not have to pay such a high delay.



(a) M-DUKs, Run vs Run (b) C-DUKs, Run vs Run (c) M-DUK Path Set vs Path Set (d) C-DUK Path Set vs Path Set

Fig. 14. Correlation between DUKs when measuring the same paths twice (a & b) and measuring different path sets yielding the same DUKs (c & d). Diagonal lines indicate difference between results in terms of $\Delta_{clock} = 1.6$ps. Thicker lines indicate $10 \cdot \Delta_{clock}$. Red dashed lines indicate error boundaries as computed in Section 5. Cyclone III LAB (27,22).

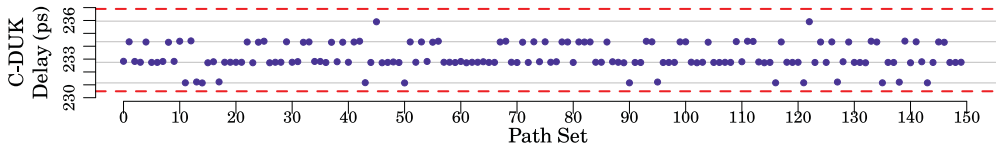


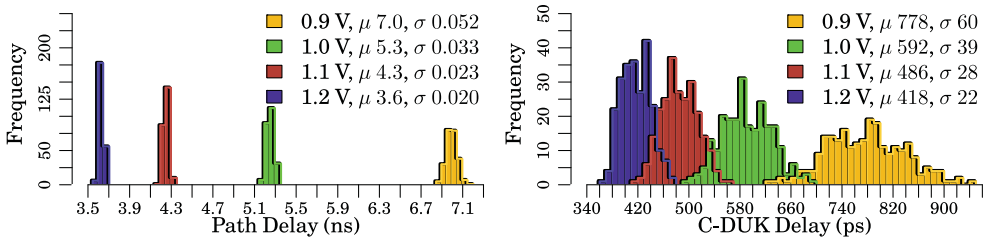
Fig. 15. Delay of C-DUK from LE 6 to LE 11 using input D, computed using 150 different pairs of paths. The distance between horizontal gray lines is $\Delta_{clock} = 1.6$ps. The region between red dashed lines shows expected error bounds as computed in Section 5. All 150 C-DUKs are within these bounds. Cyclone III LAB (27,22).

6.3. Measurement Validation

The measurement of the delay of a path can be subject to many sources of noise; therefore, we would like to build confidence that we are not measuring that noise but rather the actual delay of paths and DUKs in a consistent manner. As explained in Section 6.1, we control as many aspects as possible when performing our measurements. To measure if these controls achieve consistency, we perform the measurements twice by measuring paths, computing all DUK delays, and repeating. Figures 14(a) and 14(b) show the resulting DUK delays when we measure paths twice. Section 5 calculates the expected error due to the granularity of our clock. The red dashed lines in the figure graphically represent the range of this expected error. Therefore, having all measurements fall within this region means the two runs essentially measured the exact same value.

A second form of validation comes from the fact that we can measure distinct sets of paths that allow us to compute the delay of the same set of DUKs. Recall from Section 3.5 that we need two paths to compute the delay of C-DUKs and three for M-DUKs. These paths have a fixed set of LC Nodes that determine which DUK will be computed from their delays, and a subpath prefix of LC Nodes, which we called π, that do not form part of the final DUK. We can select a different set of LC Nodes to use for the subpath π without affecting which DUKs we compute. Figures 14(c) and 14(d) show the resulting DUKs when we compute them using two different sets of paths. Again, the red dashed lines highlight the expected error. Clearly, it matters little what the paths are, as long as they compute the correct DUK.

To gain even more confidence of our measurements, we repeat the experiment, but instead of using two different sets of paths, we use 150 different sets that all yield the same DUK. Figure 15 shows the resulting delay for one C-DUK. All 150 results lie within the expected error. Together, these figures show that we can trust our technique to correctly and consistently compute the delay of DUKs.



(a) Path delay distribution for Length 8 Paths over LUT inputs A and B required to solve C-DUKs

(b) C-DUK delay distribution for LUT inputs A and B

Fig. 16. Delays when varying V_{dd}. Differentiating varying V_{dd}, Cyclone IV LAB (28,22).

6.4. Effects of Varying V_{DD}

Lowering V_{DD} is a common and important way to save power and energy. In this section, we examine the effect that reducing V_{DD} has on variation. In particular, we ask whether scaling V_{DD} has a purely systematic effect on the variation distributions or whether there is a random component as well. To do this, we modify a DE0-Nano board containing a Cyclone IV FPGA so that we can control the internal V_{DD} [Terasic 2011]. Nominally, the board provides a 1.2V V_{DD}. For our tests, we scale at 100mV increments. At $V_{DD} = 0.8$V, a large percent of our measurements fail, and at 0.7V, the board fails to power up.

We know that a lower V_{DD} increases the propagation delay of a circuit, as well as the standard deviation of the path delay distribution [Eisele et al. 1997]. We clearly see this effect in Figure 16(a), the delay distribution for the paths of length 8 used to compute C-DUKs. As we lower V_{DD}, the distribution shifts right and becomes wider. This effect is even more pronounced when we look at the C-DUK delay distributions in Figure 16(b).

7. GENERALIZING TIMING EXTRACTION

Although Section 3 introduces Timing Extraction by applying it to a Cyclone III LAB, the approach generalizes to any FPGA that has registers and configurable PLLs. We can distill the essence of Timing Extraction into five concepts:

(1) We can measure the delay of a group of components in the FPGA using only resources already in the FPGA.
(2) LC Nodes represent the smallest group of components for which we need to compute a delay, since, if we use any component in an LC Node, we must use all other components in the LC Node.
(3) When using the measurement technique from Section 2.3, it is not possible to solve for the delay of every LC Node when a measured path begins at a Start Node, goes through zero or more Mid Nodes, and terminates at an End Node.
(4) When representing all measurable paths as a matrix, there exists a basis that will allow us to compute the delay of any path in the FPGA using only the delay of vectors in that basis.
(5) We can formulate a basis where every vector is a DUK composed of a small linear combination of LC Nodes.

The first, second, and fourth points are immediate; however, it is not obvious why the third and fifth hold true. Although a full explanation, formalization, and proof are beyond the scope of this article, we can build some intuition to address the third point. Consider a simplified circuit that, when represented in LC Nodes, has all paths being

composed by just a Start Node and an End Node. Moreover, there exists a physical path in the circuit formed by combining any Start Node with any End Node. We can represent this situation as a fully connected bipartite graph with Start Nodes forming one set and End Nodes the second. For simplicity, assume that the delay of every path is measured to be 500ps. It is easy to show that at least two solutions to the delay of the nodes exist. One solution assigns a delay of 200ps to all Start Nodes and a delay of 300ps to all End Nodes. The second solution does the opposite, assigning 300ps to Start Nodes and 200ps to End Nodes. A similar circuit with fewer paths suffers from the same problem. Therefore, this circuit, and any subset, leads to an underdetermined system. The argument becomes somewhat more complicated when considering the more general problem, which also includes Mid Nodes; however, the intuition remains the same.

A forthcoming work will address in detail the fifth point. Yet, Section 4 builds much of the intuition necessary to see why this is true. We have three types of DUKs. All three maintain the requirement that a path starts and ends at a register. M-DUKs represent the smallest path between two registers. C-DUKs grow the length of a path. Finally, replacing one node for another is achieved by S-DUKs. This defines the three main types of representations and transformations that should be sufficient to formulate any path in our circuit. Though more than one type of M-DUK, C-DUK, and S-DUK will be defined to cover all structures in the circuit, overall, a small palette of DUKs will suffice to fully describe all structures in an FPGA.

8. FUTURE WORK

The previous section suggests that Timing Extraction is more generally applicable. This article applies Timing Extraction exclusively to the LABs and LUTs. To get the full, intended benefits of this technique, it is essential to also apply Timing Extraction to intercluster routing. Moreover, the results section hints at the existence of different types of variation—systematic, spatially correlated, and random—and shows that Timing Extraction is able to provide the raw information necessary to understand variation in the FPGA. To fully harness the power of Timing Extraction, however, a mathematical analysis of the information it provides should be performed to quantify how much and what kind of variation exists within the FPGA.

Finally, we perform our measurements in a highly controlled setting (Section 6.1). This leads to clean and consistent results, but it is not clear which controls are necessary for good results. Careful experimentation will reveal how the results change when we change or relax the strong restrictions on our measurement technique, allowing us to simplify and accelerate path measurements.

9. CONCLUSIONS

We presented Timing Extraction, a method used to extract the fine-grained delay information necessary to understand variation within the FPGA and to generate component-specific mappings. We acquire this information using only resources already present in the FPGA. Essentially, we apply a launch and capture technique to measure a subset of all paths in the FPGA and extract small DUKs from these measurements. We can then compose DUKs to compute the delay of any path in the FPGA and use them to understand the amount and type of variation present.

We applied this technique to the LABs in both the Altera Cyclone III and Cyclone IV FPGAs. We also measure the intra-LUT delays and demonstrate that our measurements are within the expected margin of error. The results indicate that, on average, we see $\sigma/\mu = 4\%$ variation in the 65nm process used for the Cyclone III. Moreover, there is clear indication that random variation forms a significant part of the total variation. We expect that as we measure smaller technology nodes, the total variation

and the contribution from random variation will increase. By using Timing Extraction, we will be able to characterize and reduce the adverse effects from this increase.

ACKNOWLEDGMENTS

This research was funded in part by National Science Foundation grant CCF-0904577. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors gratefully acknowledge donations of software and hardware from Altera Corporation that facilitated this work, as well as valuable guidance from Mike Hutton and useful pointers from Justin Wong and Joshua Levine.

REFERENCES

Altera. 2003. DE0-Nano Development and Education Board. http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html.

Altera. 2005a. QUIP. http://www.altera.com/education/univ/research/quip/unv-quip.html. (2005).

Altera. 2005b. LCELL WYSIWYG Description for Cyclone II, Altera Corporation.

Altera. 2009. LCELL WYSIWYG Description for Cyclone III, Altera Corporation.

Arrow. 2009. BeMicro Embedded System Lab Instructions. http://www.arrownac.com/offers/altera-corporation/bemicro/BeMicro_Instructions_Embedded_System_Lab.pdf.

Chun Tak Chow, Lai Suen Mandy Tsui, Philip Heng Wai Leong, Wayne Luk, and Steven J. E. Wilton. 2005. Dynamic voltage scaling for commercial FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology* (December 2005), 173–180.

W. Bruce Culbertson, Rick Amerson, Richard Carter, Phil Kuekes, and Greg Snider. 1997. Defect tolerance on the TERAMAC custom computer. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*. 116–123. DOI:http://dx.doi.org/10.1109/FPGA.1997.624611

Martin Eisele, Jorg Berthold, Doris Schmitt-Landsiedel, and Reinhard Mahnkopf. 1997. The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 5, 4 (Dec. 1997), 360–368.

Benjamin Gojman, Nikil Mehta, Raphael Rubin, and André DeHon. 2011. Component-specific mapping for low-power operation in the presence of variation and aging. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*. Springer, Chapter 12, 381–432.

Benjamin Gojman, Sirisha Nalmela, Nikil Mehta, Nicholas Howarth, and André DeHon. 2013. GROK-LAB: Generating real on-chip knowledge for intra-cluster delays using timing extraction. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 81–90.

Scott Hanson, Bo Zhai, Kerry Bernstein, David Blaauw, Andres Bryant, Leland Chang, Koushik K. Das, Wilfried Haensch, Edward J. Nowak, and Dinnis M. Sylvester. 2006. Ultralow-voltage, minimum-energy CMOS. *IBM Journal of Research and Development* 50, 4–5 (July/Sept. 2006), 469–490.

David Lewis, Elias Ahmed, David Cashman, Tim Vanderhoek, Chris Lane, Andy Lee, and Philip Pan. 2009. Architectural enhancements in Stratix-III and Stratix-IV. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. ACM, 33–42.

Xiaochun Li, Jialing Tong, and Junfa Mao. 2010. Temperature-dependent device behavior in advanced CMOS technologies. In *ISSSE*, Vol. 2. 1–4. DOI:http://dx.doi.org/10.1109/ISSSE.2010.5606938

Mehrdad Majzoobi, Eva Dyer, Ahmed Elnably, and Farinaz Koushanfar. 2010. Rapid FPGA delay characterization using clock synthesis and sparse sampling. In *Proceedings of International Test Conference*. DOI:http://dx.doi.org/10.1109/TEST.2010.5699248

Larry McMurchie and Carl Ebeling. 1995. PathFinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 111–117.

Nikil Mehta, Raphael Rubin, and André DeHon. 2012. Limit study of energy & delay benefits of component-specific routing. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 97–106.

Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolic. 1999. *Digital Integrated Circuits* (2nd ed.). Prentice Hall.

Pete Sedcole, Justin S. Wong, and Peter Y. K. Cheung. 2008. Modelling and compensating for clock skew variability in FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*. 217–224. DOI:http://dx.doi.org/10.1109/FPT.2008.4762386

Jack R. Smith and Xia Tian. 2009. High-resolution delay testing of interconnect paths in Field-Programmable Gate Arrays. *IEEE Transactions on Instrumentation and Measurement* 58, 1 (2009), 187–195. DOI:http://dx.doi.org/10.1109/TIM.2008.927212

Ashish Srivastava, Dennis Sylvester, and David Blaauw. 2005. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer.

Edward A. Stott, Justin S. J. Wong, Pete Pete Sedcole, and Peter Y. K. Cheung. 2010. Degradation in FPGAs: Measurement and modelling. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 229–238.

Terasic. 2011. ALTERA Cyclone IV Development & Education Board (DE0-Nano) http://wiki.ntb.ch/infoportal/_media/fpga/boards/de0_nano/de0-nano-schematic.pdf (2011).

Tim Tuan, Austin Lesea, Chris Kingsley, and Steven Trimberger. 2011. Analysis of within-die process variation in 65nm FPGAs. In *Proceedings of the International Symposium on Quality Electronic Design*. 1–5. DOI:http://dx.doi.org/10.1109/ISQED.2011.5770808

Justin S. Wong, Pete Sedcole, and Peter Y. K. Cheung. 2009. Self-measurement of combinatorial circuit delays in FPGAs. *Transactions on Reconfigurable Technology and Systems* 2, 2 (June 2009), 1–22. http://doi.acm.org/10.1145/1534916.1534920

Yun Ye, Samatha Gummalla, Chi-Chao Wang, Chaitali Chakrabarti, and Yu Cao. 2010. Random variability modeling and its impact on scaled CMOS circuits. *Journal of Computational Electronics* 9, 3–4 (Dec. 2010), 108–113. DOI:http://dx.doi.org/10.1007/s10825-010-0336-5

Haile Yu, Qiang Xu, and Philip H. W. Leong. 2010. Fine-grained characterization of process variation in FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*. 138–145. DOI:http://dx.doi.org/10.1109/FPT.2010.5681770

Received May 2013; revised October 2013; accepted January 2014