Appearing in *IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM), May 11–13, 2014

GROK-INT: Generating Real On-chip Knowledge for Interconnect Delays Using Timing Extraction

Benjamin Gojman
Computer & Information Science. University of Pennsylvania
3330 Walnut Street Philadelphia, PA 19104
bgojman@seas.upenn.edu

André DeHon
Electrical & Systems Engineering. University of Pennsylvania
200 S. 33rd St. Philadelphia, PA 19104
andre@acm.org

***Abstract*— With continued scaling, all transistors are no longer created equal. The delay of a length 4 horizontal routing segment at coordinates (23,17) will differ from one at (12,14) in the same FPGA and from the same segment in another FPGA. The vendor tools give conservative values for these delays, but knowing exactly what these delays are can be invaluable. In this paper, we show how to obtain this information, inexpensively, using only components that already exist on the FPGA (configurable PLLs, registers, logic, and interconnect). The techniques we present are general and can be used to measure the delays of any resource on any FPGA with these components. We provide general algorithms for identifying the set of useful delay components, the set of measurements necessary to compute these delay components, and the calculations necessary to perform the computation. We demonstrate our techniques on the interconnect for an Altera Cyclone III (65nm). As a result, we are able to quantify over a 100 ps spread in delays for nominally identical routing segments on a single FPGA.**

I. Introduction

As Moore's Law scaling continues, circuit variation becomes an ever increasing problem that must be overcome to reap the benefit promised by the smaller feature sizes. Conventional techniques used to deal with this problem, such as margining and speed-grading, will become too expensive to practically deal with the enormous variation expected [1].

Reconfigurable circuits are uniquely positioned to better handle large circuit variation, since they allow for fine-grained techniques that carefully configure around this variation to avoid undesired resources, and even take advantage of variation [2], [3]. In [4] we show that by carefully mapping to the variation present in an FPGA, we reduce the energy utilization of the circuit by half. Moreover, our results indicate that a component-specific mapping will be critical in future technologies if the mapped design is to function at all. This approach, however, requires fine-grained knowledge of the underlying circuit variation. In our previous work [5], we introduce Timing Extraction, an effective technique that measures fine-grained circuit variation. The approach takes advantage of the ability to configure a large diverse set of paths in an FPGA. When measured, the delay of these paths provide all the necessary information to calculate the delay of each segment that forms the path. Essentially, we formulate a large system of equations, where each equation, representing the delay of a path, is formed by the sum of variables, representing the delay of a small set of physical devices. Solving this system gives the desired fine-grain variation measurements. Furthermore, the small set of physical devices, called a Discrete Unit of Knowledge (DUK), is carefully selected to easily allow for fine-grained variation analysis and path exploration in conventional routing algorithms [6].

In [5] we limit our application to the logic blocks in the FPGA. However, since most of the area, delay, and energy of modern FPGAs is consumed by the general interconnect, in this paper we present an expanded version of Timing Extraction that extracts fine-grain delay measurements from the whole FPGA, including the general interconnect. We generalize the idea of a Discrete Unit of Knowledge (DUK) by broadening the definition of DUKs. In [5] we present two types of DUKs, the Mother DUK (M-DUK) and Child DUK (C-DUK), each covering a very specific set of physical resources. Here, we define two classes, the M-DUK and C-DUK classes. Each contains the corresponding DUKs from [5], but also allows for more types within the class. Chiefly, a member of the M-DUK class begins every path, while a C-DUK member increases the length of a path. Moreover, as one of the primary contributions of this work, we explain how to algorithmically decompose the complete resource graph of any FPGA into a small number of M-DUKs and C-DUKs and automatically generate which paths should be measured to compute the DUK delays. Knowing their delay, these DUKs can easily be composed to compute the delay of any of the multitudinous paths in the FPGA. What's more our decomposition guarantees that only a small number of DUK types of each class gets generated, providing the information necessary to easily perform fine-grain variation characterization by directly comparing the delay of these DUKs.

Sec. II sets up the concepts required to understand basic circuit variation as well as reviews the version of Timing Extraction presented in [5]. Sec. III explains how to decompose the FPGA resource graph into DUKs. Sec. IV demonstrates this decomposition for a commercial FPGA, and Sec. V presents the actual measurements. We conclude in Sec. VII.

Novel contributions of this work include:

- Technique for systematically decomposing any FPGA into fine-grain units of delay.
  - Identification of small set of types of fine-grain units.
  - Identification of paths required to compute fine-grain unit delays.

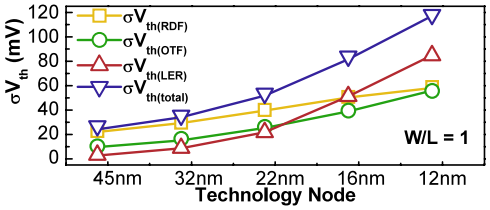


Fig. 1: $\sigma_{V_{th}}$ as a function of technology nodes, based on predictive technology models. Considering the individual effects of random dopant fluctuations (RDF), line edge roughness (LER) and oxide thickness (OTF) from [8]

- First set of measurements to characterize the fine-grain delay of both logic and interconnect.
- Delay measurement of near individual routing segments.
- Demonstration of significant delay contribution from random process variation.

II. Background

A. Process Variation

Manufacturing differences between device parameters leads to differences in device performance and energy requirements. This process variation, historically, has been correlated to a design parameter, such as physical location [2]. As such, most mitigation techniques, including modeling, binning, and biasing [7] focused on correcting these correlated parameter deviations. Unfortunately, with scaling comes a serious increase in uncorrelated, or random process variation (Fig. 1), that cannot easily be reduced simply through these techniques.

To understand how variation affects transistors, we examine the transistor delay and energy equations (Eqs. 1, 2).

$$\tau_{pd} = C_l \cdot \frac{V_{ds}}{I_{ds}} \quad (1) \qquad E_{leak} = I_{ds,sub} \cdot V_{ds} \cdot \tau_{cycle} \quad (2)$$

Both depend on the current, I_{ds}, that, depending on whether we are in saturation or subthreshold are defined by Eqs. 3 and 4 [9], [10].

$$I_{ds,sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - 0.5 V_{d,sat}\right) \quad (3)$$

$$I_{ds,sub} = \frac{W}{L} \eta C_{ox} (n-1) \cdot {v_T}^2 \cdot e^{\frac{V_{gs}-V_{th}}{n \cdot v_T}} \left(1 - e^{\frac{-V_{ds}}{v_T}}\right) \quad (4)$$

These equations include physical parameters such as transistor length, L, and width, W, as well as electrical properties, including threshold voltage, V_{th}. All suffer from random variation, but V_{th}'s exponential relationship to current marks it as the most significant, since a small V_{th} change precipitates a large change in current, considerably changing the energy and timing requirements of the transistor.

B. Altera Cyclone Architecture

Sec. V demonstrates the results of applying Timing Extraction to the Altera Cyclone III FPGA. Therefore, we briefly describe its architecture. The Cyclone III is comprised of a collection of logic array blocks (LABs) embedded in a hierarchical routing network with length 4 and 24 horizontal routing segments and length 4 and 16 vertical routing segments. A small number of embedded memory and multiplier columns provide increased flexibility, while a comprehensive set of I/O pins and PLLs form the periphery.

The LAB consists of 16 logic elements (LEs) composing a 4-LUT and an optional register. Internal connections with 50% depopulations allow for communication among the 16 LEs. An extra set of connections bring signals into the LAB from the length 4 routing segments, as well as from the horizontally adjacent LABs. Outputs from an LE can be routed to horizontally adjacent LABs as well as the immediately surrounding length 4 routing segments [11].

C. Path-Delay Measurement

Timing Extraction depends on measuring the delay of paths in the FPGA. An effective way is by using a launch-capture technique. Starting at a low test frequency, a signal is sent from the launch register, through the path in question, to the capture register. A test is preformed to determine if the signal arrived at the capture register within the allotted test frequency. The test frequency is increased until we find the threshold at which the signal fails to reach the capture register.

To mitigate operational variation such as clock jitter, we measure repeatedly at each test frequency. If at test frequency f the number of failures is a percent of the total measurements, the delay of the path is characterized as $\frac{1}{f}$. Since the transition from no failures to 100% failures is gradual, we select the 50% failure point as the representative delay. Measuring for the 50% failure point 2^{15} times at each test frequency, gives, with 99.98% confidence, a $\pm 1\%$ error. This technique has been successfully used to capture the delay of paths on FPGAs for many applications [5], [12]–[16].

It is worth noting that 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.

Additionally, we isolate the falling delay from the rising delay and record them separately. For consistency and space constraints, in this paper we only present rising delays.

D. Timing Extraction on Logic Clusters

In [5] we introduce the main ideas behind Timing Extraction by applying it to the LABs in the Cyclone III FPGAs. Revisiting the salient points of that work allows us to build upon it and ultimately fully characterize a complete FPGA. We review the idea of a smallest measurable unit, giving rise to the notion of Logical Component Nodes, that form the building blocks for Discrete Units of Knowledge (DUKs).

To non-destructively measure the delay of a path in an FPGA it is necessary to have the resources being measured be part of a bitstream programmed onto the FPGA, regardless of the measurement technique used. Although FPGAs offer great flexibility, when measuring a particular resource, it will be necessary to use other resources near it. E.g., when measuring the delay of a crosspoint connecting two wires, the two wires *must* be part of the bitstream programmed for this measurement. As such, it is not possible to simply measure the delay of the crosspoint, since it is always used with the two wires. Nevertheless, this is not limiting since any time the

crosspoint is used, the two wires will be used as well. Thus, practically there is no need to know the delay of the crosspoint in isolation but rather as part of a unit also including the two wires. This observation can be generalized and gives intuition to the concept of a *smallest measurable unit*, a set of resources that cannot be measured independently from each other.

In this work, as well as in [5], we use the launch-capture approach (Sec. II-C). This method requires measuring combinatorial paths between two registers. Combining this with the idea of a smallest measurable unit, in [5] we identify three types of Logical Component Nodes (LC Nodes). First, the Start Node, consisting of the smallest measurable units that begin at a register. Second, the End Node, representing the smallest measurable units ending at a register. Finally, the Mid Node, encompassing the smallest measurable units that are purely combinatorial, without any registers. With these Nodes, a path measurable by the launch-capture technique is formed by a Start Node followed by zero or more Mid Nodes, ending in an End Node.

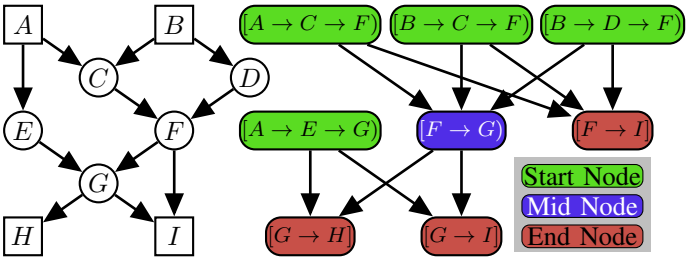
LC Nodes are the ideal candidate for a fine-grain delay measurement of FPGAs. We can form any path in the FPGA from a combination of LC Nodes and can formulate a linear system of equations where each equation represents the delay of a path formed by a number of LC Nodes. Unfortunately, as we previously point out in [5], it is not possible to solve for the delay of all LC Nodes even if all paths are measured.

To overcome this LC Node shortfall, in [5] we define the Discrete Unit of Knowledge (DUK) as small linear combinations of LC Nodes, and demonstrates how a path represented using LC Nodes can very easily be transformed to a DUK-based representation. Moreover, we show that unlike LC Nodes, it is simple to compute the delay of DUKs by measuring the delay of paths. Previously we only present two elements of the M-DUK and C-DUK classes. The member of the M-DUK class is formed by a Start Node plus an End Node, while the C-DUK member is composed of a Mid Node, plus an End Node, minus another End Node. In [5], we identified these DUKs manually. Though sufficient for logic clusters, these two types of DUKs lack the flexibility to represent other FPGA structures. In this work we address this shortcoming by introducing an algorithm to automatically extract a few types of M-DUKs and C-DUKs, that together allow us to represent any FPGA structure.

III. Timing Extraction on General Graphs

Timing Extraction decomposes the resource graph of an FPGA into Discrete Units of Knowledge (DUKs). The DUKs are small enough to provide low level variation information, large enough to be cheaply, easily and distinctly measurable, and practical to use in component-specific mappings.

Here we step through the process required to decompose the FPGA into DUKs and the paths needed to compute DUK delays. We begin by examining the transformation from physical resources to groups of resources forming LC Nodes. In the process we broaden the definition of LC Nodes by expanding the original three LC Node types (Sec. II-D) into three LC Node classes, each containing a handful of types. Building on LC Node classes, DUKs undergo a similar type to class redefinition, allowing us to cover the FPGA resource graph using a handful of distinct DUKs.



(a) Physical Graph (b) LC Node Graph

Fig. 2: Transformation of a physical resource graph, where squares highlight registers, to an LC Node graph, where node names represent encompassed physical resources.

A. LC Node Decomposition

An LC Node consists of a group of connected physical resources that cannot be independently measured. Given a graph of physical resources, we form LC Nodes by identifying paths between un-clocked resources that have fan-in greater than one. Physical resources in the same path are grouped into one LC Node. We repeat the process for paths between registers and un-clocked resources with fan-in greater than one, and similarly from un-clocked resources with fan-in greater than one to registers. LC Nodes are then connected to form an LC Node graph that maintains the relations between nodes, as described by the physical graph [1] (Fig. 2).

Sec. II-D reviews the idea of LC Nodes and mentions the three types induced by the measurement technique. Unfortunately, these three types are only sufficient for representing logic clusters, not general interconnect nor connections between logic cluster. Nevertheless, their form help us identify a more comprehensive LC Node definition. To generalize, any LC Node that begins at a register will be part of the Start Node class, regardless of the type and order of physical resources encompassed by the LC Node. Similarly, LC Nodes that end at a registers will be members of the End Node class. All other LC Nodes belong to the Mid Node class. In this way, our definition of a measurable path still holds, a Start Node followed by zero or more Mid Nodes, ending in an End Node.

Although we can now represent any path in the FPGA using LC Nodes, it is still impossible to solve for the delay of every LC Node, regardless of how many paths we measure. The proof is beyond the scope of this paper, however, in [5] we explain the intuition for why this must be the case. Therefore, we must consider an alternate decomposition to achieve our fine-grain measurement goal. DUKs, being small linear combinations of LC Nodes, are well suited for this.

B. DUK Decomposition

DUKs are small linear combinations of LC Nodes. Although they represent more physical resources than LC Nodes, they

[1] Pseudocode for algorithms available at http://ic.ese.upenn.edu/abstracts/grokint_fccm2014.html

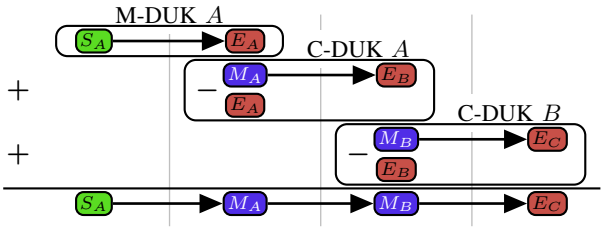


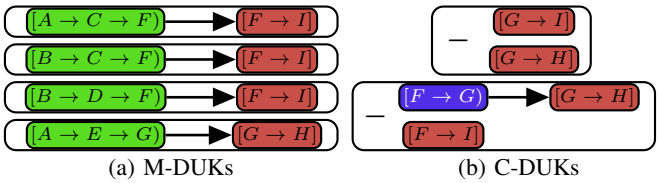
Fig. 3: Demonstration of how an M-DUK plus two C-DUKs leads to a path with the correct form. LC Nodes represent both a set of resources and their delay.

have three distinct advantages. First, we can easily compute their delay from the delay of paths. Second, they still represent a small number of physical resources, small enough to expose fine-grain delay variation. Finally, they are simple to use in conventional routing algorithm since they allow us to incrementally build the delay of paths.

The two DUKs we previously introduce [5] are only sufficient for representing logic clusters. To reach a more general DUK definition, however, it is worth reviewing their function. The Mother DUK (M-DUK), formed by a Start Node plus an End Node, serves as the beginning of any path. In fact, by itself it meets the required form of a measurable path: A Start Node, followed by zero or more Mid Nodes, ending in an End Node. The second DUK, the Child DUK (C-DUK), is used to expand the length of a path. As its name suggests, one or more Child DUKs follow a Mother DUK to form a complete path. To achieve this while maintaining the correct form of a path, the C-DUK is composed of a Mid Node plus an End Node, minus another End Node. As Fig. 3 demonstrate, by selecting and adding together the correct set of DUKs, we can form a complete path and calculate its delay.

A closer examination of what happens in Fig. 3 gives the necessary intuition to understand how DUKs work. Consider M-DUK A and C-DUK A, when combined, C-DUK A has the effect of removing LC Node E_A and replacing it with LC Nodes M_A and E_B. Essentially a C-DUK is composed of two parts, one part that identifies an LC Node that should be removed from the end of a path, and a second part that represents the LC Nodes that should be added to the end. However, a more intuitive way to view this is that a C-DUK extends a path by one Mid Node while making sure the resulting path maintains the correct form of a Start Node, plus zero or more Mid Nodes, plus an End Node. It is this definition of a C-DUK that gives rise to its generalization. For M-DUKs, the intuition is simpler, an M-DUK represents the shortest possible path that maintains correct form.

To decompose the complete FPGA fabric into DUKs we redefine M-DUKs and C-DUKs as classes with many types. The M-DUK class contains a DUK for every Start Node in our LC Graph. Specifically, for each Start Node we create an M-DUK that traces a shortest path from that Start Node to an End Node[1]. In the case of logic clusters, this M-DUK consists of only two LC Nodes, the Start and the End Nodes. However, as we will later see, some M-DUKs will be 3 or 4 LC Nodes long. Nevertheless, all M-DUKs serve the same purpose, to form the beginning of any path in the FPGA.



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

Fig. 4: Decomposition of LC Graph in Fig. 2b into DUKs.

Understanding how C-DUKs are generalized and generated is more complex. As before, C-DUKs still have two parts, the LC Nodes to remove, and the LC Nodes to add. In order for this substitution to work correctly, the first LC Node in the "to remove" part must have the same set of parents as the first LC Node in the "to add" part. In other words, the first nodes in each part must be siblings. Fortunately, the construction of the LC Graph guarantees that any two LC Nodes that are siblings will have the same set of parents, as can be seen in Fig. 2. Therefore, to make sure every resource is accounted for, we must have a C-DUK for each set of sibling LC Nodes.

A C-DUK serves to remove one or more of the LC Nodes at the end of a path, and adds a new set of LC Nodes to that path. Therefore, we must make sure that both parts of the C-DUK begin with siblings and each continues through zero or more Mid Nodes to an End Node. This way we always maintain the form of a path when applying a C-DUK. To achieve this, we search for a shortest path to an End Node from each LC Node that has siblings. Given that, for a pair of sibling LC Nodes, we can effortlessly create a C-DUK by assigning it and its shortest path to one part of the C-DUK, and the other sibling and its shortest path to the other part[1].

To clarify this, we show the results of this decomposition for the LC Graph in Fig. 2b. First, we generate M-DUKs by finding a shortest paths from each Start Node to an End Node, resulting in the 4 M-DUKs in Fig. 4a. Then we identify two sets of siblings, $\{[F \to G), [F \to I]\}$ and $\{[G \to H], [G \to I]\}$. From each pair of siblings, we generate a C-DUK by extending a shortest path from each sibling to an End Node. Fig. 4b shows the resulting 2 C-DUKs. Having these 6 DUKs, we can now combine them to generate any of the 11 paths between registers in the physical resource graph of Fig. 2a.

Thus we can decompose any resource graph into DUKs, and can formulate any path in the FPGA starting with the appropriate M-DUK and adding as many C-DUKs as necessary.

C. DUK Computation

Once we have decomposed a resource graph into DUKs, we need to measure the delay of the correct set of paths so that we can compute the delay of the DUKs. As we will show, it is relatively straight forward to discover what this set of paths should be given the DUKs of a resource graph.

At first sight, the paths required to compute the delay of an M-DUK are trivial to determine since a path that begins with a Start Node, goes through zero or more Mid Nodes, and ends at an End Node is both the definition of a launch-capture measurable path, and what an M-DUK represents. Therefore, we just need to measure one path, the one described by the M-DUK, to directly get the M-DUK's delay.

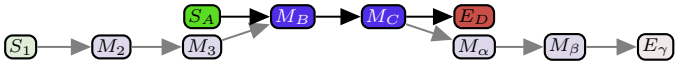
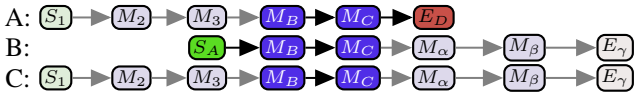


Fig. 5: LC graph structure required to measure the delay of the M-DUK encompassing the path between nodes S_A and E_D

In practice, it is not as simple. By design M-DUKs represent the smallest possible paths that go from a Start Node to an End Node. As we will later see, the length of these paths vary from only two to four LC Nodes. To accurately measure the delay of such small paths, we would need extremely high frequencies for the launch-capture measurement tests. Such high frequencies are not always achievable on commercially available FPGAs, and if they are, they may cause self-heating effects, affecting the delay measure. For example, on the Cyclone III, the maximum frequency is 402.5 MHz, thus the fastest half cycle path we can measure is 1.24 ns. On average, the delay through LUT input D is 236 ps (Fig. 13). As such, the smallest path formed by LUTs using input D that we can measure is 6 LUTs long. Therefore, in case a direct measure is not possible, or would generate too much heat, we present an indirect approach where we measure the delay of three longer paths (e.g., at least 6 LC Nodes long) and use their delay to compute the delay of an M-DUK.

To best understand the relationship between the three paths, we look at an example. Suppose the M-DUK we need to compute consists of the path from LC Node S_A to E_D in Fig. 5. Since that path is too short to directly measure, we compute the delay indirectly. As long as we have an LC graph structure similar to that in Fig. 5 we can measure the delay of the following three longer paths.



Adding the delay of paths A and B, and subtracting from that the delay of C gives the desired M-DUK delay. Fortunately, a structure of this form naturally exists in the LC Graph because of the way LC Nodes are defined. Thus, we can determine the paths needed to get the delay of a given M-DUK, either directly from one path measurement, or indirectly from three, if high enough frequencies are not available.

Tracing the paths needed to compute C-DUK delays is easier. By definition, C-DUKs consist of the difference between two short paths stemming from two sibling LC Nodes, ending at two End Nodes. Their structure implies that to compute their delay we must take the difference of two measured paths. However, the short paths begin with either a Mid Node or an End Node, as shown in Fig. 4b; therefore, we cannot measure them directly. To remedy this, we add LC Nodes to the beginning of the paths so that they begin at Start Nodes. We take advantage of the fact that the first LC Node in each short path are siblings and therefore must have a common parent node. In this way we can extend a path from that common parent node to some Start Node. Thus, when we measure the delay of these two complete paths, and take their difference, we subtract these added Nodes and get only the C-DUK delay.

Fig. 6 shows an example of this. Taking the difference between the paths delays from S_α to E_B, and from S_α to E_C, leads to the delay of the C-DUK outlined in the figure.

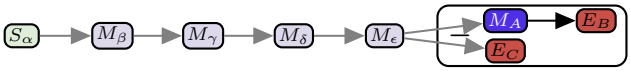


Fig. 6: Example of the LC graph structure required to measure the delay of the C-DUK enclosed in the outline

D. DUK Accounting

We have shown that it takes three path measurements to compute the delay of an M-DUK, and two paths for C-DUKs. In this section, we calculate how many DUKs of each kind to expect from a given resources graph.

The number of M-DUKs equals the number of Start Nodes. The number of Start Nodes depends on the number of registers, and the structure of the resource graph.

C-DUKs arise from pairs of sibling LC Nodes. If n LC Nodes are siblings, forming a sibling set of size n, we will generate $n-1$ C-DUKs from this sibling set. Thus, the number of C-DUKs equals the number of nodes in sibling sets minus the number of sibling sets. The number and size of sibling sets depends on the structure of the resource graph.

IV. Timing Extraction on the Cyclone III

Having explained the general method for decomposing resource graphs into DUKs, we now apply this decomposition to the Altera Cyclone III 65 nm FPGA. Our model for this FPGA assume Logic Cluster Arrays (LABs) containing 16 Logic Elements (LEs) formed by a 4-LUT and an optional register (FF). These LEs can communicate through a set of depopulated connections internal to the LAB called LAB Local Tracks (LLT). Moreover, a set of LAB Input Tracks (LIT) allow signals to enter the LAB. A separate set of LAB Output Buffers (LOB) allow signals to exit the LAB.

The interconnect between LABs consists of segment length 4, unidirectional, horizontal and vertical routing channels (W4). There is also limited direct communication between horizontally adjacent LABs, bypassing the routing network.

Although this model is a subset of the complete Cyclone III, missing the hierarchical segment length 16 and 24 routing channels, along with carry and register chains, it is worth noting that Timing Extraction is agnostic to what the resource graph represents. At its core, Timing Extraction consists of simple graph transformations, minimally differentiating between nodes in the graph. As such, it is possible to treat certain elements as black boxes, or to fully describe a particular component, depending on the level of detail desired, and Timing Extraction will return an appropriate set of DUKs to cover that resource graph.

Displaying either the physical resource graph or the LC graph for this FPGA would require a figure too large and complex to be useful. Instead, we look at the graphs for the resources types. I.e., instead of showing all 4-LUTs, we show one node demonstrating what resources connect to and from 4-LUTs. We can do this because of the high degree of regularity within the FPGA design. In essence, we demonstrate something akin to an FPGA tile.

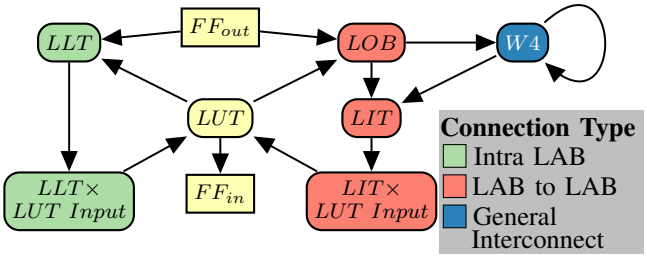


Fig. 7: Cyclone III physical resource graph. Sharp rectangles represent registers.

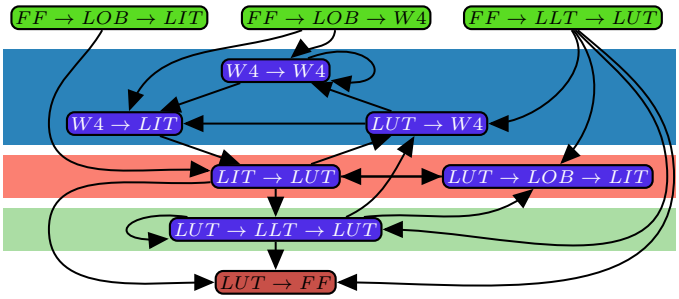


Fig. 8: Cyclone III LC graph generated from Fig. 7. Node names indicate the physical resource path encompassed by LC Node, with enough detail to avoid ambiguity. Top row shows Start Nodes. Bottom row, End Nodes. All others, Mid Nodes. Highlighted background regions follow Fig. 7's key and mark Mid Nodes involved in that connection type.

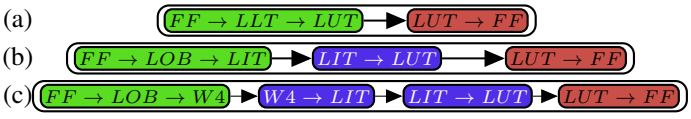


Fig. 9: Cyclone III M-DUKs. (a) Intra LAB. (b) LAB to LAB. (c) General Interconnect

Fig. 7 shows the physical resource graph. For clarity, we highlight the different levels of connections: Within a LAB, between LABs, and through general interconnect. Once our decomposition is complete, we will have all the necessary DUKs to compute the delay of any path going from FF_{out} to FF_{in}. The corresponding LC Graph is shown in Fig. 8. Although more complex, it maintains the separation between different levels of connections. Each Start Node on the top row corresponds to starting a path from a register and continuing either within the LAB, to an adjacent LAB, or to the general interconnect. Furthermore, the Mid Nodes demonstrate the interaction within and between these connection levels.

From Fig. 8 we can generate DUKs applying the decomposition from Sec. III-B. Since we have three types of Start Nodes, we generate three M-DUKs by finding a shortest path from each to an End Node (Fig. 9). Each M-DUK corresponds to one connection type previously distinguished, representing the shortest path that goes from a register (a) to a register in the same LAB, (b) to a register in an adjacent LAB, (c) through one routing resource to a register in some other LAB.

To compute C-DUKs, we look for sibling sets. The LC Graph contains two, {LUT → W4, LUT → LOB → LIT, LUT → LLT → LUT, LUT → FF} and {W4 → W4, W4 → LIT}. From these we compute 4 C-DUKs by selecting pairs of LC Nodes from a sibling set, and finding

Fig. 10: Cyclone III Intra LAB C-DUK

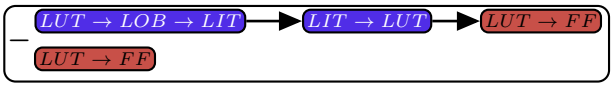


Fig. 11: Cyclone III LAB to LAB C-DUK

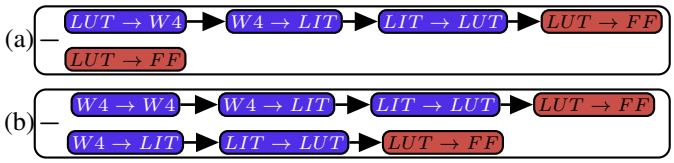


Fig. 12: Cyclone III General Interconnect C-DUKs

a shortest path from each node in the pair to an End Node. Fig. 10 through Fig. 12b show all 4 C-DUKs. For simplicity we have been showing graphs derived from physical resource types, not explicits physical resources. This means that if two LC Nodes in a C-DUK have the same label, they do not represent the same set of physical resource, but rather different sets of the same type of physical resources.

All the C-DUKs allow us to extend a path by one kind of resource or another, at some particular point in the path, and then "patch-up" the path so that it still begins at a Start Node, goes through zero or more Mid Node and terminates at an End Node; i.e., so that the path starts and ends at a register. The C-DUK in Fig. 10 extends the path by using resources in the same LAB as the end register. The C-DUK in Fig. 11 extends by using resources in a LAB adjacent to the LAB holding the end register. Finally both C-DUKs in Fig. 12 extend the path by adding general interconnect routing resources. Where they differ is in the location along the path where they do this.

All three C-DUKs in Figs. 10, 11, and 12a add resources after the last LUT in the path. In contrast, the C-DUK in Fig. 12b adds resources after the last $W4$ routing resources. It is this last C-DUK that allows us to create long routes on the general interconnect, and it is the DUK that best represents the delay of the general interconnect routing resources.

Together, the 3 M-DUKs and 4 C-DUKs can represent any path in the FPGA. By measuring the delay of the paths required to compute the delay of these DUKs, we can compute the delay of any path in the FPGA.

V. Cyclone III DUKs

We coded the DUK decomposition algorithm and gave it the complete physical resource graph for the Cyclone III FPGA, leading to a set of DUKs, and paths for computing the DUKs' delays. Employing the same highly controlled techniques we used in [5], [16], we then measured the delay of paths on 17 Cyclone III FPGAs, and computed the delay of all DUKs. Isolating our control logic from the paths measured allowed us to completely measure a contiguous 8×20 LAB region between LAB coordinate (26,4) and (34,24), including the general interconnect wires connecting all these LABs.

In total, this region is covered by 101,650 M-DUKs and 324,919 C-DUKs and requires close to a million path mea-

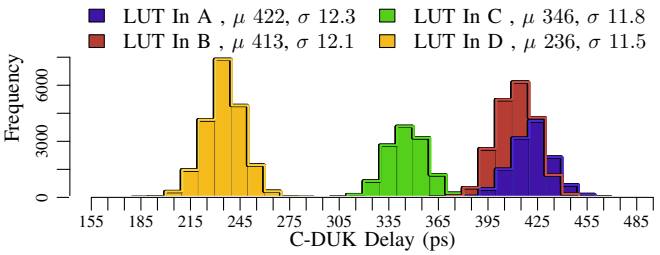


Fig. 13: Intra LAB C-DUK delay distribution, differentiating known systematic differences, 160 Cyclone III LABs.

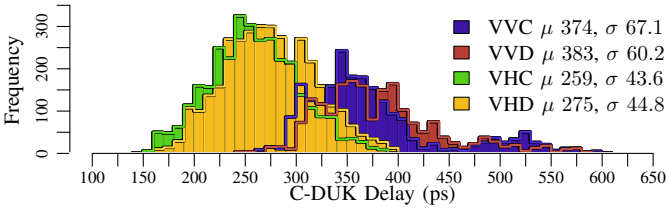


Fig. 14: CAD delay, general interconnect C-DUK.

surements. On average, this means we measure 6,000 paths to compute 2,600 DUKs per LAB. Since the DUKs are of bounded length, the per-LAB DUK and path count will converge to a constant only slightly larger than this as we scale to larger regions and larger chips. Even though we measure a million paths, we can pack many together and only need 230,231 bitstreams to measure all. After packing, the total number of bitstreams needed will also converge to a constant independent of the total number of LABs on the die.

76,800 of the C-DUKs are of the type we introduced in [5] with form shown in Fig. 10. Fig. 13 shows these C-DUK delays and is consistent with our previous results (Fig. 9 of [5]). However, here we plot measurements from 160 LABs instead of just one.

For the remainder of our results we focus on the C-DUK in Fig. 12b. This C-DUK best represents the delay of a general interconnect resource and is among the major contributions of this work. This resource can be a horizontal or vertical segment, preceded by either a horizontal or vertical segment. Moreover, both parts of the C-DUK end in a register by going through one of four LUT inputs. To prevent systematic variation due to these choices, we limit our results to four pairings: either vertical or horizontal segment preceded only by vertical segments (VH or VV), and either LUT input C on both parts of the C-DUK, or LUT input D (C or D). Also, we only compare delays within one pairing. Each pairing has the following number of C-DUKs. VVC: 2,117, VVD: 1,918, VHC: 3,213, VHD: 3,524. For all graphs, we differentiate between pairings using colors and unless specified, we plot delays for one consistent FPGA out of the 17 measured.

Fig. 14 demonstrates the delay of these C-DUKs as computed by Quartus 11.0 [17]. Although all C-DUKs in one pairing use the same type of resources, the CAD tools show a wide spread. We delve deeper into this and examine only the 573 VVC resources that connect two resources exactly 4 vertical LABs away, further removing a potential systematic

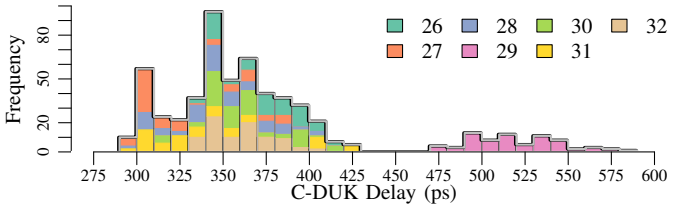
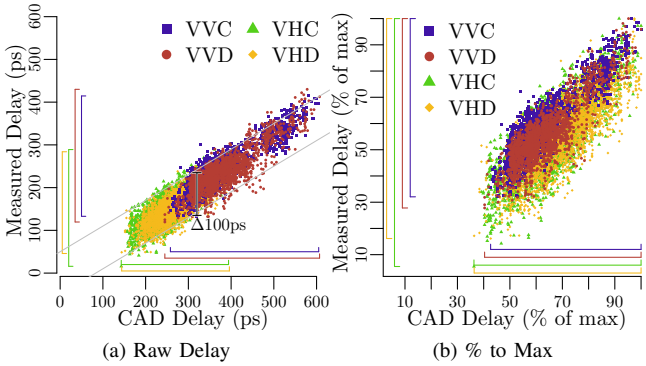


Fig. 15: CAD delay of VVC C-DUK resources connecting two resources exactly 4 vertical LABs away. Differentiating column coordinate of resource.



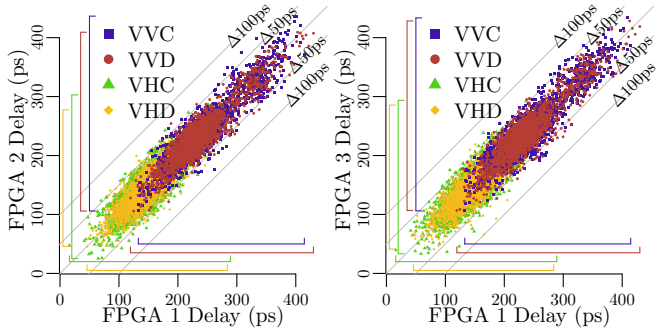
(a) Raw Delay (b) % to Max

Fig. 16: Correlation between CAD and Measured general interconnect C-DUK. Δ 100 ps bar highlights 100 ps measured difference among resources the CAD tools claim have the same delay.

difference. Fig. 15 plots these resources, differentiating the column coordinate of the vertical routing resource. Immediately we see that column 29 is systematically slower than all the rest. However, even for one coordinate column, we see a very large delay spread. This indicates that the CAD tools are aware of even more systematic differences between these resources. In fact, we explored 17 other differentiating dimensions, such as wire index or row coordinate, and discovered that no one dimension is sufficient to explain this spread, but rather a careful combination of all is necessary.

We turn now to measured DUK delays and compare these to the CAD tool delays in the correlation plot of Fig. 16a. Perfect correlation would show all points lining up at a tight diagonal line and would indicate that the CAD tools are aware of all process variation. Instead, we see some correlation as a trending diagonal but clearly observe a thick spread. This means that two C-DUKs with the same CAD delay can have widely different measured delays—a clear indication of the existence of random variation. The $\Delta 100$ ps bar marks an example of this. Normalizing the measured and CAD distributions to their corresponding maximum delay yields Fig. 16b and better demonstrates this effect. We further uncover random variation by plotting the correlation between actual C-DUK delays of different FPGAs (Fig. 17).

Our results clearly demonstrate significant contribution from random variation, both within the same FPGA, and across FPGAs. Moreover, though the CAD tools are aware of vast systematic variation, the magnitude of random variation is large enough to cause delay inversions where the CAD delay of resource a is greater than b, but the measured delay of a is



(a) FPGA 1 vs 2 (b) FPGA 1 vs 3

Fig. 17: Correlation between different Measured FPGAs, general interconnect C-DUK.

less than b (Fig. 16). Even more, we see these delay inversions between the measured DUKs of two FPGAs (Fig. 17), further demonstrating that these inversions are caused by random variation. DUKs cheaply and easily allow us to see these inversions, and component-specific mapping [4] gives us the ability to adjust to the DUKs; no other variation mitigation technique has such power. It is telling that even on a 65 nm process, random variation is large enough to easily observe. With scaling, it will be larger and more apparent, necessitating a component-specific mapping to extract the full potential of the FPGA, and at the extreme, to prevent variation-induced failures. DUKs are the key enablers for this component-specific mapping, and their delays are cheaply and easily computed with only resources already present on FPGAs.

VI. Future Work

Automating Timing Extraction on an arbitrary resource graph successfully provides fine-grain delay information from any FPGA. However, to extract the power of this information we need to perform a comprehensive analysis that differentiates among spatially correlated, systematically correlated, and random variation. Moreover, applying Timing Extraction to devices fabricated on smaller technology nodes will allow us to observe greater random variation. Finally, though our path packing allows us to measure multiple paths per bitstream, we do so under highly controlled conditions. Understanding which of these measures are necessary and which can be relaxed, will improve the performance of our measurement algorithms.

VII. Conclusions

Timing Extraction takes the resource graph of any FPGA and decomposes it into individual DUKs. Furthermore, it indicates the two or three paths that must be measured to compute the delay of a DUK. The decomposition is both simple and general, and will provide as much detail as is represented in the resource graph. The DUKs represent a small number of physical resources, and their consistent shape allows us to directly compare them to characterize the process variation of the FPGA. Moreover, they readily compose to provide the incremental delay of a path, making them ideally suited for use during a component-specific mapping.

We applied Timing Extraction to a region of 160 LABs in 17 Cyclone III FPGA. Measuring, on average 6,000 paths per LAB yields enough information to compute DUKs, and with them, the delay of any path in that region. We observe random variation in the delay of our DUKs ranging on the order of 100 ps, and see delay inversions within an FPGA, between CAD tool and measured delays, as well as between FPGAs.

Acknowledgments

This research was funded in part by National Science Foundation grant CCF-0904577 and DARPA/CMO contract HR0011-13-C-0005. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the official policy or position of the National Science Foundation, Department of Defense, or the U.S. Government. The authors gratefully acknowledge donations of software and hardware from Altera Corporation.

References

[1] "International technology roadmap for semiconductors," <http://www.itrs.net/Links/2012ITRS/Home2012.htm> , 2012.

[2] T. Tuan, A. Lesea, C. Kingsley, and S. Trimberger, "Analysis of within-die process variation in 65nm FPGAs," in *ISQED*, March 2011, pp. 1–5.

[3] Z. Guan, J. S. J. Wong, S. Chaudhuri, G. Constantinides, and P. Y. K. Cheung, "Exploiting stochastic delay variability on FPGAs with adaptive partial rerouting," in *ICFPT*, 2013, pp. 254–261.

[4] N. Mehta, R. Rubin, and A. DeHon, "Limit Study of Energy & Delay Benefits of Component-Specific Routing," in *FPGA*, 2012, pp. 97–106.

[5] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon, "GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays using Timing Extraction," in *FPGA*, 2013, pp. 81–90.

[6] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *FPGA*, 1995, pp. 111–117.

[7] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III and Stratix-IV," in *FPGA*, 2009, pp. 33–42.

[8] Y. Ye, S. Gummalla, C.-C. Wang, C. Chakrabarti, and Y. Cao, "Random variability modeling and its impact on scaled CMOS circuits," *J. Comput. Electron.*, vol. 9, no. 3-4, pp. 108–113, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10825-010-0336-5

[9] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 2nd ed. Prentice Hall, 1999.

[10] S. Hanson, B. Zhai, K. Bernstein, D. Blaauw, A. Bryant, L. Chang, K. K. Das, W. Haensch, E. J. Nowak, and D. M. Sylvester, "Ultralow-voltage, minimum-energy CMOS," *IBM J. Res. and Dev.*, vol. 50, no. 4–5, pp. 469–490, July/September 2006.

[11] Altera, "Cyclone III Device Handbook Volume I," p. 348, 2011. [Online]. Available: http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf

[12] J. R. Smith and X. Tian, "High-Resolution Delay Testing of Interconnect Paths in Field-Programmable Gate Arrays," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 1, pp. 187–195, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4559395

[13] P. Sedcole, J. S. Wong, and P. Y. K. Cheung, "Modelling and compensating for clock skew variability in FPGAs," *ICFPT*, pp. 217–224, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4762386

[14] J. S. Wong, P. Sedcole, and P. Y. K. Cheung, "Self-measurement of combinatorial circuit delays in FPGAs," *ACM Tr. Reconfig. Tech. and Sys.*, vol. 2, no. 2, pp. 1–22, June 2009. [Online]. Available: http://doi.acm.org/10.1145/1534916.1534920

[15] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, "Rapid FPGA delay characterization using clock synthesis and sparse sampling," in *Proc. Intl. Test Conf.*, 2010.

[16] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon, "GROK-LAB: Generating real on-chip knowledge for intra-cluster delays using timing extraction," *ACM Tr. Reconfig. Tech. and Sys.*, *Accepted—to appear*.

[17] Altera, "Quartus II Handbook," p. 1681. [Online]. Available: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf

APPENDIX A
TIMING EXTRACTION ALGORITHMS

Algorithm 1: LC Node Decomposition

Input: Resource Graph G_r
Output: LC Node Graph G_{lc}
$G_{comb} \leftarrow$ `RemoveRegisters`(G_r)
foreach *node* $n \in G_{comb}$ **do**
  if `Fanin` $(n) > 1$ **then**
    l.`Add`(n)
l.`AddAll`(`GetRegisters`(G_r))
$lcNodes \leftarrow$ `FindAllPathsBetweenNodes`(l)
foreach *Pair of LC nodes* (x, y) *from* $lcNodes$ **do**
  if $x.lastResource == y.firstResource$ **then**
    x.`AddChild`(y)
G_{lc}.`AddAll`$(lcNodes)$

Algorithm 2: DUK Decomposition

Input: Resource Graph G_{lc}
Output: A set of DUKs D
`MarkCanonicalChild` (G_{lc})
$D \leftarrow$ `ExtractM-DUKs`(G_{lc})
$D \leftarrow$ `ExtractC-DUKs`(G_{lc})

Function Mark Canonical Child

Input: Resource Graph G_{lc}
Queue $q \leftarrow$ `GetEndNodes`(G_{lc})
while ! `Empty` (q) **do**
  $noden \leftarrow q$.`Dequeue`()
  foreach *node* $p \in$ `Parents`(n) **do**
    p.`SetCanonicalChild`(n)
    if ! `Visited` (p) **then**
      q.`Enqueue`(p)
  `MarkVisited` (n)

For each node in the LC Graph, this function identifies the child with the shortest distance to an End Node and marks it as the canonical child.

Function Extract M-DUKs

Input: Resource Graph G_{lc}
Output: A set of M-DUKs MD
foreach *node* $s \in$ `GetStartNodes`(G_{lc}) **do**
  MD.`Add`(`GetCanonicalPath`(s))

For each Start Node, this function creates an M-DUK by extracting the Start Node's canonical path.

Function Extract C-DUKs

Input: Resource Graph G_{lc}
Output: A set of C-DUKs CD
$siblingSets \leftarrow$ `GetSiblingSets`(G_{lc})
foreach *set* $ss \in siblingSets$ **do**
  $path1 \leftarrow$ `GetCanonicalPath`$(ss[1])$
  for $i \leftarrow 2$ *to* `Size`(ss) **do**
    $path2 \leftarrow$ `GetCanonicalPath`$(ss[i])$
    CD.`Add`$(path1, path2)$

For each sibling set, this function extracts the canonical path from the first LC Node in the set. Then, for every other LC Node in that sibling set, it creates a C-DUK using that LC Node's canonical path and the canonical path of the first LC Node, previously extracted.

Function Get Canonical Path

Input: LC Node n
Output: Canonical Path p
$nextNode \leftarrow n$
while !`IsEndNode`$(nextNode)$ **do**
  p.`Add`$(nextNode)$
  $nextNode \leftarrow nextNode$.`GetCanonicalChild`()
p.`Add`$(nextNode)$

For a given LC Node n, this function returns the shortest path from n to an End Node as traced by following the canonical children.