GROK-FPGA: GENERATING REAL ON-CHIP KNOWLEDGE FOR FPGA FINE-GRAIN DELAYS USING TIMING EXTRACTION

Benjamin Gojman

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2014

Acknowledgements

First and foremost I have to thank my advisor, André DeHon. His support and motivation, infinite patience and guidance made this work possible. I have been extremely fortunate to have him as an advisor and friend. Thank you.

I also want to thank my committee members for their advice and feedback: Daniel Koditschek, Ali Jadbabaie, Insup Lee, and Mike Hutton.

The members of the IC Lab helped shape this work through discussions and feedback. In particular I want to thank Raphael Rubin, Nikil Mehta, Sirisha Nalmela, and Nicholas Howarth. Sirisha was instrumental in the developments of the Verilog used for this work. Without the work of Nick H. the experiments varying supply voltage would not have been possible. Nick M. developed the infrastructure to run millions of measurements, however, of most value was the insightful discussions we had developing the ideas presented here. Finally, Rafi's support, both technical and intellectual, was invaluable to me. Thank you all.

I want to thank my family Marcos and Karen, Mauricio, Tanya and Gabriel, and Monica Gojman for their love and encouragement, and Dita for knowing, with a pointed finger, that this work would come to a successful end.

Finally, I want to thank Emily Traver. Emily has been with me through this whole process, delighting in the ups and never failing to be there when things got rough. It has been a long road, and I am ever grateful you were with me through every step. Your boundless love and support made finishing this work possible.

ABSTRACT

GROK-FPGA: GENERATING REAL ON-CHIP KNOWLEDGE FOR FPGA FINE-GRAIN DELAYS USING TIMING EXTRACTION

Benjamin Gojman

André DeHon

Circuit variation is one of the biggest problems to overcome if Moore's Law is to continue. It is no longer possible to maintain an abstraction of identical devices without huge yield losses, performance penalties, and energy costs. Current techniques such as margining and grade binning are used to deal with this problem. However, they tend to be conservative, offering limited solutions that will not scale as variation increases. Conventional circuits use limited tests and statistical models to determine the margining and binning required to counteract variation. If the limited tests fail, the whole chip is discarded. On the other hand, reconfigurable circuits, such as FPGAs, can use more fine-grained, aggressive techniques that carefully choose which resources to use in order to mitigate variation. Knowing which resources to use and avoid, however, requires measurement of underlying variation.

We present Timing Extraction, a methodology that allows measurement of process variation without expensive testers nor highly invasive techniques, rather, relying only on resources already available on conventional FPGAs. It takes advantage of the fact that we can measure the delay of logic paths between any two registers. Measuring enough paths, provides the information necessary to decompose the delay of each path into individual components—essentially, forming a system of linear equations. Determining which paths to measure requires simple graph transformation algorithms applied to a representation of the FPGA circuit. Ultimately, this process decomposes the FPGA into individual components and identifies which paths to measure for computing the delay of individual components.

We apply Timing Extraction to 18 commercially available Altera Cyclone III (65 nm) FPGAs. We measure 22×28 logic clusters and the interconnect within and between cluster. Timing Extraction decomposes this region into 1,356,182 components, classified into 10 categories, requiring 2,736,556 path measurements. With an accuracy of ± 3.2 ps, our measurements reveal regional variation on the order of 50 ps, systematic variation from 30 ps to 70 ps, and random variation in the clusters with $\sigma = 15$ ps and in the interconnect with $\sigma = 62$ ps.

Contents

List of Tables

List of Figures

Glossary

Timing Extraction

Child DUK (C-DUK)

The difference of two short LC Node path tails. Extends and modifies paths represented with DUKs

Current Tail

The subtrahend of a C-DUK. Representing the tail a path must match for this C-DUK to be applicable

Desired Tail

The minuend of a C-DUK. Representing the path tail that will remain after the C-DUK is used

Discrete Unit of Knowledge (DUK)

Small linear combination of LC Nodes

End Node

An LC Node whose last physical resources is a register. All other physical resources are combinational

LC Node graph

A graph representing some physical resource graph using LC Nodes

Logical Component Node (LC Node)

Group of connected physical resources that cannot be independently measured. Represents short path from a source register node, or node with fan-in greater than one, to a sink register node, or node with fan-in greater than one

Marked Path

An annotated path from an LC Node to its closest End Node

Marked Sibling

An LC Node whose incoming edges are all annotated or marked

Mid Node

An LC Node whose physical resources are all combinational

Mother DUK (M-DUK)

A shortest path from a Start Node to an End Node. Forms the beginning of any path represented with DUKs

Physical Resource Graph

Graph of the physical resources of a circuit

Sibling Set

A set of LC Nodes sharing the same set of LC Node parents in an LC Node graph

Start Node

An LC Node whose first physical resources is a register. All other physical resources are combinational

Timing Extraction on Commercial FPGAs

4-LUT

4-input lookup table. Capable of implementing any 4-input boolean function

C16 bundle

In Cyclone III, group of directed vertical routing wires spanning 16 LABs

C4 bundle

In Cyclone III, group of 24 directed vertical routing wires spanning 4 LABs

Embedded Column Neighbor C-DUK

Cyclone III C-DUK interchanging the last W4 of a path with another W4. Applicable only to paths ending at a LAB next to an embedded column

FF$_D$

Input of a register

FF$_Q$

Output of a register

General Interconnect C-DUK

Cyclone III C-DUK extending the end of a path through a W4 wire

General Interconnect Sink Select C-DUK

Cyclone III C-DUK selecting LAB sink of a W4 wire

Intra-LAB C-DUK

Cyclone III C-DUK extending the end of a path to another LE within the same LAB as the end of the original path

Intra-LAB M-DUK

Cyclone III M-DUK representing a path between two LEs in one LAB

LAB Input Track (LIT)

Brings external signals into a LAB

LAB Local Track (LLT)

Enables direct communication between Logic Elements within one LAB

LAB Output Buffers (LOBs)

Allows signals to leave a LAB

LAB-to-General Interconnect C-DUK

Cyclone III C-DUK changing the end of a path from a horizontally adjacent LAB to a W4 wire

LAB-to-LAB C-DUK

Cyclone III C-DUK extending the end of a path to an LE in a horizontally adjacent LAB

LAB-to-LAB M-DUK

Cyclone III M-DUK representing a path from an LE in one LAB to an LE in a horizontally adjacent LAB

LIT Crosspoint Swap C-DUK

Cyclone III C-DUK swapping LIT crosspoint used at the end of a path

LLT Crosspoint Swap C-DUK

Cyclone III C-DUK swapping LLT crosspoint used at the end of a path

Logic Array Block (LAB)

The logic block of the Altera Cyclone III FPGA

Logic Element (LE)

A small compute block, for the Cyclone III consisting of a 4-LUT and optional register

R24 bundle

In Cyclone III, group of directed horizontal routing wires spanning 24 LABs

R4 bundle

In Cyclone III, group of 34 directed horizontal routing wires spanning 4 LABs

Subspecies Difference (SD)

A minor morphological difference within a DUK type. For example, the LUT input SD differentiates which LUT input a DUK uses

W4

group of directed routing wires spanning 4 LABs, applicable in vertical or horizontal direction

Variation Analysis

Random Regional Variation $Reg_{rand}(x, y)$

Regional variation unique to one FPGA

Random Variation $Rand$

The variation that cannot be explained by any correlated variation, quantifying the random variation in a DUK

Regional Variation $Reg(x, y)$

A regional correlation function that explains how the DUK delay changes as a function of its physical position (x, y)

Systematic Regional Variation $Reg_{sys}(x, y)$

Regional variation that is common to all FPGAs

Systematic Variation $Sys(sd_1, \ldots, sd_n)$

A systematic correlation function composed of the product of a set of functions that correlate delay changes to subspecies differences

τ_0

A base delay for a given DUK type

τ_{FPGA}

The delay bias of an FPGA, relative to τ_0

(# Chapter 1

Introduction

1.1 Thesis

The resource graph of an FPGA is algorithmically decomposed into discrete units. Using only resources within the FPGA, the delay of each of these units is computed by measuring and linearly combining the delay of two or three paths within the FPGA. These unit delays reveal the magnitude and composition of process variation within the FPGA, and provide the necessary knowledge to perform component-specific mappings to mitigate adverse process variation effects.

1.2 Motivation

Circuit process variation is quickly becoming one of the biggest problems to overcome if Moore's Law is to continue. It is no longer possible to maintain an abstraction of identical devices without incurring huge yield losses, performance penalties, and high energy costs. Current techniques such as margining, where the circuits are rated to operate slow enough to capture the worst-case transistor on a billion-transistor chip operating in worst-case conditions, and grade binning, which classifies circuits into a limited set of performance classes, are used to deal with this problem. However, they tend to be too conservative, only offering a limited solution that will not scale as variation increases. Conventional circuits use statistical models and a limited number of expensive tests to determine the margining and binning parameters needed to counteract variation. If the limited tests give a negative prognosis, the whole chip is discarded. On the other hand, reconfigurable circuits, such as FPGAs, can use more fine-grained and aggressive techniques that carefully choose which resources to use in order to mitigate the adverse variation effects. These require measurement of the underlying variation, which many considered too expensive to be practical. However, we present Timing Extraction, a methodology that allows us to measure the process variation due to manufacturing, without the need for expensive testers nor highly invasive techniques, rather, relying only on resources already available on conventional FPGAs.

Modern FPGAs have billions of transistors, each with a unique set of characteristics. Some exhibit regional variation such as differences in oxide thickness, while other effects, such as dopant density, present themselves as stochastic variation. Beyond manufacturing process variation, circuits suffer from environmental and operational variation, such as self-heating, and aging effects. In order to prevent these sources of variation from leading to a computational fault, manufacturers currently rate the performance of their circuits well below what the wires and transistors composing them are capable of delivering. In this way, they margin against all variation sources. Timing Extraction allows us to measure the process variation and, with that information, reclaim the process margins added by vendors. Reducing margins leads to FPGAs with improved performance and energy efficiency. Furthermore, as more and more, smaller and smaller devices are integrated into one circuit, it is essential to fully understand the variation in the circuit if we are to successfully utilize it and extract its full capacity, otherwise we run the future risk of variation overwhelming circuits to the point of complete yield loss.

1.3 Timing Extraction

It is not possible to directly measure the characteristics of every transistor or wire in an FPGA. Nevertheless, we introduce a novel technique that discovers the individual characteristics. Utilizing only the logic available in the FPGA, we perform a careful set of tests and use the results from those tests to discover the variation of each component.

In an FPGA, we can measure the delay of a path without the need of any extra testers by surrounding it with two registers which are already part of the reconfigurable fabric. For convenience we will label them the launch and capture registers. Starting with a low clock frequency, we send a signal from the launch register, through the path, to the capture register. If, in the alloted clock time we see the signal at the capture register, then we know the path is faster than the current clock frequency. If the signal does not appear on the capture register in time, then the path is slower. By adjusting the frequency of the clock and repeating this process it is possible to precisely measure the path delay [93, 58].

The path is composed of multiple components, our goal is to discover the variation of each component. By configuring the correct set of overlapping paths and measuring, we can setup a system of equations that, when solved, gives the individual delay of each component in the paths [30]. A simple example will give better intuition as to what the technique actually accomplishes.

Consider we measure three paths. Path 1 composed of component A and B. Path 2, B and C. Finally, Path 3, C and A. Suppose the delays of the paths are 5ps, 4ps and 3ps respectively. That leads to the system of equations below:

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

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. Timing Extraction takes as input a directed graph representation of the resources in an FPGA. After a series of graph transformations, it decomposes the FPGA into components formed by a small number of resources. Based on this decomposition, it extracts and measures the delay of a set of paths, and uses the path delays to compute the delay of each component.

Once we have the component delays, we can classify the type of variation present into *regional variation*, where variation is correlated to a physical region of the FPGA, *systematic variation*, where unique properties of the component determine the variation, such as the direction of a wire, and *random variation* brought about by the stochastic nature of some of the underlying processes used during fabrication. Moreover, we can analyze and separate variation common across FPGAs from variation unique to a particular FPGA. Quantifying how much variation there is and what type it is allows FPGA CAD tools to produce improved logic mapping results, including a component-specific mapping tailored to the FPGA being programmed [89, 33, 63]. Furthermore, this knowledge may be a requirement if we are to successfully use future technologies [28]. Finally, understanding variation at this detailed level, over such a vast number of devices can prove invaluable to circuit design in general.

1.4 Overview

The remainder of this dissertation is organized as follows. Chapter 2 presents the background necessary for the rest of this work. It begins by giving a brief review of reconfigurable logic and FPGAs in particular. Then, it sets up the relationship between physical properties, such as a transistor's length, and the electrical characteristics of wires and transistors. With this base, it relates the physical effects variation has on devices to the electrical changes it induces. Finally it enters into a short discussion on current techniques used to manage variation.

Chapter 3, details how Timing Extraction works. It begins by developing the graph transformation algorithms necessary to decompose the resource graph of an FPGA into individual components. It then explains how to extract the set of paths necessary to compute the delay of each component. Using the delay of the individual components, it constructs a new circuit graph that routing algorithms can use to find shortest paths unique to that FPGA. Finally, it calculates the expected measurement precision.

Applying Timing Extraction to an Altera Cyclone III FPGA is the subject of Chapter 4. We first present the architecture of this FPGAs. We ground the decomposition algorithms from the previous chapter by examining the result of applying them to this architecture. We then consider the extra CAD steps necessary to actually measure the paths determined by the algorithm. After explaining our experimental setup, we present the delays for the different types of measured components. The chapter then looks at a number of results enabled by knowledge of these component delays. It concludes by quantifying how long the measurement takes, and showing that we can consistently measure the same component delay multiple times.

In Chapter 5 we present the techniques to separate the raw process variation into correlated and random variation. Throughout the chapter, we apply these techniques to the results obtained in Chapter 4, quantifying the magnitude and types of variation in the Cyclone III FPGAs.

Timing Extraction provides a complete approach to decomposing and measuring the process variation of an FPGA; however, there is work that can improve its performance. Furthermore, the decomposition in Chapter 5 asumes a simplifed timing model, a more precise model would better quantify the different variation types. Chapter 6 presents future directions that address these issues. Finally, Chapter 7 reviews the main points of this work and concludes.

Chapter 2

Background

Reconfigurable circuits are more amenable than other circuits to reclaiming the performance and efficiency lost to process margins. The reason for this is that reconfigurable circuits have the capability to carefully choose which transistors and wires to use post-fabrication, once their characteristics have been measured. The CAD tools can then carefully select these resources, optimizing the circuit being mapped so that it operates well beyond the conservative operating parameters set by the manufacturer, while still maintaining logical correctness.

Before exploring how margins are reduced and how reconfigurable circuits meet these constraints despite the myriad sources of variations, we must first have a firm understanding of what modern reconfigurable circuits, such as field programmable gate arrays (FPGAs), are, what their structure is, and how they are utilized. Furthermore, a deep understanding of the different sources of variation will clarify future discussions. Finally, a presentation of how variation is currently dealt with, through margining and binning, gives the required context to frame the techniques proposed later in this thesis.

In this chapter, we present the background that will form the foundations for the rest of this work. Readers familiar with modern FPGAs may move directly to Section 2.2.

2.1 Reconfigurable Circuits

Reconfigurable circuits provide the efficiency of hardware with the flexibility of software. They consist of small programmable elements that perform simple logic computations, embedded in a general routing structure. The post-fabrication flexibility that reconfigurable circuits provide is the key feature that makes them an ideal choice for dealing with the extreme variation expected in coming technology nodes. We focus on FPGAs, as they represent the most advanced incarnation of a reconfigurable circuit to date. A simplified model of an FPGA helps present the key aspects of

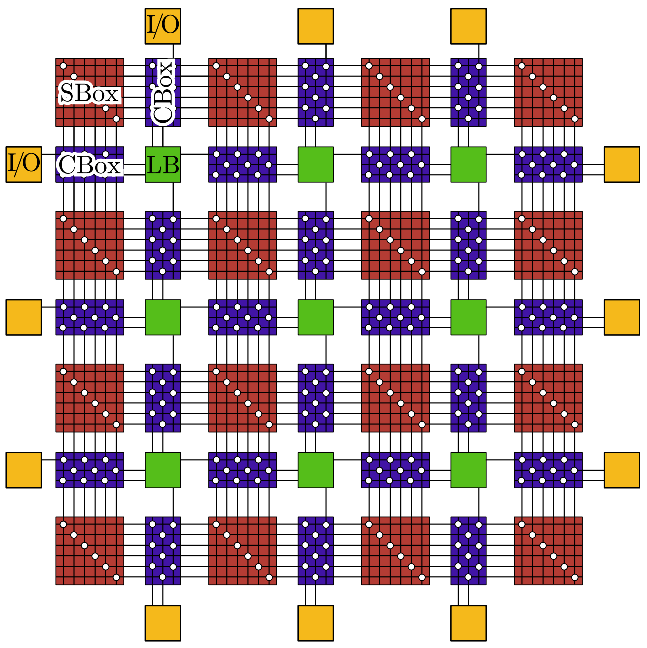


Figure 2.1: Diagram of a simple FPGA showing the logic block (LB), switch box (SBox), connection box (CBox), and external IO. Small circles in the SBox and CBox represent programmable switches.

this technology (Section 2.1.1). With the basics established, Section 2.1.2 introduces the advances applied to modern FPGAs.

2.1.1 Ideal Model

The simplest FPGA model that encompasses many of the intricacies of the technology is shown in Figure 2.1. Along with the logic blocks(LB) familiar to all reconfigurable circuits, this FPGA introduces a connection block and a switch block. The connection block or CBox connects the LB to the nearest routing resources, allowing it to pick its inputs and output from an arbitrary routing track on the interconnect network. The SBox, as the switch blocks are commonly known, appears at the intersection of horizontal and vertical routing tracks. It consist of a set of programmable connections that allow a signal coming in to the SBox to continue on another track in one of three possible direction, left, forward or right. Finally, the LB is composed of a programmable 4 to 6 input gate that can be configured to compute any 4 to 6 input logic function. The output can then, optionally be registered using a flipflop before connecting to the interconnect.

Practically, logic blocks are implemented by small memories known as lookup tables (LUTs). The input to the LUT acts as an address to a memory cell where the result is stored. Moreover, CBoxes and SBoxes are not fully populated, i.e., not every connection between CBox and LB is available, nor can a signal rout from any track in an SBox to any other routing track. This depopulation of switches is beneficial because it reduces area, while still allowing for rich interconnect that can accommodate most any routing requirement [61, 73, 50].

Typically, SRAM cells are used to store the programming of each switch. For a particular computation implemented on an FPGA, the state of these SRAM cells along with the configuration of the LUTs form what is called a *bitstream*. The bitstream is the end result of the FPGA CAD flow, a process that starts with a description of the computation, either in some high level language such as C or Java [16], or some high level hardware description language such as VHDL or Verilog [74]. Between the high level description and the bitstream the circuit description goes through a set of transformations, including a technology mapping that modifies the logic to fit the target device, placement, which distributes the logic over individual LBs, and routing, which figures out how to use the interconnect to connect the LBs as required.

2.1.2 Modern FPGAs

Modern FPGAs have evolved from the simple model described above to include many advances that improve performance, reduce energy and allow the FPGA to better capture the desired computation.

The logic block has grown in complexity from a simple LUT-flipflop pair to what Altera calls a Logic Array Block (LAB) [10] and Xilinx, a Complex Logic Block (CLB) [96]. Essentially, they are groups of LUTs and registers with a small amount of local interconnect that allows for more complex and localized computation. These structures also include dedicated carry chains to implement adders and other logic that fits that compute pattern. Moreover, modern FPGAs are no longer homogeneous arrays of LBs, rather they include a majority of LBs with a handful of specialized or embedded blocks. These include multipliers, memories and even small processors [36].

The interconnect has also matured. Instead of length one connections between SBoxes, FPGAs now include segments of different lengths. What's more, the interconnect is hierarchical, with shorter segments connecting directly to LBs and longer segments connecting only to shorter segments. This structure allows signals to more efficiently traverse long distances. LBs can also communicate directly with neighboring LBs via dedicated direct connections between LBs, giving even more flexibility when routing signals. Finally, FPGAs have moved from bidirectional wires to direct drive, where pairs of wires have a dedicated routing direction, one routing in one direction, and the other in the opposite. This advancement simplifies the interconnect switches and has the benefit of reducing both area and delay [52]. In addition, modern FPGAs contain a number

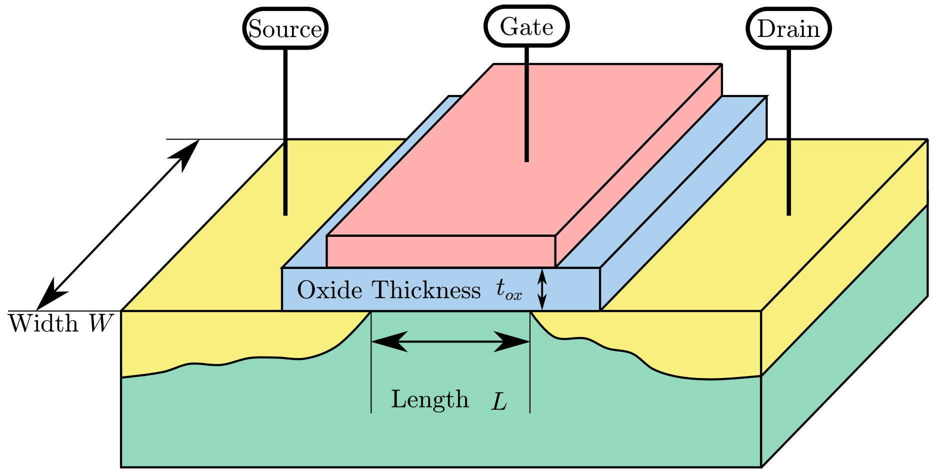


Figure 2.2: Diagram of a MOS transistor highlighting the main physical parameters, W, L, and t_{ox}.

of phase locked loops, or PLLs, that enable programmable clock frequencies. This, along with a complex series of clock distribution networks provide Timing Extraction with the necessary tools to measure path delays in FPGAs.

2.2 Transistor Properties

The MOS transistor is the workhorse of integrated circuits. Its miniaturization has enabled innovations and industries that touch almost every aspect of our lives. However, the ability to integrate billions of transistors in one chip is not without its challenges. As we continue to push down the dimensions of these devices, effects that previously were negligible become dominant. With so many small transistors in the same circuit, variations between them are inevitable.

Variations lead to changes in the performance and efficiency of our circuits. In this section we explore the physics that explain the behavior of a transistor. Figure 2.2 shows a simple diagram of a transistor to help ground some of its parameters referenced in this section. This analysis allows Section 2.3 to easily connect physical variations to circuit behavior.

2.2.1 Delay and Energy

Understanding how the delay and energy dissipation of a transistor relate to physical properties such as transistor channel length, gives context as to why physical variations directly lead to changes in circuit performance and efficiency. To start this discussion, we formally look at the equations that govern delay and energy per operation in a transistor.

To estimate the propagation delay through a transistor, to a first order, we can model the channel between the source and drain as a resistance, R, and consider how long it takes to charge or discharge the capacitive load seen by the transistor, C_l, including wires and downstream transistors, through the channel. The delay is simply

$$\tau_{pd} = C_l \cdot R$$

From Ohms law we know that R can be modeled as the voltage difference between the source and drain terminals, V_{ds}, over the current flowing through the transistor channel, I_{ds}. Therefore, the propagation delay of a transistor, τ_{pd} is given by

$$\tau_{pd} = C_l \cdot \frac{V_{ds}}{I_{ds}} \tag{2.1}$$

In conventional CMOS, V_{ds} is at most equal to the supply voltage, V_{dd}, but varies depending on how the transistor is used. I_{ds} however, varies depending on the voltages difference applied between the gate and source terminals. We will return to how I_{ds} behaves in the next section.

To talk about the energy per operation of a transistor it is helpful to consider the transistor as being part of a CMOS gate with a pull-up network formed by p-type transistors and a pull-down network implemented in n-type transistors. In this frame of reference, a transistor will either be "on", allowing charge to flow through its channel or "off", trying to prevent charge from flowing.

Due to a number of effects [75, 66], a transistor that is supposed to be off allows a small amount of current to flow. We refer to this as $I_{ds,sub}$ and will discuss it later. Assuming that the cycle time, determined by the frequency at which the circuit is running, is τ_{cycle}, the energy *leaked* by an off transistor is expressed as

$$E_{leak} = I_{ds,sub} \cdot V_{ds} \cdot \tau_{cycle} \tag{2.2}$$

To contrast, a transistor that is on will allow enough current through to charge (p-type transistor) or discharge (n-type transistor) the capacitive load C_l. The energy consumed by charging or discharging C_l through the transistor is

$$E_{switch} = C_l \cdot (V_{ds})^2 \tag{2.3}$$

The total energy used by a circuit in a cycle, E_{total}, is then given by the sum of the E_{switch} of all the transistors that are on, plus the sum of the E_{leak} for those that are not.

$$E_{total} = \sum_{i \in on} E_{switch}(i) + \sum_{j \in off} E_{leak}(j) \tag{2.4}$$

Before delving deeper into the device physics of the transistor, we can begin to see how variation will affect operation. From Equations 2.1 through 2.3 it should be apparent that a change in V_{ds}, the voltage between the drain and source terminals, will lead to a proportional change in the delay and energy utilization of a transistor. This voltage difference is supplied by the aptly named supply voltage, V_{dd}. In modern integrated circuits, V_{dd} is distributed by a complex network of wires that must reach every gate. Fluctuations of 10% in this distribution network are not uncommon [17].

2.2.2 Current-Voltage Characteristics

Both the delay, τ_{pd}, and leakage, E_{leak}, of a transistor depend on the current through the transistor I_{ds}. We can model a transistor as a voltage controlled current source. The amount of current flowing through a transistor depends on the voltage difference between the gate and the source. Depending on the state of this relationship, the transistor will be operating either in saturation mode - when "on", or sub-threshold mode - when "off". Equations 2.5 and 2.6 show the relationship between the physical properties of a transistor and the current flowing through it for the two operating modes [70, 35].

$$I_{ds,sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right) \tag{2.5}$$

$$I_{ds,sub} = \frac{W}{L} \mu C_{ox} (n-1) \cdot {v_T}^2 \cdot e^{\frac{V_{gs} - V_{th}}{n \cdot v_T}} \left(1 - e^{\frac{-V_{ds}}{v_T}} \right) \tag{2.6}$$

W and L refer to the physical width and length of the transistor. C_{ox} is the unit capacitance of the gate oxide, and is related to its thickness t_{ox} and type of material ε_{ox} (Figure 2.2). μ and v_{sat} relate to the charge mobility. v_T is the thermal voltage, a function of temperature. Finally, n is a technology specific constant.

V_{gs} represents the voltage difference between the gate and source, while the voltage difference between the source and drain terminals is V_{ds}. $V_{d,sat}$, is the velocity saturation, dictating a limit on how much current flows between drain and source. Finally, V_{th} indicates the threshold voltage of the transistor.

Together, these equations relate the physical and electrical properties of a transistor. The following section will detail the specific ways in which almost every one of the parameters in these equations varies. Reading the next section it is essential to keep in mind that some of these parameters are more important then others. In particular, observe the exponential dependence of $I_{ds,sub}$ in Equation 2.6 on the voltages, V_{gs}, V_{ds} and of utmost importance, due to the fact that almost every physical effect has an affect on it, is the threshold voltage of the transistor, V_{th}. This relationship indicates that even a small change in one of these parameters, will lead to a large change in subthreshold current, which will directly impact the energy and delay of the transistors (Equations 2.1 and 2.2), and with that, the performance and efficiency of the whole circuit.

2.3 Variation Sources

Transistors and wires in a modern CMOS integrated process will be subject to many types of physical variation. Broadly, we can separate them into three categories: Process or manufacturing variation, environmental and operational variation, and aging effects. These variations manifest as physical deviations from design, such as a change in channel width, or electromagnetic fluctuations as seen in crosstalk.

Such variation directly alter the delay and energy of a transistor or wire from the nominal design values as defined by the equations in Section 2.2. These changes, in turn, lead to a degradation in performance and efficiency of the chip, and eventually incite yield loss due to variation induced failures. Though the effects of variation may lead to similar degradations, the process by which each variation type manifests and leads to a problem is different. As such, it is beneficial to separate variation into categories to better design solutions that can manage the changes that variation inflicts on our devices. Since Timing Extraction focuses on process variation, this is the only type of variation we examine here. For readers interested, Appendix A explains how environmental, operational and aging effects affect the circuit.

2.3.1 Process Variation

As feature sizes continue to shrink, more and smaller transistors fit on one chip. The manufacturing process required to achieve this device density is complicated and requires many steps. Although great effort is expended to ensure consistency and precision in these steps, deviations from the target result inevitable occur. Manufacturing variations lead to variations in the physical properties of a device. For example, transistor geometries will vary from the design parameters, and dopant densities will fluctuate from transistor to transistor. These physical differences will lead to electrical changes in the current through a transistor, and ultimate affect the delay and energy requirements of the device.

To see how manufacturing variations impact our designs, we first briefly review the manufacturing process itself. From there we can link process variation to physical changes in the transistor, and finally to electrical properties.

2.3.1.1 Manufacturing Process

The manufacturing of an integrated circuit begins with a silicon wafer over which a number of circuits will eventually be fabricated. Each full step of the process deposits and patterns a layer of material on top of the silicon wafer. The layer closest to the wafer will have the active electrical devices such as the transistors. Subsequent layers are generally used for metal interconnect. Advanced processes commonly have 11 metal layers [83, 38], yet, for the most part, the transistor layer and each metal layer undergo the same general set of steps.

Each layer begins with a photolithographic process that deposits a masking pattern. The pattern allows some regions to be exposed through etching. Once the desired regions have been revealed, a processing step appropriate for the type of layer is applied. Finally, isolating material, deposited to prevent unwanted interactions between layers, is polished to prepare the wafer for the next layer, at which point, the process repeats.

Photolithography is a process akin to developing film photographs. Like a photograph, where a negative allows a specific pattern of light to shine on a chemically coated light-sensitive paper, in photolithography, a mask allows a specific pattern of ultraviolet light to shine on a layer of photoresist applied to the surface of the wafer. Once developed, the photoresist layer will have the shape described by the mask.

For well over a decade, the wavelength of light used for this process has remained fixed at 193nm [27]. Nevertheless, minimum feature size has steadily progressed towards smaller dimensions, well below 193nm. In fact, circuits with feature sizes in the range between 22 nm and 28 nm are regularly manufactured, and smaller features, on the order of 14 nm, are now in production. In order to produce sub-wavelength lithography, a series of shrinking and focusing lenses, along with a complex set of resolution enhancement techniques (RET) are used. These include phase shift masking (PSM) [77] which takes advantage of wave properties of light to create precise interactions to increase the resolution, optical proximity correction (OPC) which modifies the shape of the mask to compensate for the reduced resolution, and multiple exposure systems [25] which uses two or more masks to achieve the final desired result, reducing the resolution each individual mask must produce.

After the photoresist is exposed to ultraviolet energy for a controlled amount of time, it is developed and a pattern similar to the mask remains on top of the wafer. This pattern protects the covered regions from the next step, etching.

Etching removes unwanted material not protected by the photoresist developed during photolithography. The etch rate controls how material is removed and is dependent on the etching processes used. This can be chemically based, where a reaction allows material to be removed, or momentum based, where physical bombardment of the surface aims to remove atoms. The etch process may remove material isotropically, in all directions, or anisotropically, in one direction only. Together, etching and photolithography transfer the patterns that will define the circuit and prepare it for processing.

Processing creates the transistors and wires of the circuit and varies depending on which is being formed. Transistor formation requires many masking and etching steps to define each of the features. A simplified description of the processing steps follow. First, a p-well or n-well is defined, depending on the type of transistor, by implanting corresponding dopant atoms in the exposed wafer region. A second processing step deposits oxide over the well to create isolation between it and the gate. Subsequent processing steps define the polysilicon for the gate terminal and implant dopant atoms to create the source and drain regions. Finally, an isolating layer is deposited over the whole transistor to prevent it from interacting with the layers above.

Interconnect metal wires connecting the transistors together are formed in higher circuit layers. Aluminum had long been the metal of choice for wires, however, due to the higher resistance expected in the small features of modern processes, copper, with its lower resistivity, has displaced aluminum as the preferred interconnect material. The process to deposit copper wires is known as dual-damascene. Initially, a layer of silicon is deposited and vias connecting to the lower layer, along with the wires for the current layer are etched on the deposited silicon. To prevent diffusion into the surrounding silicon, a thin barrier layer of some other metal is then deposited followed by a thin layer of copper which acts as a seed for electroplating and filling the trench with copper.

Once a layer of transistors or wires are processed, polishing is required to prepare for subsequent layers.

Polishing is a chemical-mechanical process used to planarize the top most layer on top of which further layers can be formed. A wafer is held upside down over a rotating polishing pad and a chemical slurry is applied to aid in the polishing. The speed, pressure, and slurry composition control how much material is removed from the wafer. With the leveling of the top surface complete, the manufacturing process can begin again, forming each of the many layers of a modern IC.

2.3.1.2 Variation Sources and Physical Effects

The simplified glimpse into the manufacturing process above should serve to illustrate the complexities a wafers undergoes during the manufacturing of an integrated circuit. Each step must be carefully prepared and executed and each step provides variation an opportunity to infiltrate the process. In this section, we examine how variation in the manufacturing of a circuit leads to physical deviations from the design parameters of the transistors and wires being created. The next section will connect these physical variations to electrical changes in the devices.

Although there are many sources of process variation, ultimately, they manifest in two physical aspects of the devices: Geometry, and dopant concentration fluctuations. What's more, the variation can be random, where a stochastic model best represent the physical results, either because it is a random process, or because the process is too complex to methodically model, or systematic, where a clear correlation exists between a process parameter and the resulting physical device variation [72]. Often process variation is also classified as intradie, being between two devices within a die, interdie, two separate dies in a wafer, or wafer-to-wafer.

Device Geometry refers to the shape of the devices created through lithography. Specifically, we are interested in the length, width, and oxide thickness of the transistor and the shape of the interconnect wire. Given that photolithography's primary goal is to define geometries on the wafer, it should come as no surprise that almost every step has the potential to introduce variation into the shape of the device.

In order to produce sub-wavelength features, the lithographic mask has to be carefully designed to account for the many interactions of light that will occur during image transfer on to the photoresist. Despite the advanced resolution enhancement techniques employed, diffraction patterns from adjacent lines will change the line width of the feature being printed. These proximity effects are most evident at the lower layers where smaller features are more common. Also metal wires tend to exhibit line end shortening and corner rounding as a result of the light interactions [48].

A major source of variation in both wires and transistors is known as Line Edge Roughness (LER) and Line Width Roughness (LWR). LER manifest as jagged edges in geometries on the wafer and LWR is the resulting variation in width due to LER. The main sources of LER are light interactions, etching, and defocus, where the mask image is not sharply defined on the surface of the resist. Defocus, in turn, has several causes, these include mask misalignment and tilt, change in the refractive index of the reduction lenses due to heat from the energy source, and variations in the resist thickness leading from variations on the wafer surface induced by chemical-mechanical polishing. As identified by the ITRS, while circuits continue to scale, "LER of photoresist has substantially sustained the same absolute value", on the order of 5nm [76], "and therefore has attained an even larger percentage of [the random variation]"[2].

In addition to its LER contributions, defocus has a systematic effect on line width that correlates with the density of the design, where dense features have increased line width, and decreased line widths correlate with isolated features [98]. This effect can be reduced by adding features to the lithographic mask in order to maintain relatively constant density, however, these features may cause undesired light interactions, further contributing to the overall geometric variation.

Finally, chemical-mechanical polishing directly affects the shape of the interconnect since every wire layer gets polished before processing for the next layer can begin. This polishing introduces a systematic variation in the layer thickness that is, in part, dependent on the density of the patterns being polished [43].

Gate Oxide Thickness defines the thickness of the oxide separating the transistor's channel from the gate. The gate oxide is created through an iteration of the manufacturing processes and therefore, is subject to variation induced by the process. To maintain good control, the gate oxide thickness has kept up with transistor scaling. However, as the ITRS notes, a "particularly challenging issue is the control of the thickness, including its variability, of these ultra-thin MOSFETs" [2]. As such, a thickness variation of one or two atomic layers, now accounts for a significant variation percent. At these dimensions, gate tunneling becomes a significant issue, therefore, to alleviate this problem, newer technology nodes have moved to using high-κ dielectrics. The high-κ dielectrics allow for thicker oxides, while providing an Equivalent Oxide Thickness of approximately 1 nm or about 5 atomic layers [49, 67], maintaining the control provided by thin SiO_2 dielectrics. However, this technology comes with its own variation challenges since its polycrystalline structure is prone to inhomogeneities, that directly affect the electrical properties of the device [20, 15].

Random Dopant Fluctuations represent the uncertainty of charge concentration and location within the transistor's doped regions. The process of ion implantation followed by thermal annealing is inherently nondeterministic and can be modeled by a Poisson distribution [67], leading to the random nature of this variation source. Moreover, as device size scales down, the law of large numbers no longer applies, and therefore, the magnitude of this random variation increases [45]

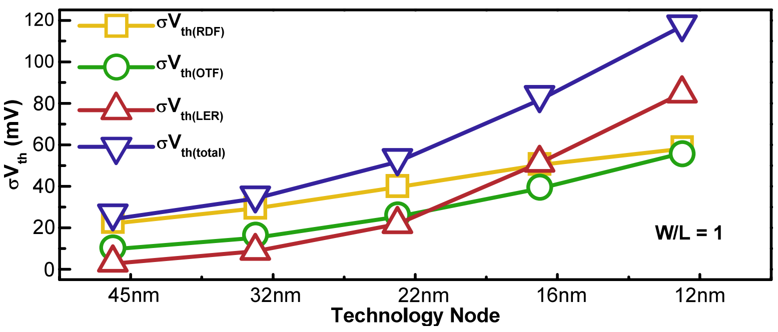


Figure 2.3: $\sigma_{V_{th}}$ as a function of technology nodes, based on predictive technology models. Considering the individual effects of random dopant fluctuations (RDF), line edge roughness (LER) and oxide thickness (OTF) from [99]

2.3.1.3 Physical Variation and Electrical Effects

We can now connect process variation to the electrical properties of a transistor. Process variation directly affects the threshold voltage, V_{th}, and both the saturation and subthreshold currents, $I_{ds,sat}$ and $I_{ds,sub}$. Detailed models of V_{th} can be found in many sources [65, 99], however, for simplicity, we focus on how V_{th} variation changes as a function of process variation. Generally the V_{th} distribution is modeled as Gaussian, having a nominal value $\mu_{V_{th}}$ and a standard deviation of $\sigma_{V_{th}}$. Figure 2.3 shows the individual effect of RDF, LER and oxide thickness on $\sigma_{V_{th}}$, as a function of technology node. Although the dominant effect changes, the overall trend shows that $\sigma_{V_{th}}$ increases as we scale. Equation 2.7 formalizes this and shows how the standard deviation of V_{th} changes with oxide thickness t_{ox}, increasing dopant density N_a, and channel area, $L_g W_g$. The electrical charge on the electron is q, ε_{ox} is the dielectric constant of the oxide material, and W_d, the width of the depletion region. [45, 47].

$$\sigma_{V_{th}} \propto \frac{qt_{ox}}{\varepsilon_{ox}} \sqrt{\frac{N_a W_d}{L_g W_g}} \tag{2.7}$$

Although saturation current, $I_{ds,sat}$, and subthreshold current, $I_{ds,sub}$ depend on V_{th} and are, therefore, already indirectly affected, they both have parameters that more directly relate process variation to current. The full equations for $I_{ds,sat}$ and $I_{ds,sub}$ can be found on page 10, below, instead, we only highlight how they are directly affected by process variation.

$$I_{ds,sat} \propto W C_{ox} v_{sat} \tag{2.8}$$

$$I_{ds,sub} \propto \frac{W}{L} \mu C_{ox} \tag{2.9}$$

Here, W and L come directly from the dimensions of the gate. Lithographic variations will impact these dimensions along with LER and LWR. C_{ox} is the electrical unit capacitance provided by the oxide. It is a function of the material, ε_{ox}, and the oxide thickness, t_{ox}.

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}} \tag{2.10}$$

Finally, the charge mobility μ is a measure of how an electron moves in the channel. It is, in part affected by the dopant concentration, decreasing with a decreased dopant concentration [88].

Process variation also has an electrical effect on wires. The delay of a wire is roughly proportional to its capacitance and its resistance. Resistance is proportional to the cross-sectional area of the wire, As geometric variations increase in the wire, resistance will vary accordingly. Capacitance on the wire is also affected by changes in its shape. Furthermore, adjacent wires can interfere with each other, as explained in Section A.1.3. The magnitude of this interference is related to the capacitance between the two wires, which in turn is determined by the wire geometries.

In this way we see how variation in the manufacturing process leads to variation in the delay and energy. As we move to smaller devices, new materials, and new technologies, we expect to see new sources of variation while some parts of the process may improve. Nevertheless, "[process] variability is here to stay and will likely play a major role in design and manufacturing of future ICs" [3].

2.4 Failure Model

In the best case, unmitigated, variation will only lead to a decrease in performance and an increase in energy consumption as we are forced to use devices with more extreme parameters. This, in itself, is enough for concern. However, as we continue towards more aggressive technology nodes, variation increases will inevitably lead to malfunctioning chips.

To understand how variation leads to failures, consider the following simplified example. Increased variation leads both to transistors that leak too much current, and transistors that are very slow to charge their downstream capacitance. Arbitrarily mapping a circuit to an FPGA runs the risk of using both types. When variation is large enough, the time it takes a leaky transistor, in the "off" state, to leak enough current to charge its downstream capacitance may be less than the time it takes a slow transistor, in the "on" state, to allow enough current through to charge its downstream capacitance. This situation leads to an inversion where a transistor that should be off behaves as if it is on, and a transistor that should be on, appears to be off. Computationally, this means that where we expect a 0 we actually get a 1, and vice versa. This behavior is likely to cause the computation to fail.

A component-specific mapping, on the other hand, carefully selects which devices to use, thus preventing this situation. Timing Extraction provides the knowledge required to successfully produce a component-specific mapping and avoid variation-induced failures.

2.5 Managing Variation

The goal of a manufacturer is to ship defect-free circuits as soon as possible. In light of all the variation a chip experiences, this is a tall order. In order to achieve this, a circuit must undergo extensive testing before it ships to the customer. However, full test coverage increases production latency, reducing profits. Quickly producing parts and shipping only perfect parts are conflicting objectives. As a compromise, exhaustive testing is not performed. Instead, circuit's performance and efficiency are derated, through guard-banding, in the hopes that any missed defect or latent aging effects will be concealed by the operating margins guard-banding provides.

2.5.1 Testing

After manufacturing, a circuit undergoes extensive functional and parametric testing. This is a time intensive procedure done on expensive equipment. In fact, testing can account for up to 40% of the total circuit cost [40]. Nevertheless, it is necessary in order to guarantee that shipped parts will be defect free with extremely high probability. Testing takes place in three general steps: Burn-in, functional testing, and parametric testing [56].

Burn-in is intended to accelerate aging faults by exposing circuits to extreme temperatures and varying voltage supplies well beyond the intended operating parameters [60]. Subjecting the circuit to these harsh environments uncovers latent defects that would otherwise would appear shortly after the customer received the product. By discarding the circuits that fail prematurely, this process increases the average lifetime of shipped products.

Functional testing looks for broken devices in the chip, either from manufacturing errors or from burn-in. Discovering functional failures requires exercising all devices in the circuit, confirming whether or not they perform as expected. These tests require applying vectors to inputs, both primary and intermediate, such as registers. Assuming there are n of these inputs, to exhaustively test, requires 2^n vectors, a time consuming proposition. Great advances have been made to reduce this number by using automatic test pattern generation (ATPG), which analyzes the circuit to determine the vectors that will exercise every wire and transistor in the chip. Nevertheless, it does not always achieve full coverage of all devices.

Finally, parametric testing measures how the circuit behaves under different operational and

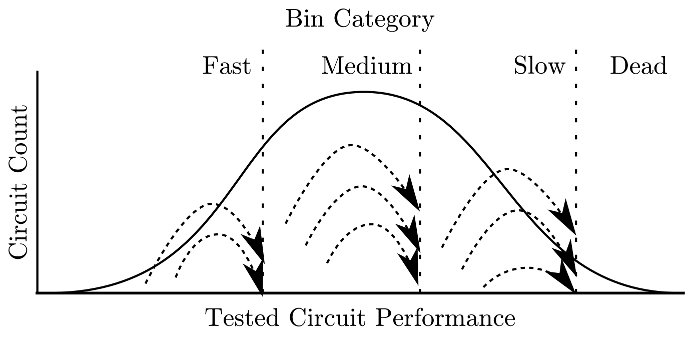


Figure 2.4: Classifying a set of tested circuits into three bins, fast, medium and slow. Circuits that fail to meet a minimum threshold are thrown out. From [54].

environmental conditions. The goal of parametric testing is twofold. First, it classifies chips based on how they perform under different conditions, second, it discards chips that fail to meet a minimum standard. In order to perform parametric testing, chips are exposed to different temperatures and voltages, however, unlike burn-in where the goal is to cause failures, the environment is not as harsh and a range of conditions are explored. Usually this range is limited to a handful of parameters that test under nominal as well as upper and lower extreme operating conditions. These are known as corners, for example, the low-temperature, high-voltage corner traditionally exposes the highest performance achievable by the circuit. Corner testing allows a manufacturer to group circuits with similar performance together. These groups or categories make up a particular device type, ranging from high performance - sold at a premium, to low grade - targeted at consumers that do not require the high capabilities. Figure 2.4 illustrates how this classification takes place.

Although these tests discover a great majority of faults, with devices shrinking and circuit size growing, it is becoming harder and more expensive to fully test a chip. Small devices are more susceptible to temperature and other burn-in techniques, therefore, burn-in may damage more chips than necessary, or it must be performed more gradually, leading to longer test times [19]. Furthermore, full functional testing is no longer feasible due to the sheer number of vectors necessary to test. This may not be as dire as one might think. It is often undesirable to exhaustively test every vector disregarding intended circuit functionality [71]. It is possible that a fault that exists in the circuit will never lead to a defect if that fault is in a path that is never activated by the intended application of the circuit. This is especially true of FPGAs as they have the ability to avoid faults all together [55, 90, 44]. Finally, nano-scale devices sometimes exhibit performance inversions. For example, at low-voltage and high-temperature, some devices show smaller delays than under a lower temperature [26, 46]. This requires testing more corners

[83], also leading to longer test times. To reduce the amount of testing, manufactures can tradeoff testing for margining, with the hope that the right combination minimize total cost.

Timing Extraction differs from other tests in that it can greatly reduce the need for prolonged burn-in and functional testing. It allows the manufacturer to functionally test the FPGA both before and after shipping, without the need for expensive testers, using only resources already available on the FPGA. Thus, if a fault is missed by burn-in or functional testing, using Timing Extraction the FPGA can detect this and reconfigure to avoid the detected fault.

2.5.2 Margining

Margining derates the voltage and frequency operation of a circuit so that any increase in critical path delay due to missed faults, environmental and operational variations, or aging effects do not lead to timing errors. In essence, it forces the circuit to operate under a worst-case scenario assumption even when conditions are closer to normal, wasting performance, power, or both. Typical guardbands, as margining is also known, are set to maintain correct operating conditions for seven to ten years [87, 7], this requires margins that are 30% beyond the nominal [91]. As circuits continue to shrink, this percent is likely to increase.

To illustrate how margins work, assume we have manufactured a circuit and need to set guardbands for its cycle time so that it operates without failures. Figure 2.5 shows a cartoon that will illustrate how we add to the cycle time to cover each of the variation types. The blue region represent the distribution of cycle times achievable by all the circuits in a particular speed-bin, due to process variation. In order to prevent failures in all these chips, we must make sure our cycle time is large enough to cover the slowest one. Moreover, since we do not know exactly what the environment might be for a particular device, we estimate a worst case environment and add enough margin to our cycle time to make sure there is never a failure regardless of what the environment is like. This is represented by the purple region in the figure. Finally, devices in the circuit will age and slow down. To make sure that over the entire lifetime of the circuit, we do not experience an aging-related failure, we add to our cycle time to cover worst-case aging, as shown by the orange region. Combining process, environment, and aging worst-case variations, leads to the required worst-case variation margin. This margin guarantees that regardless of what the circuit is doing, how it ages, or what the underlying process variation is, it will still function correctly. Unfortunately, this comes at a high price, since not all chips have worst-case process variation, nor do they run in worst-case environments, and they age over time, not immediately. In essence, all circuits within a circuit group pay the price for a few bad cases. This need not be the case.

Techniques such as dynamic frequency scaling and dynamic voltage scaling [21, 51] aim to reduce process margins by dynamically monitoring the FPGA for errors. If no errors are detected either the frequency is increased to improve performance, or the voltage decreased to reduce energy consumption. If an error is detected, the opposite is performed. The goal is to maintain the frequency, the voltage, or both at levels just beyond where errors would occur, in order to better match the performance and energy efficiency of a given FPGA.

Although these techniques no longer force us to operate at levels that account for the worst-case over all chips, they still require that we operate at levels that account for the worst-case within a given chip. In fact, as we will later see in Section 4.9, reducing the voltage actually increase the variation, thus dynamic voltage scaling must be extremely careful not to reduce the voltage too much, and therefore, must still operate with larger margins.

A different approach known as statistical static timing (SSTA) [82] analysis the timing of a circuit but takes into account the effects of variation on delay by assumes that the delay is a distribution of possible delays. At the conclusion of its analysis, it produce a distribution of the overall circuit delay. Although margins may be adjusted based on SSTA, it is still necessary to maintain a worst-case process margins.

Timing Extraction aims to go beyond this worst-case operation to a best-case scenario. Without knowledge of the process variation in an FPGA, when a design is mapped, there is a high probability of selecting a slow physical resources, similarly for energy inefficient resources. Timing Extraction fully measures the process variation in the FPGA, thus allowing us to avoid these extreme resources. By carefully selecting which resources we use in the FPGA, we can almost completely eliminate process margins and operate as well as the best resources in the FPGA allow.

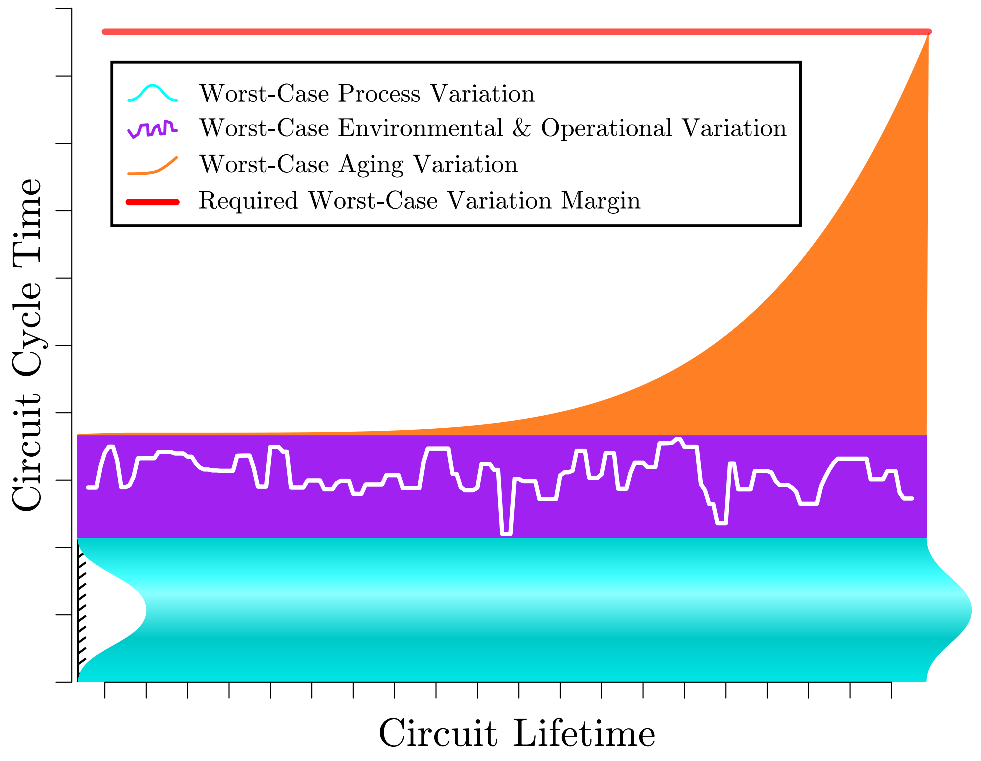


Figure 2.5: The cost of margins: Required worst-case variation margin shown as the sum of the worst-case process, environmental and operational, and aging variation. Figure not to-scale, nor meant to indicate that these three effects are completely independent

Chapter 3

Timing Extraction

The flexibility that reconfigurable circuits support enable less expensive approaches to controlling variation, as contrasted with conventional testing and guard-banding. By measuring the shift from nominal of the delay of every component in the circuit, it is no longer necessary to account for worst-case process variation through margining. Instead, the FPGA CAD tools can be tuned to map a design to a particular FPGA based on the measured variation. Measuring every device is a tall proposition, however. It could require hours on highly customized, expensive testers. This need not be the case, though, as the reconfigurable nature of FPGAs allow them to self-measure through a set of simple configurations, as this chapter will demonstrate.

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. Nevertheless, it is possible to measure the delay of paths between registers. By measuring the delay of different paths in an FPGA, it is possible to decompose the delays of these paths into their constituents. Essentially, each path constitutes 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. If we have enough equations, we can solve for all the unknowns and directly acquire the delays of every component used in these paths.

Taking as input the physical resource graph of the FPGA, Timing Extraction, through an automated process, decomposes the physical resource graph into components, determines which subset of paths to measure, measures them, and computes the delay of each component using the measured path delays. In order to be useful, components must meet the following three constraints:

1. First and foremost, the delay of the component must be computable using a linear combination of a set of path delays.

2. Second, we must be able to combine components to incrementally compute the delay of any path in the FPGA. In this way, routing algorithms [24, 62] can easily use component delays as they incrementally searches for shortest routing paths.

3. Finally, to analyze the variation in the FPGA, we want the components to encompass as few physical FPGA resources as possible, this will expose fine-grain process variation. Moreover, components should encompass the same type of physical resources, so that when we compare the delay of component A to component B and discover A to be faster, it is not because A has fewer of different physical resources, but rather, because of process variation.

In this chapter we formulate the theory of Timing Extraction, applicable to most any reconfigurable circuit. Timing Extraction uses measured path delays to compute fine grain component delays. Understanding how path delays are measured helps define some basic ideas used throughout Timing Extraction. We then turn to the actual theory of Timing Extraction. Beginning with the physical resource graph, we form Logical Components (LCs), representing components with the minimal number of physical resources. LCs meet the second and third component requirements; however, they fail to meet the first, and are thus not suitable. We show it is relatively easy to generate components that meet the first requirement, meeting all three proves harder. To meet all component requirements, we reformulate components as Discrete Units of Knowledge, or DUKs. In essence a DUK is a small linear combination of LCs. Since LCs encompass the fewest physical resources, a small linear combination of LCs has only a small factor more physical resources. Moreover, we can combine DUKs to incrementally compute path delays. Finally, it is almost trivial to figure out which paths must be measured to compute the delay of the DUKs, thus meeting all component requirements.

3.1 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. 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 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. This technique has been successfully used to capture the delay of paths on FPGAs for many applications [80, 59, 79, 94, 93, 58]. Although

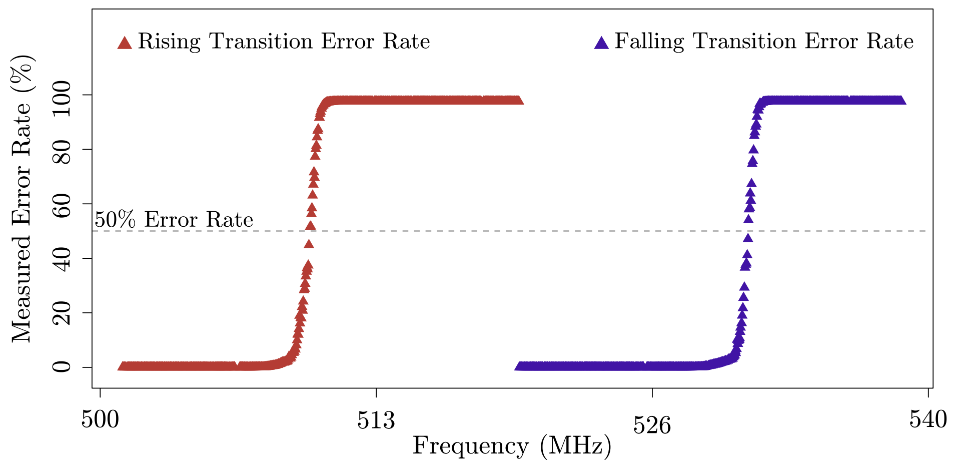


Figure 3.1: Measured error rate for a path. Differentiating falling transition error rates from rising transition error rates

this technique successfully measures path delays, simply measuring path delays is not sufficient to understand the types of variation within the FPGA.

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 (Figure 3.1). Measuring for the 50% failure point 2^{15} times at each test frequency, gives, with 99.98% confidence, a $\pm 1\%$ error. 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.

Due to the nature of CMOS, there is a possible delay difference in a rising transition, as compared to a falling transition. Figure 3.1 shows this difference for one measured path. In order to separate the falling and rising delays, for our purposes, the CUT is composed of a series of buffers connected one to the next. Figure 3.2 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 (EDC), one monitoring rising failures, the other, falling failures.

3.2 Logical Component Decomposition

It is not possible to measure the delay of a single wire or transistor in the FPGA, even indirectly. To explain why, consider the simple example of the LUT presented in Figure 3.3. Suppose we

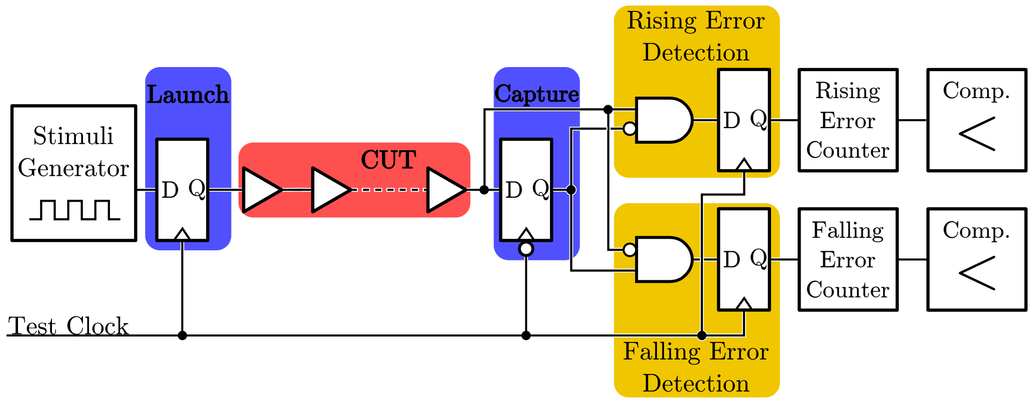


Figure 3.2: Path-delay measurement circuit.

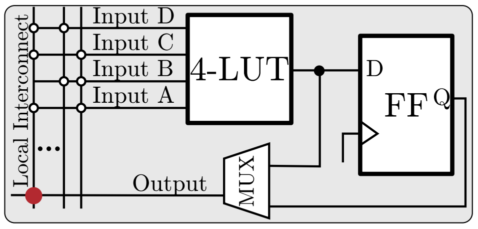


Figure 3.3: Block diagram of a 4-LUT and it's register, along with local interconnect

would like to know the delay of the highlighted crosspoint. This is not possible since any path that uses that crosspoint must also use the MUX, Output Wire and corresponding Local Interconnect Wire. 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 intuition to what a Logical Component (LC) represents.

The definition of a Logical Component naturally derives from the idea that there are sets of components where if one component is in a path, all other components in that set must be in the path as well, as the example above illustrates. Before formally defining an LC, we introduce the concept of the physical resource graph.

Timing Extraction takes as input the circuit description of the FPGA. Given this description we construct its corresponding physical resource graph by letting each component that contributes to a signal's delay be a node in the graph. Directed edges between the nodes mimic actual connections between components in the given circuit. Finally, since our measurement technique requires that CUTs be surrounded by launch and capture registers, every node representing a register splits into two nodes, a launch or source node and a capture or sink node, with no edge connection between them. Figure 3.4 shows the physical resource graph corresponding to the circuit diagram in the example above.

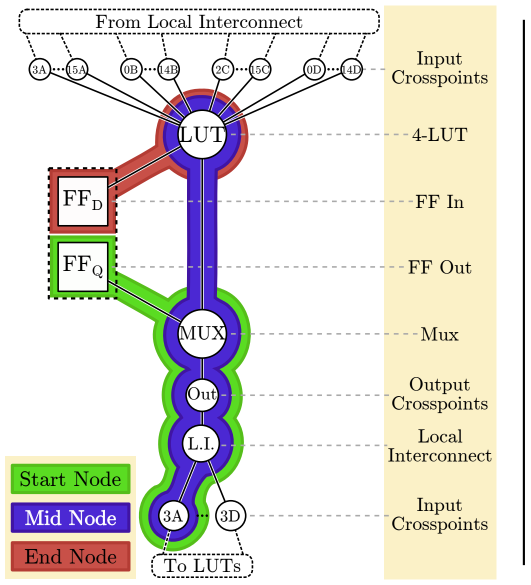


Figure 3.4: Physical resource graph of the circuit in Figure 3.3.

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 source register nodes, or nodes that have fan-in greater than one, and sink register nodes, or nodes that have fan-in greater than one. Physical resources in the same path are grouped into one LC Node. LC Nodes are then connected to form an LC Node graph that maintains the relations between nodes, as described by the physical graph (Algorithm 1).

Algorithm 1: LC Node Decomposition

```
Input: Resource Graph $G_r$
Output: LC Node Graph $G_{lc}$
                                        /* Identify nodes with fan-in > 1 */
foreach node $n \in G_r$ do
    if Fanin $(n) > 1$ then
        $l$.Add($n$)
                                        /* Add registers */
$l$.AddAll(GetRegisters($G_r$))
                                        /* Generate LC Nodes */
$lcNodes \leftarrow$ FindAllPathsBetweenNodes($l$)
                                        /* Build LC Node graph */
foreach Pair of LC nodes $(x, y)$ from $lcNodes$ do
    if $x.lastResource == y.firstResource$ then
        $x$.AddChild($y$)
$G_{lc}$.AddAll($lcNodes$)
```

The last loop in Algorithm 1 defines the edge relationship between LC Nodes. Notice that

LC Nodes are formed between two nodes in the set l, and two LC Nodes have a parent-child relationship only if the last physical resources in the parent LC Node is the same as the first resources in the child LC Node. In order to make sure that we do not count the delay of this shared resource twice, as a convention, we assign it to the delay of the child LC Node. Nevertheless, the relationships created by this construction are part of the key to the main Timing Extraction algorithms later in this chapter.

Given a circuit, the corresponding LC graph indicates the finest component granularity at which we care to measure delays in a circuit. In other words, it naturally groups components in a way that represents the fact that, if the first component in that LC is used in a path, all the rest of the physical components in that LC will always have to be used in that path and therefore, there is no reason to measure them independently.

LC Nodes represent paths through a small number of physical resources. Given their construction, these paths will either begin at a register or begin at a combinatorial node with fan-in greater than one. Similarly they will end either at a register or at a combinatorial node whose child has fan-in greater than one. This provides a natural way to classify LC Nodes, based on the type of the first and last nodes in their path. We separate LC Nodes that start and end at a register from the other three combinations. This type of LC Node can be directly measured using the path measurement technique from Section 3.1. We focus on the remaining three combinations and give them the following names:

- **Start Nodes** are LC Nodes whose path begins at a register and ends at a combinatorial node.
- **Mid Nodes** represent LC Nodes that both begin and end at combinatorial nodes.
- **End Nodes** captures those LC Nodes where the path begins at a combinatorial node and ends at a register.

Figure 3.5 brings all these ideas together in an example. Figure 3.5(a) shows a physical resource graph where source and sink registers are square nodes and all other resources are circular nodes. The result of applying the LC decomposition algorithm is shown in Figure 3.5(b), where LC Nodes are colored based on their type. LC Nodes are labeled using interval notation to indicate inclusion or exclusion of particular physical resources. E.g., LC Node $\boxed{[F \to G)}$ includes physical resource $\textcircled{F}$ and up to just before physical resource $\textcircled{G}$. The reader is encouraged to confirm that any path from a source register ($\boxed{A}$ or $\boxed{B}$) to a sink register ($\boxed{H}$ or $\boxed{I}$) in Figure 3.5(a) is also found in Figure 3.5(b) by selecting the correct LC Nodes.

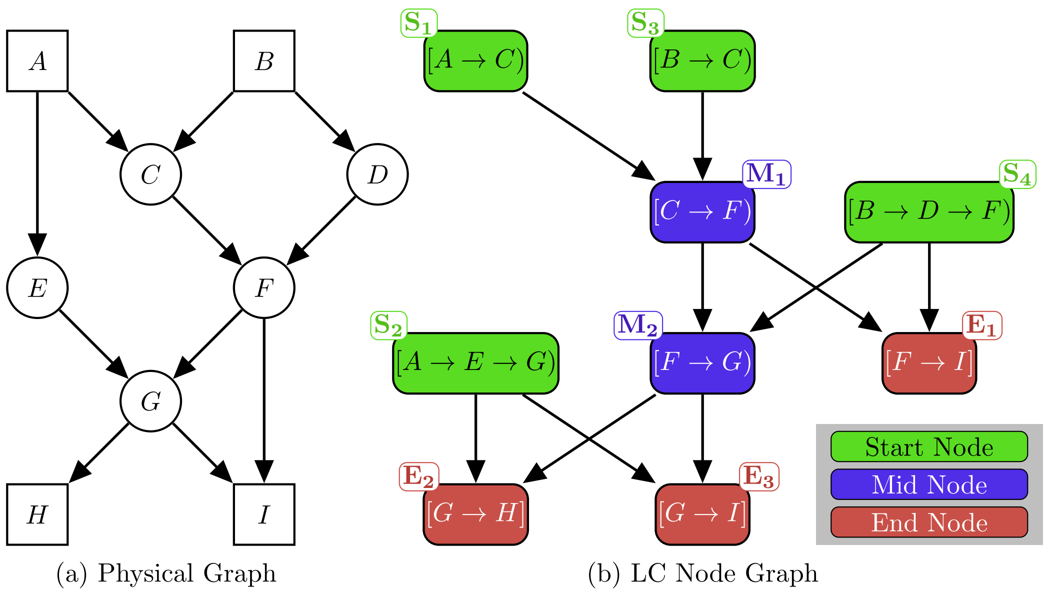

A
B
C
D
E
F
G
H
I
(a) Physical Graph
$\mathbf{S_1}$ $[A \to C)$
$\mathbf{S_3}$ $[B \to C)$
$\mathbf{M_1}$ $[C \to F)$
$\mathbf{S_4}$ $[B \to D \to F)$
$\mathbf{S_2}$ $[A \to E \to G)$
$\mathbf{M_2}$ $[F \to G)$
$\mathbf{E_1}$ $[F \to I]$
$\mathbf{E_2}$ $[G \to H]$
$\mathbf{E_3}$ $[G \to I]$
Start Node
Mid Node
End Node
(b) LC Node Graph

Figure 3.5: Transformation of a physical resource graph, where squares highlight registers and circles combinational resources, to an LC Node graph, where node names represent encompassed physical resources. Corner labels on LC Nodes, such as $\mathbf{S_1}$, provide convenient shorthand reference for later use in this chapter

The path delay measurement technique from Section 3.1 measures combinatorial paths between two registers. Looking at the LC Node Graph we see that any path between source sink registers has a consistent form. It begins with a Start Node, continues through zero or more Mid Nodes and always ends at an End Node. Thus, we can easily formulate measurable paths using LC Nodes.

3.2.1 LC Nodes as Components

LC Nodes seem a natural way to decompose an FPGA into components. To confirm whether they are suitable, however, we must ensure they meet the three component requirements presented at the beginning of this chapter. We will come back to the first requirement later, for now we examine the second and third.

The second component requirement states that we must be able to incrementally compose components to get the delay of a path as the path grows in length. Growing paths in this way allows us to easily run shortest paths algorithms. To prove that LC Nodes meet this requirement, we first show that any path between source sink registers in the physical resource graph can be represented by a corresponding path through the LC Node graph.

Theorem 3.2.1 *Given $P_{phys} = (r_1, c_2, \ldots, c_{n-1}, r_n)$, a path that starts at a register and ends at a register, we can construct a corresponding path, $P_{LC} = (s_1, m_2, \ldots, m_{m-1}, e_m)$, in the LC Node* *graph that covers the same n physical resources with m LC Nodes.*

Proof LC Nodes represent short paths between physical nodes that are either registers or physical resources with fan-in greater than one. Thus, to construct the corresponding LC Node path, we find nodes in the original path that are either registers or have fan-in greater than one, extract the short paths between these special nodes, and find the corresponding LC Nodes that cover the same short path. By doing this in the order in which these special nodes appear in the original path, we can construct the corresponding LC Node path.

Since the original path has only two registers, one at either end, our LC Node path will have a Start Node at the beginning, an End Node as the last Node, and all other LC Nodes will be Mid Modes.

Corollary 3.2.2 *LC Nodes can be incrementally composed to construct any measurable path in the physical resources graph.*

Proof From Theorem 3.2.1, we know that every measurable path in the physical resource graph has a corresponding path in the LC Node graph. Therefore, by selecting, in-order, the LC Nodes that form a given path, we can incrementally construct the measurable path at the granularity of LC Nodes.

Assuming we have the delay of all LC Nodes, Corollary 3.2.2 shows that we can incrementally get the delay of any path at an LC Node granularity. Since, by definition, we do not care to know the delay of resources below the LC Node granularity, LC Nodes satisfy the second component condition.

The third component condition requires that components be small and of similar shape. LC Nodes, by definition, are as small as we care to measure. To show that they are of similar shape we appeal to the regularity of FPGAs. Although modern FPGAs are heterogeneous, containing logic blocks, embedded blocks, and advanced hierarchical interconnect, there's only one or a few logic block designs repeated throughout the FPGA. The same holds true for embedded blocks. Moreover, the interconnect follows a regular repeating pattern throughout the FPGA fabric with few to no exceptions. As such, when we construct the physical resource graph of an FPGA, we see many identical subgraphs repeated throughout. All repeated subgraphs will decompose into the same set of LC Nodes. Therefore, we will have many LC Nodes of the same shape, meeting the third component condition.

Finally, the first component requirement states that we must be able to compute the delay of components. To get the delay of a component, Timing Extraction measures the delay of a number of paths and takes a linear combination of these delays to get the component delay. Although LC Nodes meet the second and third component constraints, they fail the first.

An LC Node-based solution for Timing Extraction will use the measured delay of paths to compute a unique delay for each LC Node. It is easy, however, to construct a second solution given a first, demonstrating that we cannot know the delay of LC Nodes. Theorem 3.2.3 formalizes this.

Theorem 3.2.3 *For a given LC Node graph, at least two different solutions for the delay of LC Nodes exist when computed from measured path-delays.*

The proof hinges on the fact that every path has *exactly one* Start Node and *exactly one* End Node.

Proof Given a solution to the delay of all LC Nodes, we can construct a second, equally consistent solution as follows. First, increase the delay of every Start Node by α. Since a path uses exactly one Start Node, using the new Start Node delays to compute the delay of any one path will result in a path-delay that is exactly α greater than the real delay of that path. Thus we need to reduce the delay of some other node in the path by α to achieve the correct result. We do this by subtracting α from every End Node. Doing this will reduce the new delay by α since a path uses exactly one End Node. At this point we have a second solution for the delay of LC Nodes that is consistent with the measured path-delay.

3.3 Alternate Basis

LC Nodes fail the first component requirement. Finding a set of components that meet this requirement, however, is easy. This section explains how to accomplish this, for the reader versed in linear algebra however, suffices to say that we formulate Timing Extraction as a matrix and use a basis for the matrix as the set of components.

Timing Extraction measures the delay of multiple paths and takes linear combinations of these path-delays to compute component delays. We can formulate this problem as a path-node matrix where rows correspond to paths and columns to LC Nodes. An entry (i, j) is 1 if path j contains LC Node i, otherwise it is 0. Figure 3.6 shows this matrix for all paths between Start and End Nodes for the graph in Figure 3.5(b).

To find a set of components, we find a basis for the path-node matrix. In linear algebra, a basis is a set of linearly independent vectors that can be linearly combined to compose any vector in a given vector space. For example, a basis we are all familiar with comes from $\mathbb{R}^3$, i.e., three

$$
\begin{array}{cccc|cc|ccc}
\mathbf{S_1} & \mathbf{S_2} & \mathbf{S_3} & \mathbf{S_4} & \mathbf{M_1} & \mathbf{M_2} & \mathbf{E_1} & \mathbf{E_2} & \mathbf{E_3} \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
\multicolumn{4}{c}{\underbrace{\hphantom{0000000000000}}_{\text{Start Nodes}}} & \multicolumn{2}{c}{\underbrace{\hphantom{000000}}_{\text{Mid Nodes}}} & \multicolumn{3}{c}{\underbrace{\hphantom{000000000}}_{\text{End Nodes}}}
\end{array}
$$

Figure 3.6: Path-node matrix for Figure 3.5(b) with all 11 possible paths between Start and End Nodes.

$$
\begin{array}{ccccccccccc}
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & -1 \\
0 & \mathbf{1} & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & \mathbf{1} & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & \mathbf{1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Figure 3.7: Gaussian Elimination on the transpose path-node matrix in Figure 3.6. Highlighted in red are the six pivots.

dimensional space. Any point in $\mathbb{R}^3$ can be represented by a linear combination of the vector in the set $\{(1,0,0),(0,1,0),(0,0,1)\}$, this set is a basis for $\mathbb{R}^3$. For Timing Extraction, our vector space is the set of path. Thus, a basis of the path-node matrix gives the smallest set of components that can be linearly combined to form any path.

The result from the previous section can be interpreted as stating that the LC Nodes do not form a basis for the measurable paths of a circuit. Nevertheless, the path-node matrix presented above must have a non-empty vector space which comprises its basis. Finding the set of paths that form a basis for the paths in the circuit is as simple as performing Gaussian Elimination on the transpose of the path-node matrix. Figure 3.7 shows the results of this applied to the matrix from Figure 3.6.

This matrix has six pivots, columns where exactly one element is 1 and the rest are 0. These columns are 1, 2, 3, 4, 6, and 9. This means that paths 1, 2, 3, 4, 6, and 9 are linearly independent and together form a basis for the transpose of the path-node matrix. Therefore, if we measure the delay of these six paths, we can compute the delay of any other path by linear combinations of these paths. In essence, this procedure tells us the minimum number of paths that must be measured to calculate the delay of any path in the circuit. More than that, it tells us which paths to measure. Finally, it even tells us which linear combination of these six paths are necessary to compute the delay of the other five. For example, for the fifth path (column 5 in Figure 3.7) we see the equation for path 5: $p_5 = -p_2 + p_3 + p_4$.

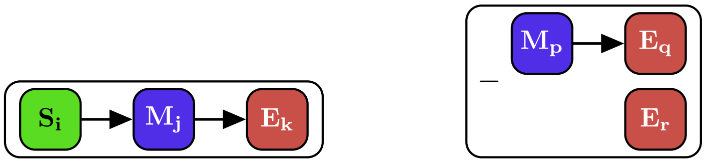
The number of paths to measure is exactly equal to the rank of the path-node matrix. When the number of paths exceeds the rank, we can perform the algorithm suggested above and reach a result where we measure rank number of paths and get enough information to compute the delay of any other path in the original path-node matrix. The rank of the path-node matrix will never be greater than the min(paths, nodes). Given the extensive connectivity available in an FPGA, the number of paths is orders of magnitude greater than the number of nodes. Thus, using this approach we will always be able to measure a small number of paths and recover the delay of all the paths in the path-node matrix.

Through this construction we see it is possible to meet the first component requirement. Unfortunately, this construction fails both the second and third component requirements. Although we can compute the delay of any path, we cannot do this incrementally since the delay of a path is computed from the linear combination of the delay of other paths, not from the delay of individual segments of the path. Second, the set of paths that we must measure may have paths of any length or shape. As such, we cannot directly compare the delay of one to the other and conclude that the difference is due to process variation only.

In the next section we present a solution that takes aspects from LC Node decomposition to meet the second and third component requirements and ideas from this section to make sure the decomposition forms a basis for the path-node matrix.

3.4 Discrete Units of Knowledge

LC Nodes represent the smallest components we care to measure. Since it is not possible to compute their delay, we define a new component, the Discrete Unit of Knowledge (DUK), as a small linear combination of LC Nodes. Unlike the previous solutions, DUKs meet all three component constraints. We first present two classes of DUKs and see how these DUKs compose to incrementally form any measurable path. Then we explain how to decompose an LC Node graph into individual DUKs. Finally, for each DUK, we determine the two or three paths that must be measured to compute its delay.



(a) M-DUK (b) C-DUK

Figure 3.8: Example of the shape of DUKs

3.4.1 DUK Types

The first DUK is the Mother DUK, or M-DUK. The Mother DUK is composed of a Start Node, zero, or a small number of Mid Nodes, and an End Node. Figure 3.8(a) shows an example of an M-DUK. As we will later see, M-DUKs are the shortest path from a Start Node to an End Node. The second type of DUK is the Child DUK, or C-DUK. Figure 3.8(a) shows an example of a Child DUK. A C-DUK is the difference of two sub-paths, where a sub path has zero or a small number of Mid Nodes followed by an End Node. Throughout this discussion we refer to the bottom path, the path subtracted, as the Current Tail. The other path is called the Desired Tail, the next section gives the intuition for these names. Because of the regularity of FPGAs, when we decompose an FPGA into DUKs, we will have a handful of M-DUK types and a handful of C-DUK types. Therefore it is easy to compare DUKs of one type and directly make conclusions about the process variation, meeting component constraint number three.

3.4.2 Incremental Path Computation

As their names imply, we build paths from DUKs beginning with a Mother DUK, followed by zero or more Child DUKs. Figure 3.9 shows an example of how combining the correct set of DUKs leads to an incrementally built path. A closer examination of what happens in Figure 3.9 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 the Current Tail of the C-DUK matches the tail of the path being built. As its name implies, the Desired Tail represents how the end of the path being built should change by removing the Current Tail Nodes and adding the Desired Tail Nodes. In a sense, *a C-DUK is like map directions that explains how to get from a path that ends with the Current Tail, to a path that ends with the Desired Tail.*

Thus, adding a C-DUK changes the tail of the path, extending, substituting, or shrinking it with new LC Nodes. Though a C-DUK changes the path, the result is always a path of the correct form, having a Start Node followed by zero or more Mid Nodes, and an End Node. For M-DUKs,

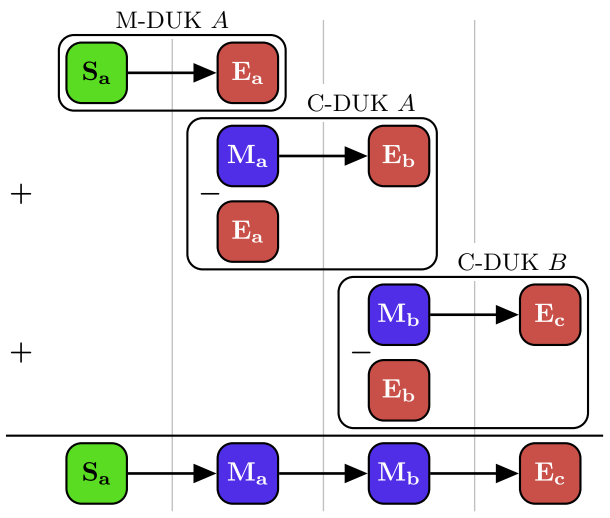


Figure 3.9: Demonstration of how an M-DUK plus two C-DUKs leads to a path with the correct form. LC Nodes represent both a set of resources and their delay.

the intuition is simpler, an M-DUK represents the shortest possible path that maintains correct form. It is this construction that allows us to incrementally compute the delay of paths and meet component constraint number two.

3.4.3 DUK Decomposition

With the definition and usage of DUKs established, we turn to the problem of decomposing an LC Node graph into individual M-DUKs and C-DUKs. The decomposition breaks into three steps. First we annotate the LC Node graph in such a way that any LC Node can easily find the shortest path between itself and an End Node. If more than one shortest path exist, we arbitrarily but *consistently* choose one. Step two extracts an M-DUK from each Start Node in the graph. Finally, we extract C-DUKs based on the sibling sets in the LC Node graph, a concept explained below.

3.4.3.1 LC Node Graph Annotation

To find the shortest path to an End Node for any LC Node we begin by pushing all End Nodes into a queue. We select the LC Node at the front of the queue, and mark all edges from it to each of its parent, unless the parent already has a marked edge. We then add to the queue all parents that had an output marked. Iterating until the queue is empty will mark exactly one output edge from each Start and Mid Node. The path traced from an LC Node through the marked edges, to an End Node will be the shortest path, in number of LC Nodes, from that LC Node to an End Node. We call this path the *marked path*. Marking the edges in this way allows us to remember and visit the same path every time, in case there was more than one shortest path. Algorithm 2 shows how edges are marked, while Algorithm 3 extracts the marked path of a given LC Node.

```
Algorithm 2: Mark Shortest End Node Path
  Input: LC Node Graph $G_{lc}$
  Queue $q \leftarrow$ GetEndNodes($G_{lc}$)
  while ! Empty ($q$) do
      node $n \leftarrow q$.Dequeue()
      foreach node $p \in$ Parents($n$) do
          if ! HasMarkedChild ($p$) then
              $p$.SetMarkedChild($n$)
              $q$.Enqueue($p$)
```

```
Algorithm 3: Get Marked Path
  Input: LC Node $n$
  Output: Path $p$
  $nextNode \leftarrow n$
  while !IsEndNode($nextNode$) do
      $p$.Add($nextNode$)
      $nextNode \leftarrow nextNode$.GetMarkedChild()
  $p$.Add($nextNode$)
```

To illustrate this process, we apply Algorithm 2 to the example LC Node graph from Figure 3.5(b). The result is show in Figure 3.10 where bold red edge show how the graph was annotated.

3.4.3.2 M-DUK Extraction

Once the LC Node graph is annotated, it is easy to extract M-DUKs. M-DUKs represent the shortest paths from a Start Node to some End Node. Therefore, we need only trace the marked path beginning at each Start Node. Algorithm 4 formalizes this, and Figure 3.11 shows the resulting four M-DUKs extracted from the annotated LC Node graph from Figure 3.10.

```
Algorithm 4: Extract M-DUKs
  Input: Annotated LC Node Graph $G_{lc}$
  Output: A set of M-DUKs $MD$
  foreach node $s \in$ GetStartNodes($G_{lc}$) do
      $MD$.AddMDUK(GetMarkedPath($s$))
```

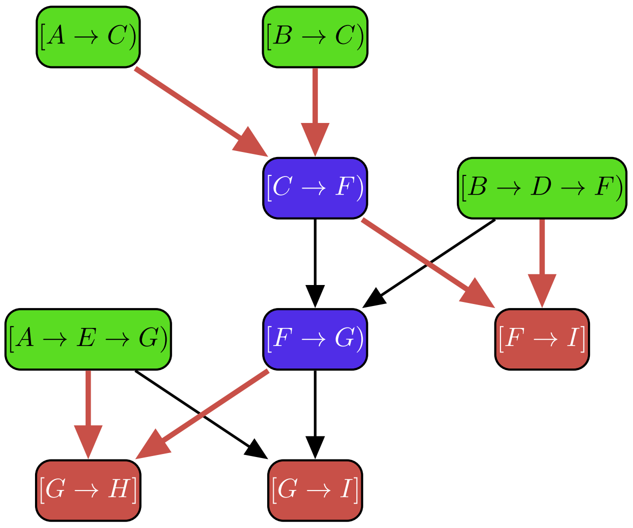


Figure 3.10: Result of applying Algorithm 2 to the example LC Node graph. Bold red edges show the edges marked by the algorithm

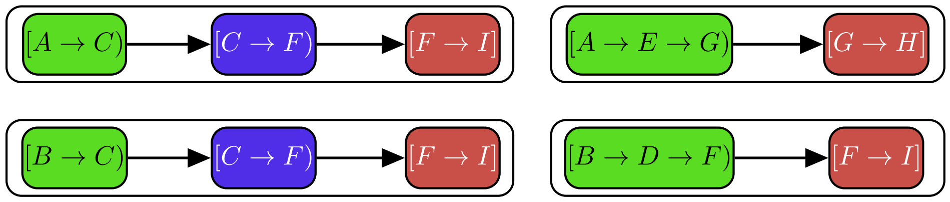


Figure 3.11: Four M-DUKs extracted from the LC Node graph in Figure 3.10 using Algorithm 4

3.4.3.3 C-DUK Extraction

Extracting C-DUKs requires that we first introduce the notion of a sibling set and its properties. A *sibling set* is a set of LC Nodes that share the same set of parents. Sibling sets have two important properties. First if the LC Nodes in a sibling set share any parent, they share all parents.

Theorem 3.4.1 *If siblings in an LC Node sibling set share an LC Node parent, they share all LC Node parents.*

A parent-child relationship between LC Nodes is established in Algorithm 1. It states that the last physical resources of a parent must be the same as the first physical resource of a child. As stated before, for convention, we assign the delay of this resources to the child LC Node, but it is the relationship this establishes that leads to the proof.

Proof If LC Nodes c_1 and c_2 have parent p_1 in common, it means that the last physical resource of p_1 and the first physical resources of both c_1 and c_2 are the same. Let this resources be r. If c_1 also has as a parent p_2, it again, means that the last physical resources of p_2 must be r. If this is the case, then there must also be a connection from p_2 to c_2, because of the way the last loop in Algorithm 1 defines the parent-child LC Node relationship.

We can this sibling property in Figure 3.10. For example, Nodes $[F \to G)$ and $[F \to I]$ have both nodes $[C \to F)$ and $[B \to D \to F)$ as parents.

The second property emerges after annotating the LC Node graph using Algorithm 2. It states that exactly one LC Node in a sibling set will have all its edges to its parents marked, no other LC Node in the sibling set will have any marked edges to their parents. Since sibling sets share all parents, when an LC Node in a sibling set is visited by Algorithm 2, either all its parents mark their edge to it, or all its parents already have marked edges. We refer to the LC Node in a sibling set with marked parent edges as the marked sibling. We see this with marked sibling $[F \to I]$ in the sibling set $\{$ $[F \to G)$, $[F \to I]$ $\}$ of Figure 3.10.

As stated before, a C-DUK can be thought of as instructions on how to get from a path that ends with the Current Tail, to a path that ends with the Desired Tail by removing from the end of a path the LC Nodes in the Current Tail, and attaching the LC Nodes in the Desired Tail. In order for this to work, both the Current Tail and the Desired Tail must have a common parent. Sibling sets, having common parents, are the ideal candidates for the beginning of Current and Desired Tails. For consistency, we always make the marked sibling be the beginning of the Current Tail, and each of the other siblings, forms the beginning of a Desired Tail. Pairing each Desired Tail with the Current Tail generates a C-DUK.

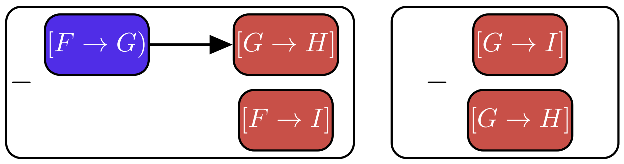


Figure 3.12: Two C-DUKs extracted from the LC Node graph in Figure 3.10 using Algorithm 5

More formally, given a sibling set with n LC Nodes, we create $n-1$ C-DUKs as follows. For each non-marked sibling in the sibling set, we create a C-DUK by subtracting its marked path, which will become the Desired Tail of the C-DUK, from the marked path of the marked sibling, identified as the Current Tail. Continuing with our example LC Node graph, we see that there are two sibling sets, { $[F \to G)$, $[F \to I]$ } and { $[G \to H]$, $[G \to I]$ }. Working with the first set, the marked sibling is $[F \to I]$, thus we trace the marked path of LC Node $[F \to G)$ and substract it from the marked path of $[F \to I]$. Figure 3.12 shows both this C-DUK on the left, and the C-DUK extracted from the other sibling set. Algorithm 5 formalizes this.

```
Algorithm 5: Extract C-DUKs
Input: Annotated LC Node Graph $G_{lc}$
Output: A set of C-DUKs $CD$
$siblingSets \leftarrow$ GetSiblingSets($G_{lc}$)
foreach set $SS \in siblingSets$ do
    $ms \leftarrow$ GetMarkedSibling($SS$)
    $pathMS \leftarrow$ GetMarkedPath($ms$)
    foreach $s \in \{SS - ms\}$ do
        $path \leftarrow$ GetMarkedPath($s$)
        $CD$.AddCDUK($path, pathMS$)
```

3.4.4 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.

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. 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 $\boxed{S_A}$ to $\boxed{E_D}$ in Figure 3.13. 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 Figure 3.13 we can measure the delay of the following three longer paths.

$$\begin{tikzcd}
\text{A:} & \boxed{S_1} \arrow[r] & \boxed{M_2} \arrow[r] & \boxed{M_3} \arrow[r] & \boxed{M_B} \arrow[r] & \boxed{M_C} \arrow[r] & \boxed{E_D} & & \\
\text{B:} & & & \boxed{S_A} \arrow[r] & \boxed{M_B} \arrow[r] & \boxed{M_C} \arrow[r] & \boxed{M_\alpha} \arrow[r] & \boxed{M_\beta} \arrow[r] & \boxed{E_\gamma} \\
\text{C:} & \boxed{S_1} \arrow[r] & \boxed{M_2} \arrow[r] & \boxed{M_3} \arrow[r] & \boxed{M_B} \arrow[r] & \boxed{M_C} \arrow[r] & \boxed{M_\alpha} \arrow[r] & \boxed{M_\beta} \arrow[r] & \boxed{E_\gamma}
\end{tikzcd}$$

Adding the delay of paths A and B, and subtracting from that the delay of C gives the desired M-DUK delay. Computing the M-DUK delay in this way requires that the structure in Figure 3.13 exist in the LC Node graph. Because of the way LC Nodes and the LC Node graph are constructed, Mid Nodes will always have fan-in and fan-out greater then one. Thus the graph structure around nodes $\boxed{M_B}$ and $\boxed{M_C}$ will always exists. What we cannot say in general is that the graph structure to extend a path to $\boxed{M_B}$ and from $\boxed{M_C}$ exist. However, both because FPGAs are large highly-connected circuits, and based on our successful application of Timing Extraction to off-the-shelf FPGAs (Chapter 4), we have very high expectations of being able to extend the paths as required. 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 Figure 3.8(b); 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

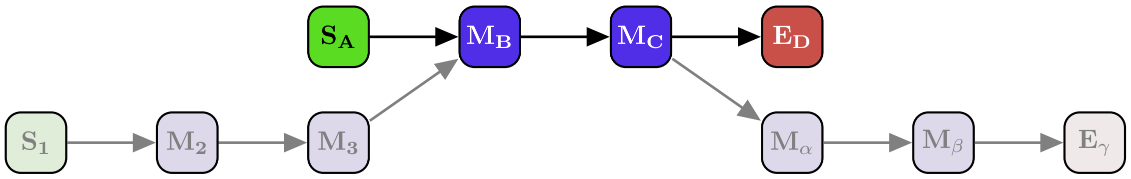


Figure 3.13: LC graph structure required to measure the delay of the M-DUK encompassing the path between nodes $\mathbf{S_A}$ and $\mathbf{E_D}$

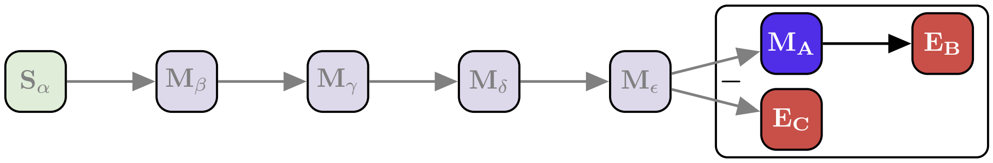


Figure 3.14: Example of the LC graph structure required to measure the delay of the C-DUK enclosed in the outline

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. Figure 3.14 shows an example of this. Taking the difference between the paths delays from $\mathbf{S}_\alpha$ to $\mathbf{E_B}$, and from $\mathbf{S}_\alpha$ to $\mathbf{E_C}$, leads to the delay of the C-DUK outlined in the figure.

In this way we determine which paths to measure for each DUK, and demonstrate that it is easy to compute the delay of DUKs from measured paths. With that we satisfy the first component requirement and have now shown that DUKs satisfy all three component requirements.

3.4.5 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. However, since we make no assumptions about the resources graph, the results in this section remain abstract and depend on the actual shape of the resource graph.

The number of M-DUKs equals the number of Start Nodes. The number of Start Nodes depends on both the number of registers, since a Start Node must begin at a register, and the structure of the resource graph. The number of Start Nodes generated from a register is equal to the fanout of that register.

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 of sibling sets equals the number of physical resources with fan-out greater than one. For each of these physical resources with fan-out greater then one, the size of the sibling set they generate will equal the size of their fanout.

Overall, we can bound the number of DUKs generated from a physical resource graph as no greater than the sum of the fanout of physical resources with fanout greater than one. Also, since a sibling set of size n leads to $n - 1$ C-DUKs, the number of DUKs will be strictly less than then number of LC Nodes.

3.4.6 DUK Graph

Once we have extracted all DUKs, as with LC Nodes, we can construct the DUK Graph, a new graph that represents the FPGA using DUKs. We connect an M-DUK to a C-DUK if the Current Tail of the C-DUK is the same as the tail of the LC Node path of the M-DUK. We do this because the C-DUK would remove these nodes from the path and add the Desired Tail nodes instead. Similarly we connect a parent C-DUK p to a child C-DUK c if the end of the Desired Tail of p matches the Current Tail of c. We do this for our example and show the result in Figure 3.15. The physical resource graph had 11 paths between registers. Keeping in mind that when using DUKs a path is formed by combining an M-DUK with zero or more C-DUKs, starting at M-DUKs, we enumerate the following 11 paths through the DUK Graph:

1. M-DUK 1
2. M-DUK 1 → C-DUK 1
3. M-DUK 1 → C-DUK 1 → C-DUK 2
4. M-DUK 2
5. M-DUK 2 → C-DUK 1
6. M-DUK 2 → C-DUK 1 → C-DUK 2
7. M-DUK 3
8. M-DUK 3 → C-DUK 1
9. M-DUK 3 → C-DUK 1 → C-DUK 2
10. M-DUK 4
11. M-DUK 4 → C-DUK 2

3.4.7 Routing on the DUK Graph

Part of the goal of Timing Extraction is to inform routing with DUK delays, to best map logic around the variation in an FPGA. Having the DUK Graph allows us to do this easily. Routing algorithms incrementally search for the shortest path between resources, hence our second component requirement. We show that with a minor modification, DUK Graphs are an ideal candidate for the Bellman-Ford shortest path algorithm [22].

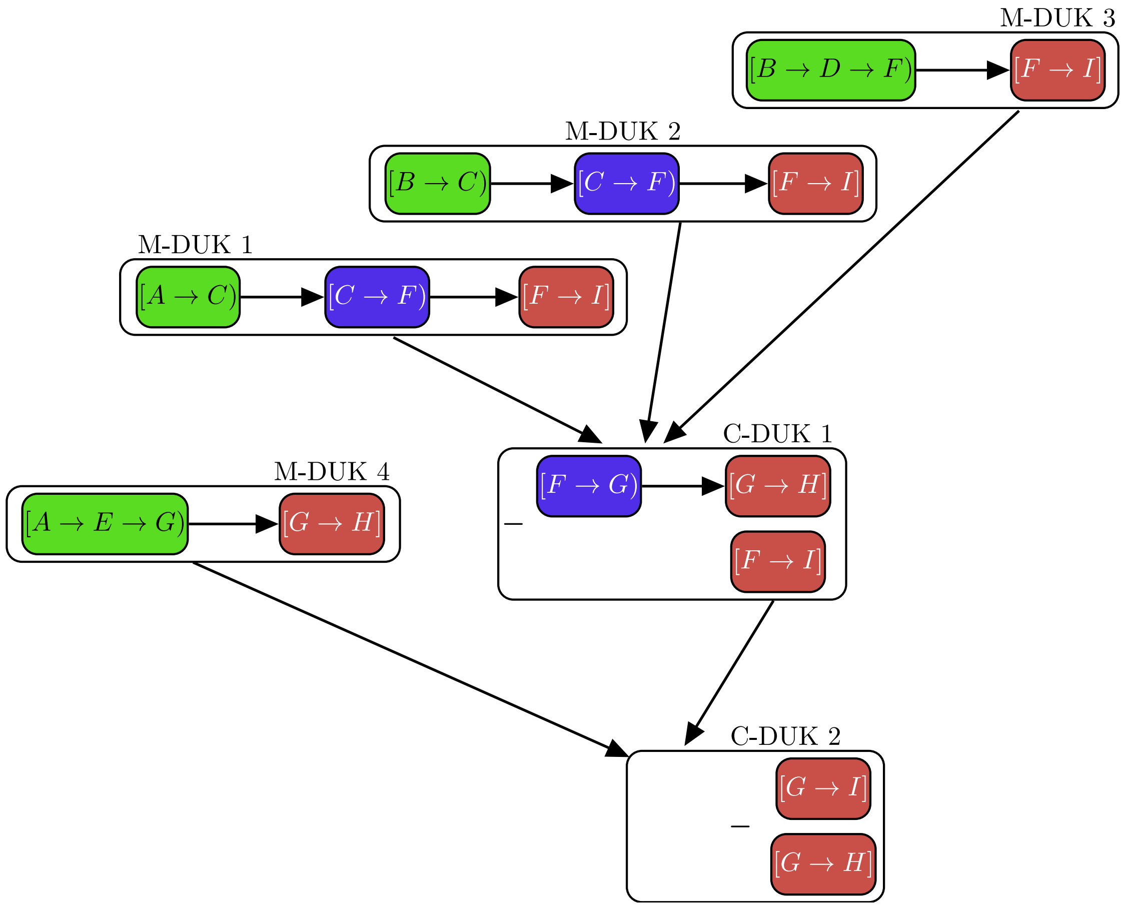


Figure 3.15: DUK Graph derived from the LC Node graph in Figure 3.10

We are interested in shortest paths between physical resources, however the nodes in the DUK graph are DUKs, not physical resources. Nevertheless, we can modify the DUK graph to easily find the paths between physical resources. First, we create a node for every source and sink register with a delay value of zero. We then add an edge from a source register node to an M-DUK if the M-DUK's path begins with that source register. Similarly, we add an edge from an M-DUK to a sink register node if the M-DUK's path ends with that sink register. Finally, we add an edge from a C-DUK to a sink register if the Desired Tail ends with that register.

Bellman-Ford looks at edge weights as it computes shortest paths. We, instead, have weights on the DUKs, not the edges. We easily rectify this by moving the weight from a node in the DUK Graph to all the fan-in edges of that node. Figure 3.16 shows what the final graph looks like. We annotated edges with delays by assuming some delay for each physical resource, and calculating

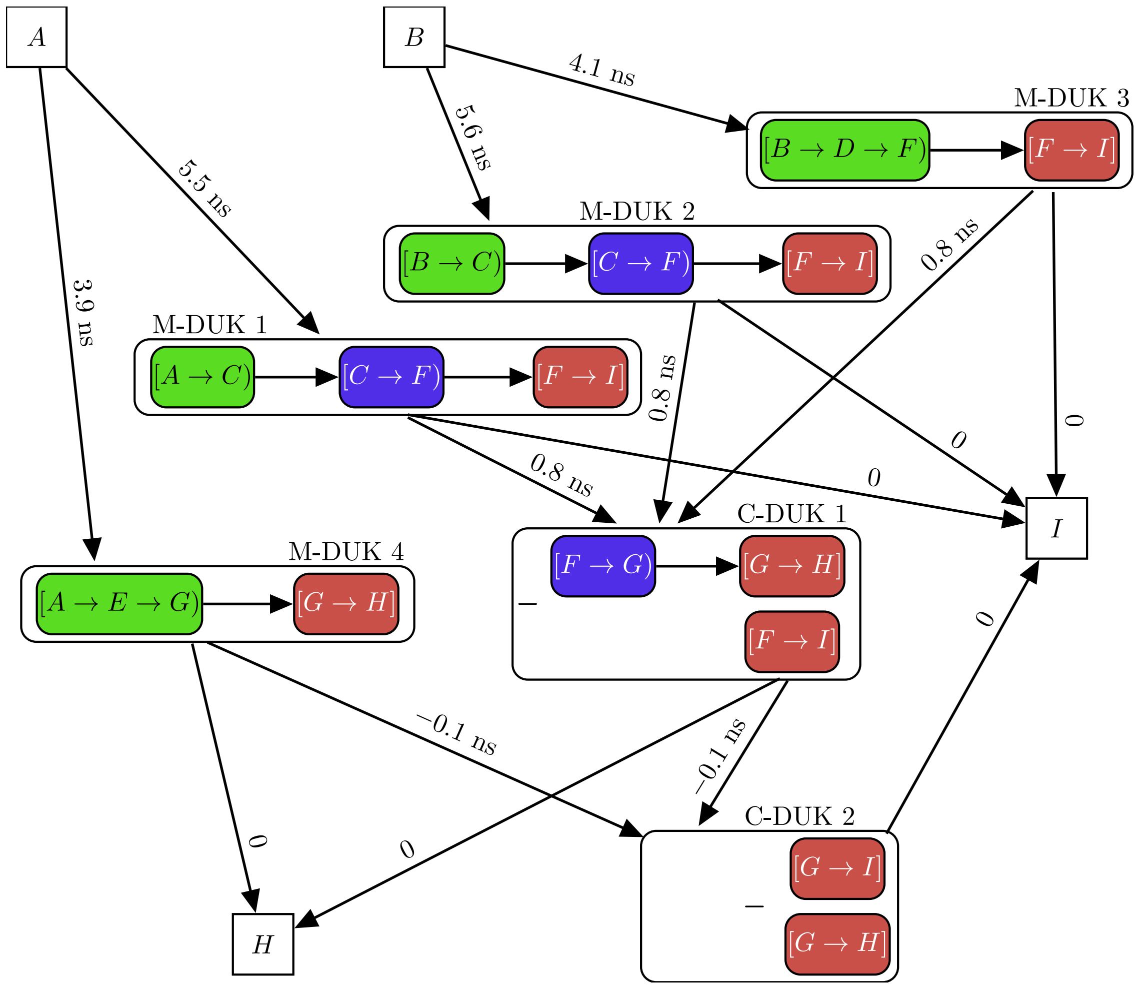


Figure 3.16: Modified DUK Graph with register nodes, ready for running Bellman-Ford. Edges are annotated with the delay of the DUK they point to

from them the correct DUK delays.

Although all physical resources were given positive delays, C-DUK 2 has a negative delay. This comes from the fact that C-DUKs are the difference of two sub-paths, and therefore, it is possible that their delay will be negative. A negative C-DUK delay simply indicates that the Current Tail is slower than the Desired Tail. Bellman-Ford can find shortest paths even with negative edge weights. It cannot find a shortest path if there are negative weight cycles. However, since the DUK Graph represents delays through an FPGA, it cannot have negative weight cycles, since there cannot be negative delays in the FPGA. Thus the modified DUK Graph is well suited for the Bellman-Ford shortest path algorithm.

3.5 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 4.12 demonstrates how our empirical results match the analysis presented here. As explained in Section 3.1, 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 3.1).

$$\tau_{measured} = \tau_{path} + \varepsilon_{path} \mid \varepsilon_{path} < \Delta_{clock} \tag{3.1}$$

To determine the error in DUK delays due to limited clock granularity, we refer to Section 3.4.4 which explains how to compute the delay of a DUK. A C-DUK's delay is the difference of the delay of two paths. Equation 3.2 expresses the delay and measurement error of a C-DUK in terms of the delay and measurement error of two paths, A and B.

$$\tau_{CDUK} + \varepsilon_{CDUK} = (\tau_A + \varepsilon_A) - (\tau_B + \varepsilon_B) \tag{3.2}$$

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 3.3.

$$-\Delta_{clock} < \varepsilon_{CDUK} < \Delta_{clock} \tag{3.3}$$

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 3.4 demonstrates.

$$\tau_{MDUK} + \varepsilon_{MDUK} = (\tau_A + \varepsilon_A) + (\tau_B + \varepsilon_B) - (\tau_C + \varepsilon_C) \tag{3.4}$$

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 3.5.

$$-\Delta_{clock} < \varepsilon_{MDUK} < 2 \cdot \Delta_{clock} \tag{3.5}$$

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, since there are few requirements set on the paths used to compute DUK delays. As Section 3.4.4 explains, given a DUK, we generate and measure two or three paths to compute the DUK delay. Some of the LC Nodes in these paths are determined by the DUK, others, however, can be arbitrarily selected from those that provide the required structure. Consequently, it is possible to formulate multiple sets of paths, that compute the delay of the same DUK, by changing the LC Nodes not constrained by the DUK. It is worth noting, however, that since every computed DUK must obey the error bounds of Equations 3.3 or 3.5, 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 the error not between a computed DUK and the actual DUK delay, as was done above, but the maximum error between two computed DUKs. We look to Equation 3.3 and determine that for a C-DUK, this error is $\pm 2 \cdot \Delta_{clock}$. For an M-DUK, Equation 3.5 leads us to conclude it is $\pm 3 \cdot \Delta_{clock}$. Throughout our results in Chapter 4, where appropriate, we highlight these bounds, and in particular, in Section 4.12 we uses these bounds to demonstrate the accuracy of our measurements.

Chapter 4

Timing Extraction on Commercial FPGAs

In this chapter we apply Timing Extraction to a commercially available FPGA, the Altera Cyclone III 65 nm FPGA. In order to understand the Timing Extraction results, we begin by presenting the architecture of the Cyclone III. With that knowledge, we introduce the structure of its LC Node graph, and subsequently examine the ten DUK types that emerge from the LC Node graph. Although Chapter 3 presents Timing Extraction, to implement it on the Cyclone III requires a few more steps, which we review next. At this point, we explain our experimental setup and present the main results, the delay distribution for the ten DUK types. The chapter continues by presenting results that either show the power of, or the need for Timing Extraction. We conclude with a sections on the runtime of Timing Extraction and the consistency of our measurements.

4.1 Cyclone Architecture

Section 2.1 introduced the basic architecture of an FPGA. Briefly, it is composed of logic blocks embedded in a routing fabric. The logic blocks, in turn, are composed of small lookup tables (LUTs), registers, and some local interconnect. The Cyclone III architecture follows this general structure. It is fabricated in a 65 nm low-power process technology, with a design optimized to further minimize power consumption [8]. Built on a mature, low-power process, it has a modest amount of process variation compared to state of the art FPGAs. Nevertheless, Timing Extraction is able to quantify this variation. We use Arrow's BeMicro FPGA Evaluation Kit with Cyclone III FPGAs model EP3C16F256C8N [14] (Figure 4.1). Table 4.1 gives the specific details for this model.

Figure 4.2 shows the logic block level layout of this FPGA. It is mostly composed of LABs with

Figure 4.1: The BeMicro FPGA Evaluation Kit with a Cyclone III FPGA.

Parameter	Value	Notes
Logic Blocks (LB)	963	Known as: Logic Array Block (LAB)
Dimensions (in LBs)	40×28	See Figure 4.2 for LB level layout
LUTs per LB	16	Configurable as carry chains
Registers per LB	16	Configurable as register chains
LUT Type	4-LUT	
Embedded Block RAMs	56	9Kb each
Embedded Multipliers	56	18×18 bits
PLLs (Phase-Locked Loops)	4	
Clock Networks	20	
Speed Grade	-8	Lowest Speed Grade
Max Clock Tree Performance	402 MHz	
Power Supply (V_{dd})	1.2 V	
Segment Lengths	4, 16, 24	Hierarchical
Operating Temperature	0 °C to 85 °C	
Process	65 nm	TSMC low-κ dielectric

Table 4.1: Device parameters for the Cyclone III EP3C16F256C8N on the BeMicro board [8]

two columns of embedded multipliers and two columns of embedded block RAMs. Surrounding the main blocks, are IO ports and PLLs for clock generation. Not shown is the interconnect consisting of a hierarchal network with length 4 vertical and horizontal routing segments that connect to both LABs and longer length 16 or 24 routing segments for quickly traversing long vertical or horizontal distances respectively.

4.1.1 LAB Architecture

Every Cyclone III LAB has 16 Logic Elements or LEs. An LE contains a 4-LUT and a register, as shown in the top right of Figure 4.3. Within the LAB there are also two interconnect routing networks, LAB Input Tracks (LITs), used to bring external signals into the LAB, and LAB Local Tracks (LLTs), that allows LEs within the LAB to communicate directly, as shown in Figure 4.3-center. Both the LITs and LLTs consists of depopulated crossbar connecting 16 or 38 routing

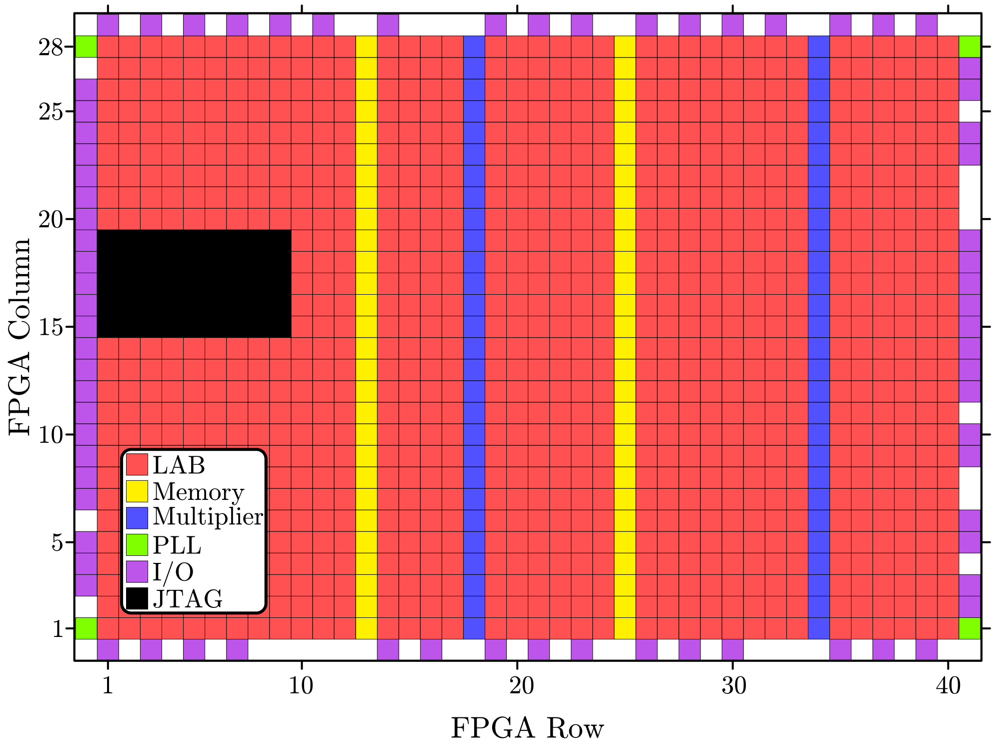


Figure 4.2: Cyclone III block-level floorplan.

tracks to the 16 LEs. Both crossbars have a 50% depopulation pattern between tracks and LEs, and is the same for every LE. We show this pattern for LLTs in the top left of Figure 4.3 and for LITs in the bottom. Essentially, either input A or input B can connect to a track, but not both. The same is true for inputs C and D. Thus inputs A and B form one *input set*, and C and D form the second. Finally, the output of an LE, which can be registered or not, can be programmed to connect to two separate LAB Output Buffers (LOBs), or back to the LLTs through one output track, as shown in the top left of Figure 4.3.

4.1.2 Interconnect Architecture

LABs in the Cyclone III live within a hierarchical interconnect. Each section of the interconnect is composed of directed wires bundled into structures identified by their orientation, C for column or vertical wires, and R for row or horizontal wires, and their length in units of LABs. E.g., a C4 wire bundle refers to a collection of vertical wires spanning 4 LABs. A LAB brings signals in from the general interconnect through LITs. It outputs signals using the LOBs. LOBs in the LAB can output directly to LITs in the two horizontally adjacent LABs, one on the left and one on the right. LOBs also output to C4 and R4 bundles and C4 and R4 bundles connect to LITs in the

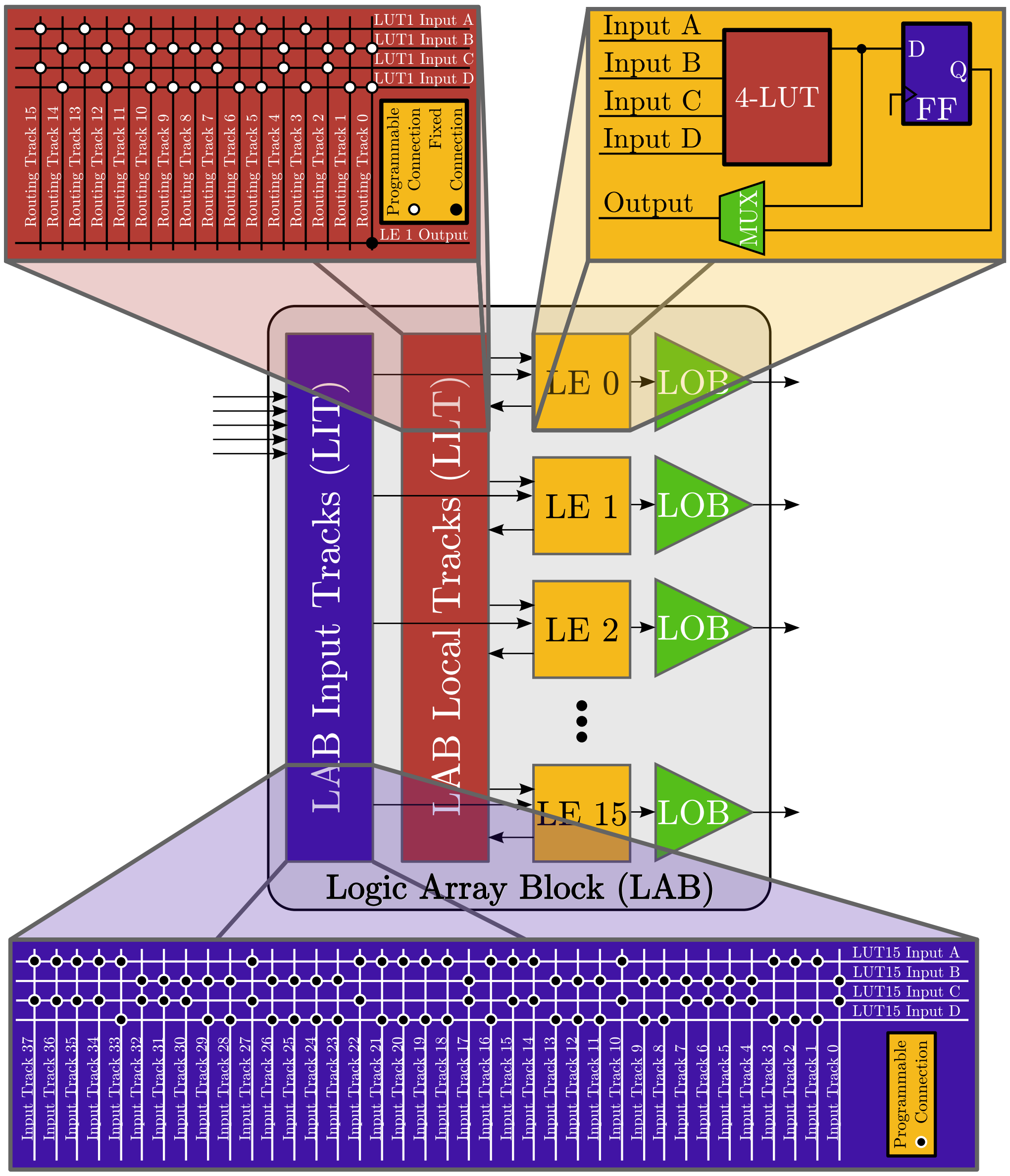


Figure 4.3: Detailed architecture of a Cyclone III LAB.

LAB. Figure 4.4 illustrates these connections. The hierarchal nature of the interconnect comes from the fact that C4 and R4 bundles can connect to C16 and R24 wires. Figure 4.5 summarizes these connections.

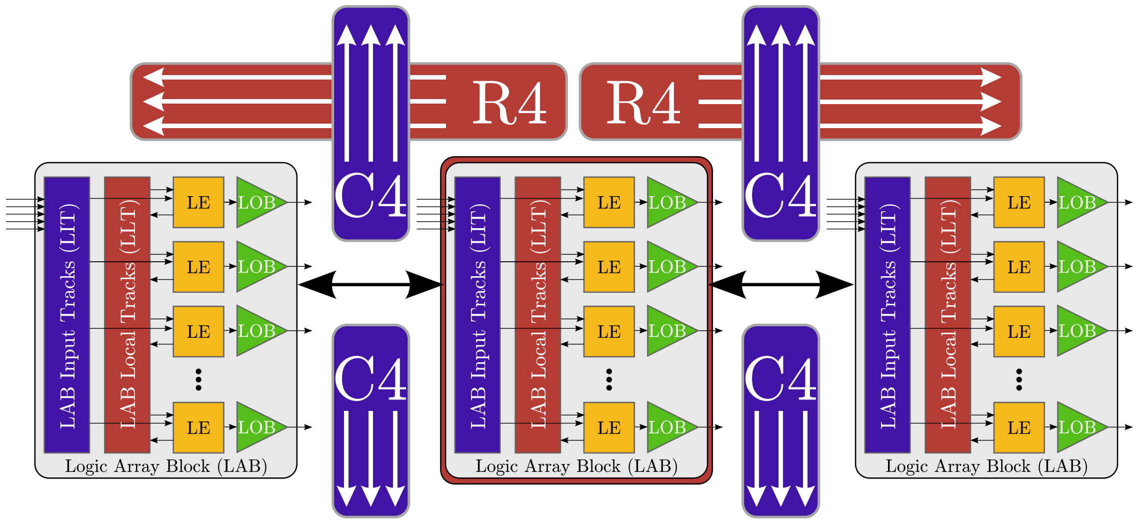


Figure 4.4: Interconnect Architecture of a Cyclone III. Showing connections for the central, highlighted LAB

Not every wire in a wire bundle can connect to every wire in the next bundle. As with LITs and LLTs in the LAB, the connections are depopulated such that a wire in one bundle can connect to only 2 to 3 wires in the next bundle. On average, we see 80% to 90% depopulation in these connections. Moreover, unlike LABs which have repeating patterns of connections, the connections between LABs and the general interconnect, and between structures in the general interconnect follow a loose pattern. Therefore, to generate the physical resource graph of the Cyclone III, we maintain a database that explicitly stores the unique connections between physical resources.

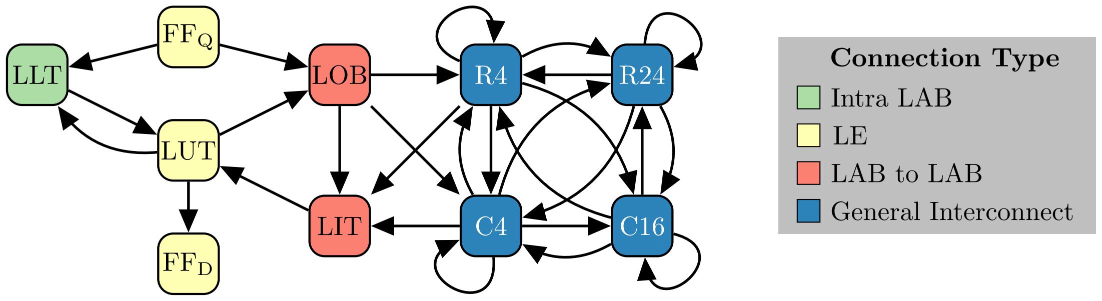


Figure 4.5: Connections between structures in the Cyclone III

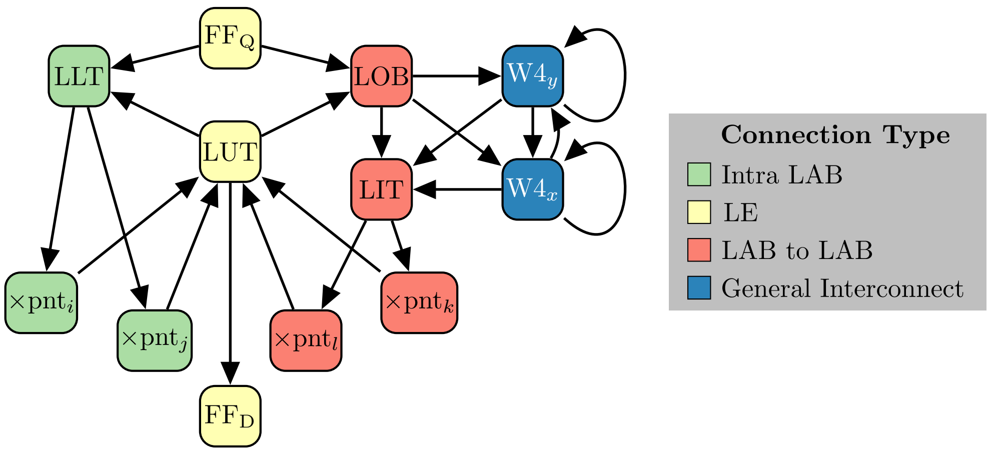


Figure 4.6: Physical resource graph for the Cyclone III

4.2 LC Graph

The first step towards implementing Timing Extraction on the Cyclone architecture is to form the physical resource graph from the description above. From that we can construct the LC graph. For the results presented later in this dissertation, we construct the complete physical resource graph using the database mentioned in the previous section, representing every physical resource we want to measure. We then process this graph through Timing Extraction and extract the correct DUKs and paths to measure. However, in order to make sense of the resulting DUKs, here, and in the next section, we analyze a region of the physical resource graph representative of a physical region akin to an FPGA tile. The regularity of the FPGA makes it so that the conclusions we reached here are applicable to the complete resource graph later.

Figure 4.6 shows this representative graph. Although similar to Figure 4.5, since it is a physical resource graph, we show wires as W4 nodes, instead of wire bundles, and also include nodes for LLT and LIT crosspoints. Moreover, we capture the interconnectivity between wires using the loops and edges between the W4 nodes. However, we limit our graph to only length 4 wires. This simplifies the graph so that we may easily make sense of the resulting DUKs. Finally, since we only show one or two nodes for each resource, it is useful to think of the nodes as representing a resource type standing for potentially multiple resources.

From the graph in Figure 4.6 we construct the LC Node graph using Algorithm 1, as shown in Figure 4.7. The result has two Start Nodes, one for starting a path that initially goes through other resources in the LAB using LLTs, and one to leave the LAB through an LOB. Sink registers are represented by the one End Node. Mid Nodes are classified into three categories, showing

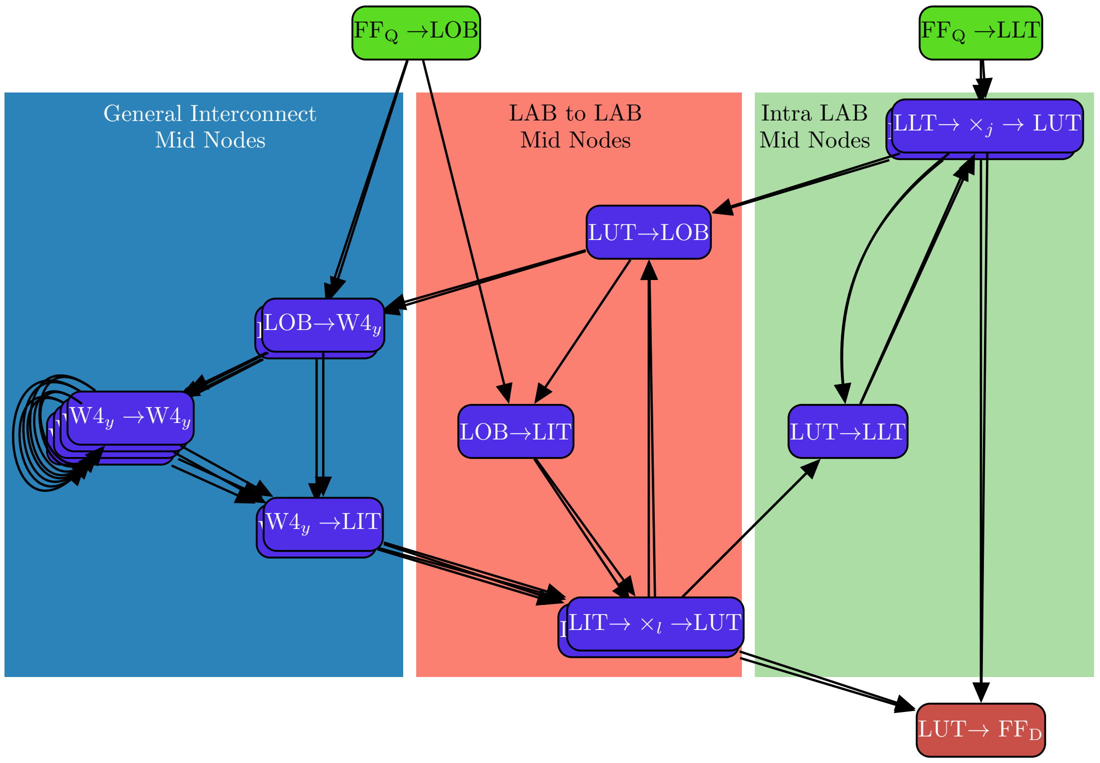


Figure 4.7: LC Node graph for the Cyclone III. LC Nodes with crosspoints or wires as physical resources are shown overlapped to signal all versions of that node generated by substituting the correct crosspoints or wires.

those Mid Nodes used to create routes in the general interconnect, between LABs, and within LABs. Overlapped LC Nodes represent LC Nodes that have the same shape (i.e., cover the same type of physical resources) but are not identlical. E.g., different versions of $LIT \rightarrow \times \rightarrow LUT$ can use different crosspoints. The same holds true for those LC Nodes that cover a W4 wire resource.

4.3 DUKs

We use the LC Node graph to extract the DUKs for the Cyclone III architecture by first marking edges (Algorithm 2), extracting M-DUKs (Algorithm 4), and finally, extracting C-DUKs (Algorithm 5). Figure 4.8 shows the result of marking the edges. This figure also tags the five sibling sets that will be used to extract C-DUKs.

Since the LC Node graph has two Start Nodes, we extract two M-DUKs by tracing their marked path. Figure 4.9 shows the result. The Intra-LAB M-DUK is used to begin a path that first uses local LAB resources through the LLTs. The LAB-to-LAB M-DUK allows us to leave the LAB and route to a horizontally adjacent LAB. With the correct C-DUK, this M-DUK will

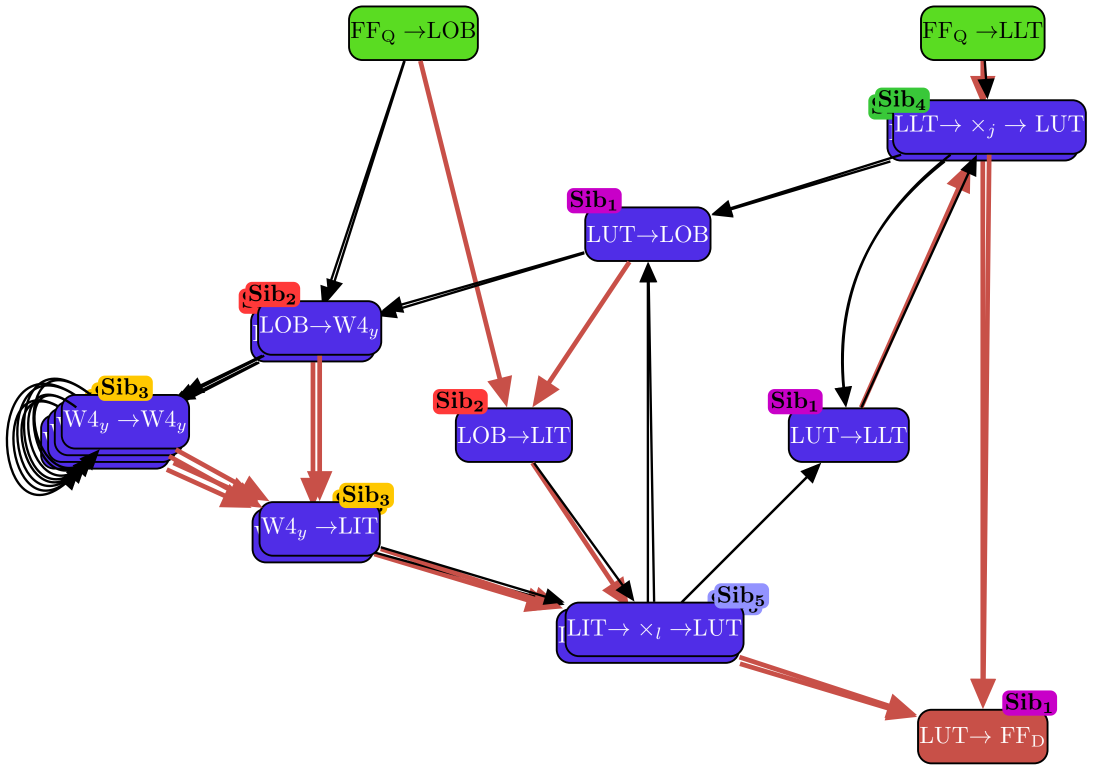


Figure 4.8: Annotated LC Node graph for the Cyclone III. Bold red edges show marked edges by Algorithm 2. Nodes belonging to a sibling set with more than one Node have tags $\mathbf{Sib_x}$, indicating which sibling set they belong to

also initiate a route that goes directly onto the general interconnect after leaving the LAB.

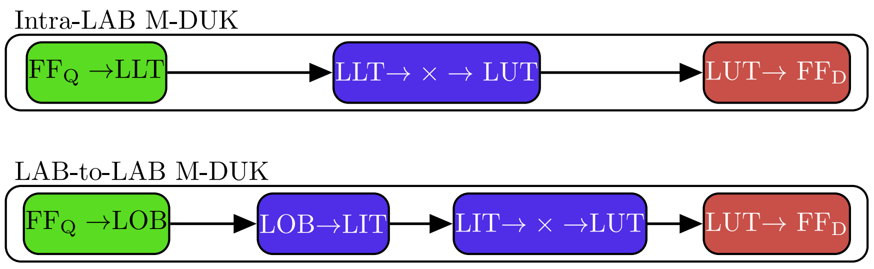


Figure 4.9: M-DUKs for the Cyclone III

To extract C-DUKs, we first look for sibling sets. The LC Node graph has five:

Sib$_1$: { LUT→ FF$_D$, LUT→LLT , LUT→LOB }

Sib$_2$: { LOB→LIT , LOB→W4 }

Sib$_3$: { W4→LIT , W4→W4 }

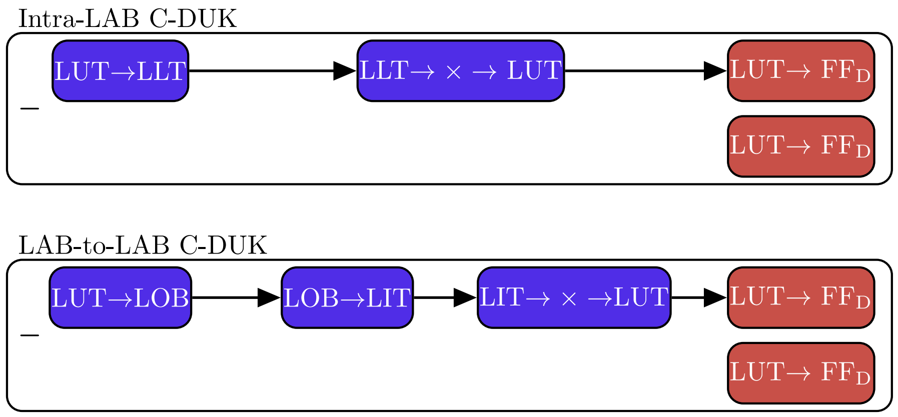


Figure 4.10: C-DUKs from sibling set 1 for the Cyclone III

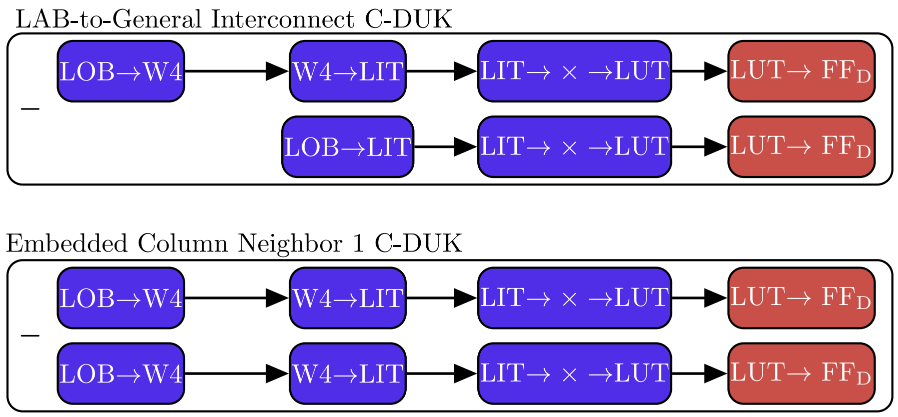


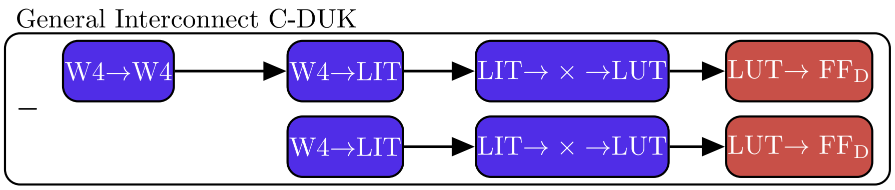
Figure 4.11: C-DUKs from sibling set 2 for the Cyclone III

Sib$_4$: { LLT$\rightarrow \times_i \rightarrow$LUT, LLT$\rightarrow \times_j \rightarrow$LUT, . . . }

Sib$_5$: { LIT$\rightarrow \times_k \rightarrow$LUT, LIT$\rightarrow \times_l \rightarrow$LUT, . . . }

Figures 4.10 through 4.14 show the resulting C-DUKs for each sibling set. In this section we are concerned with the shape of each C-DUK. I.e., understanding what resources each C-DUK uses, and how. Thus we do not explicitly differentiate between two LC Nodes with the same shape. For example, when both the Desired Tail and Current Tail of the C-DUK use a Mid Node such as W4→LIT, it is meant to indicate that both Tails have a Mid Node composed of a W4 wire followed by a LIT. The actual W4 and LIT in one Tail is likely to be different than the W4 and LIT in the other Tail.

The name of each C-DUK gives a hint of its function. To understand them better, consider the effects each C-DUK has on a path. Not every path can be extended with any C-DUK. To extend a path, the end of the original path must match the Current Tail of the C-DUK. For convenience



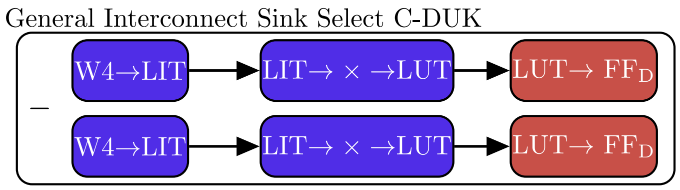


Figure 4.12: C-DUKs from sibling set 3 for the Cyclone III

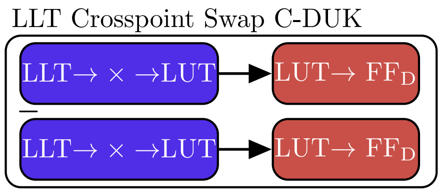


Figure 4.13: C-DUK from sibling set 4 for the Cyclone III

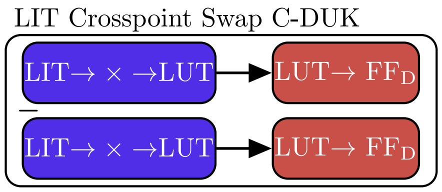


Figure 4.14: C-DUK from sibling set 5 for the Cyclone III

we call the LAB where the path ends the Last LAB.

- **Intra-LAB C-DUK:** This C-DUK allows us to extend the path through resources within the Last LAB.
- **LAB-to-LAB C-DUK:** Similarly, this DUK extends the path to a LAB horizontally adjacent to the Last LAB.
- **LAB-to-General Interconnect C-DUK:** If a path ends by going from one LAB to a horizontally adjacent LAB, this DUK modifies the path so that instead of ending by going to a horizontally adjacent LAB, the path ends by using a general interconnect routing resource and then going to some LAB.
- **Embedded Column Neighbor C-DUK:** This C-DUK is similar to the LAB-to-General Interconnect C-DUK in that both have the same Desired Tail shape. The difference is that the Current Tail of the LAB-to-General Interconnect C-DUK is a path between two horizontally adjacent LABs, while for this C-DUK, it is a path through one W4 resource wire instead. The reason is that, as its name suggests, these DUKs only exist next to embedded columns (Figure 4.2), and therefore, there is no horizontally adjacent LAB for the Current Tail to use.
- **General Interconnect C-DUK:** This C-DUK works on a path that ends by using a general interconnect routing resource, and then goes into a LAB to an End Node. It is this DUK that allows us to expands the path through another general interconnect routing resource.
- **General Interconnect Sink Select C-DUK:** General interconnect resources can connect to a number of LABs. This DUK lets us choose which of these LABs a general interconnect resource connects to.
- **LLT Crosspoint Swap C-DUK:** All previous DUKs enter a LAB through one of two resources, either a particular LLT or a particular LIT. This DUK allows us to change which LLT resources, and consequently, which LUT and register, is used.
- **LIT Crosspoint Swap C-DUK:** Similarly, this DUK allows us to change which LIT, LUT, and register is used.

Thus, with two types of M-DUKs and eight types of C-DUKs, we can represent any path through the Cyclone III. In a later section we quantify how many DUKs of each type are created from the complete physical resource graph of the Cyclone III. Moreover, we further classify these

DUKs based on known systematic differences, such as the LUT input used, or whether it uses a horizontal or vertical general interconnect wire. Nevertheless, our analysis here correctly captures the function of these DUKs.

Having a limited number of DUK types implies that we will have many DUKs of each type. This will allow us to compare DUKs within one type, and directly reach conclusions about the variation in the FPGA. Chapter 5 depends on there being orders of magnitude more DUKs than DUK types, to successfully analyze the variation in the FPGA.

4.4 CAD Flow

Timing Extraction explains how to decompose a physical resource graph into DUKs. It also determines which paths should be measured to compute the DUK delays. However, there are some practical consideration we must take into account when actually applying timing extraction to a real FPGA. In this section, we examine the complete CAD flow that allows us to go from a physical resource graph of a Cyclone III to DUK delays.

At a high level, the CAD flow consists of 8 stages:

1. **LC Node Graph:** Processes the physical resource graph and constructs the LC Node graph.
2. **DUK Extraction:** Extracts DUKs from the LC Node graph
3. **Path Extraction:** Determines which paths to measure
4. **Path Packing:** Packs multiple paths into one bitstream
5. **Constraint Generation:** Generates the placement and routing constraints for each bitstream, based on the results of path packing.
6. **Bitstream Compilation:** Compiles Verilog, a hardware description language, and placement and routing constraints into an executable bitstream compatible with the Cyclone III.
7. **Path Measurement:** Runs each compiled bitstream through the FPGA and records the delay of each path measured.
8. **DUK Computation:** Computes the delay of each DUK using the measured path delays.

We have introduced the first 3 steps as the main function of Timing Extraction. We have also described how DUK delays are computed from path delays (last CAD step). Here, we explain the

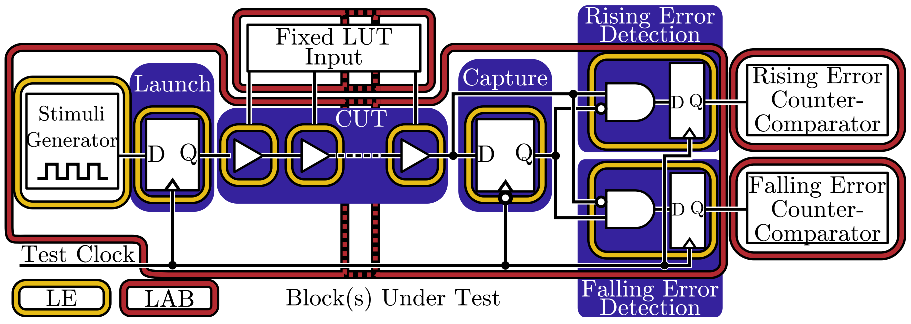


Figure 4.15: Placement of Path-delay measurement circuit in LABs and LUTs.

intermediate 4 steps. To do so, we first look at the circuit modules we must implement on the FPGA to both measure path delays and manage the measurement process.

4.4.1 Path Measurement Circuit

Although we must measure multiple paths, not every path uses every LAB. Therefore, we should be able to measure multiple paths at a time. Figure 4.15 shows how the path-measurement circuit, explained in Figure 3.2 on page 26, is placed on LABs. For the LAB or LABs that contain the actual path, or CUT, measured, we see how each block fits inside LEs. The number of LABs used by the Blocks Under Test will depend on the path being measured. In practice this is between one and two LABs. This number will be the same for the Fixed LUT Input block. Since each buffer in the CUT uses only one out of the four LUT inputs, the Fixed LUT Input block makes sure the unused LUT inputs are consistently set. Section 4.8 explores how changing these unused inputs changes the delay of the path, and yields insight into the internal structure of the LUT. Finally, the rising and falling error counter-comparator blocks keep track of how many errors have occurred in each transition, and each reports whether that number is above a threshold. The number of LABs these counter-comparators need depend on the size of that threshold. For example, if the threshold is set to 50% errors, or 2^{n-1} out of 2^n iterations, each counter-comparator will require n LEs. As previously mentioned in Section 3.1, we set $n = 15$. Thus, our error counter-comparators altogether require 30 LEs, which fit in two LABs. Therefore, each measurement circuit will use resources in 4 to 6 LABs only. Since our FPGA has 963 LABs, we can easily measure multiple paths at a time.

In theory, we should be able to place about 963 LABs/6 LABs per measurement circuit ≈ 160 path measurement circuits in a bitstream. In practice, this number is limited for two reasons. First, it is not sufficient to simply measure the delay of the path, we must use the FPGA to report the measured delay. This requires dedicated circuitry that allows the FPGA to communicate with

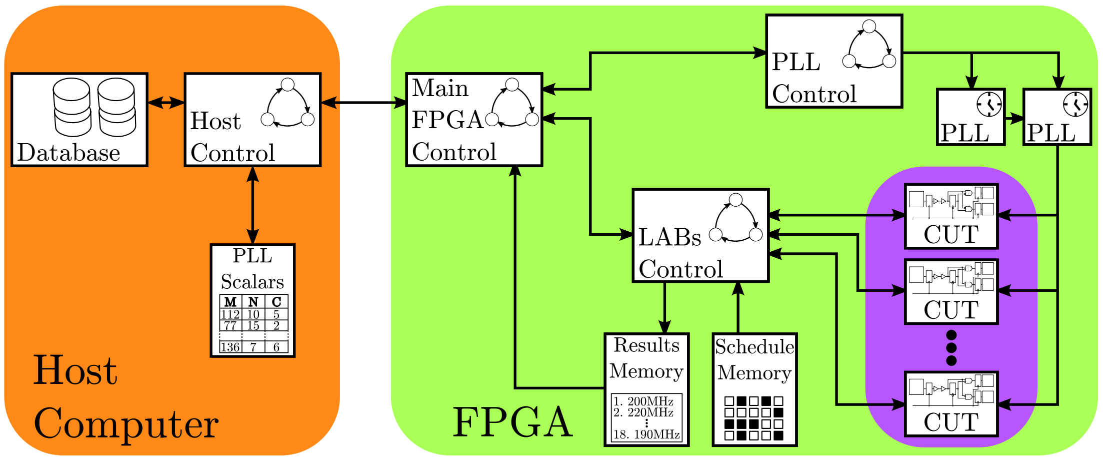


Figure 4.16: Block diagram of the full Timing Extraction system used to characterize delays in the Cyclone III FPGA.

the host computer where the FPGA is connected (Section 4.4.2). For our measurements, this leaves a region of 22 × 28 LABs. Second, our experiments show the need to carefully control the placement, routing and activity around the path being measured in order to get accurate, repeatable results (Section 4.12). We do this in part by creating a conservative isolation zone of 2 LABs around the path measurement circuit. No resources in the LABs in this region are used. When we account for this isolation region, the minimum number of labs reserved for a path measurement circuit is between 6 × 6 and 6 × 7. Therefore, at most, we can pack between $\frac{22\times28}{6\times7} \approxeq 14$ to $\frac{22\times28}{6\times6} \approxeq 17$ path measurement circuits. In practice we fix this number to at most 15 paths per bitstream.

4.4.2 Control and Communication Module

Although crucial, the path measurement circuit is just one part of the complete circuitry required to measure and report the delay of paths. Figure 4.16 demonstrates the complete circuit block diagram. Here we examine each of its sections.

In order to get the delay of the CUT, it is necessary to change the frequency on the test clock from f_{min} to f_{max}. The way to control the clock frequency is through the use of special blocks implementing Phase-Locked Loops (PLLs). The Cyclone III has four PLLs. The PLL works by locking onto the phase of the source clock and adjusting its frequency by a multiple factor. Equation 4.1 shows the parameters governing the frequency scaling produced by the PLL. M and N pre-scale the input frequency and C post-scales the final result. All three can have values ranging from 1 to 512 [8].

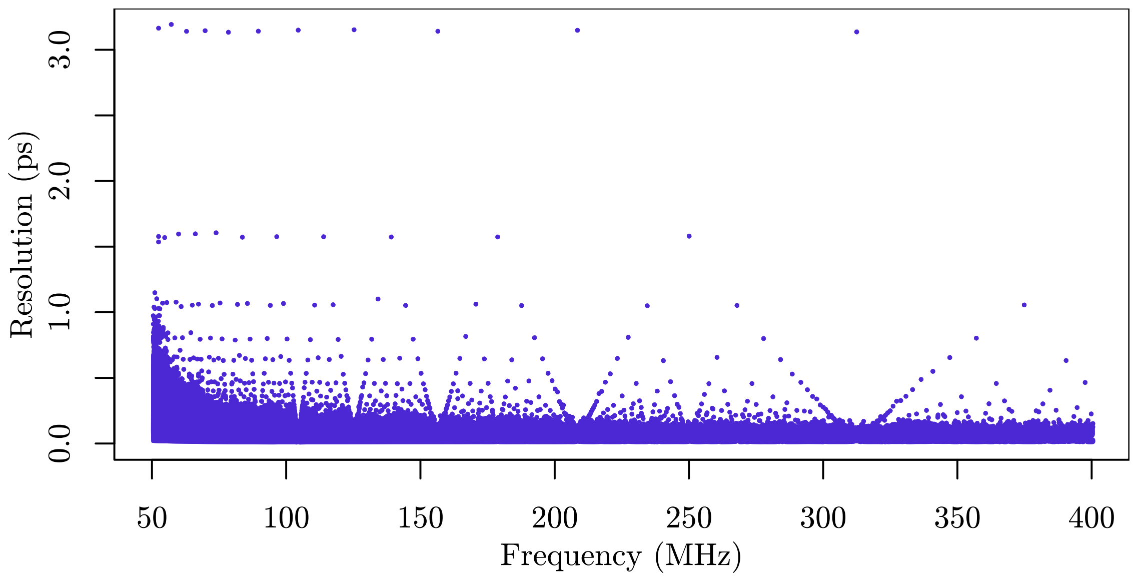


Figure 4.17: PLL resolution showing the smallest delay increment at a particular frequency.

$$f_{out} = f_{in} \cdot \frac{M}{N \cdot C} \tag{4.1}$$

For finer grained frequency granularity, we cascade the output of one PLL to the input of a second PLL, giving us greater control over the scaling of the clock. Nevertheless, the fact that the scaling factors must be integer numbers between 1 and 512, quantizes the possible frequencies we can generate. Moreover, due to details beyond the scope of this work, certain combinations of these scalars cause the PLLs not to lock onto the reference frequency. Permuting over all possible scaler combinations, we can determine the minimum step size for any frequency. Figure 4.17 shows this result for frequencies between 50 MHz and 400 MHz. As we can see, although there are many frequencies for which a very small step size is possible, there are some that are as large as 3.2 ps. Therefore, we sweep the frequency at a linear rate of 3.2 ps.

Even through our step size is 3.2 ps, we actually achieve half that step size because of the way the launch and capture registers are clocked. Notice that the capture register in Figure 4.15 is clocked on the opposite clock edge to the launch register. This means that a signal has half the cycle time to travel from the launch register to the capture register. Therefore, for our measurements, the Δ_{clock} presented in Section 3.5, is 1.6 ps.

The frequency scaling factors given to the PLLs are reconfigurable at runtime. This means that it is not necessary to stop the FPGA, clear it and load a new configuration, or bitstream, to change the frequency. Instead, we have a module that receives new scalar factors from the host machine, applies these values to the PLL and reports back to the host if the PLLs were successful at changing the frequency as the circuit is running. To speed the operation, the host pre-computes the set of scalars that will be necessary for the programmed run.

For our experiments, we sweep the frequency from 50 MHz to 400 MHz, since we double the clock speed by clocking the capture register on the opposite clock edge, this translates into paths that are between 1.25 ns and 5 ns. The maximum frequency for the Cyclon III is 402.5 MHz, which translates into 1.24 ns. In [31] we conducted preliminary measurements and discovered that the average delay of an IntraLAB C-DUK that uses LUT input D is 236 ps. The average delay for a similar IntraLAB M-DUK is approximately twice that. Therefore, the shortest path we can measure must be composed of one IntraLAB M-DUK and four IntraLAB C-DUKs, i.e., 5 DUKs long. Since the M-DUKs generated for the Cyclone III are 3 or 4 DUKs long, we use the three path measurement approach described in Section 3.4.4 to compute the delay of an M-DUK.

Since we can place multiple path measurement circuits in one bitstream, we add a control unit that tells the measurement circuits to start testing at the set frequency and records in a memory block which CUTs were too slow for that frequency. This control unit also allows us to schedule which path is measured when. This way we can minimize the activity in the FPGA during a measurement, which minimizes the effects of operational and environmental variation on our measurements.

Finally, we need a module that coordinates the PLL module and the LABs module as well as the communication with the machine hosting the FPGA. This module consists of a state machine that controls when a new frequency should be requested from the host, when LABs should initiate tests at the given frequency, and how information flows between the host and the FPGA. Once all LABs fail at a high enough frequency, it reports back to the host at which frequency each tested path failed.

4.4.3 Path Packing

Understanding the complete measurement circuit, justifies the path packing CAD step. Once Timing Extraction determines which paths to measure, path packing creates sets with at most 15 paths. The paths are selected so that no two paths share a LAB. Moreover, we reserve an isolation region two LABs thick around the bounding box for the path measurement circuit.

4.4.4 Constraint Generation

As Section 2.1.1 reviewed, the FPGA CAD tools process a high level description of the computation and generates a bitstream that can be mapped on the FPGA. In order to correctly measure our

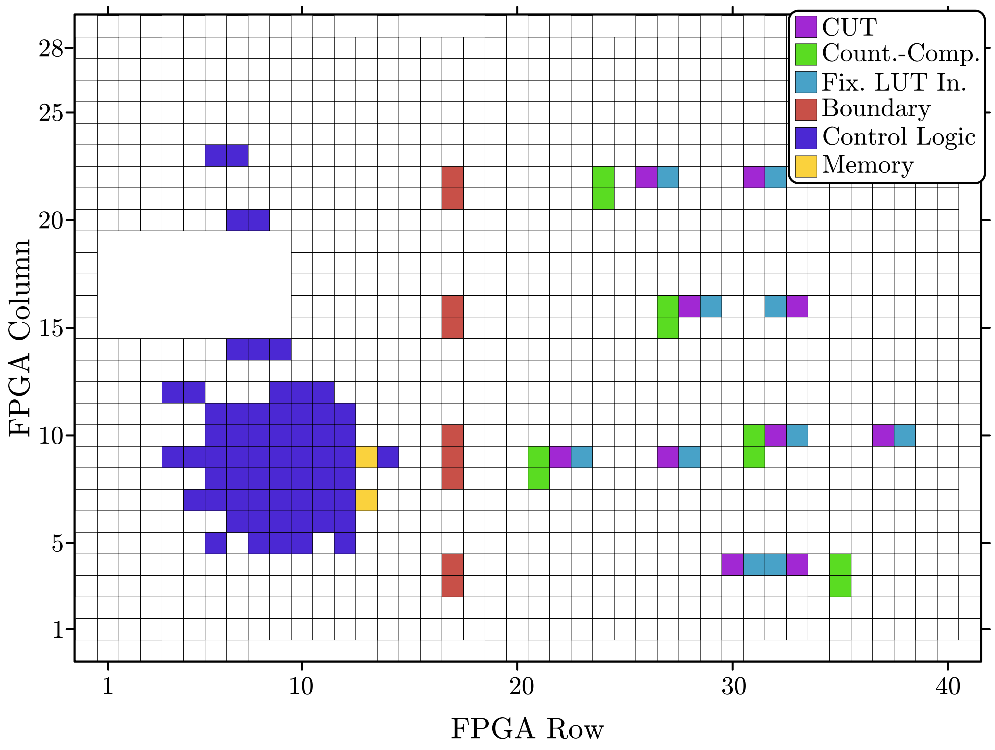


Figure 4.18: Placement of full system on the Cyclone III for a bitstream with 5 path measurement circuits.

paths, we must generate placement and routing constraints that force the FPGA CAD tool to use the physical resources we want to measure. Quartus [9], the Altera CAD tool suite, uses QUIP [6] or Quartus II University Interface Program to expose an interface allowing users to specify these routing and placement constraints. The Constraint Generation step generates the placement and routing constraints for each of the packed path sets from the previous step.

Figure 4.18 shows a floorplan of the Cyclone III, highlighting the placement of the path measurement circuit and the overall control logic into LABs. Part of the placement constraint isolates the control logic on the left, from the path measurement circuits on the right. The figure shows a bitstream with five path measurement circuits.

Figure 4.19 shows the resulting routing for this bitstream. Again, as part of the constraints, we limit the wires that can cross between the control logic and the path measurement region.

4.4.5 Bitstream Compilation and Path Measurement

Once the constraints are generated, we compile one bitstream for each packed path set. Quartus takes a Verilog description of the path measurement circuits and the control logic. It also takes

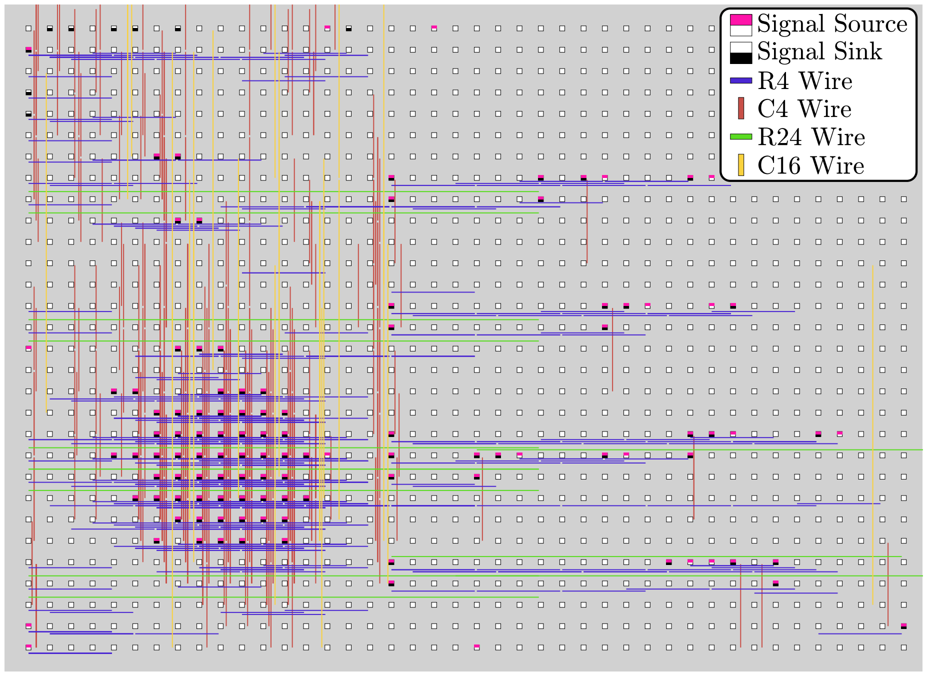


Figure 4.19: Routing resources of full system on the Cyclone III for a bitstream with 5 path measurement circuits

the constraint files generated in the previous step, and compiles a bitstream. Each compiled bitstream runs on the FPGA and, along with the host computer, records the measured path delays in a centralized database

4.4.6 DUK Computation

Finally, using the measured path delays, we compute DUK delays as described in Section 3.4.4. These delays are then recorded in the database.

4.5 Experimental Setup

We applied Timing Extraction to the architecture of the Altera Cyclone III, and computed DUKs delay for 18 Cyclone III FPGAs on Arrow's BeMicro boards. We characterize the right side of the FPGA, starting at LAB column 19. We also constraint the control logic to columns 1 to 14. Column 17 is used as the boundary crossing between the control logic and the path measurement circuits. These constraints produce bitstreams like the one shown in Figure 4.18.

The region measured spans from LAB (19,1) to LAB (40,28), encompassing all 560 LABs on the right side of the FPGA, and associated interconnect. We generated the physical resource

M-DUK Type	M-DUK Count
Intra-LAB	8,960
LAB-to-LAB	15,232
Total M-DUKs	**24,192**

C-DUK Type	C-DUK Count
Intra-LAB	8,988
LAB-to-LAB	15,260
LAB-to-General Interconnect	78,523
Embedded Column Neighbor	17,117
General Interconnect	254,077
General Interconnect Sink Select	131,474
LLT Crosspoint Swap	259,840
LIT Crosspoint Swap	566,711
Total C-DUKs	**1,331,990**

Table 4.2: Number of DUKs extracted from the region spanned from LAB (19,1) to LAB (40,28).

graph of this region and processed it through Timing Extraction. Table 4.2 shows the resulting number of DUKs extracted from this region for each type of DUK. To compute the delay of all 1,356,182 DUKs, we measure 2,736,556 paths. 3 paths $\times$ 24,192 M-DUKs = 72,576 paths for the M-DUKs, and 2 paths $\times$ 1,331,990 C-DUKs = 2,663,980 paths for the C-DUKs. Processing these paths through path packing produces 232,250 bitstreams.

The complete system is implemented in Java, Python, and Verilog, and automates the full CAD-flow described in Section 4.4. The data is stored in a MySQL database.

4.6 Results

In this section we present the DUK-delay distributions for one of the 18 measured FPGAs. Chapter 5 compares delays across the 18 FPGAs. To understand how we differentiate between distributions, we borrow a page from biological taxonomy. Up to this point we have identified two genera, the M-DUK and C-DUK. Within each genus, we presented different species, two for M-DUKs and eight for C-DUKs. Here, we introduce the concept of a subspecies as it pertains to DUKs. A DUK subspecies uses the resources prescribed by its species, but has minor morphological differences when compared to other subspecies.

For example, when a DUK species uses a LUT, one subspecies will use LUT input A, while another subspecies will use LUT input B. In his seminal work, Darwin referred to these differences as variations within a species [23]. To prevent overloading the word variation, we call these *Subspecies Differences*, or SDs. For each DUK species, we introduce its SDs, and where visually meaningful, differentiate delay distributions based on their SDs. However, for some SDs, greater

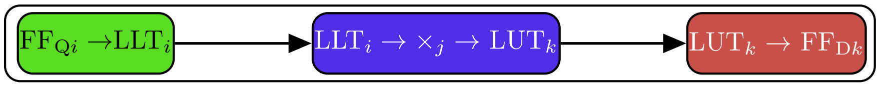

$FF_{Qi} \rightarrow LLT_i$
$LLT_i \rightarrow \times_j \rightarrow LUT_k$
$LUT_k \rightarrow FF_{Dk}$

Figure 4.20: Intra-LAB M-DUK structure annotated with three SDs identified by indices i, j, and k

analysis is required to understand how they contribute to the delay of a DUK. We defer this analysis to Chapter 5.

Overall, the goal of this section is first to demonstrate that we are measuring variation by showing that groups of physical resources that are logically equivalent do not necessarily have the same delay. However, more importantly, this section begins to demonstrate that the variation in DUK delays is composed of a combination of correlated variation, where the delay is a function of some parameter—for example, showing that using LUT input D tends to be faster than LUT input C—and random variation, where for one type of DUK subspecies, we still see a large delay distribution. Even by simply looking at delay distributions, we begin to confirm the composite nature of the delay variation. In Chapter 5 we do a more rigorous analysis that separates the random component from the correlated, and further classifies the correlated variation based on different parameters. Although we mostly focus on the correlated aspects when describing the distributions below, the reader should be mindful that the distribution comes from a combination of correlated and random variation sources. Moreover, in many cases, although the mean distribution delays indicate some correlated advantage of one resource type over another, the random variation often leads to an inversion of this advantage.

4.6.1 Intra-LAB M-DUKs

The Intra-LAB M-DUK represents a short path from a register in one LAB, through a LUT, ending at a register in the same LAB (Figures 4.20 and 4.21). This DUK has three SDs, the index of the source register, the LUT input, and the index of the sink register. Since there is a one-to-one mapping between register and LLTs, choosing the index of the source register determines which LLT is used. For a similar reason, the sink register index determines the LUT's index. Furthermore, the Timing Extraction decomposition determined that only LUT inputs C and D are necessary for this DUK. To generate the equivalent paths that use LUT Inputs A and B, we combine this M-DUK with an LLT Crosspoint Swap C-DUK (Section 4.6.9).

Figure 4.22 shows this DUK's delay distributions. We differentiate the two histograms based on the LUT input SD, since by design, we expect different LUT inputs to have different nominal delays [6] (Section 4.8). Understanding how subspecies, based on the source and sink register

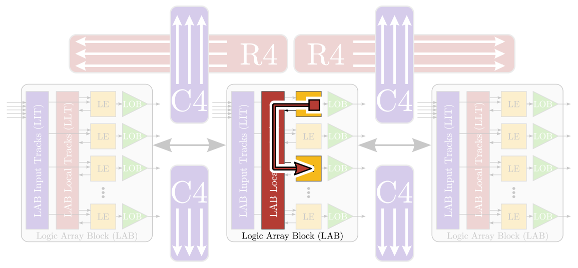


Figure 4.21: Path highlighting physical resources used by the Intra-LAB M-DUK

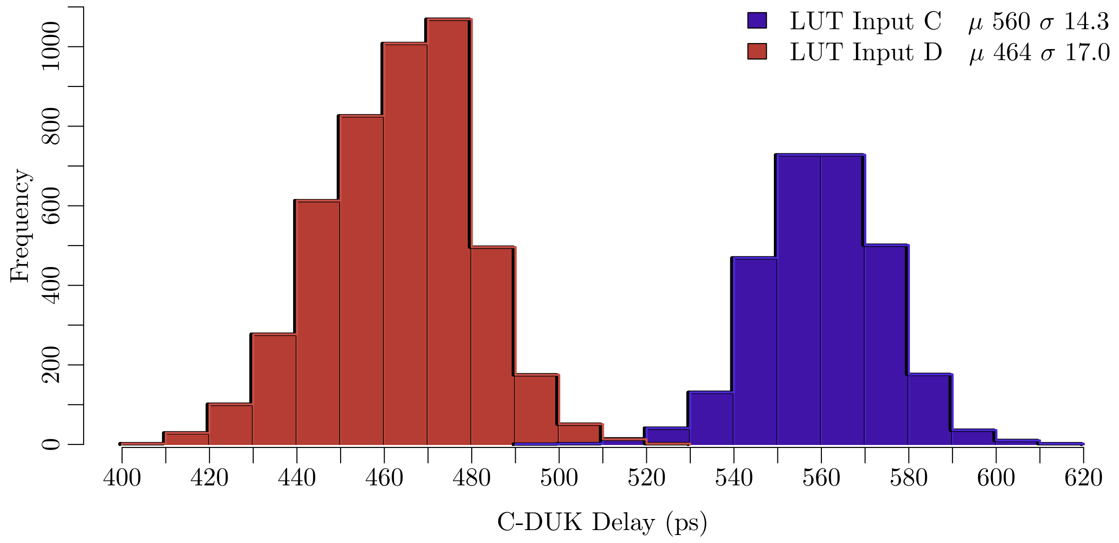


Figure 4.22: Delay distributions for the Intra-LAB M-DUK. Differentiating LUT input used by the DUK

index, differ, requires greater analysis, and is presented in Section 5.4. Therefore, in this figure each distribution is formed by the delay of all subspecies that have the same LUT input, regardless of source and sink used.

4.6.2 LAB-to-LAB M-DUKs

Covering a short path from a register in one lab to a register in a horizontally adjacent lab, this DUK has four SDs (Figures 4.23 and 4.24). We can choose the LOB, which, through a simple mapping function, uniquely determines the source register. Within the existing physical resource connectivity, we are free to select the LIT, the LUT Input, and the sink register. As with the previous DUK, for the LUT input, Timing Extraction only generates LAB-to-LAB M-DUKs that use either LUT input C or D. To use inputs A and B, we combine this M-DUK with an LIT Crosspoint Swap C-DUK (Section 4.6.9).

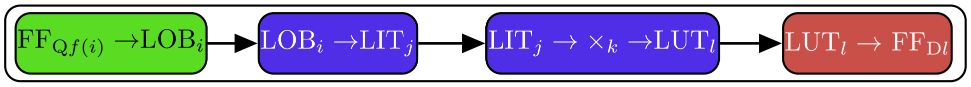

$FF_{Qf(i)} \rightarrow LOB_i$
$LOB_i \rightarrow LIT_j$
$LIT_j \rightarrow \times_k \rightarrow LUT_l$
$LUT_l \rightarrow FF_{Dl}$

Figure 4.23: LAB-to-LAB M-DUK structure annotated with four SDs identified by indices i, j, k, and l

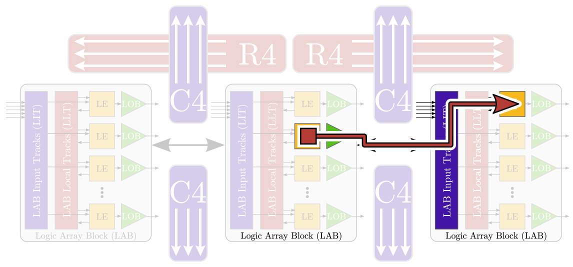


Figure 4.24: Path highlighting physical resources used by the LAB-to-LAB M-DUK

Figure 4.25 shows this DUK's delay distributions. Again, as with the previous M-DUK, we differentiate LUT inputs C and D. However, we also differentiate whether the sink register is in a LAB to the right of the source register (R) or to the left (L). This is an indirect way of representing the LIT index, as half the LITs receive signals from the left LAB, and half from the right LAB. From this graph, we can see that, on average, sending a signal to a LAB on the left is about 20 ps slower then sending one to a LAB on the right. In Section 5.4 we analyze the delay contribution of this and other SDs more carefully.

4.6.3 Intra-LAB C-DUKs

Being analogous to the Intra-LAB M-DUK, this C-DUK has the same three SDs (Figures 4.26 and 4.27). However, when we look at the delay distributions (Figures 4.28), we see that these DUKs are about 200 ps faster. This is expected as we are taking a difference, instead of just having a path between two registers.

4.6.4 LAB-to-LAB C-DUKs

As with the similar M-DUK, this C-DUK has four SDs (Figures 4.29 and 4.30b). In Figure 4.31 we differentiate LUT input C and D, as well as the direction of the C-DUK's Desired Tail. The results are consistent in that input C is slower than input D, and a path heading left is, on average, slower than a path going right. It is worth a reminder, however, that, due in part to random variation, there exist inversions where a path using input D is slower than input C, or a left path is faster than a right one.

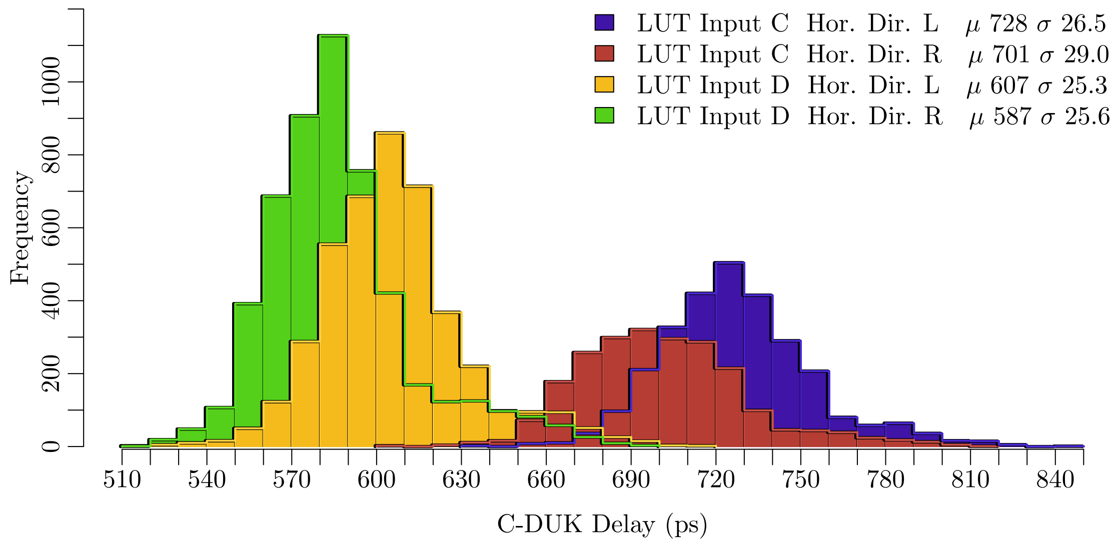


Figure 4.25: Delay distributions for the LAB-to-LAB M-DUK. Differentiating LUT input used and path direction

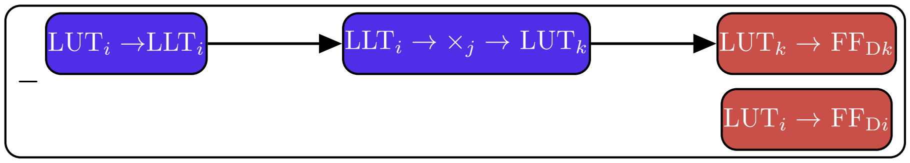
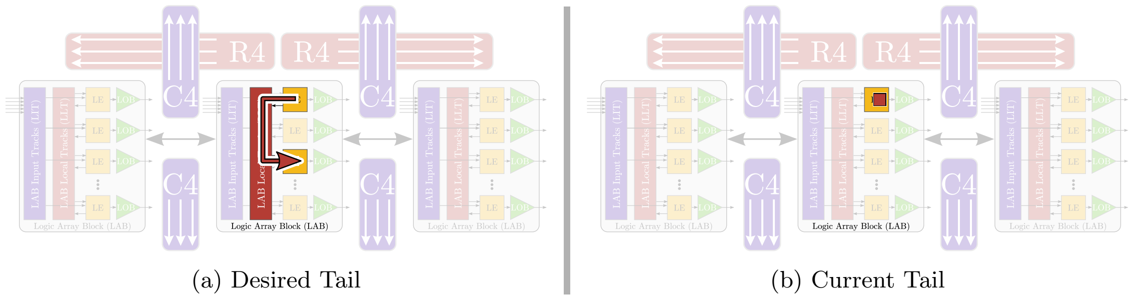

$\text{LUT}_i \rightarrow \text{LLT}_i$
$\text{LLT}_i \rightarrow \times_j \rightarrow \text{LUT}_k$
$\text{LUT}_k \rightarrow \text{FF}_{\text{D}k}$
$\text{LUT}_i \rightarrow \text{FF}_{\text{D}i}$

Figure 4.26: Intra-LAB C-DUK Structure annotated with three SDs identified by indices i, j, and k



(a) Desired Tail
(b) Current Tail

Figure 4.27: Path highlighting physical resources used by both Tails of the Intra-LAB C-DUK

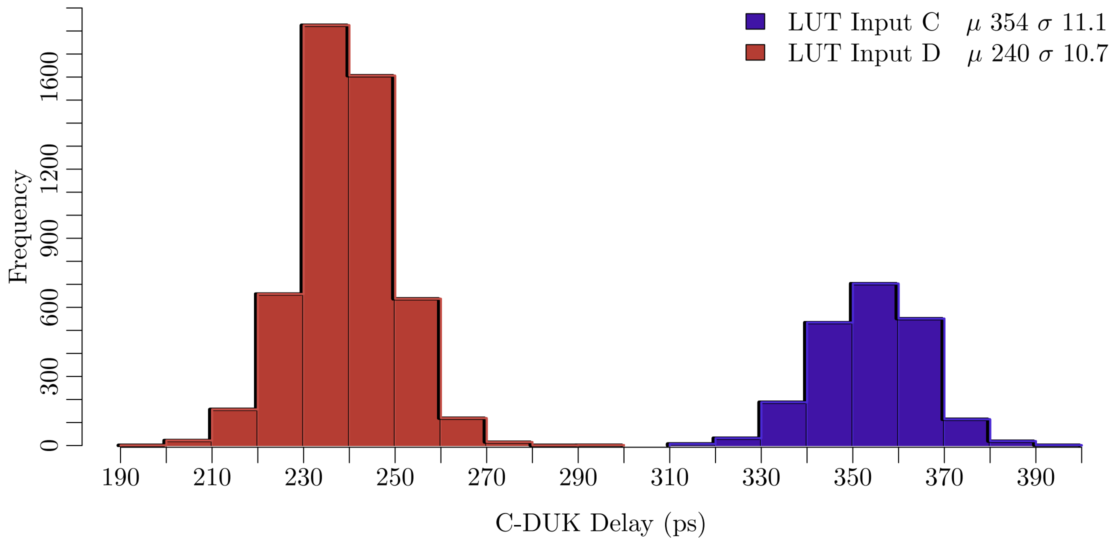
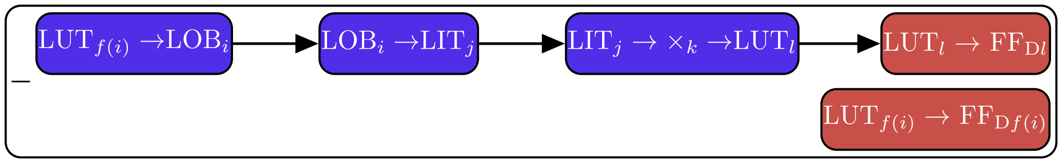


Figure 4.28: Delay distributions for the Intra-LAB C-DUK. Differentiating LUT input used by the DUK



$\text{LUT}_{f(i)} \rightarrow \text{LOB}_i$
$\text{LOB}_i \rightarrow \text{LIT}_j$
$\text{LIT}_j \rightarrow \times_k \rightarrow \text{LUT}_l$
$\text{LUT}_l \rightarrow \text{FF}_{\text{D}l}$
$\text{LUT}_{f(i)} \rightarrow \text{FF}_{\text{D}f(i)}$

Figure 4.29: LAB-to-LAB C-DUK structure annotated with four SDs identified by indices i, j, k, and l

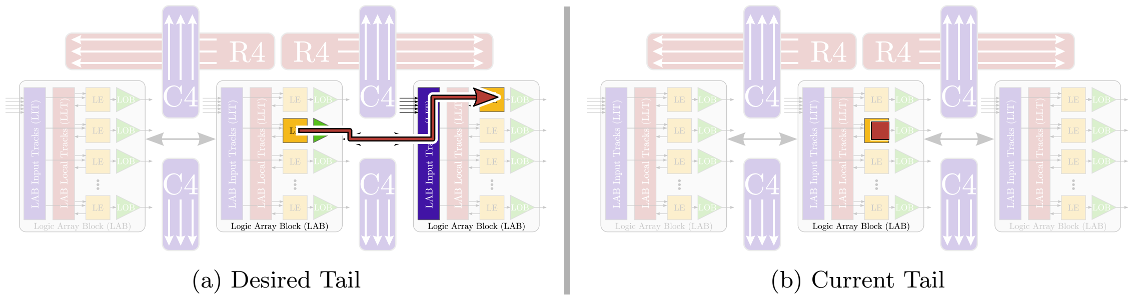


Figure 4.30: Path highlighting physical resources used by both Tails of the LAB-to-LAB C-DUK

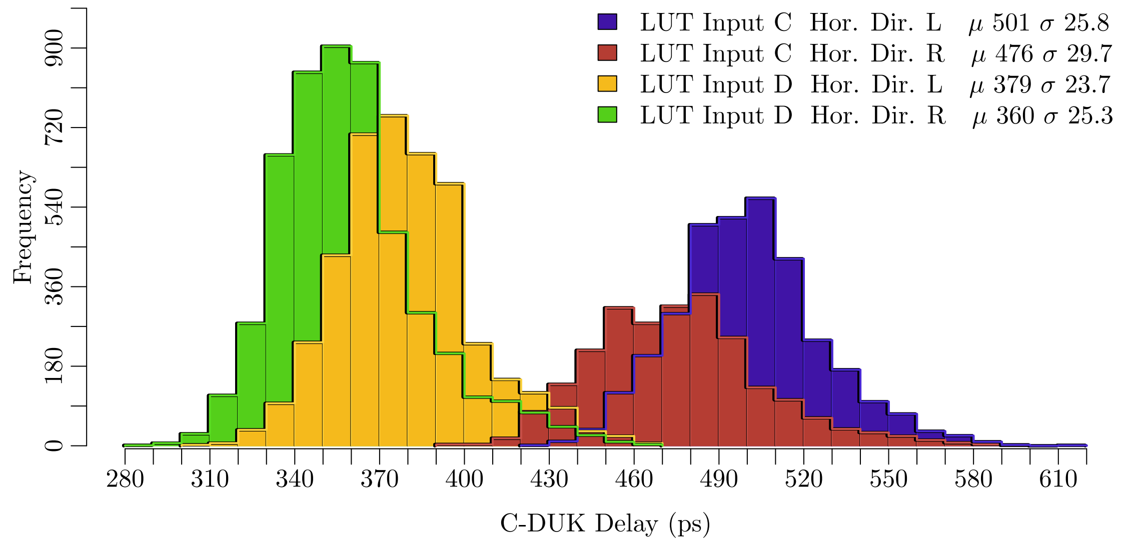


Figure 4.31: Delay distributions for the LAB-to-LAB C-DUK. Differentiating LUT input used and path direction

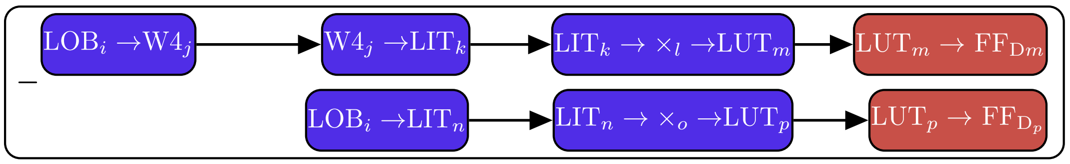

$\text{LOB}_i \to \text{W4}_j$
$\text{W4}_j \to \text{LIT}_k$
$\text{LIT}_k \to \times_l \to \text{LUT}_m$
$\text{LUT}_m \to \text{FF}_{\text{D}m}$
–
$\text{LOB}_i \to \text{LIT}_n$
$\text{LIT}_n \to \times_o \to \text{LUT}_p$
$\text{LUT}_p \to \text{FF}_{\text{D}_p}$

Figure 4.32: LAB-to-General Interconnect C-DUK structure annotated with eight SDs identified by indices from i to p

4.6.5 LAB-to-General Interconnect C-DUKs

The Desired Tail of this DUK begins at one LAB, goes through a horizontal or vertical routing wire, and ends at a register in another LAB. The Current Tail is a LAB-to-LAB path, beginning at a LAB and ending at a register in a horizontally adjacent LAB. Figures 4.32 and 4.33 show the structure with the 8 SDs of this DUK.

For this DUK's distributions, we differentiate first based on routing resource type, vertical or C4, and horizontal or R4. The second SD indicates the direction of the Current Tail, a LAB-to-LAB path going either from a LAB to another LAB to its right (R) or to another LAB to its left (L). Finally, both paths go through a LUT and into a register, thus, we differentiate on the LUT input used for each of the two LUTs.

For Figure 4.34 we fixed both LUT inputs to C to see how wire type and path direction affect the delay. Consistent with Figures 4.25 and 4.31, we see that a LAB-to-LAB path going left is slower than one going right. Unlike the past two cases, however, since in this DUK this is the

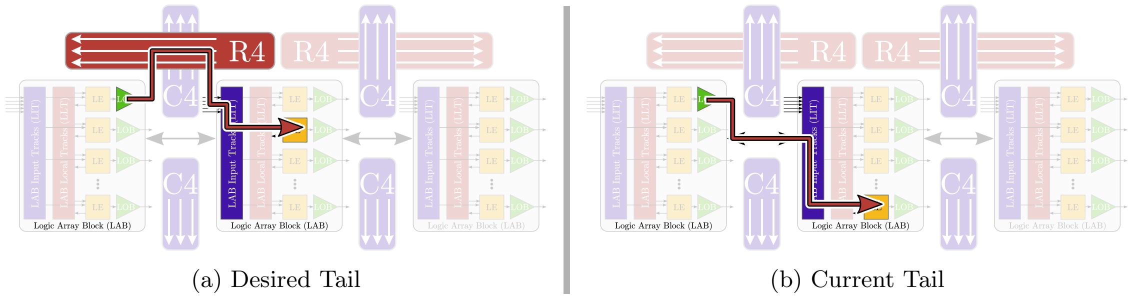


Figure 4.33: Path highlighting physical resources used by both Tails of the LAB-to-General Interconnect C-DUK

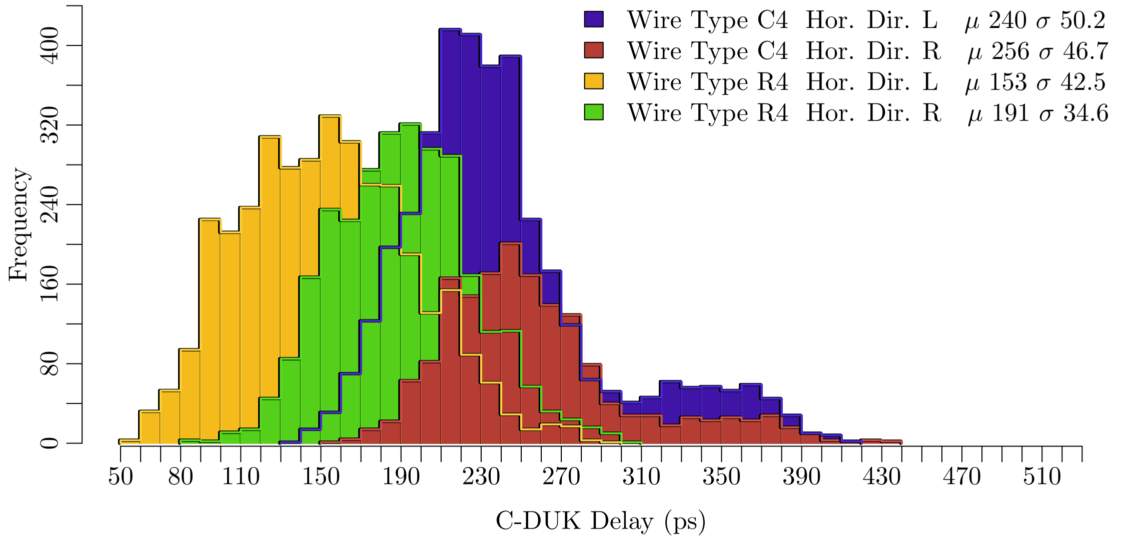


Figure 4.34: Delay distributions for the LAB-to-General Interconnect C-DUK. Both LUT inputs fixed to C. Differentiating routing resource direction of the Desired Tail and LAB-to-LAB path direction of the Current Tail

Current Tail, or path subtracted, it actually leads to DUKs with LAB-to-LAB paths going left having a smaller delay. The second thing to note is that on average, vertical routing wires are slower than horizontal routing wires (Section 5.4).

In contrast, for Figure 4.35, we fix the routing resource to horizontal wires and the LAB-to-LAB path to those going right, differentiating instead on the LUT inputs. As before, the Timing Extraction decomposition results in these DUKs only using inputs C and D. From all previous results we have seen that the mean of input C is, on average, 114 ps slower than the mean of input D. The delay distributions for this DUK are consistent with this. However, in this case, since we have pairs of LUT inputs, we expect the mean delay between the two extreme distributions—a C path minus a D path, versus a D path minus a C path—to be twice that. In the figure we see

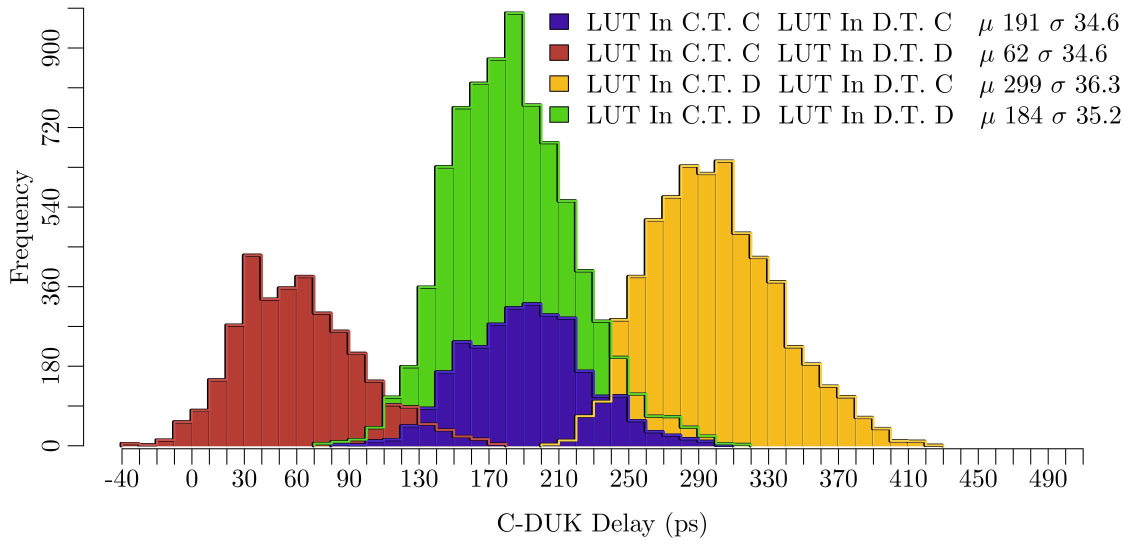


Figure 4.35: Delay distributions for the LAB-to-General Interconnect C-DUK. Fixed the Desired Tail's routing resource to a horizontal R4 and the Current Tail's direction right. Differentiating both LUT inputs used

this as the difference of the means between the red and yellow distributions, which is 237. The fact that it is not exactly 2×114 ps difference is another indication of the existence of random variation.

At this point, we begin to see certain systematic effects emerging. Differences between LUT inputs, path direction, and routing resource types. It is also clear that inversions of these systematic differences are ample and expected.

4.6.6 Embedded Column Neighbor C-DUKs

This C-DUK is a special case of the LAB-to-General Interconnect C-DUK (Section 4.6.5). All Embedded Column Neighbor C-DUKs have a Desired Tail, and consequently also a Current Tail, that begin in a LAB adjacent to an embedded block. As Figure 4.2 shows, the region we measured has embedded memories at column 25. At columns 18 and 34, it has embedded multipliers. Since we do not model embedded blocks in the physical resource graph, the LABs in columns 19, 26, and 35 do not have a LAB horizontally adjacent to their left. This is also true for LABs in columns 24 and 33, however, in this case, there is no LAB horizontally adjacent to their right.

For LAB-to-General Interconnect C-DUKs, the Current Tail is a LAB-to-LAB path (Figure 4.33b), always going from one LAB to a horizontally adjacent LAB. However, for the LABs in columns adjacent to embedded blocks, it is not possible to have a LAB-to-LAB Current Tail that

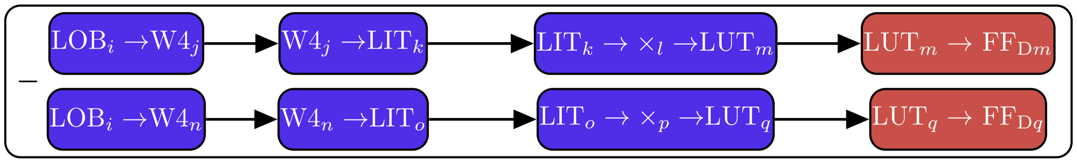
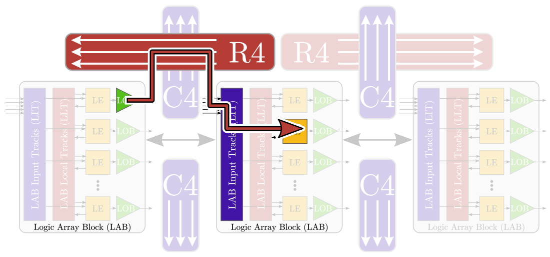
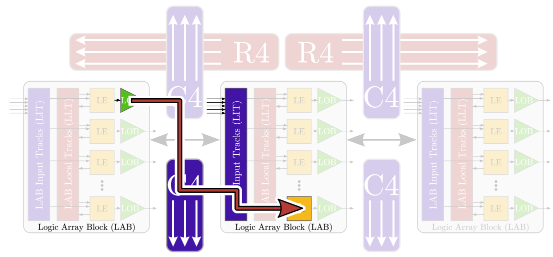

$\text{LOB}_i \rightarrow \text{W4}_j$
$\text{W4}_j \rightarrow \text{LIT}_k$
$\text{LIT}_k \rightarrow \times_l \rightarrow \text{LUT}_m$
$\text{LUT}_m \rightarrow \text{FF}_{\text{D}m}$
$-$
$\text{LOB}_i \rightarrow \text{W4}_n$
$\text{W4}_n \rightarrow \text{LIT}_o$
$\text{LIT}_o \rightarrow \times_p \rightarrow \text{LUT}_q$
$\text{LUT}_q \rightarrow \text{FF}_{\text{D}q}$

Figure 4.36: Embedded Column Neighbor C-DUK structure annotated with nine SDs identified by indices from i to q



(a) Desired Tail



(b) Current Tail

Figure 4.37: Path highlighting physical resources used by both Tails of the Embedded Column Neighbor C-DUK

goes in the direction of the embedded block. Therefore, as Figures 4.36 and 4.37 show, the Current Tail for these DUKs begins at a LAB, goes through a routing resource and ends at a LAB. Note that the Desired Tail of both this C-DUK and the LAB-to-General Interconnect C-DUK have the same structure.

This is a complex DUK, having 9 SDs. Therefore, it is difficult to fully understand the systematic contribution of a subspecies with a particular combination of the 9 SDs by simply looking at delay distributions. We have seen how fixing an SD to a particular value allows us to separate out its effect on the DUK delay. In Figure 4.38 we similarly fix the SD that determining the type of the routing resources for both the Desired Tail and the Current Tail to a vertical C4 wire, and let the two LUT inputs vary. At first glance, the distributions look similar to those in Figure 4.35. However, in this case when we use different LUT inputs, the resulting delay distribution appears to be bimodal. The reason for this comes from the fact that although in this graph we fix the routing resource types to C4 wires, the direction of the wires, either up or down, matter.

In Figure 4.39 we decompose the red bimodal distribution from Figure 4.38 based on the wire direction. At this point, all but one of the distributions is not bimodal, the distribution where both wires go up still has a bimodal component. The difference in this bimodal distribution correlates directly to the Y coordinate of the DUK. This demonstrates systematic differences between DUKs, Section 5.4 automates the discovery process of these types of variations.

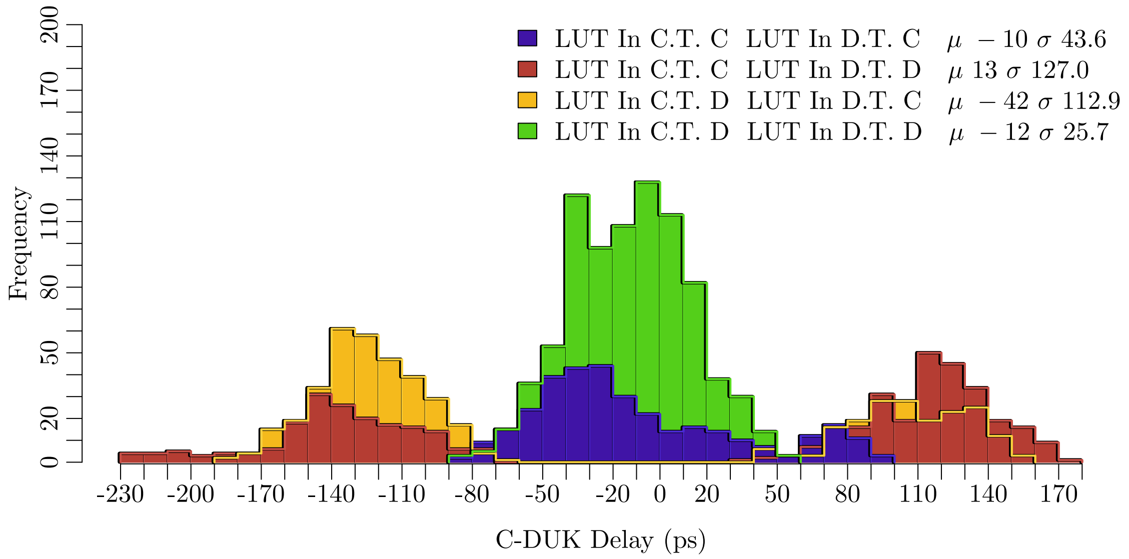


Figure 4.38: Delay distributions for the Embedded Column Neighbor C-DUK. Fixed routing resource type from C4 to C4. Differentiating both LUT inputs used

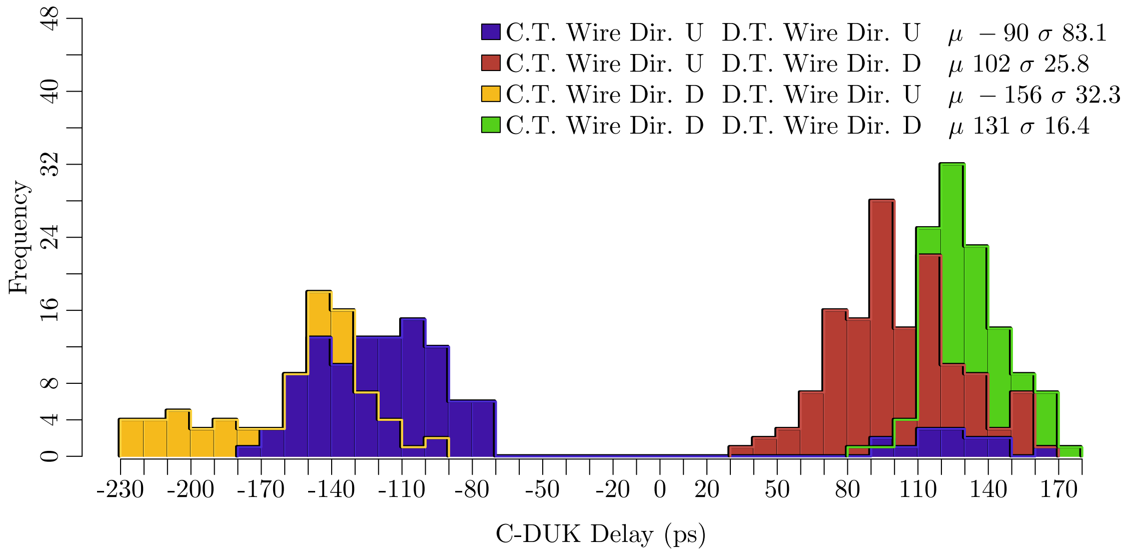


Figure 4.39: Delay distributions for the Embedded Column Neighbor C-DUK. Fixed routing resource type from C4 to C4 and LUT inputs to D for the Desired Tail and C for the Current Tail. Differentiating direction routing resource wires

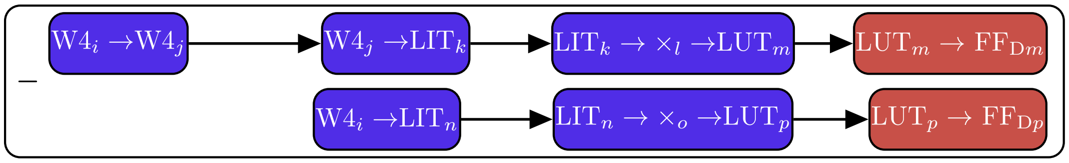


Figure 4.40: General Interconnect C-DUK structure annotated with eight SDs identified by indices from i to p

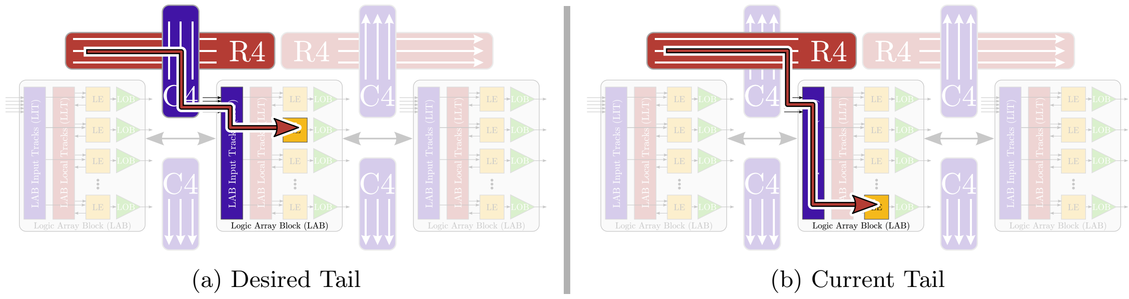


Figure 4.41: Path highlighting physical resources used by both Tails of the General Interconnect C-DUK

4.6.7 General Interconnect C-DUKs

This C-DUK will give us the best indication of the delay of a general routing resources. Its structure is shown in Figures 4.40 and 4.41, highlighting its eight SDs. When we subtract the Current Tail from the Desired Tail, the structure of the DUK is such that the delay that remains can be characterized as the delay of the W4$_j$ routing resources plus the differences between two paths of similar shape. It is this fact that gives the DUK its name.

As with all C-DUKs, both the Current Tail and the Desired Tail end by going through a LUT, thus we differentiate the distributions in Figure 4.42 based on the LUT input. This DUK also uses two routing resources, W4$_i$ and W4$_j$. For this figure, we fix both to vertical, C4, wires. Going through a similar analysis to that of Figure 4.35 in the previous section, confirms that LUT input C is slower than LUT input D.

More interesting is the result presented in Figure 4.43 where we fix the LUT inputs both to C and allow the routing resources W4$_i$ and W4$_j$ to vary. We mentioned that W4$_j$ is the dominant delay of this DUK, and Figure 4.43 confirms this. The primary difference for the means of the distributions is whether W4$_j$ is a horizontal, R4, wire, or a vertical, C4, wire. As with Figure 4.34 we again see that horizontal wires are faster than vertical, a fact confirmed in the analysis of Section 5.4. Another similarity to the distributions in Figure 4.34 is the large tails of both distributions with W4$_j$=C4. This long tail corresponds to a systematic slowdown of the C4 routing resources in columns 20 and 29 of the Cyclone III FPGA. Section 5.4 better quantifies

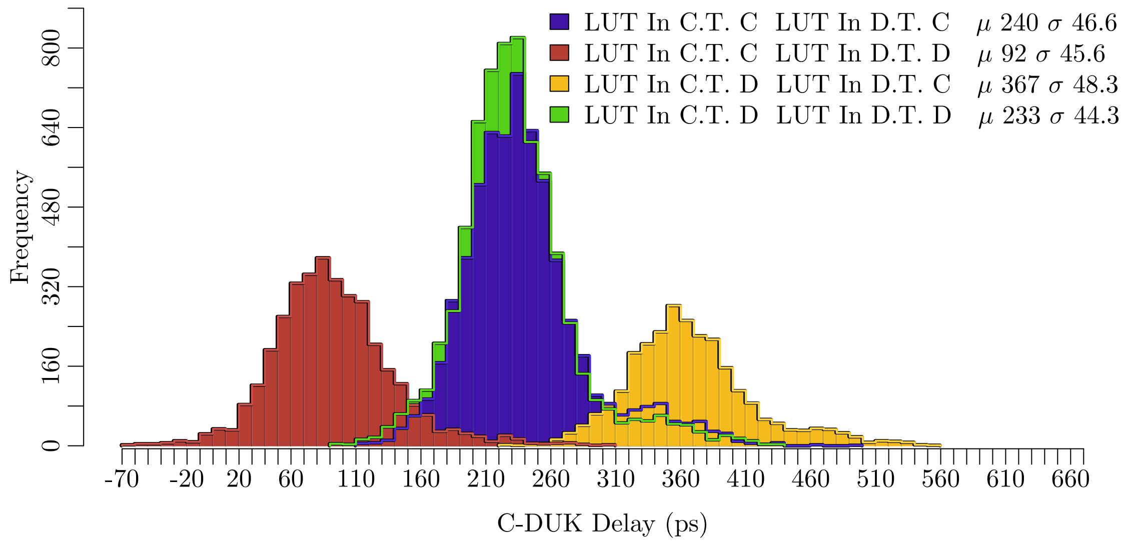


Figure 4.42: Delay distributions for the General Interconnect C-DUK. Fixed both routing resource type to C4. Differentiating both LUT inputs used

this fact.

4.6.8 General Interconnect Sink Select C-DUKs

Routing resource wires connect to multiple LABs along their length. This C-DUK allows us to change which LAB the routing resource connects to (Figure 4.45). As shown in Figure 4.44, this C-DUK has seven SDs. For our distribution plots we differentiate based on the LUT inputs of both the Desired and Current Tails, and fix the wire $W4_i$ to vertical wires. The distributions in Figure 4.46 are consistent with all previous distributions differentiated in this way, except for the fact that the two distributions with different LUT inputs are bimodal. This bimodal distribution is explained by the distance between the LAB of the Current Tail, and the LAB of the Desired Tail. The sign of this difference is responsible for the bimodal nature, and is purely the result of an arbitrary but consistent decision made by the decomposition algorithm of which sibling in a given sibling set is the marked sibling.

4.6.9 LLT and LIT Crosspoint Swap C-DUKs

In all previous DUKs, either LUT input C or LUT input D is used to go into a LUT. More than that, for most other DUKs the LUT used is always the very first LUT in the LAB, the other 15 LUTs are not used. It is these two C-DUK that allow us to change both which LUT and which LUT input is used in a path. The reason why there are two types of C-DUKs is that we can

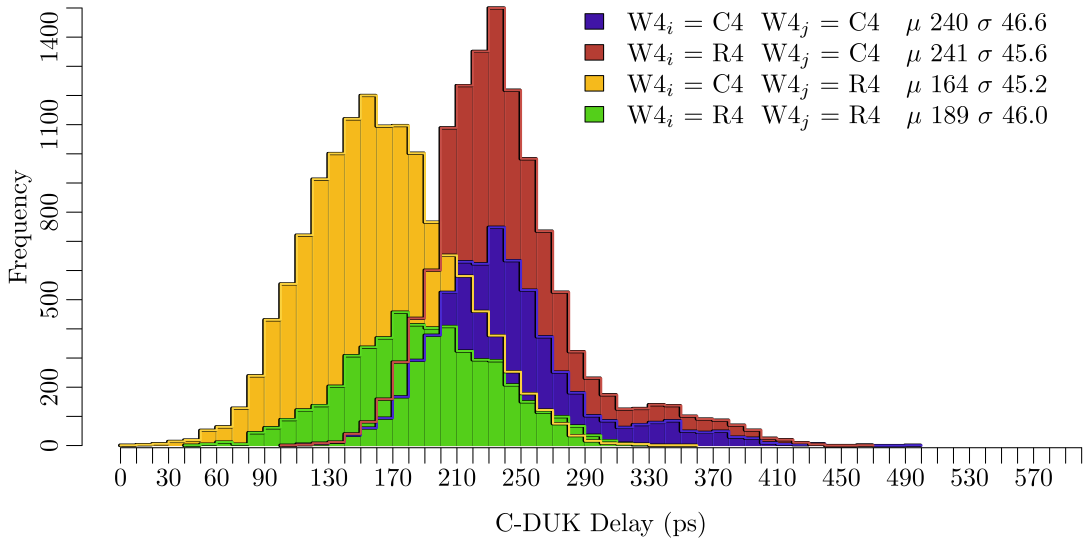


Figure 4.43: Delay distributions for the General Interconnect C-DUK. Both LUT inputs fixed to C. Differentiating routing resource type combinations

$$\begin{tikzcd} W4_i \rightarrow LIT_j \arrow[r] & LIT_j \rightarrow \times_k \rightarrow LUT_l \arrow[r] & LUT_l \rightarrow FF_{Dl} \\ W4_i \rightarrow LIT_m \arrow[r] & LIT_m \rightarrow \times_n \rightarrow LUT_o \arrow[r] & LUT_o \rightarrow FF_{Do} \end{tikzcd}$$

Figure 4.44: General Interconnect Sink Select C-DUK structure annotated with seven SDs identified by indices from i to o

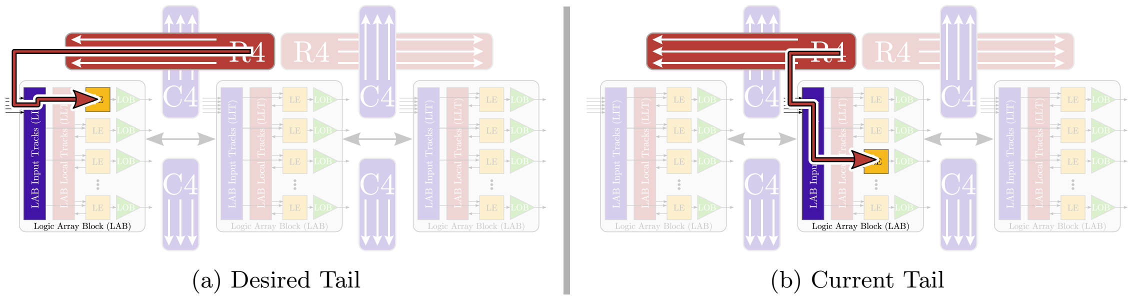


Figure 4.45: Path highlighting physical resources used by both Tails of the Sink Select C-DUK

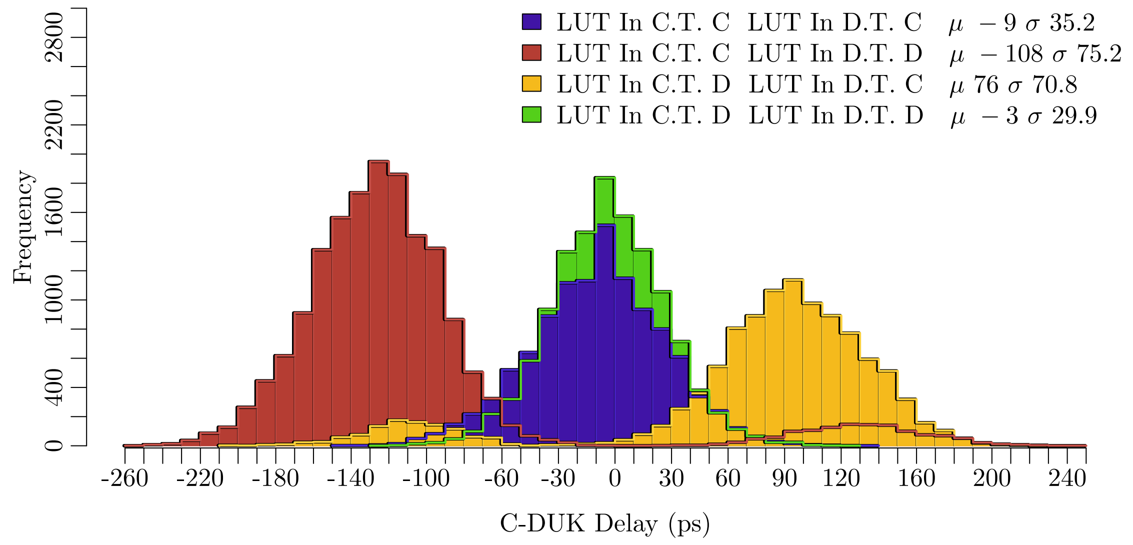
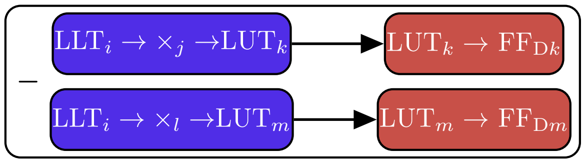


Figure 4.46: Delay distributions for the General Interconnect Sink Select C-DUK. Fixed routing resource type to C4. Differentiating both LUT inputs used

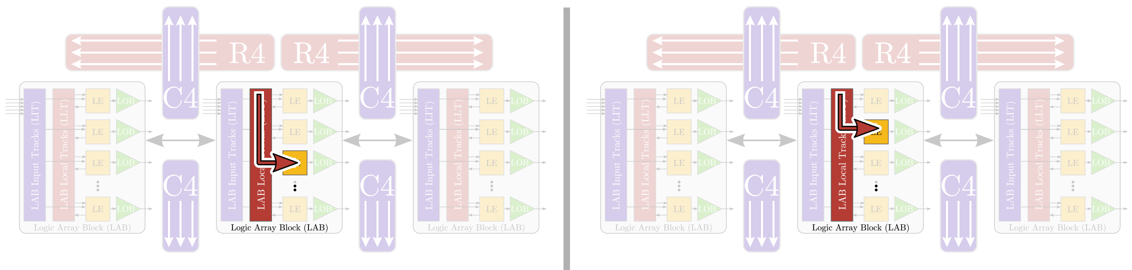
go into a LUT either from within the LAB using an LLT, or from the general interconnect or an adjacent LAB using an LIT. Thus, how a signal enters a LUT determines whether we require an LLT Crosspoint Swap C-DUK or an LIT Crosspoint Swap C-DUK to change the LUT and LUT input used. The structure for the LLT based C-DUK is shown in Figures 4.47 and 4.48. Figures 4.49 and 4.50 show the LIT based C-DUK structure. Each has five SDs determining either the LLT or LIT used, the LUT input of both the Current Tail and Desired Tail, and similarly for the LUT and register used for each part of the C-DUK.

Figures 4.51 and 4.52 show the delay distributions for both C-DUKs. In them we differentiate the LUT input used for the Current Tail and Desired Tail. Although for the LLT based C-DUKs the Current Tail only uses LUT inputs C and D, and for the LIT based DUK it is LUT inputs A and B, both provide the necessary combinations to change the selection when we consider that C-DUKs are reversible in the following sense. The Current Tail of a C-DUK represents the resources we remove from the end of a path, while the Desired Tail represents the resources we add to the end of a path. The delay of a C-DUK indicates how the path delay changes once we remove the Current Tail and add the Desired Tail. However, we can always use a C-DUK in the opposite way, where the Desired Tail is the part we remove from the end of a path, and the Current Tail the part we add. To get the delay of the C-DUK used in this way, we simply change the sign of the original C-DUK. Thus, if we use the LIT Crosspoint Swap C-DUKs in reverse, we gain the ability to change from paths that use inputs C and D, to paths that use inputs A and B.



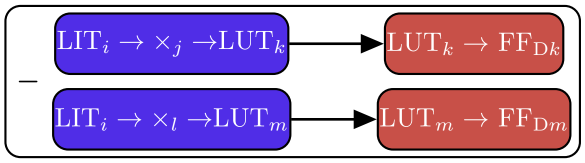
$\mathrm{LLT}_i \rightarrow \times_j \rightarrow \mathrm{LUT}_k$
$\mathrm{LUT}_k \rightarrow \mathrm{FF}_{\mathrm{D}k}$
$-$
$\mathrm{LLT}_i \rightarrow \times_l \rightarrow \mathrm{LUT}_m$
$\mathrm{LUT}_m \rightarrow \mathrm{FF}_{\mathrm{D}m}$

Figure 4.47: LLT Crosspoint Swap C-DUK structure annotated with five SDs identified by indices i, j, k, l, and m



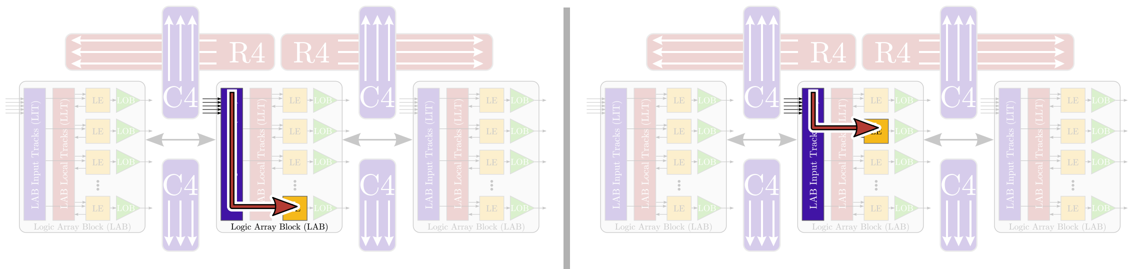
(a) Desired Tail (b) Current Tail

Figure 4.48: Path highlighting physical resources used by both Tails of the LLT Crosspoint Swap C-DUK



$\mathrm{LIT}_i \rightarrow \times_j \rightarrow \mathrm{LUT}_k$
$\mathrm{LUT}_k \rightarrow \mathrm{FF}_{\mathrm{D}k}$
$-$
$\mathrm{LIT}_i \rightarrow \times_l \rightarrow \mathrm{LUT}_m$
$\mathrm{LUT}_m \rightarrow \mathrm{FF}_{\mathrm{D}m}$

Figure 4.49: LIT Crosspoint Swap C-DUK structure annotated with five SDs identified by indices i, j, k, l, and m



(a) Desired Tail (b) Current Tail

Figure 4.50: Path highlighting physical resources used by both Tails of the LIT Crosspoint Swap C-DUK

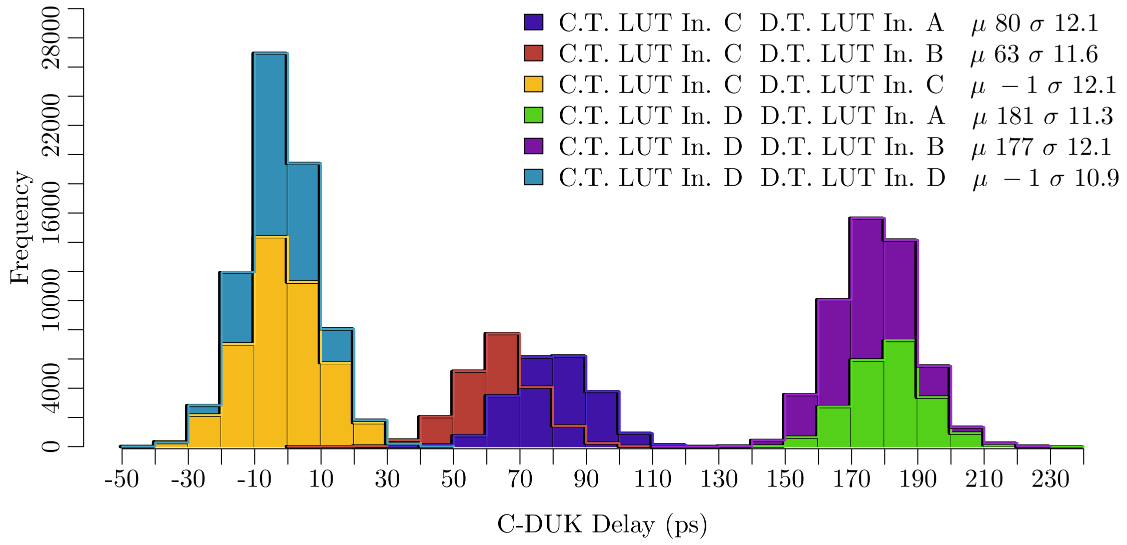


Figure 4.51: Delay distributions for the LLT Crosspoint Swap C-DUK. Differentiating both LUT inputs used

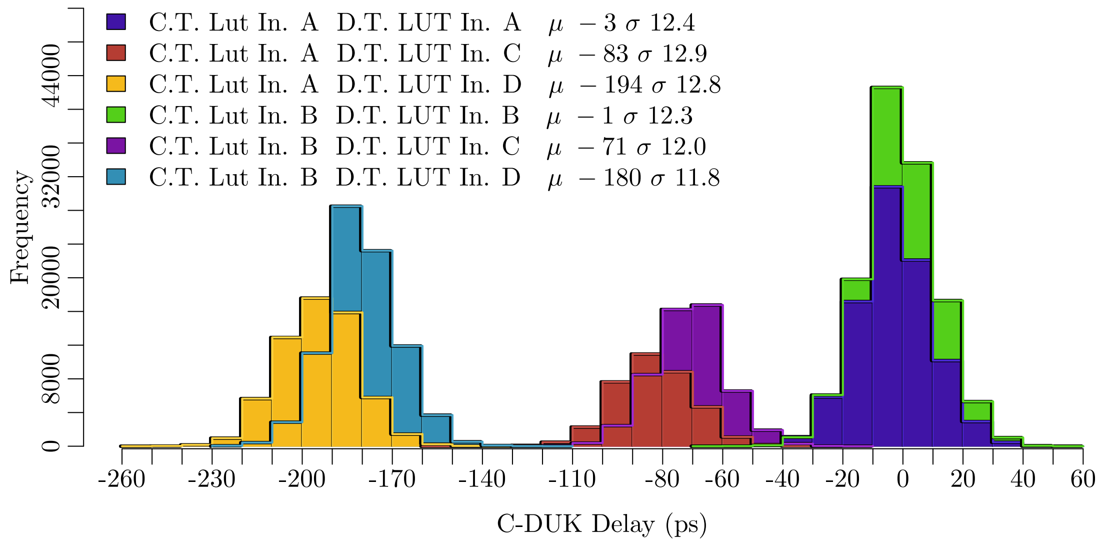


Figure 4.52: Delay distributions for the LIT Crosspoint Swap C-DUK. Differentiating both LUT inputs used

In Section 4.6.1 we mentioned that we expect LUT inputs to have different nominal delays. Up to this point we had only confirmed this for inputs C and D. Now we can see how inputs A and B relate as well. First, notice that when the LUT input of both parts of the C-DUK match, the distributions are practically centered at zero. This gives us a frame of reference from which to compare the other combinations. In Figure 4.51 the mean for the dark blue distribution, where inputs C and A are used, has a mean delay of 80 ps. Similarly in Figure 4.52 the corresponding distribution, the red one in this case, has a mean delay of −83 ps. Since the order in which the two distributions use inputs C and A is mirrored, both graphs consistently indicate that the systematic delay contribution of using input A versus using input C is about 80 ps. Looking at the correct distributions, we can come up with an ordering of LUT input delays where the systematic contribution of one LUT input, relative to another, is as follows. Input A is, on average, 12 ps slower than input B. In turn, input B is, on average, 67 ps slower than input C. Finally, on average, input C is 109 ps slower than input D. Of course, these numbers are not exact differences between LUT inputs, since random variation also contributes to the delay. Nevertheless, there is strong evidence that these values are close to the average delay difference between LUT inputs.

4.7 Falling Vs. Rising Transitions

As Section 3.1 presents, FPGAs have different falling and rising transition delays. In part this is due to the behavior of CMOS logic, where there are differences between the pull-up and pull-down networks. Also, some of the circuit structures use pass-transistors which have a harder time propagating a rising transition as compared to a falling transition. In theory, we can compute the delay difference between rising and falling transitions by examine the circuit and adjusting based on the properties of the different physical resources. Nevertheless, our path-delay measurement circuit (Section 4.4.1) separately tracks the delay of a rising transition and the delay of a falling transition. We track both because random variation greatly reduces the correlation between a falling and a rising transition, making it nearly impossible to extrapolate one from the other. In this section we examine the difference between rising and falling transitions, and confirm that the magnitude of the random variation requires that we record both rising and falling transitions. Except for the results presented in this section, the results we show in this work are for rising transitions only.

In our DUKs, we clearly detect a difference between falling and rising transitions. For example, Figure 4.53 plots the same distributions as Figure 4.25 expect that here we show both rising delay distributions (in darker shades) and falling delay distributions (in lighter shades). From this graph,

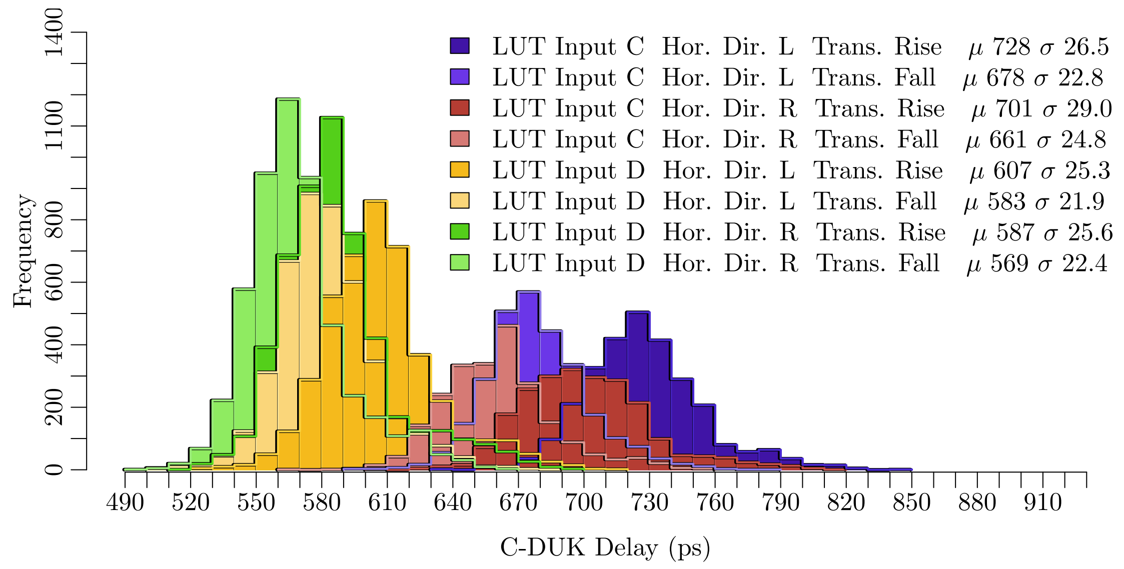
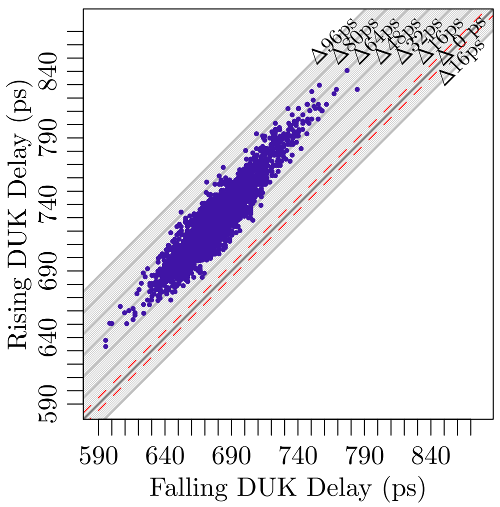


Figure 4.53: Delay distributions for the LAB-to-LAB M-DUK. Differentiating LUT input used and path direction

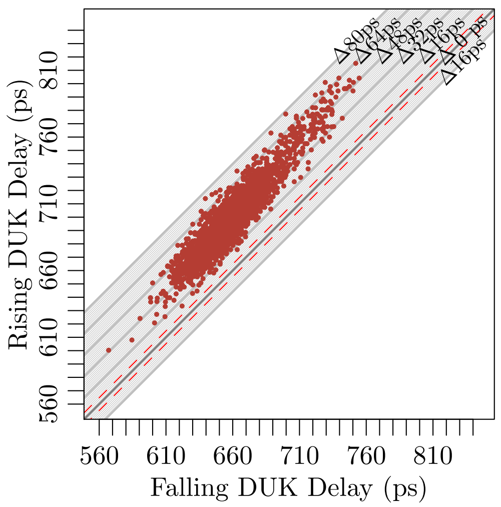
it appears as though the main difference between rising and falling delays is that rising delays are systematically slower than falling delays by an average of 4% to 5%. However, to really understand the difference between falling and rising, for each combination of LUT input and path direction, it is necessary to look at a correlation plot.

If the difference between falling and rising was purely systematic, a correlation plot, where the X-axis shows the falling delays, and the Y-axis shows the rising delays, would show all points on the diagonal that represents rising transitions 5% slower than falling transitions. Figures 4.54 and 4.55 show the corresponding four correlation plots. Although all four plots show a diagonal trend, it is clear that all points are not on the same diagonal. In fact, for some DUKs, we see up to 12% delay difference between rising and falling. Moreover, for input D, in Figure 4.55, we even see inversions where the falling delay of a DUK is slower than its rising delay.

When we examine all our DUK measurements, we observe that, on average, rising transitions are slower than falling transitions. However, due to the presence of random variation, the correlation between rising and falling delays is not sufficiently strong to allow us to only measure one and extrapolate the other. Therefore, for all our measurements, we maintain a separate entry for a DUK's rising and falling delays.

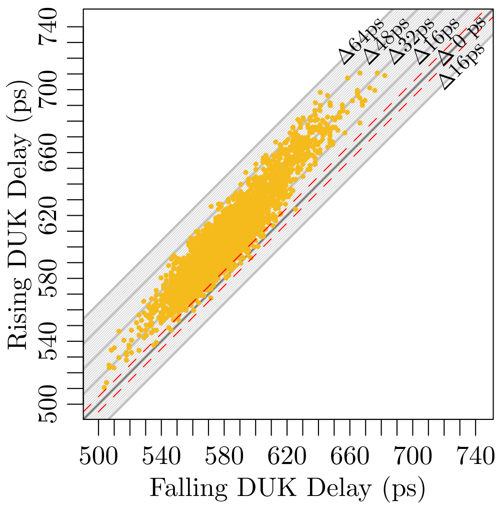


(a) LAB-to-LAB path direction: Left

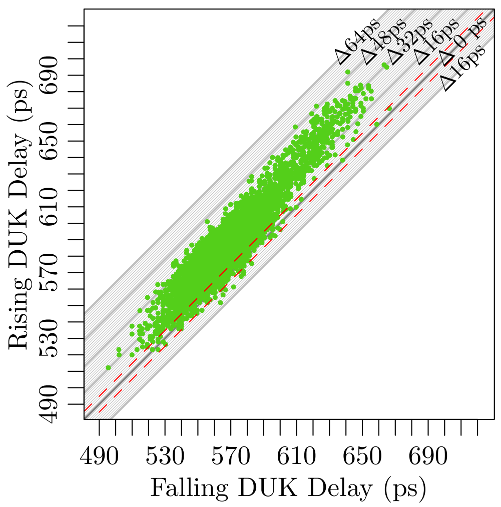


(b) LAB-to-LAB path direction: Right

Figure 4.54: Correlation between falling and rising LAB-to-LAB M-DUK delays. Figure a shows the result when LUT input is fixed at C and the path direction is left, while Figure b shows the results for the same LUT input, C, but path direction is right. Diagonal lines indicate the difference between the two results in terms of $d_\Delta = 1.6$ ps. Thicker lines indicate $10d_\Delta$. Red dashed lines indicate error boundaries as computed in Section 3.5.



(a) LAB-to-LAB path direction: Left



(b) LAB-to-LAB path direction: Right

Figure 4.55: Correlation between falling and rising LAB-to-LAB M-DUK delays. Figure a shows the result when LUT input is fixed at D and the path direction is left, while Figure b shows the results for the same LUT input, D, but path direction is right. Diagonal lines indicate the difference between the two results in terms of $d_\Delta = 1.6$ ps. Thicker lines indicate $10d_\Delta$. Red dashed lines indicate error boundaries as computed in Section 3.5.

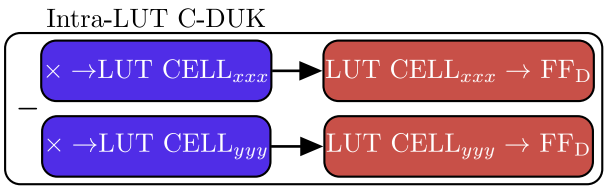

Intra-LUT C-DUK
$\times \rightarrow$LUT CELL$_{xxx}$
LUT CELL$_{xxx} \rightarrow$ FF$_D$
$-$
$\times \rightarrow$LUT CELL$_{yyy}$
LUT CELL$_{yyy} \rightarrow$ FF$_D$

Figure 4.56: C-DUK for internal LUT structures

4.8 Intra-LUT C-DUK

Up to this point we have modeled LUTs in the FPGA as black boxes. However, to demonstrate that Timing Extraction can adapt when given a more detailed resources graph, in this section we consider what happens when we explore the structure of the LUTs. Currently, we represent a LUT as one node in the physical resource graph. Understanding how we use the LUT explains how to modify the physical resource graph to represent the internals of the LUT.

A LUT with k inputs contains 2^k memory cells, usually implemented as SRAM cells. As shown in Figure 4.15, 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 1 out of k inputs is used, and only 2 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 3 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 8 pairs of SRAM cells that can be used to implement a buffer on input A. We can label the 8 pairs as A_{000} through A_{111}. A similar labeling for inputs B, C, and D leads to 32 pairs of SRAM cells, and each one of these 32 pairs is represented as a node in the physical resource graph, replacing the LUT node. To modify the graph, we look at any parent of the original LUT that connected through input A, and replace its connection to nodes A_{000} through A_{111}. A similar replacement occurs for the other three LUT inputs. Finally, any node that was a child of the original LUT node is now a child of all 32 SRAM cell nodes.

Once the graph is modified, LC Nodes and DUKs are formed using the same algorithms. However, in this case, we have a new C-DUK representing the substitution of one pair of SRAM cells for another. We call this the Intra-LUT C-DUK and is represented in Figure 4.56

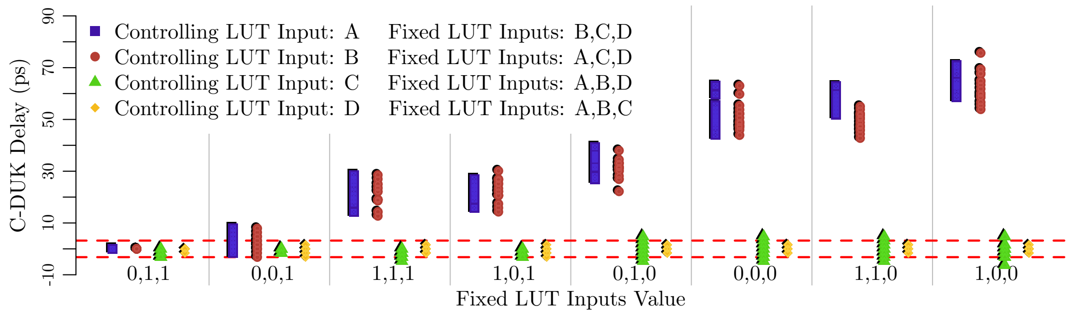


Figure 4.57: Intra-LUT C-DUK delays. Each scatter plot shows the delay of Intra-LUT C-DUKs with the same controlling and fixed LUT inputs. The region between red dashed lines is shown as a reference of the range of the expected error as computed in Section 3.5

4.8.1 Results

For all results presented in this work, except for those in this section, when a LUT input is not used to implement the buffer, it is fixed to the vector 0011 for inputs A through D respectively. We do this through the Fixed LUT Input blocks shown in Figure 4.15. The Intra-LUT C-DUK allow us to take these results and adjust them for other vectors. Figure 4.57 shows the Intra-LUT C-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. While, when C is the controlling input, A, B, and D are fixed at 0, 1, and 1.

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 architecture of the LUT as shown in Figure 4.58. When A or B are controlling, the critical path incurs the full delay of going through the internal LUTs followed by the muxes. In this case, the value of inputs C and D determine the path through the LUT. When inputs C is controlling, the critical path is determined only by input D. Looking at Figure 4.57, when input D is 1, input C's delay is more controlled. Whereas, when D is 0, input C's delay has a slightly wider distribution. Finally, when input D is controlling, it does not depend on any of the other inputs to determine its critical path, thus its delay never changes, regardless of the value of the fixed LUT inputs.

4.9 Effects of V_{DD} on Variation

Lowering the supply voltage, V_{DD}, is a common and important way to save power and energy. In this section we use DUK delays to examine the effect that reducing V_{DD} has on variation. In

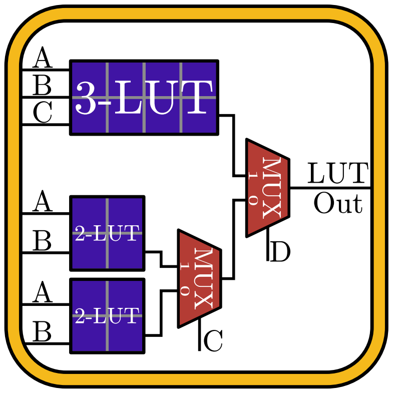


Figure 4.58: Architecture of the Cyclone-III 4-LUT [6]

particular we ask whether scaling V_{DD} has a purely systematic effect on the variation distributions or is there a random component as well.

The primary adverse effect of reducing V_{DD} is a reduction in performance. We see this directly in the equation for the propagation delay of a transistor reproduced here.

$$\tau_{pd} = C_l \cdot \frac{V_{ds}}{I_{ds}} \tag{2.1}$$

Lowering V_{DD} also has an effect on the current through the transistor. In Section 2.2.2 we present the saturation and subthreshold current equations, reproduced below.

$$I_{ds,sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right) \tag{2.5}$$

$$I_{ds,sub} = \frac{W}{L} \mu C_{ox} (n-1) \cdot {v_T}^2 \cdot e^{\dfrac{V_{gs} - V_{th}}{n \cdot v_T}} \left(1 - e^{\dfrac{-V_{ds}}{v_T}} \right) \tag{2.6}$$

Lowering V_{DD} has a direct effect on V_{gs}, and subsequently reduces the magnitude of the difference $(V_{gs} - V_{th})$. As the delay is inversely proportional to the current, the smaller magnitude of this difference also increases the delay through the transistor.

However, lowering V_{DD} also has an effect on the spread of the delay distribution, that is, the difference between the fastest and slowest resource increases as we decrease V_{DD}. We know that τ_{pd} is proportional to $\frac{1}{(V_{gs} - V_{th})}$. Therefore, as Equation 4.2 shows, the difference between the slowest and fastest transistor is proportional to $\frac{1}{V_{gs}^2}$, which increases as V_{DD}, and consequently V_{gs}, decrease.

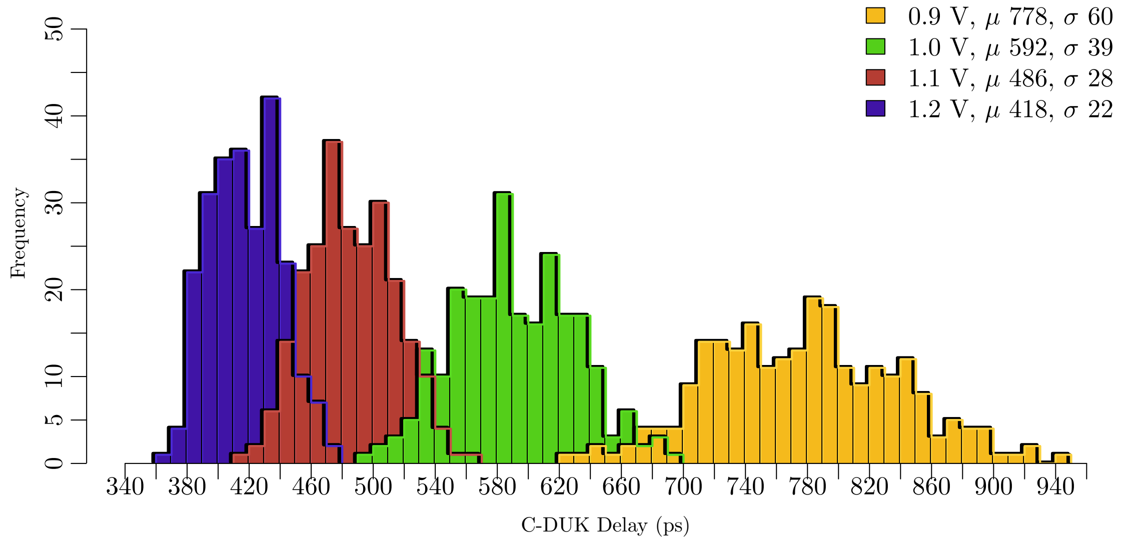


Figure 4.59: Intra-LAB delays when varying V_{dd}. Differentiating V_{dd} value, Cyclone IV

$$\tau_{pd\ max} - \tau_{pd\ min} \propto \frac{1}{(V_{gs} - V_{th\ max})} - \frac{1}{(V_{gs} - V_{th\ min})} \propto \frac{1}{V_{gs}^2} \tag{4.2}$$

By extracting DUK delay at different V_{DD}s, we can empirically observe these effects. To do this, we take advantage of the fact that the DE0-Nano [86] board, containing a Cyclone IV FPGA [12], is easier to modify than the BeMicro Arrow board, and perform the necessary changes to it so that we can control its internal V_{DD} [85]. Fortunately, four our purposes, the only difference between the Cyclone IV and Cyclone III is that the Cyclone IV is fabricated in 60 nm technology, as opposed to 65 nm technology. Aside from this, the architecture of the resources we measure are the same.

Nominally, the board provides a 1.2 V V_{DD}. For our tests, we scale at 100 mV increments. At V_{DD} = 0.8 V, a large percent of our measurements fail, and due to safeguards on the board, at 0.7 V the board fails to power up.

Figure 4.59 shows, for the same set of DUKs, what happens to their delay distribution as we reduce V_{DD}. As expected, we clearly see both effects discussed. Lowering V_{DD} both shifts the mean delay to larger delays, and increases the standard deviation. Thus, although lowering V_{DD} is useful, lowering it too far will negate its effects, since variation will increase, reducing the energy savings, and potentially leading to defects. However, when combined with a component-specific mapping that takes into account the variation in the FPGA, this approach can lead to almost 2× energy savings[63].

4.10 CAD Vs. Measured Delays

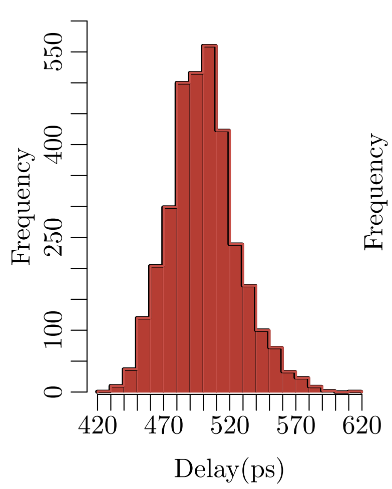
Timing extraction computes DUK delays from real on-chip measurements. However, the CAD tools provided by the FPGA vendors contain a complex timing analysis tool, backed by a detailed delay model of the FPGA, from which we could potentially get the same delay information as we do from our DUKs. To get the DUK delays as computed by the CAD tools, for every path we measure, we also ask the CAD tools for what it thinks is its delay. With that information, we can calculate both DUK delays based on our measurements, and CAD DUK delays based on the delays reported by the tools. In this section we compare the distributions from these two types of DUK delays.

Figure 4.60a shows the DUK delay distribution for LAB-to-LAB C-DUKs, using LUT input C, going left. This is the exact same distribution as the blue distribution in Figure 4.31. Figure 4.60b plots the delay distribution for the same DUKs, however, it is based on the delays computed from the CAD tools instead. The two distributions are strikingly different.

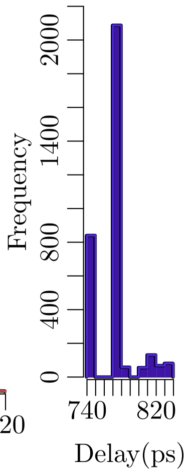
The measured distribution spans 187 ps, while the CAD distribution spans half, 93 ps. Also, on average, the CAD DUKs are 54% slower than the measured DUKs. This last fact is hardly surprising, however, since the CAD tools model more than just process variation, they also take into account environmental and operational variation, as well as aging effects. In contrast, our measurements are done in highly controlled environments. What is surprising is that the shape of the two distributions is completely different. To better compare them, Figure 4.60c shows a correlation plot between the two distributions. From this figure it is clear that there is little correlation between the two distributions. Nevertheless the fact that the CAD distribution has a spread indicates that the CAD tools are aware of some amount of variation between logically equivalent resources.

To understand what the CAD tools use to compute this variation, we need to understand which properties of the DUK the CAD tools depend on to extract their delay. Figure 4.61a shows the correlation between CAD DUK delays and the LUT index for the beginning of this DUK. From this figure, we see that different indices tend to have different CAD delays. Similarly, Figure 4.61b shows the correlation of the CAD DUK delay to the X coordinate of that LUT. Here we can separate the DUKs into three groupings. Those DUKs that have a LUT with X coordinate 21, those that use coordinate 30, and those in neither 21 nor 30.

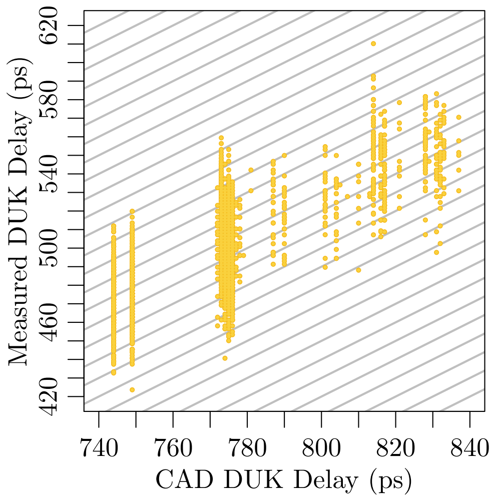
Separating DUKs by both LUT index, and the three X coordinate groups almost perfectly explains the sources of variation known to the CAD tools. Figure 4.62 shows the delay distributions and correlation when we isolate based on a LUT index of 7. Our measurements indicate that these DUKs span a range of 141 ps in a Gaussian-like distribution from 469 ps to 610 ps (Figure 4.62a).



(a) Measure DUK Distribution

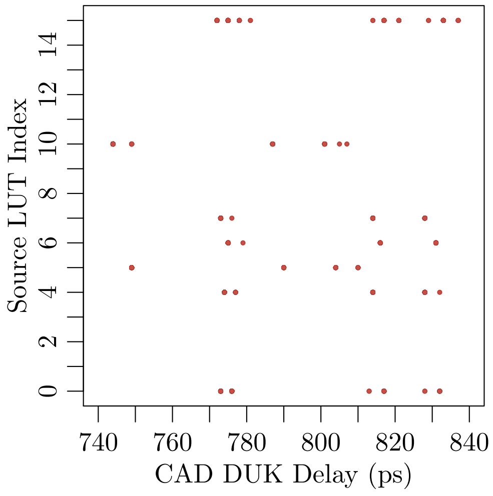


(b) CAD DUK Distribution

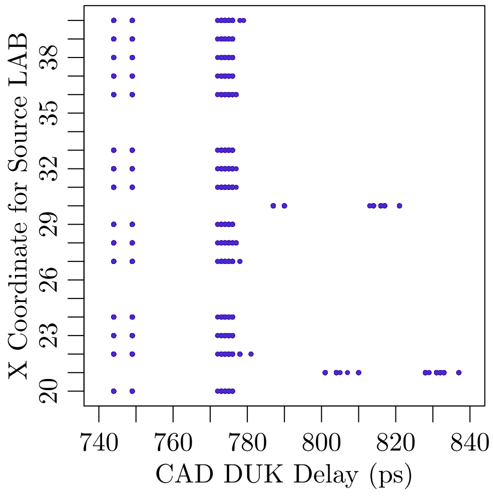


(c) Correlation

Figure 4.60: Correlation between measured DUK delays and CAD DUK delays for LAB-to-LAB C-DUKs on LUT input C going left. Diagonal lines in Figure c, spaced at 10 ps intervals, show the angle where perfect correlation would occur.

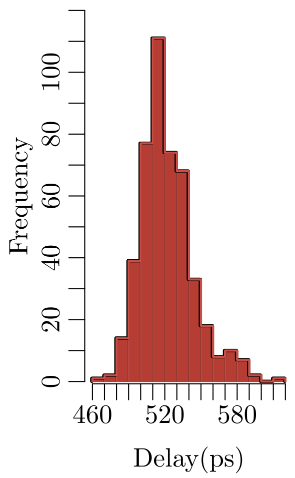


(a) Correlation to LUT Index

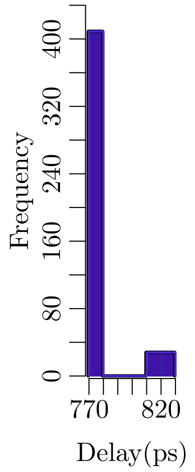


(b) Correlation to X Coordinate

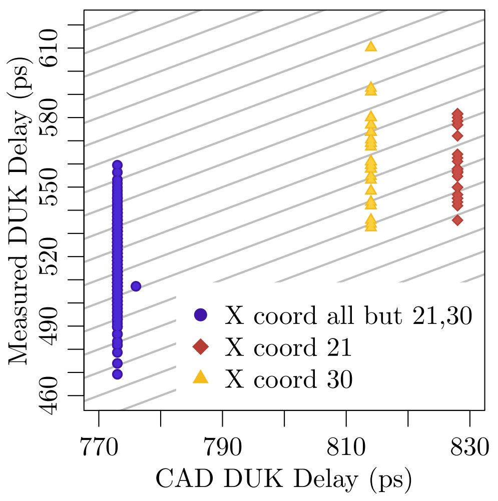
Figure 4.61: Correlation of CAD DUK delays to different DUK parameters.



(a) Measure DUK Distribution



(b) CAD DUK Distribution



(c) Correlation, separating X coordinate grouping

Figure 4.62: Correlation between measured DUK delays and CAD DUK delays for LAB-to-LAB C-DUKs on LUT input C, going left, using LUT number 7. Diagonal lines in Figure c, spaced at 10 ps intervals, show the angle where perfect correlation would occur.

On the other hand, the CAD DUKs belive these DUKs to have one of three possible delays: 773 ps, 814 ps, or 828 ps (Figure 4.62b). When we compare the two distributions, in Figure 4.62c, and account for the three X coordinate groupings, we fully explain the variation known to the CAD tools. However, it is clear that our measurements capture significantly more variation. In fact, when isolated by X coordinate grouping, for a resource the CAD tools claim to have no variation, we see spreads of 90 ps, 46 ps, and 78 ps. Although, the majority of this spread is probably explained by random variation, our measurments are aware of a systematic difference between the three X coordinate groupings. This is a strong indication that the CAD tools are not aware of all the variation in the FPGA, and Timing Extraction is essential to understand process variation in FPGAs.

4.11 Measurement Runtime

To undestand the time it takes to measure all paths, we develop the following formulas. We begin by determining the time it takes one bitstream to run. We divide it into three components, the time to load, or configure the bitstream on the FPGA, the time to run the bitstream, and the time to read back the results.

$$T_{biststream} = T_{configure} + T_{run} + T_{readResults}$$

The time to run breaks down into a sum over frequencies of the time to reconfigure the PLL for the current frequency, the time to give the start signal, and time to measure the path at the current frequency.

$$T_{run} = \sum_{f=f_{min}}^{f_{max}} T_{PLLReconfig} + T_{start} + T_{measure}$$

For each frequency, the host initiates a PLL reconfiguration, and then issues a start command, the time for these two events are tracked by $T_{PLLReconfig}$ and T_{start}. $T_{measure}$, represents the time it takes all the paths under test to measure their individual path at the current frequency and can be decomposed as follows:

$$T_{measure} = \#\text{paths in bitstream} \cdot \frac{\#\text{cycles at } f}{f}$$

It accounts for the fact that we run multiple cycles at the set frequency. The total number of frequencies also varies since the control logic stops incrementing the frequency once all path measurement circuits report having failed at least a threshold number of times.

For each of the 232,250 bitstream, we tracked how long it takes to run, broken into the individual delays described before. Figure 4.63 shows the runtime breakdown for each bitstream in one FPGA. Table 4.3 shows the averages over all bitstreams on all 18 FPGAs measured. On average we see that it takes almost 3 seconds to configure the bitstream. We believe this is a function of the board, programmed over a serial interface, and less dependent on the size of the logic being mapped to the FPGA. T_{run} also takes just under 3 seconds. The majority of this delay comes from the fact that we do a linear sweep over frequencies, on average testing 200 frequencies. A number already reduced by the fact that we perform an initial binary search to find the lowest frequency where we should begin our linear sweep. For each frequency, we test a path 2^{15} times and then, sequentially move on to test the next path in the bitstream. On average it takes 6.76 seconds to completely processes a bitstream. This means that to complete all our measurements required $6.76 \times 232,250 = 1,570,010$ seconds, or 18 days. The large runtime is not a function of the fundamentals of Timing Extraction, rather, it is due, in part, to the extremely careful measurements we take. Section 6.1 explores future approaches to greatly reduce this runtime by up to four orders of magnitude.

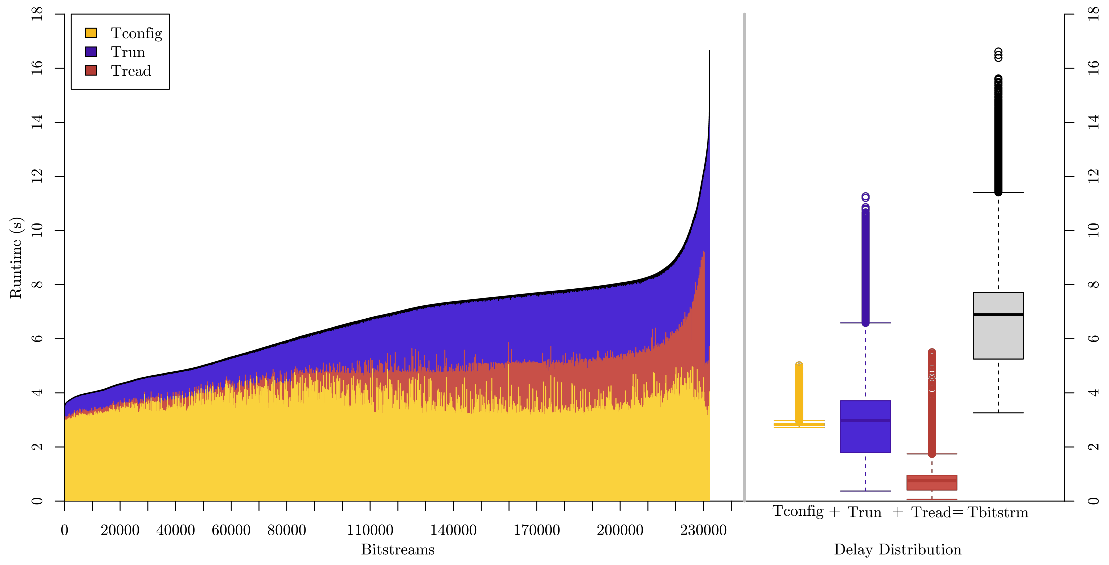
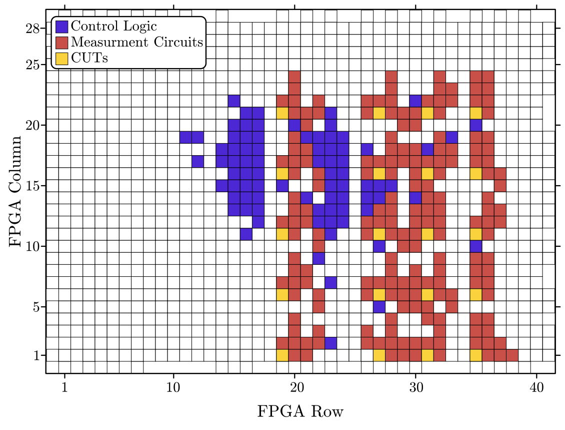


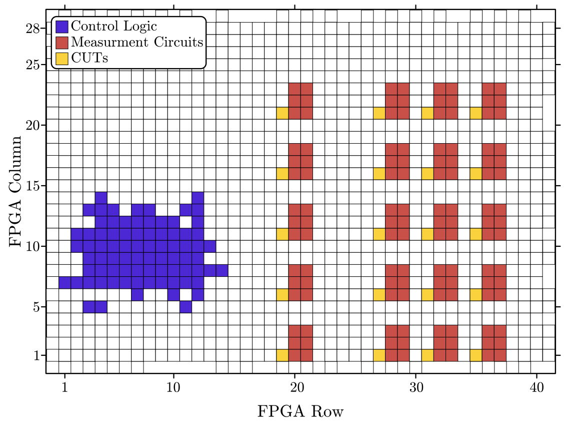
Figure 4.63: Breakdown of how long each bitstream takes to run for a given FPGA. Left side shows the breakdown for each bitstream ordered by total runtime. Right side shows the distribution for each time breakdown and the total time.

Component	**Mean**
$T_{configure}$	2.85 s
$T_{PLLReconfig}$	4.63 ms
T_{start}	46.5 μs
$T_{measure}$	9.81 ms
# frequencies	199
$T_{readResults}$	0.95 s
$T_{biststream}$	6.76 s

Table 4.3: Average delay and frequency count breakdown over all bitstreams and measured FPGAs



(a) Limited constraints



(b) Extensive constraints

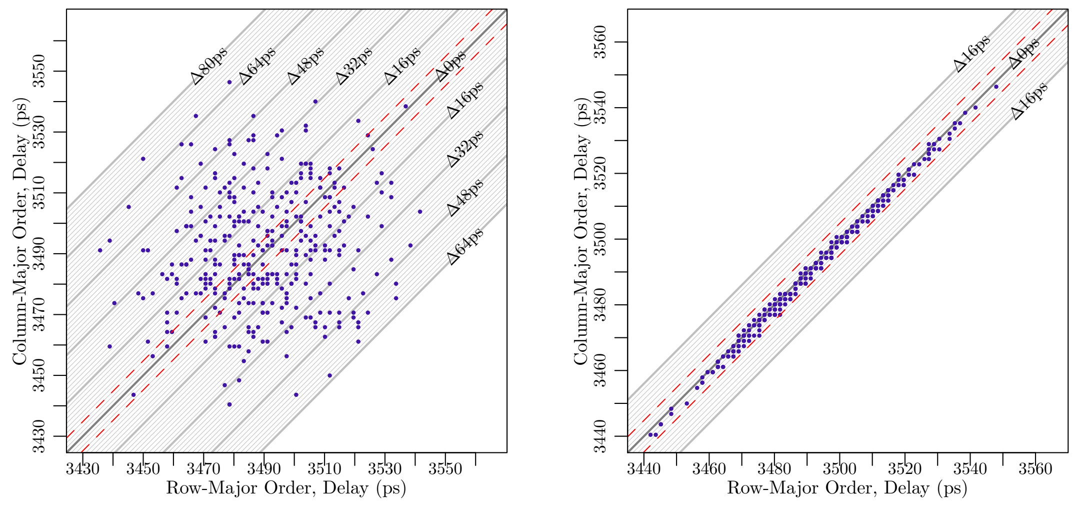
Figure 4.64: Floor plan of two bitstreams measuring the same paths, but having different placement constraints.

4.12 Measurement Consistency

Timing Extraction aims to measure process variation while minimizing the effects of environmental and operational variation. To achieve this, we highly constrain the placement and routing of the measurement and control circuits. Moreover, we measure paths sequentially so that the activity produced by measuring one path, does not affect the measurement of other paths. In this section we justify the need for this extreme control, and demonstrate that we can successfully measure process variation by consistently measuring DUK delays.

To understand the need for isolation between the measurement logic and the control logic, we perform the following simple experiment. We generated two sets of bitstreams, both sets have a 4×5 array of measurement circuits on the right side of the FPGA, each measuring the same path in a different LAB. However, for one set we only constrained the placement and routing of the launch register, CUT, and capture register, while the second completely constrains the placement and routing of all the logic, as described in Section 4.4.4. Figure 4.64 shows the placement for a bitstream in each set. We expect the order in which we measure each of the 20 paths to be independent of the path delays. To test this, we instruct the bitstreams to measure the paths first in row-major order, where every path on the same row in the array is measured before measuring paths in the next row. We then instructed the bitstreams to measure in column-major order.

If we are measuring process variation, we expect that the order in which we measure the LABs should have almost no impact on the results. We compare the results of row-major and column-major measurements by plotting the delays of the column-major run against the row-major results. Since we expect path measurement order not to affect path-delays, we expect to see all points lie



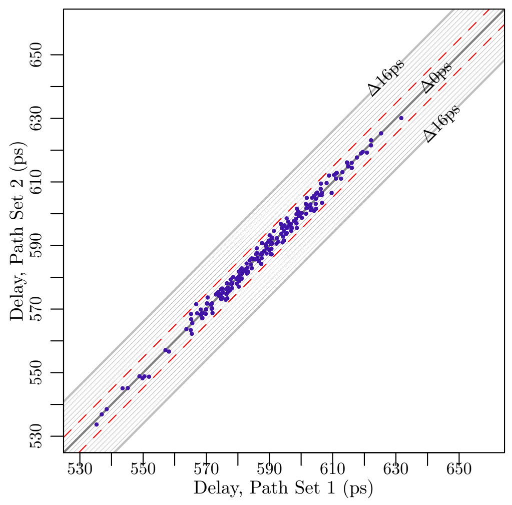
(a) Limited constraints (b) Extensive constraints

Figure 4.65: Correlation between row-major and column-major delay measurement results. Figure a shows the result when limited constraints are imposed as opposed to Figure b where placement and routing were heavily constrained. Diagonal lines indicate the difference between the two results in terms of $d_{\Delta} = 1.6$ ps. Thicker lines indicate $10d_{\Delta}$. Red dashed lines indicate error boundaries as computed in Section 3.5.

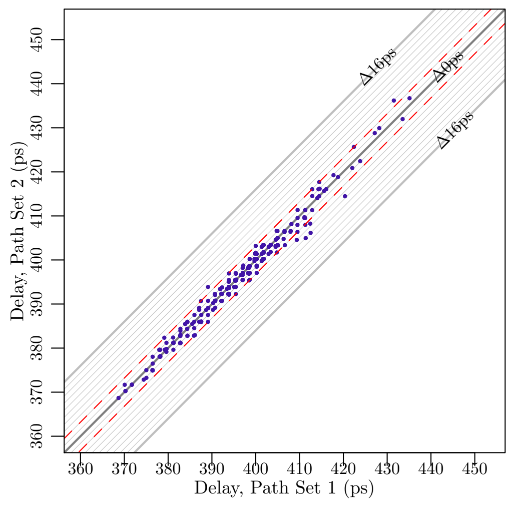
on a diagonal, within the measurement error expected (Section 3.5). Figure 4.65 shows the results for the two sets of bitstreams. Immediately, we see that placement and routing constraints have a huge effect on measurement consistency. Figure 4.65a reveals a difference of up to 72 ps, between paths that should have the same delay. Figure 4.65b shows that controlling placement and routing is a necessity if we are to measure process variation and not other sources of variation.

A second test to confirming that Timing Extraction is measuring process variation comes from the fact that we can compute the same DUK delay using many different paths. Recall that DUK delays are computed by taking linear combinations of a few paths. As explained in Section 3.4.4, we are free to choose most of the components that compose these paths, as long as we meet certain requirements. For example, for the two paths needed for a C-DUKs, there is a stated relationship that must exist between the ends of the two paths. We are, however, free to choose the beginning of the paths, as long as they match in both paths. This observation leads to a simple way to test our measurement results.

We generated two separate sets of paths for both M-DUKs and C-DUKs. These path sets differ in the LC Nodes unrelated to the DUK. After measuring the paths and computing the DUK delays, we compare the DUK-delays from both sets in Figure 4.66. The figure demonstrates that we can consistently compute DUK-delays regardless of what other resources are used in the paths we measure.



(a) M-DUK Path Set vs Path Set



(b) C-DUK Path Set vs Path Set

Figure 4.66: Correlation between DUKs when measuring different path sets yielding the same DUKs. 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 3.5.

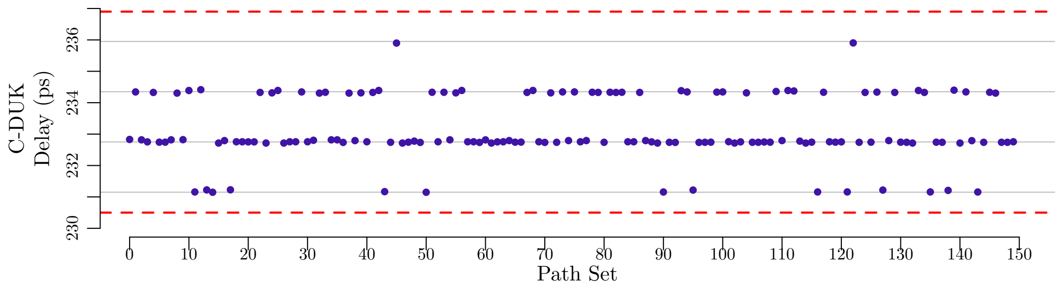


Figure 4.67: Delay of a C-DUK from LE 6 to LE 11 in LAB (27,22) using input D, computed using 150 different pairs of paths. The distance between horizontal gray lines is $\Delta_{clock} = 1.6$ ps. The region between red dashed lines shows expected error bounds as computed in Section 3.5. All 150 C-DUKs are within these bounds. Cyclone III

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 4.67 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.

4.13 Scaling

Section 3.4.5 explains how the number of DUKs is bounded by the fanout of the physical resources graph. The number of edges in a physical resource graph of a circuit is linear in the number of physical resources. This fact places a bound on the fanout of a circuit. As such, the number of DUKs is also linear in the number of physical resources. Therefore, the work required to compute all DUKs is linear in the number of resources. Also, since the number of paths that must be measured to compute the delay of a DUK is constant, either two or three, the number of paths that we must measure, and the time to measure these paths, is also linear in the number of physical resources in the FPGA.

Finally, we consider the number of bitstreams we must generate to measure all paths. Although the number of DUKs is linear in the number of physical resources, the number of bitstream is asymptotically constant, regardless of the size of the FPGA. The reason for this stems from the regularity of FPGAs. In an FPGA we can identify a small region of physical resources that is repeated to create the complete FPGA. For the resources we characterized on the Cyclone III, this repeating region is approximately a 6×6 LAB region. As previously explained (Section 4.4), we can pack multiple path measurement circuits in each bitstream. Thus, although the DUKs within the repeating region need to be measured sequentially—requiring a separate bitstream for each path measured in the repeating region—we can pack multiple path measurement circuits within one bitstream, as long as each comes from a different repeating region. Therefore, the total number of bitstreams is only determined by the number of paths that we must measure in one repeating region, not by the total number of physical resources we want to characterize. The total number of paths within one bitstream will be on the order of the number of repeating regions. In general, we may need twice as many paths since any time we measure paths, we reserve a portion of the FPGA for the control logic (Section 4.4.2).

4.14 Chapter Summary

Applying Timing Extraction to a commercially available FPGA reveals a vast amount of information about the FPGA. Given the physical resource graph of the FPGA, Timing Extraction decomposes the graph into a small number of M-DUKs and a large number of C-DUKs. Each C-DUK type allows us to transform a different aspect of the path, such as which LUT input to use, or to which LAB a general interconnect wire should connect. Once the physical resource graph is decomposed into DUKs, and we have determined which paths to measure, we pack multiple paths into one bitstream, along with the logic to control and report back the measurement results. Placement and routing constraints are generated for each bitstream, that is then complied and run on the FPGA. Computing DUK delays from these measurements reveals significant process variation. Separating this variation based on known systematic design differences, we begin to understand the sources of variation. However, a detailed analysis of this variation is still necessary to grok [37] the contributions from different variation types. The next chapter presents this analysis.

Beyond revealing raw variation, Timing Extraction gives insight into other aspects of the FPGA. We observe a systematic difference between transitions, where rising transitions are, on average, slower than falling transitions. By exploring the internal structure of the LUTs, we see how Timing Extraction naturally adjusts based on what the physical resource graph exposes. Giving Timing Extraction a physical resource graph that exposes the internals of the LUT, Timing Extraction can provide detailed insight into the behavior of the LUTs. Using DUKs, we can also understand how variation is affected by the different properties of the FPGA. We adjust the supply voltage and empirically observe that variation significantly increases as the supply voltage decreases. Finally, a comparison between our measured delays and the delays reported by the CAD tools reveals that the CAD tools are unable to model the full spread of process variation.

We conclude the chapter by exploring properties of Timing Extraction, including its runtime, consistency, and scalability. In order to achieve repeatable, consistent measurements, we measure each path in isolation, many many times. This leads to a long runtime. However, as Section 6.1 explains, it is likely that we are being overly cautious and, after some experimentation, we can potentially greatly reduce the runtime. Our cautious measurements lead to repeatable results, giving high confidence that we are cleanly measuring the delay of physical resources, and not other effects such as environmental or operational variation. Finally, we explain that the number of DUKs extracted by Timing Extraction is linear in the number of physical resources. Moreover, the amount of work required to extract these DUKs is also linear, as is the time to measure and compute their delays. We also note that due to the regular structure of FPGAs, the number bitstreams is asymptotically constant, regardless of the FPGA size, since we can pack more path measurement circuits into one bitstream as the FPGA grows.

Chapter 5

Variation Analysis

Timing Extraction provides the fine grain delay of individual DUKs. Simply using DUKs to perform a component-specific mapping leads to significant energy savings [63]. However, using DUKs to understand how much variation is correlated to a process or design parameter, and how much is random variation, can inform and improve the manufacturing process as well as the CAD algorithms. Using the correlations discovered, we can adjust both to mitigate this variation. Moreover, by understanding the magnitudes of both random and correlated variation, we can determine if random variation dominates, and instead decide that the best approach to mitigate the variation, is a component-specific mapping. For example, if we detect that variation correlated to a particular region of the FPGA is high, our mitigation technique would involve changing the placement of our logic to a better region. On the other hand, if random variation dominates, moving from one location on the FPGA to another will not help. Simply selecting a better resources from the locally available resources, however, will.

In this chapter we present an analysis that decomposes the variation of the DUK delays. It determines how the delays are correlated to different parameters, and how much delay is due to random processes. We use the simple delay model from Section 2.2.1, as well as our understanding of the architecture of the FPGA (Section 4.1) to build our analysis. Since both models are limited, our analysis is only an initial attempt at understanding the variation. Building detailed architectural and delay models is beyond the scope of this work, and is left as future work. Nevertheless, these simple models illuminate our understanding of variation, and provide a structure on which to demonstrate how this kind of analysis works.

5.1 Variation Model

We model delay as the time it takes to charge a capacitive load through a resistance, as Equation 2.1, reproduced below, shows.

$$\tau_{pd} = C_l \cdot \frac{V_{ds}}{I_{ds}} \tag{2.1}$$

Section 2.3 explains that the primary source of variation of the current, I_{ds}, is due to threshold voltage, V_{th}, variation, which, in turn, varies due to both random and correlated physical variations. The capacitive load, C_l, comes from both charging the transistors and the wires in the FPGA. Both of which also suffer from physical variations.

There are three sources for these variations. First, as Section 2.3.1.1 describes, the complex manufacturing process has many avenues through which variation can manifest. The second source is design variation, that is, the variation introduced, purposefully or not, through the architectural design of the FPGA. We have already seen proof of this variation source in Section 4.8, where we clearly see that the LUT inputs have different delays by design. The third source comes from the fact that we have different types of DUKs, where each type has a different number of physical resources, used in a particular configuration. Since we only have a limited model of what the architecture looks like, we are unable to cleanly separate design variation from process variation. We do, however, know exactly how DUKs differ. Thus, our results will show the combined effect of design and process variation, while isolating DUK type.

Since variation affects both the resistance and capacitance of our circuit, and in the simple delay model, these terms are multiplicative, our variation decomposition is multiplicative, as shown in Equation 5.1. A more complex delay model would decompose variation accordingly.

$$\tau_{DUK} = \tau_0 \cdot \tau_{FPGA} \cdot Reg(x, y) \cdot Sys(sd_1, \ldots, sd_n) \cdot Rand \tag{5.1}$$

Our decomposition has five components:

- τ_0: A base delay
- τ_{FPGA}: The delay bias of an FPGA, relative to τ_0.
- $Reg(x, y)$: A regional correlation function that explains how the DUK delay changes as a function of its physical position (x, y).
- $Sys(sd_1, \ldots, sd_n)$: This term represents systematic variation, and is composed of a set of functions that correlate delay changes to subspecies differences (SDs Section 4.6). For

DUK Type	τ_0 (ps)
Intra-LAB M-DUK	475
LAB-to-LAB M-DUK	620
Intra-LAB C-DUK	265
LAB-to-LAB C-DUK	411
LAB-to-General Interconnect C-DUK	213
Embedded Column Neighbor C-DUK	-32
General Interconnect C-DUK	218
General Interconnect Sink Select C-DUK	-11
LLT Crosspoint Swap C-DUK	63
LIT Crosspoint Swap C-DUK	-60

Table 5.1: τ_0 for each of the 10 DUK types

example, $Sys_{LUT_{in}}(lut_{in})$ correlates the delay to the LUT input used by the DUK.

- *Rand*: This term captures the variation that cannot be explained by the other terms, and is what quantifies the random variation in the DUK.

To extract the contribution of the first four terms, we capitalize on the vast number of DUKs we extract, and depend on the central limit theorem to correctly compute their value. The remainder is assigned to *Rand*.

5.2 Computing τ_0 and τ_{FPGA}

To calculate τ_0, for each DUK type, we compute the mean delay from all DUKs, from all FPGAs measured. Similarly, we compute τ_{FPGA} by performing the same computation, but using only the DUKs from one FPGA on either the rising transitions or falling transitions. In this way we can see how FPGAs differ and how rising versus falling transitions differ.

Table 5.1 shows the value of τ_0 for each DUK. Figure 5.1 shows τ_{FPGA} for LAB-to-LAB MDUKs. To get a more meaningful sense of how different FPGAs compare, the figure actually shows $\tau_0 \cdot \tau_{FPGA}$ scaled by the slowest τ_{FPGA}. For LAB-to-LAB MDUKs, $\tau_0 = 620$ ps. As we also saw in Section 4.7 rising transitions are slower than falling transitions. From τ_{FPGA} we see that, on average, rising transitions are systematically 4% slower than falling transitions.

5.3 Regional Variation

Regional variation consists of delay changes that correlate to the physical location of the DUK. There are many manufacturing processes that may cause one region of the FPGA to be faster

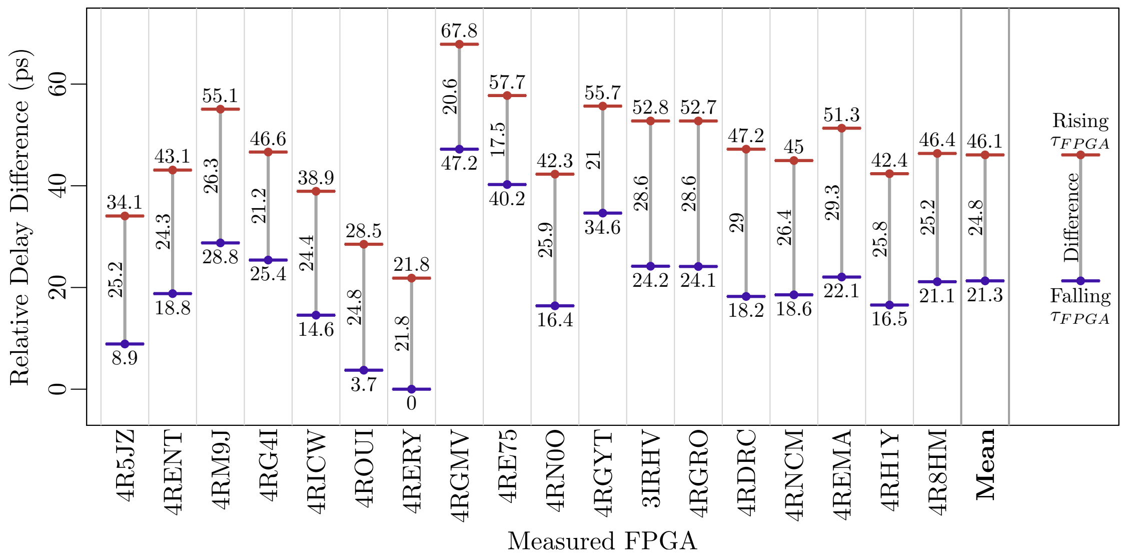


Figure 5.1: τ_{FPGA} for LAB-to-LAB MDUKs from the 18 measured FPGAs, showing the difference between falling and rising transition

than another (Section 2.3.1.1). By averaging all resources in a given region, we can discover these regional changes.

To understand the process, we begin by considering what τ_{FPGA} represents. When we average all DUK delays of one kind from one FPGA, we essentially reduces the contribution from other sources of variation, including random variation. This allows us to separate the DUK delays into τ_{FPGA} times a remainder. τ_{FPGA} describes the delay contribution for the region consisting of the whole FPGA, while the remainder includes all other variation. Taking this idea, we consider what happens when, instead of taking the mean over the whole FPGA, we do so over a moving window as described by Equation 5.2.

$$Reg(x,y) = \frac{\displaystyle\sum_{i=x-z}^{i=x+z}\sum_{j=y-z}^{j=y+z}\frac{\tau_{DUK}(i,j)}{\tau_0\cdot\tau_{FPGA}}}{(2z+1)^2} \tag{5.2}$$

When z is large enough, $Reg(x,y)$ considers the whole FPGA, and equals $\frac{1}{\tau_0}$, since we scale τ_{DUK} by $\tau_0\cdot\tau_{FPGA}$. As we reduce the size of z, we begin to consider individual regions. However, a smaller window does not reduce the contribution form other variation sources as well as a larger window. $z=2$ is a good compromise [89].

Computing $Reg(x,y)$ for each FPGA coordinate gives us a good sense of the regional variation in the FPGA. However, to get a better idea of how regional variation behaves, we further break down regional variation into systematic regional within-die variation $Reg_{sys}(x,y)$, and random regional within-die variation $Reg_{rand}(x, y)$. Equation 5.3 shows the relationship between the three regional measures.

$$Reg(x, y) = Reg_{sys}(x, y) \cdot Reg_{rand}(x, y) \tag{5.3}$$

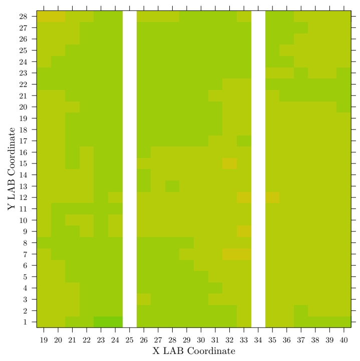
$Reg_{sys}(x, y)$ represents the regional variation that is common to all FPGAs, while $Reg_{rand}(x, y)$ is the regional variation unique to one FPGA. We easily compute $Reg_{sys}(x, y)$ as the mean of $Reg(x, y)$ over all FPGAs (Equation 5.4).

$$Reg_{sys}(x, y) = \frac{\sum_{FPGAs} Reg(x, y)}{|FPGAs|} \tag{5.4}$$

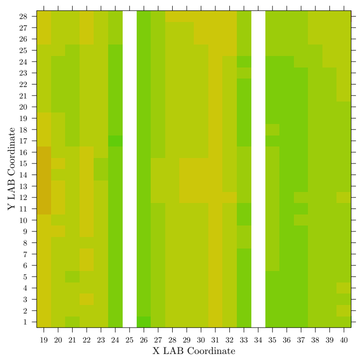
Finally, having both $Reg(x, y)$ and $Reg_{sys}(x, y)$ we compute $Reg_{rand}(x, y)$ using Equation 5.3.

5.3.1 Systematic Regional Variation

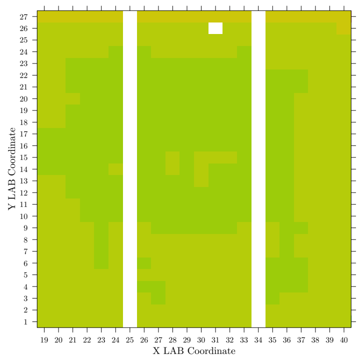
Figure 5.2 shows the $Reg_{sys}(x, y)$ for each DUK type, except for Embedded Column Neighbor C-DUK, since this DUK only exists next to embedded columns, and the windowed average does not behave as required to compute regional variation. We scale $Reg_{sys}(x, y)$ and center it so that it is easy and meaningful to compare across DUK types. From this figure, we see that only LAB-to-General Interconnect C-DUKs (Figure 5.2g) and General Interconnect C-DUKs (Figure 5.2h), both of which use general interconnect resources, exhibit meaningful systematic regional variation. There appears to be a gradient, where slower DUKs, on the top left, give way to faster DUKs towards the bottom right. Although General Interconnect Sink Select C-DUKs (Figure 5.2i) also use routing resources, the structure of its two paths is the same, which cancels out any regional variation effects. As for DUKs that do not use general interconnect resources, it is worth noting that for both LAB-to-LAB M-DUKs and C-DUKs, it appears as though DUKs next to embedded column have a slightly lower delay. The reality, however, is that there is a systematic correlation to the X LAB coordinate, where columns 20, 21, 29 and 30 actually have a higher DUK delay, as we will see in Section 5.4. The windowed averaging has the effect of blurring and spreading this delay among adjacent DUKs. Nevertheless, the window size is of sufficient dimension to make the magnitude of this blurring so small, it is almost negligible in the systematic regional variation.



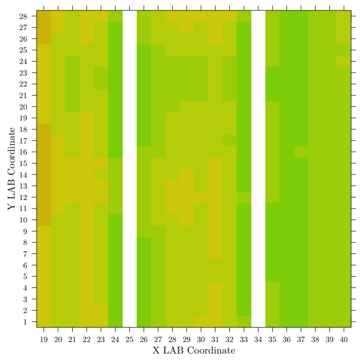
(a) IntraLAB M-DUK



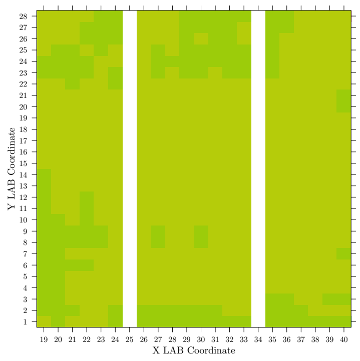
(b) LAB-to-LAB M-DUK



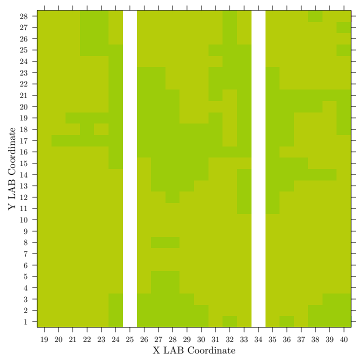
(c) IntraLAB C-DUK



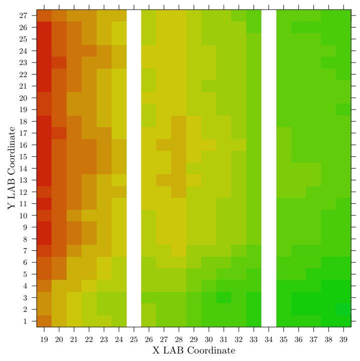
(d) LAB-to-LAB C-DUK



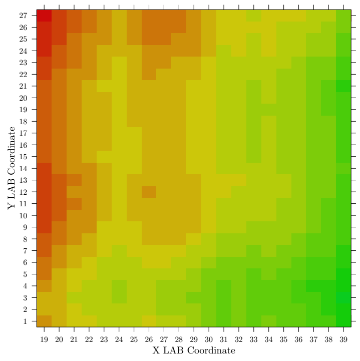
(e) LLT Xpt C-DUK



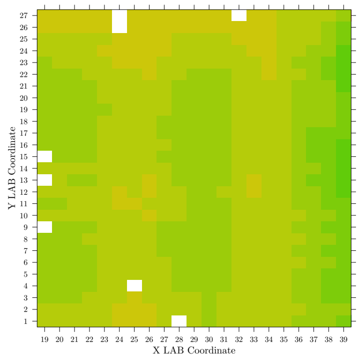
(f) LIT Xpt C-DUK



(g) LAB-to-GI C-DUK



(h) GI C-DUK



(i) GI Sink Select C-DUK

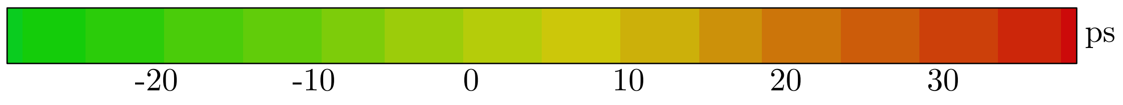


Figure 5.2: Regional Systematic Variation shown on a floor plan of the Cyclone III FPGA

5.3.2 Random Regional Variation

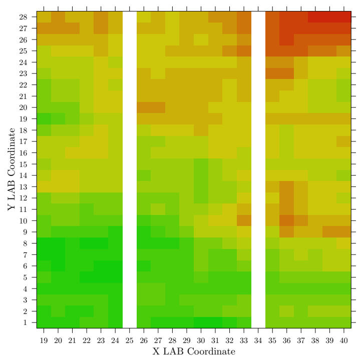
We perform a similar plot for $Reg_{rand}(x, y)$. In contrast to Figure 5.2, in Figure 5.3 we only show the plots for one FPGA, since random regional variation is FPGA specific. From this figure, we see that M-DUKs have a larger random regional variation range than C-DUKs, also aside from IntraLAB and LAB-to-LAB C-DUKs, other C-DUKs have almost no random regional variation. This is a direct result of the different structures of the DUKs. M-DUKs represent small paths, while C-DUKs represent the difference of two small paths. It is this difference that reduces the magnitude of $Reg_{rand}(x, y)$. The structure of a C-DUK is such that all its LC Nodes come from the same, or close to the same region in the FPGA. When a C-DUK subtracts two LC Nodes that use exactly the same type of physical resources, any regional variation cancels out. Both IntraLAB and LAB-to-LAB C-DUKs have a Current Tail consisting of only an End Node, as such, most of their LC Nodes in the Desired Tail do not have a corresponding LC Node in the Current Tail, which explains why they manifest the largest random regional variation range among C-DUKs. The other C-DUKs have almost no random regional variation either because most of their regional variation is systematic, or their structure hides any regional variation.

Since it is M-DUKs that best demonstrate the magnitude of $Reg_{rand}(x, y)$ for all other FPGAs, we simply show the random regional variation for their LAB-to-LAB M-DUK in Figures 5.4 and 5.5. Although the overall magnitude range is only 30 ps, it is clear that each FPGA has a unique random regional variation pattern, almost akin to a fingerprint.

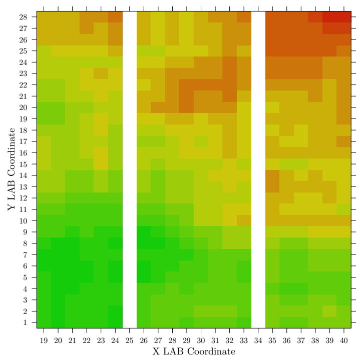
5.4 Systematic Variation

Having τ_0, τ_{FPGA}, and $Reg(x, y)$, we turn to the systematic variation, $Sys(sd_1, \ldots, sd_n)$. In Section 4.6 we present the idea of a DUK subspecies. Each DUK has a number of subspecies differences, or SDs, such as the wire index used, or the LUT input used. A subspecies has a unique combination of SDs. $sys(sd_1, \ldots, sd_n)$ aims to identify differences between subspecies by indentation how each SD contributes to the delay of the DUK. For clarity, capitalized SD_i refers to the type of subspecies difference, while lower case sd_i refers to a particular value for this type of subspecies differences. For example, SD_{LUTIn} differentiates DUKs by the LUT input used. While sd_{LUTin} can take the values A, B, C, or D, for the different LUT inputs.

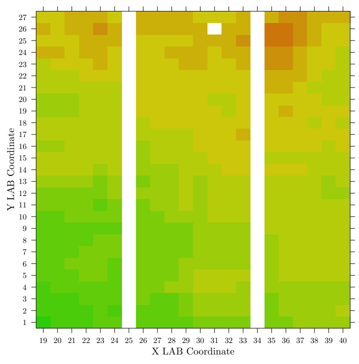
A shortfall of this systematic decomposition is that it only considers systematic correlations based on given SDs, and thus, is based solely on a priory assumptions of possible systematic effects. It cannot discover new correlations based on the given DUK delays. Nevertheless, if a different effect is suspected of leading to systematic variation, we can easily test that hypothesis



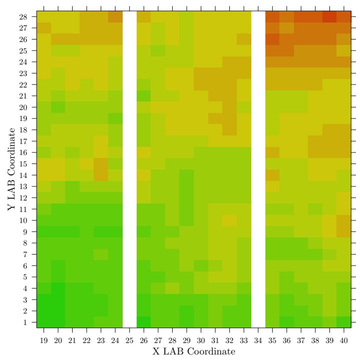
(a) IntraLAB M-DUK



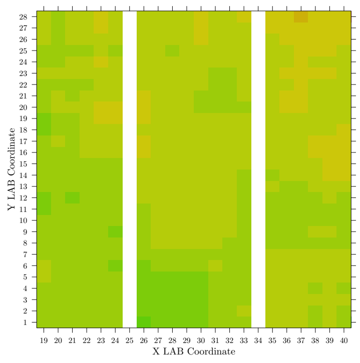
(b) LAB-to-LAB M-DUK



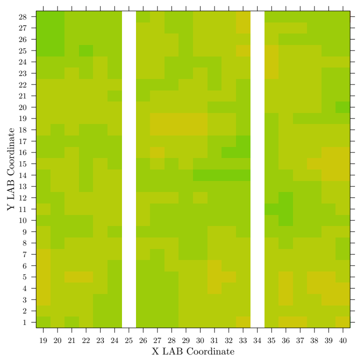
(c) IntraLAB C-DUK



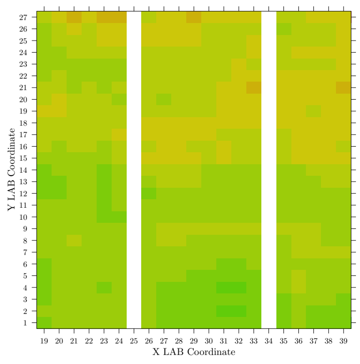
(d) LAB-to-LAB C-DUK



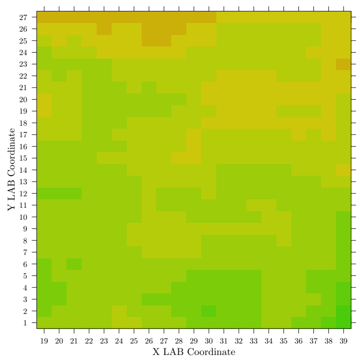
(e) LLT Xpt C-DUK



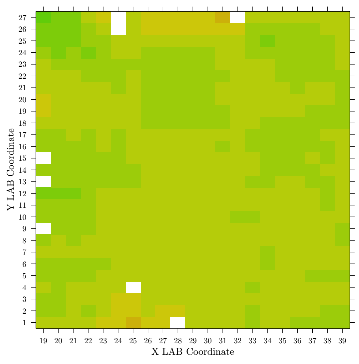
(f) LIT Xpt C-DUK



(g) LAB-to-GI C-DUK



(h) GI C-DUK



(i) GI Sink Select C-DUK

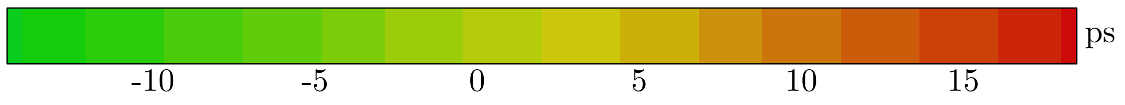


Figure 5.3: Regional Random Variation of FPGA 4RE75, shown on a floor plan of the Cyclone III FPGA

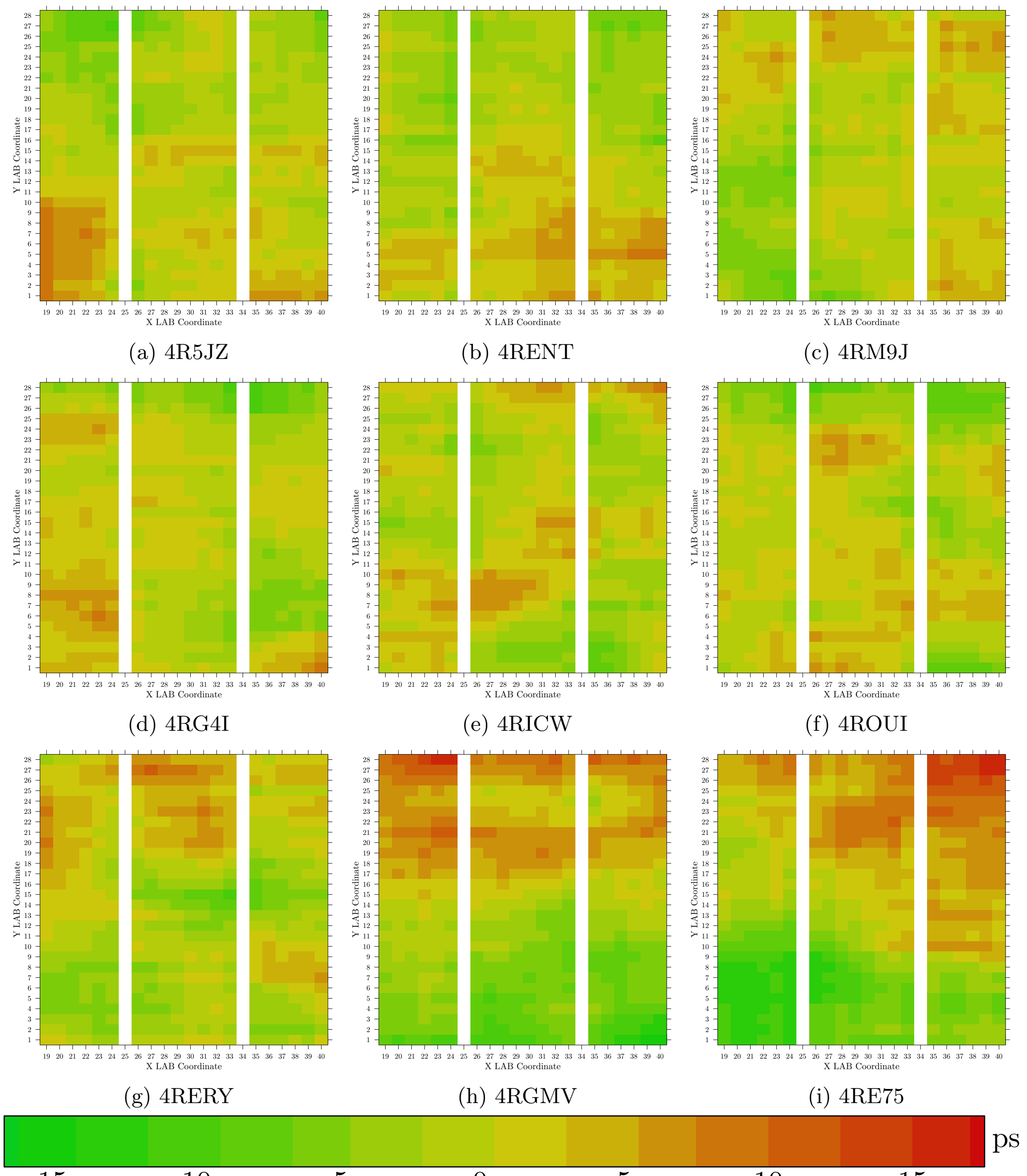
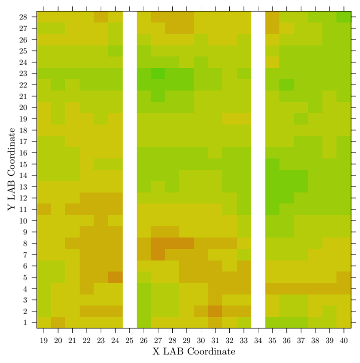
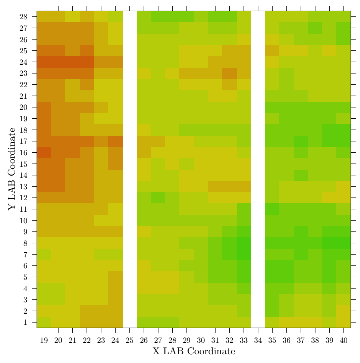


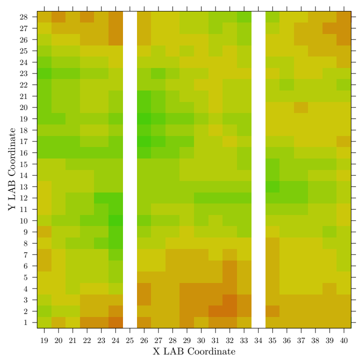
Figure 5.4: Regional Random Variation of the LAB-to-LAB M-DUK for first 9 Measured FPGAs, shown on a floor plan of the Cyclone III FPGA



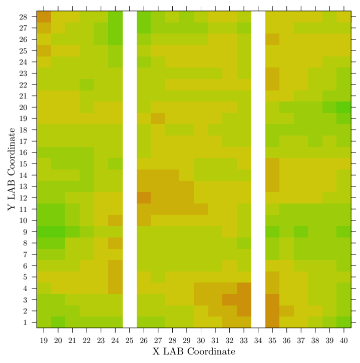
(a) 4RN0O



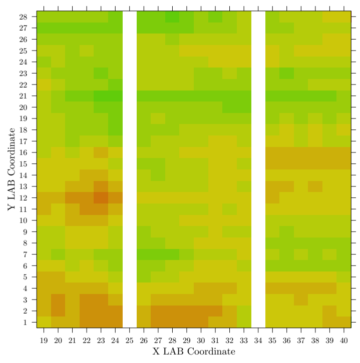
(b) 4RGYT



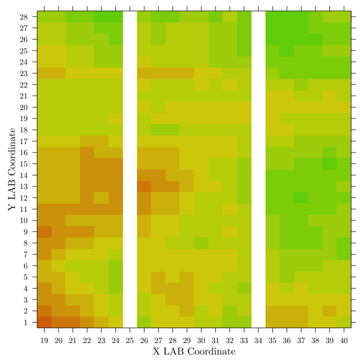
(c) 3IRHV



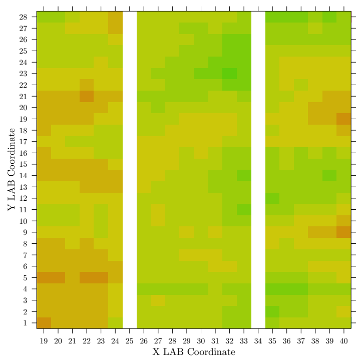
(d) 4RGRO



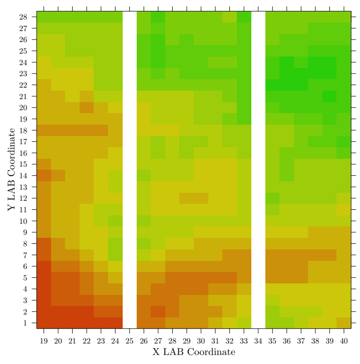
(e) 4RDRC



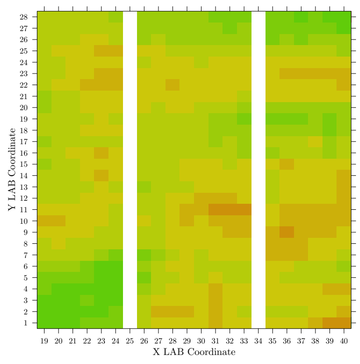
(f) 4RNCM



(g) 4REMA



(h) 4RH1Y



(i) 4R8HM

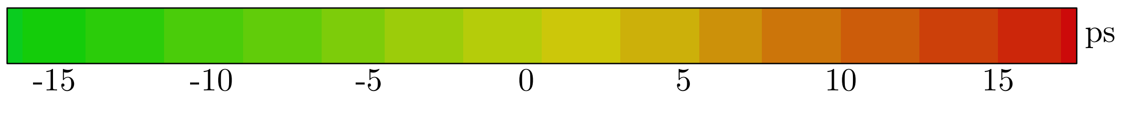


Figure 5.5: Regional Random Variation of the LAB-to-LAB M-DUK for second 9 Measured FPGAs, shown on a floor plan of the Cyclone III FPGA

using this decomposition.

To compute $Sys(sd_1, \ldots, sd_n)$, we begin by scaling τ_{DUK} by τ_0, τ_{FPGA}, and $Reg(x, y)$ in order to remove the effect of these variation sources from the DUK delay. What remains is the product of systematic variation times random variation. For convenience, we refer to this as $\tau_{SysRand}$ (Equation 5.5).

$$\tau_{SysRand} = \frac{\tau_{DUK}}{\tau_0 \cdot \tau_{FPGA} \cdot Reg(x, y)} = Sys(sd_1, \ldots, sd_n) \cdot Rand \tag{5.5}$$

Assuming that a DUK type has n SDs, $Sys(sd_1, \ldots, sd_n)$ is the product of n functions, each identifying the contribution of a particular SD (Equation 5.6).

$$Sys(sd_1, \ldots, sd_n) = \prod_{i=1}^{n} Sys_{SD_i}(sd_i) \tag{5.6}$$

To compute $Sys_{SD_1}(sd_1)$ we group the DUKs by the characteristic described by SD_1, and take the mean of each group. This give us the contribution of SD_1 to $\tau_{SysRand}$. In order to extract $Sys_{SD_2}(sd_2)$, we first scale $\tau_{SysRand}$ by $Sys_{SD_1}(sd_1)$, and then repeat the process using SD_2 as the grouping factor (Equation 5.7). Continuing until we compute $Sys_{SD_n}(sd_n)$ gives us the value of $Sys(sd_1, \ldots, sd_n)$.

$$Sys_{SD_i}(sd_i) = \frac{\displaystyle\sum_{DUK_{SD_i} = sd_i} \frac{\tau_{SysRand}(DUK)}{\prod_{j=1}^{i-1} Sys_{SD_j}(sd_j)}}{|DUK_{SD_i} = sd_i|} \tag{5.7}$$

The value of $Sys(sd_1, \ldots, sd_n)$ is independent of the order in which we compute the $Sys_{SD_i}(sd_i)$ functions. However, in order to understand how a particular SD affects the DUK delays, it is desirable to look at $Sys_{SD_i}(sd_i)$ itself. Unfortunately, the order in which SDs are evaluated will affect the value of this function.

Our process requires that we compute $Sys_{SD_i}(sd_i)$ before computing $Sys_{SD_{i+1}}(sd_{i+1})$. What this means is that $Sys_{SD_{i+}}(sd_{i+1})$ quantifies the systematic contribution of SD_{i+1} *after* removing the contributions of SD_1 through SD_i, but *before* considering the contributions of SD_{i+2} through SD_n. If all SDs were independent, the order in which we evaluate them, would not affect the value of $Sys_{SD_i}(sd_i)$. However, subspecies differences are not always independent. For example, in Section 4.8 we saw that LUT input C is slower than LUT input D because of the structure of the LUT. However, since LUT input C and D can take a signal from a mutually exclusive set of LLTs (Figure 4.3 top left), the order in which we compute $Sys_{SD_{LLT}}(sd_{LLT})$ and $Sys_{SD_{LUTin}}(sd_{LUTin})$, will determine whether the difference between inputs C and D is quantified by $Sys_{SD_{LLT}}(sd_{LLT})$

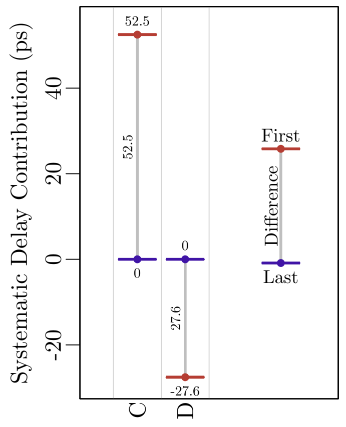
or by $Sys_{SD_{LUTin}}(sd_{LUTin})$. In this case, we know that the difference is due to the structure of the LUT, and therefore, would make more sense to attribute to $Sys_{SD_{LUTin}}(sd_{LUTin})$; however, lacking intimate knowledge of the FPGA, and even with this knowledge, it may not always be clear which SD should be examined first. As such, in our results, when we look at a particular $Sys_{SD_i}(sd_i)$, we always show what its value would be if this SD was the first SD computed, and what its value would be if it was the last SD computed, giving an upper and lower bounds on the magnitude of the SD's contribution to the DUK delay. A large difference between these values indicates that this SD correlates with other SDs, and therefore other SDs would capture the delay contribution if they were computed first. On the other hand, zero, or a minimal difference is strong indication that this SD is exclusively responsible for the contribution to the delay.

5.4.1 Systematic Delay Contributions

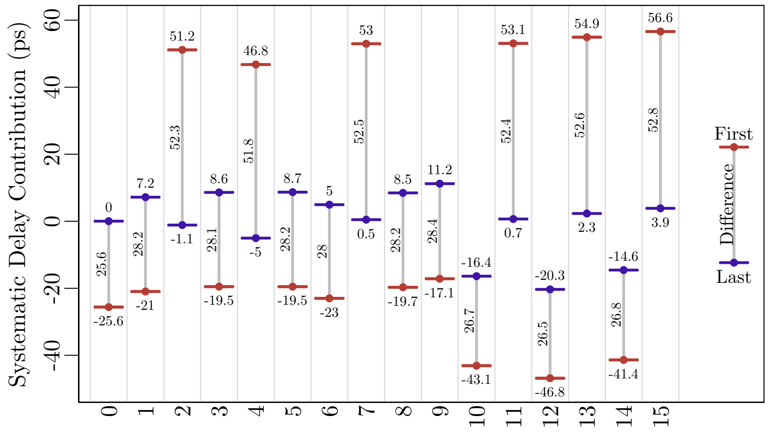
The first set of results we look at are the LLT versus LUT input example from above. Figure 5.6a shows the systematic contribution of LUT input C and D. Next to it, Figure 5.6b shows the LLT systematic contribution. Consistent with the results in Sections 4.6 and 4.8, we find that input C is slower than input D if we compute $Sys_{SD_{LUTin}}(sd_{LUTin})$ first (red bars). However, if it is computed last (blue bars), Figure 5.6a claims there is no delay contribution from different LUT inputs. The reason lies in Figure 5.6b. If $Sys_{SD_{LLT}}(sd_{LLT})$ is computed first, we see that indices 2, 4, 7, 11, 13, and 15 have a delay contribution comparable to that expected of input C, and these indices are the only LLTs from which input C can take a signal (Figure 4.3 top left). Similarly for the other LLTs and input D. Regardless of when $Sys_{SD_{LLT}}(sd_{LLT})$ is computed, however, this figure clearly shows a difference between LLTs, where LLTs 10, 12, and 14 are systematically faster than the rest.

Using the LAB-to-LAB M-DUK, we turn our attention to the connections between LABs. Figure 5.7a shows how the direction of this connection affects the DUK delay. Measured first, this SD shows that left connections are approximately 30 ps slower than right connections. On the right side, Figure 5.7b tracks how the X coordinate of the beginning of the connection contributes different systematic delays. Looking at the delays when this SD is measured last reveals two things. First, coordinates 20, 21, 29 and 30 are significantly slower than the other columns. This delay correlates directly with the long slow distribution tails in Figure 4.25 and 4.31. The second thing to note is the wave-like pattern in the delay of the other columns.

For the most part, there is very little difference when measuring this SD first or last, however, columns 19, 24, 26, 33, 35, and 40 show a difference. This difference is easily explained when we look at the delay contribution from the direction of the path (Figure 5.7a), and notice that these

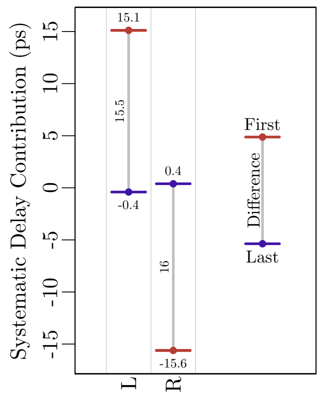


(a) LUT Input

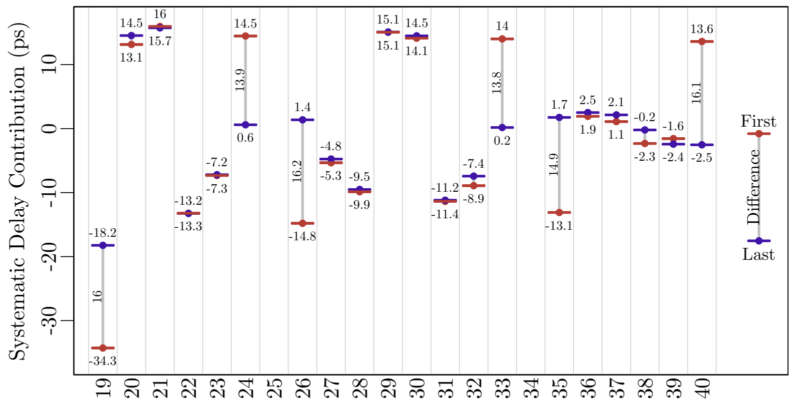


(b) LAB Local Track (LLT) Index

Figure 5.6: Systematic variation internal to the LAB



(a) Path Direction



(b) Source LAB X Coordinate

Figure 5.7: Systematic variation of LAB-to-LAB connections

columns are all adjacent to either embedded block columns or the edge of the FPGA (Figure 4.2). For all other columns, we measure DUKs that go to adjacent LABs to the left and right; however, DUKs in these columns only have adjacent LABs in one direction, and thus paths that go in only one direction. Columns 19, 26, and 35 only go right, while 24, 35 and 40 only go left. As such, if this SD is measured first, we see that these columns combine the contribution of their delay with that of the path direction.

Finally, we analyze the systematic delays of general interconnect wires. Figure 5.8 demonstrates the difference between horizontal, R4, wires and vertical, C4, wires. Clearly vertical wires are slower than horizontal wires. Since both vertical and horizontal wires are 4 LAB units long, their delay difference is a strong indication that physically, LABs are taller than they are wide. This is consistent with other Altera designs [92].

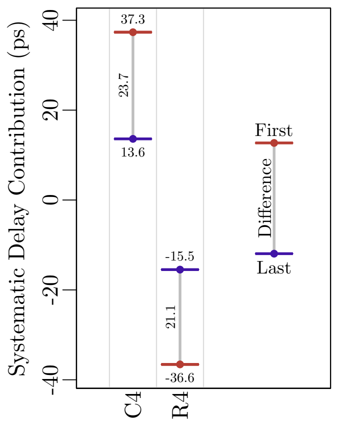
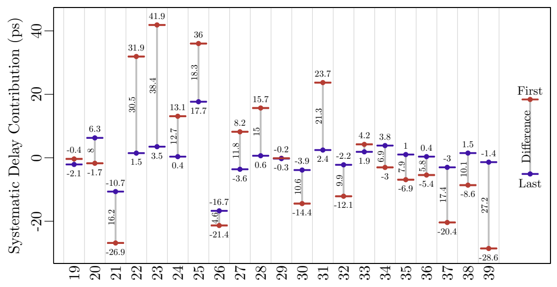


Figure 5.8: Systematic variation of general interconnect wire type

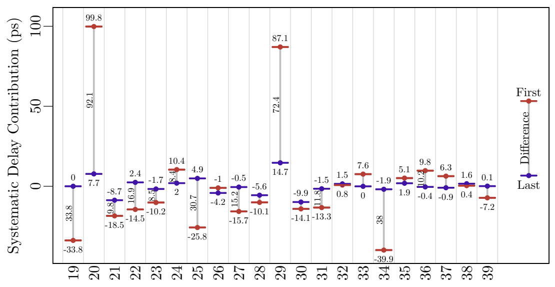
Horizontal and vertical wires differ not only in their delay, but also, horizontal wires come in bundles of 34 wires, with half going left and half going right. Vertical wires, in contrast, are bundled into groups of 24, with a similar split up and down. As such, we separate the systematic contributions of horizontal wires from those of vertical wires.

Figure 5.9 shows $Sys_{SD}(sd)$ for the X and Y coordinate of the wire, as well as the wire index, for both horizontal wires on the left and vertical wires on the right. For both wire types, the Y coordinate has almost no systematic contribution to the DUK delay, as Figures 5.9c and 5.9d show for when this SD is measured last. When measured first, the extremes seem to have significant contribution. The fact that the contribution is non-existent when measured last, however, is indicative that an explanation of this extreme requires deeper understanding of how this SD correlates to others.

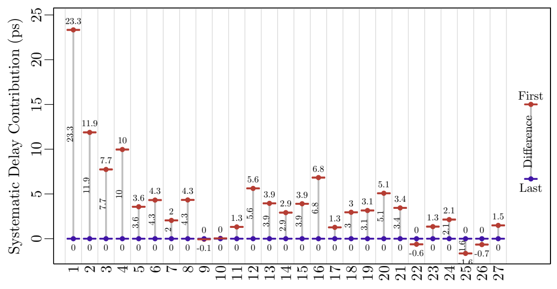
$Sys_{SD_{Xwire}}(sd_{Xwire})$ is perhaps a more interesting function. By convention, wires coordinates refer to the left bottom side of the wire, regardless of the direction of the wire. For horizontal wires, in Figure 5.9a we see that columns 22 through 25 appear to be slower than the rest. Wires that have 22 through 25 as their X coordinate cross over the embedded column of memories at coordinate 25. This increase in delay likely indicates that the embedded memories are physically wider than LABs [92]. Column 33 is also an embedded column, however, it is multipliers instead of memories. For the wires near this coordinate, we do not see as clear a slowdown as with the memories. Better knowledge of the architecture of the FPGA would help explain the effects seen in Figure 5.9a. The systematic contribution of the X coordinate for vertical wires is more easily analyzed (Figure 5.9b). Regardless of when this SD is measured, columns 20 and 29 are systematically slower than the rest.



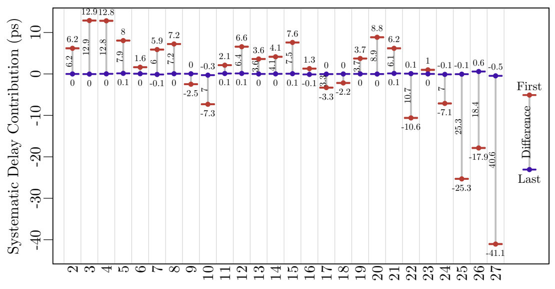
(a) Horizontal (R4) Wire X Left Coordinate



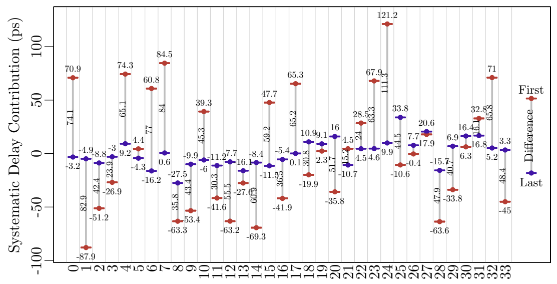
(b) Vertical (C4) Wire X Left Coordinate



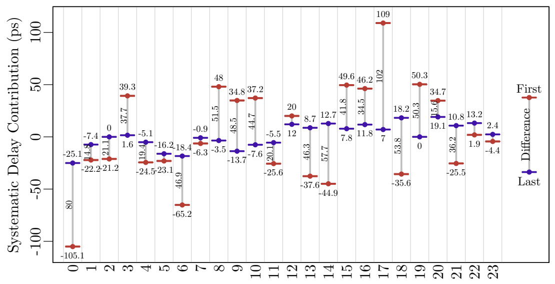
(c) Horizontal (R4) Wire Y Bottom Coordinate



(d) Vertical (C4) Wire Y Bottom Coordinate



(e) Horizontal (R4) Wire Index



(f) Vertical (C4) Wire Index

Figure 5.9: Systematic variation of the General Interconnect

To conclude this section, we look at how the wire index contributes to the DUK delay (Figures 5.9e and 5.9f). As Section 4.1 explained, connections between wires and between wires and LABs are depopulated. This means that a given wire index can only connect to a small number of other physical resources. As with the internal LAB connections, where LLTs and LITs can only connect to two out of the four LUT inputs (Figure 4.3), the limited connectivity of the wire ultimately leads to similarly only being able to reach two out of the four LUT inputs. Since DUKs always end by going through a LUT to a register, the systematic delay contribution of the wire index will similarly mask the contribution of the LUT input, as we saw in Figure 5.6. Therefore, we focus instead on the contribution when measured last. For horizontal wires, indices 0 through 16 are wires that go right, while the rest go left. For vertical wires, indices 0 through 11 go up, the rest, down. Looking at the two figures, we see that lower index wires, i.e., wires that go right or up, are, on average, 20 ps faster than higher index wires, those that go left or down.

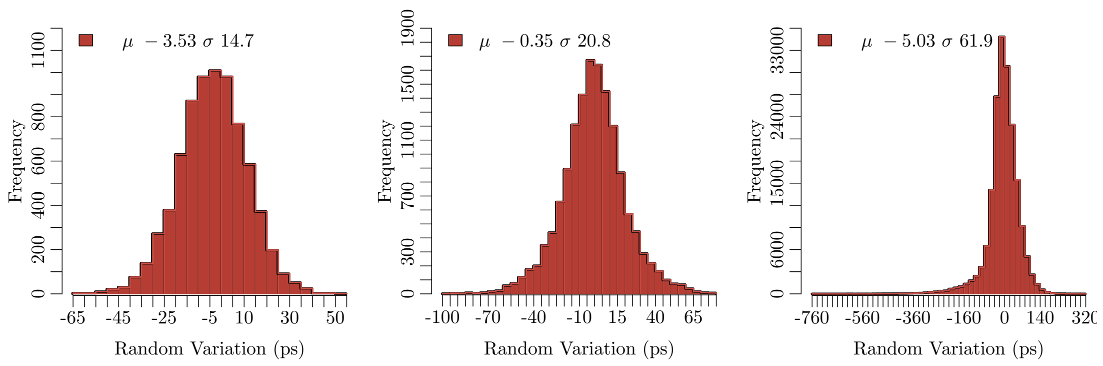
5.5 Random Variation

Once we have $Sys(sd_1, \ldots, sd_n)$, we can now solve for *Rand*, the random variation, as Equation 5.8 demonstrates. We expect *Rand* to have a mean at or extremely close to 1, since any other value would indicate a mean systematic shift in the DUK delay. This mean shift would be correlated variation, and should have been captured by one of the previous four variation decomposition terms.

$$Rand = \frac{\tau_{DUK}}{\tau_0 \ \cdot \ \tau_{FPGA} \ \cdot \ Reg(x,y) \ \cdot \ Sys(sd_1, \ldots, sd_n)} \tag{5.8}$$

Figure 5.10 shows the random variation for three representative DUKs. The Intra-LAB M-DUK, internal to the LAB (Figure 5.10a), the LAB-to-LAB M-DUK, using connections between LABs (Figure 5.10b), and the General Interconnect C-DUK, representing general interconnect resources (Figure 5.10c). In order to move it from a multiplier to a more meaningful delay, we multiply the variation by $\tau_0 \times \tau_{FPGA}$ and center it by subtracting the same value, $\tau_0 \times \tau_{FPGA}$. All three are centered close to 0, as expected. More interesting is the standard deviation of the three figures, showing that the general interconnect resources experience the most random variation, followed by LAB-to-LAB connections. The LABs themselves appear to be less affected by random variation. This observation is consistent with the standard deviations from the raw DUK delays in Section 4.6.

A powerful approach to mitigate random variation is to perform a component-specific mapping [63]. The fact that the general interconnect is most affected by random variation indicates that a



(a) Intra-LAB M-DUK (b) LAB-to-LAB M-DUK (c) General Interconnect C-DUK

Figure 5.10: Random Variation

component-specific mapping solution will see more benefit if it first focuses on routing by carefully selecting the general interconnect routing resources, and worries less about the internals of the logic clusters.

Chapter 6

Future Work

6.1 Measurement Control and Runtime

When measuring a path on the FPGA, we are very careful to isolate each path measurement circuit from others, and from the control circuitry. We also make sure to minimize the activity on the FPGA during a measurement. This leads to consistent measurements (Section 4.12) which primarily capture process variation and not environmental or operational variation. From the tests in Section 4.12, it is clear that control is necessary for consistency, however, when in use, FPGAs have multiple signals, in close proximity, activating at the same time. To make Timing Extraction even more relevant, it is necessary to understand how activity near a measured path affects its delay. Activity near signals will also generate heat. As Section A.1.1 explains, heat will change the delay of the path. Thus, it is also necessary to understand how environmental effects, such as heat, affect the path delays.

In order to explore these effects, controlled circuitry, placed near and around a path measurement circuit, can simulate both activity and self-heating effects. Changing the proximity and activity of these circuits will reveal how these effects affect circuit delay.

Related to the measurement control is the time it takes to measure a path. Currently, we do a linear sweep of the frequency, incrementing by 1.6 ps each step. At each frequency we measure 2^{15}, and only enable one of the measurement circuits in the bitstream at a time. The reason for this is to get correct, consistent measurements. As Section 4.11 indicates, it takes, on average, three seconds to run through the measurement process. Currently, the measurement circuit explores, on average, 200 frequencies per bitstream. This can be significantly reduced by incrementing the frequency following a multiple value binary search algorithm instead. Furthermore, understanding how activity near measurement circuits affects the measured delay would potentially allow multiple path measurement circuits to run at the same time. Finally, testing how the number of measurements at each frequency affects the quality of the results, may reveal that measuring 2^{15} times is more than what is necessary for actuate measurements. Thus, instead of measuring 200 frequencies 2^{15} times for each of the 10 measurement circuits per bitstreams, for a total of 98,304,000 measurements per bitstream, Timing Extraction could potentially reduce that by simultaneously enabling all 10 measurement circuits, measuring $10 \times \log_2(200)$ frequencies 2^8 times, for a total of 20,480 measurements per bitstream, or more than four orders of magnitude speedup.

This speedup only affects the time it takes to measure paths, it does not change how long it takes to program a bitstream or read the results. There are potential improvements here as well. The BeMicro FPGA board we use connects to the host computer through USB. A faster connection can be achieved if the FPGA sits on a faster bus, such as the PCIe bus [69], or even, on a processor socket. This is bound to have an improvement on both the time it takes to program the FPGA, as well as the time it takes to read out results. However, it is possible that through partial reconfiguration [53], where one part of the FPGA is being reconfigured while another part runs, both bitstream loading and reading time can be improved by pipelining loading new measurement circuits with measuring paths, and reading results. This, again, requires detailed understanding of how significantly more activity on the FPGA affects the measurements themselves. Regardless of which of these approaches is used, it is clear that the time it takes Timing Extraction to measure a path is not an inherent limitation of Timing Extraction, but rather, a function of the FPGA board being measured.

6.2 Beyond Logic Blocks and General Interconnect

Modern FPGAs are highly complex heterogeneous circuits. Logic blocks and the general interconnect, though a significant part of the FPGA, represent only a subset of the circuits in the FPGA. Logic blocks contain carry chains, connecting LUTs together through circuitry that aptly allow us to implement carry-based operation, such as adders. Registers are also connected to each other in a register chain, enabling logic such as shift registers or other designs fitting this structure. Moreover, the General interconnect is a hierarchical routing network, where logic blocks connect to short wires, which in turn, connect to longer wires. Beyond logic blocks and general interconnect, FPGAs have embedded memories and digital signal processing (DSP) blocks; some even have full processors embedded within them [13, 97]. Finally, a significant portion of the FPGA real estate is dedicated to complex input and output (I/O) ports implementing multiple communication specifications [11].

Although we did not characterize these structures, as Section 4.8 aims to demonstrate, given the appropriate physical resource graph, Timing Extraction can decompose the circuit into DUKs, and decide which paths should be measured to compute the DUK delays. Some of these structures may require external support. For example, in order to characterize the I/O, it would be necessary to connect another circuit, such as a second FPGA, to the I/O ports, and include portions of this second circuit in the physical resources graph. Multi-FPGA circuits are not uncommon. For example, Microsoft's BEE3 platform [64] consists of 4 FPGAs communicating through ring interconnect. Timing Extraction can easily measure the connections between these FPGAs if it is given a physical resources graph that models these external connections. Essentially, growing the physical resource graph to include components beyond one FPGA. Also, due to the analog nature of some of the I/O ports, it may be necessary to treat certain regions of the circuit as a black box in the physical resources graph. Nevertheless, if the FPGA CAD tools allow for detailed placement and routing of path measurement circuits that encompass these structures, we believe it is possible to characterize many, if not all structures inside the FPGA.

6.3 Beyond the Cyclone III and Cyclone IV FPGAs

The Cyclone III and Cyclone IV FPGAs are low power FPGAs, built on well established, older 65 nm or 60 nm technology. Both these aspects imply that the variation in the FPGA, especially the random variation, is small. Newer, high performance FPGAs, built on cutting-edge technology such as 22 nm or even 14 nm nodes, are bound to exhibit greater variation. This is both because of the increased variation expected with technology scaling (Figure 2.3), and because a high performance process has a lower threshold voltage, V_{th}. Applying Timing Extraction to these FPGAs should be very revealing, quantifying greater random variation, and better demonstrating the necessity for a component-specific mapping CAD approach.

As long as we have a physical resources graph of the FPGA, the general algorithms in Timing Extraction are agnostic to the FPGA being measured. They can directly extract DUKs, and determine which paths to measure. Having the physical resource graph also lets us determine which paths to pack together. However, generating placement and routing constraints is FPGA model specific. Different vendors provide different ways to specify these constraints, and even the syntax to specify these constraints varies from one FPGA model to another, for a given vendor. Compiling and running the bitstreams will also be FPGA specific, since different vendors use different CAD tools. Also, different FPGAs and FPGA boards provide different clocks, requiring that we compute a unique set of PLL control values for each FPGA model. These requirements place a barrier of entry to measuring an arbitrary FPGA. Nevertheless, the knowledge gained from measuring newer FPGAs, both from Altera and other vendors, would be highly valuable.

6.4 Improved Variation Analysis

Chapter 5 presents an approach to decompose the measured DUK variation into regional, systematic, and random variation. It uses a simple model of the delay to perform this decomposition. Even with this simple model, it is able to do a meaningful decomposition, discovering several aspects of the variation within the Cyclone III FPGA. Nevertheless, this is just the beginning of the type of analysis that will yield very useful information for both FPGA manufacturers and users. Several aspects will improve the decomposition.

Primarily, a better understanding of the FPGA architecture will allow the separation of design variation from process variation. As a simple example, knowing that a particular wire index in the FPGA is slower due to design variation or process variation has very different implications. As a user, in both cases we may choose to avoid that wire. As a manufacturer, if it is slow by design, it is likely the CAD tools know of this and adjust accordingly. Moreover, the decomposition analysis can take this into account and adjust the wire delay to find the true process variation of the wire. On the other hand, if it is previously unknown process variation, it is possible that a design adjustment may mitigate this. Otherwise, the CAD tool timing models could be modified to account for the delay difference [89].

Another improvement lies in the delay model. DUKs are composed of multiple physical resources, the delay model we use is only an approximation of the delay. Better matching the delay model to the individual structure of the different DUKs would allow for a delay decomposition below the DUK level. For example, if the delay of a DUK is modeled as the addition of the delay of a resources wire and a LUT, it may be possible to better separate the variation from the two components.

Finally, as we point out in Section 5.4, the decomposition is only as smart as the parameters we feed it. That is to say, if we fail to consider correlation to some parameter, it is unlikely that our decomposition will discover this correlation on its own. For example, as wires traverse the FPGA, it is possible that their physical position in the wire bundle may shift, while its logical wire index remains fixed. Thus a wire with index 9 in one region of the FPGA may share more characteristics with a wire with index 4 in another region of the FPGA, instead of the one with index 9. Our current decomposition is unable to detect this kind of correlation. A more complex decomposition that discovers new correlations would be required. Machine learning algorithms may be appropriate for these kinds of discoveries.

Chapter 7

Conclusions

Timing Extraction decomposes an FPGA into individual Discrete Units of Knowledge (DUKs). Using only resources within the FPGA, it measures the delay of a minimal set of paths, and uses their delays to compute the delay of each DUK. DUKs can then be composed to compute the delay of any path in the FPGA between two registers. A simple modification to the circuit graph allows conventional routing algorithms to perform component-specific mappings, where logic is custom mapped to best fit a given FPGA, using the computed DUK delays. It also provides the measurements needed to analyze the variation in the FPGA, and decompose it into correlated variation and random variation.

We implemented Timing Extraction for the Altera Cyclone III 65 nm FPGA, and applied it to 18 FPGAs. For each, decomposing logic blocks and the general interconnect network into 1,356,182 individual DUKs. This required running 232,250 bitstreams on each FPGA, to measure the 2,736,556 necessary paths. Our decomposition revealed ten different DUK types, each representing a different structure of the FPGA. Together these DUKs allow us to compose any path between two registers.

We demonstrate how Timing Extraction is agnostic to the structure being decomposed, and can well handle large resources as black boxes, or given their structure, can decompose the circuit further. We explore the effects of lowering the supply voltage, a common approach for energy reduction. Our results confirm the modeling equations by showing that delay and variation increase as supply voltage decreases. We also compare our measured results to the timing models shipped with the manufacturers CAD tools. Though aware of some variation between resources, our measurements reveal variations not modeled by the CAD tools. This is due, in part, to the existence of random variation in the FPGA. Lastly, we demonstrate that we can repeatedly measure and compute DUK delays, and consistently reach the same results.

We conclude this work by decomposing the DUK delays into regional variation, correlated to the physical coordinates of a DUK, systematic variation, correlated to a descriptive parameter of the DUK, and uncorrelated random variation. Our analysis reveals a distinct regional delay gradient where, over a range of 50 ps, slow resources on the top left give way to faster resources on the bottom right. The systematic correlations demonstrate differences such as horizontal wires being, on average, at least 30 ps faster than vertical wires. It also reveals a directional preference where wires going right or up are, on average, 20 ps faster, compared to wires going left or down. Finally, our random variation analysis clearly demonstrates that general interconnect resources encounter grater random variation than logic blocks. With general interconnect experiencing random variation on the order of $\sigma = 62$ ps, while logic blocks only have a $\sigma = 15$ ps.

Appendices

Appendix A

Operational, Environmental, and Aging Effects

A.1 Operational and Environmental Variation

Process variation, in a sense, can be thought of as static variation. By that we mean that once the device is manufactured, the parameters that define the device will not change.[1] To contrast this, environmental and operational variation is dynamic, shifting the performance and energy requirement of a device depending on how and where the device is used. For example, the delay of a transistor will be larger when operating under high temperatures.

It is crucial to understand how a circuit will change as it is running if we are to develop solutions that can deal with more than just process variation. In particular, we focus on three types of dynamic variation: temperature variation, crosstalk and power supply variation.

A.1.1 Temperature

During the operation of a circuit, wires and transistors will experience fluctuations in temperature. The temperature of a device is the sum of the ambient temperature and the temperature rise caused by power dissipation in the package, known as self-heating.

$$T = T_{env} + T_{op}$$

Ambient temperature affects a whole chip and can range from 0 °C to 85 °C for commercial applications and even more extreme, from -40 °C to 125 °C for military or industrial applications, depending on where the circuit is deployed. Self-heating occurs when current flows through a

[1]This ignores the effects of aging on device, aging is covered in Section A.2.

device, dissipating some of its energy as heat. It is a function of the computation being done and, therefore, the temperature change caused by it will vary from device to device as well as over time. Moreover, due to the poor heat conductivity of the electrical insulator used, heat dissipation occurs over milliseconds, affecting an area on the order of millimeters.

Temperature has a significant impact on the electronic properties of a wire. Although the capacitance is independent of temperature, the wire resistance at temperature T, $R(T)$, increases linearly from the reference resistance $R(T_0)$

$$R(T) = R(T_0)(1 + \alpha \cdot (T - T_0)) \tag{A.1}$$

The temperature coefficient, α, of copper is about 0.004/K [91]. The reference temperature, T_0, is often 300 K. Furthermore, since the material surrounding the interconnect is not only an electrical insulator but also a thermal insulator, the heat does not quickly dissipate.

The transistor current also suffers from temperature changes. Both threshold voltage, V_{th}, and carrier mobility, μ, decrease with temperature as defined by Equations A.4 and A.3 respectively.

$$V_{th}(T) = V_{th}(T_0) - k_{V_{th}} \cdot (T - T_0) \tag{A.2}$$

$$\mu(T) = \mu(T_0) \left(\frac{T}{T_0} \right)^{-k_\mu} \tag{A.3}$$

The fitting parameters have been found to be 0.8 mV/K for $k_{V_{th}}$ and 1.5 for k_μ[95]. Moreover, v_{sat} also decreases with temperature, dropping 20% from 300 K to 400 K [91]. Finally v_T, the thermal voltage, as defined in thermodynamics, is proportional to temperature and is written in terms of Boltzmann's constant, k, and the electron charge, q.

$$v_T = \frac{kT}{q} \tag{A.4}$$

As with wires, the insulator surrounding transistors is a poor heat conductor, trapping the heat on the transistor. This is significantly worse for silicon on insulator (SOI) transistors where an extra insulating layer increases the temperature by 10% to 15% [41].

Taking these temperature effects together, it is possible to see from Equation 2.6 that the subthreshold current increases exponentially with a rise in temperature since V_{th} decreases with temperature and the subthreshold slope is directly proportional to temperature.

This is not the case for the saturation current $I_{ds,sat}$. In general, $I_{ds,sat}$ decreases with temperature, however, since the ratio of supply voltage to threshold voltage is decreasing with scaling,

[46] shows that an inversion occurs where the saturation current increases with temperature.

These sustained high temperatures have an immediate effect on the characteristics of a circuit. Moreover, the common silicon substrate causes heat generated at one point to spread and cause a temperature rise at nearby points within the chip, further affecting other wires and transistors. Finally, temperature has a long-term fatal effect on devices, causing increased aging, which leads to shorter circuit lifetimes, as explained in Section A.2.

A.1.2 Power Supply

Standard CMOS design requires that both power (V_{dd}) and ground (GND) be routed to every single gate. The increase in active devices, drawing current from the power supply at different points in space and time, lead to deviations of V_{dd} and GND from their nominal values. In fact, it is not uncommon for the power supply to fluctuate by $\pm 10\%$ [17] and variations as high as 18% have been recorded [68]. Furthermore, clock-gating and power-gating techniques drastically change how many devices draw current from the power supply at one time. The consequence of a drop in voltage supplied to the transistor is a similar drop in performance. Two dominant sources of power supply variation exist, IR-Drop, due to parasitic impedance of the power distribution network, and $L\frac{\mathrm{d}I}{\mathrm{d}t}$, from inductive coupling.

IR-Drop has both an instantaneous and an average effect. Circuits will draw current based on the computation being performed. Close to the clock edge, the current drawn can locally spike due to the many devices switching simultaneously. This spike can be partially managed by well placed bypass capacitors that provide the extra current necessary. Nevertheless, the average current will vary over time, and with it, the voltage supplied to each gate.

Inductance noise occurs across the power supply as the current rapidly changes. Components that are idle for some cycles and then begin to switch are especially susceptible to this noise.

Although power supply variations are deterministic, subject to the activity on the circuit, the sheer number of variables needed to model this variation makes the problem intractable. Therefore, it is usually considered to be a source of random operational variation.

A.1.3 Crosstalk

Continued scaling leading to denser interconnect has elevated crosstalk from an insignificant phenomena to a serious source of variation. Wires in modern ICs experience capacitive, and to a lesser extent, inductive coupling to adjacent wires as well as to the layer above and below. Therefore, a transitioning wire will affect its neighbors. This influence is known as crosstalk. Crosstalk can induce a pulse on an adjacent wire with an otherwise fixed signal, or, if the adjacent wire is also transitioning, it can delay or speed up that transition, depending on the direction of the transition [34].

Crosstalk happens on the time frame of a cycle and is dependent on the activity in the circuit. As with power supply variation, though it is theoretically possible to model and account for all crosstalk effects, this is only practical for small circuits. It is, however, possible to ameliorate the effects of crosstalk by laying out interconnect in such a way that no two long wires run parallel for their full length, or, interleaving signal wires with power and ground wires [39]. Nevertheless, crosstalk effects induce a high frequency random variation on modern chips that directly affects the delay of the circuit.

A.2 Aging Effects

Over the lifetime of a circuit, individual devices will wear out. In transistors, this leads to a shift in threshold voltage. An increase in resistance is the deterioration for wires. For both, it translates into slower and less efficient devices. The end result, however, is a fault which can lead to a defective circuit. The mean time to failure is, in part, determined by process variation and, in part, by the environment and activity factor [84].

Aging effects are different for wires and for transistors. Electromigration is the predominant failure mechanism for wires. Transistors degrade in a more complex way, through hot carrier injection, negative bias temperature instability (NBTI) and time dependent dielectric breakdown (TDDB).

A.2.1 Electromigration

Electromigration is the displacement of atoms due to the movement of electrons through a conductor under an electric field. Current flowing through a wire creates a flow of electrons in the opposite direction. In turn, this causes the atoms to be displaced. If enough atoms are displaced, a void in the wire appears, increasing the resistance of the wire. Eventually, the void grows large enough to cause an open fault on the wire. What's more, the atoms that migrated, pile on one end of the void and can form bridging faults with neighboring wires. To make matters worse, as voids appear, self-heating in the wire increases, which, in turn, accelerates the process of electromigration. Also, the geometry of the wire determines the rate at which electromigration occurs. In particular, line edge roughness (Section 2.3.1.2) increases the susceptibility of a wire to electronmigration. Formally, Black [18] shows how the mean time to failure (MTTF) due to electromigration relates to the temperature, T, current density, J, and geometric properties, A of a wire as well as the activation energy E_a.

$$MTTF_{EM} = \frac{A \cdot e^{\left(\frac{E_a}{k_b \cdot T}\right)}}{J^{1.2}} \tag{A.5}$$

Current technologies use strict design rules to reduce the effects of electromigration, nevertheless functional and parametric failures due to electromigration are expected to get worse as wire cross-sections shrink [81].

A.2.2 Hot carrier injection

In the presence of a strong electric field, electrons and holes in the channel region of a transistor gain kinetic energy. When the energy reaches a critical level, they can overcome the barrier between the oxide and the channel region and lodge in the oxide. These hot carriers form interface traps which degrade the performance of the transistor.

Although both p-type and n-type transistors are affected by hot carriers, p-type devices suffer less from this aging effect because the voltage required for hot carriers to become interface traps is higher than for n-type devices.

Interface traps corrupt a transistor in two ways [78]. First, the threshold voltage, V_{th} is affected. Second, carrier mobility is reduced. Together, these effects debase the transistor. However, the effect is not abrupt, rather, it happens over a period of time. Nevertheless, once enough traps exist, the transistor will fail to operate correct. In general, the mean time to failure (MTTF) due to hot carriers can be modeled as in [1]:

$$MTTF_{HC} = \frac{A \cdot e^{\left(\frac{E_a}{k_b \cdot T}\right)}}{I_{substrate}} \tag{A.6}$$

Where A is a function of the transistor parameters, such as dimensions and dopant concentration, $I_{substrate}$ is the substrate current induced by the hot carriers, T is temperature, E_a, the activation energy, and k_b, Boltzmann's constant.

A.2.3 Time-Dependant Dielectric Breakdown

Time-dependent dielectric breakdown (TDDB) results when the oxide separating the gate from the channel in a transistor breaks down, creating a path from the gate to the channel. Similarly, it applies to the breakdown of oxide separating two metal layers. TDDB occurs when a high voltage difference exist across the oxide. Over time, charge carriers get trapped in the oxide, as is the case with hot carriers explained above. Eventually, when enough carriers are trapped, a conductive path through the oxide will be created leading to shorts between interconnect wires and failures in transistors.

Once a small conductive path exists, self-heating will exacerbate the creation of trapped charge, leading to a positive feedback look where the oxide breakdown accelerates. For a thin dielectric ($\leq$ 4 nm) separating two metal layers, or the gate and channel in a transistor (t_{ox}), [1] models MTTF as:

$$MTTF_{TDDB} = A \cdot e^{\left(\frac{E_a}{k_b \cdot T} - B_{ox} V\right)} \tag{A.7}$$

Where, again, A is a function of the transistor parameters, B_{ox} is a voltage acceleration constant that depends on oxide characteristics, V is the voltage applied, T is temperature, E_a, the activation energy, and k_b, Boltzmann's constant.

A.2.4 Negative Bias Temperature Instability

Negative bias temperature instability (NBTI) is an aging effect that affects p-type devices. As the name implies, the negative effects of NBTI develop when the transistor is negatively biased, i.e. the gate-source voltage $V_{gs} = -V_{dd}$ and the source-drain voltage $V_{sd} = 0$. The effect is highly dependent on temperature, worsening as temperature increases.

As with the previous two aging effects, NBTI's failure model involves interface traps, in particular, at the silicon-oxide barrier. Due to the mismatch between Si and SiO_2 crystal lattice, many free Si bonds exist on the boundary. Hydrogen is bonded to these free atoms to prevent electrical interference. However, in the presence of hole carriers, the Si-H bond is broken, generating interface traps. The reaction-diffusion model [42, 4] explains the process as occurring in two phases. First, holes in the channel dislodge hydrogen atoms during the reaction phase. In the diffusion stage the hydrogen atoms diffuse through the oxide, leaving free Si bonds at the barrier. The rate at which this process produces interface traps is explained in [4]. Of note is the fact that the reaction rate increases significantly at higher temperatures since it allows holes to more easily break the hydrogen silicon bond.

The increase in interface traps changes the threshold voltage of the transistor. Equation A.8 shows how V_{th} changes over time as a function of the rate of trap generation, N_{it} [47].

$$\Delta V_{th}(t) = (\mu + 1)\frac{q \cdot \Delta N_{it}(t)}{C_{ox}} \tag{A.8}$$

Where q is the charge in the channel, C_{ox} is the oxide capacitance as previously defined, and μ, the charge mobility. It is worth noting that NBTI also has an effect on μ.

Threshold voltage degradation due to NBTI is similar to hot carrier injection, however whereas hot carrier injection occurs when a devices is switching, NBTI happens when it is not switching. This is particularly damaging for designs that use power gating to reduce power consumption. Moreover, unlike other aging failure modes, NBTI is reversible to an extent. When the negative bias is removed from the device, the free hydrogen atoms can anneal with the free silicon atom reducing the total number of free silicon bonds [57]. As with other sources of aging variation, NBTI is more easily modeled as a random source.

Bibliography

[1] Failure Mechanisms and Models for Semiconductor Devices. *JEDEC Solid State Technology Association* (2011). A.2.2, A.2.3

[2] International technology roadmap for semiconductors. <http://www.itrs.net/Links/2011ITRS/2011Chapters/2011ExecSum.pdf>, 2011. 2.3.1.2, 2.3.1.2

[3] International technology roadmap for semiconductors. <http://www.itrs.net/Links/2011ITRS/Home2011.htm>, 2011. 2.3.1.3

[4] Alam, M., Kufluoglu, H., Varghese, D., and Mahapatra, S. A comprehensive model for PMOS NBTI degradation: Recent progress. *Microelectronics Reliability 47*, 6 (June 2007), 853–862. A.2.4

[5] Alon, E., Stojanovic, V., and Horowitz, M. Circuits and techniques for high-resolution measurement of on-chip power supply noise. *IEEE Journal of Solid-State Circuits 40*, 4 (Apr. 2005), 820–828.

[6] Altera. Quartus II University Interface Program. http://www.altera.com/education/univ/research/quip/unv-quip.html. (document), 4.4.4, 4.6.1, 4.58

[7] Altera. Reliability Report 57 1H 2014. http://www.altera.com/literature/rr/rr.pdf. 2.5.2

[8] Altera. Cyclone III Device Handbook Volume I. http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf, 2011. (document), 4.1, 4.1, 4.4.2

[9] Altera. Quartus II Handbook Version 11.1. http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf, 2011. 4.4.4

[10] Altera. Stratix IV Device Handbook. http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf, 2011. 2.1.2

[11] Altera. Stratix V Device Handbook. http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf, 2012. 6.2

[12] Altera. Cyclone IV Device Handbook Volume I. http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf, 2014. 4.9

[13] Altera. Cyclone V SoC Hard Processor System. `http://www.altera.com/devices/fpga/cyclone-v-fpgas/hard-processor-system/cyv-soc-hps.html`, 2014. 6.2

[14] Arrow. BeMicro FPGA Evaluation Kit. http://www.arrownac.com/offers/altera-corporation/bemicro/. 4.1

[15] Bayerl, A., Lanza, M., Porti, M., Nafria, M., Aymerich, X., Campabadal, F., and Benstetter, G. Nanoscale and Device Level Gate Conduction Variability of High-k Dielectrics-Based Metal-Oxide-Semiconductor Structures. *IEEE Transactions on Device and Materials Reliability 11*, 3 (Sept. 2011), 495–501. 2.3.1.2

[16] Bellows, P., and Hutchings, B. JHDL — an HDL for reconfigurable systems. In *IEEE Symposium on FPGAs for Custom Computing Machines* (Los Alamitos, CA, 1998), K. L. Pocek and J. Arnold, Eds., IEEE Computer Society Press, pp. 175–184. 2.1.1

[17] Bernstein, K., Frank, D. J., Gattiker, A. E., Haensch, W., Ji, B. L., Nassif, S. R., Nowak, E. J., Pearson, D. J., and Rohrer, N. J. High-performance CMOS variability in the 65-nm regime and beyond. *IBM J. Res. and Dev. 50*, 4/5 (July/September 2006), 433–449. 2.2.1, A.1.2

[18] Black, J. R. Electromigration - A brief survey and some recent results. *IEEE Transactions on Electron Devices 16*, 4 (Apr. 1969), 338–347. A.2.1

[19] Borkar, S. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro 25*, 6 (Nov. 2005), 10–16. 2.5.1

[20] Cho, M., Maitra, K., and Mukhopadhyay, S. Analysis of the impact of interfacial oxide thickness variation on metal-gate high-K circuits. In *2008 IEEE Custom Integrated Circuits Conference* (Sept. 2008), IEEE, pp. 285–288. 2.3.1.2

[21] Chow, C. T., Tsui, L. S. M., Leong, P. H. W., Luk, W., and Wilton, S. J. Dynamic voltage scaling for commercial FPGAs. *ICFPT* (December 2005), 173–180. 2.5.2

[22] Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms*. MIT Press, 1990. 3.4.7

[23] Darwin, C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859. 4.6

[24] Dijkstra, E. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (1959), 269–271. 2

[25] Dusa, M., Finders, J., and Hsu, S. Double Patterning Lithography: The Bridge Between Low k1 ArF and EUV, 2008. 2.3.1.1

[26] Franco, J. J. L., Boemo, E., Castillo, E., and Parrilla, L. Ring oscillators as thermal sensors in FPGAs: Experiments in low voltage. In *2010 VI Southern Programmable Logic Conference (SPL)* (2010), IEEE, pp. 133–137. 2.5.1

[27] French, R. H., and Tran, H. V. Immersion Lithography: Photomask and Wafer-Level Materials. *Annual Review of Materials Research 39*, 1 (Aug. 2009), 93–126. 2.3.1.1

[28] Gojman, B., and DeHon, A. VMATCH: Using Logical Variation to Counteract Physical Variation in Bottom-Up, Nanoscale Systems. In *ICFPT* (December 2009), IEEE, pp. 78–87. 1.3

[29] Gojman, B., and DeHon, A. GROK-INT: Generating real on-chip knowledge for interconnect delays using timing extraction. In *FCCM* (2014), pp. 88–95.

[30] Gojman, B., Mehta, N., Rubin, R., and DeHon, A. Component-specific mapping for low-power operation in the presence of variation and aging. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*. Springer, 2011, ch. 12, pp. 381–432. 1.3

[31] Gojman, B., Nalmela, S., Mehta, N., Howarth, N., and DeHon, A. GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays using Timing Extraction. In *FPGA* (2013), pp. 81–90. 4.4.2

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

[33] Guan, Z., Wong, J. S. J., Chaudhuri, S., Constantinides, G., and Cheung, P. Y. K. Exploiting stochastic delay variability on FPGAs with adaptive partial rerouting. In *ICFPT* (2013), pp. 254–261. 1.3

[34] Gupta, S., and Breuer, M. Analytic models for crosstalk delay and pulse analysis under non-ideal inputs. In *Proceedings International Test Conference* (1997), Int. Test Conference, pp. 809–818. A.1.3

[35] Hanson, S., Zhai, B., Bernstein, K., Blaauw, D., Bryant, A., Chang, L., Das, K. K., Haensch, W., Nowak, E. J., and Sylvester, D. M. Ultralow-voltage, minimum-energy CMOS. *IBM J. Res. and Dev. 50*, 4–5 (July/September 2006), 469–490. 2.2.2

[36] Hauck, S., and DeHon, A., Eds. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Systems-on-Silicon. Elsevier, 2008. 2.1.2

[37] Heinlein, R. *Stranger in a Strange Land*. Remembering Tomorrow. Ace Books, 1987. 4.14

[38] Hirose, Y., Hashikawa, N., Fukumoto, K., and Mashiko, Y. Microsampling technique for EBSP inspection on the cross-sections of copper trench lines in ULSIs. *Journal of Electron Microscopy 53*, 5 (2004), 567–570. 2.3.1.1

[39] Horowitz, M. Managing wire scaling: a circuit perspective. In *Proceedings of the IEEE 2003 International Interconnect Technology Conference (Cat. No.03TH8695)* (2003), IEEE, pp. 177–179. A.1.3

[40] Huang, L., and Xu, Q. Economic Analysis of Testing Homogeneous Manycore Chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29*, 8 (Aug. 2010), 1257–1270. 2.5.1

[41] Jenkins, K., and Franch, R. Impact of self-heating on digital SOI and strained-silicon CMOS circuits. In *2003 IEEE International Conference on Robotics and Automation (Cat No 03CH37422) SOI-03)* (2003), IEEE, pp. 161–163. A.1.1

[42] Jeppson, K. O., and Svensson, C. M. Negative bias stress of MOS devices at high electric fields and degradation of MNOS devices. *Journal of Applied Physics 48*, 5 (1977), 2004. A.2.4

[43] Kahng, A. B. A. A., Member, S. S. S. S., and Samadi, K. CMP Fill Synthesis: A Survey of Recent Studies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27*, 1 (Jan. 2008), 3–19. 2.3.1.2

[44] Krishnan, G. Flexibility with EasyPath FPGAs. *Xcell Journal 0*, 4 (2005), 96–98. 2.5.1

[45] Kuhn, K., Kenyon, C., Kornfeld, A., Liu, M., Maheshwari, A., Shih, W.-k., Sivakumar, S., Taylor, G., VanDerVoorn, P., and Zawadzki, K. Managing Pro-

cess Variation in Intels 45nm CMOS Technology. *Intel Technology Journal 12*, 02 (2008). 2.3.1.2, 2.3.1.3

[46] Kumar, R., and Kursun, V. Reversed Temperature-Dependent Propagation Delay Characteristics in Nanometer CMOS Circuits. *IEEE Transactions on Circuits and Systems II: Express Briefs 53*, 10 (Oct. 2006), 1078–1082. 2.5.1, A.1.1

[47] Kundu, S., and Sreedhar, A. *Nanoscale CMOS VLSI Circuits: Design for Manufacturability*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, Apr. 2010. 2.3.1.3, A.2.4

[48] Kundu, S., Sreedhar, A., and Sanyal, A. Forbidden pitches in sub-wavelength lithography and their implications on design. *Journal of Computer-Aided Materials Design 14*, 1 (Feb. 2007), 79–89. 2.3.1.2

[49] Lee, B. H., Oh, J., Tseng, H. H., Jammy, R., and Huff, H. Gate stack technology for nanoscale devices. *Materials Today 9*, 6 (June 2006), 32–40. 2.3.1.2

[50] Lemieux, G., and Lewis, D. Using sparse crossbars within LUT. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays - FPGA '01* (New York, New York, USA, Feb. 2001), ACM Press, pp. 59–68. 2.1.1

[51] Levine, J. M., Stott, E., and Cheung, P. Y. Dynamic voltage &#38; frequency scaling with online slack measurement. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays* (New York, NY, USA, 2014), FPGA '14, ACM, pp. 65–74. 2.5.2

[52] Lewis, D., Betz, V., Jefferson, D., Lee, A., Lane, C., Leventis, P., Marquardt, S., McClintock, C., Pedersen, B., Powell, G., Reddy, S., Wysocki, C., Cliff, R., and Rose, J. The Stratix routing and logic architecture. In *FPGA* (2003), pp. 12–20. 2.1.2

[53] Lim, D., and Peattie, M. *Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations*. Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, May 2002. XAPP 290 <http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf>. 6.1

[54] Lin, Y., Hutton, M., and He, L. Placement and Timing for FPGAs Considering Variations. In *2006 International Conference on Field Programmable Logic and Applications* (2006), IEEE, pp. 1–7. (document), 2.4

[55] Ling, Z.-M., Cho, J., Wells, R. W., Johnson, C. S., and Davis, S. G. Method of using partially defective programmable logic devices. United States Patent Number: 6,664,808, December 16 2003. 2.5.1

[56] Littlebury, H. W. Method for assembling, testing, and packaging integrated circuits. United States Patent Number:, 4,985,988, Jan. 1991. 2.5.1

[57] Mahapatra, S., Islam, A. E., Deora, S., Maheta, V. D., Joshi, K., Jain, A., and Alam, M. A. A critical re-evaluation of the usefulness of R-D framework in predicting NBTI stress and recovery. In *2011 International Reliability Physics Symposium* (Apr. 2011), IEEE, pp. 6A.3.1–6A.3.10. A.2.4

[58] Majzoobi, M., Dyer, E., Elnably, A., and Koushanfar, F. Rapid FPGA delay characterization using clock synthesis and sparse sampling. In *Proc. Intl. Test Conf.* (2010). 1.3, 3.1

[59] Majzoobi, M., Koushanfar, F., and Potkonjak, M. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Transactions on Reconfigurable Technology and Systems 2*, 1 (Mar. 2009), 1–33. 3.1

[60] Mak, T. Infant Mortality–The Lesser Known Reliability Issue. In *13th IEEE International On-Line Testing Symposium (IOLTS 2007)* (July 2007), IEEE, pp. 122–122. 2.5.1

[61] Masud, M. I., and Wilton, S. A new switch block for segmented FPGAs. In *FPL* (1999), pp. 274–281. 2.1.1

[62] McMurchie, L., and Ebeling, C. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *FPGA* (1995), pp. 111–117. 2

[63] Mehta, N., Rubin, R., and DeHon, A. Limit Study of Energy & Delay Benefits of Component-Specific Routing. In *FPGA* (2012), pp. 97–106. 1.3, 4.9, 5, 5.5

[64] Microsoft. BEE3 (Berkeley Emulation Engine, version 3). `http://research.microsoft.com/en-us/projects/bee3/`. 6.2

[65] Morshed, T. H., Yang, W., Dunga, M. V., Xi, X., He, J., Liu, W., Cao, M., Xiaodong, J., Ou, J. J., Chan, M., Niknejad, A. M., and Hu, C. BSIM4.6.4 MOSFET Model, 2009. 2.3.1.3

[66] Narendra, S. G., and Chandrakasan, A. P. *Leakage in nanometer CMOS technologies.* Springer-Verlag New York Inc, 2006. 2.2.1

[67] Nassif, S., Bernstein, K., Frank, D. J., Gattiker, A., Haensch, W., Ji, B. L., Nowak, E., Pearson, D., and Rohrer, N. J. High Performance CMOS Variability in the 65nm Regime and Beyond. In *2007 IEEE International Electron Devices Meeting* (2007), IEEE, pp. 569–571. 2.3.1.2, 2.3.1.2

[68] Nigussie, E., Plosila, J., and Isoaho, J. Monitoring and reconfiguration techniques for power supply variation tolerant on-chip links. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (May 2010), IEEE, pp. 2892–2895. A.1.2

[69] PCI-SIG. Pcie specifications. https://www.pcisig.com/specifications. 6.1

[70] Rabaey, J. M., Chandrakasan, A. P., and Nikolic, B. *Digital Integrated Circuits*, 2nd ed. Prentice Hall, 1999. 2.2.2

[71] Rearick, J. Too much delay fault coverage is a bad thing. In *Proceedings International Test Conference* (2001), IEEE, pp. 624–633. 2.5.1

[72] Reda, S., and Nassif, S. Analyzing the impact of process variations on parametric measurements: Novel models and applications. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.* (2009), pp. 375–380. 2.3.1.2

[73] Rose, J., and Brown, S. Flexibility of interconnection structures for field-programmable gate arrays. *IEEE Journal of Solid-State Circuits 26*, 3 (March 1991), 277–282. 2.1.1

[74] Rose, J., Luu, J., Yu, C. W., Densmore, O., Goeders, J., Somerville, A., Kent, K. B., Jamieson, P., and Anderson, J. The VTR project. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA '12* (New York, New York, USA, Feb. 2012), ACM Press, p. 77. 2.1.1

[75] Roy, K., Mukhopadhyay, S., and Mahmoodi-Meimand, H. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer {CMOS} Circuits, 2003. 2.2.1

[76] Saha, S. K. Modeling Process Variability in Scaled CMOS Technology. *IEEE Design & Test of Computers 27*, 2 (Mar. 2010), 8–16. 2.3.1.2

[77] Schellenberg, F. A Little Light Magic. *IEEE Spectrum 40*, 9 (2003), 34–39. 2.3.1.1

[78] Schwerin, A., Hansch, W., and Weber, W. The relationship between Oxide charge and device degradation: A comparative study of n- and p- channel MOSFET's. *IEEE Transactions on Electron Devices 34*, 12 (Dec. 1987), 2493–2500. A.2.2

[79] Sedcole, P., Wong, J. S., and Cheung, P. Y. K. Modelling and compensating for clock skew variability in FPGAs. *2008 International Conference on FieldProgrammable Technology* (2008), 217–224. 3.1

[80] Smith, J. R., and Xia, T. X. T. High-Resolution Delay Testing of Interconnect Paths in Field-Programmable Gate Arrays, 2009. 3.1

[81] Srinivasan, J., Adve, S., Bose, P., and Rivers, J. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks, 2004* (2004), IEEE, pp. 177–186. A.2.1

[82] Srivastava, A., Sylvester, D., and Blaauw, D. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Integrated Circuits and Systems. Springer, 2005. 2.5.2

[83] Stackhouse, B., Bhimji, S., Bostak, C., Bradley, D., Cherkauer, B., Desai, J., Francom, E., Gowan, M., Gronowski, P., Krueger, D., Morganti, C., and Troyer, S. A 65 nm 2-Billion Transistor Quad-Core Itanium Processor. *IEEE Journal of Solid State Circuits 44*, 1 (2009), 18–31. 2.3.1.1, 2.5.1

[84] Stott, E. A., Wong, J. S. J., Pete Sedcole, P., and Cheung, P. Y. K. Degradation in FPGAs: measurement and modelling. In *FPGA* (2010), p. 229. A.2

[85] Terasic. http://wiki.ntb.ch/infoportal/_media/fpga/boards/de0_nano/de0-nano-schematic.pdf, 2011. 4.9

[86] Terasic. http://www.terasic.com.tw/cgi-bin/page/archive.pl?\Language=English&No=593, 2012. 4.9

[87] Tiwari, A., and Torrellas, J. Facelift: Hiding and slowing down aging in multicores. *2008 41st IEEEACM International Symposium on Microarchitecture* (2008), 129–140. 2.5.2

[88] Transactions, I., Electron, O. N., Arora, N., Hauser, J., and Roulston, D. Electron and hole mobilities in silicon as a function of concentration and temperature. *IEEE Transactions on Electron Devices 29*, 2 (Feb. 1982), 292–295. 2.3.1.3

[89] Tuan, T., Lesea, A., Kingsley, C., and Trimberger, S. Analysis of within-die process variation in 65nm FPGAs. In *ISQED* (March 2011), pp. 1–5. 1.3, 5.3, 6.4

[90] Wells, R. W., Ling, Z.-M., Patrie, R. D., Tong, V. L., Cho, J., and Toutounchi, S. Application-specific testing methods for programmable logic devices. United States Patent Number: 6,817,006, November 9 2004. 2.5.1

[91] WESTE, N., AND HARRIS, D. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4 ed. Addison-Wesley Publishing Company, USA, Mar. 2010. 2.5.2, A.1.1, A.1.1

[92] WONG, H., BETZ, V., AND ROSE, J. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2011), FPGA '11, ACM, pp. 5–14. 5.4.1, 5.4.1

[93] WONG, J. S., SEDCOLE, P., AND CHEUNG, P. Y. K. Self-measurement of combinatorial circuit delays in FPGAs. *ACM Tr. Reconfig. Tech. and Sys. 2*, 2 (June 2009), 1–22. 1.3, 3.1

[94] WONG, J. S. J., CHEUNG, P. Y. K., AND SEDCOLE, P. Combating process variation on FPGAS with a precise at-speed delay measurement method. *Field Programmable Logic and Applications 2008 FPL* (2008), 703–704. 3.1

[95] XIAOCHUN, L., JIALING, T., AND JUNFA, M. Temperature-dependent device behavior in advanced CMOS technologies. In *2010 International Symposium on Signals, Systems and Electronics* (Sept. 2010), IEEE, pp. 1–4. A.1.1

[96] XILINX. 7 Series FPGAs Configurable Logic Block User Guide, 2012. 2.1.2

[97] XILINX. Zynq-7000 All Programmable SoC. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm, 2014. 6.2

[98] YAN, P.-Y. Understanding Bossung curve asymmetry and focus shift effect in EUV lithography. In *Proceedings of SPIE* (Mar. 2002), vol. 4562, SPIE, pp. 279–287. 2.3.1.2

[99] YE, Y., GUMMALLA, S., WANG, C.-C., CHAKRABARTI, C., AND CAO, Y. Random variability modeling and its impact on scaled CMOS circuits. *Journal of Computational Electronics 9*, 3-4 (2010), 108–113. (document), 2.3, 2.3.1.3