An Ultra-Low-Energy, Variation-Tolerant FPGA Architecture Using Component-Specific Mapping

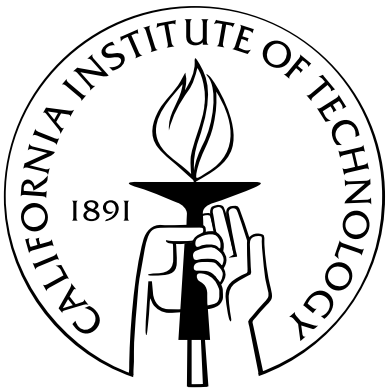
Thesis by

Nikil Mehta

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2013

(Defended August 31, 2012)

Abstract

As feature sizes scale toward atomic limits, parameter variation continues to increase, leading to increased margins in both delay and energy. Parameter variation both slows down devices and causes devices to fail. For applications that require high performance, the possibility of very slow devices on critical paths forces designers to reduce clock speed in order to meet timing. For an important and emerging class of applications that target energy-minimal operation at the cost of delay, the impact of variation-induced defects at very low voltages mandates the sizing up of transistors and operation at higher voltages to maintain functionality.

With post-fabrication configurability, FPGAs have the opportunity to self-measure the impact of variation, determining the speed and functionality of each individual resource. Given that information, a delay-aware router can use slow devices on non-critical paths, fast devices on critical paths, and avoid known defects. By mapping each component individually and customizing designs to a component's unique physical characteristics, we demonstrate that we can eliminate delay margins and reduce energy margins caused by variation.

To quantify the potential benefit we might gain from component-specific mapping, we first measure the margins associated with parameter variation, and then focus primarily on the energy benefits of FPGA delay-aware routing over a wide range of predictive technologies (45 nm–12 nm) for the Toronto20 benchmark set. We show that relative to delay-oblivious routing, delay-aware routing without any significant optimizations can reduce minimum energy/operation by 1.72× at 22 nm. We demonstrate how to construct an FPGA architecture specifically tailored to further increase the minimum energy savings of component-specific mapping by using the following techniques: power gating, gate sizing, interconnect sparing, and LUT remapping. With all optimizations considered we show a minimum energy/operation savings of 2.66× at 22 nm, or 1.68–2.95× when considered across 45–12 nm. As there are many challenges to measuring resource delays and mapping per chip, we discuss methods that may make component-specific mapping more practical. We demonstrate that a simpler, defect-aware routing achieves 70% of the energy savings of delay-aware routing. Finally, we show that without variation tolerance, scaling from 16 nm to 12 nm results in a net increase in minimum energy/operation; component-specific mapping, however, can extend minimum energy/operation scaling to 12 nm and possibly beyond.

Acknowledgements

First, and most importantly, I would like to thank my advisor André DeHon for his guidance on this work. He is a vast resource of insight and technical expertise, and his tireless support made this work possible. It has been a privilege to work with him over the years.

I would also like to thank Alain Martin for his mentorship and for making me a part of his research group, which helped make my last few years at Caltech enjoyable. I would also like to thank the other members of my committee for their support and feedback: Ben Calhoun, Azita Emami, and Ali Hajimiri.

Each student in the IC Lab has helped support this work, but two people deserve special mention. First is Rafi Rubin, for the incredible amount of insight and code he contributed to this project. Second is Ben Gojman, for his ideas and assistance in making this work come together. This thesis represents only the broad strokes of the much larger, more interesting picture of component-specific mapping. Ben and Rafi's dissertations will solve the hard problems that will complete this picture; I am confident that their work will far surpass what I have put forth here.

I would like to specially thank Kevin Cao and Sani Nassif for their collaboration in developing circuit failure models that contributed to this work. I specifically owe Kevin thanks for his support in answering my many questions about the predictive technology models. I also need to thank several people from academia and industry who provided invaluable suggestions: Guy Lemiuex, Tim Tuan, Sinan Kaptanoglu, Sean Keller, Peter Grossman, Peter Jamieson, Jonathan Rose, and Peter Cheung and each of his outstanding students.

I need to thank my family for their support over the many years it took for me to complete this project. I owe both of my parents and my sister so much for their love and support that helped get me through this process.

Finally, I would like to thank my wife Katie Shilton for her constant love, patience, encouragement, and belief in me. She has been a joy to be with even during the parts of this process that transformed me into a fixated, isolated monk. The key for me to finish has been her boundless support and love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis

An FPGA using post-fabrication component-specific mapping and an optimized architecture can reduce minimum energy/operation in future technology nodes by 1.68–2.95×, and can extend minimum-energy scaling by at least one additional technology generation.

1.2 Motivation

The scaling down of transistor feature sizes during the last several decades has led to unparalleled growth in the computational capability of integrated circuits. As individual transistors get smaller they get faster, and more transistors can fit in a fixed area. Historically, designers have considered performance and density to be the most important metrics driving the design of integrated circuits. Early circuits were so limited in density and performance that almost every design choice revolved around utilizing the large, slow transistors as efficiently as possible. For example, the Intel 4004 (the first widely available commercial microprocessor) contained a mere 2,300 transistors operating at 740 kHz. Forty years of scaling has led to processors with billions of transistors operating at clock frequencies of several GHz, enabling designers to have much more freedom in allocating transistors.

While density and performance are still important in modern integrated circuits, in the last decade two new metrics have emerged as primary design constraints: energy and reliability. In a keynote speech in 2005, Intel Fellow Shekhar Borkar described how energy/power and reliability will be the two biggest challenges facing the integrated circuit industry [21].

Energy and power have already become primary design constraints of current circuits: no longer is it possible to deliver performance at any cost, as designs must fall within a power density or energy budget [40]. Smaller feature sizes mean more transistors in a fixed area, but because voltages have largely remained constant [3], this increased density translates to more power dissipated per unit area. Increased power density leads to substantial heat generation, which may be too high to

cool easily. The absence of voltage scaling has also limited the scaling down of energy/operation, which limits the amount of time (i.e., number of operations) that a circuit can compute using a fixed supply of battery energy. An emerging and important class of applications such as micro-sensor networks [92] and biomedical sensors [103] have extreme battery and cost limitations; for these applications, minimal energy/operation is absolutely essential.

Reliability is starting to become a primary design constraint, and will likely become the dominant constraint for future technologies: Borkar estimated that 100 billion transistor designs in 2016 will be subject to substantially higher failure rates due to 20% fabrication-time defects [22]. With transistors currently sized at $\approx$ 50 silicon atoms long (22 nm), it becomes impossible to perfectly control their physical structure and atomic composition, leading to defects or variations in how an individual transistor will operate. Designing a functioning integrated circuit with billions of unique transistors with shapes and sizes that cannot be known a priori because of these process variations is an enormous challenge, and can lead to a significant percentage of non-functioning chips. With every new generation of yet smaller transistors, these reliability problems will only get worse.

Unfortunately, reliable operation and low-energy computation are largely competing goals. Most techniques that attempt to increase system reliability also increase energy (e.g., gate sizing, voltage margining, redundancy). Consequently, current techniques and methodologies that simultaneously lower energy/power and increase reliability are extremely scarce. Because reliable operation is of paramount importance, designers are more often than not willing to accept higher-energy operation in exchange for functional devices.

The motivation behind this report is to demonstrate one core technique, component-specific mapping (and its many possible optimizations), that will enable a device to both operate at very low energy and maintain reliable operation in future technologies. In the next section we will illustrate the idea behind component-specific mapping with a simple example, and then describe the scope and contributions of this report.

1.3 Component-Specific Mapping

The core of the reliability challenge in dealing with fabricating small transistors is that their exact electrical characteristics cannot be known prior to fabrication. Fabricated circuits may contain transistors that are either defective or too slow; additionally, a transistor that may be acceptable at nominal voltage may slow down or fail at a low voltage. The key idea behind component-specific mapping is to somehow measure the electrical characteristics of every transistor after a chip is fabricated, and to use that knowledge to customize the circuit in a way that avoids bad transistors and utilizes good ones at low voltages.

In order to do this, we require an integrated circuit that can be customized at the hardware

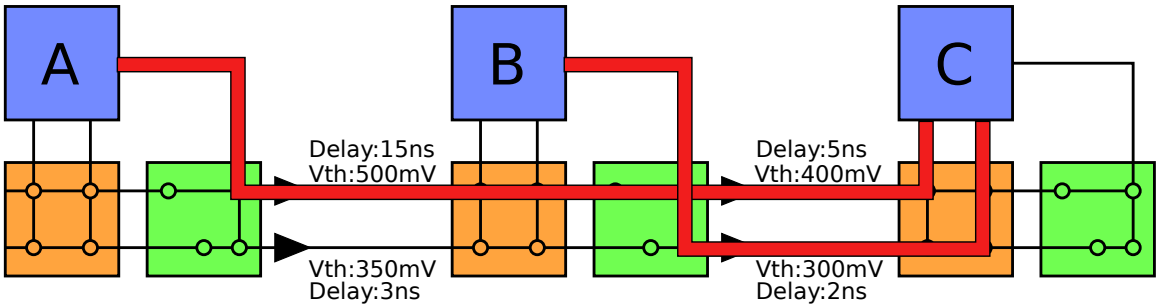
level after fabrication. The majority of chips today are fabricated as application-specific integrated circuits (ASICs), meaning that they are designed for a single fixed use; transistors and wires cannot be chosen or avoided after manufacturing. However, field-programmable gate arrays (FPGAs) are reconfigurable devices that can be programmed after fabrication to implement any digital logic (Section 2.1). In FPGAs blocks of configurable logic are connected via programmable interconnect, which are simply switches controlled by memory cells. By assigning appropriate values to those memory cells (i.e., the device configuration), any set of logical functions and connectivity between those functions can be mapped or re-mapped to the component.

The flexibility of making connections configurable (e.g., replacing a directly connected wire with a switch and memory cell) typically comes at a cost in performance, area, and energy. However, configurability means that instead of relying on external testers, one can configure in situ circuits to self-test and measure transistor characteristics in a device post-fabrication. Then, using that information, designs can be mapped to each fabricated FPGA in a component-specific manner. These per chip customized designs attempt to exploit the fact that every transistor is unique, and specifically target low energy, reliable operation.

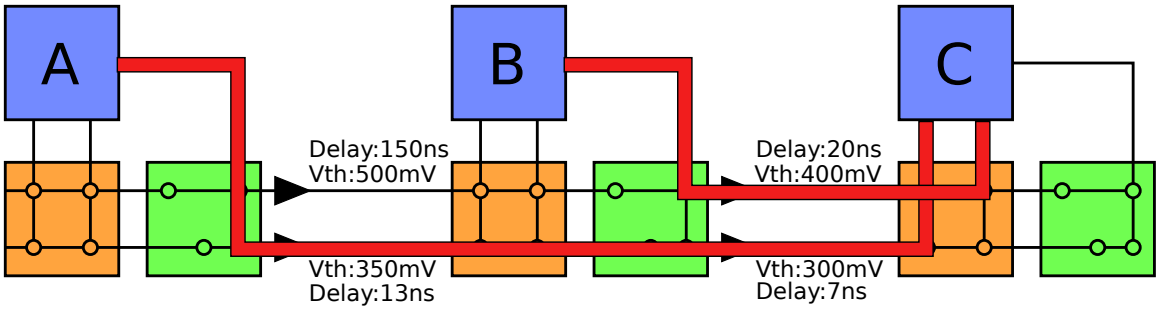
The first step in component-specific mapping is to perform a series of measurements per chip to extract delays for every resource. Once that information is obtained, it can be used by FPGA mapping algorithms to minimize energy under variation. To illustrate the basic idea and potential energy and delay benefits of post-fabrication component-specific mapping, we start with a simple, illustrative example. The goal will be to reduce the overall energy consumed by carefully selecting the assignment of FPGA routes to resources based on the component characteristics.

Figure 1.1 shows a cartoon version of simplified, 1D FPGA organization with three logic blocks (blue) computing functions A B C, connection boxes (orange) connecting wire segments to the inputs of the logic blocks, and switch boxes (green) connecting wire segments to each other and the output of the logic blocks. Programmable circular switches can be configured to connect wires and blocks together. The wires are driven by buffers at the start of the segment; each driver has a threshold voltage V_{th} which can be used to calculate the delay of the driver. The expected nominal threshold voltage is 400 mV. However, due to process variation, only one out of the four segments has its nominal value.

Assume our goal is to compute $C = A + B$ within 20 ns. The outputs of function blocks A and B must be connected to the input of function block C; there are two such possible configurations (Figures 1.1a and 1.1b). Ignoring logic delays, to meet our timing target the maximum delay of either path $A \rightarrow C$ or $B \rightarrow C$ through the interconnect must take at most 20 ns. Figure 1.2 shows the delay of a wire segment as a function of supply voltage V_{dd} and threshold voltage variation $\sigma_{V_{th}}$. We see that raising V_{th} makes the segment slower, while raising V_{dd} makes the segment faster. Because V_{th} is fixed at fabrication time, in order to tune segment delays we must change V_{dd}.



(a) Delay-oblivious route, $V_{dd} = 550\text{mV}$



(b) Delay-aware route, $V_{dd} = 400\text{mV}$

Figure 1.1: Component-specific mapping example

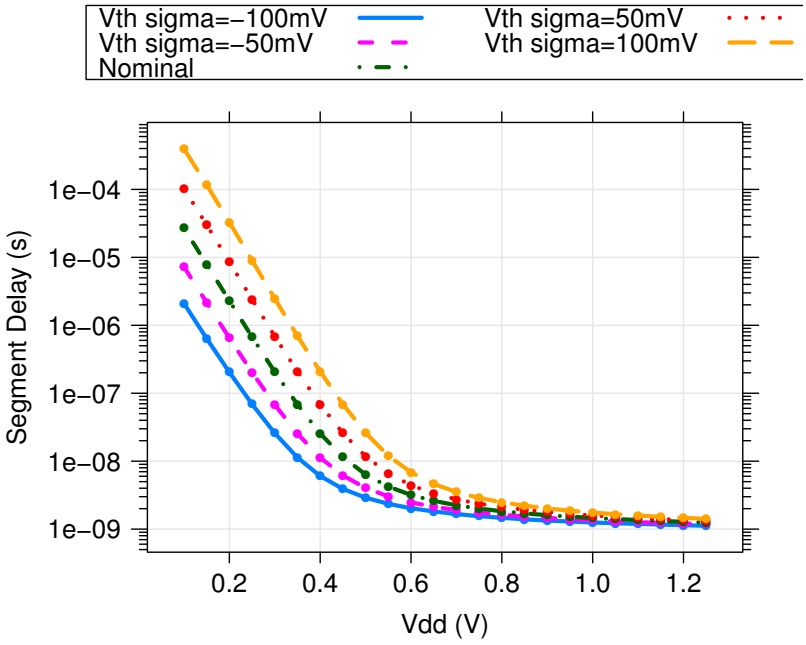


Figure 1.2: Delay as a function of V_{dd} and $\sigma_{V_{th}}$ for a 22 nm FPGA switch driver

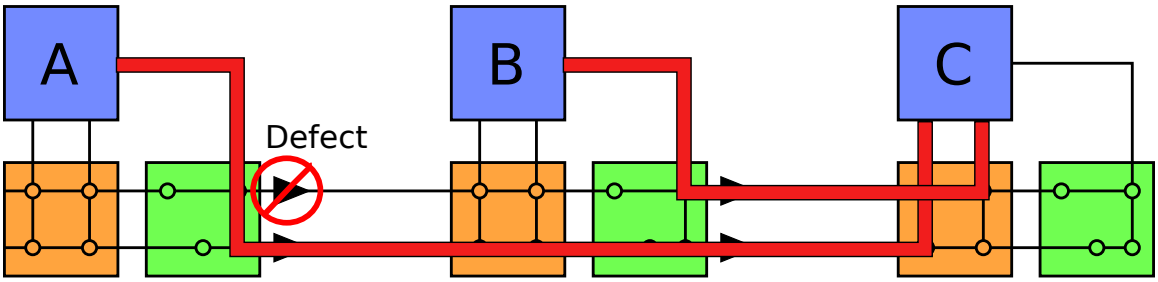


Figure 1.3: Defect-aware routing example

For a router that is oblivious to the variation present in the FPGA, all segments have drivers with $V_{th} = 400$mV, and both possible configurations are identical. Thus, the router may produce the results shown in Figure 1.1a, which use the two slowest, highest V_{th} drivers on the critical path $A \rightarrow C$. Using Figure 1.2 to determine delays, we see that to meet our interconnect timing requirement of 20 ns for this particular route configuration, we must set our supply voltage to at least $V_{dd} = 550$mV. This yields a path delay $A \rightarrow C = 20ns$ (with the parallel path $B \rightarrow C = 2ns$).

If we instead give a delay-aware router the delays of each segment as a function of V_{dd}, it can find the route shown in Figure 1.1b. Here, path $A \rightarrow C$ is still the slowest path, but using the full knowledge of component characteristics, the router assigns this long connection to fast, low V_{th} segments, avoiding the slowest driver with $V_{th} = 500mV$. Since the parallel path $B \rightarrow C$ is less delay critical, the router can afford to assign it to a higher V_{th}, slower segment while still meeting timing. At $V_{dd} = 400mV$, both paths $A \rightarrow C$ and $B \rightarrow C$ have a delay of 20 ns. In this example, using knowledge of the underlying characteristics of the FPGA can reduce the voltage and active energy consumption by a factor of $\left(\frac{550}{400}\right)^2 = 1.9$ compared to the delay-oblivious route. Delay-aware routing can be tuned to either improve delay at a fixed V_{dd}, or achieve identical delay but at a lower V_{dd}.

Delay-aware routing can also avoid defects, as defects can be easily detected in the delay measurement process. We will see that defects are not uncommon at very low V_{dd} and large amounts of V_{th} variation. Figure 1.3 shows the same example where the high V_{th} driver is actually defective at low enough V_{dd}; delay-aware routing can detect this defect and route around it. Avoiding these variation induced defects is critical to being able to safely lower V_{dd} to minimize energy.

While these simplified cases illustrate the use of routing to navigate interconnect resources and reduce energy consumption, it should be clear that the general idea can be applied much more broadly. Instead of just customizing configurations for interconnect, we can perform similar substitution or assignment for all resources (e.g., logic, memory). Furthermore, mapping based on post-fabrication resource knowledge can be exploited in not just routing, but in placement, memory assignment, clustering, and function binding.

1.4 Scope

Component-specific mapping is a large, challenging problem with many pieces and possible optimizations. This report does not represent a complete solution, as the scope and effort required for a working solution is worthy of several dissertations.

The most significant portion of component-specific mapping that is not addressed in depth is the problem of measurement. While Chapter 6 does illustrate the basic strategy for performing measurements, this work assumes that component-specific delay measurements are attainable and available for use in mapping.

The second major problem not addressed in this work is the cost of actually mapping every chip individually. This breaks the typical FPGA model of one-mapping-fits-all, and requires a CAD algorithm to be run for every chip. Chapter 6 also describes this problem in more detail with some existing, partial solutions that allow one CAD run to be customized per chip. However, these techniques are not explored in depth here.

Additionally, this work does not attempt to address all possible sources of unreliability. Effects such as transient errors (e.g., soft errors, shot noise, thermal noise), environmental effects (e.g., temperature variation, V_{dd} fluctuation), and lifetime failures (e.g., NBTI, HCI, TDDB, electromigration) are left for future work. Instead, we focus on the phenomena that is predicted to have one of the most important impacts on transistor reliability in near future technologies: random V_{th} variation [16, 47].

Finally, the scope of this work is to focus specifically on the metrics of minimum energy/operation and fabrication-time parametric yield. Minimum energy/operation is the critical metric for several important applications [92, 103], and also represents an important, fundamental limit on the energy efficiency of technologies. We will quantify aspects of performance and area; however, they will primarily be optimized in so far as they can help reduce energy. Additionally, this work will not focus on reducing power density, although low energy operation typically goes hand-in-hand with low-power operation.

1.5 Contributions

In the following chapters, we will explore the answers to the following questions involving component-specific mapping and make the following contributions:

- **Modeling:** How can we efficiently and accurately model key characteristics (performance, energy) of FPGAs in future technologies? Chapter 3 illustrates our techniques for performing HSPICE simulations and curve fitting to build an accurate model for FPGA circuit delay and energy. We also document our modifications to the academic CAD mapping tool VPR [95]

which allows us to explore a wide range of FPGA architectures.

Contributions:

- Construction of an HSPICE based device model for computing delay and energy, accurate to within 10% across a wide range of technologies, circuits, voltages and parameter variation.
- Enhancements to VPR which stabilize routing results under variation and allow for the exploration of more detailed architectures.

- **Benefits:** What are the benefits of component-specific mapping? Chapter 4 details our experimental methodology and builds an intuition for the expected benefits using experimental results without extensive optimizations.

  Contributions:

  - Quantification of delay and energy margins of delay-oblivious routing due to random V_{th} variation in interconnect.
  - Demonstration of the potential energy savings of component-specific routing.

- **Optimizations:** What optimizations can we perform to increase those benefits? Chapter 5 explores the energy and reliability tradeoff between adjusting gate sizes and the number of spare resources. We show that delay-aware routing is able to use smaller, less reliable gates to save energy, and that adding extra resources for more mapping flexibility results in even more energy savings. We also show that fine-grained power gating is necessary to achieve these benefits. Finally, we demonstrate techniques for LUT remapping to deal with variation in LUTs, further saving energy.

  Contributions:

  - Quantification of the impact of switch-level power gating on energy.
  - Quantification of the impact of interconnect buffer sizing on energy.
  - Quantification of the impact of interconnect sparing (extra channels, extra pins) on energy.
  - Quantification of the impact of LUT variation and subsequent LUT sizing and remapping on energy.
  - Design space exploration across all sizing and sparing variables.
  - Summary of energy savings of all optimizations combined.

- **Practicality:** Given the large cost in measuring every transistor in every chip and performing component-specific customization, what are the benefits of component-specific mapping without perfect measurement? Chapter 6 demonstrates that with imprecise delay information (100

Table 1.1: Component-specific mapping energy savings per optimization

Technique	**Energy Savings**
Component-Specific Mapping (Baseline, 22 nm)	1.72×
Power Gating	+5%
Uniform Interconnect Buffer Sizing	+6%
Selective Interconnect Buffer Sizing	+8%
Interconnect Channel Sparing	+0%
CLB I/O Sparing	+12%
LUT Remapping	+17%
Overall (22 nm)	2.66×
Overall (45 nm–12 nm)	1.68–2.95×

ps measurement precision or 8-bit storage precision) we can obtain the same energy savings achieved with full precision delay information. In fact, using only defect maps instead of full delay measurements, we can achieve 70% of the energy savings obtained by full delay-aware mapping.

Contributions:

- Quantification of energy savings of only defect-aware mapping.
- Exploration of the impact of delay measurement precision on component-specific mapping.
- Exploration of the impact of delay measurement storage on component-specific mapping.

- **Sensitivity:** How sensitive are these results to changes in technological or design assumptions? Chapter 7 shows that as we increase the pipelining of benchmark circuits, the size of circuits, and the magnitude of variation, the energy savings of component-specific mapping increase. We also show how our results scale from 45 nm to 12 nm feature sizes and demonstrate that energy savings increase with scaling. We further show that delay-oblivious mapping results in a minimum energy/operation *increase* at 12 nm due to variation, while delay-aware routing is able to extend minimum energy scaling.

  Contributions:

  - Sensitivity analysis of component-specific mapping to pipeline depth of the mapped circuit.
  - Sensitivity analysis of component-specific mapping to size of the mapped circuit.
  - Sensitivity analysis of component-specific mapping to magnitude of V_{th} variation.
  - Quantification of energy savings of component-specific mapping for 45 nm, 32 nm, 22 nm, 16 nm, and 12 nm technology nodes.

Table 1.1 quantifies the energy savings achievable by each of the optimizations and situations examined in this work. In the next several chapters we will examine these in depth.

Chapter 2

Background

To provide a baseline for understanding the optimizations performed in this work, in this chapter we will summarize important, basic concepts in FPGAs, energy, and variation. We will also describe prior work in low-power techniques for FPGAs, variation tolerance techniques for FPGAs, and component-specific mapping in order to ground our contributions.

2.1 FPGA Architecture

To understand delay-aware routing and its potential energy benefits, it first helps to understand the architecture of an FPGA. A conventional, island-style FPGA can be viewed as an array of configurable logic blocks (CLBs) connected by programmable interconnect (switchboxes and connection boxes, or SBoxes and CBoxes for short, respectively) (Figure 2.1). Inside the CLB (Figure 2.2a), N programmable lookup tables (LUTs) are connected together using internal interconnect. The LUT is simply a circuit that selects the output of an SRAM cell based on the LUT's k inputs: by programming appropriate values in the SRAM cells the LUT can implement any k-input function (Figure 2.2b).

All programmable interconnect is implemented using the simple, unidirectional switch that takes several inputs and selects one output (Figure 2.3b). The circuit consists of input stub buffers for each input that serve to isolate the switch, a multiplexer, and an output buffer that drives a long wire segment. Each of the input and output buffers can be built using a single or multiple staged inverters. Figure 2.3a shows how to assemble these switches into a switchbox.

With an FPGA any circuit can be realized; however, to support this programmability the FPGA must provision significantly more resources (buffers, wires, memory cells) than those required by a custom designed and fabricated implementation. This resource overhead costs area, performance, and energy (typical overheads are described later in Table 8.1).

The main sources of energy utilization in an FPGA are the logic, interconnect, clock network, and configuration memory. Modern FPGAs also consume non-negligible energy in embedded blocks

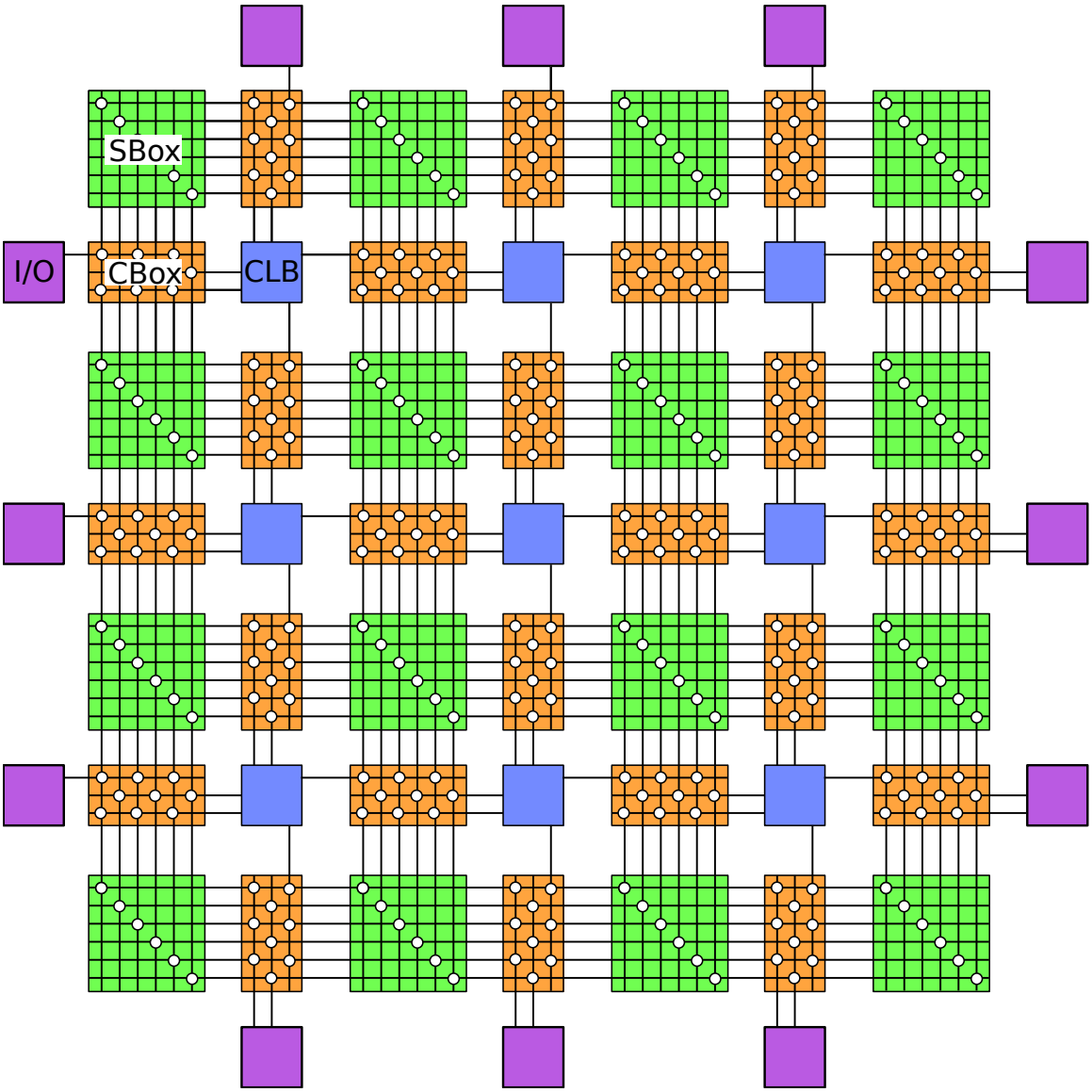
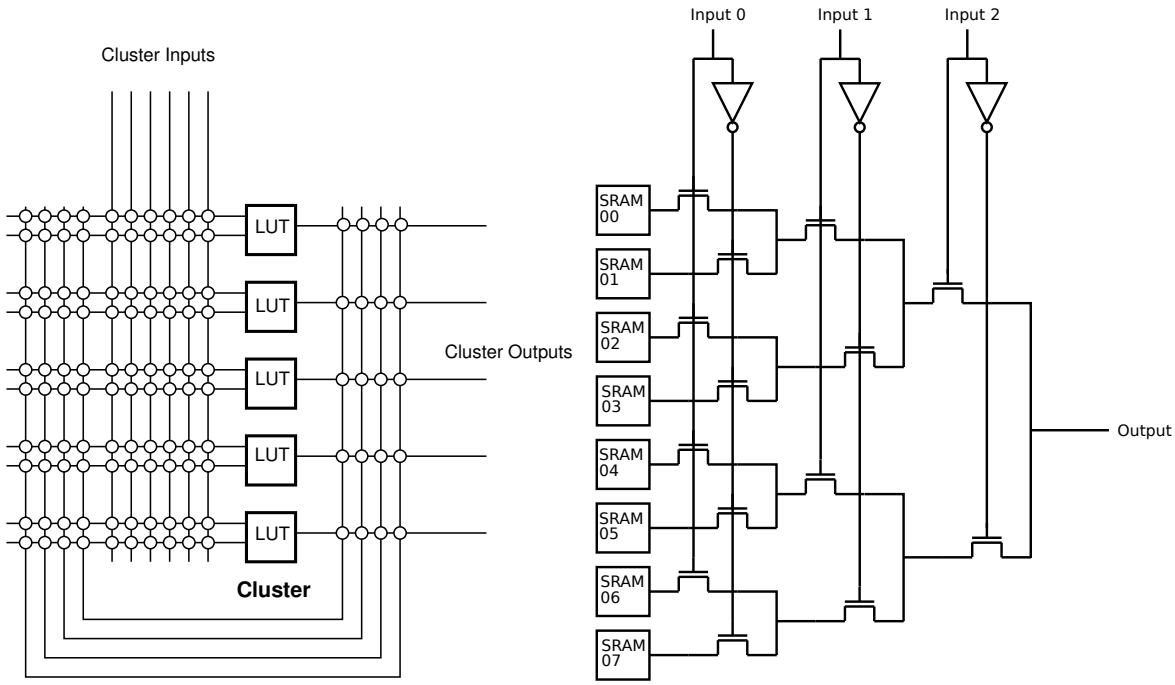


Figure 2.1: FPGA architecture

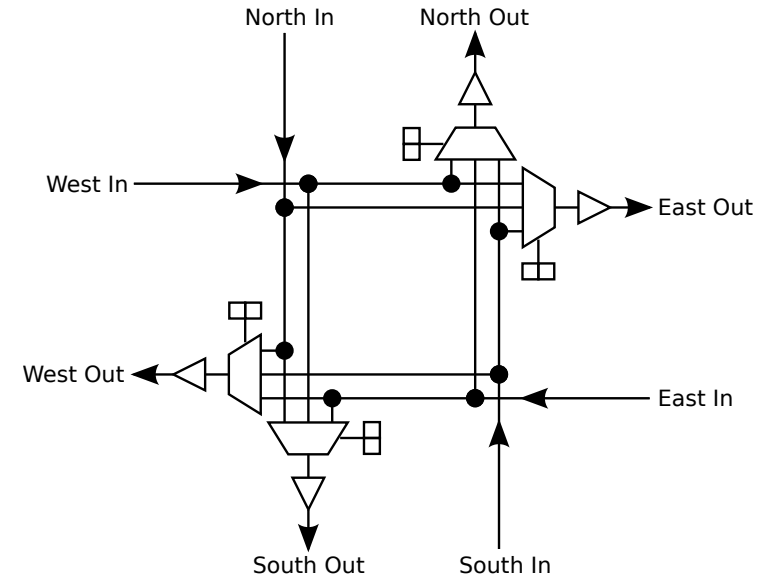


(a) CLB with 5 LUTs (b) 3-input LUT

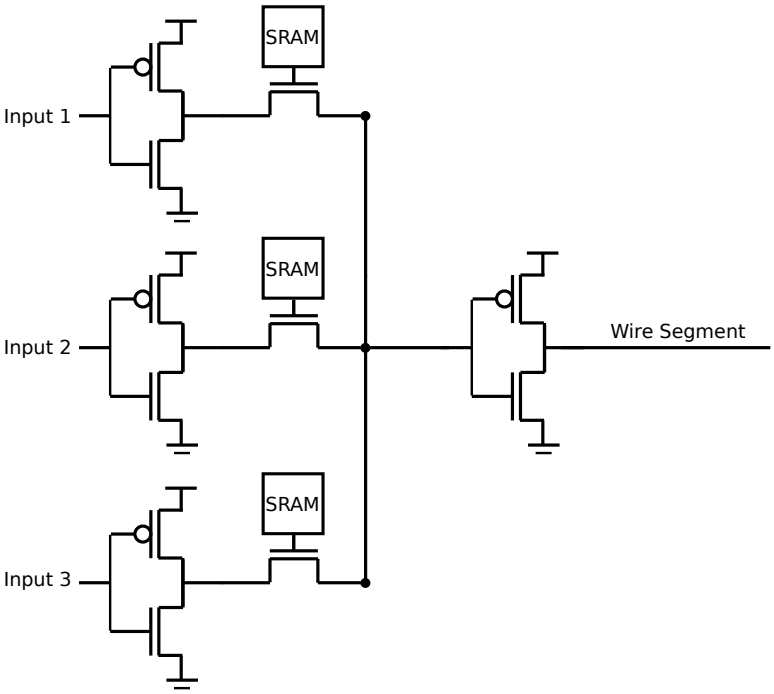
Figure 2.2: FPGA CLB and LUT circuits

(memories, multipliers, etc.) and in I/O drivers. Several studies have estimated the power dissipated in each of these elements to provide a breakdown of both static and dynamic power dissipation [60, 106,115]. Each study arrives at similar breakdowns for power dissipation; Figure 2.4 shows the most recently published results for a 90 nm Xilinx Spartan-3 [115]. Dynamic and static power charts are shown; in this particular device dynamic power is typically an order of magnitude larger than static power. In Figure 2.4a we see that interconnect dominates dynamic power, contributing 62%. Clock and logic power are the next two largest contributors and are roughly equivalent at 19%. For static power (Figure 2.4b) configuration bits contribute the largest amount of leakage at 44%, as every n-input switch requires n configuration bits and every k-input LUT requires 2^k configuration bits. Routing and logic transistors also leak significantly. If we assign the static power of configuration bits to their sources (either interconnect or logic), interconnect static power will dominate. Hence, interconnect is the primary source of both dynamic and static energy dissipation.

In order to configure a design onto an FPGA, several CAD operations are performed. A user typically starts with a design specified in a hardware description language (HDL), such as Verilog or VHDL. The design is then synthesized into gates and mapped into a netlist of LUTs via technology mapping. The LUTs are then grouped into clusters using a clustering algorithm, and then those clusters are assigned to physical CLBs via a placement algorithm. Finally, the routing algorithm specifies all the necessary routing tracks and configurations of switches needed to connect all the

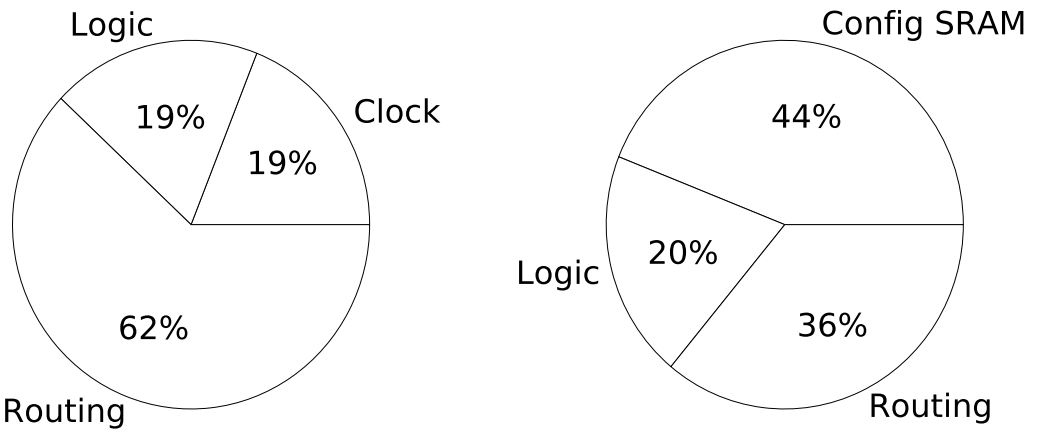


(a) Unidirectional switchbox



(b) 3-input switch

Figure 2.3: FPGA switchbox and switch circuits



(a) Dynamic Power

(b) Static Power

Figure 2.4: Measured power breakdown for 90 nm Xilinx Spartan-3 [115]

CLBs together.

Because interconnect is the dominant source of energy and delay in FPGAs, the assignment of nets to routing resources is critical. Further, as future technologies will be dominated by random V_{th} variation which operates at the scale of individual transistors, any successful variation mitigation scheme must have fine-grained control over which transistors to select. For example, placement only serves to relocate CLBs in the FPGA fabric, which will only have a very coarse-grained impact on which routing transistors are selected for configured paths. Routing, however, allows for fine-grained selection of routing transistors, and is the best opportunity for mitigating the delay and energy margins induced by variation.

Hence, in this work we focus largely on delay-aware routing as our primary variation mitigation technique. However, as LUTs still experience variation and can have an impact on total energy if ignored, we perform a level of component-specific LUT mapping using delay information (Section 5.4).

2.2 Energy

Power is a significant design challenge in modern integrated circuits, particularly in microprocessors. Using conventional air cooling technology (i.e., thermal paste, heatsinks and fans), the practical limit of power density that can be cooled is approximately 100 watts/cm^2 [101]. Total chip power can be

expressed as:

$$P_{dynamic} = \alpha \times C \times V_{dd}^2 \times f \tag{2.1}$$

$$P_{static} = I_{leak} \times V_{dd} \tag{2.2}$$

$$P_{transistor} = P_{dynamic} + P_{static} \tag{2.3}$$

$$P_{chip} = \sum_{i=0}^{N_{transistor}} P_{transistor}. \tag{2.4}$$

We see that the power of a single transistor can be expressed as the sum of dynamic and static power. Dynamic power depends on the switching probability α (0–1), the switched capacitance C, the supply voltage V_{dd}, and the clock frequency f. P_{static} is typically several orders of magnitude lower than the 100 watts/cm^2 limit, so $P_{dynamic}$ is usually the dominant factor. We can immediately see that one simple way to reduce power, and hence power density, is to just reduce the clock frequency f. By accepting lower performance, a given chip can always operate within a power envelope of 100 watts/cm^2. For many modern systems that are cooling limited, this is an acceptable tradeoff.

A more fundamental challenge is reducing energy/operation. Energy/operation is the primary design constraint for many low-power, embedded systems. While power is the key metric for describing how efficiently an integrated circuit can be cooled, energy is the key metric for circuits that operate using batteries. To perform each operation, a circuit must draw some number of joules from its battery; hence the joules/operation of a circuit expresses how many operations it can perform and thus how long it can operate with a given battery capacity. Energy/operation can be derived by the fact that power is simply energy per unit time:

$$E_{dynamic} = \alpha \times C \times V_{dd}^2 \tag{2.5}$$

$$E_{static} = I_{leak} \times V_{dd} \times \frac{1}{f} \tag{2.6}$$

$$E_{transistor} = E_{dynamic} + E_{static} \tag{2.7}$$

$$E_{chip} = \sum_{i=0}^{N_{transistor}} E_{transistor}. \tag{2.8}$$

Unlike power, reducing energy/operation is not simply a function of accepting slower operation by reducing clock frequency, which almost any design could accomplish. Here, we must reduce physical parameters such as capacitance and supply voltage.

Both C and V_{dd} can be reduced by accepting reduced transistor performance. The current and

delay of a transistor can be expressed as [119]:

$$I_{sat} = W \times v_{sat} \times C_{ox} \times \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right)^{\gamma} \tag{2.9}$$

$$I_{sub} = \frac{W}{L} \times \mu C_{ox} \times (n-1) \times (v_T)^2 \, e^{\frac{V_{gs} - V_{th}}{n \times v_T}} \times \left(1 - e^{\frac{-V_{ds}}{v_T}} \right) \tag{2.10}$$

$$I_{on} = \begin{cases} I_{sat} \;\; for \;\; V_{ds} = V_{dd} \geq V_{th} \\ I_{sub} \;\; for \;\; V_{ds} = V_{dd} < V_{th} \end{cases} \tag{2.11}$$

$$\tau_p = \frac{C V_{dd}}{I_{on}} \tag{2.12}$$

where W and L are the channel width and length, C_{ox} is the oxide capacitance, μ is mobility, v_{sat} is saturation velocity, n is the subthreshold slope, v_T is the thermal voltage, and V_{gs}, V_{ds}, V_{th} are the gate-source, drain-source, and threshold voltages respectively. The constant γ is between 1–2 and depends on the degree of velocity saturation.

Because $C \propto WL$, we can reduce dynamic energy linearly by sizing down transistors (until the fundamental technology limit of $W = L = 1$), at the cost of reducing drive strength I_{on} and thus increasing delay τ_p. Alternatively, we can drop V_{dd}, which will reduce $E_{dynamic}$ quadratically, again with a delay cost. Voltage scaling is a preferred technique for achieving low energy/operation because of its quadratic effect.

Many circuits may need to simply minimize energy/operation at the cost of delay. For these circuits, reducing V_{dd} to near zero may appear to be the optimal energy point; however, there exists a $V_{dd} > 0$ where energy/operation is minimized [24]. With $V_{dd} > V_{th}$ the delay of a transistor depends on I_{sat} and is super linear; however, when $V_{dd} < V_{th}$ delay depends on I_{sub} making it exponential in both V_{dd} and V_{th}. The exponential dependence in V_{dd} means that operations become significantly longer at lower voltages. As static energy/operation is expressed as leakage power times the length of an operation (Eq. 2.6), at low V_{dd} static energy will increase dramatically and eventually become the dominant source of energy dissipation.

Figure 2.5 shows the energy/operation of a 16-bit multiplier mapped to a 22 nm FPGA. We see that there exists a V_{dd} at which energy is minimized. This gives a well-defined target point of operation—we should operate at the energy-optimal V_{dd} when minimizing energy/operation. Furthermore, note that the minimum-energy point occurs below the threshold voltage in this example ($V_{th} = 300$ mV), as it does for most designs. These types of circuits where $V_{dd} < V_{th}$ are termed subthreshold circuits.

Minimal energy/operation is an important target for many designs, and represents fundamental limits on energy efficiency for CMOS technologies. As we scale to future technologies, we want the minimum-energy point to continue to reduce. However, parameter variation has a significant impact

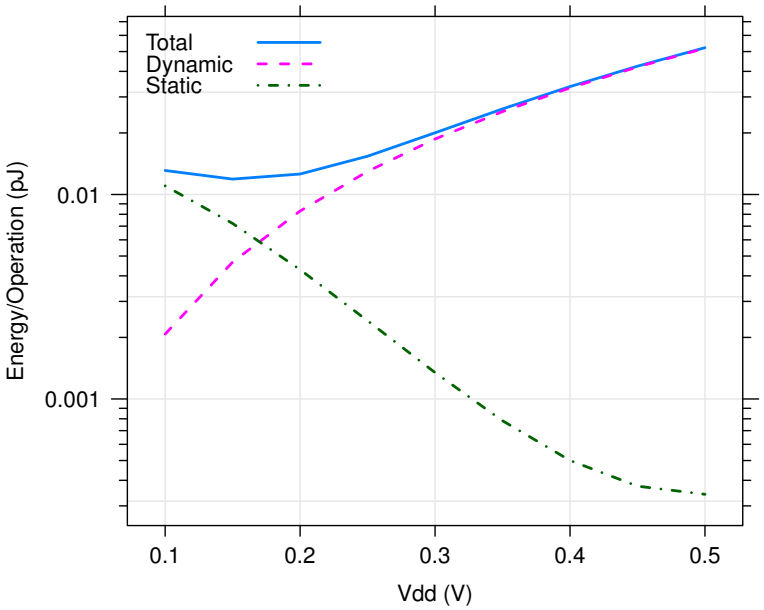


Figure 2.5: Minimum energy/operation for a 16-bit FPGA multiplier (22 nm HP)

on this minimum-energy point, and unmitigated it can increase energy/operation significantly; even to the point where it is no longer beneficial to scale to a new technology [20]. One of the goals of our work is to extend this range of beneficial, minimum-energy scaling through component-specific mapping.

2.3 Process Variation

While there are numerous, varied effects that can reduce the reliability of a device over its lifetime (NBTI, HCI, TDDB, electromigration [7, 8, 97, 111]), during operation (temperature variation, V_{dd} fluctuation [23]), or instantaneously (soft-errors, shot noise, thermal noise [15, 102]), a substantial source of unreliability in modern technologies is process variation. Traditionally, circuits have been designed using the assumption that all transistors have identical electrical characteristics. However, because fabrication is an imperfect process, it is impossible to have the physical control necessary to ensure uniform transistors. Process variations describe deviations of transistor electrical characteristics at fabrication time.

Slight process variations have always been present since the birth of CMOS processing; however, it is only recently with submicron-sized device that they have begun to significantly alter the characteristics of designs. Because of law of large numbers effects [33], variations on the scale of 10–100s of atoms (2–22 nm) in a transistor were not readily apparent for micron sized devices; however, for sub-100 nm-sized devices these variations are significant.

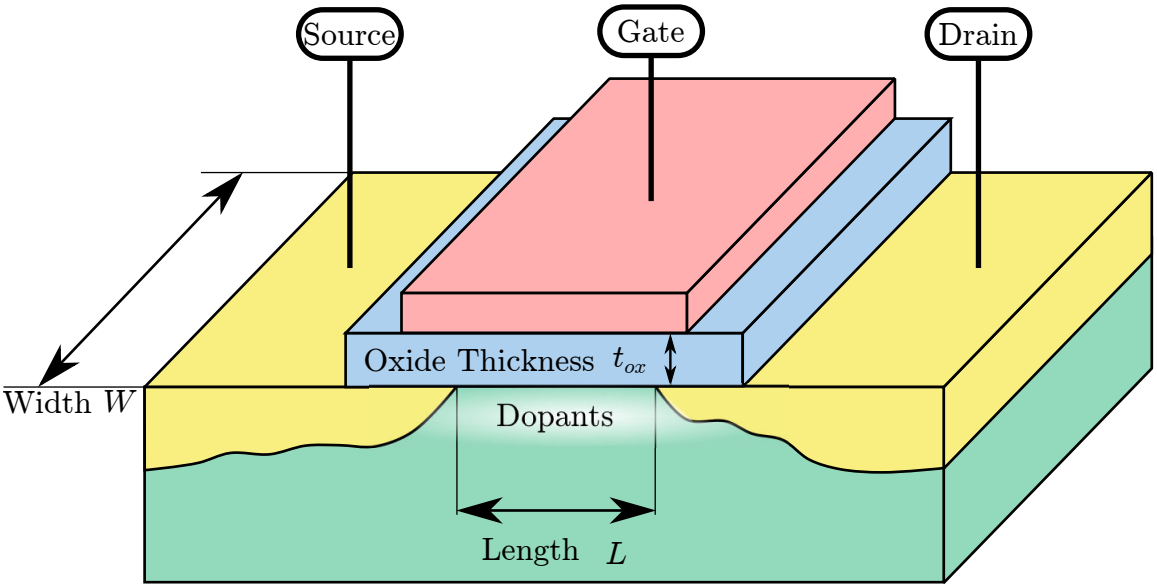


Figure 2.6: Transistor layout

Process variations can be classified based on the scale in which the variability of parameters manifest: lot-to-lot, wafer-to-wafer, die-to-die and within die (WID) variations. Historically, large scale variations such as wafer-to-water and die-to-die variations dominated smaller scale variations. However, with submicron processes die-to-die variations have become more significant, and with nanometer designs WID variations contribute the most to parameter variability [88].

Each type of variation can be further categorized based on predictability into systematic and stochastic. Systematic variations are changes in transistor characteristics that are typically layout dependent, and given enough pre-processing effort, can generally be predicted and corrected for [48]. Sources include mask errors, chemical mechanical polishing (CMP), optical proximity effects (OPE) and other tool optics [48, 127]. Variations that are either too difficult to predict based on layout or inherently unpredictable are classified as stochastic variations. Stochastic variations that can be modeled as distance dependent or regional (e.g., lens aberrations, oxide thickness variation [12]) are termed spatially correlated; variations that are at the scale of individual transistors are characterized as random.

Random WID V_{th} variation is caused by effects like random dopant fluctuation (RDF), channel length variance, and line-edge roughness [11,13]. RDF is the dominant source of random variation [3, 16], and arises from the injection of dopant atoms into a transistors channel (Figure 2.6). Because the injection process is not perfectly precise, the exact number and placement of dopant atoms will vary from transistor to transistor. This variation in dopant atoms results in threshold voltage variation, where V_{th} can be accurately modeled as a Gaussian random variable. Random variation, due to its inherently stochastic nature, cannot be predicted prior to fabrication unlike systematic or

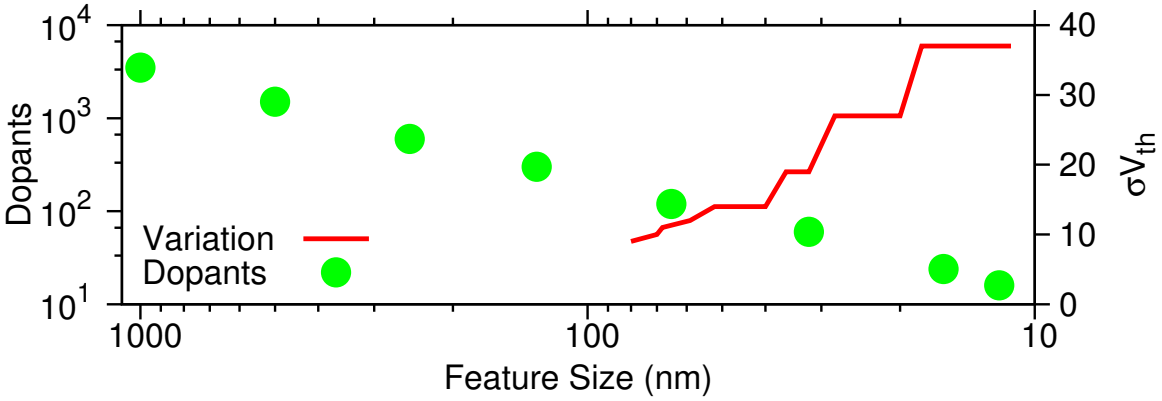


Figure 2.7: Decreasing dopants and increasing V_{th} variation from ITRS 2010 [3]

spatially correlated variation.

Further, as transistor volumes shrink with scaling, the magnitude of this variation will increase, as the standard deviation of V_{th} is inversely proportional to the transistor's cross-sectional area (assuming constant height):

$$\sigma_{V_{th}} \propto \frac{1}{\sqrt{WL}} \tag{2.13}$$

Random variation is expected to dominate future sources of variation [3, 16]. Figure 2.7 shows how V_{th} variation increases due to decreases in dopant count as a function of feature size. The impact of this random variation is three-fold:

Increased Delay: The first impact of V_{th} variation is that gates will exhibit a large spread in delays, reducing the speed of designs as the delay of a circuit is set by its slowest path. From Equation 2.12, we see that the delay τ_p is dependent on V_{th}; larger V_{th} reduces drive strength and increases gate delay.

Increased Energy: The second impact of V_{th} variation is that it raises energy per operation. For a circuit operating at a target delay, to compensate for slower gates, designers are often forced to raise V_{dd} to increase transistor drive strength (I_{sat}, Eq. 2.9), increasing energy/operation.

Increased Failures: The third impact of parameter variations is the increase in functional failures due to V_{th} mismatch [89]. SRAMs are commonly the first circuits to fail due to V_{th} variation which causes read upsets, write upsets, hold failures, and access time failures. Static logic, however, can also fail due to V_{th} variation. We can define a CMOS inverter to be defective due to variation when

leakage current overpowers on current:

$$I_{PMOS,off} > I_{NMOS,on} \text{ or } I_{PMOS,on} < I_{NMOS,off} \tag{2.14}$$

Under these conditions the inverter can never switch; this can only happen when V_{th} variation is large enough such that a very high V_{th} device is paired with a very low V_{th} device. Furthermore, as V_{dd} decreases, the probability of a defect increases as I_{on} degrades for both PMOS and NMOS transistors (Eq. 2.9) [61]. Consequently, this effect is particularly acute for subthreshold operation, preventing operation near the minimum-energy point [26].

In general, to avoid the negative effects of random V_{th} variation, designers typically opt to either increase V_{dd} or increase the size of transistors, both of which cost energy. Here, we can see how energy and reliability are competing goals. Conventional design techniques create energy margins, potentially wasting energy; this underscores the need for other techniques such as component-specific mapping to improve reliability without inducing significant energy margins.

2.4 Prior Work

In this section we summarize prior work in reducing FPGA power dissipation and improving FPGA variation tolerance. A large design space of CAD, architecture, and circuit optimizations have been previously explored. We will see that a some of these techniques are complimentary to component-specific mapping, while others are necessary to maximizing its benefit. Those techniques that are necessary (e.g., power gating, dual V_{th} design) will be explored in more depth in future sections.

Finally, we will summarize the energy savings achieved by each technique. We will see that this work attempts to dramatically reduce energy under variation by integer factors, in comparison to most prior work which achieves typically modest energy savings.

2.4.1 Low-Power Techniques for FPGAs

Optimizations for low-power operation in FPGAs have been explored at all levels of design: CAD, architecture, and circuits. CAD level optimizations typically operate by adapting the core CAD algorithm's cost function to target low-power operation. Architectural optimizations examine how FPGA power scales as a function of standard architectural parameters (e.g., number of LUTs per CLB, segment length).

2.4.1.1 CAD

Technology Mapping: The goal of low-power technology mapping is to reduce dynamic power (Equation 2.5) by either minimizing the switching activity of connections between LUTs (α) or by

reducing the total number of these connections and therefore the overall capacitance (C). Previous work has reduced interconnect switching activity by identifying the highest activity connections between gates and mapping in such a way that these nodes are internal to a LUT [39,62,71]. Other work has reduced the total number of interconnect segments by limiting duplication [9], a common technique used in technology mapping. In general, low-power technology mapping has been shown to yield approximately 7.6–17% power savings using both switching activity reduction and limited duplication [62,71].

Clustering: Low-power clustering operates on a similar principle as low-power technology mapping. Instead of covering high activity gate to gate connections within the same LUT, in low-power clustering, the goal is to place high activity LUT to LUT connections within the same CLB, utilizing the more energy efficient local interconnect. Lamoureux et al. demonstrated that for a cluster size of 4, an average energy reduction of 12.6% is achievable. [62] Alternatively, depopulation (removing LUTs from fully occupied CLBs and placing them elsewhere) has been shown to reduce routing resource demand, decreasing both average wirelength and switch utilization [32]. Approximately 13% total power can be saved through depopulation-based clustering [107].

Placement: Low-power placement algorithms attempt to place CLBs with high activity connections between them close together. Several works have implemented simulated annealing [55] placers with cost functions modified to reflect switching activity [46,62,116]. Each of these placers calculate the cost of a swap by adding a power term to the existing wirelength and delay terms; which includes the switching activity of the nets involved in the swap along with their capacitance. However, because net capacitance is not known prior to routing it must be estimated. On average, placers without accurate capacitance estimation reduce total power by 3.0% with 4% delay cost; placers with accurate capacitance estimation can achieve reductions of 8.6–13% with delay cost of 1–3% [46,116].

Routing: PathFinder [81], the standard FPGA routing algorithm, can be modified for low-power routing by adjusting the cost function to account for net activity and capacitance [46,62]. The cost of a resource is modified to include the activity of the net being routed and the capacitance of the candidate routing resource. In modern FPGA architectures, routing resources are not identical and have differing capacitances; hence, lower capacitance paths can be selected. Ideally, high activity nets should be mapped to low capacitance routes. However, as in placement, power-aware routers must be careful to balance delay and power minimization; low activity routes may end up being critical. In fact, power-aware routers have been shown to reduce total power by 2.6% but with a delay increase of 3.8%.

Glitch Reduction: A glitch is a switching event in a combinational circuit that does not contribute to the computation of that circuit's final value. Glitches occur when inputs to a gate arrive at different times, causing the gate to switch multiple times prematurely. Wilton et al. proposed pipelining circuits to reduces glitches [120]. A highly pipelined design has fewer glitches because it limits the number of logic levels between registers. Pipelining can reduce energy per operation substantially (40–90%) for circuits with significant glitching at a cost of latency. Glitches can also be minimized during CAD steps in either technology mapping [28] or routing [36]; glitch-aware CAD has been shown to reduce dynamic power by 11–18.7%.

Summary: We see that CAD optimizations to reduce FPGA power typically attempt to reduce switched capacitance and switching activity. Each of these techniques yields savings that would benefit both delay-aware mapping and delay-oblivious mapping under variation, and would not significantly change our results.

2.4.1.2 Architecture

Logic Block Architecture: Several researchers have attempted to determine energy favorable values of k (number of LUT inputs) and N (number of LUTs per CLB) when designing an FPGA logic block [67, 72, 91]. Changing k can have the following tradeoffs in terms of power: first, if k is increased, CLBs require more local routing to connect all LUT input pins (Figure 2.2a), increasing the dynamic energy of the interconnect local to the CLB. However, larger LUTs can implement more complex functions, reducing the total number of LUTs and CLBs. Fewer CLBs mean less demand for global routing resources, saving dynamic interconnect energy. The tradeoff in selecting N is similar: large values of N increase CLB capacity and functionality, which increases local interconnect energy but reduces global interconnect energy. k and N impact leakage energy solely by changing the total area of a design—larger area mean more devices leaking. It has been shown that $k = 4$ minimizes area [6] and therefore leakage energy [72]. Lin et al. examined total FPGA energy as a function of k and N [72], finding that a LUT input size of $k = 4$ and smaller cluster sizes (N = 6–10) typically use the least energy.

For this work we re-examined these results and found very similar architectural parameters that we used for this study ($k = 4$, $N = 4$) (Section 4.1).

Interconnect Architecture: Poon et al. [91] studied the impact of segment length L_{seg} and switchbox configurations on FPGA energy. A longer wire segment connects to more switches, increasing its capacitance; however, longer segments should lead to fewer total switches. They found the shortest segments $L_{seg} = 1$ are the most energy efficient.

Jamieson et al. [51] studied the effect of directional versus bi-directional switches on interconnect

energy. In the past, FPGAs were manufactured with bi-directional switches. These switches suffer from the inefficiency that, once configured, an FPGA only uses a switch in one direction; hence, only 50% of drivers are ever utilized. Directional switches drive segments in a single direction, ensuring that all drivers could be utilized. While directional drivers would seem to require more wiring because wires can only be utilized in one direction, in practice they use the same number of wires as the bi-directional case, saving area and improving delay [65] and energy [51].

For this work we also found directional drivers with $L_{seg} = 1$ to be energy optimal (Section 4.1).

Dynamic Voltage Scaling: Dynamic voltage scaling (DVS) reduces V_{dd}, saving both dynamic and static energy, but at the expense of increasing delay. DVS is useful in scenarios where a design needs to operate at a target frequency. In these cases, V_{dd} can be lowered to the point where the target is still achieved, minimizing the energy wasted on margins. The exact value of V_{dd} can be different between chips due to variation and can change over time due to environmental variation; hence, an on-chip control circuit with delay feedback is typically used to adjust V_{dd}. DVS in FPGAs was examined by [30]. They use a design-specific measurement circuit that tracks the delay of a design's critical path to provide feedback to the voltage controller. Through this technique energy savings of 4–54% can be observed.

DVS is one of the most effective ways to reduce energy/operation because of its quadratic effect. For this work, our main goal is to reduce V_{dd} but to tolerate variation. Our delay-oblivious router effectively represents a DVS case where the minimum V_{dd} is found given a set of yield and delay constraints. The delay-aware router attempts to further scale V_{dd} through component-specific mapping.

Power Gating: Leakage power is a significant source of FPGA power dissipation, and is expected to increase as technology scales [114]. FPGAs have significant area overhead due to programmability—hence, large portions of an FPGA are often unused. Instead of leaving these unused portions to sit idle and leak, it is possible to use power gating (Figure 2.8). In power gating a large, high V_{th} sleep transistor is inserted between the power supply and the block to be gated. The high threshold ensures that leakage through the sleep transistor will be negligible. The gate of the sleep transistor is tied to a control bit that can either be set at configuration time (i.e., static power gating) or during runtime (i.e., dynamic power gating).

Many researchers have explored different points in the design space for power gating granularity in FPGAs. Calhoun et al. [25] perform power gating at the gate level; Gayasen et al. [42] use larger power gating blocks, choosing to gate off regions of 4 CLBs at a time. They show 20% leakage power savings and that coarse-grained power gating with improved placement achieves the same results as fine-grained power gating. Rahman et al. [93] suggest that a combination of fine and coarse-grained

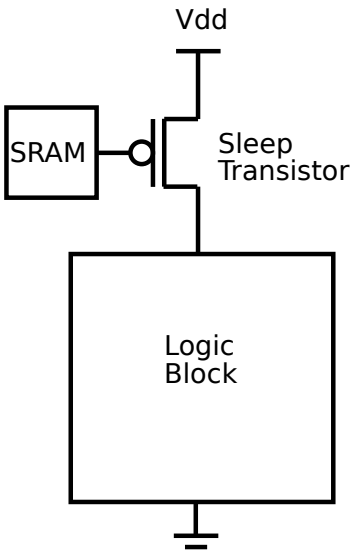


Figure 2.8: Power gating circuit

power gating provides the best results.

In this work we found power gating to be essential to minimizing energy/operation. We used a simple, fined-grained, switch-level power gating scheme to reduce energy (Section 5.1).

Dual V_{dd}: Instead of simply using sleep transistors to disable or enable blocks of logic, they can be used to select from different supply voltages (V_{dd}) to power the block. Because not all paths in a circuit are critical, only elements on critical paths need to be placed in high V_{dd} regions to ensure fast operation; all other block can be on a low V_{dd} region to save power. Dual V_{dd} design has been studied extensively in the FPGA literature [10, 41, 50, 68, 69, 73], typically achieving $\approx 50\%$ leakage power savings.

Dual V_{th} and Body Biasing: Dual V_{th} FPGAs define low and high V_{th} regions at fabrication time. High V_{th} regions reduce leakage at the cost of increased delay. Body biased FPGAs are very similar, using embedded circuitry to change the body the source voltage for regions at configuration time, which effectively changes V_{th}.

Much work has also been done in dual V_{th} design for FPGAs [49, 54, 70, 115] with body biasing being integrated into commercial FPGAs [66]. Similar tradeoffs exist when mapping circuits to either dual V_{th}/body biased architectures or dual V_{dd} architectures. Critical paths must be placed on low V_{th} blocks to ensure minimal delay reduction, and timing slack must be available to save leakage energy on non-critical paths. Block granularity is important [49] and selection of appropriate body bias voltages is important [54].

While the previous discussion of V_{th} and V_{dd} selection applies at the level of the mapped design, it is also profitable to use different V_{th} devices for circuits that play different roles in the FPGA

Technique	Level	Reduces	Benefit Type	Benefit
Technology Mapping	CAD	C, α	P_{total}	7.6–17%
Clustering	CAD	C, α	P_{total}	13%
LUT Input Transformation	CAD	I_{leak}	P_{static}	50%
Placement	CAD	C, α	P_{total}	3.0–13%
Routing	CAD	C, α	P_{total}	3%
Glitch Routing/Placement	CAD	α	$P_{dynamic}$	11–19%
Logic Block Architecture	Architecture	C	P_{total}	48%
Interconnect Architecture	Architecture	C	P_{total}	12%
Dynamic Voltage Scaling	Architecture	V_{dd}	P_{total}	4–54%
Power Gating	Architecture	I_{leak}	P_{static}	20%
Dual V_{dd}	Architecture	V_{dd}	P_{total}	50%
Dual V_{th}/Body Biasing	Architecture	I_{leak}	P_{static}	43%
Low Swing Interconnect	Architecture	V_{dd}	P_{total}	50%
Subthreshold Operation	Architecture	V_{dd}	P_{total}	22×

Table 2.1: Roundup of low-power FPGA techniques

architecture. Notably, high V_{th} devices can significantly reduce the configuration SRAM bit leakage reduced without impacting area or delay [115]. Configuration bits are a prime candidate for high V_{th} transistors because they constitute a significant fraction of FPGA area and are always on and leaking. Fortunately, configuration bits are set only at configuration time and do not contribute any delay or switching energy to mapped circuits. Increasing configuration SRAM V_{th} has been shown to reduce leakage energy by 43% for a particular implementation [115]. Today's commercial FPGAs are fabricated with three different effective threshold voltages to reduce leakage [56].

In this work we found multiple V_{th} voltages to be very beneficial in controlling SRAM energy. We utilize high V_{th} transistors for SRAM cells to substantially reduce their leakage energy (Section 3.1.6).

Low Swing Interconnect: A low swing interconnect segment consists of a driver and receiver operating at nominal voltages, with the wire between them operating at reduced voltage. The driver converts a full swing input into a low swing interconnect signal and the receiver converts it back. With this technique the amount of dynamic energy dissipated in interconnect segments can be reduced significantly; for an FPGA, interconnect energy can be reduced by a factor of 2 [43].

Subthreshold Design: We have seen that minimal energy/operation is achieved when V_{dd} is set below the threshold voltage (Section 2.2). Ryan et al. fabricated an FPGA designed for subthreshold operation, demonstrating a very significant 22× energy reduction relative to a conventional FPGA at full V_{dd} [100]. The design used dual V_{dd}, low swing interconnect to reduce energy and improve delay. Our work builds on this idea, similarly constructing an FPGA architecture targeted for minimum energy, subthreshold operation. We explore the impact of variation on a subthreshold FPGA, and demonstrate the benefits of component-specific mapping and architectural design space exploration.

Summary: Table 2.1 summarizes low-power FPGA techniques, the level at which they operate, the terms in the power equations they attempt to reduce, the type of power reduction, and the demonstrated benefit. It is important to note that the experimental parameters for each technique explored in prior work is very different, so the savings achieved per technique may not be directly comparable.

This work uses many low-power concepts explored in prior work. Most importantly, we build upon the idea of operating an FPGA in subthreshold, which yields tremendous energy savings. In terms of CAD we focus primarily on routing for variation tolerance; we do not alter the routing cost function to target low power. Similar to prior work, we found the energy savings to be negligible considering the delay cost.

We leverage the work done in architectural exploration, selecting our minimum-energy architectural parameters (k, N, L_{seg}) according to those found in prior work. Additionally, we examine the energy impact of several architectural optimizations not explored before: CLB I/O sparing, extra channels, and buffer sizing. We also re-examine the impact of power gating at different granularities in detail in Chapter 5 for the purposes of on component-specific routing.

Finally, we demonstrate energy savings of 1.68–2.95×, larger than the savings shown by most prior work in FPGA power reduction.

2.4.2 Variation Tolerance in FPGAs

To tolerate FPGA parameter variation, researchers have employed many of the same techniques used for ASICs and CPUs. Typically these optimizations fall under statistical CAD techniques or architectural parameter selection. To describe the impact of variation and the benefits of variation tolerance on FPGA designs, metrics like timing yield and power yield are often used.

SSTA: A heavily researched method for dealing with variation during the technology mapping, place and route stages of an ASIC design is the use of statistical static timing analysis (SSTA). SSTA attempts to model variability directly in CAD algorithms to help produce circuits that inherently account for variation. Traditional CAD algorithms do not aggressively minimize near-critical paths since their reduction would not reduce the delay of the circuit. However, near critical paths are important under variation because there is a probability that they will become critical. SSTA is a methodology that identifies statistically critical paths and enables CAD algorithms to optimize those paths. Integrating SSTA into clustering, placement and routing algorithms of the FPGA CAD flow simply involves replacing the nominal STA routine with an SSTA routine.

FPGA CAD algorithms modified for SSTA have been studied for placement [78] and routing [58, 109]. Lin et al. studied a full SSTA CAD flow with clustering, placement, and routing and characterized the interaction between each stage [75]. For a 65 nm predictive technology with

$3\sigma_{L_{eff}}/\mu_{L_{eff}}$ and $3\sigma_{V_{th}}/\mu_{V_{th}}$ of 10%, 10%, and 6% for D2D, WID spatially correlated, and WID random variation, they observed that SSTA based clustering alone can improve μ_τ and σ_τ by 5.0% and 6.4% respectively, and improve timing yield by 9.9%. Placement improves μ_τ, σ_τ and timing yield by 4.0%, 6.1%, and 8.0%; routing achieves 1.4%, 0.7%, and 2.2% improvement. All three SSTA algorithms combined yield 6.2%, 7.5%, and 12.6% improvement.

A critical distinction between this work and SSTA is that SSTA is used for one-sized-fits-all mapping, not component-specific mapping. SSTA attempts to make predictions for a single mapping that will be statistically beneficial for many different mappings. The advantage of component-specific mapping is that it is able to customize mappings per chip, rather than estimating optimized mappings prior to fabrication.

Architecture Optimization: Similar to finding architectural values of N and k that reduce energy, one can attempt to find the values of architectural parameters that improve timing and leakage yield.

In terms of logic, larger values of N and k increase the area, delay and leakage of individual CLBs, which will hurt leakage and timing yield. However, the total number of CLBs and required routing resources may decrease, which would improve leakage yield. Additionally, the number of CLBs and switches on near-critical paths may decrease with larger k and N, improving timing yield. Wong et al. studied the impact of N and k on timing and leakage yield [123]. They observe that while N has little impact on yield, $k = 7$ maximizes timing yield, $k = 4$ maximizes leakage yield, and $k = 5$ maximizes combined yield. These results are not surprising since leakage roughly scales with area and timing yield is directly related to delay; these results largely match the k values that optimize delay, area, and area-delay product, respectively.

Wire segmentation can have an impact on timing yield for similar reasons as N and k. For a fixed distance net, smaller values of L_{seg} increase the number of buffers on the path, increasing mean delay but decreasing variance due to delay averaging. Kumar et al. found that compared to an FPGA with 50% $L_{seg} = 8$ and 50% $L_{seg} = 4$ segments in a 45 nm predictive technology with $3\sigma/\mu = 20\%$ variation, using a mix of shorter segments can reduce both mean delay and variance [58]. They find that architectures with between 20–40% $L_{seg} = 2$ segments can improve μ_τ by 7.2–8.8% and σ_τ by 8.7–9.3%. This is in contrast to the minimum-energy segmentation of $L_{seg} = 1$.

Asynchronous Design: Choosing the right timing target to achieve high performance and acceptable timing yield is a significant concern with synchronous designs under process variation. If a timing target is chosen too pessimistically, performance of manufactured designs will suffer; if it is chosen too aggressively, many devices may fail. While DVFS can help mitigate this problem, another possible technique is to design circuits that are not clocked and do not run at a uniform

Technique	$3\sigma/\mu$ V_{th} Variation		Timing Improvement		Yield Improvement	
	Regional	Random	μ	σ	Timing	Leakage
SSTA Clustering	10%	6%	5.0%	6.4%	9.9%	-
SSTA Placement	10%	6%	4.0%	6.1%	8.0%	-
SSTA Routing	10%	6%	1.4%	0.7%	2.2%	-
Logic Block Architecture	?	?	-	-	9%	73%
Interconnect Architecture	-	20%	7.2–8.8%	8.7–9.3%	-	-

Table 2.2: Roundup of FPGA techniques for variation tolerance

frequency. Instead of using a global clock for synchronization, sequencing can be performed by using handshaking circuitry between individual gates. These self-timed, or asynchronous circuits have been studied in depth by ASIC designers and have been explored by FPGA designers as well. Asynchronous FPGAs have been demonstrated in [79,122] and have even begun to be commercially manufactured [4].

If one adopts a fully asynchronous system model, the system will always work regardless of the variability, which reduces the design burden of achieving timing closure in synchronous circuits. The asynchronous design decouples the delay of unrelated blocks allowing each to run at their natural speed. Nonetheless, the throughput and cycle-time of asynchronous circuits is impacted by variation, and high-variation can prevent the asynchronous circuit from meeting a performance yield target as well. The main drawback of asynchronous circuits is that they require some area and energy overhead for handshaking circuitry, although energy is simultaneous reduced by eliminating the clock network and through implicit clock gating (as only circuit elements that are performing computation are switching). There has been no published work quantifying the advantages of asynchronous FPGAs under variation, which may be a very promising area for future work.

Summary: Table 2.2 summarizes techniques for variation tolerance and their demonstrated benefit. These techniques show that FPGAs can begin to tolerate variation by using very similar techniques as those used by ASICs. However, none of these techniques use the most powerful tool at the disposal of an FPGA: reconfiguration. By relying on static, pre-fabrication optimizations (e.g., architectural explorations) or pre-fabrication variation estimates (e.g., SSTA), these techniques cannot fully compensate for the fact that every transistor has a unique, random V_{th} that significantly impacts both delay and energy. Hence Table 2.2 shows relatively meager benefits for prior work in FPGA variation tolerance.

2.4.3 Component-Specific Mapping

Our goal is to surpass the energy and reliability benefits quantified by prior work by using component-specific mapping. Component-specific mapping is not, however, a new idea— several prior works have investigated customized mapping for defect tolerance and even parameter variation.

Defect-Tolerance: Initial work in component-specific mapping focused on defect tolerance, with HP's TERAMAC demonstrating the ability to locate and map around defective elements and tolerate defect rates of 3–10% [31]. A modern day instance of using component-specific defect mapping is Xilinx's EasyPath program, where Xilinx tolerates defects in fabricated FPGA by matching the resource needs of a particular design to individual fabricated components [57, 76]. Since no design uses all of the features and resources in an FPGA, defects in the unused resources are tolerable. Given a customer design, Xilinx simply needs to make sure that they identify particular FPGAs where the defects do not interfere with the user's logic. This avoids additional work to map the design individually for each component; using design-specific testing [118] can also avoid the need to create a map of the defects on the FPGA.

Multiple Configurations: Generation of multiple bitstreams is another way to perform defect avoidance per component. Many possible valid mappings exist for a design when mapping to an FPGA. To avoid unwanted FPGA resources, at a coarse granularity one could simply place and route a design several times to produce multiple bitstreams and then test the bitstreams on the component. If any of the bitstreams avoids all the defective devices, we have a successful mapping [80, 105, 113]. With care, one can generate a set of complementary mappings that deliberately use distinct resources. This technique also avoids the need for per-component mapping and the need to generate a defect map at the cost of performing design-specific testing. However, this scheme is most viable when there are just a few defective resources in an FPGA; it does not scale well to high defect rates and high variation as generation of sufficient numbers of mutually exclusive configurations becomes intractable.

Dual V_{dd} and Body Biasing: Dual V_{dd} and body biasing can be performed on a component-specific basis to improve both timing yield and leakage yield. In both techniques blocks that contain critical paths are assigned either a high V_{dd} or low V_{th} to maintain performance. Because the primary impact of variation is that some devices become fast and leaky while others become slow and less leaky, these same techniques can be used to improve both timing yield and leakage yield.

Nabaa et al. examined body biasing for an FPGA to improve timing yield [85]. In their scheme each CLB and switchbox is assigned a body bias circuit. They find that for a 130 nm technology (with unspecified variation) body biasing can reduce timing variation by 3.3× and leakage variation by 18×. Bijansky et al. studied the impact of tuning dual V_{dd} FPGA architectures to compensate for variation [18]. In their architecture each CLB can choose from two voltages, with voltage assignment taking place post-fabrication on a component-specific basis. For a 65 nm predictive technology with $3\sigma_{V_{th}}/\mu_{V_{th}} = 20\%$ random variation, they report an average timing yield improvements of 15%.

The primary drawback of these techniques is that they have limited granularity: they can only

impact the V_{dd} or V_{th} of block, not of individual transistors. While they are ideal for region-based variation, these techniques do not scale well with high random variation; supporting individual V_{dd} connections or bias circuitry on a per transistor basis would add excessive area overhead, defeating the benefits offered by this tuning mechanism.

Post-Silicon Clock Tuning: Component-specific mapping concepts have also been incorporated into ASIC designs, such as the Itanium 2 processor [112]. The Itanium 2 includes programmable delay buffers in the clock distribution network that can be configured after manufacturing to compensate for clock skew variations. Critical path delay can also be improved—by slowing down/speeding up clock signals to the input/output registers of a critical path, time can be borrowed from adjacent paths (if those paths are not also critical). Algorithms for automatically tuning delay buffers such as these on a per-chip basis is a subject of active research (e.g., [86]).

Post-*e.g.*silicon tuning has also been demonstrated for FPGA clock networks [109] at a fully component-specific level. Programmable delay buffers are inserted at the leaf of the clock tree at every flip-flop clock input. Delays are assigned by measuring certain designated paths (as opposed to having full delay knowledge) using techniques from [104]; these measurements are then used to predict actual critical path delays through statistical analysis. This technique improves timing yield by 12% on average

Component-Specific Placement: Some existing work has explored modifying the FPGA CAD flow to incorporate component-specific mapping using full delay knowledge [29, 53]. These schemes use delay knowledge during placement to place critical paths in fast chip regions and non-critical paths in slow regions (Figure 2.9). Every chip receives a unique placement and is then routed without delay knowledge.

To perform chipwise placement, a variation map must be generated on a per-chip basis. A critical assumption of this technique is that an FPGA can be divided into regions (typically sized as a CLB plus surrounding routing resources) where each region has similar performance characteristics. Once a regional variation map is obtained, placement proceeds precisely as in the knowledge-free case, except instead of assigning all resources the same delay in the placement algorithm, resources within a region are assigned a delay from the variation map.

Katsuki et al. custom fabricated 31 90 nm FPGA test chips and constructed a delay map of each chip [53]. Under an unspecified amount of variation, they demonstrated a delay improvement of 4.1%. Cheng et al. performed a similar knowledge placement experiment under simulation [29]. For $3\sigma/\mu = 10\%$ variation in both L_{eff} and V_{th}, they report on average between 6.91–12.10% performance improvement.

An important limitation of component-specific placement is the assumption that variation is

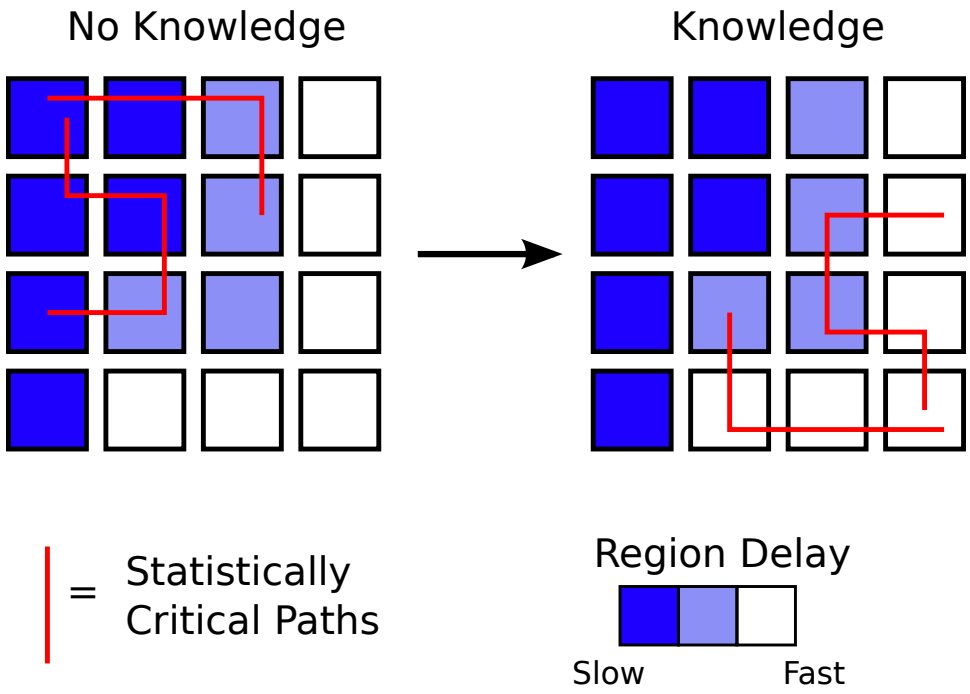


Figure 2.9: Full knowledge placement for region-based variation

primarily due to the spatially correlated component (regional) rather than the uncorrelated, random component. Variation in future technologies is expected to be dominated by random effects; hence, there will a stronger effect from fast or slow individual resources rather than fast or slow regions.

Component-Specific Routing: Gojman et al. [45] perform fine-grained, component-specific routing on an reconfigurable NanoPLA [35] using an algorithm termed VMATCH. The NanoPLA is a nanowire based architecture that uses crossed 5 nm diameter silicon nanowires to form a PLA like structure, with each wire crossing representing either a programmable diode performing a wired-OR, or a FET-like device performing negation. When these two planes of devices are combined they form the OR-INVERT function, which is logically equivalent to NAND. The NanoPLA shares some similarities to a conventional FPGA in that both use Manhattan routing to connect discrete clusters of logic. However, routing in the NanoPLA is done through the blocks rather than using an independent switching network. In order to allow signal routing, the output of the OR-plane of every block connects to itself and neighboring blocks.

The goal of VMATCH is to mitigate the dramatic amount of V_{th} variation present in 5 nm nanowires. One property of circuits mapped to the NanoPLA is the large amount of fanout variation present in the nets. VMATCH attempts to match net fanouts to physical threshold voltages, where larger, slower fanout nets are matched to faster devices. By adding a modest amount of extra resources they are able to restore 100% yield in a 5 nm technology with $\sigma_{V_{th}}/\mu_{V_{th}} = 38\%$.

While this work shares the core idea of component-specific routing with VMATCH, conventional

FPGA architectures cannot take advantage of the same fanout matching properties of VMATCH. Further, it is useful to determine the benefits of component-specific mapping on a more standard reconfigurable architecture such as an FPGA. In addition to yield enhancement, we demonstrate results showing delay, energy, and energy at a fixed delay improvements.

Summary: While component-specific mapping is not a new idea, this work attempts to quantify the benefits of per-chip customization using modern FPGA architectures, predictive technology models, and accurate models of V_{th} variation. Unlike prior work other than VMATCH, we focus our efforts on routing because of the significant impact of interconnect on both the delay and energy of mapped circuits.

Chapter 3

Modeling

To quantify the energy savings of component-specific mapping under variation, we need to model the area, delay, and energy of designs mapped to an FPGA architecture. The device and circuit models must accurately calculate delay and energy over a wide range of V_{dd} and V_{th}, while the CAD software must perform efficient, consistent mappings that can be modified such that knowledge of delays and defects can be integrated into the mapping algorithms. This chapter describes the process used to construct accurate device and circuit models using HSPICE E-2010.12-SP1 [2], and the necessary modifications to add variation models, energy models, and router stability to VPR 5.0.2 [95], the standard academic FPGA CAD mapping tool.

3.1 Devices and Circuits

3.1.1 Motivation

To model the delay and energy of a mapped FPGA design, one first needs to determine the delay and energy of the basic FPGA circuit elements: the LUT (Figure 2.2b) and the switch (Figure 2.3b). CAD mapping software can then take these delay and energy values and use them to perform steps like technology mapping, placement, routing, timing analysis, and energy analysis to determine the final mapped delay and energy of a design.

Typically, to calculate delay and energy of both LUTs and switches, a static set of SPICE simulations are performed under worst-case process corner conditions. The CAD software then maps assuming this uniform, worst-case delay and energy for every LUT and switch. Using worst-case corners results in a design that performs slower and dissipates more energy than may be needed for a particular chip and its unique V_{th} map.

Component-specific mappings allows for per-chip customization, but the CAD tools must know the precise delay of each circuit element and must be able to accurately calculate energy under a wide range of V_{dd} and V_{th}. As this work relies on simulating component-specific mapping (rather

than using actual physical measurements from a real chip), we must develop a technique to calculate delay and energy of FPGA circuits as a function of V_{dd} and V_{th}.

Performing SPICE simulations at CAD runtime for each of the hundreds of thousands of circuit elements in an FPGA, with their own unique V_{th} values, at a particular V_{dd}, and for hundreds of unique chips, is intractable. Therefore, most prior work in modeling FPGAs under variation have taken a simple, computationally efficient approach in calculating delay and energy under variation by using first-order device equations [74, 78]. Equation 2.12 shows how to calculate the propagation delay τ_p of a gate given V_{dd}, V_{th}, and a host of device parameters.

This method, while simple and fast, can result in highly inaccurate delays at low voltages [94]. There are numerous possible sources of error: inaccurate device constants, second and third order device effects (e.g., drain-induced barrier lowering, V_{th} roll off), and inaccuracies in modeling the near-threshold region of operation. Our simulations confirmed the inaccuracy of modeling low V_{dd} circuits under variation with first order equations.

Instead of trying to simulate the delay and energy of each LUT and switch for their exact, unique values of V_{dd} and V_{th}, we instead simulate these circuits across a regular, closely spaced range of values and store the calculated delay and energy in a lookup table. To evaluate a particular gate, we perform interpolation between simulated points. The interpolation can be either linear or cubic, depending on which technique yields the most accurate result (typically linear interpolation is better given the fact that interpolated cubic curves tend to oscillate on outlier points and boundary conditions). This is a fairly standard technique for modeling gates under variation [19], but has not yet been developed for the academic FPGA CAD flow [91]. We observe that our models have a delay and energy prediction error of less than 10%.

The following sections will provide more details on how we simulate each of the primitive FPGA circuits and how variation impacts delay, energy, and defects.

3.1.2 Parameter Extraction

For this work we use SPICE models from the Predictive Technology Models (PTM) [128]. Level 54 Models are provided for the 45 nm, 32 nm, 22 nm, 16 nm and 12 nm nodes, both for high performance (HP) and low-power (LP) technologies.

The PTM models come with assigned values of nominal V_{dd} and predicted $\sigma_{V_{th}}$; however, some key technology parameters that are unavailable from simply examining the SPICE model, and must be extracted to accurately model both delay and energy. Table 3.1 compiles all the relevant parameters for each technology node.

Capacitance: To calculate energy and delay we need to extract values of drain, source, and gate capacitance for both NMOS and PMOS devices. Figure 3.1 shows the circuit used for extraction,

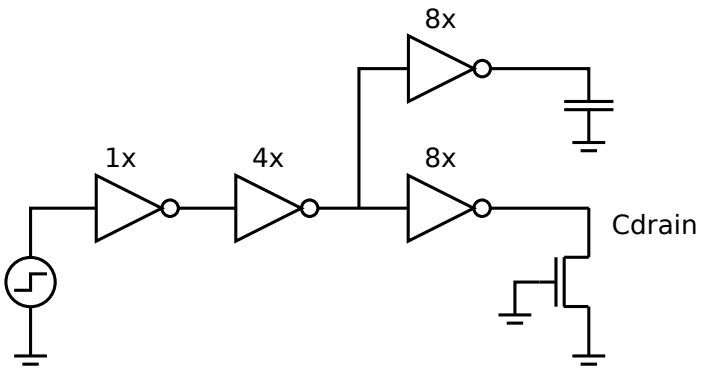


Figure 3.1: Drain capacitance extraction circuit

specifically for C_{drain} [119]. An input signal is conditioned through 2 inverters and then split to inverters that drive a static capacitor and the drain of the NMOS device. The HSPICE optimization feature is used to tune the static capacitance until the delay through the inverters driving the capacitor and the NMOS terminal are the same. A similar procedure is used to find C_{source} and C_{gate}. For C_{gate} the diffusion area and perimeter are set to zero to extract the correct value.

Threshold Voltage: To extract the nominal threshold voltage, we use the constant current method that defines the threshold voltage as V_{gs} where $I_{ds} = 0.1\mu A \times W/L$. This is not the most accurate method for determining V_{th} as the choice of critical current is somewhat arbitrary; however, the exact definition of V_{th} is inherently arbitrary, so it is most beneficial to use a commonly agreed upon technique for consistency [129]. Further, we effectively only use extracted V_{th} to estimate variation as a percentage for Table 3.1.

Threshold Voltage Variation: There are two primary ways to calculate $\sigma_{V_{th}}$. The first is to calculate V_{th} and then calculate $\sigma_{V_{th}}$ using a fixed percentage obtained from the ITRS [3]:

$$\sigma_{V_{th}} = \mu_{V_{th}} \times V_{th} \text{ Percent Variation} \tag{3.1}$$

The second, more widely accepted technique is to calculate a technology specific constant Avt [117]:

$$Avt = \sqrt{\frac{qT_{ox}\left(V_{th} - V_{fb} - 2\phi_b\right)}{3\epsilon_{ox}}} \tag{3.2}$$

$$\sigma_{V_{th}} = \frac{Avt}{\sqrt{WL}} \tag{3.3}$$

The correct values of Avt are provided with the PTM models.

Supply Voltage: The nominal values of V_{dd} are also supplied with the PTM models.

Table 3.1: Predictive technology model parameters

Parameter	High Performance					Low Power				
	45	**32**	**22**	**16**	**12**	**45**	**32**	**22**	**16**	**12**
$V_{dd,NOMINAL}$ (V)	1.10	1.00	0.95	0.90	0.85	1.00	0.90	0.80	0.70	0.65
$V_{th,NMOS}$ (mV)	220	243	285	332	399	356	373	385	417	456
$-V_{th,PMOS}$ (mV)	242	266	301	372	412	352	379	412	460	467
$1\sigma_{V_{th,NMOS}}$ (mV)	17.5	24.0	32.3	37.2	43.7	23.1	31.1	45.5	57.7	77.7
$1\sigma_{V_{th,PMOS}}$ (mV)	21.9	29.2	36.4	42.7	48.4	24.2	32.6	47.2	64.2	77.7
$\frac{3\sigma_{V_{th,NMOS}}}{\mu_{V_{th,NMOS}}}$ (%)	23.9	29.6	34.0	33.6	32.9	19.5	25.0	35.5	41.5	51.1
$\frac{3\sigma_{V_{th,PMOS}}}{\mu_{V_{th,PMOS}}}$ (%)	27.1	32.9	36.3	34.4	35.2	20.6	25.8	34.4	41.9	49.9
$C_{drain,NMOS}$ $(fF/\mu m)$	1.46	1.43	1.41	1.40	1.38	1.45	1.41	1.38	1.00	1.34
$C_{source,NMOS}(fF/\mu m)$	1.29	1.25	1.23	1.22	1.20	1.27	1.23	1.00	1.18	1.34
$C_{gate,NMOS}$ $(fF/\mu m)$	1.08	0.97	0.88	0.82	0.77	1.18	1.07	0.92	0.85	0.80
$C_{drain,PMOS}$ $(fF/\mu m)$	1.46	1.44	1.42	1.42	1.41	1.44	1.42	1.39	1.38	1.37
$C_{source,PMOS}(fF/\mu m)$	1.29	1.26	1.24	1.23	1.22	1.27	1.24	1.21	1.20	1.19
$C_{gate,PMOS}$ $(fF/\mu m)$	1.06	0.95	0.86	0.80	0.76	1.20	1.06	0.91	0.84	0.80

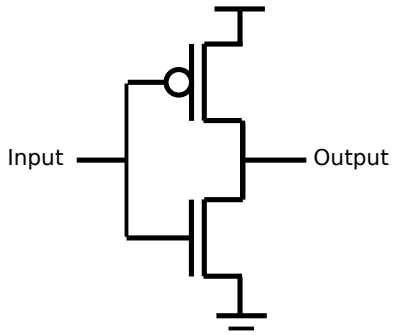


Figure 3.2: CMOS inverter

3.1.3 Inverter Circuit

The inverter (Figure 3.2) is the most frequently occurring logic circuit in the FPGA. Several inverters are used in each switch (Figure 2.3b), and inverters are also used to buffer the input and output to LUTs.

Our general simulation methodology is to determine the critical input parameters for each circuit, determine the range and interval for the values of those parameters, and then simulate all combinations of these values. Then, using the generated lookup table we perform n-dimensional interpolation to calculate delay and energy for arbitrary values of the input parameters.

The standard possible input parameters for simulating an inverter are as follows:

- V_{dd}**:** We simulate inverters from $V_{dd} = 0.1$–$1.0V$ at 10 mV intervals.
- $V_{th,NMOS}$ **&** $V_{th,PMOS}$**:** We simulate using $V_{th,NMOS}$ and $V_{th,PMOS}$ offsets from -0.4–$0.4V$ at 10 mV intervals.

- W_{PMOS} & W_{NMOS}**:** For the PTM models $W_{PMOS} \approx W_{NMOS}$ across all technologies to obtain equal rise and fall times [128]. For our simulations we simulate a minimum sized inverter and scale energy and delay up for larger sizes.

- L_{PMOS} & L_{NMOS}**:** We use all minimum length transistors.

- **Input slew rate:** The rise and fall time of the input to the inverter can have an impact on the output propagation delay. However, this has a second-order impact on delay, and requires timing analysis that uses the rise and fall times of prior gates to calculate the propagation delay of a given gate. Modeling slew would significantly multiply the number of HSPICE simulations needed to build our circuit models. While slew rate can have an important effect on calculating the delay of subthreshold circuits (between 10–20%) [77], we leave this modeling to future work, and instead using a simple derating constant to account for slew rate.

Our final inverter model is then interpolated over our values of V_{dd}, V_{thn}, and V_{thp}, which constitute approximately 500K datapoints.

Output parameters of interest include:

- **Propagation delay (τ_p):** We simulate propagation delay as the delay between the input rising to 50% and the output falling to 50%.

- **Rise and fall time (τ_r and τ_f):** Rise and fall times of the output are measured between the 80% and 20% points.

- **Equivalent resistance (R_{eq}):** We simulate the inverter driving a fixed capacitance that is approximately 100× the intrinsic output capacitance, and calculate $R_{eq} = \tau_p/C$. Equivalent resistance is used for Elmore delay calculations in VPR [95].

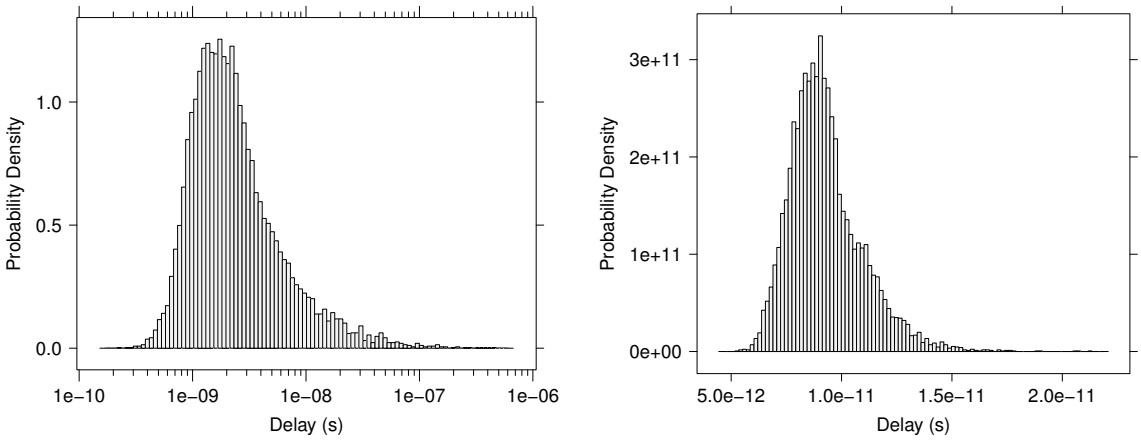
- **Dynamic energy:** Instead of simulating dynamic energy dissipation, we simply calculate it from CV^2 which is highly accurate.

- **Subthreshold leakage current (I_{sub}):** This is measured by the current drawn through the power supply.

- **Short-circuit current:** We capture the short-circuit current dissipated by the inverter, but it is small enough that it can be ignored.

- **Gate leakage current:** For the PTM models, gate leakage current is negligible compared to subthreshold leakage current due to high-κ metal gates.

Figure 3.3 plots the delay distribution of a minimum sized FO4 inverter at 22 nm under variation for nominal and subthreshold V_{dd}'s using an HSPICE Monte Carlo simulation with 10,000 samples



(a) $V_{dd} = 0.3V$ (b) $V_{dd} = 0.8V$

Figure 3.3: Inverter delay distribution (22 nm LP, 10,000 samples)

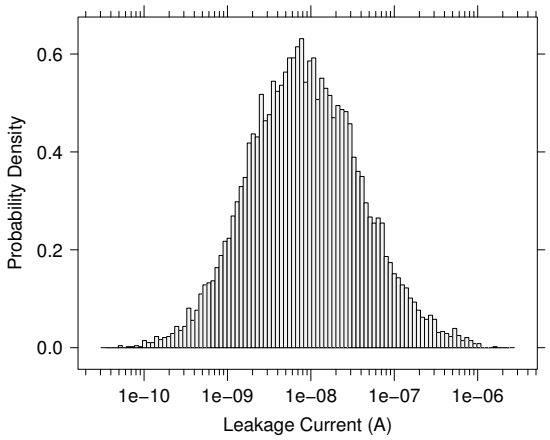


Figure 3.4: Inverter leakage distribution (22 nm LP, $V_{dd} = 0.8V$, 10,000 samples)

(approximately 4σ). We see that the delay spread is significant (several orders of magnitude) at low voltages. Figure 3.4 plots the subthreshold leakage current distribution, which also spans several orders of magnitude.

3.1.4 Switch Circuit

Figure 2.3b shows the design of a unidirectional, 3-input FPGA switch. The switch consists of an inverter for each input (generally referred to as stub inverters, as they serve to isolate the capacitance of the switch from upstream drivers), a flat NMOS pass transistor multiplexer, and a inverter at the output that drives a long wire segment that may span multiple CLBs.

To calculate the delay of this circuit, we can simply use our inverter model combined with an NMOS pass transistor model (simulated at the same values of V_{dd} and V_{th} as the inverter). It is important to note that only one pass transistor of the multiplexer is turned on; therefore, we can calculate the delay of this circuit as follows, where N is the number of inputs to the multiplexer:

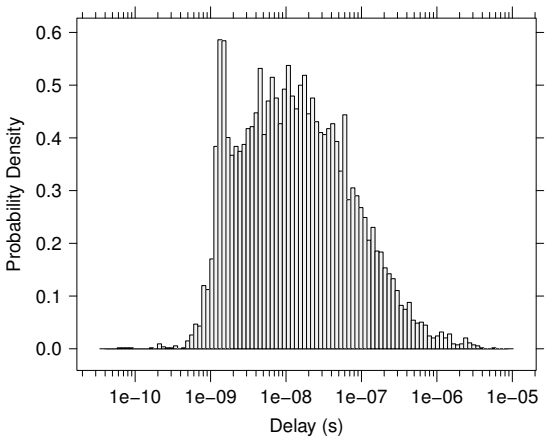
$$\begin{aligned}\tau_p = {} & R_{eq,INVERTER} \times (C_{drain,PMOS} + 2C_{drain,NMOS}) \\ & + (R_{eq,INVERTER} + R_{eq,NMOS}) \times (NC_{source,PMOS}) \\ & + R_{eq,INVERTER} * (C_{drain,PMOS} + C_{drain,NMOS} + C_{wire} + C_{downstream})\end{aligned} \tag{3.4}$$

Figures 3.5a and 3.5b plot the delay distribution of a minimum sized NMOS passgate at 22 nm under variation for different V_{dd}'s using an HSPICE Monte Carlo simulation with 10,000 samples (approximately 4σ). We see that the delay spread is significant. However, it is possible to substantially improve both the nominal delay and the delay spread of the passgate by overdriving the gate voltage. The gate of the pass transistors in the switch multiplexer are driven by configuration SRAM bits (Figure 2.3b), which can be engineered to output a higher voltage. Figures 3.5c and 3.5d show the same NMOS pass gate delay distribution with the gate overdriven by 50%. We see a much smaller delay spread, and therefore the multiplexer delay becomes less significant than the delay of the inverter driving a wire segment under variation.

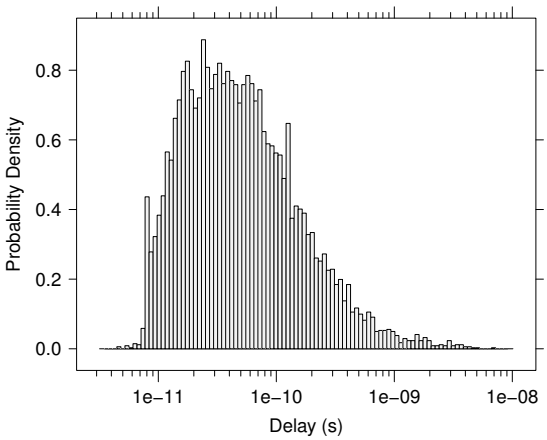
3.1.5 LUT Circuit

Figure 2.2b shows the design of a 3-input LUT. The circuit consists of 2-to-1 pass transistor multiplexers connected in a tree. To calculate the delay of this circuit, we create a delay model for the primitive 2-to-1 pass transistor multiplexer shown in Figure 3.6. We characterize the delay of this circuit as a function of V_{dd}, $V_{th,NMOS1}$, and $V_{th,NMOS2}$ (with the same values as our inverter model). As the inputs to the pass transistor gates are driven by signals and not SRAMs, they cannot be overdriven like in the FPGA switch case.

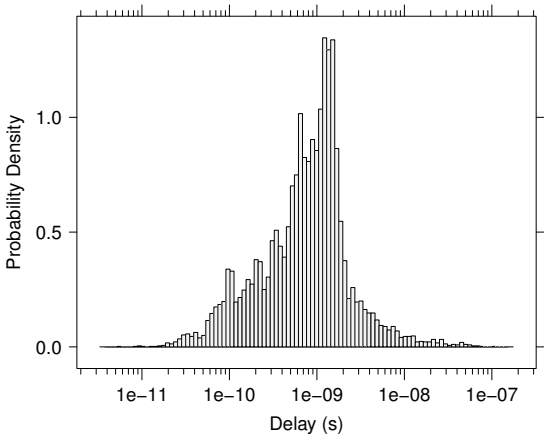
Calculating the delay of the full pass transistor tree can be difficult due to V_{th} variation inducing



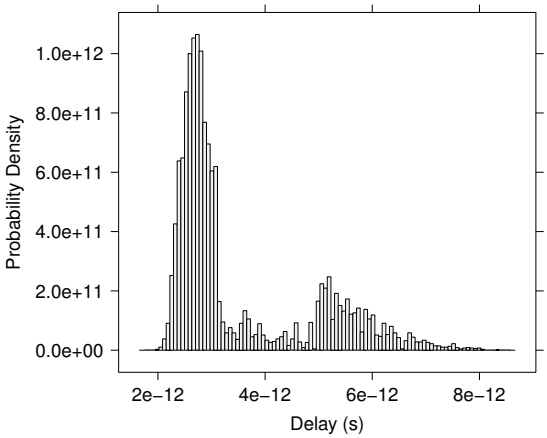
(a) $V_{dd} = 0.3$, Nominal gate input



(b) $V_{dd} = 0.8$, Nominal gate input



(c) $V_{dd} = 0.3$, Overdriven gate input



(d) $V_{dd} = 0.8$, Overdriven gate input

Figure 3.5: NMOS pass gate delay distribution (22 nm LP, 10,000 samples)

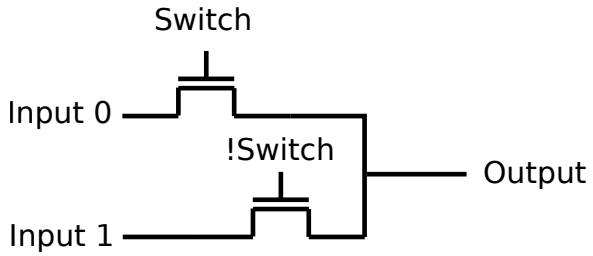


Figure 3.6: 2-to-1 multiplexer

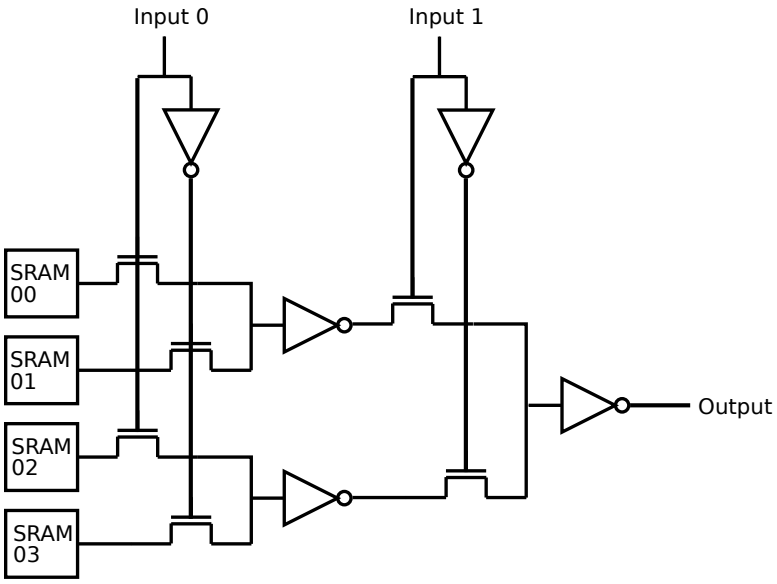


Figure 3.7: 2-input LUT with buffering

different voltage drops across each of the pass transistors. Further, the delay of the LUT can suffer significantly if the signals internal to the tree are not re-buffered. Increased failures can also occur without this re-buffering. To both improve the reliability of the LUT and increase the accuracy of our circuit model, we re-buffer after every 2-to-1 mux primitive in the LUT tree (Figure 3.7).

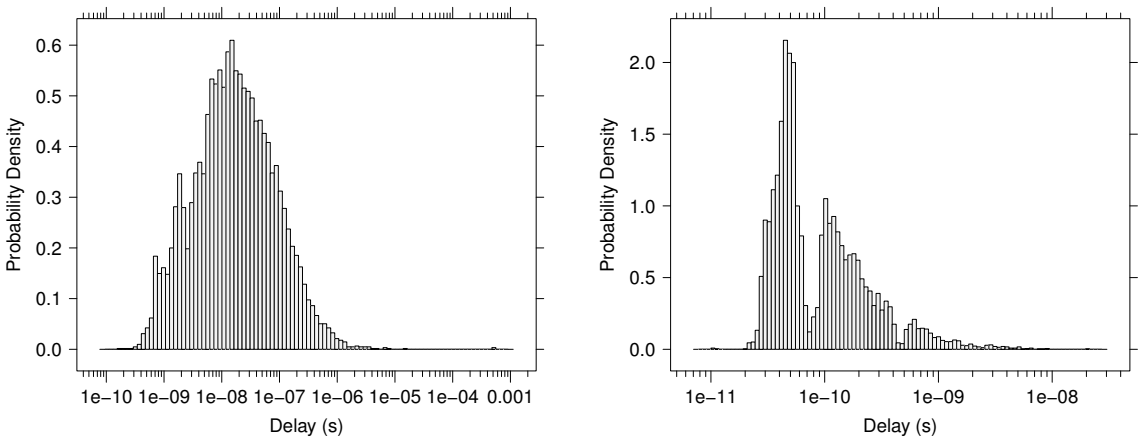
Figure 3.8 show the delay distribution of the 2-to-1 mux (without buffer) at 22 nm under variation for different V_{dd}'s using an HSPICE Monte Carlo simulation with 10,000 samples (approximately 4σ).

3.1.6 SRAM Circuit

SRAMs are key circuits in FPGAs as every configurable logic and interconnect element is controlled by an SRAM bit. Figure 3.9 shows the design of a 6T SRAM cell.

The energy and reliability impact of SRAMs can be significant in FPGAs. If we examine a 3-input FPGA switch (Figure 2.3b), we see that while the switch logic has 4 inverters (3 stubs and one driver), there are 3 SRAM bits with 2 cross coupled inverters each, for a total of 6 inverters. Therefore, there are more inverters leaking in SRAM configuration bits than there are in inverters used for logic. Further, if an SRAM fails, one can lose the ability to configure a given switch for a particular input.

However, the way that SRAMs are utilized in FPGAs differs significantly to their typical use in large, dense, random access memory arrays, which affords different techniques to alleviate energy and reliability problems. Configuration SRAMs in FPGAs are only written once, at configuration time. Further, they are never read like a normal SRAM—there is no charging of bitlines and activation



(a) $V_{dd} = 0.3V$ (b) $V_{dd} = 0.8V$

Figure 3.8: 2-to-1 mux delay distribution (22 nm LP, 10,000 samples)

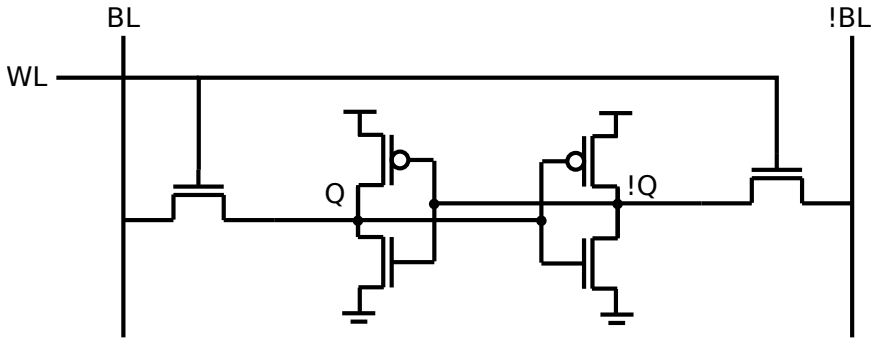
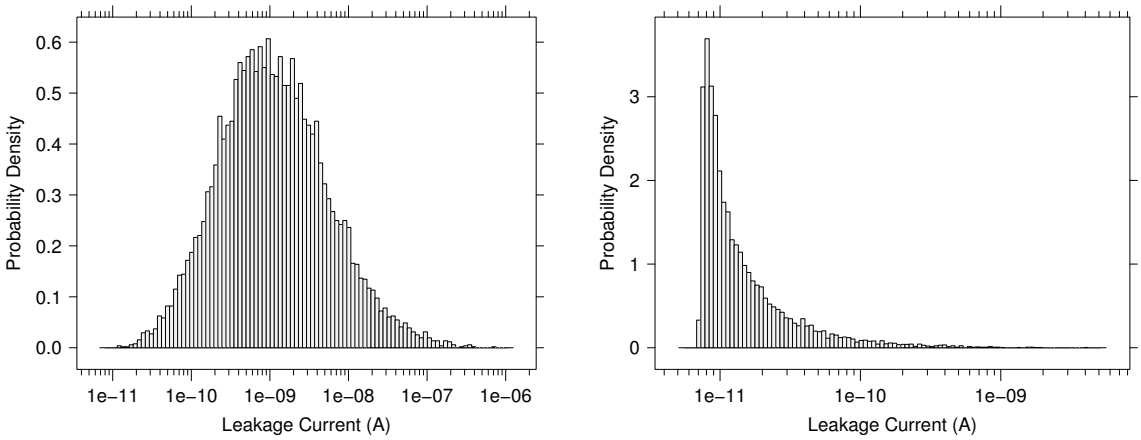


Figure 3.9: 6T SRAM cell



(a) Nominal V_{th}

(b) 75% Higher V_{th}

Figure 3.10: SRAM leakage distribution (22 nm LP, 10,000 samples)

of word lines. Instead, the internal value of the SRAM can just be directly connected to either the gate in the pass transistor mux of a switch, or to the drain of a pass transistor mux in a LUT.

There are several implications to this mode of operation. First, FPGA SRAMs do not necessarily need both access transistors, and instead can have a 5T cell with a single sided write. Second, one can engineer both the V_{th} and V_{dd} of the SRAM without regard to read or write speed, as the cell is never effectively read and only written once. Using a higher V_{th} can dramatically reduce the leakage energy due to SRAMs [115], while using a higher V_{dd} will allow our NMOS pass transistors in the switch circuit to be overdriven. Third, the failure modes of the SRAM are greatly reduced (as will be described in the next section), which will increase the reliability of FPGA SRAMs.

Figure 3.10 plots the leakage distribution of a minimum sized SRAM 22 nm under variation for nominal V_{th} and for a V_{th} boosted by 75%, using an HSPICE Monte Carlo simulation with 10,000 samples (approximately 4σ). For the increased nominal V_{th} case we increase $\sigma_{V_{th}}$ proportional to the trends in Table 3.1. We see that increasing V_{th} dramatically reduces the magnitude of SRAM leakage. This multi-V_{th} technique is the same one that is used in commercial FPGAs [56].

3.1.7 Defect Rates

While we have shown how to accurately model both the delay and energy of switch and LUT circuits, Section 2.3 described another important effect of variation that must be modeled: defects. For example, given enough V_{th} variation, an inverter can fail to switch if the off-current of one transistor is greater than the on-current of the other transistor (Equation 2.14). This type of defect is even more likely at low V_{dd} and for scaled technologies with high variation [26].

However, this failure condition for an inverter is actually conservative. The inverter can also

practically be considered defective if it is no longer restoring; i.e., it has a gain < 1, or it has zero static noise margins. To model this effect, we simulate the voltage transfer characteristic (VTC) for inverters across the same parameters as our delay simulation. We then differentiate the VTC curve and determine if the inverter has no gain or if noise margins are violated (similar to the technique in [61]). This provides a more accurate measure of failure.

Each of our other circuit primitives (NMOS pass transistor, 2-to-1 multiplexer, SRAM bit) is subject to failure; however, these failures can be mitigated through a variety of techniques.

The NMOS pass transistor can fail if the threshold voltage varies enough that the transistor can no longer be turned on properly. However, we saw that the pass transistor can have its gate overdriven, which will reduce this failure rate as a higher gate voltage will turn on a transistor with a higher V_{th} due to variation.

The 2-to-1 multiplexer can fail just like the NMOS pass transistor, but here we cannot overdrive the gate terminals. Therefore, when switching the select bit of the mux, one or both of the NMOS pass transistors might fail to switch the output to its correct value. However, we can observe that the inputs to the multiplexer can only have four states: 00, 01, 10, and 11. In the 00 state, the multiplexer cannot fail—there is no voltage will flip the output to 1. In the 11 state, the multiplexer is highly unlikely to fail, as V_{dd} must be low enough and V_{th} high enough that the threshold drop across the transistor will lower the output voltage enough to violate noise margins. In the 01 or 10 states the multiplexer is most likely to fail, as either transistor failing to switch will make the gate defective. We will see in Chapter 5 that this asymmetry in failure modes enables us to perform optimizations that will actively seek configurations of the 00 or 11 states to reduce failure rates.

When considering the failure rate of SRAMs, traditionally SRAMs have been known to have four major modes of failure:

1. **Read failure:** Flipping the stored value while reading.
2. **Write failure:** Inability to write the cell.
3. **Access time failure:** Reading the cell too slowly, violating timing.
4. **Hold failure:** Losing the value of the cell during operation.

The most likely SRAM failure modes under variation are read and access time failures [84]; however, because FPGA SRAMs are never read, they cannot have read related failures. Write failures can be avoided by overdriving or underdriving the input to the cell, which can be afforded since this operation is only done once without significant time constraints. Hold failures are the least likely of the four failure modes; however, they may still occur in FPGA SRAMs if the V_{th} map on the cell is such that the cell cannot hold a proper value (one or both inverters fail).

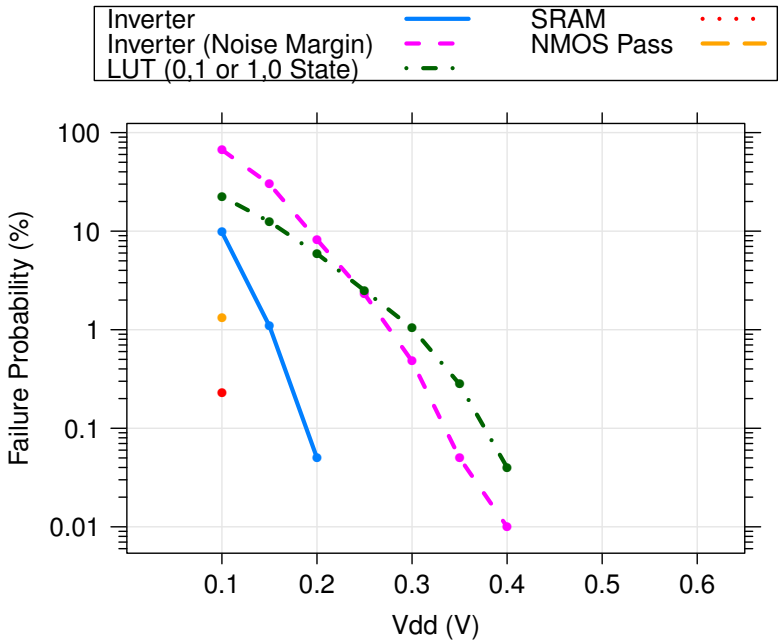


Figure 3.11: Primitive circuit failure rates (22 nm LP, 10,000 samples)

Figure 3.11 plots the failure rates for each of our primitive circuits as a function of voltage at 22 nm using using an HSPICE monte carlo simulation with 10,000 samples (approximately 4σ). For the inverter we plot both the conservative and noise margin related failure estimates. For the NMOS pass transistor we plot the failure rates without overdriving the gate; with overdriving we observe no failures. For the 2-to-1 multiplexer we plot the failure rates 01/10 states, as we observe no failures in the 00 or 11 states. We see that through careful design we can significantly reduce the failure rates of all of our primitives except the inverter. We also see that SRAM failure rates are negligible, meaning that we can ignore SRAM failures during mapping.

3.2 CAD

3.2.1 VPR: Variation and Energy

Figure 3.12 shows the standard academic FPGA mapping flow using VPR 5.0.2 [95]. Technology mapping is performed using ABC [83], followed by clustering with T-VPack [95], and placement and routing both using VPR.

We modify VPR to integrate our HSPICE device and circuit models to calculate both delay and energy as a function of V_{dd} and V_{th} variation. Prior work has integrated both variation [74] and energy [91] models into VPR. However, the variation models used simple device equations that are inaccurate for large amounts of V_{th} variation and low V_{dd} , and at the time of this work the energy

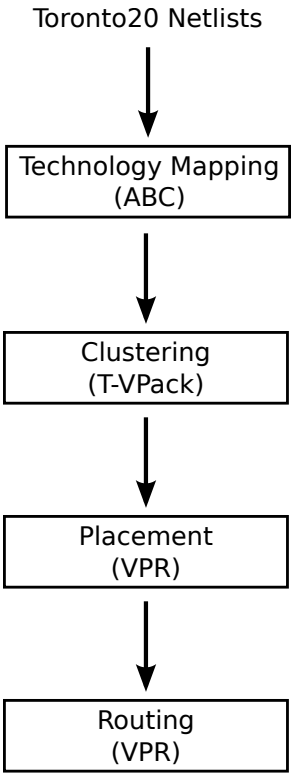


Figure 3.12: Standard FPGA mapping CAD flow

models only existed for VPR 4.0, which utilized older, bi-directional interconnect. However, we leverage some prior work in computing FPGA dynamic energy by calculating switching activity (α in Eq. 2.5) using the ACE 2.0 switching activity estimator [63] with random (50%) input probabilities.

With variation enabled, every transistor is assigned a randomly generated V_{th} sampled from a Gaussian distribution. Routing can either be performed using full knowledge of actual circuit delays or no knowledge. The no knowledge, delay-oblivious case can then be evaluated post-route based on the actual delays. While we modify VPR to measure energy, we do not change VPR's cost function to target energy minimization—we simply provide the router with delay information.

3.2.2 Timing-Target Routing

The routing algorithm used in VPR 5.0.2 (and in all modern FPGA CAD software) is PathFinder [99]. PathFinder is known to introduce experimental noise by producing inconsistent results in routing. Rubin et al. [99] showed that innocuous perturbations of initial conditions can cause critical path delays to vary over ranges of 17–110%, and that it is not uncommon for VPR/PathFinder to settle for solutions that are >33% slower than necessary. This is for a standard architecture at nominal V_{dd} and no V_{th} variation.

Figure 3.13 demonstrates this effect, plotting the ratio of the routed delay of two architectures

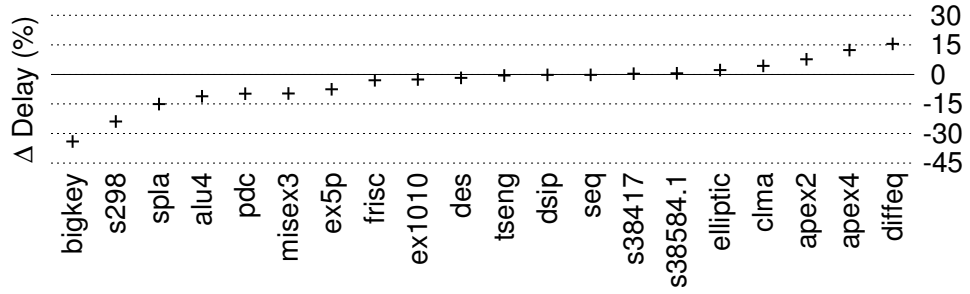


Figure 3.13: Percent delay improvement for faster-wire architecture over uniform architecture for the Toronto20 benchmarks [99]

across a series of benchmarks. The first architecture is the standard 4×4 cluster architecture distributed with VPR. The second architecture is identical to the first except half of the wires are faster by a trivial amount, which should only result in mapped designs being 0.5% faster. However, we see a range of delay changes from -34% to $+15\%$.

This effect is greatly magnified when mapping to resources that each have different delays; moreover, with high variation and low V_{dd} these delays can vary by several orders of magnitude.

Algorithm 1: Delay Target Search [99]

```
T_current=CongestionObliviousRoute;
max = min = T_current ;                          /* Initial lower bound */
repeat                                           /* Find initial upper bound */
  | max *= 2;
until try_route(max);
stage=0;
repeat                                                          /* Refine */
  | retry = 0;
  | stage++;
  | success=false;
  | repeat
  | |  T_target = (max + min)/2;
  | |  if ((T_current = try_route(T_target))!=FAIL) then
  | |  |  T_target+ = (max − T_target)/1000;
  | |  |  success=true;
  | until retry++ ¿= retries or success ;
  | if success then
  | |  max = T_current
  | else
  | |  min = T_target
until max <= min ∗ (1 + target precision)
      or stage >= max_stages;
```

To minimize router noise we use the timing-targeted router modification to PathFinder proposed by Rubin *et al.* The conventional PathFinder algorithm attempts to resolve congestion while minimizing delay by cyclically routing nets and ripping up prior routes, with each route being performed

by a least-cost routing function. The cost of a resource n is:

$$\alpha_{ij} \times d_n + (1 - \alpha_{ij}) \times (b_n + h_{n,t}) \times p_n \tag{3.5}$$

Where α_{ij} is criticality of net $i \rightarrow j$, d_n is the delay of the resource, b_n is the base cost of the resource, $h_{n,t}$ is the congestion history of the resource, and p_n is the pressure factor. To succeed, PathFinder must balance the congestion cost $b_n + h_{n,t}$ with the delay cost d_n by using the criticality factor α_{ij}, which can prove to be difficult, resulting in routes with unnecessarily high delays.

Instead of simultaneously trying to minimize congestion and delay, the timing-target algorithm simply changes the heuristic in PathFinder from an optimization problem to a decision problem. Algorithm 1 shows the timing-target algorithm. Here, each routing step attempts resolve congestion at a target delay—if a valid route is not found, a slower delay is attempted. The outer loop of the algorithm performs a binary search on the target delay in order to minimize the final routed delay.

We cannot overstate the importance of making this change to VPR when routing with V_{th} variation at low V_{dd}.

Chapter 4

Delay-aware Routing

In this chapter we will provide baseline results that demonstrate the delay and energy benefits of component-specific mapping. Specifically, we will focus on delay-aware versus delay-oblivious routing. We will quantify the delay and energy margins induced by variation and routing oblivious to delays, and quantify by how much those margins can be reduced through delay-aware routing. In the following chapters we will build our extensive optimizations upon this baseline comparison.

One important note is that this chapter and most of the following sections will only focus on the impact of variation in interconnect. As interconnect is the dominant source of area, delay, and energy in an FPGA, random V_{th} variation most impacts switches and routing. Hence, we primarily explore using delay-aware routing to intelligently assign interconnect resources. Section 5.4 will separately quantify the impact of LUT variation and specialized techniques for LUT variation tolerance.

The results outlined here a very similar to those initially presented in [82]. However, several changes have been made to our models and methodology and our results have been updated.

- Improved device modeling (higher resolution models, Avt variation methodology, inverter noise margin failure modeling).
- Improved interconnect modeling that accurately scales wire length according to area.
- Targeted architecture parameters for low energy instead of high performance.

4.1 Experimental Setup

As previously described (Chapter 3), we will use HSPICE device and circuit models to determine the delay and energy of primitive FPGA circuits. As this work primarily focuses on reducing energy dissipation, we will use the low-power (LP) PTM models.

VPR provides a parametric FPGA architecture where architectural features can be tuned, and a full CAD flow where the area and delay of fully mapped benchmarks can be measured. Section 2.4.2

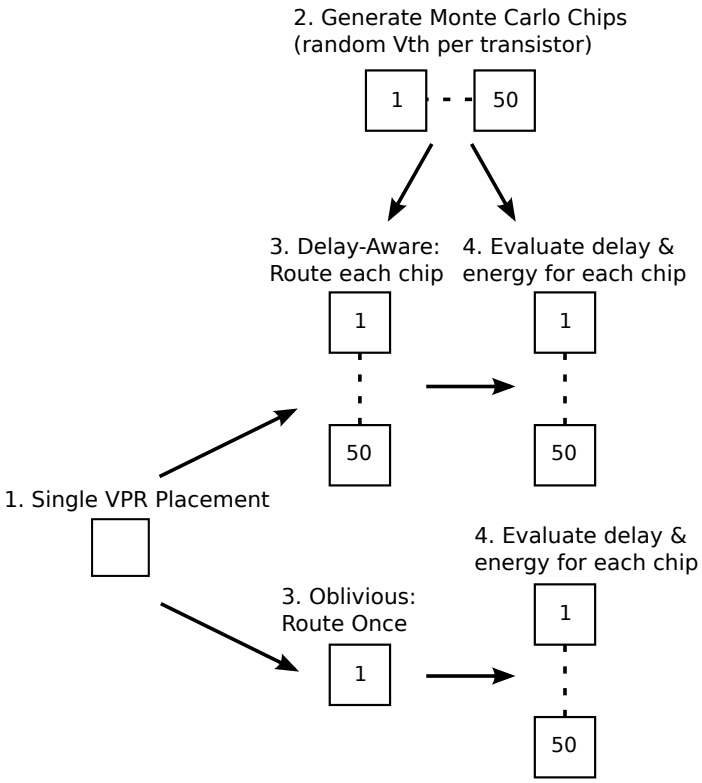


Figure 4.1: Experimental CAD flow

described prior work in determining energy-optimal values for several key architectural parameters [58, 123]; we found similar values through our own internal simulations. The most important architectural parameters of VPR used in this work include the following:

- **Cluster Size (N):** The number of LUTs per CLB = 4.
- **LUT Size (k):** The number of inputs per LUT = 4.
- **Cluster Pins:** The number of input and output pins attached to the CLB = 10 and 4, respectively.
- **Interconnect directionality:** Unidirectional.
- **Segment Length (L_{seg}):** The number of CLBs that a wire segment spans = 1.
- **CBox input connectivity (F_{cin}):** The fraction of channel segments that can connect to the CLB input pins = 0.25.
- **CBox output connectivity (F_{cout}):** The fraction of channel segments that can connect to the CLB output pins = 0.25.

Table 4.1: Toronto20 benchmark characteristics

Benchmark	CLBs	LUTs	Min Chan Width	Nets	Crit Path Segments	Registered
alu4	18×18	1296	46	866	24	No
apex2	19×19	1444	52	1041	26	No
apex4	17×17	1156	56	878	30	No
bigkey	27×27	2916	26	923	21	Yes
clma	34×34	4624	62	3486	58	Yes
des	32×32	4096	32	1225	29	No
diffeq	16×16	1024	32	695	16	Yes
dsip	27×27	2916	26	704	23	Yes
elliptic	24×24	2304	44	1615	30	Yes
ex1010	31×31	3844	84	2979	53	No
ex5p	15×15	900	48	669	25	No
frisc	25×25	2500	52	1603	42	Yes
misex3	17×17	1156	48	855	23	No
pdc	29×29	3364	68	2378	47	No
s298	16×16	1024	44	643	33	Yes
s38417	30×30	3600	34	2420	22	Yes
s38584.1	31×31	3844	36	2920	26	Yes
seq	18×18	1296	52	994	23	No
spla	27×27	2916	62	1975	41	No
tseng	14×14	784	32	603	14	Yes

We compare delay-aware routing to delay-oblivious routing under variation for the Toronto 20 benchmark set [17]. Table 4.1 show characteristics of each of the benchmarks when mapped to our architecture. The mapping flow for each benchmark is the same as in Figure 3.12. We perform technology mapping with ABC, clustering with T-VPack, and placement in VPR.

Figure 4.1 shows the experimental CAD flow after placement for comparing delay-aware and delay-oblivious routing. A single placement per benchmark is created for all routing experiments. Each data point is obtained by running both routers on a set of 50 Monte Carlo generated chips with V_{th} variation. The 50 chips are routed individually by the delay-aware router, while the delay-oblivious router performs a single, nominal route and evaluates that route across all chips. We report all delay and energy data at the 90% parametric yield point (i.e., we discard the 5 slowest/highest energy chips and report the max delay and energy). With 50 Bernoulli trials the 90% confidence interval for the results reported as 90% yield is 85–95%.

We configure VPR to use timing-targeted routing, and route using 200 iterations and a `-max_crit` value of 0.9999.

For this chapter, to establish an intuitive baseline for comparing delay-aware to delay-oblivious routing, we use a few key optimizations that will be explored in more depth in Chapter 5.

- **Sizing:** All switches use energy optimized sizing for a given V_{dd}. This means that data at different V_{dd}'s may use different switch sizes to obtain minimal energy. For example, at a low

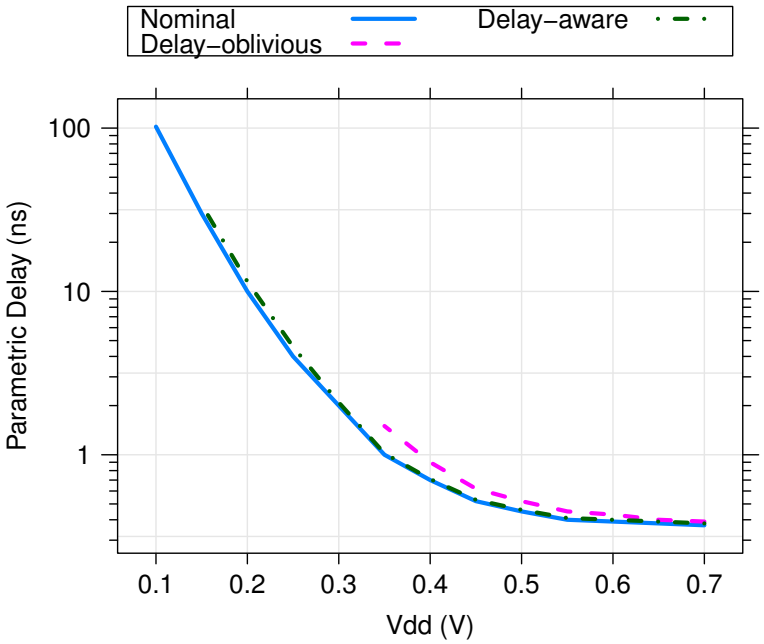


Figure 4.2: Delay vs V_{dd} (`alu4`, 22 nm LP)

V_{dd} variation induced defects may cause a design with minimum sized gates to fail to achieve 90% yield. Therefore, larger gates may be required, but at a cost in energy.

- **Channel Sparing:** All routes are performed at 20% above the minimum channel width required for a congestion-free route. This is a common, low stress routing condition used when not attempting to minimize the number of channels used [110].

- **I/O Sparing:** All routes are performed with 4 extra CLB I/O pins. We will see in Chapter 5 that I/O pins are highly vulnerable to unavoidable defects at low V_{dd}; adding spares alleviates this bottleneck.

4.2 Delay

Our first goal is to characterize the delay margins induced by routing without delay knowledge under variation, and the corresponding delay improvement obtainable by delay-aware routing. Figure 4.2 plots parametric delay as a function of V_{dd}, for nominal, delay-oblivious, and delay-aware routes for the `alu4` benchmark at 22 nm. For the nominal, no variation case, we see that delay increases as V_{dd} is reduced. As delay drops below threshold ($V_{th} \approx 400$ mV), we see that delay increases exponentially, in accordance with Equation 2.12.

As we reduce V_{dd} in the delay-oblivious case, we begin to see functional failures as described in Sec. 2.3 and shown in Figure 3.11. The delay curves end at voltages where the defect rate becomes

too high to achieve 90% yield. At 22 nm there is sufficient variation that enough circuit elements at low V_{dd} fail to switch, which the delay-oblivious router cannot avoid. Further, we see even where the delay-oblivious case is able to achieve 90% yield, it performs slower than the nominal case. We can see the delay margins induced by variation as a function of V_{dd} by comparing the nominal and delay-oblivious curves. At high V_{dd} these margins are typically negligible, less than 2%. From Table 4.1 we see that `alu4` is a fully combinational circuit with a critical path length of 24 segments—hence, delay variation at high V_{dd} is largely eliminated due to path length averaging. However, as we drop the supply voltage these margins increase, up to around 1.5× the nominal delay at 350 mV. Because the delay-oblivious router has no knowledge of real delays it may choose suboptimal, slower resources for the critical path.

We see similar effects for the delay-aware router, where functional failures occur for small switches at low voltages. However, the delay-aware router is able to remain functional for lower voltages through defect avoidance. We will see in the next section that the ability for the delay-aware router to operate at less than half the V_{dd} of the delay-oblivious router yields significant energy savings.

Additionally, delay-aware routing produces faster routes through delay knowledge, performing nearly as well the nominal, no variation case. In this particular case, delay-aware routing can almost completely eliminate delay margins induced by variation.

4.3 Energy

To quantify the energy margins from variation and the savings from delay-aware routing, Figure 4.3 plots parametric energy versus V_{dd}. In the no variation case we observe minimal energy/operation at 150 mV. At high voltages dynamic energy dominates and is reduced quadratically by scaling down V_{dd}; however, at low voltages static energy begin to increase, raising the total energy/operation. As delay increases we spend more time leaking in a single operation.

The delay-oblivious curve shows the same functional yield issues as in Figure 4.2: as V_{dd} is reduced, the delay-oblivious router fails to provide 90% functional yield. Additionally, we see that as V_{dd} is reduced, the energy gap between delay-oblivious routing and the nominal case increases. This is for two reasons: first, the delay-oblivious case must use larger gates to obtain 90% yield, which increases energy. Second, the delay-oblivious route is slower, which increases leakage energy/operation. In the worst case we observe an energy overhead of almost 2× at $V_{dd} = 350$mV. If we compare the energy-minimal nominal point to the energy-minimal delay-oblivious point, we see a difference of approximately 6×.

For the delay-aware case we also see functional failures; however, delay-aware routing extends the range over which smaller gates can function through defect avoidance. As in the nominal case, we see energy/operation minimize, albeit at a higher voltage of 250 mV. When comparing delay-oblivious

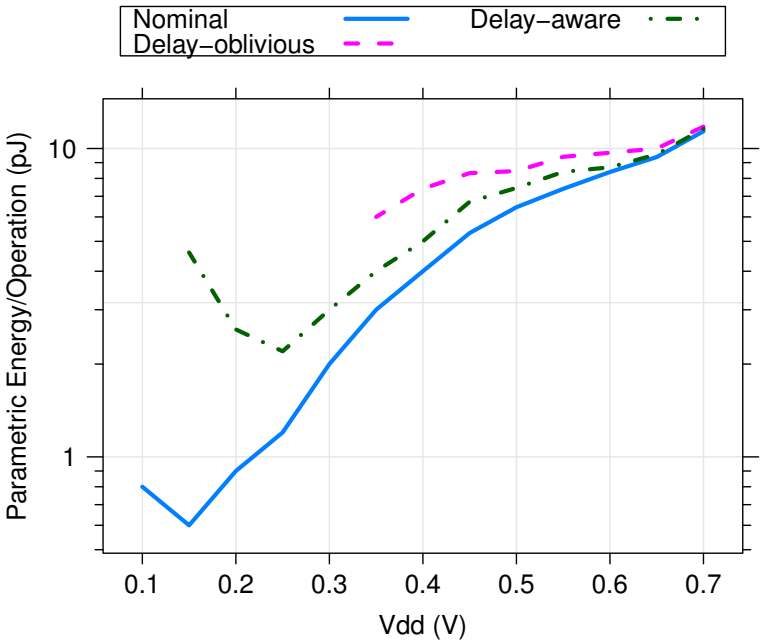


Figure 4.3: Energy/operation vs V_{dd} (`alu4`, 22 nm LP)

to delay-aware, we see that delay-aware routing is able to cut the energy margin nearly in half at 350 mV, but is not able to complete eliminate it. Delay-aware routing is further able to prolong the range over which we can scale down V_{dd} and still reduce energy from 350 mV to 250 mV. We see that the ratio of minimum energy/operation of the delay-aware and nominal cases is $\approx 3\times$, a reduction of $2\times$ over delay-oblivious routing.

4.4 Energy at Target Delay

Minimal energy/operation is critical in many applications; however, the goal of most applications is to minimize energy for a particular performance target. Figure 4.4 demonstrates the energy benefits of delay-aware routing when targeting minimal energy operation under a performance constraint (as opposed to minimal energy ignoring delay). For very high performance requirements ($>$ 2GHz) we see that there is little difference between delay-aware, delay-oblivious, and nominal routing. Low delay routes require high V_{dd}, and at high V_{dd} the impact of variation is minimal. However, as we begin to target slower performance constraints, we begin to see differences in energy dissipation. For slower delay targets delay-aware routing achieves the required cycle times at a lower V_{dd} and hence lower energy/operation than delay-oblivious routing. At 100 MHz we see an energy savings of $\approx 3\times$.

In summary, we see that delay-aware routing is able to achieve the following:

- Nearly eliminate delay margins.

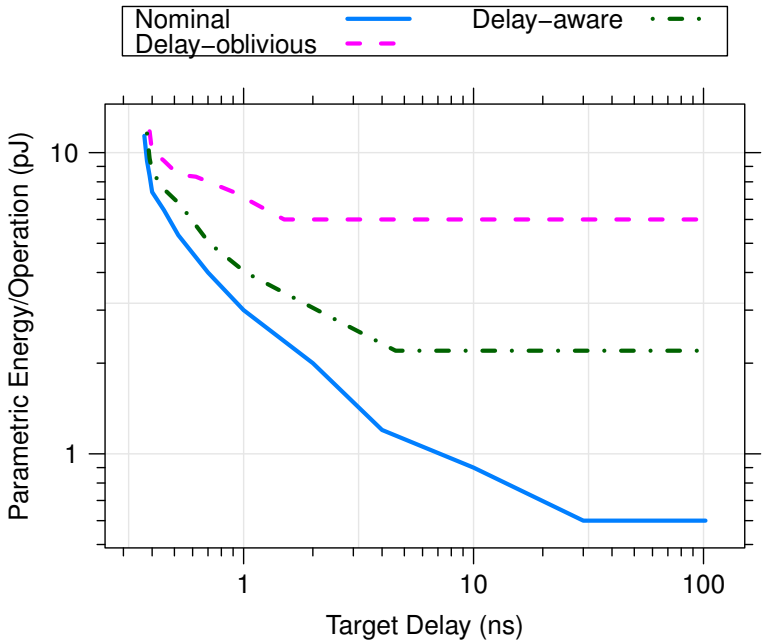


Figure 4.4: Energy/operation vs delay target (`alu4`, 22 nm LP)

- Scale to lower V_{dd} than delay-oblivious routing by avoiding defects.
- Reduce energy/operation by avoiding defects and using smaller gates.
- Reduce energy/operation given a performance constraint.
- Cut energy margins induced by variation in half.

In the next chapter we will explore how to improve these results even further.

Chapter 5

Optimizations

The previous chapter briefly examined some of the energy and delay benefits of component-specific mapping under parameter variation using delay-aware routing. This chapter will focus on techniques that improve those benefits; specifically, we will focus on reducing energy/operation. By scaling down V_{dd} we expect a significant reduction in energy; however, at low V_{dd} parameter variation will induce defects and increase leakage energy. Component-specific mapping can help mitigate these effects, but without the proper optimizations, defect rates may be too high to tolerate, and leakage energy may be large enough to prevent beneficial voltage scaling. We will see that by power gating, carefully sizing gates, adding spares, and remapping LUTs, component-specific mapping can perform even better than our baseline case, for a total of 2.66× energy savings.

As in the previous chapter, the first several sections here will focus on interconnect, the dominant source of energy in FPGAs, by only modeling interconnect variation. In the final section of this chapter we will demonstrate the impact of LUT variation and how to combat failure by using LUT remapping.

5.1 Power Gating

Section 2.4.1 described several techniques developed in prior work to reduce energy dissipation in FPGAs. One standard technique is power gating. Figure 2.8 shows how a single large, high V_{th} sleep transistor can be inserted between the power supply and desired block to be gated. This substantially reduces the leakage energy dissipated in the gated block by taking advantage of the stack effect that reduces the leakage current through transistors in series [87].

Reducing leakage energy is particularly important in low voltage circuits where leakage energy can become the dominant source of total energy. Energy/operation can be expressed as the sum of dynamic and static energy (Equations 2.5 and 2.6). At high voltages dynamic energy/operation dominates. However, as V_{dd} is reduced, the delay of gates and the length of an operation increases (Equation 2.11), causing gates to leak more during an operation, and eventually static

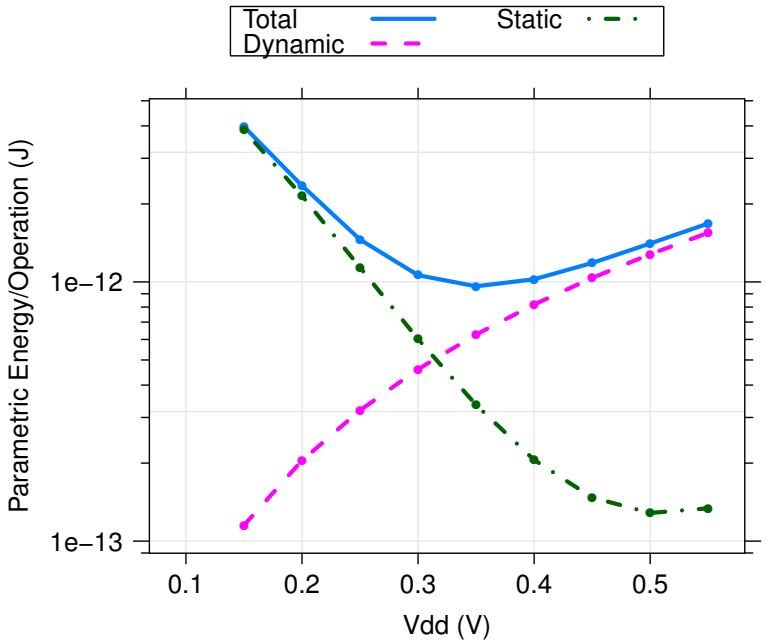


Figure 5.1: Energy/operation vs V_{dd} without power gating (`des`, 22 nm LP, minimum sizes, no variation)

energy increases to point at which it is larger than dynamic energy.

Figure 5.1 shows the energy/operation as a function of V_{dd} for a minimum sized `des` at 22 nm for the nominal, no variation case. We see that as V_{dd} decreases, dynamic energy decreases while static energy increases. An energy minimum is reached around 350 mV which is just below the threshold voltage; below 350 mV there are no energy savings from scaling down V_{dd}.

Ideally, we would like to continue to reduce energy as we scale V_{dd}. Most of the leakage energy in an FGPA is dissipated in the interconnect and SRAM (Figure 2.4b). We showed in Section 3.1.6 because of the unique write once, static read usage mode of FPGA configuration bits, we can reduce SRAM leakage energy by orders of magnitude by using appropriately high V_{th} transistors. This leaves the majority of leakage energy dissipated in the interconnect switches. Fortunately, FPGAs significantly over-provision interconnect to enable maximum routability [32]—in our architecture and for our benchmarks we typically see segment utilization of less than 10%. This means that over 90% of interconnect leakage energy could possibly be eliminated by power gating.

Power gating on interconnect can be implemented at several granularities in FPGAs: at the gate level, at the switch level, and at the tile level [25, 42, 93]. Tile level power gating is a coarse-grained technique that turns off an entire tile (CLB and surrounding SBox and CBoxes) at once. While our segment utilization is typically very low, our tile utilization is high ($> 90\%$ on average) because VPR allocates the minimum square grid of CLBs needed to map a given benchmark. Therefore, power gating on an architecture that is sized to each benchmark would yield minimal savings.

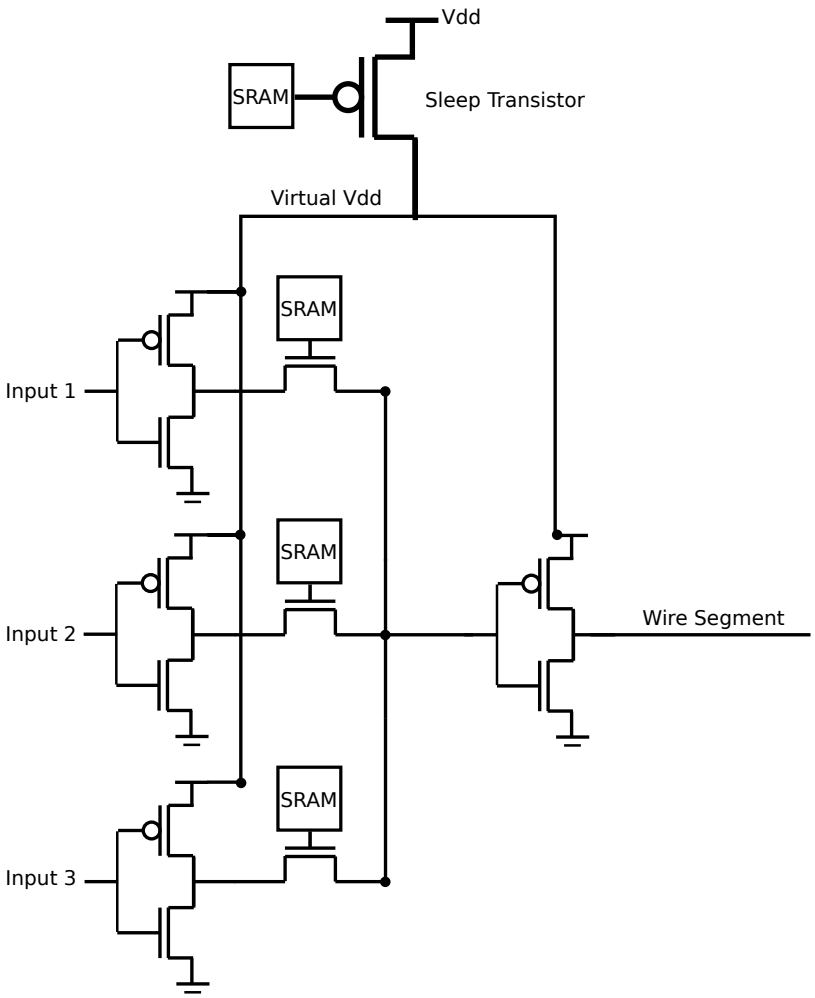


Figure 5.2: Power gated 3-input switch

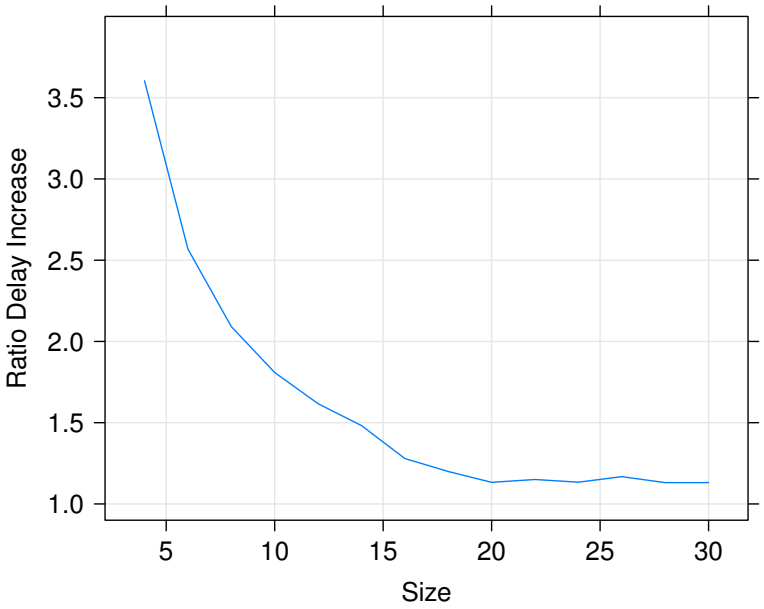


Figure 5.3: Sleep transistor delay as a function of size (22 nm LP, 16-input switch circuit)

Figure 5.2 shows our scheme for power gating, where we gate at the level of an individual switch circuit. Because gate leakage current is negligible in our technology, we can ignore gating the passgate multiplexer, as only the subthreshold leakage in the inverters contribute significantly to leakage energy/operation. For every switch (input stub inverters, multiplexer, output driver) we insert a large PMOS header controlled by an SRAM between the power supply and the switch, creating a virtual V_{dd} for the switch. A similar power gating scheme might insert individual headers at each inverter. In this case the SRAM control bit for the header could be shared with the SRAM control bit in the multiplexer, as the inverter can be turned off if it is not selected by the multiplexer. However, sharing the sleep transistor across multiple inverters is more area efficient.

The key design decision in implementing this scheme is the sizing of the sleep transistor. Larger sleep transistors have a smaller impact on the switch delay, as they are able to provide more current to active inverters. However, large sleep devices require more area, which may increase energy/operation by increasing the length of wires.

Figure 5.3 plots the delay overhead of the sleep device as a function of size (relative to minimum sized switch inverters) for a 16-input switch, obtained using HSPICE simulations. Like the pass gates in our switch multiplexer, the output of the SRAM is overvoltaged to provide additional drive strength to the sleep transistor when turned on. We see that a sleep transistor sized to 20× yields only a 10% increase in delay. The area impact of this device can be calculated as follows. A 16-input switch uses 34 transistors for inverters, 16 for the mux, and 96 for the 16 SRAM configuration bits,

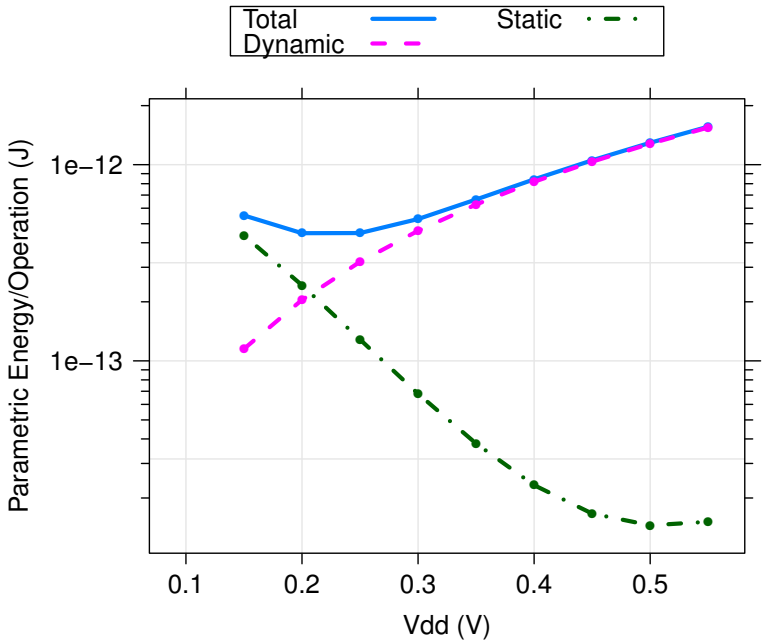


Figure 5.4: Energy/operation vs V_{dd} with power gating (`des`, 22 nm LP, minimum sizes, no variation)

for a total of 146 transistors. A single 20× sleep transistor (and the extra 6 transistors for the independent SRAM configuration bit) is therefore only a 16% increase in switch area, which will have a negligible impact on segment delays.

Figure 5.4 shows the energy/operation as a function of V_{dd} for `des`, again for the no variation case, but with power gating. We see that leakage energy can be reduced by approximately an order of magnitude, increasing the range over which V_{dd} can be scaled down and still reduce energy. Here, the minimum-energy V_{dd} is 200 mV instead of 350 mV, and the minimum energy/operation is 2.2× lower.

To explore the impact of power gating on delay-oblivious versus delay-aware routing, Figures 5.5 and 5.6 plot energy/operation as a function of V_{dd} for each router using minimum sized switches. Without power gating, we see that delay-aware routing reaches its energy minimum at 450 mV, while delay-oblivious routing minimizes at 600 mV. By scaling to lower V_{dd}, tolerating defects, and routing faster to reduce leakage energy/operation, delay-aware routing is able to reduce minimum energy/operation by 1.72× (45%). Because at 450 mV we encounter the energy minimum, any techniques that might scale below 450 mV would not achieve any energy savings.

However, with the addition of power gating in Figure 5.6, we can dramatically reduce leakage and continue scaling to lower V_{dd} to save energy. Here delay-aware routing can scale V_{dd} down to 400 mV instead of 450 mV. While oblivious still cannot scale down beyond 600 mV, the reduction in leakage has moved its energy minimum to 550 mV. When comparing the minimum energy of

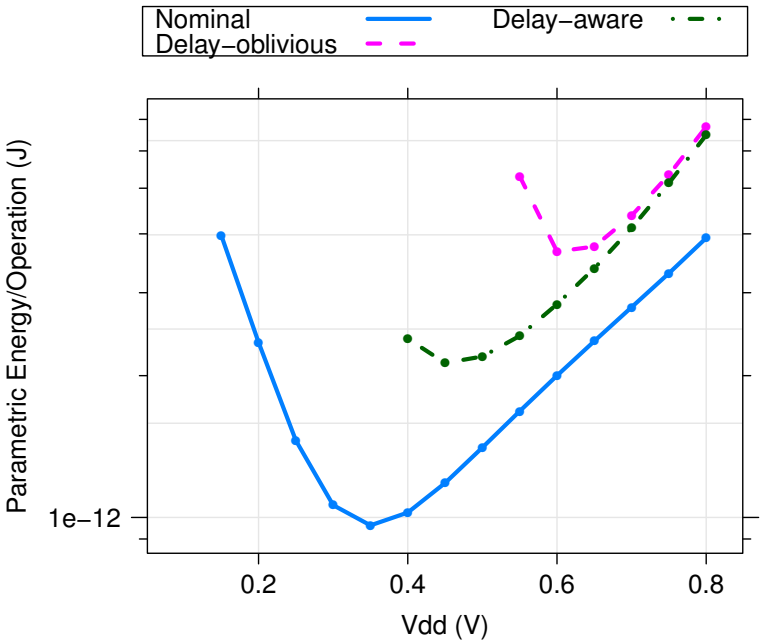


Figure 5.5: Energy/operation vs V_{dd} without power gating (`des`, 22 nm LP, minimum sizes)

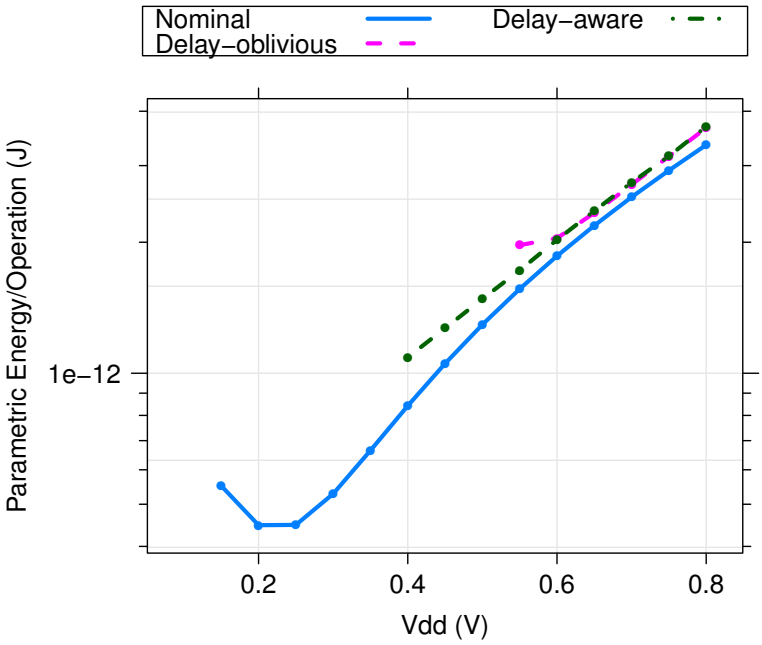


Figure 5.6: Energy/operation vs V_{dd} with power gating (`des`, 22 nm LP, minimum sizes)

delay-oblivious and delay-aware routing, we see that power gating yields an energy reduction of 1.81×, which equates to 5% additional energy savings, since both routers benefit. However, we will later see how to increase the defect tolerance of delay-aware routing, allowing further voltage scaling beyond 400 mV.

One hypothesis of using delay-aware routing to reduce energy/operation is the possibility to route faster at a lower V_{dd} where leakage energy dominates; by reducing the length of an operation, delay-aware routing can reduce the total energy and extend minimum voltage scaling to even lower V_{dd}. However, we see that power gating provides enough benefit that leakage energy/operation does not dominate total energy/operation until very low (< 250mV) voltages. Instead of reducing leakage energy/operation, we will see that the primary benefit of delay-aware routing will be defect avoidance at very low V_{dd}.

The remaining results in this chapter will explore the defect tolerance capabilities of delay-aware mapping, and all future energy/operation values will assume our power gating scheme. Because the size of the sleep transistor is significant, from Equation 2.13 we can safely ignore the impact of V_{th} variation on the sleep transistor.

5.2 Interconnect Sizing

One of the most important decisions in designing circuits is the assignment of gate sizes. Larger gates can switch faster due to increased drive strength (Equation 2.12), are more resilient to variation because they reduce the magnitude of variation (Equation 2.13), but dissipate more dynamic energy due to larger gate and diffusion capacitances (Equations 2.5).

From Figure 3.11 we see that failures due to variation at low voltage can be significant. SRAM and NMOS pass gate defect rates are low enough that they can be safely avoided. Because the SRAM bits connected to the gates of the NMOS pass gates in the switch multiplexer output an overdriven voltage, we can minimum size NMOS pass gates without encountering failures. However, LUT and inverter failure rates are non-negligible at subthreshold and near-threshold voltages. Therefore, sizing of inverters and LUTs can have a significant impact on their failure rates and functional yield.

We expect that larger gates will have higher yield and allow scaling down to very low voltages without failure. Smaller gates will not be functional down to the lowest voltages. However, larger gates are both more capacitive and require more area. Because we model the length of wires based on total area, increased area means increased wire length, which even further increases dynamic energy dissipation. Hence, very large gate sizes come at significant energy cost. The energy-optimal gate size will balance yield and energy dissipation. Additionally, since delay-aware routing provides defect tolerance, we expect it to be able to utilize smaller gates at lower voltages.

To explore the impact of sizing, we examine at two cases. First, the case where all interconnect

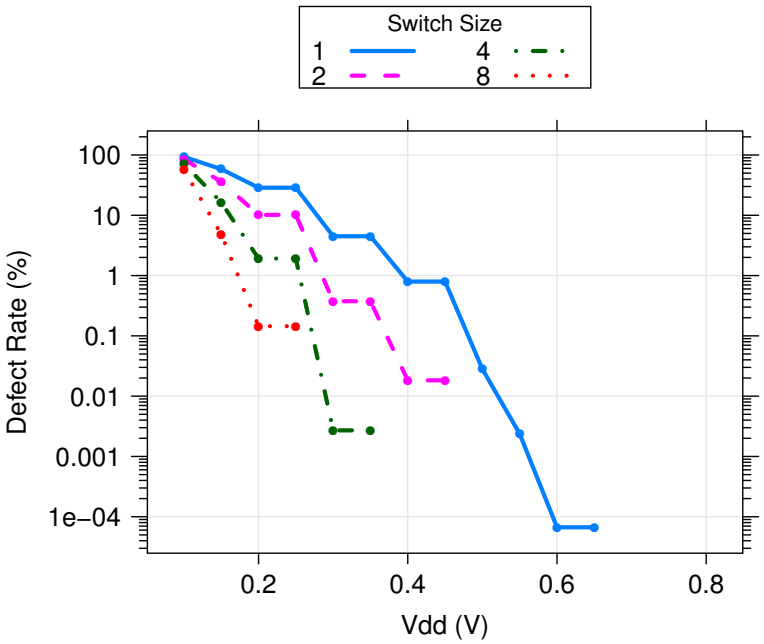


Figure 5.7: Defect rates vs V_{dd} for uniform sizing (`des`, 22 nm LP)

inverters are sized equally. Second, the case where inverters in the SBoxes and CBoxes (both input and output) are each sized differently.

5.2.1 Uniform Sizing

We first examine the uniform sizing of the switch circuit in Figure 5.2, which is the switch primitive for all interconnect switches: CBox inputs, CBox outputs, and SBoxes (Figure 2.1). We will see that the impact of sizing largely depends on the defect rates induced by variation as a function of gate sizes; Figure 5.7 plots measured average defect rates as a function of V_{dd} for a variety of switch sizes under variation for the `des` benchmark at 22 nm. We see that as we scale down V_{dd} below threshold, defect rates increase. Below 100 mV, 100% of gates are non-functional due to V_{dd} approaching the subthreshold slope. We observe that minimum sized inverters have the highest defect rate.

Figure 5.8 plots yield as a function of V_{dd} for various switch sizes for both the delay-oblivious and delay-aware routers. Here, we see that to obtain our yield target of 90%, larger gates or higher voltages are required. Delay-aware routing can obtain 90% yield with minimum sized transistors at 400 mV, while delay-oblivious routing requires 550 mV. By examining Figure 5.8 in conjunction with Figure 5.7 we can determine the maximum tolerable defect rate for both delay-oblivious and delay-aware routing. In general we see that delay-aware routing can tolerate defects at a rate of 1%, while delay-oblivious can only tolerate defects below 0.001%, a difference of three order of magnitude.

To examine the impact of sizing on delay, Figure 5.9a plots parametric delay as a function of

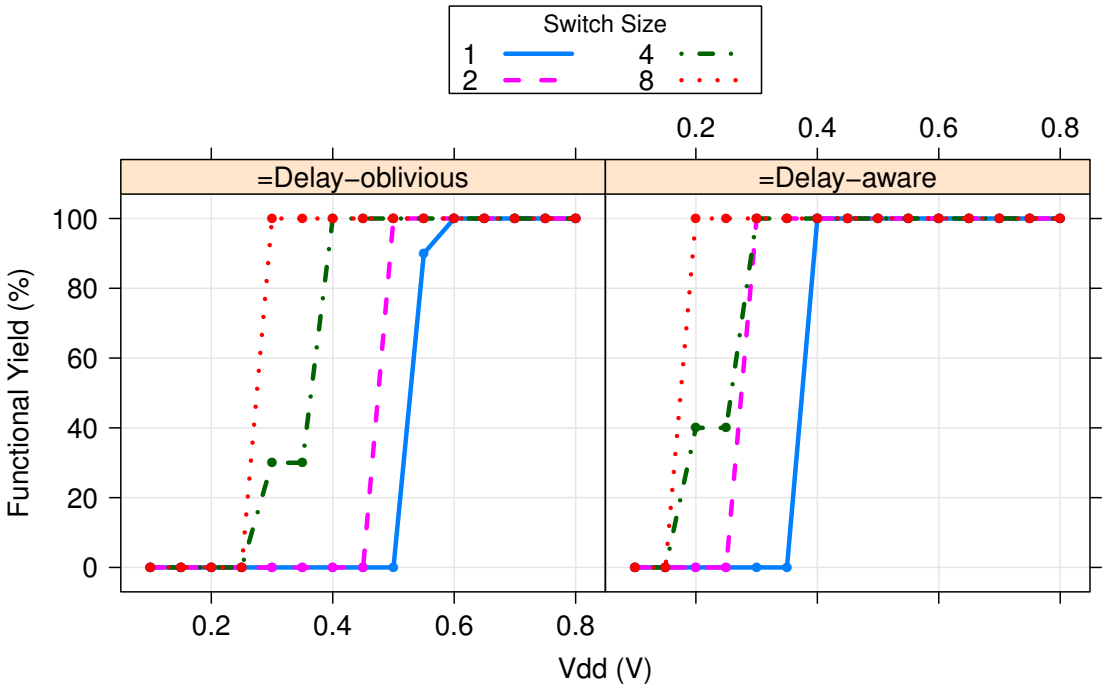


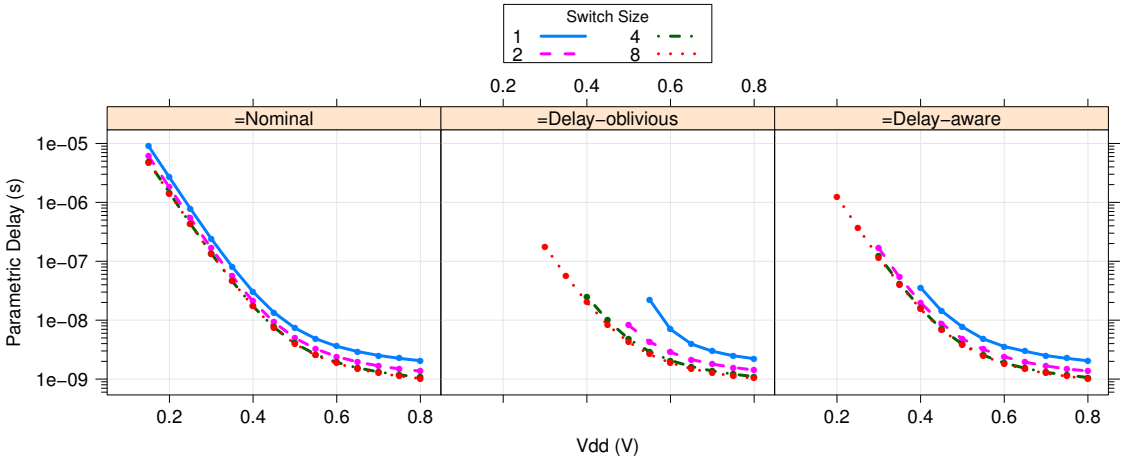
Figure 5.8: Functional yield vs V_{dd} for uniform sizing (`des`, 22 nm LP)

V_{dd} across a series of switch sizes, for nominal, delay-oblivious, and delay-aware routes for `des` at 22 nm. For the nominal, no variation case, we see that, at higher voltages, size 8 switches generally provide a good tradeoff between drive strength and capacitive load, which corroborates prior work in determining delay-optimal switch sizes [65]. The same basic delay-optimal sizing trends hold for delay-oblivious and delay-aware routing.

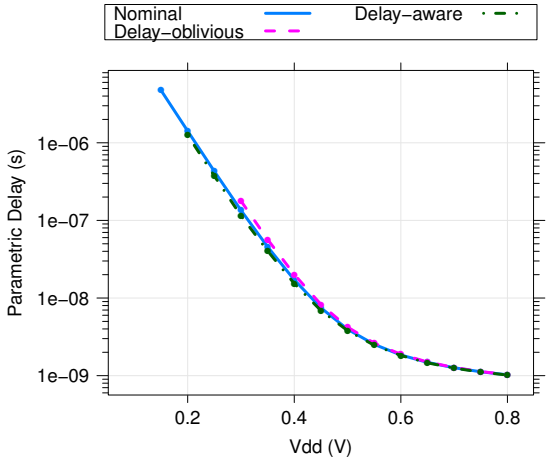
As we drop V_{dd} in the delay-oblivious case, we see functional failures as shown in Figure 5.8, represented by delay curves ending at voltages where the defect rate becomes too high to achieve 90% yield. As we increase switch size and hence decrease the magnitude of variation, we are able to scale down V_{dd} and remain operational, down to 300 mV for 8× sized gates.

The delay-aware router also encounters functional failures, but is able to avoid defects at lower voltages and smaller switch sizes. We see that the delay-aware router can retain 90% yield for 8× sized gates at a V_{dd} that is 100 mV lower than the delay-oblivious case (300 mV vs 200 mV).

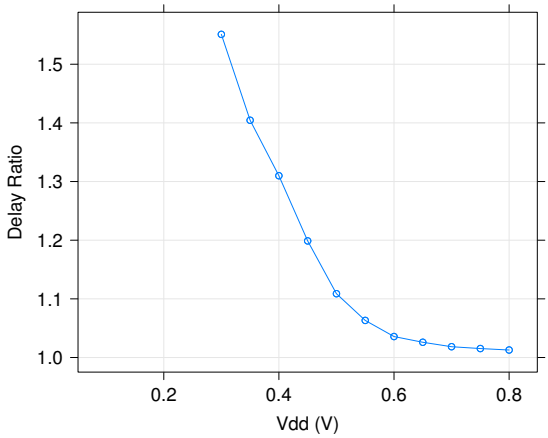
Figure 5.9b plots parametric delay for delay-optimal sizes across all V_{dd}'s (i.e., the composite minimum curve of Figure 5.9a). Again, we see that the delay margins induced by variation (the delay gap between delay-oblivious and nominal routing) at high V_{dd} is typically very small because of excessive path length averaging. However, as we drop the supply voltage these delay margins increase, up to around 1.2× the nominal delay at 300 mV. We see that delay-aware routing is able to completely eliminate variation induced delay margins at 300 mV. In fact, delay-aware routing is



(a) Uniform switch sizes



(b) Delay-optimal optimal uniform switch sizes



(c) Delay ratio of delay-oblivious/delay-aware routing for optimal uniform switch sizes

Figure 5.9: Delay vs V_{dd} for uniform sizing (`des`, 22 nm LP)

able to perform *faster* than the nominal case, by utilizing low V_{th} transistors that are sped up by variation on the critical path. Figure 5.9c plots the delay ratio of delay-oblivious to delay-aware routing; we see that delay-aware routing improves delay by 1.53×, in the best case.

To demonstrate the energy trends of uniform sizing, Figure 5.10a plots parametric energy as a function of V_{dd} and switch sizes. For the nominal case we see a characteristic minimum energy at 200 mV. We also see that minimum sized gates always provide energy-minimal operation, which is a well known result in subthreshold circuit design [27]. Without variation, larger gates significantly dynamic energy and wire capacitance and are not energy minimal.

For the delay-oblivious router, we see that in order to achieve lower V_{dd} and reduced energy, we must actually increase gate sizes in order to avoid defects. We see that 4× sized switches provide the minimal energy per operation at 400 mV. Here, larger, more reliable gates are actually more energy efficient.

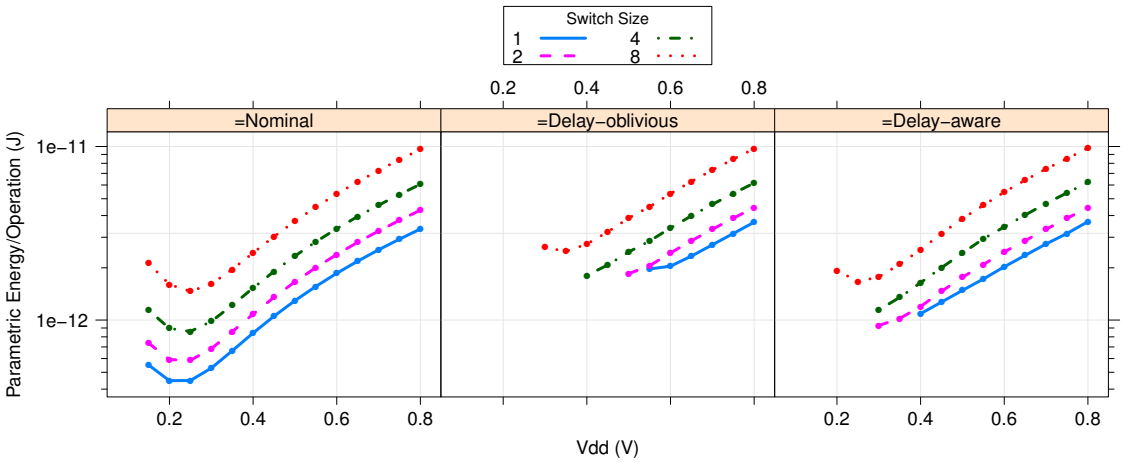
Because of its ability to avoid defects, delay-aware routing can use utilize the smaller, less reliable, lower energy gates at low V_{dd} without failing. However, we still see that minimum energy/operation is not achieved with minimum sized gates— here the energy minimum is achieved with 2× uniformly sized gates at 300 mV. Because delay-aware routing cannot tolerate more than 1% defects (Figure 5.7), we cannot scale below 300 mV with 2× or 4× gates, nor can we utilize 1× gates below 400 mV. Nevertheless, the energy benefit of using smaller gates at lower voltages is apparent: compared to the delay-oblivious case, delay-aware routing uses 2× smaller gates and can scale V_{dd} down by 100 mV (from 400 mV to 300 mV).

Figure 5.10b plots total energy/operation for energy-optimal sizes, again the composite minimum of the previous figure. Here we can clearly see the energy advantage from delay-aware routing, which is able to scale V_{dd} down by 100 mV more than delay-oblivious routing. The energy ratio of delay-oblivious and delay-aware routing is plotted in Figure 5.10c. At 300 mV we see a difference of more than 2.7× between delay-oblivious and delay-aware routing; however, this voltage is not the minimum energy/operation. If we examine the minimum energy/operation for both routers we observe that delay-aware routing yields a benefit of 1.91×, which is a 6% improvement over the uniform minimum size case.

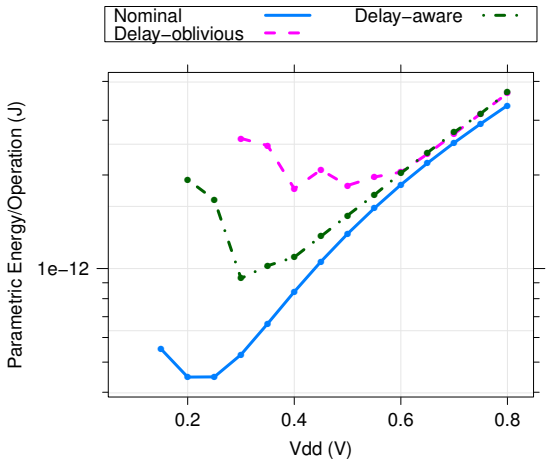
In summary, we see that energy-optimal, uniform sizing yields sizes of 4× for delay-oblivious and 2× for delay-aware routing. This smaller gate size combined with lower voltage operation provides delay-aware routing with 1.91× energy savings.

5.2.2 Selective Sizing

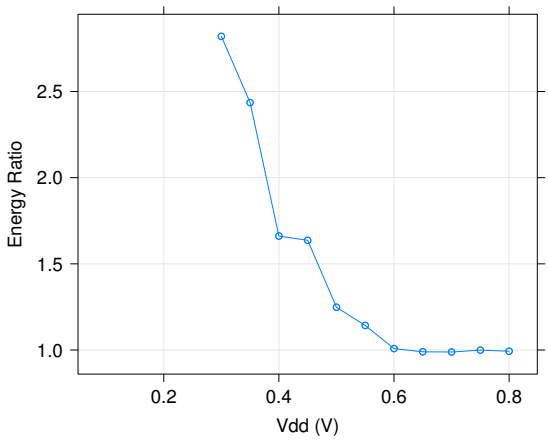
Uniform sizing across all switch types (SBox, CBox input, CBox output) may not be the most energy-optimal solution. The advantage of delay-aware routing is the ability to utilize smaller gates through defect avoidance; however, defect avoidance rates may differ between switch types. For



(a) Uniform switch sizes



(b) Energy-optimal uniform switch sizes



(c) Energy ratio of delay-oblivious/delay-aware routing for optimal uniform switch sizes

Figure 5.10: Energy/operation vs V_{dd} for uniform sizing (`des`, 22 nm LP)

example, we observed that the delay-aware router can generally tolerate up to 1% uniform defect rates across all switch types. However, if the router can actually tolerate higher SBox defect rates, we can size SBox switches down to save energy. This means that it may be more energy efficient to size up switch types where delay-aware routing has trouble avoiding defects, and to size down switch types that can easily be avoided.

Defect avoidance in FPGAs is afforded by the configurable, over-provisioned interconnect in FPGAs, which enables the delay-aware router to choose between multiple possible paths from source to sink when routing a net. The distribution and connectivity to those spare paths is not uniform in an FPGA. For example, there are many, many possible channel segments that can be selected for a route when routing a net from one CLB to another. There are several channels and possible directions a route can utilize; nets even have the option of taking non-optimal, non-shortest path routes. However, there are only a limited number of input pins that a net can use to enter a CLB. In our 4×4 architecture there are 10 possible input pins that can connect to 16 possible LUT inputs. Because of clustering, these inputs are often shared, so the utilization of input pins is most often under 100%. The number of output pins that a net can use to exit a CLB is also limited; this is customarily set to the number of LUTs in the CLB. If the CLB is fully populated, no output pins can be spared. The next section will more thoroughly explore impact of the number of channels and CLB pins on defect avoidance.

First, we can observe that it may be more energy-optimal to selectively size down resources that have more connectivity and spares, while sizing up the more important, limited connectivity resources. To compare the impact of selectively sizing SBox switches, CBox inputs, and CBox outputs switches on defect avoidance, we compare three cases, each where one resource is sized up $8\times$ while the other two resources are minimum sized. By measuring the yield in each of these selectively sized cases, we can determine which resource has the largest quantitative impact on defect avoidance.

Figure 5.11 plots average defect rates for CBox and SBox switches as a function of V_{dd} for each of our three cases. Each case is denoted by a tuple which refers to the size of SBox, CBox input, and CBox output respectively. For example, 1-1-8 refers to the case where only the CBox outputs are sized up $8\times$. Very similar to Figure 5.9a, we see that the minimum sized gates have significantly higher defect rates as V_{dd} decreases.

Because the delay-oblivious router has no knowledge of defects, we expect that changing defect rates of individual types will be of no benefit. Figure 5.12 plots yield as a function of V_{dd} for the delay-oblivious router for each sizing case. We see that each of the three cases is nearly identical for delay-oblivious router—90% yield cannot be achieved below 550 mV. These cases are the same as the uniform 1-1-1 sizing case explored in the previous section. Delay-oblivious routing cannot take advantage of differential defect rates without knowledge of defects.

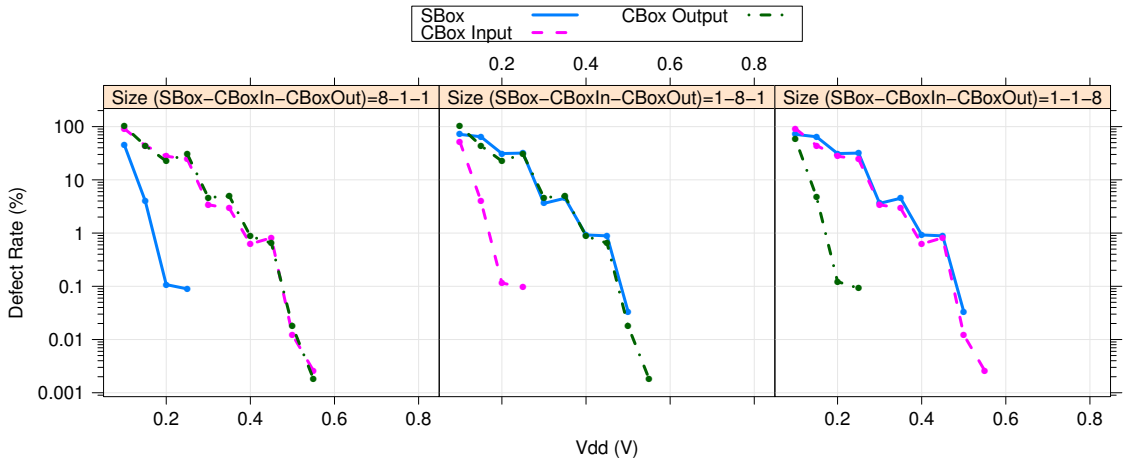


Figure 5.11: Defect rates vs V_{dd} for selective sizing (`des`, 22 nm LP)

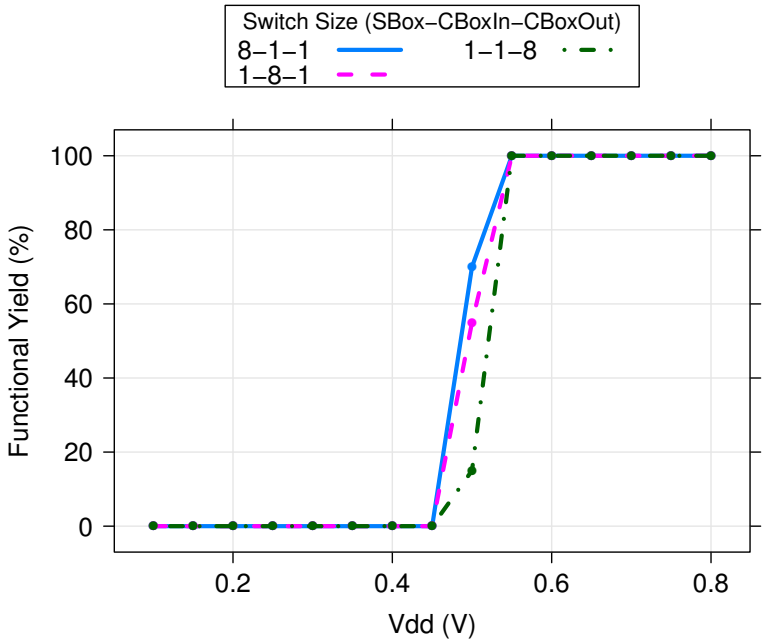


Figure 5.12: Delay-oblivious functional yield vs V_{dd} for selective sizing (`des`, 22 nm LP)

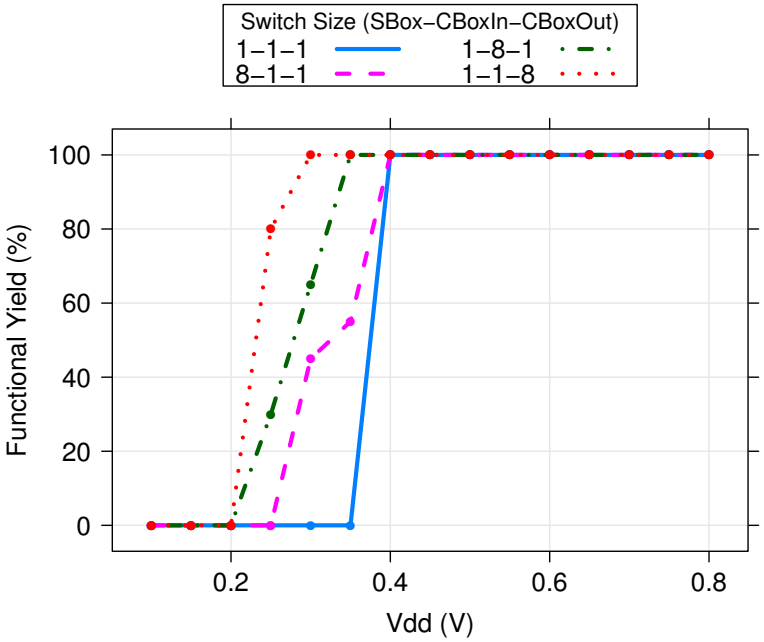


Figure 5.13: Delay-aware functional yield vs V_{dd} for selective sizing (`des`, 22 nm LP)

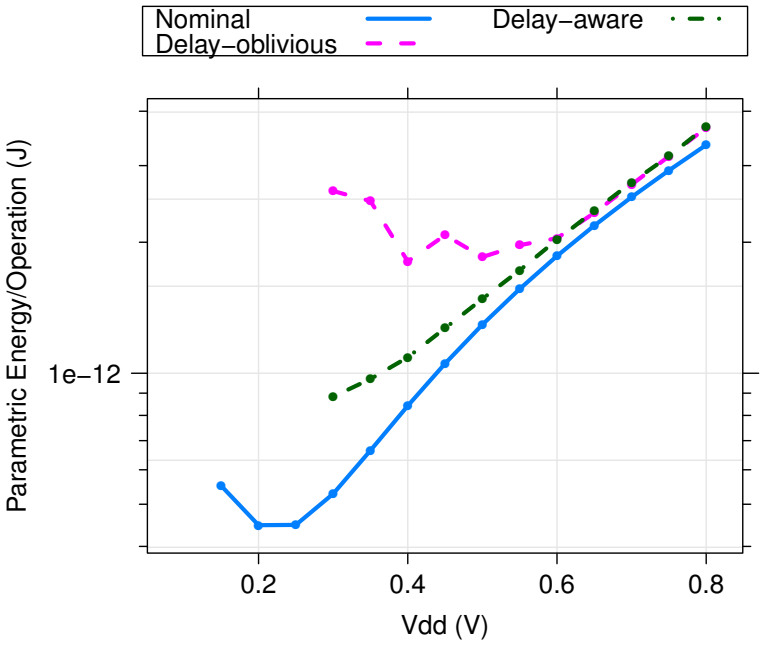


Figure 5.14: Energy/operation vs V_{dd} for energy-optimal selective sizing (1-2-2) (`des`, 22 nm LP)

Figure 5.13 plots yield as a function of V_{dd} for the delay-aware router for each sizing case. As previously discussed, we expect higher defect rates in channel segments (SBoxes) to be more easily avoided than CBox defects due to the larger number of spares and increased connectivity in SBox switches. Indeed, we see that the 8-1-1 has very similar yield as the uniform 1-1-1 sizing case in the previous section (90% yield at 400 mV). Sizing up SBox switches to 8× does not provide additional defect avoidance because of the already exiting flexibility in the FPGA interconnect. However, we see that the 1-8-1 case and the 1-1-8 case have slightly improved yield, achieving 90% yield at 350 mV and 300 mV respectively. Sizing up the more critical resources improves overall defect avoidance; we see that CBox outputs are the most important resource. Sizing to 1-1-8 is provides roughly the same yield as the previous uniform 2-2-2 case.

In general, we find that the energy-optimal selective sizing for the delay-aware router is 1-2-2 (assuming integer, power of 2 sizes). Figure 5.14 plots energy/operation as a function of V_{dd} for the 1-2-2 delay-aware router and the uniform sized delay-oblivious router. We see that the delay-aware router is able to scale down to 300 mV instead of 400 mV in the uniform 2-2-2 case from the previous section. What this means in terms of defect tolerance at 300 mV is that the delay-aware router can actually tolerate defect rates of 10% in the SBoxes and 1% in the CBoxes (Figure 5.7)

We see that the delay-aware router sized at 1-2-2 instead of 2-2-2 is able to reduce energy by an additional 8% when compared to the uniform sizing case. This slight increase is achieved by only sizing up CLB I/Os.

5.3 Interconnect Sparing

Selective sizing demonstrated that certain FPGA interconnect resources (CBox inputs and outputs) are more critical than others (SBoxes) when considering defect avoidance for delay-aware routing. While sizing up those resources to directly reduce the magnitude of defect rates is one approach to reliability, another approach is to make more spare resources available to the delay-aware router. In this section we will briefly explore the impact of adding extra channels and extra CLB I/O pins on the defect avoidance of delay-aware routing, and the possible energy benefits. The delay-oblivious router, as expected, achieves absolutely no benefit by adding spare resources, hence those results are omitted.

5.3.1 Extra Channels

We saw that sizing switches to 1-2-2 yielded the best energy/operation, which utilizes minimum sized SBox switches. Sizing up only SBox switches yielded no benefit for the delay-aware router; defect tolerance and energy benefits were only achieved by sizing up CBox inputs and outputs to 2×. This correlates to tolerable defects rates of 10% in SBoxes and 1% in CBox inputs and outputs,

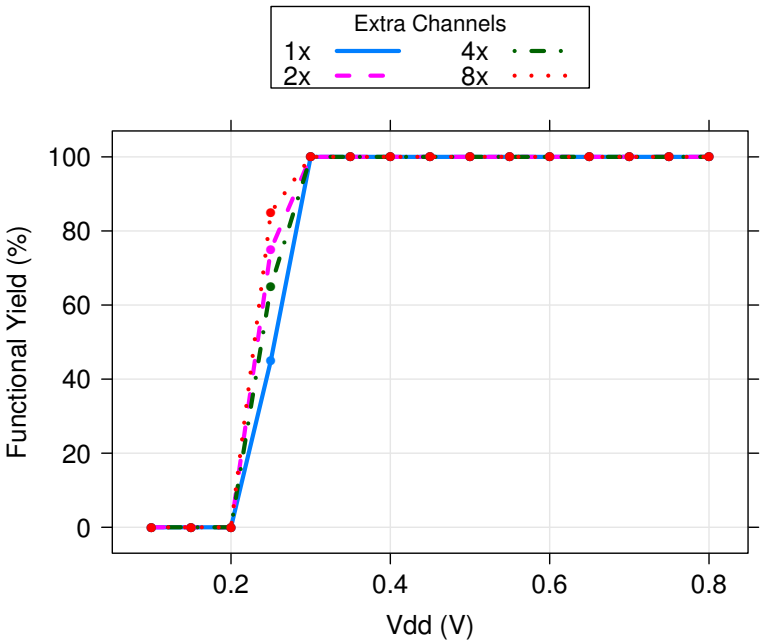


Figure 5.15: Delay-aware functional yield vs V_{dd} for extra channels (`des`, 22 nm LP)

which limited V_{dd} scaling to 300 mV.

Below 300 mV, minimum sized SBox switches will encounter defect rates >30%. To see if adding more SBox spares may aid in tolerating these increased rates, Figure 5.15 plots the yield of the delay-aware router for the 1-2-2 sizing using several values of extra channels. Unfortunately, we see that more than 8× the minimum number of channels still does not tolerate > 30% defects, preventing of scaling of minimum sized SBox switches below 300 mV. In general we observe that the benefit of adding extra channels is negligible.

5.3.2 Extra I/O Pins

We have seen thus far the CBox input and CBox output switches are the critical resources with regards to defect tolerance: selectively sizing up these switches to 2× to reduce defect rates improved yield far more than sizing up SBox switches. Hence, we saw that adding spare SBox resources (extra channels) did not appreciably improve defect avoidance. Instead, it may be more useful to add extra CLB I/O pins.

The ability to tolerate only defects rates of 10% in SBox switches and 1% in CBox switches limits voltage scaling to 300 mV. While adding spares to CLB I/O will increase the tolerated defect rate for CBoxes, scaling below 300 mV with minimum sized switches will yield > 30% defect rates which we have shown we cannot tolerate.

Therefore, it may actually be beneficial to re-examine the 2-2-2 sizing case, where the raw SBox

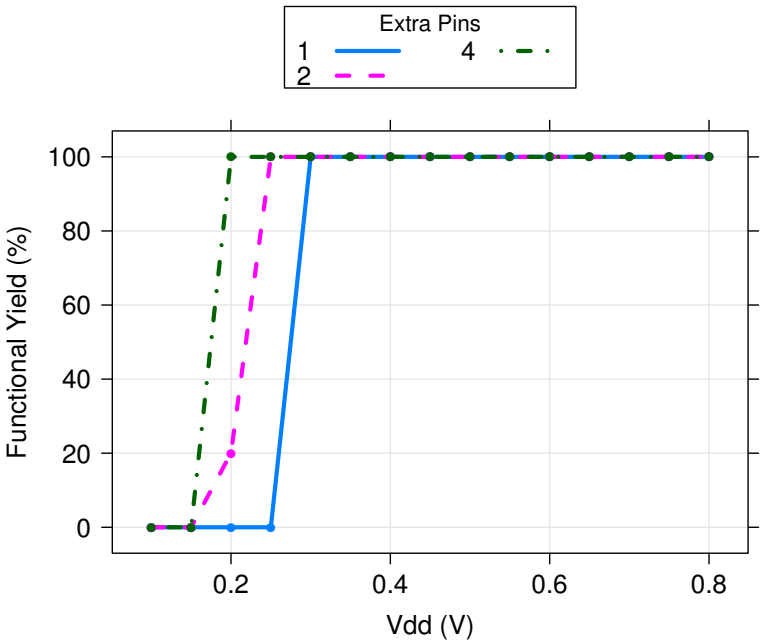


Figure 5.16: Delay-aware functional yield vs V_{dd} for extra pins (`des`, 22 nm LP)

defect rate will be reduced at lower V_{dd}, enabling voltage scaling below 300 mV. Scaling below 300 mV will increase the CBox defect rate to 1–10%. Without CBox sparing we saw that we can tolerate only up to 1% defects; here we will attempt to tolerate 1–10% through extra I/O pins.

Figure 6.4 shows the block diagram for a CLB including local interconnect and pins. We see that adding spare pins on the input or output comes at a cost of increasing the size of the internal CLB crossbar that connects the external CLB pins to the internal LUT pins. However, we observe that, for our architecture, the majority of interconnect energy (>75%) is the global interconnect fabric. This is due to the use of a smaller cluster size (4×4) and to the fact that more than 60% of all energy is dissipated in long wires, which are only used in global interconnect segments.

For simplicity, we model the impact of adding equal numbers of pins to both the CLB input and output. To see if we can tolerate the 1–10% CBox defects induced by a sizing of 2-2-2 and scaling V_{dd} below 300 mV, Figure 5.16 plots yield as function of V_{dd} for the delay-aware router for different values of extra pins. We see that adding extra pins enables scaling down to 200 mV where we encounter 1–10% defect rates across all switch types. Effectively, we observe that the natural FPGA interconnect can tolerate 10% SBox rates, while 4 extra pins are required for the same level of tolerance in the CBox pins. By adding these CLB I/O spares, we increases the overall defect avoidance of delay-aware routing, saving energy.

To see how much energy savings can be achieved by sizing at 2-2-2, using 4 extra pins, and scaling V_{dd} down to 200 mV, Figure 5.17 plots parametric energy as a function of V_{dd} for the 2-2-2

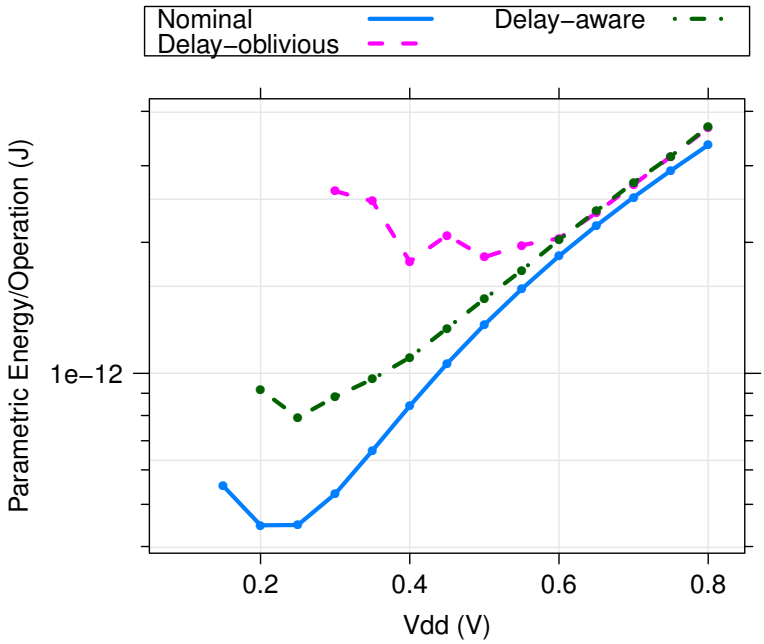


Figure 5.17: Energy/operation vs V_{dd} for energy-optimal sizing (2-2-2) and 4 extra pins (`des`, 22 nm LP)

delay-aware router with extra pins and the uniform sized delay-oblivious router. We see that the delay-aware router is able to reduce energy by an additional 15% by using extra pins, compared to the 1-2-2 selectively sized case. As it turns out, lowering V_{dd} to 200 mV is not energy-optimal as the minimum-energy point occurs at 250 mV. In total, we see that fully optimized delay-aware routing reduces the minimum energy with respect to delay-oblivious routing by 2.28×, an improvement of 12%.

5.4 LUT Remapping

Thus far we have only considered V_{th} variation in interconnect. We have shown that delay-aware routing is very effective at reducing delay margins and avoiding variation induced defects in interconnect switches at low V_{dd}. However, LUTs are also subject to variation, and the configuration of LUTs are fixed during routing. Because LUT delays are a very small fraction (less than 10%) of total delay, the delay impact from V_{th} variation in LUTs is minimal. LUTs however are still subject to the same variation induced defects as interconnect switches, and require some form of defect avoidance at low voltages to maintain high yield.

Our results so far have assumed minimum sizes for all transistors in the LUT (i.e., pass transistors, inverters). The simplest way to avoid defects at low V_{dd} in the LUT is to size up these devices. As around 80% of both static and dynamic energy are dissipated in interconnect switches and wires,

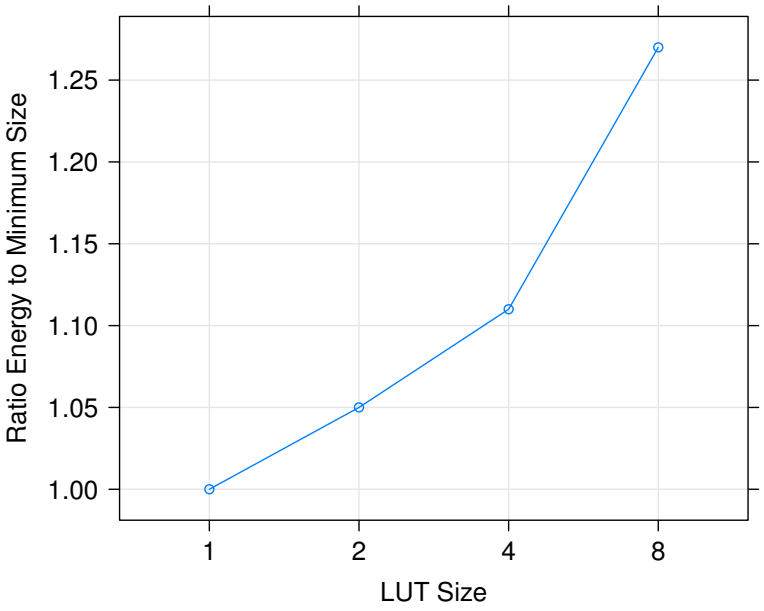


Figure 5.18: Energy ratio of sized LUT to minimum as a function of LUT size

sizing up LUT transistors will not substantially directly increase energy. However, sizing up LUT transistors can indirectly increase energy, by increasing area and lengthening wires. Figure 5.18 shows the impact of LUT size on the total dynamic energy for a mapped `des` circuit. We see that 8× sized LUTs increase total energy by 27%.

Unfortunately, we must size LUTs up to 8× in order to yield at low voltages. Figures 5.19 and 5.20 plot both defect rates and functional yield for LUTs at different sizes. We note that delay-aware routing has no way of mapping around defects in LUTs—any failure in the inverters or pass transistor multiplexers in the LUT will result in a failed chip. These results are identical for delay-oblivious routing. We see that as soon as defects begin to emerge, yield drops. Thus, without any techniques to avoid defects, we must either raise V_{dd} with small LUT sizes (i.e., >550 mV with 1×) or use large sizes for low V_{dd} (i.e., 8× for <250 mV).

Figure 5.21 plots energy as a function of V_{dd} for our optimized delay-aware routing results, but with LUT variation and 8× sized LUTs. We see that dynamic energy increases by ≈ 25% due to longer wires, increasing the gap between both delay-oblivious and delay-aware with respect to nominal. Interestingly, we note that while delay-oblivious total energy increases by 25% because of the increase in dynamic energy, delay-aware energy only increases by 5% because at 250 mV static energy dominates dynamic energy. Hence, the ratio of delay-oblivious to delay-aware minimum energy increases to 2.6×.

In order to reduce the size of the LUT, we would like to have an analog of component-specific rout-

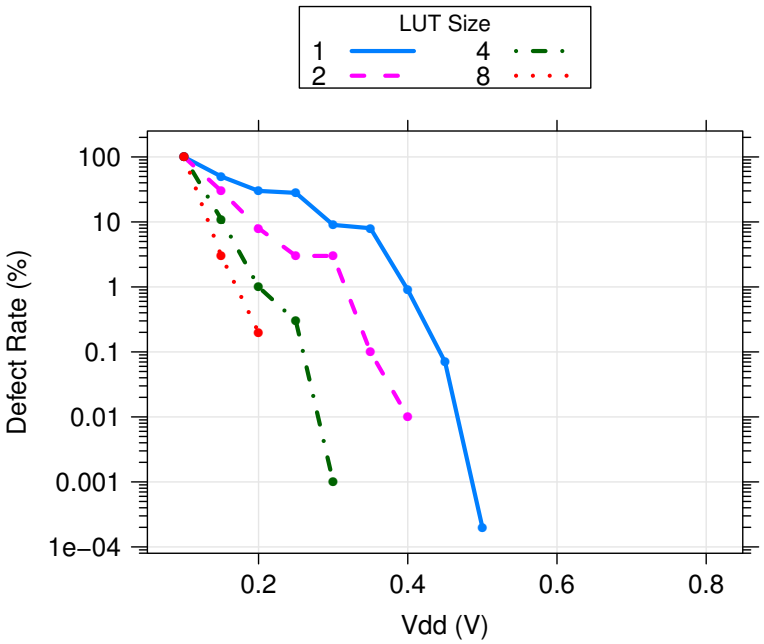


Figure 5.19: Defect rates vs V_{dd} for LUT sizes (`des`, 22 nm LP)

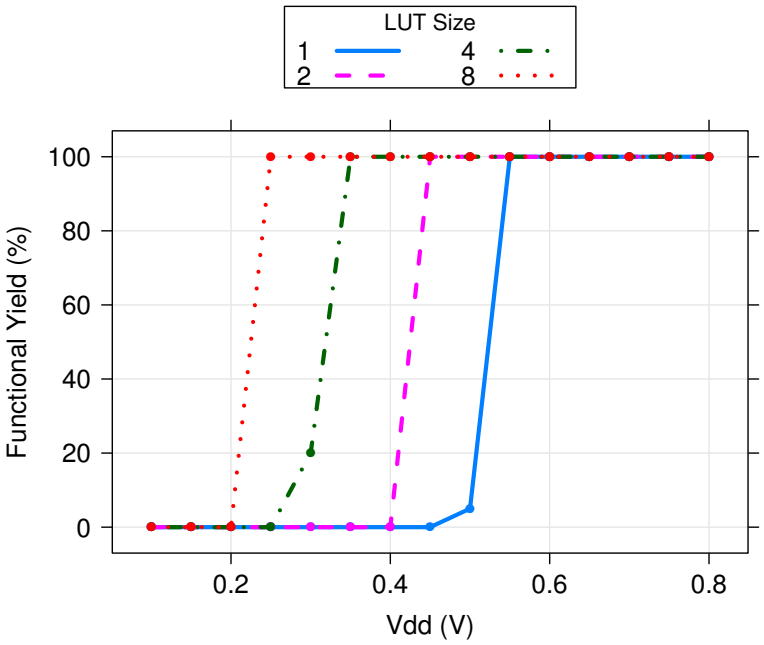


Figure 5.20: Functional yield vs V_{dd} for LUT sizes (`des`, 22 nm LP)

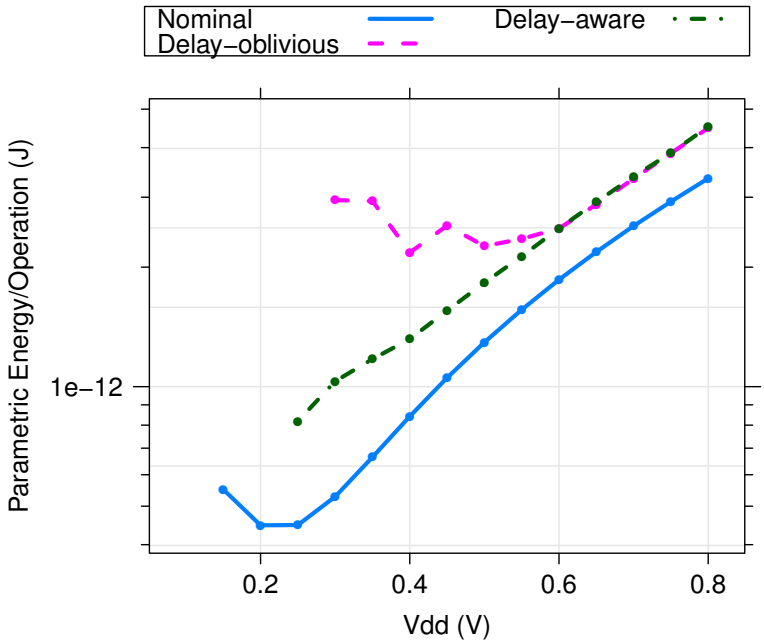


Figure 5.21: Energy/operation vs V_{dd} for LUT sizes (`des`, 22 nm LP)

ing for LUT mapping to tolerate failures at low V_{dd}. Recall in Section 3.1.7 we saw that for the 1-bit LUT primitive circuit, the failure probability is dependent on the precise configuration of the LUT. We observed that configurations of 00 and 11 do not fail at any appreciable rate. Configurations of 01 or 10, however, fail at rates very similar to those of inverters. This configuration dependent failure probability may provide us with some leverage to avoid LUT defects post-fabrication, at configuration time.

Figure 5.22 shows the circuit schematic of a simple 2-input LUT under variation with a configuration of 0100. As noted previously (Section 3.1.5), we re-buffer after every mux primitive to isolate each of the mux stages. In this example each of the transistors have nominal V_{th} except for the two transistors in the first mux primitive, which have both a low and high V_{th} transistor due to variation. With the given configuration (01 in the first two SRAM bits), we see that with enough variation the first mux primitive will be subject to failure. The I_{off} of the low V_{th} transistor will be large, while I_{on} of the high V_{th} transistor will be small; if these currents are comparable the mux will fail to switch the output to a high enough value.

However, if we have a defect map of the LUT prior to mapping, we can potentially avoid the defective mux primitive. Figure 5.22 shows the same circuit, but with the configuration bits re-ordered. Instead of a LUT configuration of 0100, we use a configuration of 0001. To perform this remapping, we simply invert Input1 (the input to the second stage of the LUT), which results in a functionally equivalent LUT but with a different LUT configuration. Now, the faulty mux primitive

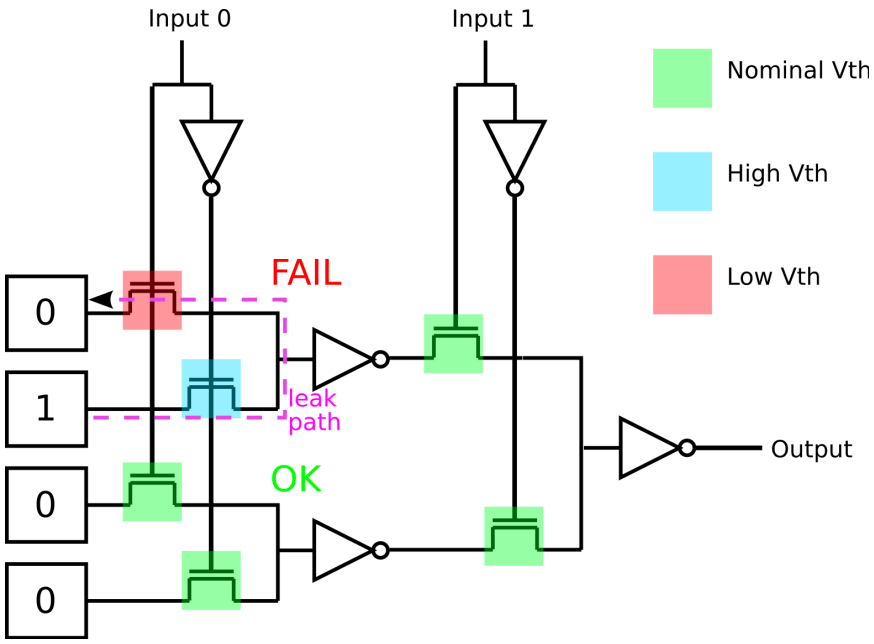


Figure 5.22: Defective LUT configuration under variation

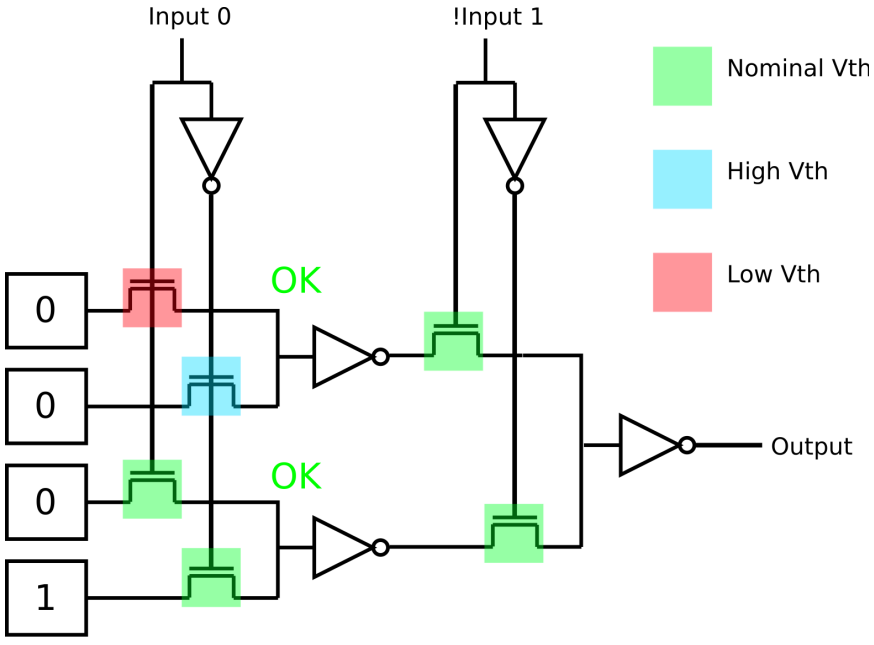


Figure 5.23: Valid, remapped LUT configuration under variation

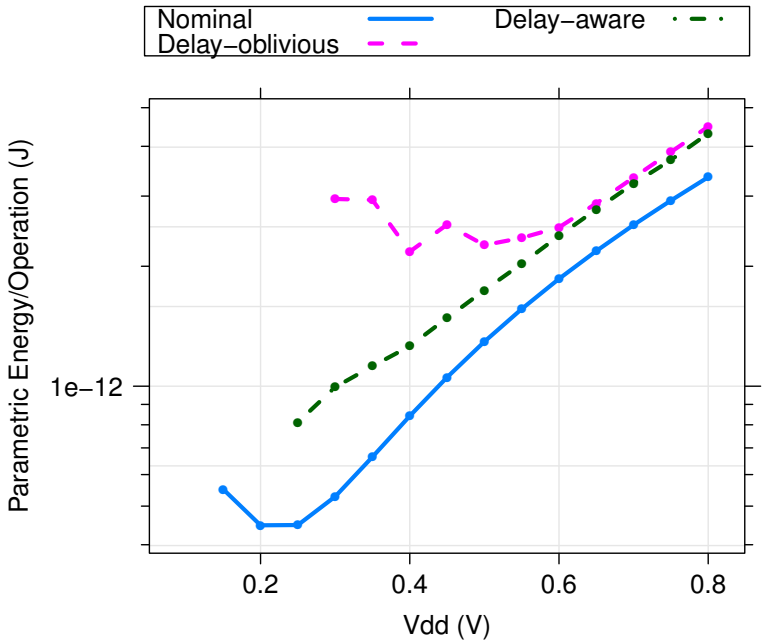


Figure 5.24: Energy/operation vs V_{dd} for LUT sizes (`des`, 22 nm LP)

is assigned a 00 configuration, which we know will not be subject to failure. The 01 configuration pair is assigned to mux with nominal V_{th}, and the LUT is now functional.

This remapping of configuration bits can enable us to size down transistors in the LUT as we can now avoid defects. To perform our defect-tolerant mapping, we first determine the nominal configuration of the logical LUTs at routing time via the benchmark netlist. Then we count the number of 00 and 11 pairs in each logical LUT to determine how many defects that LUT can avoid. We then match logical LUTs to physical LUTs, using a greedy matching strategy to assign the most defect-tolerant logical LUTs to the most defective physical LUTs [34]. Finally, we explore all possible input permutations for each physical LUT to determine the proper defect-free assignment. If the assignment succeeds without encountering 01/10 mapped to a defective mux, the chip passes.

Avoiding defective mux primitives attached to the configuration bits will allow us to size down the first stage of the LUT; however, later stages may not be able to be reduced in size. For example, to size down a potentially defective mux in the second stage of the LUT, the configuration bit subtree but be entirely 0000 or 1111 for the mux to yield. As we progress up the tree to the output of the LUT, it becomes much less probable to find a subtree of all 0's or 1's. We have found that we cannot maintain yield while sizing down gates past the first two stages of the LUT.

To examine the benefits of LUT remapping, we examine an architecture using our interconnect optimizations and where the first 2 stages of the LUT are sized down to 2× while the rest of the LUT is sized to 8×. Figure 5.24 plots energy/operation vs V_{dd} for our architecture relative to the oblivious

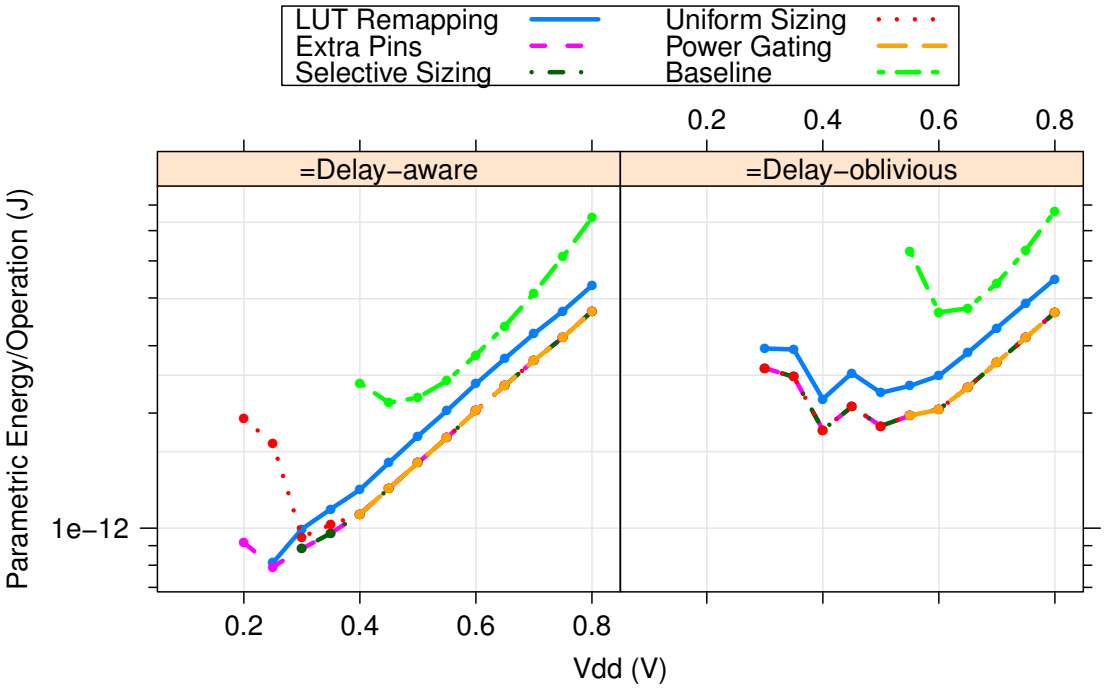


Figure 5.25: Energy/operation vs V_{dd} for each optimization technique (`des`, 22 nm LP)

case where LUTs are uniformly sized to 8×. We see approximately a 8% reduction in dynamic energy, and we see that delay-aware routing reduces energy by 2.66× compared to delay-oblivious. This is an improvement of 17% over the case that did not consider LUT variation.

5.5 Summary

In this chapter we have explored several optimizations to enhance the benefits of component-specific mapping. These optimizations have primarily been targeted at reducing leakage energy and increasing the defect tolerance of delay-aware routing. Additionally, we developed a technique to increase the resilience of LUTs by performing defect-aware mapping.

Figure 5.25 plots energy/operation for each of our optimizations for the delay-aware and delay-oblivious routers for `des` at 22 nm. We see that each successive optimization lowers the minimum energy/operation point. The LUT remapping optimization does show an net energy increase for both delay-oblivious and delay-aware cases, as the interconnect-only optimizations did not consider LUT variation and the required energy increase of sizing up LUTs. However, the LUT remapping optimization is able to reduce the energy gap created by this increased sizing. In summary, our thorough design space exploration has led us to the following architectural design point for the purpose of creating a low energy, variation-tolerant FPGA architecture:

- 4×4 **architecture with** $L_{seg} = 1$ **directional wires:** In agreement with prior work, we use a small cluster and lut size with minimum length segments for low-energy operation.

- **Delay-aware routed interconnect:** To tolerate defects induced by variation at low V_{dd}, and to reduce leakage energy/operation, we perform delay-aware routing assuming full delay knowledge of all interconnect resources. This saves a factor of 1.72× energy/operation.

- **Power gated switches:** We use fine-grained power gated at the individual switch level to reduce leakage at low V_{dd}, improving our results by 5%.

- **Size** 2× **SBox and CBox switches:** We size SBox and CBox gates uniformly at 2× for the optimal balance of energy efficiency and reliability, reducing energy by an additional 14%.

- **20% extra channels:** We route with only 20% extra channels, showing that increased channel counts do not improve component-specific routing.

- **4 Extra CLB I/O pins:** We add 4 extra input and output pins to the CLB to tolerate I/O failures, saving an additional 12%.

- **Hybrid size** 2× **and** 8× **LUT:** We size down the first two stages of the LUT while sizing up the latter stages. This differential sizing depends on LUT remapping.

- **Defect-aware mapped LUT:** We remap LUTs around defects to tolerate low V_{dd} failures in the first two stages of a LUT. This improves our energy savings by 17%.

In total, we show that architecture optimized component-specific mapping reduces energy by 2.66×, an improvement of 1.54× over the non-optimized architecture. In the following chapters we will evaluate this optimized architecture under a variety of different contexts, to determine both the sensitivity of the architecture to technological assumptions, and to determine how our results change under more practical delay measurement assumptions.

Chapter 6

Practicality

As described Section 1.4, the scope of this work is not to provide a complete solution to component-specific mapping. The goal of this work is to quantify the potential benefits obtainable through component-specific mapping and its possible optimizations. The two key pieces missing for a complete solution to component-specific mapping are determining how to actually perform component-specific measurement of individual resource delays, and how to map each chip individually without exploding CAD runtime by running a full mapping for every chip. This chapter outlines existing and ongoing work in solving both of these problems in order to provide perspective that they are not insurmountable.

6.1 Component-Specific Measurement: Timing Extraction

Without reconfiguration, measuring every device on a chip would be extraordinarily difficult: each device would need to somehow be individually addressable and able to be contacted through a highly complex tester. Testing would also have to consist of delay measurement, rather than the typical pass/fail testing for chips today. Further, testing each of the millions or even billions of devices on a chip would take an extraordinarily long amount of time at great cost.

Reconfiguration allows for the possibility of configurable, fast self-testing without complex external testers. However, reconfiguration does not in and of itself solve the problem of determining the delay of every resource. If the goal is to *directly* measure the delay of every single inverter, pass transistor, and wire in an FPGA by configuring a path through each element individually, this is impossible. Individual circuit primitives are not addressable in an FPGA; for example, when configuring the switch in Figure 2.3b, we see that we cannot decouple the output driver inverter and any input stub inverters from downstream switches. However, as the delay of a path is the linear sum of these element delays, it may be possible to make many different path measurements to construct a linear system of path delay equations. Then, if the system is determined, one could solve the system of equations and resolve each individual element delay. This idea forms the basis of the technique

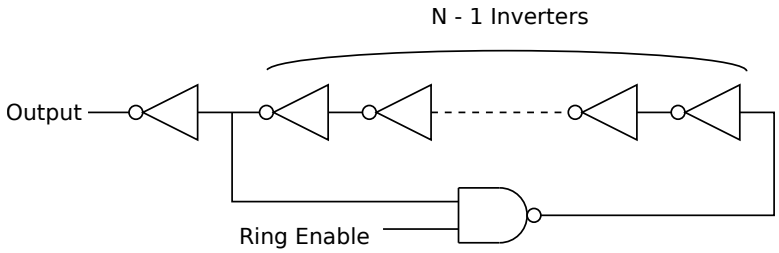


Figure 6.1: Ring oscillator

of FPGA timing extraction.

Prior work has demonstrated several techniques that quickly and precisely measure path delays through self-measurement in FPGAs. Sedcole et al. performed the first delay variability characterization of an entire commercial FPGA using an array of configured ring oscillators [104]. N-stage ring oscillators (Figure 6.1) are configured on the FPGA, where each stage delay is the delay through a single LUT, interconnect switches, and associated wiring. Connecting the output of the ring oscillator to a counter allows us to count the number of oscillations within a known time interval and compute the ring oscillator frequency. A NAND gate can be used to enable/disable the oscillator so the count is taken for a well-defined period of time. A ring oscillator only requires at minimum N=3 stages. However, at N=3, the oscillator frequency may be too high for reliable operation of the counter; in practice values of N=5 or N=7 are required. This limits the granularity of measurement to 5 stages of LUT + interconnect delays. Although individual stage delays cannot be obtained, individual stage delay variance can be estimated by dividing the oscillator delay variance by the number of stages. Three separate estimates can be made and correlated by measuring the delay of the N=5 ring, the N=7 ring, and the difference between the two measurements:

$$\sigma^2_{stage} \approx \frac{\sigma^2_{N=7}}{7} \approx \frac{\sigma^2_{N=5}}{5} \approx \frac{\sigma^2_{difference}}{3} \tag{6.1}$$

Figure 6.2 shows a possible, chip-level measurement scheme. Ring oscillators are configured in an array on each CLB within the device, and can be individually selected and controlled to avoid local self-heating effects.

Because this measurement technique lumps together LUT and interconnect delays, it is more useful for characterizing and isolating LUT delay variability than for characterizing the delay of specific resources. Ring oscillators are not fine-grained enough to measure the delay of an individual resource and are difficult to use for higher levels of interconnect. Additionally, as they do not represent real circuits mapped to FPGAs, actual oscillator delay measurements can only be used indirectly for component-specific mapping.

Instead of using ring oscillators, actual configured path delays which can consist of any number of LUT and interconnect stages can be measured using the circuit shown in Figure 6.3 [124]. A

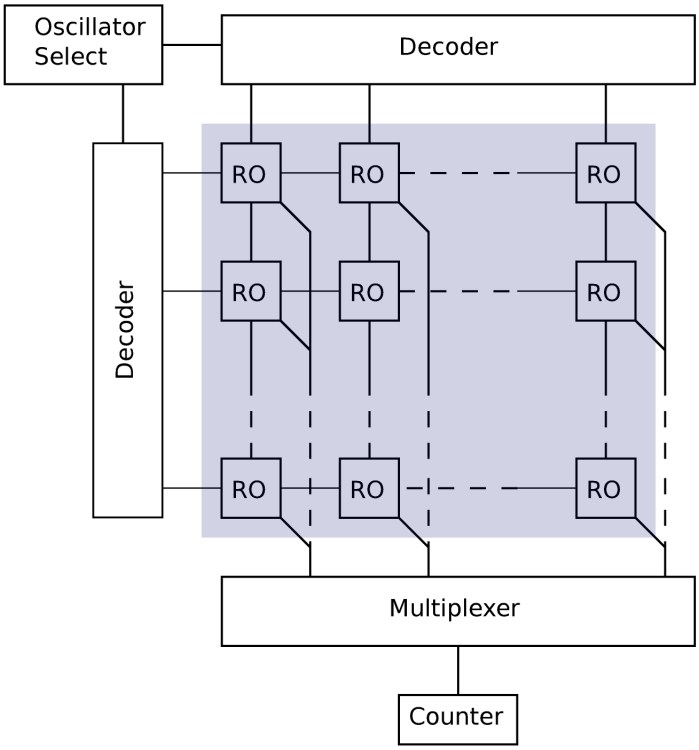


Figure 6.2: Measurement array of ring oscillators [104]

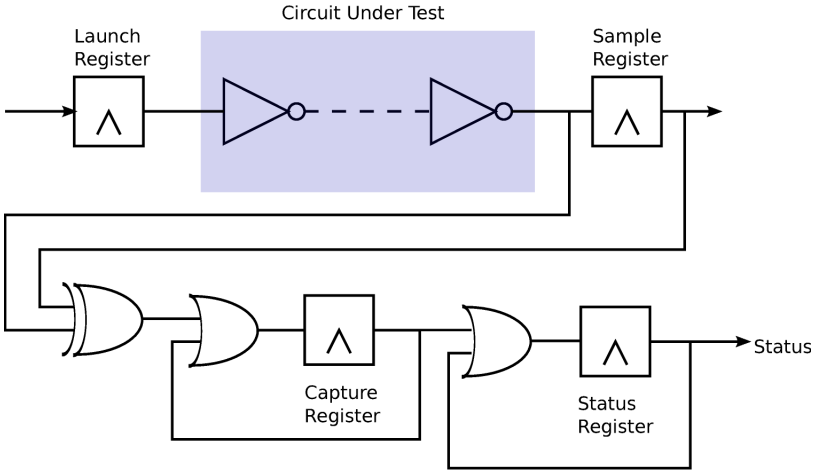


Figure 6.3: Path delay measurement circuit [124]

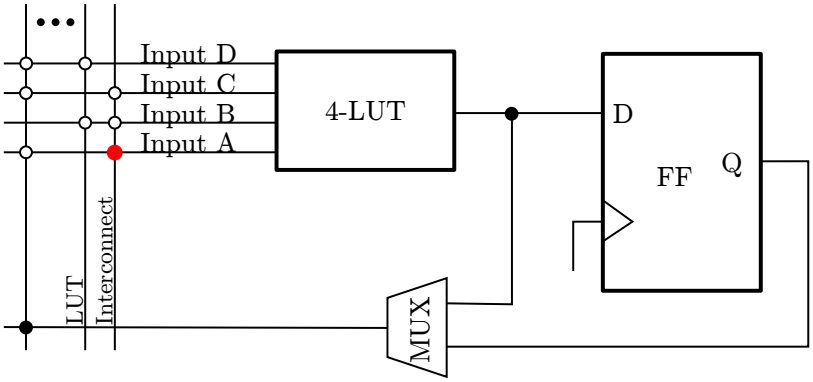


Figure 6.4: CLB with 4-LUT, register, and local interconnect

test stimulus is clocked into the launch register, through the combinational path, and into a sample register. The clock rate is generated by a test clock generator circuit. Conveniently, modern commercial FPGAs contain clock generators with picosecond timing precision. Increasing clock rates are applied for each test, and the input and output of the sample register are compared to determine if the correct value was latched for the tested clock frequency. If an error is detected a status register is set, indicating a timing failure.

With this strategy, precise combinational delays of real circuits can be obtained solely through configuration and self-measurement. Only three cycles are required per test, and one test can be performed immediately after another. Further, with appropriate consideration to self-heating and coupling, testing may be applied in parallel, where status register outputs can be combined into words and written in parallel into on-chip FPGA memory. Wong et al. [124] report full self-characterization of all LUTs (using internal CLB paths, as in the ring oscillator case) on a commercial FPGA in 3 seconds with 2 ps precision. Arbitrary path delay measurements using custom embedded structures such as carry-chains and embedded multipliers were also demonstrated for commercial parts.

Given this technique for precisely measuring full paths delays, it may be possible to resolved the delays of individual circuit elements. However, resolving the delays of elements as fine-grained as individual inverters and wires will still be impossible. Figure 6.4 shows the schematic of a CLB with its LUT, register and local interconnect. We see that any signal using the switch at the highlighted crosspoint must then traverse the LUT through input A. Both the switch and LUT are comprised of multiple circuit primitives (Figures 2.3b and 2.2b), which are impossible to decouple through different path measurements. However, decoupling down to the level of individual gates is not necessary if signals must always pass through groups of gates together. Instead, we can try to resolve the composite delay of these groups of circuit elements, or Logical Components (LCs) [44].

Figure 6.5 shows how to construct a graph of LCs given the schematic in Figure 6.4. With a set of measured paths, construction of a set of linear equations, and an appropriate change of basis, it can be shown that resolving the delays of individual LCs is possible [44].

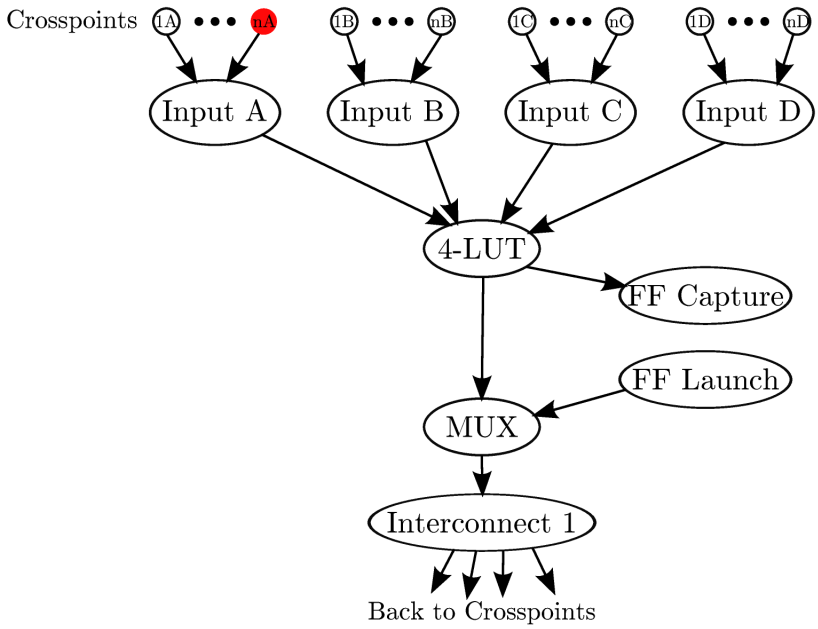


Figure 6.5: Graph of logical components (LCs) for CLB

6.2 Component-Specific Mapping: CYA Routing

Once measurements for every LC on a chip are obtained using timing extraction, to perform delay-aware mapping, each chip must be routed individually with its own delay and defect map. While the technology mapping, clustering and placement steps would only need to be performed once per design (Figure 3.12), the routing step would need to be performed once per chip (Figure 4.1). Routing is the most computationally intensive step in the FPGA CAD flow: routing every chip individually could potentially explode both CAD effort and handling time for customers of large volumes of FPGAs, as each chip would require a machine with a workstation class processor and memory to run routing. Further, every chip would require a separate bitstream, which complicates the process of assigning and tracking bitstreams. To avoid the severe computational time, energy, and financial costs, running full routing for every chip is something to be avoided if possible.

One proposed technique that can enable component-specific routing, but maintain the existing mapping model of routing once per design to produce a single bitstream, is Choose-Your-own-Adventure routing (CYA) [98]. CYA produces a bitstream containing a base routing configuration, but also includes both tests and precomputed alternative paths for each net mapped to the FPGA. The bitstream loader uses the embedded test to choose the best among the set of alternatives during the load operation. For example, the test operation can be a simple reachability test for defects (e.g., send a $0 \rightarrow 1$ and $1 \rightarrow 0$ transition down the configured path and see if the register at the receiving end receives them correctly), and the first path (out of the base path or the set of alternatives) to pass the test is chosen in the final configuration. The technique is named after Choose-Your-own-

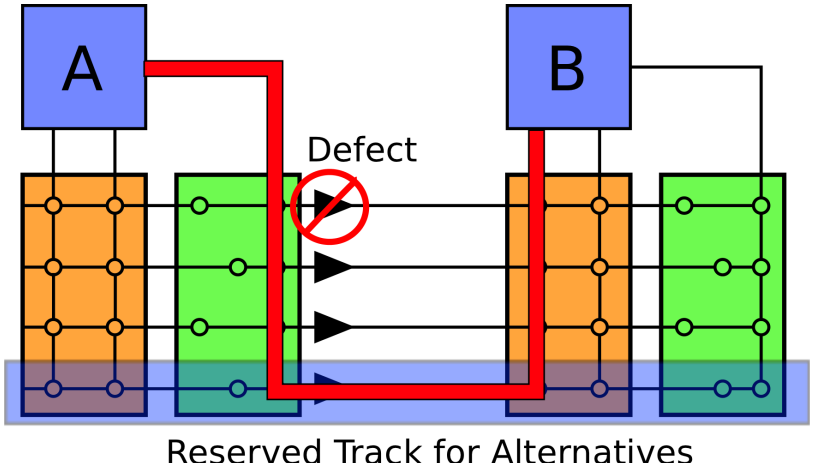


Figure 6.6: CYA example

Adventure novels [90], a popular series of children's novels from the 1980s and 1990s where multiple paths in a branching story lead to different endings, and readers often test story paths until they find the best possible ending.

Figure 6.6 shows an example using CYA for defect tolerance for the routing of a single net ($A \rightarrow B$). The architecture contains four tracks, where the fourth track is exclusively reserved for alternative paths generated by the CYA router. Routing is initially performed with the fourth track labeled "off limits", and a route using the first track is found. Next, the router generates an alternative path using the fourth, reserved track. Both of these routes are embedded in the bitstream. At load time the first path is programmed, and a simple reachability test is performed using the first route. The test fails due a defect on the first track and the route is deprogrammed. The alternative path is then programmed, tested, and finally configured as valid.

CYA relies on three key pieces: bitstream composition, bitstream generation, and bitstream loading.

Bitstream composition: The bitstream is composed of a list of two point nets, each with (1) a set of testing instructions, (2) a base path, and (3) a list of alternative paths. The CYA bitstream format does not require changes to the core of the FPGA, including the FPGA routing architecture.

Bitstream generation: The base route is generated by a standard FPGA router, with the only modification being to mark certain resources as reserved for later use by alternatives. Alternatives must then be generated, and must not conflict with the base route. Mutual exclusion can be guaranteed by a simple strategy of reserving routing tracks as in the example above, or by generalizing that idea to using independent routing domains [125].

Bitstream loading: The bitstream loading process consists of four components: the programmer, deprogrammer, tester, and controller. The programmer and deprogrammer simply set and reset the configuration bits in the FPGA; these components require very small modifications to current FPGA loading mechanisms (e.g., re-organizing the structure of current FPGA programming frames, and allowing state rollbacks). The tester can leverage existing FPGA structures (LUTs, flip-flops) to help perform any number of tests. The simplest tests would be end-to-end reachability tests as previously described; more complex tests to determine the actual timing of the configured paths can also be performed using "launch-from-capture" transition fault testing (e.g., Figure 6.3, [105, 118]. Finally, the controller co-ordinates between the programmer, deprogrammer and tester to run a very simple program, test, deprogram loop.

The costs of CYA are the following: some amount of extra resources for alternative paths, increased bitstream size scaling linearly with the number of alternative per net, and increased loading time. Rubin et al. showed orders of magnitude yield improvement with only 20% extra channels above the minimum, which is a common, low stress routing case easily achieved with the over-provisioning of interconnect in modern FPGAs. Bitstream size was shown to be 2–50× larger depending on the number of alternatives, with load times taking 2–100× longer depending on the configuration architecture. For many FPGA customers bitstream size and configuration time are of minimal importance.

While Rubin et al. demonstrated results for defect tolerance, the basic technique of testing precomputed alternatives embedded in the bitstream can also be used to perform delay-aware routing for variation. As noted above, by including timing requirements and substituting a timing test for the correctness test, alternatives can be used to identify and replace slow paths on a fabricated component. Instead of having complete knowledge of all resource delays to route a net, CYA can instead obtain the aggregate delay of resources on a path. Then, CYA can select the fastest path among a set of possible paths. Additionally, the CYA loading stage could be performed across a range of voltages, detecting and avoiding defects and slow paths at low voltages, in order to determine the minimum-energy configuration. However, because CYA relies on local substitutions and not global routing information, we expect the results from using CYA for variation to be of lower quality than those obtained by full knowledge delay-aware routing.

6.3 Impact of Delay Precision

The solutions outlined for component-specific measurement (timing extraction) and component-specific mapping (CYA) raise questions about how the benefits of using full knowledge delay-aware routing scale to more practical solutions. The delay-aware router presented in this work assumed complete knowledge of all primitive circuit delays (i.e., inverters, muxes, LUTs) and full double

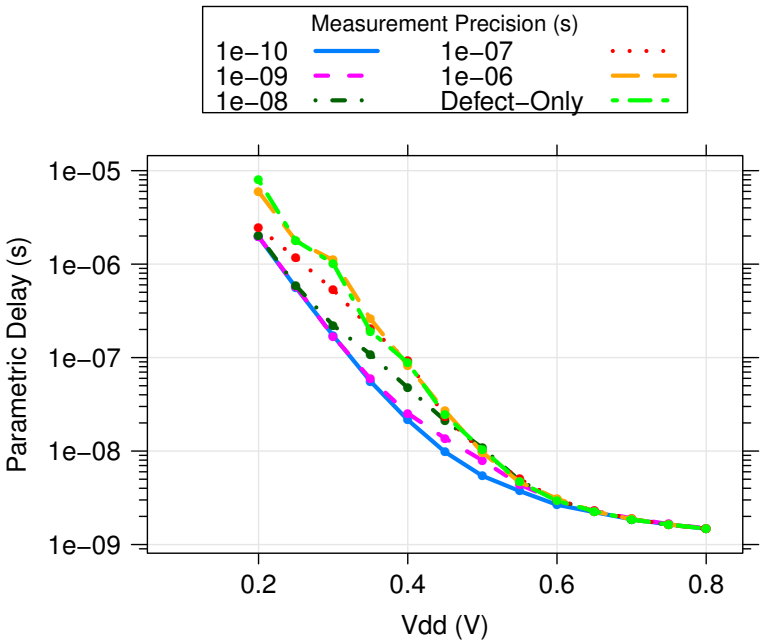


Figure 6.7: Delay vs V_{dd} of delay-aware router for different measurement precisions (`des`, 22 nm LP)

floating point precision for every delay. However, both timing extraction and the delay tests in CYA are limited in the precision at which delays can be measured. This section will examine the impact of delay precision on the energy benefits of delay-aware routing.

6.3.1 Limited Measurement Precision Mapping

We saw that the path delay measurement scheme in Figure 6.3 has a maximum precision of 2ps (for the particular 90 nm component used). It is likely that timing extraction will produce delays with precisions on the order of picoseconds; it is possible that this limit on measurement precision will have an impact on the delay and energy benefits of component-specific routing.

Figure 6.7 plots parametric delay as a function of V_{dd} for various measurement precisions, ranging from 10s of picoseconds to microseconds, for the `des` benchmark at 22 nm using optimized sizing and sparing. Measurement precision limits are applied to segment delays, which are assumed to be the atomic unit of measurement. We also include a special case where only defects are measured; all other delays are assumed to be nominal by the router. The maximum precision plotted is 100 ps, as we found that 100 ps was the maximum precision needed to track the full measurement precision case within 1% error.

We see that as measurement precision is reduced, the routed delay increases. One observation is that the delay increase depends on both the measurement precision and the voltage. For example, at 500 mV, 1 ns of measurement precision increases delay by 40%; however, at 300 mV, 1 ns of

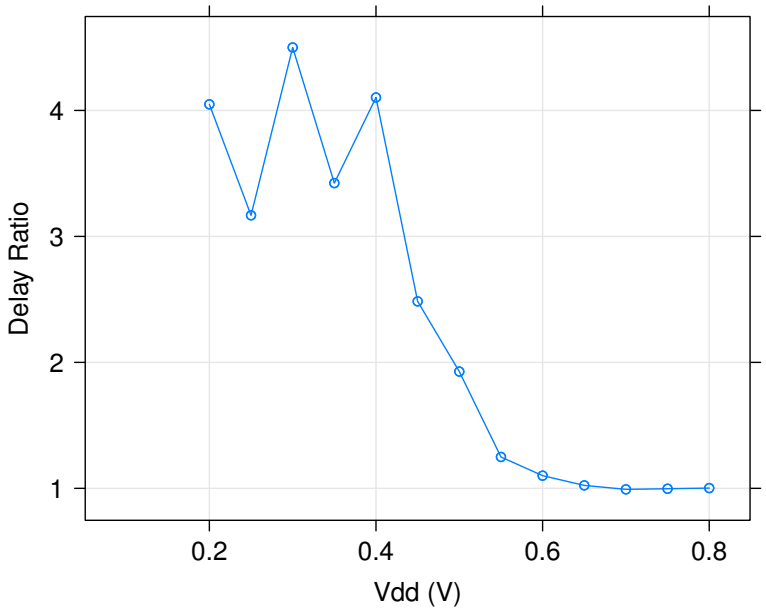


Figure 6.8: Delay ratio of defect-only to full precision to routing vs V_{dd} (`des`, 22 nm LP)

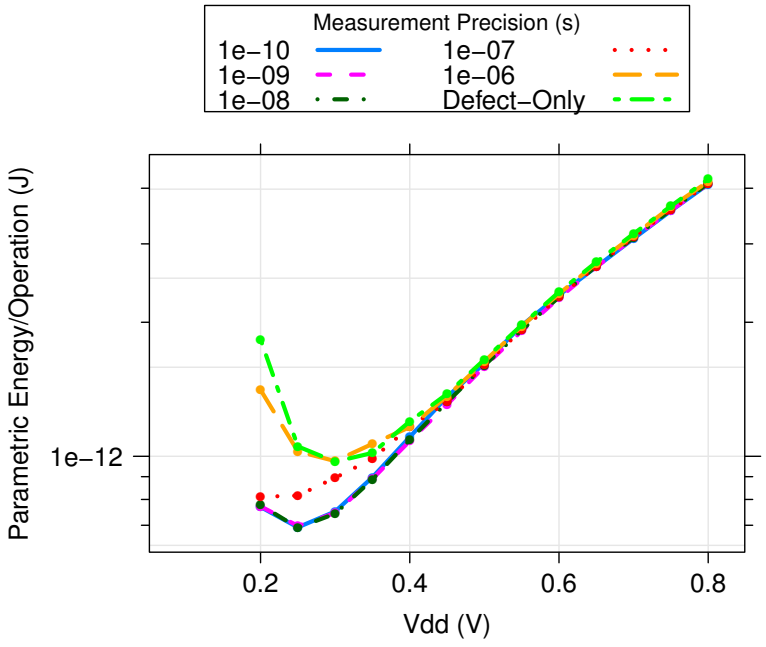


Figure 6.9: Energy/operation vs V_{dd} of delay-aware router for different measurement precisions (`des`, 22 nm LP)

measurement precision yields a delay identical to the full precision case. This is because segment delays at 500 mV are much faster (on the order of 10s of nanoseconds) and measurement errors have a more significant effect than at 300 mV where segments are much slower (100s of nanoseconds). At low voltages high precision measurements are of less importance because of slow delays. We see that the defect-only case is roughly equivalent to a measurement precision of $1\mu s$. Figure 6.8 plots the delay ratio of the defect-only route to full precision routing. In the worst case (300 mV) we see that defect-only routing is approximately 4.5× slower than 100 ps–10 ns precision equivalent routing.

Figure 6.9 plots energy vs V_{dd} for each measurement precision case. At large V_{dd} we see that energy is identical for all cases: at high V_{dd} delay and hence leakage energy have no impact on total energy/operation. As we reduce V_{dd}, we see that reduced measurement precision increases energy/-operation. Again we see that 100 ps–10 ns at low voltages is equivalent to the full precision case, with minimum energy/operation achieved at 250 mV. As limited measurement precision increases delay, we see the energy/operation curves begin to turn around due to the increase in leakage energy/operation. For the defect-only and $1\mu s$ cases, the 4.5× increase in delay and leakage energy/operation shifts the energy minimum up to 300 mV. The overall energy cost of performing defect-only routing is 30% relative to the 100 ps–10 ns measurement precision case. In Chapter 5 we demonstrated that full precision delay-aware routing can reduce energy with respect to delay-oblivious routing by 2.66×; with defect-only routing these savings fall to 2.04×.

While these results do quantify the impact of measurement precision, they do not explore the impact of actually using a technique such as CYA routing for variation. Because CYA uses local instead of global routing information and hence produces lower quality routes than a full CAD routing step, we expect CYA routed designs to have both higher delay and energy than full knowledge designs. However, the results presented here are a hard bound: CYA routing will perform no worse than our defect-only case. With the addition of even moderately accurate delay tests, it is possible that CYA will approach the full benefits of full knowledge delay-aware routing.

6.3.2 Limited Storage Precision Mapping

If component-specific mapping is to be practically usable on a large scale, it will be necessary to implement techniques such as CYA where a single routing step is used to produce a bitstream that can be customized per chip. However, it may be possible to run full routing on a component-specific basis on a small scale. Several FPGA vendors manufacture very low volume, high cost, highly specialized FPGAs for domain-specific applications [5]. Customers of such parts may be willing to dedicate CAD resource to individual chips if the benefit of component-specific mapping is great enough.

In these cases, measurement time and mapping time may not be the primary concern. However, storing the entire delay and defect map for a high density FPGA may be infeasible—for a component

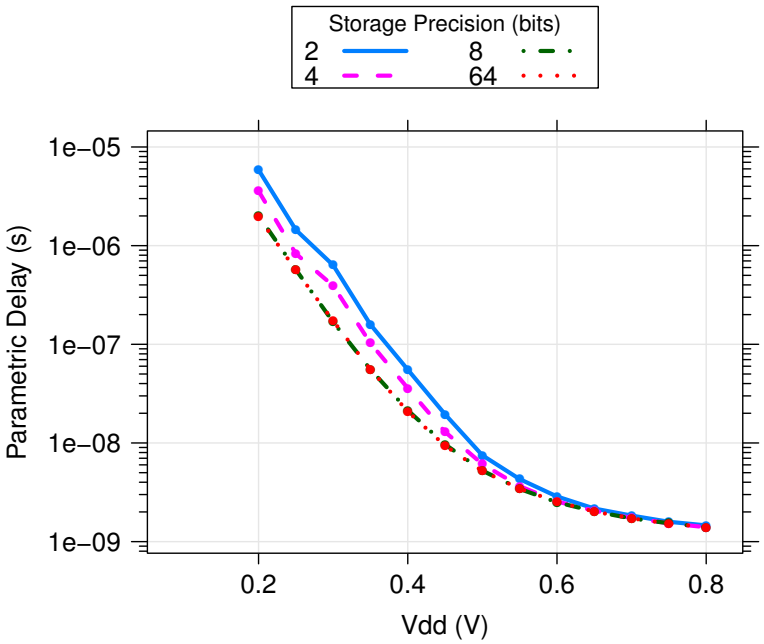


Figure 6.10: Delay vs V_{dd} of delay-aware router for different storage precisions (`des`, 22 nm LP)

with billions of resources, a complete delay map may require tens of gigabytes of RAM allocated to the processor performing delay-aware routing. However, it may not be necessary to store delays in the delay map at their full numerical precision, which could significantly cut memory requirements. This section will briefly examine the impact of reduced storage precision on the benefits of delay-aware routing.

Given a fixed number of n storage bits allowed per segment delay, we use a simple, linear binning scheme to break the range of delays into 2^n bins. We then assign segment delays into their appropriate bins, and set their actual delay as seen by the router as the max delay of the bin.

Figure 6.10 plots parametric delay as a function of V_{dd} for the 2-bit, 4-bit, 8-bit, and full 64-bit precision cases for the `des` benchmark at 22 nm using optimized sizing and sparing. We see that 8-bits of precision using our very simple linear binning method provides results identical to the full storage precision case. If less than 8-bits of precision are used, routed delay will increase. However, even with only 2-bits of precision (4 total bins), we can achieve within a factor of 2× the full precision delay in the worst case at 200 mV.

Figure 6.11 similarly plots energy vs V_{dd} results for each storage precision case. Again, we see that 8-bits of storage precision is equivalent to the full precision case. When comparing the minimum-energy points, we actually note that 4-bits of precision achieves within 2% of the full precision minimum energy at 250 mV. The worst case of 2-bits of precision has an energy overhead of an additional 26%.

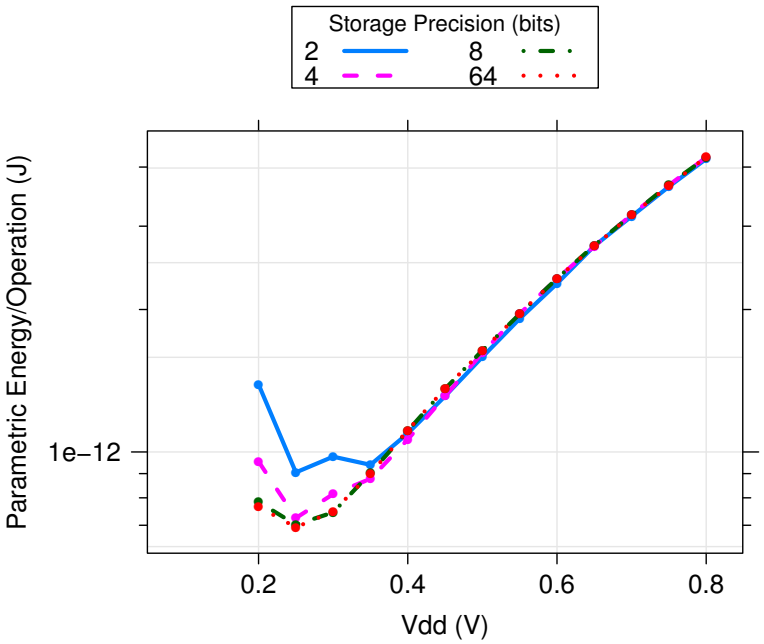


Figure 6.11: Energy/operation vs V_{dd} of delay-aware router for different storage precisions (`des`, 22 nm LP)

In conclusion, we find that measurement and storage precision can have a significant impact on both the delay and energy of component-specific routing. However, very high measurement and storage precisions are not needed to perform delay-aware routing. We find that 100 ps of measurement precision and 8-bits of storage precision produce results identical to the full precision cases. Additionally, we find that using a defect-aware router only incurs a 30% energy overhead with respect the fully delay-aware router.

Chapter 7

Sensitivity

Thus far we have demonstrated the benefits of component-specific mapping, explored several optimizations, and examined how these benefits might scale with a more practical implementation. In this chapter we will further explore how the benefits of component-specific mapping change under design and technological assumptions. Specifically, we will see how sensitive our results are to circuit pipeline depth, circuit size, the magnitude of V_{th} variation, and feature size. We will quantify the energy margins of delay-oblivious mapping, and the corresponding energy savings of component-specific mapping, across all benchmarks and feature sizes.

7.1 Pipeline Depth

In this work we have examined results from mapping the Toronto20 benchmark set [17], the standard benchmark set for academic FPGA research. Unfortunately, from Table 4.1 we can see that these benchmarks are very small by modern standards, using only 4,624 LUTs in the largest case. Modern FPGAs have hundreds of thousands of LUTs; for example, the largest Xilinx Virtex-6 contains 760,000 LUTs. Additionally, we note that half of the Toronto20 benchmarks use completely combinational logic. Modern FPGA designs are typically heavily pipelined to achieve high throughput; therefore, our benchmark set is not fully representative of large, high performance modern designs.

However, at the time of the development of this work, no modern benchmark set with large pipelined circuits that easily maps to VPR was publicly available. Additionally, VPR is known to have significant problems placing highly pipelined circuits; without significant modifications to the placement engine, pipelined circuits can have post-routing delays that are more than 40% above their actual delays [38]. Therefore, to simulate some of the effects of large, pipelined circuits, we create a simple, parameterized, hand-placed multiplier benchmark circuit where we can vary both the circuit size and the pipeline depth.

Figure 7.1 shows our multiplier design. The design is a simple array multiplier where every single signal exits a register, traverses a single segment, and then enters another register, guaranteeing a

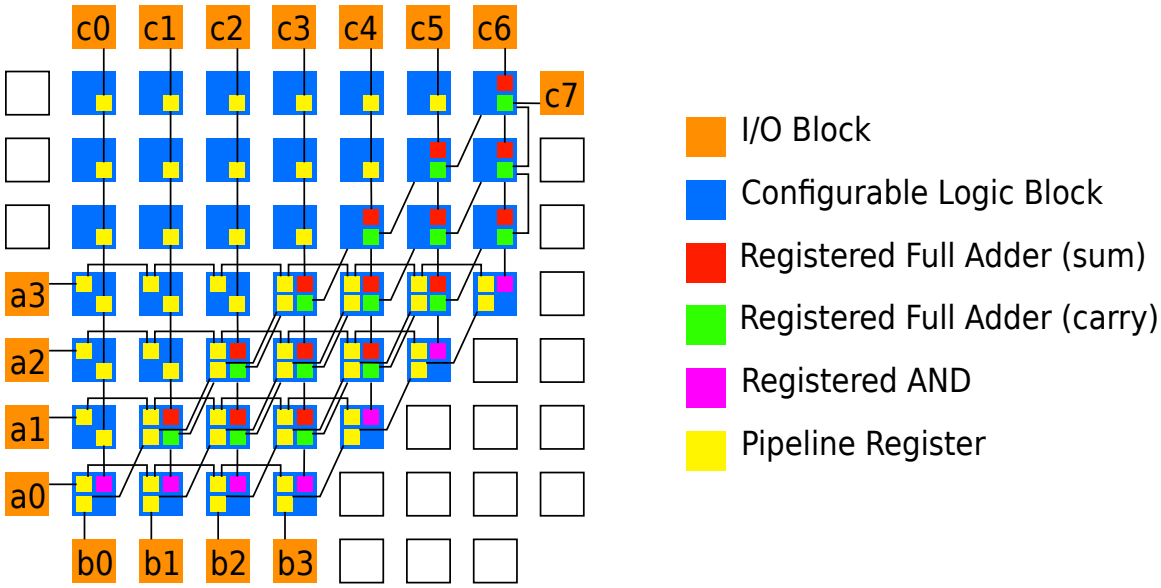


Figure 7.1: Fully pipelined 4× FPGA multiplier

pipeline stage delay of only one LUT + segment. The circuit can be scaled up in size by powers of 4, and register pipeline stages can removed arbitrarily to vary the pipeline depth. This section will examine the impact of pipeline depth while the next section will vary the size of the multiplier.

We expect that longer pipeline stages (more segments and LUT delays between registers) will experience more path length averaging, meaning that delay variation will be reduced. Hence, both the delay margins induced by variation (and the corresponding reduction those margins by delay-aware routing) will be relatively small. With short pipeline stages (fewer segments and LUT delays between registers) we expect to see much more delay variation. Additionally, since *every net* in the multiplier is critical, it will be extremely difficult for the delay-oblivious router to minimize the delay of every routed path simultaneously. The delay-aware router will also encounter difficultly minimizing delay—without a large amount of extra resources, it will be impossible for the delay-aware router to make all paths the same delay as in the nominal case. However, we expect the delay-aware router to significantly outperform the delay-oblivious router.

Figure 7.2 plots the delay of a 16-bit multiplier as a function of pipeline stage length at 22 nm. We plot delays with V_{dd} fixed at 600 mV for simplicity. We see that as we increase pipelining, the delay of the nominal multiplier decreases by an order of magnitude as expected. The delay-aware router is nearly able to track the nominal, no variation delays, eliminating delay margins induced by variation. However, we see as pipeline stage length scales down, the ability of delay-aware router to reduce delay margins decreases. The delay-oblivious router produces routes that are substantially slower than both other routers.

Figure 7.3 plots the delay margins (ratio to nominal delay) for both the delay-oblivious and delay-

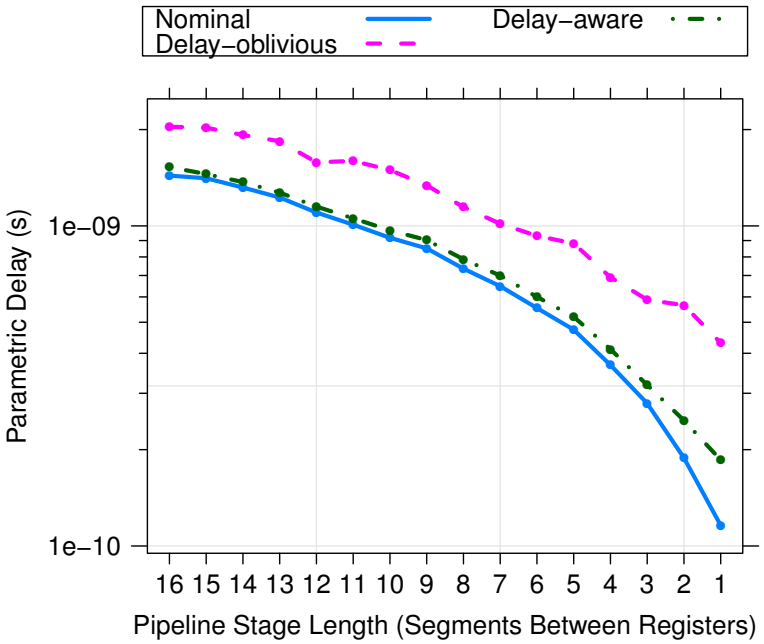


Figure 7.2: Delay vs pipeline stage length (`mult16`, 22 nm LP, $V_{dd} = 600\text{mV}$)

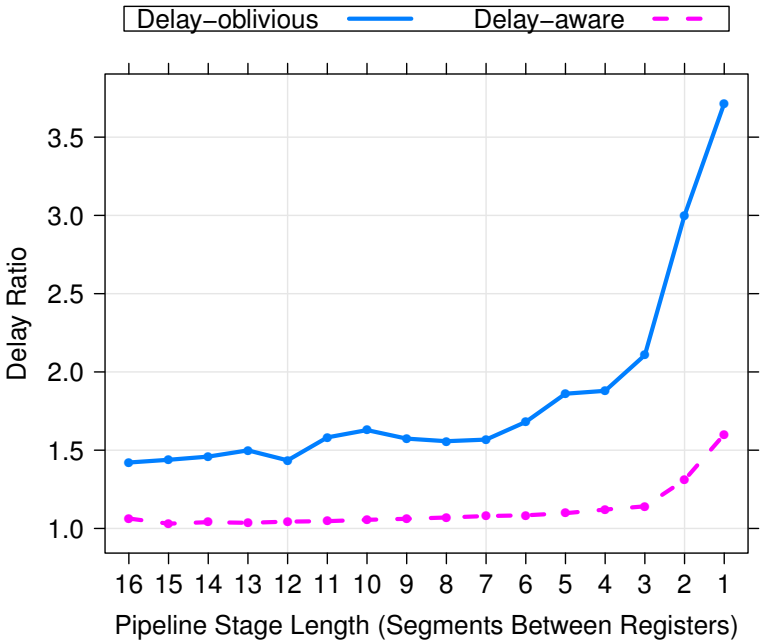


Figure 7.3: Delay ratio to nominal vs pipeline stage length (`mult16`, 22 nm LP, $V_{dd} = 600\text{mV}$)

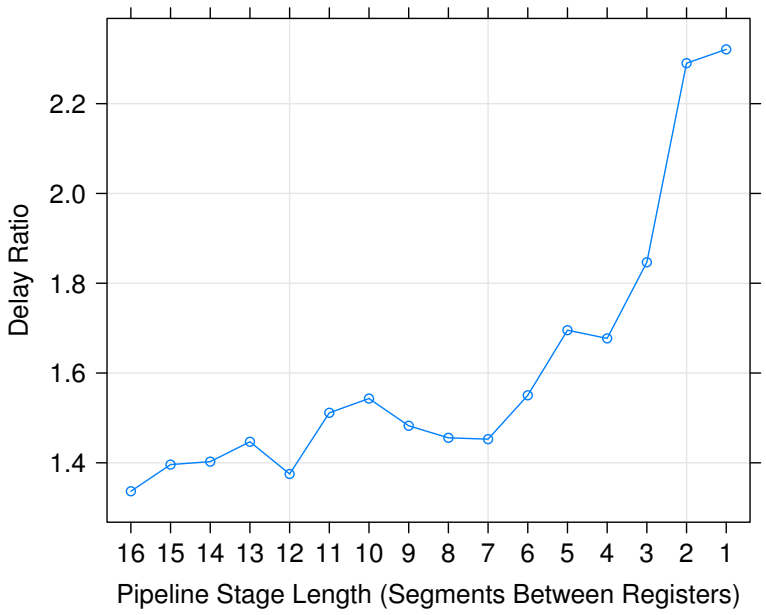


Figure 7.4: Delay ratio of delay-oblivious/delay-aware routing vs pipeline stage length (`mult16`, 22 nm LP, $V_{dd} = 600\text{mV}$)

aware routers. We see that as pipeline stage lengths decrease, the delay margins of delay-oblivious routing grows substantially, from $1.4\times - 3.7\times$ when moving from the fully combinational case to the fully pipelined case. As expected, delay-oblivious routing struggles to reduce the critical path delay when faced with all near-critical paths with large delay variance. The delay-aware router, however, is able to handle the pipelined case fairly well, incurring a 60% delay margin. Figure 7.4 plots the ratio of delay-oblivious to delay-aware routing. We see that for a fully combinational multiplier under variation, delay-aware routing is able to route $1.3\times$ faster than the delay-oblivious router; for the fully pipelined case the delay improvement is $2.3\times$. Therefore, pipelining increases the delay savings of delay-aware routing by $1.76\times$.

When considering minimum energy, for the nominal, no variation case we expect more heavily pipelined multipliers to dissipate less energy. As the length of an operation decreases, the amount of leakage energy/operation decreases, lowering the minimum-energy point. The delay-oblivious router, however, produces significantly slower routes relative to nominal with more pipelining. Therefore, we expect the energy of the delay-oblivious router to increase relative to the nominal router. We note that the delay and leakage increase will be even greater than we have seen in Figure 7.3, as the minimum energy for each pipelined multiplier case will be below 600mV.

Figure 7.5 plots minimum energy/operation as a function of pipeline stage length for the 16-bit multiplier. We see that as we increase pipelining, the energy/operation of the nominal multiplier decreases. The energy/operation of the delay-oblivious router, however, remains relatively con-

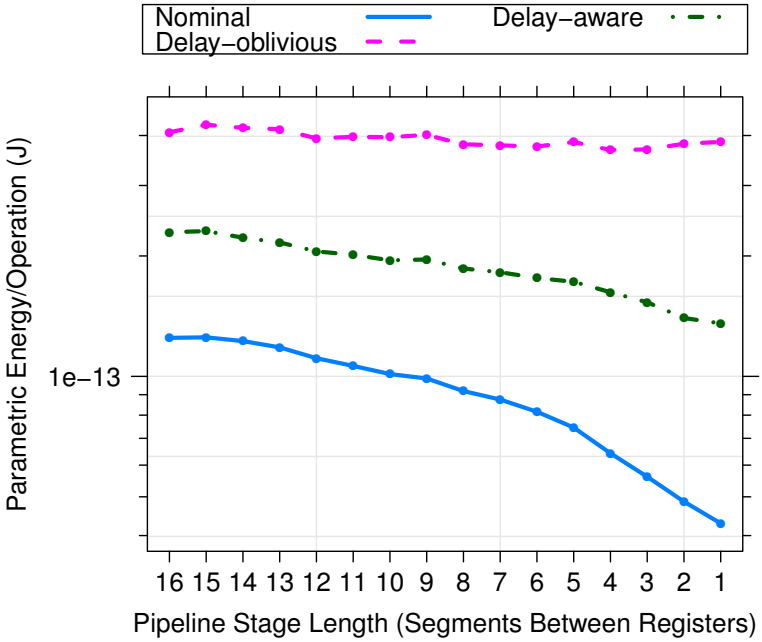


Figure 7.5: Minimum energy/operation vs pipeline stage length (`mult16`, 22 nm LP)

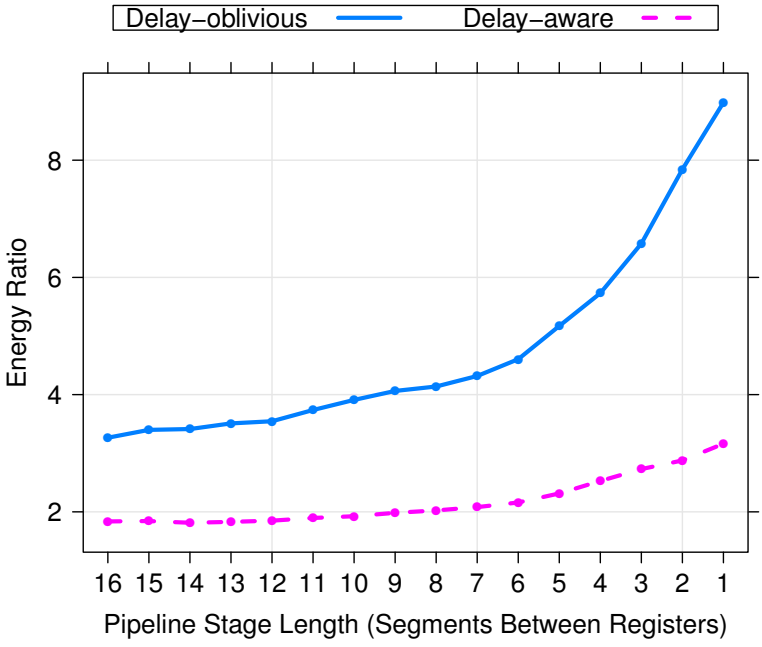


Figure 7.6: Minimum energy/operation ratio to nominal vs pipeline stage length (`mult16`, 22 nm LP)

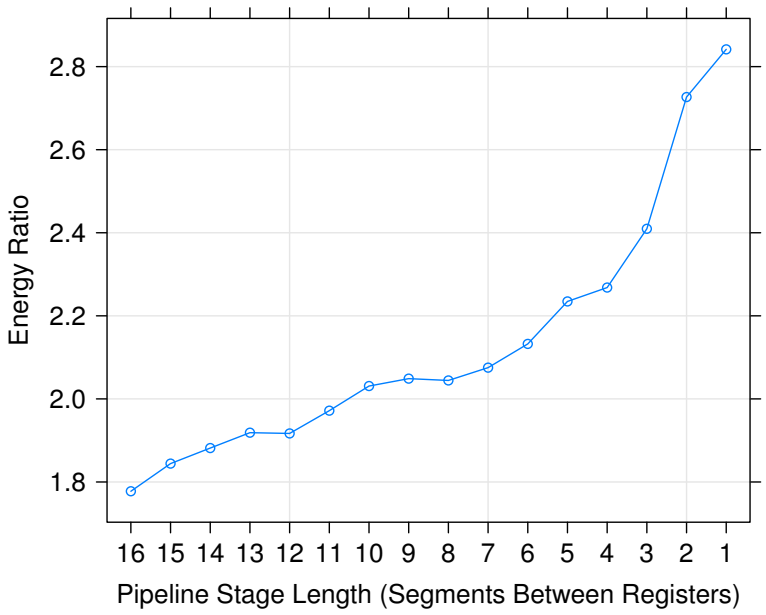


Figure 7.7: Minimum energy/operation ratio of energy-oblivious/energy-aware routing vs pipeline stage length (`mult16`, 22 nm LP)

stant. At low V_{dd} the delay-oblivious router produces routes with enough delay overhead at short pipeline stage lengths that leakage energy/operation remains relatively constant at minimum energy. This implies that, because of variation, we see no minimum energy/operation benefit to increasing pipelining.

The delay-aware router is able to restore these energy/operation benefits of pipelining by routing faster relatively to the delay-oblivious case with shorter pipeline lengths. Figure 7.6 plots the minimum energy/operation ratio to nominal of both routers. In the unpipelined case, the energy margin due to variation is 3.4×; this margin increases to nearly 9× for the fully pipelined multiplier. The delay-aware router is able to maintain margins between $1.9-3.1\times$. Figure 7.7 plots the minimum energy/operation ratio of delay-oblivious to delay-aware routing. We see that the energy savings of delay-aware routing increase as we increase pipelining, from approximately $1.8 - 2.8\times$. This means that moving from a combinational circuit to a fully pipelined circuit improves the benefit of delay-aware routing by $1.55\times$. Hence, we expect to see even better results for more heavily pipelined benchmarks because the ability of delay-aware routing to substantially reduce delay (and therefore energy) margins.

Table 7.1: Multiplier benchmark characteristics

Benchmark	CLBs	LUTs	Min Chan Width	Nets	Crit Path Segments	Registered
mult4	8×8	256	8	101	1	Yes
mult16	32×32	4096	10	1771	1	Yes
mult64	128×128	65536	12	29091	1	Yes

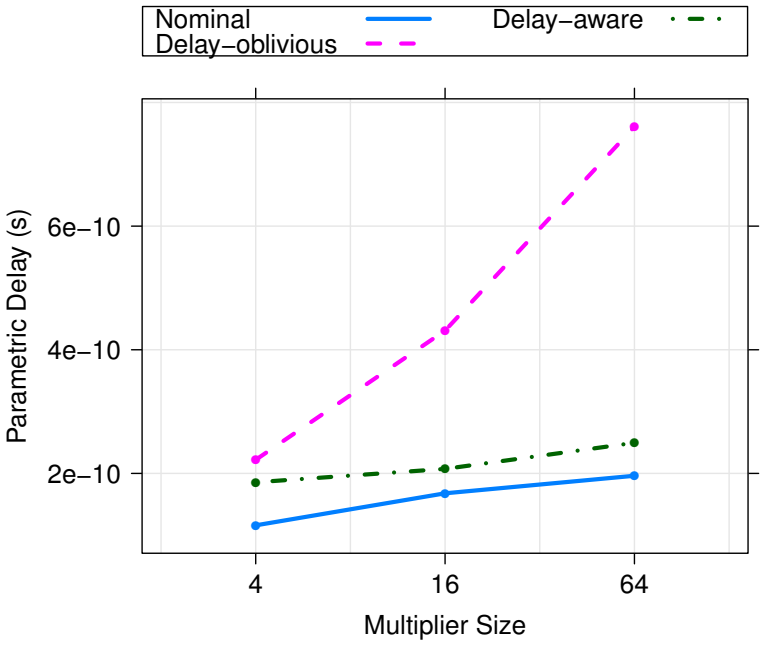


Figure 7.8: Delay vs multiplier size (22 nm LP, $V_{dd} = 600$mV)

7.2 Circuit Size

In addition to pipeline depth, we can also vary the size of our multiplier to measure how the delay and energy savings of delay-aware routing scale as a function of circuit size. Table 7.1 lists the circuit characteristics of the three multipliers we examined: 4×4, 16×16, and 64×64. The 64×64 multiplier is over an order of magnitude larger than the largest Toronto20 circuit. We use the fully pipelined case for each of these designs.

Larger circuits have many more switches, which means that we sample farther out on the V_{th} distribution when assigning threshold voltages to transistors. This means larger delay variance, which should increase the delay margins of delay-oblivious routing. The 16-bit multiplier contained 1771 critical nets; for the 64-bit multiplier we see 29091 critical nets. This increase in delay should also manifest as an increase in the minimum energy/operation margins, as we saw in the previous section.

Figure 7.8 plots delay as a function of multiplier size at 22 nm, again with V_{dd} fixed at 600 mV. Although our multiplier is fully pipelined, we see that as we increase circuit size the delay of the

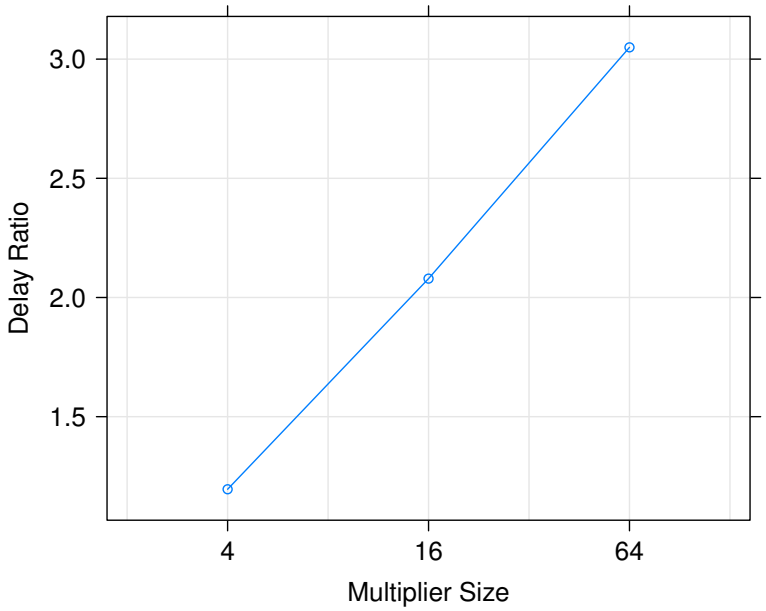


Figure 7.9: Delay ratio of delay-oblivious/delay-aware routing vs multiplier size (22 nm LP, V_{dd} = 600mV)

nominal, no variation multiplier, delay increases very slightly. This is due to the minimum channel width router assigning slightly larger channels widths for the larger multipliers, which increases wire length. Examining the delay-oblivious router, we see that the delay margins induced by variation increase as we scale up circuit size. Delay-aware routing is able to reduce those margins; Figure 7.9 plots the ratio of delay-oblivious to delay-aware routing. We see that moving from a 4-bit to 64-bit multiplier, the delay improvement of delay-aware routing increases from approximately $1.1 - 3\times$.

Figure 7.10 plots minimum energy/operation as a function of multiplier size. As we increase the size of the multiplier by powers of 4 we see that energy also increases exponentially. To determine the energy savings of delay-aware routing, Figure 7.11 plots the minimum energy/operation ratio of delay-oblivious to delay-aware routing. As in the previous section, we see an energy savings of $2.8\times$ for the fully pipelined 16-bit multiplier. For the smaller, 4-bit multiplier we only see an energy savings of $1.6\times$, while for the larger 64-bit multiplier we see an improvement of $3.2\times$. Therefore, in addition to seeing better results for increased pipelining, we also expect to see increased minimum energy/operation savings as we move to larger benchmarks.

7.3 V_{th} Variation

The PTM models predict $3\sigma_{V_{th}}$ as shown in Table 3.1. These values of σ are calculated using the *Avt* method described in Section 3.1.2. While the *Avt* method is quite accurate and typically used

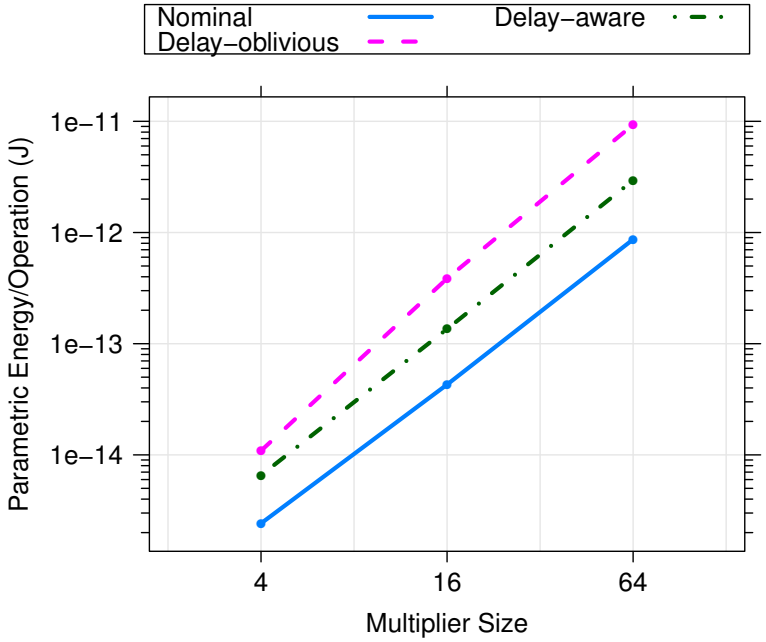


Figure 7.10: Minimum energy/operation vs multiplier size (22 nm LP)

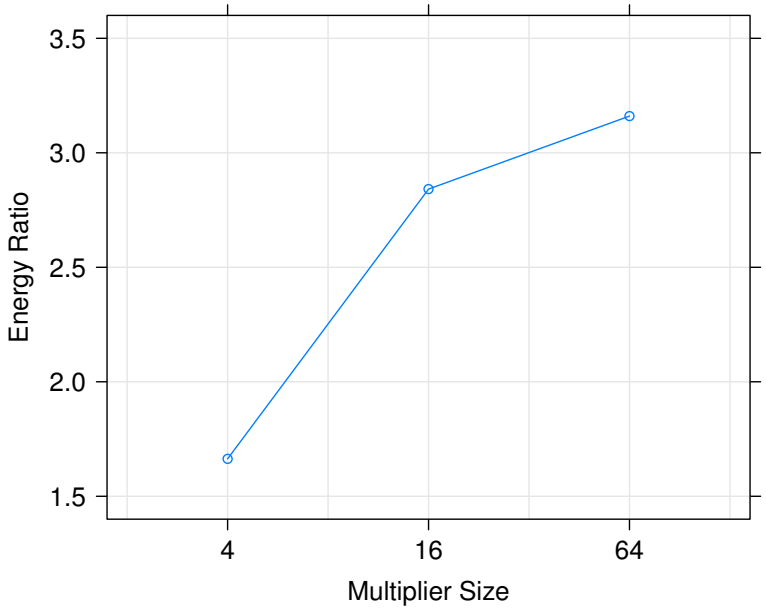


Figure 7.11: Minimum energy/operation ratio of energy-oblivious/energy-aware routing vs multiplier size (22 nm LP)

Year	2013	2014	2015	2016	2017	2018	2019	2020	2021
Feature Size (nm)	27	24	21	18.9	16.9	15	13.4	11.9	10.6
ITRS $3\sigma_{V_{th}}/\sigma_{V_{th}}$ Variation (%)	58	81	81	81	81	112	112	112	112
PTM $3\sigma_{V_{th}}/\sigma_{V_{th}}$ Variation (%)	-	-	35	-	-	42	-	50	-

Table 7.2: ITRS predicted V_{th} variation (Tables PIDS2 and DESN9 in [3])

in industry, it is still only a predictive technique. Another commonly used way of predicting V_{th} variation is to cite the $3\sigma_{V_{th}}/\mu_{V_{th}}$ values predicted by the ITRS [3]. The ITRS uses MASTAR [1] to generate technology predictions; in general, the V_{th} variation values in the ITRS considered to be pessimistically high. Table 7.2 shows the V_{th} variation predicted by the ITRS as a function of feature size, obtained by ITRS Tables PIDS2 and DESN9. We can see that the magnitude of V_{th} variation predicted by the ITRS is indeed high; for example, the ITRS predicts $3\sigma/\mu$ variation at 22 nm to be 81%, which is approximately 2.3× larger than the PTM predictions.

Because predictions of V_{th} can vary, it is useful to examine the sensitivity of the energy savings of component-specific to the magnitude of V_{th} variation for a fixed feature size. Figure 7.1 plots the ratio of the minimum energy of delay-oblivious and delay-aware to nominal as a function of V_{th} variation for `des` at 22 nm. The plot is marked at values of $\sigma_{V_{th}}$ predicted by the PTM and ITRS for 22 nm. At low $\sigma_{V_{th}}$ the energy overhead of component-specific mapping is relatively small; at $\sigma_{V_{th}} = 10$mV the ratio of delay-oblivious to nominal is only 1.6×. Component-specific mapping is able to close the gap to 1.2×, for an energy savings of 1.33×. As $\sigma_{V_{th}}$ increases, the energy overhead of delay-oblivious mapping increases, up to 4.81× at the PTM predicted variation and 9.68× at the ITRS predicted variation. Delay-aware mapping is able to close this gap at the PTM and ITRS points, for an energy savings of 2.66× and 2.80× respectively. In general, we see that the energy benefits of delay-aware mapping increase as the magnitude of V_{th} variation increases.

7.4 Feature Size

Thus far we have focused on the 22 nm node for consistency and to constrain our parameter space for developing optimizations. To analyze how our results scale with respect to feature size, we plot minimum energy/operation as a function of technology in Figure 7.13 for the nominal, no variation case and both the delay-oblivious and delay-aware cases. As technology scales we expect nominal energy/operation to decrease due to reduced capacitance; however, V_{th} variation increases substantially (Table 3.1) and will induce energy margins. When examining the no variation case, we see energy/operation does decrease with each technology generation, as expected. We also see substantial energy margins in the delay-oblivious case, up to 8.44× in the worst case at 12 nm. Delay-aware mapping is able to reduce this energy margins across all technologies, with a reduction of 2.98× in the best case at 12 nm.

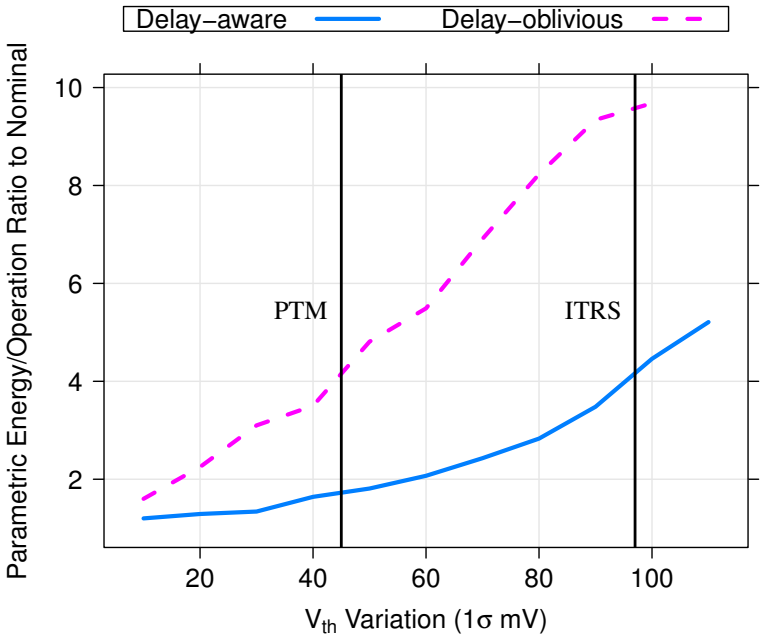


Figure 7.12: Minimum energy/operation ratio to nominal vs V_{th} sigma (`des`, 22 nm LP)

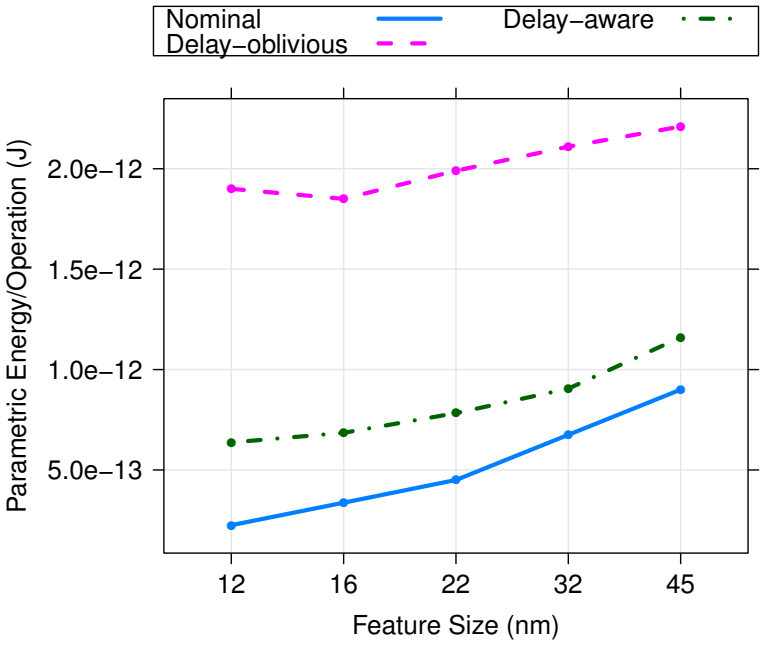


Figure 7.13: Minimum energy/operation vs feature size (`des`)

Table 7.3: Minimum energy/operation variation-induced margins and component-specific mapping benefits

Design	**Margin (Oblivious/Nominal)**					**Benefit (Oblivious/Knowledge)**				
	45nm	**32nm**	**22nm**	**16nm**	**12nm**	**45nm**	**32nm**	**22nm**	**16nm**	**12nm**
alu4	2.42	3.24	4.25	5.23	7.45	1.95	2.59	2.43	2.27	2.78
apex2	2.34	3.20	3.84	5.88	7.05	2.09	2.32	2.33	2.79	2.36
apex4	2.85	3.10	4.76	6.22	8.57	1.54	2.11	2.52	2.96	2.74
bigkey	1.93	2.94	3.88	4.99	8.89	1.58	2.45	2.64	2.67	2.63
clma	1.84	2.81	4.65	5.78	8.68	1.37	2.23	2.64	2.90	3.47
des	2.45	3.12	4.42	5.49	8.44	1.90	2.33	2.54	2.70	2.98
diffeq	1.82	3.45	4.55	5.87	7.29	1.73	3.03	2.68	2.64	2.62
dsip	2.45	3.51	4.40	5.70	6.99	1.88	2.42	2.84	2.84	2.55
elliptic	2.23	3.68	4.39	5.71	8.24	1.54	2.20	3.16	3.02	3.05
ex1010	2.46	2.90	3.90	4.88	8.65	1.68	1.97	2.23	2.57	3.04
ex5p	2.40	3.25	4.12	5.32	8.68	1.86	1.99	2.62	2.61	2.90
frisc	2.37	3.29	4.23	5.55	9.02	1.68	2.35	2.53	2.66	3.00
misex3	2.47	3.10	4.33	5.34	9.12	1.65	2.31	2.61	2.54	3.59
pdc	1.94	2.45	4.21	5.45	8.87	1.72	1.86	2.63	2.81	3.10
s298	1.55	3.15	4.77	5.68	8.55	1.26	2.28	3.08	3.17	3.53
s38417	1.58	3.55	4.54	5.38	8.42	1.39	2.75	2.52	2.54	2.86
s38584.1	1.58	3.22	4.87	6.23	8.98	1.26	2.48	2.88	3.19	3.25
seq	2.14	2.84	3.99	5.69	8.67	1.89	1.81	2.35	2.66	2.79
spla	2.56	2.90	4.22	5.99	7.94	1.77	1.86	2.54	2.58	2.65
tseng	2.65	3.01	4.00	6.01	8.88	2.17	2.08	2.22	3.25	3.46
Geomean	2.17	3.12	4.31	5.61	8.34	1.68	2.25	2.59	2.76	2.95

This energy margins induced by variation are large enough that we can see it no longer beneficial in terms of minimum energy to scale from 16 nm to 12 nm: the delay-oblivious case shows a net *increase* in energy when scaling from 16 nm to 12 nm. Here, the energy savings from scaling down to a smaller feature size is lost due to the energy overhead of variation. This corresponds to the result from [20] that demonstrated a similar trends for ASICs, and a similar turning point at 32 nm. When examining delay-aware mapping, we see energy/operation decrease across the range of 45 nm–12 nm. *This means that component-specific mapping is effectively able to allow technology scaling to continue delivering reductions in minimum operating energy for at least one more technology generation.*

Table 7.3 compiles our results for delay-aware and delay-oblivious mapping across all the Toronto20 benchmarks and PTM feature sizes. We tabulate the energy margin induced by variation (delay-oblivious/nominal energy/operation) and the energy savings achieved by component-specific mapping. Across all benchmarks and technologies, we observe minimum energy/operation savings of 1.68–2.95×.

Chapter 8

Future Work

This work is not a complete solution to component-specific mapping as it does not solve the issues surrounding measurement and the one-mapping-fits-all model for current FPGA development. Therefore, the most pressing future work would be to advance the partial solutions described in Chapter 6.

However, there are a number of possible improvements that could be made towards the goal of demonstrating the potential benefits of component-specific mapping:

Improved circuit modeling: Chapter 3 described in detail the circuits simulated for this work and the methodology used for developing accurate and fast delay and energy estimates for academic FPGA CAD tools. These models could be expanded to include a number of improvements.

Input slew rates have non-negligible impact on gate delay, developing a simulation model that accounts for slew rate would make our results more accurate. This would also involve modifying the timing analysis routines of VPR to account for prior gate delays when calculating the delay through a given gate.

Improved circuits: Only particular implementations of the FPGA circuit primitives were explored in this work. There are a number of possible circuit implementation of LUTs, multiplexers and switches. Some other circuit topologies have been explored in [64]. A thorough exploration of these other possible switch implementations may yield a design with lower total energy dissipation. Correct modeling of these switch types would require more extensive modifications to VPR.

Improved architecture modeling: There are several other additional sources of potentially significant energy dissipation on modern FPGA architectures that are not modeled in this work: I/O drivers, embedded DSPs, and embedded memories.

While none of these block are likely to contribute nearly as much energy as FPGA interconnect, because they are hard, non-reconfigurable blocks, in order to maintain high-yield they may need

to be sized up, which would increase their energy dissipation proportionately. Memories have well developed techniques for defect tolerance such as ECC and sparing [126]. However, it may be useful to examine ways to use coarse-grained sparing for other blocks to help tolerate some amount of failure. Another possibility is to explore making these block partially reconfigurable for the sole purpose of increased reliability.

Additional benchmarks: This work only examined a hand mapped multiplier and the Toronto20 benchmark set, which is almost 15 years old [17]. We see in Table 4.1 that these benchmarks are very small and mostly purely combinational. Very recently the FPGA community has compiled a set of modern benchmarks [96]. These benchmarks are significantly larger and more pipelined than the Toronto20. Section 7.1 and 7.2 attempted to scale our simple, pipelined multiplier up to demonstrate the potential increased energy savings from using component-specific mapping on large, pipelined circuits. Verifying these results for real circuits would prove useful.

Fully component-specific CAD flow: This work only examined random sources of variation. While these sources are anticipated to be the most dominant future sources of variation, regional and global variation are still significant. Future work would include modeling all possible sources of variation. To combat regional variation, component-specific placement and clustering may be essential. A complete delay-aware CAD flow would be able to demonstrate the full potential of component-specific mapping to tolerate variation.

Benchmark analysis: Section 7.2 showed a correlation between circuit size and pipeline depth and the energy savings from component-specific mapping. There are a number of other ways to quantify the structure of benchmarks, and it may prove that other benchmark metrics correlate strongly to increased benefits from component-specific mapping. For example, near-critical paths are important under variation because there is a probability that they will become critical. The larger the number of near-critical paths, the less likely that oblivious routing will find an adequate solution. However, given enough routing flexibility component-specific routing should be able to compensate for those near-critical paths that become critical. It may be useful to determine the correlation between number of near-critical paths in a benchmark using techniques such as SSTA to component-specific routing benefits. Ideally, one would be able to perform a simple set of circuit analyses on the structure of a given benchmark to predict the energy reduction possible from component-specific mapping.

Additional technologies: At the onset of this work, commercial FPGA feature sizes were at 90 nm. By the time this work was completed, 28 nm FPGAs are nearing production. The range of technologies examined in this work (45 nm, 32 nm, 22 nm, 16 nm, 12 nm) is no longer completely

Metric	Ratio (FPGA/ASIC)			
	Logic-Only	**Logic, DSP**	**Logic, Memory**	**Logic, Memory, DSP**
Area	35	25	33	18
Delay	3.4	3.5	3.5	3.0
Dynamic Power	14	12	14	7.1

Table 8.1: FPGA/ASIC gap [59]

predictive—only three nodes project into the future. It would be beneficial to quantify the benefits of component-specific mapping for more technology nodes into the future. In particular we would like to see if component-specific mapping is still able to maintain a net minimum energy/operation reduction under variation for the next technology node after 12 nm.

Additionally, there are a number of significant process changes that are on the horizon for FPGAs. FinFET transistors are currently in production for commercial microprocessors [52] and will likely be used for FPGAs. Other technologies such as fully depleted SOI [37] and CNTFETs [121] are also candidate technologies for FPGAs. The PTM group has begun to develop predictive models for some of these technologies that could be used for additional component-specific mapping experiments [108].

Component-specific mapped FPGA vs ASIC comparison: This work compared a component-specific mapped FPGA to an oblivious mapped FPGA. A more interesting comparison may be to compare the component-specific mapped FPGA to an ASIC, to demonstrate how close an FPGA can come to an ASIC in terms of minimum energy/operation under variation.

Kuon and Rose [59] began to quantify the FPGA/ASIC power gap by synthesizing a set of benchmark circuits for both a 90 nm FPGA and a 90 nm ASIC process using a standard, commercial EDA flow without any specific power optimizations. They compared area, delay and dynamic power of the ASIC to that of the FPGA in several different configurations: logic-only and logic with different combinations of modern embedded structures (memory and DSPs). Table 8.1 summarizes their results, showing that on average a logic only FPGA without any optimizations utilizes 14× the dynamic power of a process equivalent ASIC. With non-configurable embedded memories and DSPs this gap decreases to 7×. Embedded elements do not contain programmable interconnect, saving capacitance and hence dynamic power. When comparing static power, they found that the gap is roughly correlated (correlation coefficient of 0.8) to the area overhead of 18–32×, with embedded blocks again reducing power overhead.

These results, however, were generated without quantifying the impact of variation. It may be possible in future technologies that the energy margin induced by variation may be so large in ASICs that FPGAs using component-specific mapping can close this energy gap.

Recreating the Kuon and Rose experiments using industrial CAD flows but with modeling variation and low V_{dd} operation would prove to be very challenging. It may however be possible to

perform some simple estimations and limit studies to develop an ASIC power model.

The dynamic energy overhead of FPGAs lies in the programmable interconnect. This interconnect adds switched capacitance from drivers, and increases area which increases wire lengths. Signals in ASICs do not need to traverse several switches to reach their destination, and will travel shorter wire lengths. The reduction in switched capacitance is the main difference between the dynamic energy of our delay-oblivious case from a real ASIC; however, because ASIC paths do not travel through several switches, they will experience far less path length averaging than FPGAs. Therefore, we expect an ASIC to perform much more similarly to our fully pipelined multiplier circuit than the Toronto20 benchmarks.

We saw that in the best case at 12 nm component-specific mapping can save approximately 3× energy; we also saw that pipelining typically increases energy savings by 1.5× because of the delay increase of delay-oblivious mapping. Therefore, if we can save 4.5× minimum energy/operation, we may be able to close the FPGA/ASIC gap (assuming no embedded blocks) from 14× down to 3.1×. If we assume some sort of embedded blocks that allow variation hardening, but maintain the same energy ratio that Kuon and Rose measured, the 7.1× gap may be reduced to 1.5×. An energy overhead of only 50% from using an FPGA would be very significant.

One simple study that might help to validate these estimates would be to use our existing FPGA infrastructure, but to change the delay and energy cost of interconnect. For example, we could assume that switches cost nothing in terms of delay and energy, and we could scale down wire lengths proportional to the area ratios estimated by Kuon and Rose. To model logic, we could perform technology mapping targeted to 2-input NAND gates instead of LUTs. We would then need to simply develop a composable device model for NAND gates to develop delay and energy estimates. These two relatively simple modifications may be enough to provide a rough idea on how the energy of an ASIC might compare to an FPGA under variation, and by how much an FPGA with component-specific mapping would be able to close the energy gap.

Chapter 9

Conclusions

In this thesis we have quantified the benefits of post-fabrication, component-specific mapping for FPGAs targeting low-energy operation under variation. We have measured the impact of variation at low V_{dd}, demonstrating that the energy margins induced by variation are substantial (2.17–8.34× from 45 nm–12 nm). These margins are large enough that continued scaling to 12 nm does not result in a net reduction in energy. We have shown that component-specific mapping can reduce these energy margins significantly, saving 1.68–2.95× across all benchmarks and technologies. This means we can continue minimum-energy scaling to 12 nm and possibly beyond.

To obtain these results, we developed accurate FPGA circuit and CAD models using HSPICE and VPR. We then extensively explored the FPGA architectural design space to determine circuit and architectural optimizations that enhance the benefits of component-specific mapping. We showed that power gating, gate sizing, interconnect sparing, and LUT remapping optimizations increase the energy savings of component-specific mapping by 1.54×. We propose an architecture that uses small 4×4 clusters, minimum length segments, 2× sized switches, 4 extra CLB I/O pins, and mixed 2× and 8× sized LUTs, to maximize the benefits of component-specific mapping.

We highlight some of the key lessons of this thesis as follows:

- **Accurate circuit and CAD models are important:** When characterizing the energy and performance of an FPGA over a very wide range of V_{dd} and V_{th}, it is critical to use accurate circuit models. Without accurate device models, predictions of delay and energy at low voltages can contain substantial errors (potentially orders of magnitude) than can accumulate. Without timing-targeted routing, stable routed results are difficult to obtain given V_{th} variation.

- **Defects occur at non-negligible rates at low V_{dd}:** While variation does slow devices down, for our relatively unpipelined benchmarks, path length averaging reduces the delay impact of variation. However, tolerating switching failures at low V_{dd} are very important when trying to minimize energy.

- **Defects for active nodes that cannot be overdriven matter:** FPGAs consist of several

circuit primitives (i.e., inverters, SRAMs, passgates, muxes) that are each subject to failure. However, nodes that are inactive (SRAMs) or can be statically overdriven (passgates) can be specifically hardened. Circuits such as inverters that actively switching are harder to design for increased reliability.

- **Power gating is important for minimizing energy:** Because FPGAs have so many unused resources, leakage energy/operation is substantial. Extraneous leakage energy/operation raises the minimum V_{dd} point, hindering minimum-energy scaling. Power gating is an effective technique for extending scaling to lower voltages.

- **Minimum sized gates are not always energy-optimal under V_{th} variation:** Without variation, minimum sized gates are always energy-minimal; however, we have shown cases where minimum sized gates actually dissipate more energy due to decreased reliability and the required scaling up of V_{dd}. In some cases larger, more reliable gates enable scaling to lower voltages.

- **FPGAs are vulnerable to CBox defects, while SBox defects can be avoided:** When examining both sizing and sparing, we found it much more important to increase the size and connectivity of CBox switches. These switches bottleneck signals in and out of CLBs, and are commonly allocated without a large amount of natural sparing. The flexibility of FPGA interconnect allows for more defect avoidance with SBox switches.

- **Defect-only routing approaches the benefits of delay-aware routing:** While delay information provides high quality, lower energy routes, defect-only routing is adequate in reducing energy, costing only 30% energy.

- **Minimum-energy scaling will end with unchecked V_{th} variation:** Without techniques like component-specific mapping to deal with variation, it will become more beneficial to use older feature sizes when designing parts for very low energy.

Component-specific mapping is not a solved problem; actually performing fine-grained measurement and per-chip mapping without excessive CAD costs are key challenges that must be overcome. Future work will be necessary to make actual, complete component-specific mapping on real parts a reality. However, we have demonstrated that the energy benefit of overcoming these challenges is large. Component-specific mapping relies on the very simple fact that it is always better to have more information about the characteristics of the substrate being mapped. That is, mapping circuits while ignorant to the specific physical characteristics of a device comes at a cost, in both delay and energy. Hence, we believe that **knowledge is power** [14] (or more precisely, *energy*).

Bibliography

[1] Model for Assessment of CMOS Technology and Roadmaps (MASTAR). <http://www.itrs.net/Links/2007ITRS/LinkedFiles/PIDS/MASTAR5/MASTARDownload.htm> , 2010.

[2] HSPICE. <http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE> , 2010.

[3] International Technology Roadmap for Semiconductors. <http://www.itrs.net/Links/2010ITRS/Home2010.htm> , 2010.

[4] Achronix Semiconductor Corporation, Inc. *Achronix Speedster Data Sheet*, preliminary (v1.0) edition, 2010.

[5] Actel. *Design Techniques for Radiation-Hardened FPGAs*. Actel, Inc., 955 East Arques Avenue, Sunnyvale, CA 94086, 1997. Dual and TMR Application Note AC128 <https://www.actel.com/documents/Des_Tech_RH_AN.pdf> .

[6] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 3–12, New York, NY, USA, 2000. ACM.

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

[8] S. Alam, G. C. Lip, C. Thompson, and D. Troxel. Circuit level reliability analysis of Cu interconnects. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 238–243, 2004.

[9] J. Anderson and F. Najm. Power-aware technology mapping for LUT-based FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 211–218, 2002.

[10] J. Anderson and F. Najm. Low-power programmable routing circuitry for FPGAs. In *Proceedings of the International Conference on Computer-Aided Design*, 2004.

[11] A. Asenov. Random Dopant Induced Threshold Voltage Lowering and Fluctuations in Sub-0.1 μm MOSFET's: A 3-D "Atomistic" Simulation Study. *IEEE Trans. Electron Devices*, 45(12):2505–2513, December 1998.

[12] A. Asenov. Intrinsic Threshold Voltage Fluctuations in Decanano MOSFETs Due to Local Oxide Thickness Variation. *IEEE Trans. Electron Devices*, 49(1):112–119, January 2002.

[13] A. Asenov, S. Kaya, and A. R. Brown. Intrinsic Parameter Fluctuations in Decananometer MOSFETs Introduced by Gate Line Edge Roughness. *IEEE Trans. Electron Devices*, 50(5):1254–1260, May 2003.

[14] F. Bacon. *Meditationes Sacræ. De Hæresibus.* 1597.

[15] R. Baumann. Soft errors in advanced computer systems. *IEEE Design and Test of Computers*, 22(3):258–266, May–June 2005.

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

[17] V. Betz and J. Rose. FPGA Place-and-Route Challenge. <`http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html`> , 1999.

[18] S. Bijansky and A. Aziz. TuneFPGA: post-silicon tuning of dual-Vdd FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 2008.

[19] A. Bogiolo, L. Benini, and B. Riccò. Power estimation of cell-based CMOS circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 433–438, New York, NY, USA, 1996. ACM.

[20] D. Bol, R. F. Ambroise, and J.-D. D. Legat. Impact of Technology Scaling on Digital Subthreshold Circuits. In *Proceedings of the International Symposium on VLSI*, pages 179–184, 2008.

[21] S. Borkar. Microarchitecture and Design Challenges for Gigascale Integration. <`http://www.microarch.org/micro37/presentations/MICRO37f`> , December 2004. Keynote Talk at the 37th Annual IEEE/ACM International Symposium on Microarchitecture.

[22] S. Borkar. Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, November–December 2005.

[23] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 338–342, New York, NY, USA, 2003. ACM.

[24] B. Calhoun and A. Chandrakasan. Characterizing and Modeling Minimum Energy Operation for Subthreshold Circuits. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 90–95, August 2004.

[25] B. Calhoun, F. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in MTCMOS. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2003.

[26] B. Calhoun, S. Khanna, R. Mann, and J. Wang. Sub-threshold circuit design with shrinking CMOS devices. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2541–2544, May 2009.

[27] B. Calhoun, A. Wang, and A. Chandrakasan. Device sizing for minimum energy operation in subthreshold circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 95–98, October 2004.

[28] L. Cheng, D. Chen, and M. D. Wong. GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 318–323, June 2007.

[29] L. Cheng, J. Xiong, L. He, and M. Hutton. FPGA Performance Optimization via Chipwise Placement Considering Process Variations. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 1–6, 2006.

[30] C. Chow, L. Tsui, P. Leong, W. Luk, and S. Wilton. Dynamic voltage scaling for commercial FPGAs. *Proceedings of the International Conference on Field-Programmable Technology*, pages 173–180, December 2005.

[31] W. B. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider. Defect Tolerance on the TERAMAC Custom Computer. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 116–123, April 1997.

[32] A. DeHon. Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization). In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 69–78, February 1999.

[33] A. DeHon. Law of Large Numbers System Design. In S. K. Shukla and R. I. Bahar, editors, *Nano, Quantum and Molecular Computing: Implications to High Level Design and Validation*, chapter 7, pages 213–241. Kluwer Academic Publishers, Boston, 2004.

[34] A. DeHon and H. Naeimi. Seven Strategies for Tolerating Highly Defective Fabrication. *IEEE Design and Test of Computers*, 22(4):306–315, July–August 2005.

[35] A. DeHon and M. J. Wilson. Nanowire-Based Sublithographic Programmable Logic Arrays. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 123–132, February 2004.

[36] Q. Dinh, D. Chen, and M. Wong. A routing approach to reduce glitches in low power FPGAs. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 29(2):235–240, 2010.

[37] B. Doyle, S. Datta, M. Doczy, S. Hareland, B. Jin, J. Kavalieros, T. Linton, A. Murthy, R. Rios, and R. Chau. High performance fully-depleted tri-gate CMOS transistors. *Electron Device Letters, IEEE*, 24(4):263–265, April 2003.

[38] K. Eguro and S. Hauck. Enhancing timing-driven FPGA placement for pipelined netlists. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 34–37, New York, NY, USA, 2008. ACM.

[39] A. H. Farrahi and M. Sarrafzadeh. FPGA Technology Mapping for Power Minimization. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 66–77, London, UK, 1994. Springer-Verlag.

[40] D. J. Frank. Power Constrained CMOS Scaling Limits. *IBM Journal of Research and Development*, 46(2/3):235–244, March 2002.

[41] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. Irwin, and T. Tuan. A Dual-Vdd Low Power FPGA Architecture. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 145–157. Springer, 2004.

[42] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan. Reducing leakage energy in FPGAs using region-constrained placement. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 51–58, 2004.

[43] V. George, H. Zhang, and J. Rabaey. The design of a low energy FPGA. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 188–193, 1999.

[44] B. Gojman. Timing Extraction Method. *Personal Communications*, August 2011.

[45] B. Gojman and A. DeHon. VMATCH: Using Logical Variation to Counteract Physical Variation in Bottom-Up, Nanoscale Systems. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 78–87. IEEE, December 2009.

[46] S. Gupta, J. Anderson, L. Farragher, and Q. Wang. CAD techniques for power optimization in Virtex-5 FPGAs. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 85–88, 2007.

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

[48] L. He, A. Kahng, K. H. Tam, and J. Xiong. Simultaneous Buffer Insertion and Wire Sizing Considering Systematic CMP Variation and Random Leff Variation. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 26(5):845–857, May 2007.

[49] M. Hioki, T. Kawanami, T. Tsutsumi, T. Nakagawa, T. Sekigawa, and H. Koike. Evaluation of granularity on threshold voltage control in flex power FPGA. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 17–24, December 2006.

[50] Y. Hu, Y. Lin, L. He, and T. Tuan. Physical synthesis for FPGA interconnect power reduction by dual-Vdd budgeting and retiming. *ACM Transactions on Design Automation of Electronic Systems*, 13(2):1–29, 2008.

[51] P. Jamieson, W. Luk, S. J. Wilton, and G. A. Constantinides. An energy and power consumption analysis of FPGA routing architectures. *Proceedings of the International Conference on Field-Programmable Technology*, pages 324–327, December 2009.

[52] E. Karl, Y. Wang, Y.-G. Ng, Z. Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr. A 4.6GHz 162Mb SRAM design in 22nm tri-gate CMOS technology with integrated active VMIN-enhancing assist circuitry. In *Proceedings of the International Solid-State Circuits Conference*, pages 230–232, February 2012.

[53] K. Katsuki, M. Kotani, K. Kobayashi, and H. Onodera. A Yield and Speed Enhancement Scheme under Within-die Variations on 90nm LUT Array. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 601–604, 2005.

[54] T. Kawanami, M. Hioki, Y. Matsumoto, T. Tsutsumi, T. Nakagawa, T. Sekigawa, and H. Koike. Optimal set of body bias voltages for an FPGA with field-programmable Vth components. *Proceedings of the International Conference on Field-Programmable Technology*, pages 329–332, December 2006.

[55] S. Kirkpatrik, C. D. Gellatt, Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[56] M. Klein. The Virtex-4 Power Play. *Xcell Journal*, (52):16–19, Spring 2005.

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

[58] A. Kumar and M. Anis. FPGA Design for Timing Yield Under Process Variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):423–435, March 2010.

[59] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design*, 26(2):203–215, February 2007.

[60] E. Kusse and J. Rabaey. Low-Energy Embedded FPGA Structures. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 155–160, August 1998.

[61] J. Kwong and A. P. Chandrakasan. Variation-Driven Device Sizing for Minimum Energy Subthreshold Circuits. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 8–13, 2006.

[62] J. Lamoureux and S. Wilton. On the interaction between power-aware FPGA CAD algorithms. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Computer Society Washington, DC, USA, 2003.

[63] J. Lamoureux and S. Wilton. Activity Estimation for Field-Programmable Gate Arrays. *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 1–8, August 2006.

[64] E. Lee, G. Lemieux, and S. Mirabbasi. Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays. *Journal of Signal Process. Systems*, 51(1):57–76, April 2008.

[65] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and Single-Driver Wires in FPGA Interconnect. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 41–48, December 2004.

[66] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan. Architectural enhancements in Stratix-III and Stratix-IV. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 33–42. ACM, 2009.

[67] F. Li, D. Chen, L. He, and J. Cong. Architecture evaluation for power-efficient FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, page 175, New York, New York, USA, 2003. ACM Press.

[68] F. Li, Y. Lin, and L. He. FPGA power reduction using configurable dual-Vdd. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 735–740. ACM, 2004.

[69] F. Li, Y. Lin, and L. He. Vdd programmability to reduce FPGA interconnect power. In *Proceedings of the International Conference on Computer-Aided Design*, pages 760–765. IEEE, 2004.

[70] F. Li, Y. Lin, L. He, and J. Cong. Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, page 4250. ACM, 2004.

[71] H. Li, S. Katkoori, and W.-K. Mak. Power minimization algorithms for LUT-based FPGA technology mapping. *ACM Transactions on Design Automation of Electronic Systems*, 9(1):33–51, January 2004.

[72] Y. Lin and J. Cong. Power modeling and characteristics of field programmable gate arrays. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 24(11):1712–1724, November 2005.

[73] Y. Lin and L. He. Dual-Vdd Interconnect With Chip-Level Time Slack Allocation for FPGA Power Reduction. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 25(10):2023–2034, October 2006.

[74] Y. Lin and L. He. Stochastic Physical Synthesis for FPGAs with Pre-routing Interconnect Uncertainty and Process Variation. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 80–88, 2007.

[75] Y. Lin, L. He, and M. Hutton. Stochastic physical synthesis considering prerouting interconnect uncertainty and process variation for FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(2):124, 2008.

[76] Z.-M. Ling, J. Cho, R. W. Wells, C. S. Johnson, and S. G. Davis. Method of Using Partially Defective Programmable Logic Devices. United States Patent Number: 6,664,808, December 16 2003.

[77] N. Lotze, J. Goppert, and Y. Manoli. Timing modeling for digital sub-threshold circuits. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*, pages 299–302, March 2010.

[78] G. Lucas, C. Dong, and D. Chen. Variation-aware placement for FPGAs with multi-cycle statistical timing analysis. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 177–180, New York, New York, USA, 2010. ACM.

[79] R. Manohar. Reconfigurable asynchronous logic. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 13–20, 2006.

[80] Y. Matsumoto, M. Hioki, T. Kawanami, H. Koike, T. Tsutsumi, T. Nakagawa, and T. Sekigawa. Suppression of Intrinsic Delay Variation in FPGAs using Multiple Configurations. *Transactions on Reconfigurable Technology and Systems*, 1(1), March 2008.

[81] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 111–117, 1995.

[82] N. Mehta, R. Rubin, and A. DeHon. Limit Study of Energy & Delay Benefits of Component-Specific Routing. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 97–106, 2012.

[83] A. Mishchenko, S. Chatterjee, and R. K. Brayton. Improvements to Technology Mapping for LUT-Based FPGAs. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 26(2):240–253, February 2007.

[84] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 24(12):1859–1880, December 2005.

[85] G. Nabaa, N. Azizi, and F. N. Najm. An Adaptive FPGA Architecture with Process Variation Compensation and Reduced Leakage. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 624–629, 2006.

[86] K. Nagaraj and S. Kundu. Process Variation Mitigation via Post Silicon Clock Tuning. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 227–232, 2009.

[87] S. Narendra, V. De, D. Antoniadis, A. Chandrakasan, and S. Borkar. Scaling of stack effect and its application for leakage reduction. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 195–200, New York, NY, USA, 2001. ACM.

[88] S. Nassif. Delay variability: sources, impacts and trends. In *Proceedings of the International Solid-State Circuits Conference*, pages 368–369, 2000.

[89] S. R. Nassif, N. Mehta, and Y. Cao. A Resilience Roadmap. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*, March 2010.

[90] E. Packard. *The Cave of Time*. Bantam Books, 1979.

[91] K. Poon, S. Wilton, and A. Yan. A detailed power model for field-programmable gate arrays. *ACM Transactions on Design Automation of Electronic Systems*, 10:279–302, 2005.

[92] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[93] A. Rahman, S. Das, T. Tuan, and S. Trimberger. Determination of Power Gating Granularity for FPGA Fabric. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 9–12, September 2006.

[94] A. Ramalingam, S. V. Kodakara, A. Devgan, and D. Z. Pan. Robust analytical gate delay modeling for low voltage circuits. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 61–66, Piscataway, NJ, USA, 2006. IEEE Press.

[95] J. Rose et al. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. <`http://www.eecg.utoronto.ca/vpr/`> , 2009.

[96] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. The VTR project: architecture and CAD for FPGAs from verilog to routing. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 77–86, New York, NY, USA, 2012. ACM.

[97] E. Rosenbaum, P. Lee, R. Moazzami, P. Ko, and C. Hu. Circuit reliability simulator-oxide breakdown module. In *Technical Digest of the IEEE International Electron Device Meeting*, pages 331–334, December 1989.

[98] R. Rubin and A. DeHon. Choose-Your-Own-Adventure Routing: Lightweight Load-Time Defect Avoidance. *Transactions on Reconfigurable Technology and Systems*, 4(4), December 2011.

[99] R. Rubin and A. DeHon. Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 173–176, 2011.

[100] J. Ryan and B. Calhoun. A sub-threshold FPGA with low-swing dual-VDD interconnect in 90nm CMOS. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 1–4, September 2010.

[101] I. Sauciuc, R. Prasher, J.-Y. Chang, H. Erturk, G. Chrysler, C.-P. Chiu, and R. Mahajan. Thermal Performance and Key Challenges for Future CPU Cooling Technologies. *ASME Conference Proceedings*, 2005(42002):353–364, 2005.

[102] A. Scholten, H. Tromp, L. Tiemeijer, R. Van Langevelde, R. Havens, P. De Vreede, R. Roes, P. Woerlee, A. Montree, and D. Klaassen. Accurate thermal noise model for deep-submicron CMOS. In *Technical Digest of the IEEE International Electron Device Meeting*, pages 155–158, 1999.

[103] L. Schwiebert, S. K. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *Proceedings of the International Conference on Mobile Computing and Networking*, pages 151–165, New York, NY, USA, 2001. ACM.

[104] P. Sedcole and P. Y. K. Cheung. Within-die Delay Variability in 90nm FPGAs and Beyond. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 97–104, 2006.

[105] P. Sedcole and P. Y. K. Cheung. Parametric Yield Modeling and Simulations of FPGA Circuits Considering Within-Die Delay Variations. *Transactions on Reconfigurable Technology and Systems*, 1(2), June 2008.

[106] L. Shang, A. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, page 164. ACM, 2002.

[107] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 59–66, New York, NY, USA, 2002. ACM.

[108] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao. Exploring sub-20nm FinFET design with predictive technology models. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 283–288, New York, NY, USA, 2012. ACM.

[109] S. Sivaswamy and K. Bazargan. Statistical Analysis and Process Variation-Aware Routing and Skew Assignment for FPGAs. *Transactions on Reconfigurable Technology and Systems*, 1(1):1–35, 2008.

[110] J. S. Swarz, V. Betz, and J. Rose. A Fast Routability-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 140–149. ACM/SIGDA, February 1998.

[111] E. Takeda and N. Suzuki. An empirical model for device degradation due to hot-carrier injection. *Electron Device Letters, IEEE*, 4(4):111–113, April 1983.

[112] S. Tam, R. D. Limaye, and U. N. Desai. Clock Generation and Distribution for the 130-nm Itanium 2 processor with 6-MB on-die L3 cache. *IEEE Journal of Solid State Circuits*, 39(4):636–642, 2004.

[113] S. M. Trimberger. Utilizing multiple test bitstreams to avoid localized defects in partially defective programmable integrated circuits. United States Patent Number: 7,424,655, September 9 2008.

[114] T. Tuan and B. Lai. Leakage Power Analysis of a 90nm FPGA. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, page 57. IEEE, 2003.

[115] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao. A 90-nm Low-Power FPGA for Battery-Powered Applications. *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, 26(2):296–300, 2007.

[116] K. Vorwerk, M. Raman, J. Dunoyer, A. Kundu, and A. Kennings. A technique for minimizing power during FPGA placement. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 233–238, 2008.

[117] O. Weber, O. Faynot, F. Andrieu, C. Buj-Dufournet, F. Allain, P. Scheiblin, J. Foucher, N. Daval, D. Lafond, L. Tosti, L. Brevard, O. Rozeau, C. Fenouillet-Beranger, M. Marin, F. Boeuf, D. Delprat, K. Bourdelle, B.-Y. Nguyen, and S. Deleonibus. High immunity to threshold voltage variability in undoped ultra-thin FDSOI MOSFETs and its physical understanding. In *Technical Digest of the IEEE International Electron Device Meeting*, pages 1–4, December 2008.

[118] R. W. Wells, Z.-M. Ling, R. D. Patrie, V. L. Tong, J. Cho, and S. Toutounchi. Application-Specific Testing Methods for Programmable Logic Devices. United States Patent Number: 6,817,006, November 9 2004.

[119] N. H. E. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, third edition, 2005.

[120] S. Wilton, S. Ang, and W. Luk. The impact of pipelining on energy per operation in field-programmable gate arrays. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 719–728. Springer, 2004.

[121] S. J. Wind, J. Appenzeller, R. Martel, V. Deycke, and P. Avouris. Vertical Scaling of Carbon Nanotube Field-Effect Transistors using Top Gate Electrodes. *Applied Physics Letters*, 80(20):3817–3819, 2002.

[122] C. Wong, A. Martin, and P. Thomas. An architecture for asynchronous FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 170–177, 2003.

[123] H. Wong, L. Cheng, Y. Lin, and L. He. FPGA device and architecture evaluation considering process variations. In *Proceedings of the International Conference on Computer-Aided Design*, page 24. IEEE Computer Society, 2005.

[124] J. S. Wong, P. Sedcole, and P. Y. K. Cheung. Self-Measurement of Combinatorial Circuit Delays in FPGAs. *Transactions on Reconfigurable Technology and Systems*, 2(2):1–22, June 2009.

[125] Y.-L. Wu, S. Tsukiyama, and M. Marek-Sadowska. Graph Based Analysis of 2-D FPGA Routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(1):33–44, January 1996.

[126] T. Yamada, H. Kotani, J. Matsushima, and M. Inoue. A 4-Mbit DRAM with 16-bit Concurrent ECC. *IEEE Journal of Solid-State Circuits*, 23(1), February 1988.

[127] A. Yen, A. Tritchkov, J. P. Stirniman, G. Vandenberghe, R. Jonckheere, K. Ronse, and L. Van den hove. Characterization and correction of optical proximity effects in deep ultraviolet lithography using behavior modeling. *Journal of Vacuum Science Technology B: Microelectronics and Nanometer Structures*, 14(6):4175–4178, November 1996.

[128] W. Zhao and Y. Cao. New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration. *IEEE Transactions on Electron Devices*, 53(11):2816–2823, 2006.

[129] X. Zhou, K. Lim, and D. Lim. A simple and unambiguous definition of threshold voltage and its implications in deep-submicron MOS device modeling. *Electron Devices, IEEE Transactions on*, 46(4):807–809, April 1999.