Effects of and Mitigation Strategies for Fast Voltage Transients on Current and Future FPGAs

by

Linda L. Shen

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Computer Engineering
University of Toronto

© Copyright 2019 by Linda L. Shen
Abstract

Effects of and Mitigation Strategies for Fast Voltage Transients on Current and Future FPGAs

Linda L. Shen
Master of Applied Science
Graduate Department of Computer Engineering
University of Toronto
2019

As FPGAs grow in size and speed, so too does their power consumption. Power reduction techniques such as dynamic voltage scaling and clock gating can potentially be applied to FPGAs, however, it is unclear whether they are safe in the presence of fast voltage transients caused by large changes in on-chip signal activity. We measure the impact transients have on applications and present a mitigation strategy to prevent them from causing timing failures. We create transient generators that can significantly reduce an application’s measured maximum frequency, by up to 25%. We also show that transients are very fast and hence transient mitigation must occur within the same clock cycle as the transient. We create a clock edge suppressor that detects a transient event and delays the next clock edge to prevent timing failure. Additionally, we present new architectures that would make future FPGAs more robust to voltage transients.
Contents

List of Figures v

List of Tables vii

List of Acronyms viii

1 Introduction 1
  1.1 Motivation .................................................... 1
  1.2 Contributions ............................................... 2
  1.3 Thesis Organization ......................................... 3

2 Background 4
  2.1 Field Programmable Gate Arrays ............................. 4
    2.1.1 Current FPGA Power Consumption Trends ............... 4
    2.1.2 Voltage Scaling ......................................... 6
    2.1.3 Measuring Delay Variation in FPGAs ..................... 7
  2.2 On-Chip Voltage Variations .................................. 8
    2.2.1 PDN Design for Low Inductance ......................... 9
    2.2.2 Transients in CPUs ..................................... 11

3 Effects of Transients on Delay 14
  3.1 Impact of Transients on Fmax ................................. 14
    3.1.1 Transient Pattern ...................................... 16
  3.2 Comparing Transient Generators ................................ 17
  3.3 Transients and Scaling Supply Voltage ....................... 18
  3.4 Measuring Delay .............................................. 20
    3.4.1 Cyclone IV: Carry-Chain Approach .................... 20
    3.4.2 Stratix V: LUT Based Approach ......................... 21
  3.5 Transient Effects on Delay .................................. 23
    3.5.1 Reduction in Bits Propagated ........................... 23
    3.5.2 Increase in Delay and Reduction in Fmax ............. 26
    3.5.3 Size of Timing Margin .................................. 27
# List of Figures

2.1 Basic Logic Element of an FPGA. .................................................. 5
2.2 Basic structure of an FPGA and zooming in on an FPGA tile. ................. 5
2.3 Intel CPU and FPGA power consumption trend with technology node. .......... 6
2.4 Model of the on-chip PDN as an LRC network. .................................. 9
2.5 Elements that create decoupling capacitors. Taken from [40]. ................... 10
2.6 General technique to mitigate transients. ......................................... 11
2.7 A simple delay chain to measure voltage. ....................................... 11
2.8 Simulation of worst case voltage droop with and without the adaptive clock. Taken from [51]. ................................................................. 12
2.9 AMD’s clock stretcher. Taken from [52] ............................................ 13
3.1 Transient generators in pink, fill the remaining area of the FPGA. ............... 15
3.2 Transient generators that we evaluated. .......................................... 15
3.3 Frequency vs. Voltage of DFT benchmark. Dashed lines and the hollow bullet indicate the frequency measured without transients and the solid lines and bullet represents the frequency measured with transients. ........................................ 18
3.4 Frequency vs. Voltage of FIR filter benchmark. Dashed lines and the hollow bullet indicate the frequency measured without transients and the solid lines and bullet represents the frequency measured with transients. ........................................ 19
3.5 The delay measuring unit which uses a carry-chain to measure delay. ............ 20
3.6 Floor-plan of quadrants and how we number them. .............................. 21
3.7 The delay measuring unit which uses a chain of inverters to measure delay. The inverters are implemented using LUTs. ................................. 22
3.8 Stratix V delay chain path. ......................................................... 22
3.9 Number of bits propagated measured at different cycles after the transient has occurred on Cyclone IV at 200 MHz. ........................................ 24
3.10 Number of bits propagated measured at different cycles after the transient has occurred on Stratix V at 200 MHz. ........................................ 24
3.11 Reduction in bits propagated due to transients on Cyclone IV. .................. 25
3.12 Reduction in bits propagated due to transients on Stratix V. .................... 25
3.13 Delay push-out on Cyclone IV. ...................................................... 26
3.14 Delay push-out on Stratix V ....................................................... 27
3.15 Size of margin of Cyclone IV. ....................................................... 28
List of Tables

3.1 Devices used. ................................................................. 14
3.2 Fmax and logic utilization of Built in Self Test (BIST) and design under test (DUT) without transient generators reported by Quartus. ........................................ 15
3.3 Fmax measured of the DFT benchmark with different patterns of transients. Fmax without transients is 240 MHz. ................................................................. 16
3.4 Fmax drop on DFT benchmark due to transients. ........................................ 17
3.5 Fmax and margin over Quartus of DFT benchmark on the fixed Vdd board with and without transients being produced by transient generator C. ........................................ 17
3.6 Power consumption of DFT benchmark on variable Vdd board0 with voltage scaling at frequency prescribed by Quartus (204 MHz). ........................................ 19
3.7 Cyclone IV delay per bit from each quadrant at nominal voltage. .......................... 21
3.8 Stratix V delay per bit from each quadrant at nominal voltage. ............................ 22
4.1 Fmax of droop detector when monitoring different number of bits. ....................... 31
4.2 MAC unit benchmark using the clock suppressor. ................................................. 37
4.3 Clock insertion delay on and delay to determine if a droop has occurred Cyclone IV and Stratix V in ns. ................................................................. 39
4.4 Performance of hardened edge suppressor on Cyclone IV and Stratix V. ................. 39
4.5 Half the clock insertion delay and horizontal crossing delay on Cyclone IV and Stratix V in ns. ................................................................. 41
5.1 Power consumption of lutmasks implementing AND and OR of our baseline LUT, for a single input toggling in a single LUT at 250 MHz, measured in nW. ....................... 46
5.2 Different buffer locations ............................................................. 48
5.3 Different design for predecoding. ........................................................................ 49
5.4 Area, delay, and cost of FPGA tile for each LUT designs at 0.8 V normalized to \( P_{T_{\text{base}}} \). ................................................................. 51
5.5 Area, delay, and cost of the FPGA tile for each LUT designs at 0.6 V normalized to \( P_{T_{\text{base}}} \). ................................................................. 51
5.6 Voltage Tolerance as a percent of \( V_{\text{norm}} \) at \( \Delta d = 10\% \). ................................. 52
5.7 Voltage Tolerance with scaled supply voltage at \( \Delta d = 10\% \). ................................. 55
List of Acronyms

$V_{dd}$ core supply voltage
$v_{ind}(t)$ induced voltage

**ASIC** Application-Specific Integrated Circuit

**BIST** Built in Self Test

**BLE** Basic Logic Element

**CAD** computer-aided design

**CB** connection block

**CPU** Central Processing Unit

**DLL** delay locked loop

**DPLL** digital phase-locked loop

**DUT** design under test

**DVS** Dynamic Voltage Scaling

**FPGA** Field Programmable Gate Array

**HDL** hardware description language

**HLS** high level synthesis

**LB** logic block

**LUT** lookup-table

**PDN** power distribution network

**SB** switch block

**VDM** voltage-droop monitor
Chapter 1

Introduction

1.1 Motivation

Technology advancements have enabled modern Field Programmable Gate Arrays (FPGAs) to have larger logic capacity and faster clock frequencies, allowing FPGAs to perform more computations per chip compared to their predecessors, and opening up new application areas like data centre acceleration and machine learning [1]–[7]. With the end of Dennard scaling [8], these larger and faster FPGAs are handling much larger workloads without a significant nominal voltage reduction and hence are also consuming much more current and power. Since a lower operating voltage is one of the most effective means to reduce power, FPGAs are increasingly applying adaptive voltage techniques to lower voltage further, making them more sensitive to voltage disturbances that could lead to timing errors. In this work we seek to quantify the magnitude of these voltage disturbances, determine their risk to correct operation of the FPGA, and devise mitigation strategies to reduce this risk.

The workload and switching activity of an FPGA is unknown at manufacturing time and varies from application to application as well as over time within the same application. The changes in load that occur when an application is running can produce large supply current transients. During these large load transients, the change in the current drawn by the FPGA will cause a voltage transient, leading to fluctuations to the on-chip voltage and hence fluctuations to on-chip delay. Recent work showed that voltage fluctuations caused by fast load transients create the largest delay variations [9]. With chips getting bigger, we expect to see even larger transients, aggravating this problem. Intel even cautions about using clock gating in their Stratix 10 devices as they believe some power distribution networks may not be able to handle the resulting large changes in current safely [10].

Even though transient events occur on only a small minority of clock cycles, they can cause large fluctuations in delay and previous work have shown that they can be used by maliciously crafted designs to cause the FPGA to crash [9], [11]. Although a malicious application’s potential to crash remote FPGAs is concerning, a more imminent threat is that even non-malicious applications can be susceptible to voltage transients that they themselves produce. Those transients may not be large enough to crash the FPGA, but they could be large enough to cause timing failures. We believe this vulnerability is very common as most designs have different modes of operation, come out of reset at some point, and/or use clock gating: all these situations produce large changes in switching activity that could potentially produce large voltage transients.
Timing failures are more subtle compared to causing the chip to crash; however, they are just as devastating since the results of the FPGA in no longer correct. What makes this especially dangerous is that timing failures could occur without leaving any observable artifacts. That is, the user’s application would produce incorrect outputs; however, there is no indication that the outputs are incorrect.

Currently, FPGAs do not have any circuitry that could mitigate voltage transients and solely rely on passive solutions such as on-board, on-package, and on-die decoupling capacitance. These passive elements do lessen the effects of transients, but these elements can be large and FPGA architects must consider other trade-offs such as speed, area, and cost. What makes handling transients more difficult for an FPGA compared to other VLSI devices is that the workload and configuration are unknown at manufacturing time. FPGA architects and computer-aided design (CAD) tools could design for and model the absolute worst case, but this would lead to overly conservative voltage or timing guard-bands. To avoid designing FPGAs that are power hungry or extremely slow, it is clear that a new approach is needed.

As FPGA power consumption has grown, so too has interest in methods to reduce power. Some of the most promising research seeks to scale down supply voltage by taking advantage of the large guard-bands in the timing models that CAD tools. However, these guard-bands are in place to defend against many possible variations, including voltage variation. The drawback of scaling down supply voltage is that the voltage guard band is also reduced, making the FPGA more vulnerable to voltage fluctuations. This suggests that any effort to scale down the supply voltage needs to account for transients to guarantee safe operation making it likely that transients can impede voltage scaling. Since transients are the largest contributor to delay variation [9], they will require a significant timing or voltage guardband. That is, without transients or if we could mitigate the effects of transients, we could run FPGAs at a higher frequency or a lower voltage.

Power consumption is also a major concern for CPUs, and commercial CPUs have adopted a number of techniques to mitigate transients so that they do not need to run at a worst-case voltage. Even though FPGAs have different challenges compared to CPUs, mostly that the application is unknown, we believe it is likely we can leverage some of these techniques and adapt them for FPGAs.

It is apparent that research is needed to quantify how load transients on an FPGA can affect an application. Additionally, FPGAs would benefit from new approaches to mitigate the effects of transients.

1.2 Contributions

The research presented in this thesis explores the effects load transients have on FPGAs and then explores ways to mitigate these effects. Specifically, our contributions include:

- Building circuits to measure on-chip transients on FPGAs.
- Quantifying transients on wire bond and flip-chip FPGAs: magnitude, duration, and their risk to correct operation.
- Implementing a mitigation strategy for transients on current commercial FPGAs using existing soft logic. Additionally, we explore hardened solutions that could be applied to future FPGAs.
- Proposing architectural changes to the LUT circuitry to improve voltage robustness in future FPGAs. These changes lead to better tolerance to voltage transients in addition to allowing for more aggressive voltage scaling.
1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 presents background on FPGAs, load transients, and a few examples of mitigation strategies used in commercial CPUs. In Chapter 3, we build delay measuring circuits and transient generators, which we use to measure the effects transients can have on the delay. We present and implement transients mitigation solutions on current FPGAs using existing FPGA hardware in Chapter 4, showing that we can mitigate their effects. Also in Chapter 4, we present optimizations of our transient mitigation strategy that could be implemented in future FPGAs. Next, in Chapter 5, we explore architectural changes to the LUT that could make FPGAs more robust against transients. Finally in Chapter 6, we present a summary of our work.
Chapter 2

Background

2.1 Field Programmable Gate Arrays

FPGAs are devices that consist of arrays of reconfigurable logic elements connected by programmable routing; their programmability allows them to implement any hardware application. FPGAs have the advantage that they can be used to build applications that are massively parallel compared to Central Processing Units (CPUs) without the high cost and development time of custom Application-Specific Integrated Circuits (ASICs). Typically, logic on an FPGA is implemented using Basic Logic Elements (BLEs), which consist of a lookup-table (LUT) and a register as shown in Fig. 2.1. These BLEs are packed together to form logic block (LB), which are connected to the routing fabric through connection blocks (CBs) and switch blocks (SBs) then form the connections within the routing fabric. These blocks are arranged into a grid-like structure to form the FPGA shown in Fig. 2.2. The group of blocks that Fig. 2.2 zooms in on is referred to as an FPGA tile; replicating this tile can generate the logic and routing of an entire FPGA.

Compared to CPUs which can implement any software application by executing a different set of instructions, FPGAs implement different hardware applications by reprogramming their internal hardware. To program an FPGA, a designer uses a hardware description language (HDL) such as Verilog or VHDL, either hand-written or generated using high level synthesis (HLS), to describe the hardware that they want to implement. Then a CAD tool is used to compile the design into a bitstream that sets individual configuration bits on the FPGA to implement the circuit that the designer described. Once the design is compiled, a CAD tool computes the maximum frequency of the design using its timing models. These timing models must ensure correct functionality for every chip under all allowed operating conditions, such as temperature, and account for variations, such as process, voltage, and device aging [12]. This causes the CAD tool’s timing models to be pessimistic as the models must assume close to worst-case timing over all these variations [13].

2.1.1 Current FPGA Power Consumption Trends

For many years, FPGA architects leveraged Dennard scaling [8] to help reduce dynamic power consumption by reducing supply voltage with smaller process technologies [14]. Dennard scaling ended at approximately the 90 nm to 65 nm generation, and voltage scaling had greatly slowed down in recent process nodes. Reducing supply voltage is an effective power reduction technology as power consumption
Chapter 2. Background

Figure 2.1: Basic Logic Element of an FPGA.

Figure 2.2: Basic structure of an FPGA and zooming in on an FPGA tile.
scales quadratically with voltage. For example, Statix V, which is a 28 nm device, has a core voltage of 0.9 V while Stratix 10, a 14 nm device, has a core voltage of 0.85 V [15], [16]. Stratix 10 is also a much larger device compared to Stratix V (2.8 million vs. 622 thousand logic elements), and also has higher maximum frequency targets (1 GHz vs. 500 MHz core clock frequency) [15], [16]. Without significant operating voltage reductions, the increase in frequency and logic cell counts outweights the reduction in capacitance per element with process scaling, and thus, the current trend for FPGAs is towards higher power consumption.

In Figure 2.3, we show the power consumption of CPUs and FPGA across different technology nodes. We obtained the CPU power consumption data from the thermal design power (TDP) reported by Intel for desktop CPUs [17]. The FPGA power consumption was computed using Intel’s Early Power Estimator (EPE) assuming an application with 70% core utilization on high performance (Stratix series) FPGA. The clock frequencies used to generate Intel’s FPGA power numbers in Figure 2.3 are: 125 (Stratix), 180 (S II), 207 (S III), 230 (S IV), 250 (S V) and 430 (S 10) MHz; these frequencies come from typical achievable speeds in those generations. The Xilinx Power Estimator shows a similar trend for Xilinx devices to the one shown in the figure. Although exact power consumption depends of the application, the overall trend for FPGA is that we see a huge spike in power in recent years. The previous assumption that FPGAs consume much less power than CPUs no longer holds true when we look at 14-nm FPGA (Stratix 10).

2.1.2 Voltage Scaling

\[ P_{\text{dynamic}} \propto \alpha fV^2 \]  

Dynamic power is linearly proportional to clock frequency (f) and quadratically proportional to the
supply voltage (V) as shown in Eq. (2.1) where $\alpha$ is the activity factor. Although decreasing the activity factor and clock frequency both reduce power, they also reduce the throughput of the application. Reducing supply voltage has the advantage that it does not impact performance and reduces power significantly due to the quadratic relationship.

Even though the core supply voltage ($V_{dd}$) on current FPGAs is no longer decreasing significantly with each process node, it is still possible to lower power consumption by reducing $V_{dd}$. Recently, FPGA vendors are starting to provide the ability to operate the chips that are faster than nominal (due to process variations) at a range of voltages lower than the nominal voltage [18], [19]. For Intel’s Smart VID [18] devices, each chip is speed tested as part of the manufacturing process and a minimum voltage (in 50 mV increments) that meets the CAD tool timing model for each resource on the chip is determined. Xilinx has a similar scheme, where each chip is speed tested to either operate at a nominal or a lower voltage [19]. These techniques seek to reduce voltage per chip regardless of application.

Dynamic Voltage Scaling (DVS) is an application specific power reduction technique to dynamically adjust $V_{dd}$. Previous work by Ahmed et al. show one such scheme [20]. In this scheme, they replicate the placement and routing of a number of timing critical paths and modify the lutmasks so that a rising or falling edge at the input causes a rising or falling edge at the output. They measure the minimum supply voltage by sweeping $V_{dd}$ and observing whether a rise or fall at the input can propagate to the output. This value is measured across a range of temperatures and frequencies to create a calibration table. At run-time, $V_{dd}$ is scaled based on this calibrations table. Another approach to dynamically scale voltage is presented in [21], through the use of shadow registers. These shadow registers have the same inputs as the registers on the critical paths (these are the registers being monitored) but are clocked with a phase shifted clock. The results of the shadow register and the monitored register are XOR’d to indicate if a timing violation has occurred. This information is used to determine the minimum supply voltage at each frequency. These works have shown that DVS can reduce FPGA power consumption by 20-30% [20]–[23].

Unfortunately, by reducing the supply voltage the design becomes more vulnerable to voltage fluctuations, making it more likely that these fluctuations could cause timing failures. The work in [20] adds a 5% guard-band to the minimum voltage found by their DVS approach to accommodate voltage variations. Not only does this 5% guard-band reduce the power savings, it is also not clear whether the guard-band is conservative enough to account for all variations. Recent work has shown that fast load transients by far are the largest contributors to on-chip delay variations [9]. Intel even cautions about using clock gating in their Stratix 10 devices as, they believe some power distribution networks may not be able to handle the resulting large changes in current [10]. In this thesis, we investigate how load transients can affect the achievable power savings of DVS.

### 2.1.3 Measuring Delay Variation in FPGAs

FPGAs do not contain delay measuring circuits that are exposed to the user. However, delay and delay variations can still be measured with circuits that are built using the soft-fabric of the FPGA. A few examples include:

1. Measuring delay by computing critical path delay using a BIST system [24], [25]. This involves monitoring the outputs of a design with known or predictable outputs. Then Fmax can be measured by sweeping frequency and observing when the design starts producing incorrect outputs. The delay
through the critical path is the inverse of the Fmax.

2. Measuring delay variation by observing propagation through a soft-delay chain [26], [27]. This is similar to ASIC techniques that involve a chain of gates with close to identical delays, and in FPGA is often implemented with LUTs or a carry-chain, and registers capture the output after each segment. By sending a pulse through the delay-chain the propagation should reach the same gate an register given the same operating conditions (frequency, voltage, temperature). Then when there is a change in operating conditions, the signal propagation will end at a different gate. This method is well suited for measuring voltage variations, as voltage fluctuations will cause fluctuations in delay, and alter how far a pulse will propagate in the delay-chain.

3. Measuring delay variation by measuring the frequency of ring oscillators [28], [29]. A ring oscillator is a ring consisting of an odd number of inverters and its frequency can be computed by counting the number of oscillations in a given time period. This frequency is dependent on the path delay through the ring oscillator. Since this method is computing an average frequency over a time period, it is better suited for monitoring or measuring temperature than other methods, as is it less sensitive to noise or infrequent fluctuation. This averaging behaviour also makes it unsuitable for measuring rapid changes in delay.

Delay variation on FPGA can come from many factors including process variation (the variation in transistor geometry) [28], [30], temperature [29], supply voltage fluctuation [9], and aging [31]. Work by Gnadt et al. compared the various causes of delay variation on FPGA and showed that voltage fluctuations caused by fast load transients create the largest delay variations [9]. To measure delay variations, the authors used carry-chains to build a time-to-digital converter and used toggling registers which they control by gating the clock to create transients. They varied the duty cycle of their registers, which is the percentage of time that they are active, and showed that in general, producing transients at lower duty cycles produced larger voltage droops. Later work by the same authors showed that by creating transients at specific frequencies, the on-chip voltage can experience fluctuations large enough to cause the board to crash [11].

This thesis builds upon their findings by evaluating what effects power transients can have on real designs. The transients we evaluate are created much more infrequently – the transient generators we use have a duty cycle of less than 0.1%, whereas the lowest duty cycle evaluated by Gnadt et al. is 10%. This ensures that we are isolating for the effects of power transients and not mixing in other factors such as IR drop or power supply resonant frequencies. While Gnadt et al. were able to crash their board with a malicious design, our work instead focuses on how susceptible a non-malicious design is to load transients and the impact load transients have on the design’s critical path delay.

### 2.2 On-Chip Voltage Variations

Two main sources of voltage variation come from IR drops and current transients. IR drops arises from the power distribution network (PDN) resistance and reflects the steady state voltage drop [32]. The voltage variations produced by current transients are the result of inductance from the PDN [32].

Load current transients are non-periodic, and often sudden, changes in load, due to changes in the design’s switching activity. From Faraday’s law of induction and Lenz’s Law, we know that an inductive voltage will be created to oppose the change in current [33]. The induced voltage \( v_{ind}(t) \), described
by Eq. (2.2) is proportional to the change in current. \( L \) is the inductance; a proportionality constant which is dependent on material and geometry [33]. Fig. 2.4 shows a simplified model of the entire PDN as an LRC circuit. Inductance exists both on-chip, on-package, and from the board and all have on-chip voltage.

\[
\Delta v_{\text{ind}}(t) = -L \frac{di(t)}{dt} \tag{2.2}
\]

A large change in current leads to a large induced voltage. For a large increase in load current, \( v_{\text{ind}}(t) \) will be large and negative, and hence will produce a large decrease in \( V_{dd} \). This decrease in \( V_{dd} \) causes a increase in delay, which has the potential to cause timing failures.

To understand the magnitude of \( \frac{di(t)}{dt} \), we can perform some back of the envelope calculations using power estimates from Intel’s Early Power Estimator. On a Stratix V device (5SGXEA7N2F45C2), for a design with 70% logic utilization operating at 250 MHz, the power consumption with an activity factor of 5% is 1.8 W with and 17.5 W with an activity factor of 95%. This means that transitioning from an activity factor of 5% to 95% in a single clock cycle could see a \( \frac{di(t)}{dt} \) of 4.4 GA/s.

It is possible to design for lower inductance [32], [34]; for instance, flip-chip packaging typically has lower inductance compared to wire-bond packaging [35]. Flip-chip packaging allows power to be connected directly into the core of the chip, where as wire bond chips packaging can only connect at the die edges, shortening current return paths and reducing inductance. Packaging connections in flip-chip devices are made through solder bumps, which are shorter than the wire connections in wire-bond devices, which also reduces inductance. Altogether, these differences have been shown to reduce inductance by 70% in flip chip vs. wire bond packaging [35].

### 2.2.1 PDN Design for Low Inductance

The PDN can be designed to minimize the effects of fast load transients through the use of on-chip decoupling capacitors. The decoupling capacitors effectively act as a low pass filter on the current seen by the inductor in Eq. (2.2) and can be designed to filter out the effects of fast load transients. The use
of decoupling capacitors to reduce inductance is done at all stages of the PDN: on the board, packaging and FPGA. Additionally, a feedback path from the board near the FPGA to the power supply is used to monitor and compensate for changes in voltages such as those caused by load fluctuations or IR drops. Typically, the power supply can react and respond to voltage fluctuations around 50 kHz [36], and more capable power supplies can reach ranges around 200 kHz [37]. On the board and packaging level, the decoupling capacitors can maintain low impedance responses for frequencies on the order of tens of MHz [38], [39]. Since FPGA designs typically target frequencies above 100 MHz, the fastest transients can only be mitigated at the chip level.

On-chip decoupling capacitance can come from many different circuit elements. Although capacitance already exists on-chip, more can be added using the following components [40]:

- n-well: by reverse biasing the pn junction, a depletion region is created and a capacitor is formed between the n-well and n-substrate. (Fig. 2.5a)
- non-switching circuits: for a non-switching transistor, a capacitor is formed between $V_{dd}$ and ground. (Fig. 2.5b)
- thin oxide layer: the thin oxide layer between the $V_{dd}$ rail and the n-well creates a capacitor. (Fig. 2.5c)

The response of an LRC network is dependent on frequency, and PDNs should be designed to have low impedance and hence low voltage drop across a wide range of frequencies. However, fast load transients have a very wide bandwidth in the frequency domain, and it is difficult to produce a PDN that is both cost-effective and very low impedance across all frequencies. On-die capacitance for responding to the fastest frequencies is expensive in area – typically, 5-20% of the chip’s total area can be allotted to decoupling capacitors [41]–[44]. Yet as prior works have shown, this is still not enough to fully mitigate transients. Additionally, it is difficult to place these capacitors such that the same capacitance is seen on all parts of the chip [45], [46]. Given a finite cost and area constraint, it is improbable that all transients can be mitigated solely using decoupling capacitors.
2.2.2 Transients in CPUs

There are a number of solutions in the ASIC world that address transients [47]–[49] and which have been adopted in current CPUs. Intel, AMD, and IBM all use similar strategies that detect a voltage droop and reduce the clock frequency [50]–[52]. These solutions attempt to mitigate transients by operating at a decreased frequency until the voltage droop passes. They each have a structure similar to that shown in Fig. 2.6. A droop detector monitors for changes in voltage and when it determines that it has detected a significant droop, some control logic will modify how the clock is generated such that a slower clock is produced. Voltage is measured indirectly by monitoring delay. This is often implemented with a delay chain as shown in Fig. 2.7.

IBM’s Adaptive Clocking

IBM’s POWER9 chip is a 14 nm device which contains up to 24 cores [53] and operates with clock frequencies up to 4GHz [54]. This chip is divided into 6 structures which are called Quads, where each Quad has 5 power domains [51]. To handle transients, it reduces the clock frequency in the event of a voltage droop; processors may experience such voltage droops when they become active after being idle. The communication between Quads is asynchronous, and hence, they are able to perform frequency scaling per Quad. In their approach, an analog circuit monitors for voltage droops, which they call voltage-droop monitors (VDMs). There is one VDM for each power domain (5 per quad, 30 per chip). Filtering is performed over a 1 ns window and droops can be detected within 1.5 ns. If a droop is detected by any of the VDMs, the digital phase-locked loop (DPLL) controlling the clock for its Quad will reduce its frequency; thus compensating for the increase in delay. The total response time to detect
a voltage droop and for the digital phase-locked loop to lower the frequency is 6 ns. Once the droop event passes and $V_{dd}$ recovers, the DPLL returns back to normal.

Another strategy IBM uses is to set the core voltage to slightly above the nominal value when the processor is idle, which for their processor, does not significantly increase power consumption. This increases the voltage guard-band and allows for a longer time to react to voltage droops due to idle-to-active state transitions. Fig. 2.8 shows their modeling of the worst case voltage droop and how the adaptive clock is able to reduce its frequency to prevent timing errors (noise timing margin becoming negative). This in a technique that would be difficult to implement on FPGAs since the application is user defined. Therefore, any implementation would require the user to generate a signal to communicate with the off-chip power supply when it is idle. Not only is this not possible if the application does not have a well defined idle state, it is also susceptible to user error. If we operate the FPGA at a higher supply voltage when it is not truly idle, we would be consuming more power than necessary.

Overall, this mitigation technique allows the POWER9 to either reduce its power consumption by up to 8% or increase it’s performance by up to 3.5%.

**Intel’s Variable Frequency Clock System**

Intel’s Itanium processor (90 nm) uses a variable frequency clock system where the core clock and supply voltage are both modulated such that it operates within an power envelope. We will focus on the frequency modulation, which is the feature that allows it to react to transients. The processor is divided into 14 zones, where each zone has its own clock and each clock is controlled by a digital frequency divider. The input to the digital frequency divider is a reference clock and a value (D) that ranges from 0 to 64. Based on D, the frequency of the output clock ranges between 1 to 0.504 times that of the reference clock. For each clock, there are a number of a voltage detectors (each which is implemented with a delay chain) placed around each zone. The output of these voltage detectors directly impacts D, enabling the output frequency to track core voltage. During a transient event that results in a voltage droop, the frequency is decreased within 1.5 cycles. Overall, this technique allows the Itanium processor to operate an average of 6% faster across various benchmarks.
Figure 2.9: AMD’s clock stretcher. Taken from [52]

**AMD’s Clock Stretcher**

AMD owns a patent introducing a voltage droop mitigation strategy using what they call a clock stretcher [52] (shown in Fig. 2.9) which is currently used in both their CPUs and GPUs [55]. Unlike the previous two techniques which reconfigures a DPLL or a PLL, this techniques muxes between different clock signals. These clock signals are generated by the same delay locked loop (DLL); they have the same frequency but different phase shifts and an N-ary mux selects which clock signal will be used as the core clock. By muxing to a clock that is slightly delayed during a voltage droop event, they are effectively "stretching" the clock for one cycle. Once the droop event passes, no action is needed since all the clock signals are all at the same frequency. This in theory should be faster at reacting to voltage droops since muxing to a different clock signal should be faster than reconfiguring the clock PLL. Additionally, once the droop event passes, it is able to continue operation at the nominal frequency, whereas the previous techniques require the clock to be reconfigured a second time. Therefore, with this method, the clock spends less time operating at a lower frequency. Although no performance result is presented in the patent, AMD does claim that their clock stretcher allows them to reduce power by up to 18% and increase performance by 5% [55].
Chapter 3

Effects of Transients on Delay

In this chapter, we quantify the delay impact of load transients that typical (non-malicious) designs may experience. We explore different transient generating circuits and measure the impact they can have on application frequency. We also create delay measuring circuits that can be configured onto an FPGA to quantify the amount of delay increase that transients can produce. We use two families of FPGAs, Cyclone IV and Stratix V. Exact part numbers and board names are listed in Table 3.1.

A portion of this chapter was published in [58].

3.1 Impact of Transients on Fmax

For our first experiment, we quantify the effects of transients on the measured Fmax of an application. We use a BIST system to measure a design’s Fmax [20] as shown in Fig. 3.1. The BIST is composed of a linear feedback shift register (LFSR) to provide the DUT with input vectors, and a multiple-input signal register (MISR) that hashes the outputs of the design to produce a 256-bit signature. We test the design by running the DUT for $2^{32}$ clock cycles. We use this system to test two designs, an FIR filter and a DFT. Each time the test is run, it should produce the same output. To measure the benchmark’s Fmax, we first run the test at a low frequency at which we know the benchmark can safely operate and store the signature as a reference. Then, we run the test at different frequencies and compare the signature to the reference. We employ a binary search scheme to find the highest frequency at which the signatures are the same.

We design three different circuits to generate transients as shown in Fig. 3.2 then use them to measure their impact on Fmax. The run signal controls when transients are generated. We seek to determine which circuit produces the largest Fmax degradation. Note that a design using DSP and RAM blocks could experience even larger transients, as we are using only the soft fabric as transient generators.

We perform our experiment using three different Cyclone IV chips. We have one board that is unmodified from the manufacturer; this board uses a closed-loop 1.2V power supply, which we call the

<table>
<thead>
<tr>
<th>Table 3.1: Devices used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
</tr>
<tr>
<td>Cyclone IV</td>
</tr>
<tr>
<td>Stratix V</td>
</tr>
</tbody>
</table>
Figure 3.1: Transient generators in pink, fill the remaining area of the FPGA.

Figure 3.2: Transient generators that we evaluated.

Table 3.2: Fmax and logic utilization of BIST and DUT without transient generators reported by Quartus.

<table>
<thead>
<tr>
<th></th>
<th>Fmax (MHz)</th>
<th>Logic Utilization(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFT</td>
<td>204</td>
<td>15</td>
</tr>
<tr>
<td>FIR</td>
<td>86</td>
<td>59</td>
</tr>
</tbody>
</table>
fixed Vdd board. We also have two other boards that are modified so that we can control the core voltage using a bench voltage supply; these we call the variable Vdd board0 and variable Vdd board1.

To measure the impact of these transient generators, we lock down the placement and routing of our BIST and DUT then fill the rest of the chip with the transient generators. No placement constraints are set for the transient generators. The Quartus-reported Fmax and logic utilization of each benchmark with the BIST but without the transient generators are reported in Table 3.2. With the transients generators added, the logic utilization is 99%. An important difference is that the FIR filter design is slower and uses more area compared to the DFT which means that fewer transient generators can be placed in the FIR filter design.

### 3.1.1 Transient Pattern

Before we compare the different transient generators, we need to determine how long transient events should last and at what frequency they should occur to produce the greatest impact on Fmax. We varied different patterns that we can use to create transients.

We define 2 terms:

- **cycles_toggling**: the number of continuous cycles the transient generators are toggling. We achieve this by holding the run signal high for that many cycles.

- **duty_cycle**: The fraction of time transients are being generated

For example, if we have duty_cycle of 1/32 and cycles_toggling of 4, then we repeat holding the run signal high for 4 cycles and low for 124 cycles. cycles_toggling=1 is a special case where only one transition happens in a given transient event. For example, if we set cycles_toggling=1 and duty_cycle=1/64, all the registers in the transient generator would flip high and stay high for 64 cycles, and then flip low and stay low for another 64 cycles.

Using the fixed Vdd board and the DFT benchmark, we experiment with creating transients at different frequencies and durations for all three transient generators. Table 3.3 shows the Fmax we measured using Transient Generator C. All the values we tested are able to produce a significant Fmax drop compared to the 240 MHz that we measure without transients. With the same transient generators at different cycles_toggling duration and duty_cycle, there is little variation between the different transient patterns, but all produced a significant reduction in Fmax. The worst transient, by a small margin is the shortest one, with cycles_toggling=1.

<table>
<thead>
<tr>
<th>cycles toggling</th>
<th>duty cycle</th>
<th>1/8</th>
<th>1/16</th>
<th>1/32</th>
<th>1/64</th>
<th>1/128</th>
<th>1/256</th>
<th>1/512</th>
<th>1/1024</th>
<th>1/2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>211</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
</tr>
<tr>
<td>2</td>
<td>210</td>
<td>210</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>4</td>
<td>212</td>
<td>210</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>210</td>
<td>212</td>
</tr>
<tr>
<td>8</td>
<td>208</td>
<td>214</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>216</td>
<td>212</td>
</tr>
<tr>
<td>16</td>
<td>210</td>
<td>210</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>32</td>
<td>212</td>
<td>210</td>
<td>212</td>
<td>210</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
</tr>
</tbody>
</table>

Table 3.3: Fmax measured of the DFT benchmark with different patterns of transients. Fmax without transients is 240 MHz.
Table 3.4: Fmax drop on DFT benchmark due to transients.

<table>
<thead>
<tr>
<th>Generator</th>
<th>Variable Vdd 0</th>
<th>Variable Vdd 1</th>
<th>Fixed Vdd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>13.5%</td>
<td>12.9%</td>
<td>9.2%</td>
</tr>
<tr>
<td>B</td>
<td>16.1%</td>
<td>15.7%</td>
<td>14.3%</td>
</tr>
<tr>
<td>C</td>
<td>25.4%</td>
<td>25.2%</td>
<td>23.7%</td>
</tr>
</tbody>
</table>

Table 3.5: Fmax and margin over Quartus of DFT benchmark on the fixed Vdd board with and without transients being produced by transient generator C.

<table>
<thead>
<tr>
<th></th>
<th>Fmax (MHz)</th>
<th>Margin Over Quartus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quartus reported</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>Measured with no transient</td>
<td>278</td>
<td>36%</td>
</tr>
<tr>
<td>Measured with transients</td>
<td>212</td>
<td>4%</td>
</tr>
</tbody>
</table>

We choose to produce transients at a high enough frequency so that our tests do not take too long, but low enough to ensure that we are isolating the effects of transients and not measuring other factors. We find that by creating transients every 5000 cycles, which is a duty cycle of 0.02%, we cannot measure an increase in power consumption, which indicates that IR drop should also be unaffected. For the rest of the measurements in this chapter, we use \( \text{duty\_cycle}=1/5000 \) and \( \text{cycles\_toggling}=2 \). Although it is possible that we are not creating transients when the critical path is being sensitized, we believe this is unlikely. We run each DUT for \( 2^{32} \) cycles in each iteration we are measuring Fmax, by producing transients every 5000 cycles, we are testing a total of \( 8.59 \times 10^5 \) transients events. Additionally, Table 3.3 shows that generating transients with different values of \( \text{cycles\_toggling} \) produces similar Fmax values, suggesting that we are indeed producing transients when the critical path is being sensitized. In later sections, we measure the effects of transients where we are only testing the critical path and therefore guarantee its sensitization.

### 3.2 Comparing Transient Generators

In Table 3.4, we report the drop in Fmax of the DFT benchmark due to transients on both the variable Vdd and fixed Vdd boards. The Fmax drop is the percent difference between the Fmax measured with transients relative to that without transients. All three boards show the same trends, with the variable Vdd boards receiving a slightly larger hit in Fmax. Transient generator C, which produces the largest effect, was able to reduce Fmax by 23.7% on the fixed Vdd board and 25.4%/25.2% on the two variable Vdd boards.

For the rest of this thesis we solely use transient generator C. Table 3.5 shows the margin over Quartus of the DFT design on the fixed Vdd board, which is the percent difference between Quartus’ reported Fmax vs. the measured Fmax. The design has a significant (36%) timing margin on this board when there are no transients, but this margin is greatly reduced to 4% by transients. Although the measured Fmax is still above Quartus’ reported Fmax, this suggests that Quartus’ timing model requires a large guardband for transients. Likewise, it highlights that transients must be considered for safe DVS or over-clocking.
CHAPTER 3. EFFECTS OF TRANSIENTS ON DELAY

3.3 Transients and Scaling Supply Voltage

Using the variable Vdd boards, we measure the Fmax of both benchmarks at different voltages. Intel specifies that the voltage supply is allowed to range ±50mV of the nominal 1.2V, which is why we specifically measure at 1.15V. The results of the sweep are in Fig. 3.3 and Fig. 3.4 for the DFT and FIR benchmarks, respectively. As Fig. 3.3 shows, for the DFT benchmark at nominal voltage with our transient circuits, the Fmax we measure is only marginally above Quartus’ reported Fmax. However, The Fmax we measure at 1.15V on the same benchmark is slightly lower than Quartus’ reported Fmax. This indicates that a worst-case, but still within-spec, power supply of 1.15V will result in timing failures for this design in the presence of large load transients. On the FIR benchmark (Fig. 3.4), we also see a drop in Fmax due to transients; however, there is still a large margin above Quartus’ reported Fmax. We believe the smaller impact is due both to the fact that the FIR benchmark is larger, so we could not add as many transient generator circuits, and also that it is a slower design. Most importantly, these two charts show that for both circuits tested, the 5% guard-band used by Ahmed et al. [20] in their DVS solution is insufficient in the presence on large voltage transients.

Looking at the DFT design in Fig. 3.3, if we want to use DVS to reduce the $V_{dd}$ while operating the design safely at Quartus’ reported Fmax, with the presence of transients our margins are too slim for significant power savings. In Table 3.6 we measure the power consumption of the DFT design on variable Vdd board0 when we lower the $V_{dd}$ to the minimum allowed with and without transients present. We run the benchmark at the maximum frequency allowed by Quartus which is 204 MHz. If there are no transients present, we are able to reduce power consumption by 31.8%, however, with transients, the power saved dwindles to just 7%. This shows that despite being infrequent events, transients can greatly
Figure 3.4: Frequency vs. Voltage of FIR filter benchmark. Dashed lines and the hollow bullet indicate the frequency measured without transients and the solid lines and bullet represents the frequency measured with transients.

Table 3.6: Power consumption of DFT benchmark on variable Vdd board0 with voltage scaling at frequency prescribed by Quartus (204 MHz).

<table>
<thead>
<tr>
<th></th>
<th>Supply</th>
<th>Power Consumption</th>
<th>Power Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>nominal</td>
<td>1.2V</td>
<td>2.6W</td>
<td>0%</td>
</tr>
<tr>
<td>min voltage</td>
<td>1.16V</td>
<td>2.4W</td>
<td>7%</td>
</tr>
<tr>
<td>with transients</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min voltage</td>
<td>1.00V</td>
<td>1.8W</td>
<td>31.8%</td>
</tr>
<tr>
<td>without transients</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
reduce the power savings potential of DVS, highlighting the importance of mitigating transients.

3.4 Measuring Delay

To better understand how transients can impact a design, we set out to measure the delay impact due to transients. We indirectly measure on-chip voltage by measuring the delay of a specific path, and then any changes to on-chip voltage can be observed as a change in delay. This is also important if we want to mitigate transients since we need to be able to detect when a transient is occurring.

We perform these measurements on both Cyclone IV and Stratix V, so we need delay measurements circuits for both FPGA families. Performing measurements on both devices allow us to first confirm that transients are a problem for FPGAs in general and not just one family of FPGAs. The flip-chip packaging of Stratix V will reduce PDN inductance, which will reduce the voltage drop used to load transients. Stratix V is a flip-chip device whereas Cyclone IV is wire-bond, which suggests smaller transients due to reduced inductance. However, it is also a larger chip, with higher frequency targets, which will lead to larger current transients, which tends to increase the voltage drop due to transients. Measuring both would let us see how these factors influence the delay effect of transients.

We also set out to measure on-chip variation; we want to find out if transients produced in one part of the chip can affect delay in another. We divide the chip into four quadrants, with each quadrant contains a delay measuring unit where the delay chain of the delay measuring unit is placed at the center of the quadrant. The floor-plan of our setup is shown in Fig. 3.6 for Cyclone IV. As shown, we have 4 placement regions, one for each quadrant. The quadrants are tall enough so that we can constrain the delay chain in the delay measuring unit to a horizontal placement region that is one LB wide. We filled the chips with unconstrained transient generators so that 90% of the chip’s logic was used.

3.4.1 Cyclone IV: Carry-Chain Approach

On Cyclone IV, we can measure delay by implementing an analog to digital converter by using an adder and observing how far the carry bit propagates in one clock cycle; we call this structure a delay chain. The adder is implemented with logic blocks and uses a hardened carry-chain. By varying the input clock frequency and recording the number of bits through which the carry propagates at each frequency, we can calculate the delay to propagate to the next bit, which we call the delay per bit. We choose to use an adder because it gave us finer granularity compared to using LUTs since the delay across one bit in the carry-chain is smaller than across a LUT. The Cyclone IV chip’s adder is a ripple-carry adder, simplifying our implementation. The delay measuring unit shown in Fig. 3.5 shows our setup to measure bits propagated. The registers that capture the adder’s output are placed in the same LE as the adder.
Chapter 3. Effects of Transients on Delay

Figure 3.6: Floor-plan of quadrants and how we number them.

Table 3.7: Cyclone IV delay per bit from each quadrant at nominal voltage.

<table>
<thead>
<tr>
<th>Quadrant</th>
<th>Quartus Offset (ps)</th>
<th>Quartus Delay per Bit (ps)</th>
<th>Measured Offset (ps)</th>
<th>Measured Delay per Bit (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadrants 0</td>
<td>2019</td>
<td>66.0</td>
<td>1380</td>
<td>44.2</td>
</tr>
<tr>
<td>quadrants 1</td>
<td>2019</td>
<td>66.0</td>
<td>1430</td>
<td>43.4</td>
</tr>
<tr>
<td>quadrants 2</td>
<td>2021</td>
<td>66.0</td>
<td>1310</td>
<td>44.0</td>
</tr>
<tr>
<td>quadrants 3</td>
<td>2021</td>
<td>66.0</td>
<td>1420</td>
<td>43.8</td>
</tr>
</tbody>
</table>

The bitcounter block counts the number of bits that are high starting from the least significant bit and stopping at the first bit that is not high; the result is the number of bits propagated. We also have a vertical placement region at the center of each quadrant so that the carry-chain and the registers capturing the carry-chain’s output are forced to be in the same LE. This guarantees the routing delay from each adder sum output to the register capturing it is small and consistent.

The first set of measurements we take is to compute the delay per bit. We do so by sweeping the operating frequency and recording the number of bits propagated. All the transient generators are disabled for these measurements. We expect to see a linear relationship between bits propagated and clock period because each full adder should be identical. We do indeed see this linear relationship. The measured delays per bit and the offset, which is the delay to propagate 0 bits, are shown in Table 3.7. The measured delay to propagate one bit is approximately 44 ps. There is some (albeit small) variation between the different quadrants and therefore any solution to detect transients should calibrate the delay detection circuits. In Table 3.7, we see that there is a large margin between what Quartus reports and what we measure.

3.4.2 Stratix V: LUT Based Approach

Stratix V uses a carry-skip adder, which means that the delay to propagate to the $n$-th bit in the carry chain does not monotonically increase with $n$. In this architecture, we therefore use a LUT-based delay measuring circuit.

Ideally, we want a chain of LUTs, each implementing a buffer, where we can measure how far an
Figure 3.7: The delay measuring unit which uses a chain of inverters to measure delay. The inverters are implemented using LUTs.

Figure 3.8: Stratix V delay chain path.

Table 3.8: Stratix V delay per bit from each quadrant at nominal voltage.

<table>
<thead>
<tr>
<th>Quadrant</th>
<th>Quartus Offset (ps)</th>
<th>Delay per Bit (ns)</th>
<th>Measured Offset (ps)</th>
<th>Delay per Bit (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadrant 0</td>
<td>356</td>
<td>178</td>
<td>357</td>
<td>152</td>
</tr>
<tr>
<td>quadrant 1</td>
<td>480</td>
<td>173</td>
<td>346</td>
<td>150</td>
</tr>
<tr>
<td>quadrant 2</td>
<td>494</td>
<td>172</td>
<td>321</td>
<td>149</td>
</tr>
<tr>
<td>quadrant 3</td>
<td>439</td>
<td>172</td>
<td>325</td>
<td>147</td>
</tr>
</tbody>
</table>
input propagates in a fixed time. The challenge is getting the Quartus synthesis engine to create this design, since anything we describe using behavioural Verilog is obviously redundant and will of course be optimized away. We need to describe this structurally using WYSIWYGs [59]. Additionally, Quartus does not keep WYSIWYGs that simply implement a buffer, so instead, we implement a chain of NOT gates. Once we capture the outputs of the chain, we invert every other bit so that we can observe a single value propagating rather than alternating bits to ensure small and consistent routing delays. The LUT to LUT delays are mostly local within-LAB interconnect, with every 10th bit being a fast direct link interconnect between adjacent LABs. The implementation is shown in Fig. 3.7. We constrained the placement of both the LUT and registers capturing their outputs. The path to propagate to the last bit is shown in Fig. 3.8.

The delays per bit and the offset, which is the delay to propagate 0 bits are shown in Table 3.8. Again, there is a small variation between quadrants. The delay to propagate through one bit is approximately 150 ps which is larger compared to Cyclone IV. The means that the granularity of the measurements from our delay measuring unit is smaller on Cyclone IV.

3.5 Transient Effects on Delay

We compare the number of bits through which a transition propagates (bits propagated) on the delay chain during a transient event versus when there are no transients. We can use this comparison to determine how much transients impact delay as the delay is linearly proportional to bits propagated. Additionally, by turning on the transient generators in different quadrants of the chip, we can see how transients in one part of the chip will affect another part. We define 3 types of transients for a delay chain in quadrant $n$.

- **t-self**: only transient generators in quadrant $n$ are enabled
- **t-others**: transient generators from all quadrants except quadrant $n$ are enabled
- **t-all**: all transient generators are enabled

3.5.1 Reduction in Bits Propagated

In Fig. 3.9 and Fig. 3.10, we record the number of bits propagated at various cycles after the transient occurs, that is, the cycle the run signal goes high. We report the average from all 4 quadrants. The transients are created in cycle 0 with a clock frequency of 200 MHz. In these experiments, only a single transition (registers rising from 0 to 1) occurs at cycle 0 and no transitions occur until the cycle after we sample the delay chain. We see that the transient’s effects can linger for 3 cycles on Cyclone IV and at least 7 cycles on Stratix V. The worst impact occurs during the cycle in which the transient occurs. Fig. 3.9 and Fig. 3.10 also shows that, as one would expect, the number of transient generators impacts the size of the transient: t-all produces the largest transients. Transients close to the victim circuit have somewhat more impact, as the transient generators in the detector’s quadrant (t-self) have more than 1/3 the delay impact of the transient generators in the other 3 quadrants combined (t-other).
Figure 3.9: Number of bits propagated measured at different cycles after the transient has occurred on Cyclone IV at 200 MHz.

Figure 3.10: Number of bits propagated measured at different cycles after the transient has occurred on Stratix V at 200 MHz.
Chapter 3. Effects of Transients on Delay

Figure 3.11: Reduction in bits propagated due to transients on Cyclone IV.

Figure 3.12: Reduction in bits propagated due to transients on Stratix V.
Fig. 3.11 and Fig. 3.12 show the number of bits reduced at different clock periods for Cyclone IV and Stratix V receptively. The number of bits reduced is the difference between the number of bits propagated with and without transients. The number of bits reduced varies slightly depending on the clock period; this suggests that transients are fast, regardless of the clock frequency, and that most of the degradation happens early in the clock cycle. This also means that transients are more impactful for faster designs than slower designs, which aligns with the larger transient impact we saw on the DFT vs. the FIR benchmark in Section 3.3.

Fig. 3.11 and Fig. 3.12 show opposite trends, although it important to note that these trends are small and that as a whole, the number of bits reduced is closer to being constant with respect to the clock period. Cyclone IV sees larger absolute increases in delay with larger clock periods and Stratix V sees larger increases at lower clock periods. This is likely due to second-order effects and possible on-chip power \( V_{dd} \) ringing in Stratix V. However, more research would be needed to determine the exact cause. The overall trend for both these devices is that the number of bits reduced shows little variance across different clock frequencies.

### 3.5.2 Increase in Delay and Reduction in Fmax

We want to understand the implication of transients in terms of the delay push-out that they can cause.

To compute this the increase in delay, we define the following:

- \( B_n(T) \): furthest bits propagated to without transients with respect to clock period \( T \)
- \( B_t(T) \): furthest bits propagated to with transients with respect to clock period
Chapter 3. Effects of Transients on Delay

(3.1)

\[ \Delta d(T) = \frac{B_i^{-1}(B_n(T)) - T}{T} \times 100\% \]

Eq. (3.1) states that for a path that has a delay of T without transients, \( \Delta d(T) \) describes how much slower this path is with the presence of transients. To compute the decrease in frequency, we use the relationship that frequency is the inverse of the clock period.

Looking at Fig. 3.13 and Fig. 3.14, we see that the overall trends on Cyclone IV and Stratix V are very similar. That is, faster clocks see a bigger impact from transients and that enabling more transient generators also produces bigger impact. We see that the delay push out on Cyclone IV and Stratix V from t-all and t-others are also very similar. However, t-self is much more aggressive on Cyclone IV than Stratix V. That is, transients generators in one quadrant is less likely to draw power from another quadrant on Stratix V. This is likely due to less robust PDN on Cyclone IV: not having a metal redistribution layer and having its I/O on the periphery.

3.5.3 Size of Timing Margin

In Fig. 3.15a and Fig. 3.16a we plot the furthest bit we can propagate to with and without transients, as well as what Quartus models. Using this data, we can extrapolate the margin that we have over
Chapter 3. Effects of Transients on Delay

(a) Comparing bits propagated vs. clock period on Cyclone IV.

(b) Margin over Quartus of Quadrant 0 on Cyclone IV.

Figure 3.15: Size of margin of Cyclone IV.

(a) Comparing bits propagated vs. clock period on Stratix V.

(b) Margin over Quartus of Quadrant 0 on Stratix V.

Figure 3.16: Size of margin of Stratix V.
Chapter 3. Effects of Transients on Delay

Quartus.
Computing margin is similar to how we compute $\Delta d$. We define the following:

- $B_q(f)$: furthest bits propagated from Quartus’s model with respect to frequency ($f$)
- $B_m(f)$: furthest bits propagated measured with respect to frequency
- $m(f)$: margin over Quartus with respect to frequency

Then we can compute $m(f)$ using Eq. (3.2)

$$m(f) = \frac{f - B_m^{-1}(B_q(f))}{f} \times 100\%$$ (3.2)

That is, for a design that Quartus says should run at a frequency of $f_q$, the real Fmax of the design is actually faster or slower by $m$ percent. A positive value means the design can operate at a faster frequency and a negative value means the design needs to operate at a slower frequency. We plot this margin in Fig. 3.15b and Fig. 3.16b.

Looking at Cyclone IV, we see that without transients, there is a very large margin. However with transients, although there is still a large margin at lower frequencies, this margin is diminished as we reduce the clock period. Although there is still a margin over Quartus with transients, suggesting that on their own, transients will not cause timing failures, but any attempts to save power by exploiting this margin using DVS will be limited.

Conversely, Stratix V’s timing margin without transients is much smaller than that of Cyclone IV. With transients, we see that this margin has become negative, indicating that transients are large enough to cause timing failures.

Comparing Cyclone IV and Stratix V, the margins without transients are much larger on Cyclone IV. An interesting observation is that if we refer back to Fig. 3.13 and Fig. 3.14, even though $t$-others produce a similar delay push-out on Cyclone IV, the margins are large enough to cover them. However, on Stratix V, this is not the case.

In this chapter, we’ve shown that load transients can have a large impact to on-chip delay. Clearly, load transients must be considered in creating timing models and in deciding on safe levels during DVS, or timing failures will result.
Chapter 4

Mitigating Transients

In Chapter 3, we saw that fast load transients have a fast and large impact on delay. Current methods to deal with them such as better PDN design or guard-banding are insufficient to handle large transients without major impact to the speed of the FPGA timing models, or the cost and practicality of the PDN design.

In this chapter, we propose two techniques to mitigate transients, and implement one on Cyclone IV and show that we can mitigate the effects of load transients for moderate frequency clock domain. We believe a similar strategy can be applied to Stratix V and other families. Then, we harden key circuitry for this mitigation approach and show how integrating this additional circuitry in future FPGAs would allow transient mitigation at higher frequencies.

A portion of this chapter was published in [58].

4.1 Mitigating Transients In Current FPGAs

4.1.1 Detecting Voltage Droops

To detect a voltage droop, we look for an increase in delay. Our droop detector, shown in Fig. 4.1, continuously measures the number of bits propagated and raises the droop signal if the number of bits propagated drops below a certain value. We need to calibrate how many bits propagated we consider to be the boundary between the non-droop and droop cases on a specific chip. This boundary would be different depending on the location of the droop detector and the current clock frequency. The

![Droop Detector Diagram](image-url)
Table 4.1: Fmax of droop detector when monitoring different number of bits.

<table>
<thead>
<tr>
<th>bits monitored</th>
<th>droop detector frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>228.26</td>
</tr>
<tr>
<td>32</td>
<td>246.06</td>
</tr>
<tr>
<td>16</td>
<td>277.62</td>
</tr>
<tr>
<td>8</td>
<td>284.33</td>
</tr>
<tr>
<td>4</td>
<td>334.56</td>
</tr>
<tr>
<td>2</td>
<td>338.98</td>
</tr>
<tr>
<td>1</td>
<td>342.00</td>
</tr>
</tbody>
</table>

\[
1 \ 1 \ 1 \ 1 \ 1 \ 1 \ldots \ 1 \ 1 \ 1 \ 1 \ 1 \\
+ \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \ 0 \ 0 \ 0 \ 0 \ m \\
= !m!m!m!m!m\ldots !m!m!m!m!m
\]

Figure 4.2: Addition performed by delay measuring unit

calibration stage takes 32 samples from the delay chain; all these samples are bit-wise AND’ed together to produce the worst case, and this is stored. By taking multiple samples, we account for clock jitter. We then shift this value by a user-specified number of bits (margin) to reduce the number of bits that must propagate through the delay chain to be considered the "non-droop" case. A margin makes the droop detector more permissive so that it will not flag small voltage droops. We store this value and call it the golden pattern.

The droop detector requires two cycles to determine if a droop has occurred. In the first cycle, it captures the output from the delay chain, and then in the next cycle, it compares the captured output to the golden pattern. If a bit is high in the golden pattern but not high in the delay chain’s output, then we determine a droop has occurred. The delay chain requires one cycle to reset after each measurement. To be able to detect droops at every cycle, we use two delay chains and offset when each chain is being measured, using an independent golden pattern for each delay chain to account for on-chip variation. The droop detector raises the droop signal if a droop on either delay chain is detected.

Although we only need to monitor one index on the delay chain – which is the index of the most significant high bit in the golden pattern – this index is different for every chip and at every frequency and detector location. However, we want to use one design across many chips. If we try to cover a very large range, then we would have too many bits to compare, resulting in a slow design. Table 4.1 shows the fmax of our droop detector when monitoring different number of bits. The balance we chose was to monitor 32 bits and we will reduce the number of bits we monitor in the next section. We tested this droop detector with all transient generators of Table 3.4 and it was able to reliably detect all the transients (t-self, t-others, and t-all).

Reducing Number of Bits to Compare

The fastest droop detector is one that only monitors 1 bit. To achieve this, we created a more optimized delay measuring unit which we can calibrate so that for every frequency, without transients, the propagation will always reach the same bit. The delay measuring unit in Fig. 3.5 uses an adder to perform the addition in Fig. 4.2 where \( m \) is the measure signal. For a delay chain of length \( n \), instead of a measure
signal that is 1 bit, which alternates between 0 and 1, we have an \( n \) bit signal \( X \), where \( X[i] \) is the \( i \)th bit of \( X \). \( X \) alternates between \( A \) and \( B \), where \( A \) is all 1’s and \( B \) is all 0’s except 1 bit, at index \( b \), where \( B[b] \) is 1. Previously, the propagation started from the first bit in the delay chain; by alternating between \( A \) and \( B \), index \( b \) toggles between 0 and one and therefore, the propagation starts from index \( b \). The new addition performed is shown in Fig. 4.4. We call this new circuit the variable length chain shown in Fig. 4.3. Measure toggles so the input to the adder toggles between \( A \) and \( B \) and instead of storing the entire output of the adder, we only store one bit. We use the following steps to calibrate our delay measuring unit without the presence of transients. The examples values \( A \) and \( B \) are for \( n = 8 \).

1. Set \( A \) to 00000000 and \( B \) to 0x00000001. With this configuration, the propagation starts from the first index.

2. Takes 32 samples from the delay chain then AND these samples together. An example of something we sample in the first iteration is 0x00001111, and in the second iteration, we could see 0x00011111.

3. If the propagation has reached the \( n - 1 \)-th bit, then we are done. Otherwise, left shift \( B \) and return to step 2. We should never reach the \( n - 1 \)-th bit in the first iteration, as this would indicate that our delay chain is too short, and instead, we should increase \( n \).

4. Calibration is complete.
Once the calibration is complete, then in the no transient case, the propagation should always reach the $n - 1$-th bit; hence we only need to monitor one bit for transients. If we do not allow for any margins, then we would monitor the $n - 1$-th bit. If we want an margin of size $m$, we would monitor the $n - 1 - m$-th bit. Then to detect a transient, if the propagation does not reach the bit that we are monitoring, that is, does not transition from 0 to 1 in the cycle that we are measuring, then we have detected a droop. The new droop detector using the variable length chain is shown in Fig. 4.5. Instead of comparing to a golden pattern, we compare if the value was stored during the sampling cycle (measure is selecting $B$). Using this method, we were again able to detect all transients that we produce.

4.1.2 Mitigating Transients

We have shown that transients are fast and that their impact is seen in the same cycle that they were created. The implication is that to handle transients, we must also be able to react within the same cycle. To ensure that transients do not cause timing failures, we propose delaying the rising edge of the clock, giving extra timing margin during the cycle the transient has attacked. To achieve this, we present two methods: the first uses the clock enable on the FPGA’s clock control block; the second switches between different clocks that are phase-shifted 120° (1/3 of the clock period) from each other using the multiplexer on the clock control block.

To be able to detect that a transient has occurred and delay the next clock edge within the same cycle that the transient is produced, we use two different clocks. We have a system clock, which is the clock that the user application would use, and a faster detector clock, to detect transients and control the system clock.

4.1.3 Using Clock Enable

The idea behind this method is to disable the clock when a transient is detected so that the circuit will not see a rising edge until the transient has passed. When the clock enable on the Cyclone IV’s clock control block is low, the clock stays low; otherwise, it follows the input clock. The enable is latched on the negative edge, so we must raise the droop_detected signal before the negative edge. This gives us half a clock cycle to disable the clock. Our droop detector requires two clock cycles to detect a droop, so the detector clock must be 4x faster than the system clock.

Fig. 4.6 shows how we delay the next edge using the clock enable. The droop_detected signal has to arrive before the negative edge of the system_clk. In the diagram above, the clock enable falls before
$t = 4$, thus suppressing the system clock from rising at $t = 6$. The system clock edge rises at $t = 10$ when the droop has passed. By suppressing the clock edge for one cycle, the clock period is doubled for the cycle that a droop is detected. We can make the clock cycle even longer by keeping the enable low. However in Section 3.5.2, we do not see transients causing more than a 2x slow down in delay, so it unlikely that we would need to delay the clock edge for more than one cycle. By gating the clock for 1 cycle, we are effectively doubling the clock period and halving the frequency for that cycle.

### 4.1.4 Using Clock Multiplexing

The clock select on Cyclone IV’s clock control block allows us to select between 3 clock signals. We chose 3 clocks that are phase-shifted $120^\circ$ from each other. We can use one clock from a dedicated clock pin and 2 coming from a PLL that uses the first clock as its reference clock to create the phase shifts. When a droop is detected, we switch to the clock $-120^\circ$ shifted from the current clock. This switch-over must be done during the window when both clocks are low so that we delay the clock edge rather than make the clock edge come sooner. If we can guarantee that there is enough slack in the clock select signal, we can implement this method using a detector clock that is 3x the frequency of the system clock.

In Fig. 4.7, clk0-2 are the three clocks that are the inputs to the clock mux and clock_select selects which of these three clock currently drive the system clock. The system clock starts off by following clk0. We detect a droop at $t=4$. The window to switch to clk1 happens after clk1 falls but before clk0 rises (between $t=5.5$ and $t=6$) and clock_select switches to clk1 in this window. After the switch happens, the system clock stays low until $t=7$, increasing the clock period by 33%. If necessary, we can increase the clock period by another 33% by switching to clk2 between $t=6.5$ and $t=7$. When we detect another transient further in time, we will wrap around to clk0.
4.1.5 Implementation

We choose to use the clock enable approach to delay the clock edge because it is more straightforward. Our setup, which we call the clock edge suppressor, is shown in Fig. 4.8 and directly connects the output of the droop detector to the clock enable. One droop detector is sufficient to detect droops in all parts of the chip; however, a more conservative approach would be to have several droop detectors placed at different locations of the chip to ensure we are always reacting to the largest transient in any region. We choose to use just one droop detector because we can achieve higher frequencies by using one droop detector placed as close to the clock control block as possible. We use a PLL to generate two clock signals, a 1x clock and a 4x clock, which are in-phase with each other. The clock input of the clock control block is the 1x clock, and its output clock is the system clock. The 4x clock is used as the detector clock. The Fmax of the detector clock as reported by Quartus for Cyclone IV is 271 MHz, letting us test applications up to 67 MHz. On Stratix V, Quartus reports a detector clock frequency of 154 MHz, which allows application of up to 39 MHz.

To test how well the clock edge suppressor can prevent delay increases caused by transients, we test the edge suppressor with another delay measuring unit, shown in Fig. 4.9. The edge suppressor and the delay measuring unit uses 5% of the logic elements. We added 13K transient generators (C, from Table 3.4), with the entire design using 91% of the logic elements of the chip. Both the delay measuring unit and the transient generators use the system clock. Fig. 4.10 shows the number of bits propagated on the delay chain in the delay measuring unit. Without a margin in the droop detector, we can nearly negate the effects of transients. However, we find that the edge suppressor disables the clock too often when no transients are being produced, which effectively decreases the frequency of the system clock. With a 1, 3, or 5-bit margin, the edge suppressor only disables the clock during a transient event. Since the transients are being created every 5000 cycles, disabling the clock for one cycle ever 5000 cycles
does not significantly reduce the average frequency. With 1, 3, and 5 bits margins, the number of bits propagated in the delay measuring unit is reduced compared to the no transients case which indicates some delay push out due to transients. However, the reduction in bits propagated is significantly less when compared to not using the edge suppressor, particularly for the 1-bit and 3-bits case.

Next, we test an application circuit, a 64-bit multiply-accumulate (MAC) unit implemented using logic cells, by measuring how fast we can run the application with and without our edge suppressor. The results are in Table 4.2. Again with a 0-bit margin, we see that the clock is disabled too often. With a 1 bit margin, we can run the circuit at the same frequency as without transients; however, using a 3 or 5-bit margin was not effective. This shows that using a 1-bit margin, our design was able to mitigate the effects of transients. As we have shown in the previous Chapter, transients have larger effects for faster designs. Although these differences in frequency are small, they can be consistently reproduced.
Table 4.2: MAC unit benchmark using the clock suppressor.

<table>
<thead>
<tr>
<th></th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no transients</td>
<td>50</td>
</tr>
<tr>
<td>without suppressor</td>
<td>49.375</td>
</tr>
<tr>
<td>edge suppressor with 0 bit margin</td>
<td>—</td>
</tr>
<tr>
<td>edge suppressor with 1 bit margin</td>
<td>50</td>
</tr>
<tr>
<td>edge suppressor with 3 bit margin</td>
<td>49.375</td>
</tr>
<tr>
<td>edge suppressor with 5 bit margin</td>
<td>49.375</td>
</tr>
</tbody>
</table>

4.1.6 Limitations

For our current transient mitigation strategy to work, we make the following assumptions.

1. Transients have an impact on delay during the first quarter of the clock period. It is possible that some transients only impact the latter portion of the clock cycle, which our circuit will not be able to detect and react to quickly enough. However, in our experiments, we have shown that transients are fast, and will have an impact on delay very quickly, so we expect typical transients (e.g. coming out of reset, or ungating the clock) will occur at or shortly after the active clock edge.

2. Transients are generated from a single clock domain. One scenario that would not be covered is if there are multiple unrelated clocks which can produce transients at any time. In this scenario, we would need our edge suppressor to be fast enough so that most of the clock period is covered. If we are be able to cover three-quarters of the clock cycle. For the remaining quarter, we believe a guard-band will be sufficient.

3. The application can be paused. For latency-sensitive designs, delaying the clock may not be an option. However, the system clocks we target hold most of the FPGA logic, and are almost always pausable as they often communicate with I/O interfaces like DDR that do not have deterministic latency; hence they are decoupled from these I/O interfaces by FIFOs that can absorb a lost clock edge. There can be clocks that cannot tolerate a pause, but typically these latency-sensitive portions of the design are inside I/O interfaces and involve small clocks that are increasingly hardened inside the I/O controllers, and/or placed on a separate power supply to isolate them from the larger voltage transients produced by the larger system clocks.

4.2 Hardened Implementation

Although our implementation to mitigate transients can only target applications below 67 MHz, we believe there are several ways to increase this frequency. On Cyclone IV, we are limited by the clock enable being latched on the negative edge, giving us only half a clock cycle to react. If the enable was latched on the positive edge, or level sensitive as in Stratix10 [60], then we could potentially use the whole clock cycle to react, then potentially double the frequency. Additionally, if we can harden the droop detector, then we can target even higher frequencies. Previous work has shown that the typical frequency gap between FPGAs and ASICs is at least 3x [61]. Then with both of these optimizations, we could potentially target applications up to 366 MHz. Additionally, Cyclone IV is built on TSMC’s 60 nm node; we would also naturally achieve significantly higher performance on a smaller process node.
In the previous section, we showed that our soft-implementation of our edge suppressor was successful in mitigating the delay push-out effects of load transients; however, our solution was only effective at low frequencies. In this section, we explore what a hardened solution will look like and the performance we would expect. Although a hardened implementation will not address all the limitations in Section 4.1.6, it should allow us to target higher clock frequencies. We use TSMC’s 65 nm process kit and design flow for standard cells, which is similar to the 60 nm low power process node used for Cyclone IV.

4.2.1 Computing Hardened Path Delays

Our hardened solution assumes that we have direct control of the clock control blocks; that is, we can gate the clock directly at the root of the clock tree.

We define the following clocks:

- **gen_clk**: The signal generated by the clock control block. This is the signal at the root of the clock tree.
- **system_clk**: This is the clock that the user’s application employs. This is the signal at the leaf nodes of the clock tree.
- **detector_clk**: Used by the droop detector as detailed in the previous section.

The detector_clk is no longer constrained to be 4\times faster than the system_clk, but we are still constrained to having the detector_clk being an integer multiple of the system_clk. There are three tasks we need to perform: measuring bits propagated, comparing to a golden value to determine if a droop has occurred, and disabling the clock gen_clk. We define the following terms that we will use to calculate the performance of our hardened design:

- **transient detection delay** $\Delta_{td}$: Delay required to detect that a transient has occurred by comparing a bits propagated sample to a previously-sampled without transients golden pattern.
- **clock insertion delay** $\Delta_{ci}$: The delay for the clock to propagate from the root of the clock tree to its leaf nodes. This is the phase shift between gen_clk and system_clk in terms of delay.
- **m**: Multiplier between system_clk and detector_clk.
- **$T_d$**: Minimum clock period of the detector_clk.
- **$f_s$**: Maximum frequency of the system_clk.

We have the option of operating the detector_clk 2\times or 3\times faster than the system_clk, that is $m = 2$ and $m = 3$. If we select $m = 2$, we have two cycles to prevent the next clock edge of the system_clk from rising. In the first cycle, we sample the delay chain, and do not have a critical path. The second cycle is when we compare to the captured delay chain sample with the stored sample, and suppress the next clock edge if necessary. The limiting delay has to compare if a droop has occurred ($\Delta_{td}$) before the clock edge arrives at the leaf ($\Delta_{ci}$) since it is gated a the root. $T_d$ and $f_s$ can be computed with Eqs. (4.1) and (4.2) for $m = 2$. 
Table 4.3: Clock insertion delay on and delay to determine if a droop has occurred Cyclone IV and Stratix V in ns.

<table>
<thead>
<tr>
<th></th>
<th>Cyclone IV</th>
<th>Stratix V</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta_{ci}$</td>
<td>2.187 ns</td>
<td>3.191 ns</td>
</tr>
<tr>
<td>$\Delta_{td}$</td>
<td>0.046 ns</td>
<td>0.023 ns</td>
</tr>
</tbody>
</table>

Table 4.4: Performance of hardened edge suppressor on Cyclone IV and Stratix V.

<table>
<thead>
<tr>
<th></th>
<th>$m = 2$</th>
<th>$m = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone IV</td>
<td>$T_d$ = 2.233 ns, $f_s$ = 224 MHz</td>
<td>$T_d$ = 2.187 ns, $f_s$ = 152 MHz</td>
</tr>
<tr>
<td>Stratix V</td>
<td>$T_d$ = 3.214 ns, $f_s$ = 156 MHz</td>
<td>$T_d$ = 3.191 ns, $f_s$ = 104 MHz</td>
</tr>
</tbody>
</table>

$$T_d = \Delta_{td} + \Delta_{ci}$$ (4.1)

$$f_s = \frac{1}{2T_d}$$ (4.2)

If we select $m = 3$, then we have 3 cycles to suppress the next edge of the system_clk. The first cycle is to sample the delay chain. In the second cycle we compare the sampled pattern with the stored pattern to determine if there is a droop; in this cycle, the limiting path is $\Delta_{td}$. And finally, in the third cycle, we suppress the next clock edge, and the limiting path is $\Delta_{ci}$. $T_d$ and $f_s$ can be computed with Eqs. (4.3) and (4.4) for $m = 3$.

$$T_d = \max(\Delta_{td}, \Delta_{ci})$$ (4.3)

$$f_s = \frac{1}{3T_d}$$ (4.4)

Our goal is to maximize $f_s$ so that we can target a faster system_clk. Whether Eq. (4.4) or Eq. (4.2) yields a larger $f_s$ depends on the values of $\Delta_{td}$ and $\Delta_{ci}$. To obtain $\Delta_{td}$ for Cyclone IV, we use TSMC’s 65nm process kit, which we believe is similar to TSMC’s 60 nm low power technology node, which it uses. Stratix V uses TSM’s 28 nm process technology, which we do not have access to. Instead, we apply a scaling factor of 2, which is the factor between DSP speeds on Cyclone IV and Stratix V (250 MHz vs. 600 MHz for an 18×18 multiplier [16], [62]). We obtain $\Delta_{ci}$ for both Cyclone IV and Stratix V, from Quartus’ timing analyzer. In Table 4.3 shows values for $\Delta_{td}$ and $\Delta_{ci}$, and in Table 4.4, we report the computed values of $T_d$ and $f_s$ for both possible $m$ values.

From Table 4.4, $m = 2$ yields the highest system_clk frequency. Fig. 4.11 shows the timing diagram of how this system would operate. The detector_clk are system_clk are all aligned to each-other and slower than gen_clk by 0.3 cycles, which represents $\Delta_{ci}$. The droop_detected and disable_gen_clk, which is the signal that gates gen_clk, are combinations signal produced by ciruirty that is it is aligned to gen_clk. In the scenario presented, we detect a voltage droop at $t = 4$, and therefore must prevent the system_clk from rising at $t = 5$. To do this safely, we must disable gen_clk at $t = 4.4$, that is, before the rising edge of gen_clk. On Cyclone IV and Stratix V, we can achieve a frequency of 199 MHz and 156 MHz.
4.2.2 Gating at a Clock Tree Branch

In Section 4.2.1, we saw that the clock insertion delay $\Delta_{ci}$ is much higher than the delay of the comparison to determine if a transient is occurring $\Delta_{td}$. To reduce the clock insertion delay, we investigate gating the clock at an intermediate node in the clock tree rather than the root. In Fig. 4.12, we show a possible arrangement of a clock tree. In Section 4.2.1, we looked at gating the clock at the root, shown as the green dot; however, we could instead gate at one of the intermediate nodes, as shown red. Since Quartus does not expose the exact structure and delays within the clock network, we assume that we can gate at a node that halves the clock insertion delay. The delay required to cross half the chip should also allow us to propagate 70% of the way from the center of the chip to each corner, which should be sufficient to get to any intermediate clock node. Additionally, we assume that the delay to get to this node is approximately equal to the routing delay to cross half the chip. We introduce two new terms:

- Half the clock insertion delay $\Delta_{hci}$: $\frac{\Delta_{ci}}{2}$
- Half horizontal crossing delay $\Delta_{hhc}$: The delay required to propagate a signal vertically across half the chip. We use this to approximate the delay required to propagate the signal to gate the clock to all the intermediate nodes. $\Delta_{hhc(soft)}$ refers to this delay using the existing soft
Table 4.5: Half the clock insertion delay and horizontal crossing delay on Cyclone IV and Stratix V in ns.

<table>
<thead>
<tr>
<th>Delay Type</th>
<th>Cyclone IV</th>
<th>Stratix V</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta_{hci} )</td>
<td>1.094</td>
<td>1.596</td>
</tr>
<tr>
<td>( \Delta_{hhc(soft)} )</td>
<td>1.487</td>
<td>1.280</td>
</tr>
<tr>
<td>( \Delta_{hhc(hard)} )</td>
<td>0.620</td>
<td>0.533</td>
</tr>
</tbody>
</table>

Interconnect of the FPGA and \( \Delta_{hhc(hard)} \) refers to using a dedicated hardened wire for this path.

By gating at an intermediate node, there are 2 ways we could potentially achieve a faster \( f_s \). First, if we use \( m = 2 \), the new clock insertion delay is \( \Delta_{hci} + \Delta_{hhc} \). Then if \( \Delta_{hhc} \) is less than \( \Delta_{ci} \), we have successfully reduced the clock insertion delay. If we use \( m = 3 \), then we can propagate the signal to gate the clock in the second cycle to each intermediate clock node, which effectively allows us to break the clock insertion delay into two cycles.

Additionally, we update our formulae to compute \( T_d \) and \( f_s \). For \( m = 2 \), we use Eqs. (4.5) and (4.6). In the first cycle, we again sample the delay chain. In the second cycle is our timing limited path were we need to determine if a droop has occurred (\( \Delta_{td} \)), if a droop is detected, we need to propagate the signal to gate the clock to all the intermediate nodes (\( \Delta_{hhc} \)) and before the rising edge of the clock reaches the intermediate nodes (\( \Delta_{hci} \)).

\[
T_d = \Delta_{td} + \Delta_{hhc} + \Delta_{hci} \tag{4.5}
\]
\[
f_s = \frac{1}{2T_d} \tag{4.6}
\]

For \( m = 3 \), \( T_d \) and \( f_s \) can be computed with Eqs. (4.7) and (4.8). In the first cycle, we again sample the delay chain. In the second cycle, we need to determine if a droop has occurred then propagate this signal to all the intermediate clock nodes hence the limiting path is (\( \Delta_{td} + \Delta_{hhc} \)). Then in the third cycle, we gate the intermediate clock nodes and the limiting path is \( \Delta_{hci} \).

\[
T_d = \max(\Delta_{td} + \Delta_{hhc}, \Delta_{hci}) \tag{4.7}
\]
\[
f_s = \frac{1}{3T_d} \tag{4.8}
\]

We created a simple design consisting of two registers placed at half way across the chip to measure \( \Delta_{hhc(soft)} \) achievable using this existing soft interconnect. For \( \Delta_{hhc(hard)} \), we scale \( \Delta_{hhc(soft)} \) by 2.4, which is the factor found in [63] to be the difference between a soft and hardened interconnect. We report \( \Delta_{hhc} \) in Table 4.5. On Cyclone IV, \( \Delta_{hhc(soft)} \) is larger than \( \Delta_{hci} \) which means that we will not see an advantage with \( m = 2 \) since our new critical path is now longer than \( \Delta_{ci} \). We could still see a benefit by using a hardened interconnect or with \( m = 3 \) since we are breaking up the critical path. For Stratix V, \( \Delta_{hhc} \) is smaller than \( \Delta_{hci} \) and similar in magnitude, suggesting that we will see a benefit for both \( m = 2 \) and \( m = 3 \).

We recompute \( f_s \) in Fig. 4.13 and Fig. 4.14. With this approach, \( m = 3 \) results in the highest \texttt{system\_clk} frequency. On Cyclone IV, we can target applications up to 457 MHz, and on Stratix V we can target up to 313 MHz, which is much larger than what we achieved previously.
### Chapter 4. Mitigating Transients

<table>
<thead>
<tr>
<th>m = 2</th>
<th>m = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_d$</td>
<td>$f_s$</td>
</tr>
<tr>
<td>Cyclone IV</td>
<td>2.627 ns</td>
</tr>
<tr>
<td>Stratix V</td>
<td>2.899 ns</td>
</tr>
</tbody>
</table>

Figure 4.13: Performance of hardened edge suppressor when gating at an intermediate clock node on Cyclone IV and Stratix V using a soft interconnect.

<table>
<thead>
<tr>
<th>m = 2</th>
<th>m = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_d$</td>
<td>$f_s$</td>
</tr>
<tr>
<td>Cyclone IV</td>
<td>1.759 ns</td>
</tr>
<tr>
<td>Stratix V</td>
<td>2.152 ns</td>
</tr>
</tbody>
</table>

Figure 4.14: Performance of hardened edge suppressor when gating at an intermediate clock node on Cyclone IV and Stratix V using a hardened interconnect.

Figure 4.15: Using gating clock to suppress next edge using our hardened approach. We assume the clock insertion delay is 0.55 of the system_clk period. This approach uses $m = 3$. 
With \( m = 3 \), the timing, shown in Fig. 4.15 is slightly different from the previous section. We also introduce a node clk, which is the clock signal at the intermediate node. Instead of disable_gen_clk, we have disable_node_clk. The detector_clk, system_clk, and droop_detected are all still aligned to each other and are slower than node clk and gen_clk. The disable_node_clk is the signal that gates node_clk and is a combinations signal produced by circuitry that is aligned to the detector_clk.

In the scenario presented, we detect a voltage droop at \( t = 5 \), and therefore must prevent the system_clk from rising at \( t = 7 \). To do this safely, we disable node_clk at \( t = 6.1 \); this is after the rising edge of gen_clk, but before the rising edge of node_clk.

Using our hardened solution, new FPGAs could have a solution to mitigate transients that is approximately 7 times faster than a soft implementation.
Chapter 5

Improving Tolerance to Voltage Fluctuation

The prior chapters have examined the magnitude of voltage transients on FPGAs, and circuit techniques to detect and react to transients to avoid a timing failure. In this chapter, we explore a complementary approach: can we change the circuit design of an FPGA to make its delay less sensitive to droops in voltage? Through on-chip measurements, we can show that the delay though LUT dominated paths degrades significantly when reducing $V_{dd}$ compared to routing [64]. This observation is also supported by using HSPICE simulations. We focus on designing LUTs that are more robust to voltage variations, which makes them more tolerant of voltage transients and a better choice for lower-power FPGAs that support variable $V_{dd}$ levels. We explore different LUT designs and evaluate their tolerance to voltage variation while also keeping in mind their power consumption.

This work was done in collaboration with Ibrahim Ahmed and a portion of it was published in [64].

5.1 Methodology

We explore changes to the LUT so that future FPGAs could be more tolerant to voltage transients. To evaluate these designs, we use COFFE (Circuit Optimization For FPGA Exploration), an open-source automated transistor sizing tool for FPGAs. COFFE models an FPGA tile described in Fig. 5.1. It produces a complete transistor level netlist that replicates the critical path of each sub-circuit in the FPGA tile with appropriate loading. Using a divide and conquer approach, COFFE optimizes the transistor sizes for delay and area. Delay through each sub-circuit is measured using HSPICE, and a representative critical path delay, which is the weighted sum of delays through each sub-circuit based on their contribution to the critical paths of a set of benchmarks, is computed. To estimate area, COFFE uses its area models based on the transistor width after sizing that assumes close to square layout and accounts for N-well spacing. The count for each sub-circuit in an FPGA tile can be found in [65].

For all designs, we use PTM 22-nm HP models with a nominal $V_{dd}$ ($V_{nom}$ of 0.8V). We also scale the supply voltage between 0.6 V and 1.0 V, which is what we have determined to be the safe range for this process. Below 0.6 V, we start seeing pass-gates in the LUT no longer rising to a level that can be restored to $V_{dd}$. To model the designs that we want to evaluate, we needed to modify COFFE’s source code to describe our circuitry. The only modifications to the FPGA architecture were in the LUT; all
other sub-circuits, such as routing, were left untouched.

After COFFE sizes, it performs the following measurements which we will use to compare our designs:

- delay: The representative critical path delay, which we simply refer to as delay.
- area: The area of an FPGA tile. COFFE models areas in terms of minimum width transistor area.
- cost: area-delay product (area $\times$ delay)

Although our ultimate goal is to design a LUT that is robust towards transients, we want to do so in a way that also does not compromise on delay and area. Ideally, we would find LUT designs that not only are faster at lower voltages, but also have the same or better area-delay product at nominal voltage as the traditional (baseline) pass transistor LUT design.

In addition to delay and area, we evaluate two more metrics: $\Delta V_{\text{max}}$ and power, which are described in Section 5.1.1 and Section 5.1.2 respectively. $\Delta V_{\text{max}}$ quantifies tolerance to voltage fluctuations, and hence, how robust a design is against transients. All these metrics are relative, and for this reason, all measurements we present will be normalized to the baseline at 0.8 V for ease of comparison.

### 5.1.1 Voltage Tolerance

We define voltage tolerance in terms of delay margin. For example, if we allow a 5% variation in delay, then, the voltage tolerance is how much the voltage can decrease by, such that the delay does not increase by more than 5%. A higher voltage tolerance means that the design is more robust against transients. For each design, we measure delay across a set of supply voltages then perform a polynomial regression to obtain $d_n(v)$, which is the delay normalized to delay at nominal voltage, with respect to voltage; $d_n(V_{\text{nom}})$ is to be 1. We define $\Delta d$ as the delay increase that we can tolerate. Then, the voltage tolerance $\Delta V_{\text{max}}$ can be calculated by Eq. (5.1).

\[
\Delta d = d_n(\Delta V_{\text{max}}) - 1
\] (5.1)
TABLE 5.1: Power consumption of lutmasks implementing AND and OR of our baseline LUT, for a single input toggling in a single LUT at 250 MHz, measured in nW.

<table>
<thead>
<tr>
<th>lutmask</th>
<th>Input Toggling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td>AND</td>
<td>19.75</td>
</tr>
<tr>
<td>OR</td>
<td>6.70</td>
</tr>
</tbody>
</table>

There are two solutions for $\Delta V_{max}$, one when the supply voltage is above the nominal voltage, and one when below. We only report $\Delta V_{max} < d_n(V_{nom})$. This metric tells us that if we can tolerate a delay increase of $\Delta d$, that is, we apply a guard-band of $\Delta d$ to our circuit timing, then we can withstand the supply voltage decreasing by $\Delta V_{max}$. A larger $\Delta V_{max}$ indicates a design that is more robust against transients.

### 5.1.2 Modeling LUT power

In addition to area, delay, and voltage tolerance, we must also compare each design’s relative power consumption. We can use COFFE to model the power consumption of all sub-circuits except the LUT. This is because COFFE only models and sizes the critical path of the LUT instead of the entire LUT; this is sufficient for delay, but it neglects a large amount of switching circuitry that is important for power. Beyond having to model more circuitry, there are other key differences between the LUT and the other sub-circuits that makes modeling the LUT’s power more difficult:

- The function of the LUT is dependent on the lutmask, which can greatly affect the power consumption of the LUT. In Table 5.1, we report the power consumption of a 6-LUT input transition at 250 MHz that leads to a transition at the output for lutmasks that implement an AND and an OR, receptively. Each column in the table corresponds to a different input toggling, with a being the slowest input and f being the fastest, as shown in Table 5.1. For the slowest input, which toggles the largest number of pass transistor gates, the power difference between an AND vs. OR lutmask is over 3x.

- A single transition at the input does not always cause a transition at the output; however, it could cause internal switching. Additionally, multiple input transitions will produce different internal switching patterns compared to a single input switching. All these need to be considered when we model power.

- Not all nodes at the same LUT level transition together. That is, if we model only the critical path, and extract the power consumption at each level and multiply that value with the number of nodes at that level, we would over estimate power.

- There is skew between when inputs arrive. This causes significant glitching within the LUT as inputs arrive in succession. Any method that models all the inputs switching simultaneously would omit the power consumed by these glitches.

Since we are comparing LUT designs, we need as accurate a model of power consumption as possible. The method we use is as follows:

1. Create an HSPICE netlist for a full LUT.
2. Randomly generate a lutmask and randomly generate an input vector.

3. For each of the 6 LUT inputs $i$, measure power $P_i$ from input $i$ transitioning.

4. Measure static power $P_s$, where no inputs transition.

5. Since each measurement in 3 includes static power, then the total LUT power is computed as:
   \[ \sum_i P_i - 5 \times P_s \]

6. Repeat steps 2-5 100 times, and compute the average.

We are therefore modelling what we believe to be an average LUT power rather than the worst-case or best case power. We believe this method is more accurate than that used in prior works [66] because we are modeling the case where all the input transitions are not aligned in time, and hence model internal glitching.

Now that we have power measurements for each sub-circuit, we can compute the power consumption for an FPGA tile. To do this, we first use VPR to place and route a set of benchmarks to get the utilization of each type of sub-circuit. Then for each sub-circuit, we scale its power by its average utilization. Then, to compute the power of the entire FPGA tile, we sum the scaled power of each sub-circuit. This implicitly assumes each benchmark has a switching activity of 100% (one transition per signal per cycle), which is sufficient for the relative power comparisons across different LUT designs we are interested in, but would not be sufficient to obtain accurate absolute power for each benchmark.

Although we can compare the power consumption for all LUT designs at one frequency, this approach would ignore the reality that the different LUT designs have different delays, and hence, the same benchmark would not operate at the same frequency with each architecture. Consider two designs that consume different amounts of power when operating at the same frequency, but the lower power design is slower. It is entirely possible that we can reduce the supply voltage of the faster design to match the delay of the slower LUT design consuming less power. To properly reflect this, we also plot benchmark frequency vs power consumption for each LUT circuit with the supply voltage being adjusted to the minimum that meets each frequency. That is, for a given benchmark at each possible target frequency, we determine the best power consumption achievable by each LUT circuit design with adaptable voltage. We will go into more details of how we compare power consumption later on in Section 5.4.

### 5.2 LUT Designs

#### 5.2.1 Baseline design

The baseline design we use is the one presented in the original COFFE paper, which is a conventional pass-gate based design of a 6-LUT as shown in Fig. 5.2. Each local mux select which wires to connect to the LUT’s input; the local mux’s output connect to input drivers which drive the gates one stage of the pass transistor mux. Each of our proposed designs is a variant of a 6-LUTs so to not affect other factors such as how benchmarks are mapped to the FPGA. We describe a 6-LUT as having 6 levels, where a level $L_i$ refers to a layer of pass gates with the same input at the gate. $L_1$ is the first level of pass-gates where the drain of each nmos is connected to an SRAM cell. The number of pass-gates at each level $L_1$ to $L_6$ are: 64, 32, 16, 8, 4, 2. We label our inputs $a$ to $f$, where input $a$ controls the gates at $L_1$. We call the baseline design $PT_{\text{base}}$. 
5.2.2 Proposed designs

Revisiting Buffer Location of Baseline Design

We believe it is worth exploring whether the placement of buffers at different LUT levels could yield a faster FPGA. Even though non-gate boosted pass-gates are sensitive to voltage fluctuations, which changing buffer locations does not address, it is still possible to still see slight improvements. Each level is implemented using pass-gates and each buffer is implemented as an inverter.

We define three additional variations similar to the baseline which are described in Table 5.2. Squares represent a LUT level and triangles represent a buffer. We group these variations in a set which we name PT.

Transmission Gates

The study in [65] showed that gate-boosted pass gates result in an FPGA tile with a better area-delay product (at nominal $V_{dd}$) than using transmission gates. The use of transmission gates instead of pass-gates results in a better delay, but at the expense of more area. The LUT however, was not gate-boosted, leaving them very sensitive to voltage fluctuations. Transmission gates have the attractive property that they are less sensitive to voltage variation as they can drive their output to full $V_{dd}$ without the use of gate boosting or weak pull-up transistors. Our idea is to explore using transmission gates in LUTs only, while still using gate-boosted pass gates in all routing multiplexers. This design enables better LUT delays, especially at lower voltages. Although we will be adding area in the LUT, unlike [65], we will not be adding area to all the routing multiplexers of the FPGA tile.

Additionally, we also explore different configurations of buffers, which we call $TG_{lut, base}$, $TG_{lut, 1}$,
Pre-Decoding

LUTs have traditionally been built using a tree-based multiplexer, with each level of the tree controlled by a different input. We can instead decode groups of inputs and use the decoded values to control a single level of pass transistors for that group of inputs. By reducing the number of pass transistors in series, input decoding could yield a LUT with lower delay sensitivity to voltage. Fig. 5.3 shows part of a 6-input LUT that decodes the slowest two inputs ($a$ and $b$). By decoding the first two inputs, we can remove the $L_2$ of the LUT shown in Fig. 5.2, which saves 32 pass transistors. However, we add 24 transistors for the decoding circuitry, resulting in a net reduction of 8 transistors. We implement the decoding circuitry as NAND and inverter gates using CMOS logic, which is known to be tolerant to voltage variation [67]. Using a NAND and an inverter allows us to use minimum width transistors for the NAND gate and size the inverter to drive the decoder load (16 pass transistors). Using NOR gates for decoding instead would result in adding fewer transistors, but would require larger transistors in the NOR gate to drive its load of 16 pass transistors. Decoding the first three or four inputs (instead of two) results in adding a net of 16 and 104 transistors, respectively, to the LUT. The number of transistors in the decoding circuitry grows exponentially with the number of decoded inputs, so we designed LUTs that decode either the first two or three inputs only. While one could decode any group of inputs, we focus on the first (slowest) group of inputs as decoding them saves the most pass transistors. In addition,
both Intel and Xilinx’s fracturable 6- LUT designs share at least the first two inputs across both 5-LUTs in fractured modes, so the same decoding circuitry works without modification. Decoding inputs also reduces the capacitive loading on the input drivers. Without decoding, the driver of the first LUT input $a$ is loaded by 32 pass transistors, but with decoding, it is loaded by only two NAND gates. The load reduction enables a smaller driver and thus a smaller area.

We explore decoding 2 and 3 inputs, as well as where to buffer within the LUT, which are described in Table 5.3. We group these variations in a set which we name $PD$. The first subscript for each PD indicates how many levels of inputs are pre-decoded (2 or 3), and the second subscript indicates how many pass transistor levels exist before the first stage of buffering.

**Gate-Boosting the LUT**

Motivated by the observation that the delay of gate-boosted routing multiplexers is more tolerant of voltage variations, our idea is to gate boost the LUT multiplexer. To achieve this, we power the LUT input drivers by the gate-boosted SRAM voltage ($V_{ddh}$) instead of the lower and varying logic supply voltage. Since the remaining logic and routing is still powered by the lower supply voltage ($V_{ddl}$), we modify the local multiplexer (MUX) to convert the voltage level from ($V_{ddl}$) to the higher SRAM voltage ($V_{ddh}$). Fig. 5.4a and Fig. 5.4b shows two variations of the local MUX that are able to convert the voltage from $V_{ddl}$ to $V_{ddh}$. It is important to perform this conversion to prevent excessive leakage of current consumption from the input drivers. Both local MUX variations implement a two-stage multiplexer followed by a buffer. In the variation shown in Fig. 5.4a, the buffer is powered by $V_{ddh}$ and a conventional level shifter [68] is used to convert the output of this buffer to $V_{ddh}$ so that it can be connected to the input drivers. In the variation shown in Fig. 5.4b, the buffer is powered directly with $V_{ddh}$, but it does not leak because the weak pullup is also connected to $V_{ddh}$. This pull-up ensures that when there is a logic high at the input of the buffer, the PMOS in the buffer is completely turned off. We call these design $GB_{shift}$ and $GB_{pmos}$ which we group in a set named $GB$.

**5.3 Optimizing for Delay and Area**

We first optimize for area and delay, then select the best design within each set ($PT$, $TG_{lut}$, $PD$, and $GB$). The cost, area, and delay at 0.8 V and 0.6 V are shown in Table 5.4 and Table 5.5. We omit $PT_3$ as it is not robust as many internal nodes would not transition during sizing; this makes it both a dangerous circuit choice and led to it having poor delay even with transistor sizing that functioned. At 0.8 V, all of our designs perform better than our baseline, with designs in $TG_{lut}$ and $GB$ showing better potential than $PD$. Since we want designs with high tolerance to voltage fluctuation, we consider cost
## Table 5.4: Area, delay, and cost of FPGA tile for each LUT designs at 0.8 V normalized to \( PT_{\text{base}} \).

<table>
<thead>
<tr>
<th>LUT Types</th>
<th>Tile Area</th>
<th>Rep. Crit. Delay</th>
<th>Cost (Area × Delay)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PT )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PT_{\text{base}} )</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>( PT_1 )</td>
<td>0.90</td>
<td>0.94</td>
<td>0.84</td>
</tr>
<tr>
<td>( PT_2 )</td>
<td>0.92</td>
<td>0.86</td>
<td>0.80</td>
</tr>
<tr>
<td>( PT_3 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>( TG_{\text{lut,base}} )</td>
<td>1.10</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>( TG_{\text{lut,1}} )</td>
<td>0.95</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td>( TG_{\text{lut,2}} )</td>
<td>0.92</td>
<td>0.79</td>
<td>0.73</td>
</tr>
<tr>
<td>( TG_{\text{lut,3}} )</td>
<td>0.82</td>
<td>1.14</td>
<td>0.93</td>
</tr>
<tr>
<td>( PD )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PD_{2,2} )</td>
<td>0.98</td>
<td>0.97</td>
<td>0.95</td>
</tr>
<tr>
<td>( PD_{2,3} )</td>
<td>1.02</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>( PD_{3,1} )</td>
<td>0.99</td>
<td>0.97</td>
<td>0.96</td>
</tr>
<tr>
<td>( PD_{3,2} )</td>
<td>1.04</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>( GB )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( GB_{\text{shift}} )</td>
<td>1.08</td>
<td>0.80</td>
<td>0.87</td>
</tr>
<tr>
<td>( GB_{\text{pmos}} )</td>
<td>1.08</td>
<td>0.79</td>
<td>0.86</td>
</tr>
</tbody>
</table>

## Table 5.5: Area, delay, and cost of the FPGA tile for each LUT designs at 0.6 V normalized to \( PT_{\text{base}} \).

<table>
<thead>
<tr>
<th>LUT Types</th>
<th>Tile Area</th>
<th>Rep. Crit. Delay</th>
<th>Cost (Area × Delay)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PT )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PT_{\text{base}} )</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>( PT_1 )</td>
<td>0.90</td>
<td>1.72</td>
<td>1.54</td>
</tr>
<tr>
<td>( PT_2 )</td>
<td>0.92</td>
<td>0.87</td>
<td>0.80</td>
</tr>
<tr>
<td>( PT_3 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>( TG_{\text{lut,base}} )</td>
<td>1.10</td>
<td>0.56</td>
<td>0.62</td>
</tr>
<tr>
<td>( TG_{\text{lut,1}} )</td>
<td>0.95</td>
<td>0.55</td>
<td>0.52</td>
</tr>
<tr>
<td>( TG_{\text{lut,2}} )</td>
<td>0.92</td>
<td>0.51</td>
<td>0.48</td>
</tr>
<tr>
<td>( TG_{\text{lut,3}} )</td>
<td>0.82</td>
<td>0.86</td>
<td>0.71</td>
</tr>
<tr>
<td>( PD )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PD_{2,2} )</td>
<td>0.98</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>( PD_{2,3} )</td>
<td>1.02</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>( PD_{3,1} )</td>
<td>0.99</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>( PD_{3,2} )</td>
<td>1.04</td>
<td>0.87</td>
<td>0.90</td>
</tr>
<tr>
<td>( GB )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( GB_{\text{shift}} )</td>
<td>1.08</td>
<td>0.45</td>
<td>0.48</td>
</tr>
<tr>
<td>( GB_{\text{pmos}} )</td>
<td>1.08</td>
<td>0.45</td>
<td>0.49</td>
</tr>
</tbody>
</table>
Chapter 5. Improving Tolerance to Voltage Fluctuation

Figure 5.5: $\Delta V_{\text{max}}$ vs. allowable $\Delta d$ for each LUT design at 0.8 V.

<table>
<thead>
<tr>
<th>$\Delta V_{\text{max}}$</th>
<th>PT-2</th>
<th>TG-2</th>
<th>gb-lvl</th>
<th>gb-pmos</th>
<th>Decode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.93%</td>
<td>7.65%</td>
<td>10.54%</td>
<td>10.36%</td>
<td>4.91%</td>
</tr>
</tbody>
</table>

Table 5.6: Voltage Tolerance as a percent of $V_{\text{nom}}$ at $\Delta d = 10\%$.

at 0.6 V as well. Again $TG_{\text{lut}}$ and $GB$ show the most potential. We see that there are designs in $PT$ that outperform all the designs in $PD$; indicating that pre-decoding pass transistor LUTs is not helpful. It is possible that predecoding is beneficial for other metrics such as power. However, since our main goal is to optimize for voltage tolerance, we will eliminate the set $PD$ from further consideration.

When comparing the designs in sets $PT$ and $GB$, we see that $PT_2$ and $TG_{lut,2}$ outperform the other designs within their sets both in terms of cost and delay. Compared to the base, these designs use more inverters, 42 vs 18, however, the inverters and each LUT level are sized much smaller. Even with the additional of 24 inverters, $PT_2$ and $TG_{lut,2}$ still outperform $PT_{\text{base}}$ and $TG_{\text{base}}$ in terms of both area and delay and therefore also in cost.

$GB_{\text{shift}}$ and $GB_{\text{p}}$ are very similar on all three metrics even though the local MUX shown in Fig. 5.4b requires fewer transistors. Since we want tolerance to voltage fluctuations, we also compare the same metrics at 0.6 V. We select $GB_{\text{shift}}$ because it has a slightly better performance at low voltage. In general $GB_{\text{shift}}$ and $GB_{\text{p}}$ are very similar, but we believe $GB_{\text{shift}}$ is slightly more optimal when considering voltage tolerance.

The LUT designs that we will compare to $PT_{\text{base}}$ are: $PT_2$, $TG_{lut,2}$, and $GB_{\text{shift}}$. 
5.4 Optimizing for Voltage Tolerance and Power

Now that we have narrowed down our designs to those with the best area and delay at 0.6 V and 0.8 V, we can evaluate which design is the best able to handle voltage fluctuation and is thus most robust against transients.

Fig. 5.5 shows $\Delta V_{max}$ vs $\Delta d$ and Table 5.6 shows $\Delta V_{max}$ at $\Delta d = 10\%$. We see that at every $\Delta d$, all our designs have better voltage tolerance than the base design. $PT_2$ only marginally outperforms the base. $GB_{shift}$ has much higher voltage tolerance than $TG_{lut,2}$ and both are significantly more tolerant to voltage fluctuations than the baseline. At $\Delta d = 10\%$, the baseline design was only able to tolerate a 4.7% decrease in voltage, whereas $TG_{lut,2}$ and $GB_{shift}$ can tolerate 7.7% and 10.5% receptively. This shows that we have achieved our goal of creating a LUT design that is more tolerant towards voltage fluctuation and hence more robust towards transients. Since $PT_2$’s voltage tolerance is not nearly as high compared to the other two designs, we will not be analyzing it further.

5.5 Comparing Power

As one of the key goals of this thesis is to enable power savings through the use of lower voltages, we also strive for a design that is low power. Even though $TG_{lut,2}$ and $GB_{shift}$ are significantly more tolerant to voltage fluctuations than our base design, we still need to consider their power consumption versus that of the baseline LUT design.

Fig. 5.6 shows normalized power when operating at 250 MHz at different supply voltages. This shows that when operating at the same frequency and same supply voltage, $TG_{lut,2}$ and $GB_{shift}$ both
consume more power than our baseline. However, from Table 5.4 and Table 5.5, we know that $TG_{lut,2}$ and $GB_{shift}$ are actually faster devices, and therefore, at the nominal voltage, $TG_{lut,2}$ and $GB_{shift}$ should be able to perform more computations at the same frequency. Under the assumption that we can scale supply voltage, we can compare two designs at the same frequency but different voltages so that they have the same delay. There are can be up to two voltages that meet this frequency target since delay increases as the supply voltage is increased to be above nominal [64]. Of the two solutions, one will consume more power. Fig. 5.7 shows this comparison averaged across all VTR benchmarks when $V_{dd}$ is scaled between 0.6 V and 1 V. The dotted line represents the higher voltage, higher power solution that also meets the frequency target. We believe this is a better comparison of power as it reflects both power as well as how fast each design is. To understand this graph, we first compare the values horizontally. This shows that for the same power budget, $TG_{lut,2}$ and $GB_{shift}$ can achieve higher frequency targets. Conversely, if we look at the values vertically, we can say that $TG_{lut,2}$ and $GB_{shift}$ can meet the same frequency targets and consume less power.

It is interesting to note that while $TG_{lut,2}$ uses more transistors, and $GB_{shift}$ brings the LUT to a higher voltage, both of which typically suggests higher power consumption, this is not the case. These designs do indeed consume more power when comparing at a fixed frequency and voltage, but they consume less power when we allow variable $V_{dd}$ and compare power at the same delay. From Fig. 5.7 we can see that it if we scale their supply voltage of $TG_{lut,2}$ and $GB_{shift}$ to 0.7 V and 0.75 V receptively, they consume the same power as $PT_{base}$ with a supply voltage of 0.8 V yet can achieve a higher frequency.

However, at lower voltages, we are more vulnerable to transients potentially causing catastrophic failures since we are operating at a voltage closer to one where the FPGA no longer operates. Hence we also need to look at voltage tolerance at 0.7 V and 0.75 V. From Table 5.7, $GB_{shift}$ still has a higher voltage tolerance then the base, however $TG_2$ does not. Using $GB_{shift}$, we can withstand a supply
voltage decrease of 8.52% to 0.69 V, which is still within the safe range of operation for the supply voltage, before we see more than a 10% increase in delay.

Our top design $GB_{shift}$ is a better LUT in terms of area-delay product than our baseline across the entire 0.6 to 0.8 V range. It has higher voltage tolerance and therefore is more robust to transients. When operating at a fixed frequency and nominal voltage, it consumes more power than the baseline. However, given that the LUT design is faster, we can either perform more computations at nominal supply voltage, or we could meet the same power budget as the baseline LUT by operating at a lower supply voltage, while still being faster and more robust towards transients.

<table>
<thead>
<tr>
<th>Design</th>
<th>$V_{dd}$ (V)</th>
<th>$\Delta V_{max}$ (%)</th>
<th>$V_{dd} \times (1 - \Delta V_{max})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PT_{base}$</td>
<td>0.8</td>
<td>4.93</td>
<td>0.76</td>
</tr>
<tr>
<td>$TG_2$</td>
<td>0.7</td>
<td>4.14</td>
<td>0.67</td>
</tr>
<tr>
<td>$GB_{shift}$</td>
<td>0.75</td>
<td>8.52</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Table 5.7: Voltage Tolerance with scaled supply voltage at $\Delta d = 10\%$. 
Chapter 6

Conclusion

In this thesis, we have shown that transients can have a large impact on delay in FPGAs and we proposed and evaluated strategies for current and future FPGAs to mitigate their impact. Our results highlight the need for transient mitigation strategies to enable lower voltage FPGA operation, and maintain their power advantage over CPUs.

In Chapter 3, we showed that transients can reduce the measured Fmax of a design by up to 25%. One implication of this for dynamic voltage scaling is that designs are either vulnerable to timing failures or are sacrificing a lot of potential power savings. Using our delay measuring circuits, we were able to quantify the effects of transients on both Cyclone IV and Stratix V. The effects of transients can be fast and have an immediate impact on the cycle in which they were produced. The largest transients on both devices have similar delay push-out; however, the hardware-measured safe frequency margin over Quartus’ reported clock frequency is much smaller on Stratix V. In fact, the margins we measured with transients at nominal $V_{dd}$ were negative at frequencies higher than 150 MHz for Stratix V.

In Chapter 4, we used existing FPGA resources to build a droop detector that is able to detect when the on-chip voltage drops. We then leveraged this detector to build a clock edge suppressor, which delays the next clock edge when a large transient is occurring. Using our clock edge suppressor, we can mitigate the effect of the delay increases caused by fast load transients, opening the door to more aggressive yet still robust DVS for FPGAs. Additionally, we proposed changes to the FPGA’s clock network that would optimize our edge suppressor and allow it to target higher frequencies. With our proposed changes, we should be able to mitigate transients for designs up to 457 MHz on Cyclone IV and 313 MHz on Stratix V.

In Chapter 5, we proposed new circuit designs for LUTs that are more tolerant to voltage fluctuations and therefore more robust to transients. In addition to being more robust to transients, our best LUT design is faster at nominal voltage or can be operated at a lower voltage at the same speed as a traditional LUT, thereby consuming less power while delivering the same performance.

6.1 Future Works

In this thesis, we’ve shown that FPGAs have a real need to mitigate transients. In our work, the most recent device we use is Stratix V. Possible future work could include measuring and mitigating transients on more recent FPGAs such as Stratix 10. This would involve porting our designs to Stratix 10 and
then performing the same experiments. It would be interesting to see how this larger and faster device is
affected by transients. We’ve also shown that with a hardened droop detector and hardened interconnect,
we can mitigate transients for designs at higher frequencies. Our mitigation strategy involves gating the
clock for one cycle, effectively decreasing the clock frequency by half. Future work could look at exploring
hardened changes to the clock control block similar to that used by AMD or Intel previously described in
the background section. This would give us finer control of the clock and allow for mitigation strategies
that produces smaller changes to the clock frequency.

Another avenue for future work is exploring CAD approaches that could prevent transients from
occurring in a user’s design. This could involve creating a tool to examine a design to look for large
resets, clock gates, and produce a report to assist the designer in removing the possibility of large
transients. This tool could also modify the user’s design to spread out a load transient such as releasing
a reset in several cycles to reduce the size of the transient. Additionally, it could add a signal to warn of
an impending transient; this would allow our transient mitigation circuit to respond faster or to decrease
the clock frequency in advance.
Bibliography


