RELAXED QUASI DELAY-INSENSITIVE CIRCUITS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Christopher Charles LaFrieda

February 2010

RELAXED QUASI DELAY-INSENSITIVE CIRCUITS

Christopher Charles LaFrieda, Ph.D.

Cornell University 2010

Deep submicron technologies are beginning to scale poorly with respect to both power and performance. It is well known that adding timing assumptions to asynchronous circuits can help to simplify circuits and improve performance. Thus, applying timing assumptions can help to extend the effectiveness of technology scaling. However, employing timing assumptions in deep submicron technologies is risky because of the large process variations that are present. This thesis explores the use of low risk timing assumptions to improve asynchronous circuits.

We begin with a well-established and robust asynchronous logic style, quasi-delay insensitive (QDI) circuits. We expose a timing assumption that exists in the feedback of QDI circuits and extend it for general use. We refer to the resulting logic family as relaxed quasi delay-insensitive circuits (RQDI). RQDI circuits maintain much of the robustness of QDI circuits while providing improved power and performance. Evaluations show that replacing QDI circuits with RQDI equivalents can reduce area and energy by 20% and 36%, respectively.

RQDI also allows for new types of circuits which are difficult to design using strictly QDI logic. We present RQDI circuits for voltage scaling and two phase signaling. The voltage scaling circuits are novel because they allow for independent voltage scaling of the forward path (data rails) and the return path (acknowledges). The two phase circuits are presented in the context of static switching networks, such as those found in the routing networks in a field-programmable gate array (FPGA). Evaluations show that our two phase circuits can reduce energy con-

sumption in these structures by more than 50% with an area overhead of less than 10%.

To further evaluate RQDI circuits, we design an asynchronous FPGA using RQDI two-phase circuits and RQDI voltage scaling circuits. For eight of the MCNC LGSynth93 benchmarks, RQDI two-phase circuits provide up to a 70 % performance improvement and up to a 40 % power reduction. The RQDI voltage scaling circuits provide an additional 30 % power reduction across these benchmarks.

BIOGRAPHICAL SKETCH

Christopher Charles LaFrieda attended Midwood High School in Brooklyn, NY. He graduated in 1997 from their Medical Sciences Program. Christopher then began his undergraduate studies at the State University of New York at Buffalo. In May of 2001, Christopher graduated from SUNY at Buffalo with a B.S. in both Computer Science and Computer Engineering.

Christopher began his study at Cornell University in August of 2001. He became a member of Rajit Manohar's Asynchronous VLSI design group in 2003 and received his MS in 2005. His research interested were wide-ranging and included design automation, fault tolerance, computer graphics, 3D integrated circuits and asynchronous FPGA design. In February 2007, he took a leave of absence to work at Achronix Semiconductor. At Achronix, he was a principal designer on their first commercial product, the SPD60. He returned to Cornell in 2008 to complete the Ph.D. program.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Prof. Rajit Manohar, for taking a wayward graduate student under his wing. For all our discussions and debates, and for his continued support in all my research endeavors, I thank him. I would also like to thank the other members of my committee, Jose Martinez and David Albonesi, for our collaborations, their critiques of my work, and helping to make the Computer Systems Lab (CSL) here at Cornell the success it is today. I thank Ed Suh for agreeing to proxy during my B Exam on Dave Albonesi's behave.

I thank the other members of the AVLSI research group here at Cornell: Filipp Akopyan, Benjamin Hill, Sandra Jackson, Rob Karmazin, Carlos Tadeo Ortega Otero, Basit Sheikh, and Jonathan Tse. Their hard work, dedication and eagerness to help has created an environment conducive to the unbridled exchange of ideas and concepts. Special thanks to Ben Hill for providing me with the FPGA benchmarks used in Chapter 6.

I thank my colleagues at Achronix Semiconductor, especially: Virantha Ekanayke, David Fang, Ilya Ganusov, Clint Kelly, Chris Liu, Rajit Manohar, and Lily Tam. Working with these experts in asynchronous design is a truly enriching experience.

I would like to thank my parents, Patrick LaFrieda and Barbara Galvagni, my brothers and sister, Patrick, Joseph and Michele, for supporting me in all ways possible.

Most of all, I have to thank my wife Amy for always being encouraging and understanding, for putting up with all the boring nights watching me work, for having to attend social events solo, and for all the delicious home-cooked meals that have sustained me the past few years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Technology Trends

At the time this introduction is being written, something interesting is happening in the semiconductor world. The usual progression of 30%-scaled technology nodes is being circumvented. Many semiconductor customers are skipping the 45 nm technology node [11] and going straight to a 32 nm or 28 nm technology from a 65 nm technology. This practice is being encouraged by foundries, such as IBM and Intel, who are focusing more on developing 32 nm and 28 nm processes than on their 45 nm process [8]. *What is the reason for the sudden jump in technology scaling?*

The reason for this paradigm shift in process evolution is simple. Process scaling is no longer packing the punch that it used to. Transistor threshold voltage, V_{th}, no longer scales well [12] resulting in a slower scaling of the supply voltage, V_{dd}. This has a negative impact on static power, dynamic power, and performance scaling. The drop in performance scaling can be mitigated by skipping process technologies. However, this also introduces some unwanted side effects.

With each new process technology comes new challenges in achieving an adequate yield. These challenges are overcome by a combination of tweaking current process steps and adding new ones. In some cases, the design rules for a technology must be altered to avoid low yield structures. This not only increases a circuit's complexity, but may also increase its area. Accelerating the schedule for developing process nodes will undoubtedly lead to more complex design rules. Evidence of this can be seen by observing the dummy transistors added to design rules at the 45 nm and 32 nm process nodes [3]. In order to protect the vulnerable transistors

at the edges of a stack, dummy transistors need to be added at both ends of each stack. Such rules add new levels of complexity for circuit designers and hurt overall circuit density.

Technology scaling through the deep submicron has greatly increased the importance of minimizing power consumption in circuit design. With poor supply voltage scaling, dynamic power and static power increase with each technology node. Scaling through multiple nodes exacerbates this problem. When working with a power budget, circuit performance may need to be throttled down in order to meet power constraints. In this sense, lower power circuits may perform better than circuits that are designed solely for high speed. Therefore, the performance benefits of accelerated scaling may be lost without reducing the additional power consumption.

1.1.1 Power Scaling

Figure 1.1 shows how the supply voltage, V_{dd}, scales from the 130 nm process node to the 22 nm process node. Up until the 65 nm node, V_{dd} scaled by roughly 20 % per technology. For the 45 nm and 32 nm nodes, V_{dd} only scales by 10% and 5.6%, respectively. Although it is too early to tell exactly how V_{dd} scales going into the 22 nm node, based on the current trends it is not unreasonable to assume that it will not scale much at all. Such poor V_{dd} scaling leads to dramatic increases in power consumption.

Typically, the equation for dynamic power consumption is given as follows:

$$P_D = \alpha C {V_{dd}}^2 F \tag{1.1}$$

The symbol α is the activity factor, C is the capacitance, V_{dd} is the supply voltage and F is the frequency. We can redefine C in terms of the number of transistors,

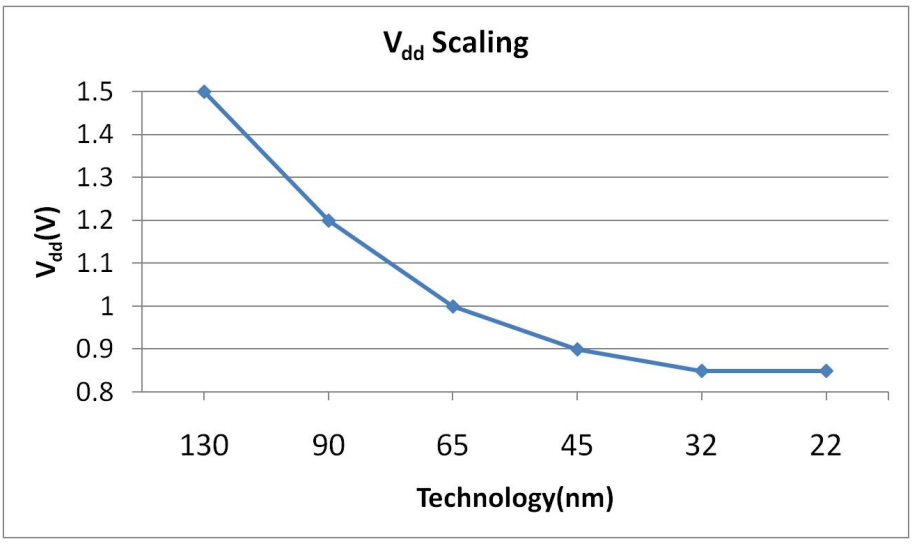

Figure 1.1: Supply voltage scaling.

N, and the average capacitance per transistor, C_{avg}, as follows:

$$P_D = \alpha N C_{avg} {V_{dd}}^2 F \tag{1.2}$$

We can readily determine how each of these terms scale with the exception of frequency. Without major modifications to a design, the activity factor will remain roughly constant with scaling. As per Moore's Law [22], the total number of transistors will double with each technology node. The average capacitance will decrease by 30% per major technology node due to the smaller feature sizes. This information yields the following equation for power scaling:

$$\frac{P_{Di}}{P_{Di-1}} = \frac{{V_{ddi}}^2}{{V_{ddi-1}}^2} \frac{F_i}{F_{i-1}} \tag{1.3}$$

By applying the supply voltage scaling in Figure 1.1, we can plot dynamic power as it scales with a fixed frequency as shown in Figure 1.2. Prior to the 45 nm node, scaling with a fixed frequency resulted in a dynamic power savings of up to 10%. At the 45 nm, 32 nm, and 22 nm nodes there is a 13%, 25%, and 40% increase in dynamic power, respectively. The power increase is exacerbated

by scaling through multiple nodes. For example, scaling from 65 nm to 32 nm results in a 44% increase in dynamic power.

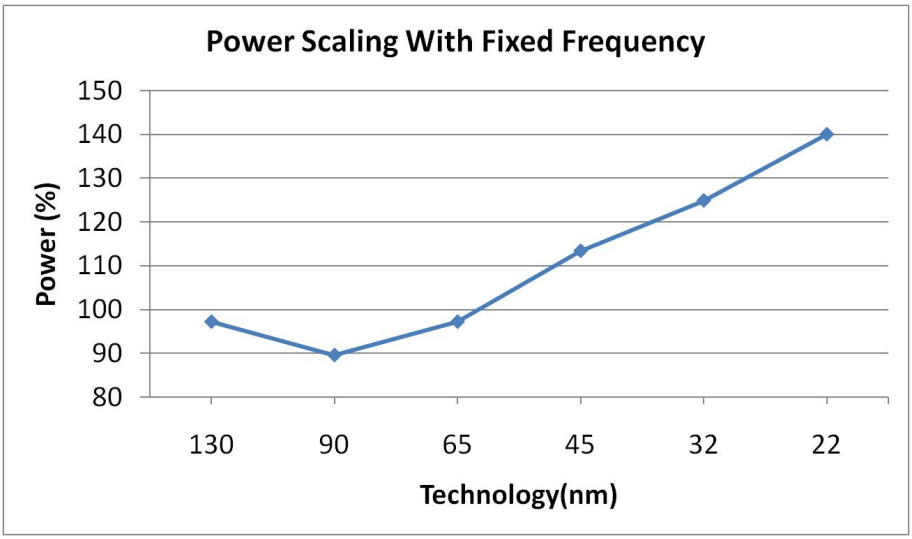

Figure 1.2: Dynamic power scaling with a fixed frequency.

As one would imagine, static power also increases significantly due to poor V_{dd} scaling. However, this thesis deals primarily with logic, rather than memory, where static power is overshadowed by dynamic power. Figure 1.3 depicts the ratio of dynamic power to static power for a typical asynchronous logic cell across a wide range of temperatures. Even at 125° Celsius, dynamic power is still 300 times greater than static power in this logic cell. For this reason, we will not consider static power in this thesis. Although, all power numbers reported will be total power (including the rather insignificant static power).

1.1.2 Process Variations

Another design concern in deep submicron technologies is the increasing impact of process variations. Process variations are the result of both systematic and random effects. Systematic effects exhibit a high degree of spatial correlation and they usually manifest themselves as die to die variations. Random effects can

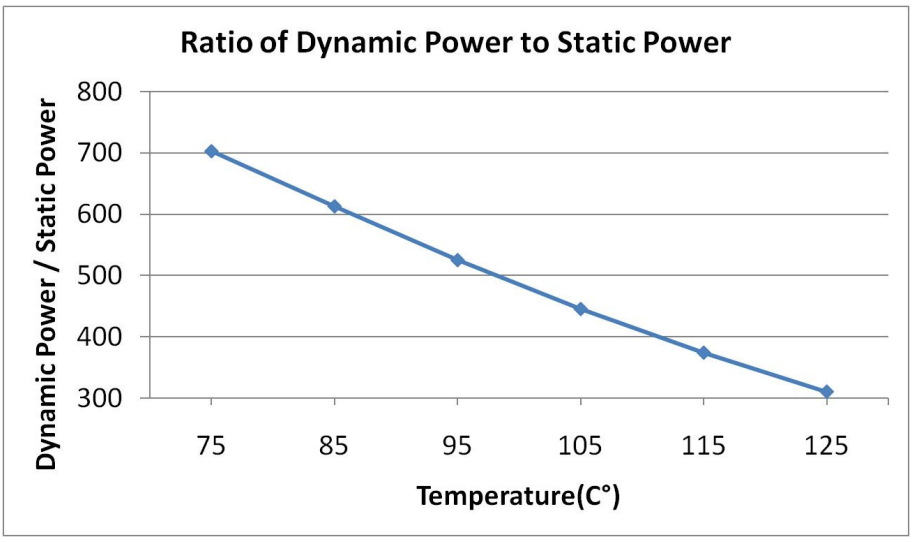

Figure 1.3: The ratio of dynamic power to static power for a typical asynchronous logic cell in 65 nm.

cause the critical dimensions of adjacent devices to differ significantly. These types of variations are particularly bothersome because they can throw off the relative timing of neighboring circuits. According to the The International Technology Roadmap for Semiconductors (ITRS), the occurrence of random process variations will increase sharply as processes scale down to the 22 nm technology node[1]. Table 1.1 lists the FO4 delays across the slow-slow (SS), typical-typical (TT), and fast-fast (FF) process corners. In this 65 nm process there is a 70% difference in delay between the slowest, SS, and fastest, FF, corners.

Table 1.1: FO4 delay across process corners in a 65 nm process.

SS Corner	**TT Corner**	**FF Corner**
13.6 ps	18.2 ps	22.6 ps

1.2 Asynchronous Circuits

Asynchronous circuits have some advantages over synchronous circuits that may help to mitigate the effects of the process trends outlined in the previous section.

Specifically, asynchronous circuits are low power, robust to process variations, and high throughput due to fine-grain pipelining.

1.2.1 Low Power Design

Asynchronous circuits have a couple of advantages over synchronous circuits in terms of low power design. The lack of a clock network is a substantial advantage. High-speed clock networks have been known to account for 30-35% of total power in microprocessors [7]. In addition, asynchronous circuits have the equivalent of perfect clock gating. High performance asynchronous circuits are composed of many parallel processes (fine-grain pipelined circuits). These processes communicate over channels using handshakes. Processes that are not involved in the current computation do not burn dynamic power.

Unfortunately, asynchronous circuits lose some of their power savings in orchestrating handshakes between processes. Four phase handshakes, used extensively in quasi delay-insensitive (QDI) circuits, charge and discharge wires in their data channels four times per cycle. The power dissipated in channels is significant since wires there tend to be longer than wires that are local to a process. A significant amount of power is also lost in generating enable/acknowledge signals. This is particularly frustrating since those signals are not directly involved in computing the function of a particular process, but rather in detecting the validity and neutrality of inputs and outputs.

1.2.2 Robustness

Perhaps the greatest benefit that asynchronous circuits provide is their high tolerance to process variations, temperature and voltage scaling. QDI circuits detect each transition in a computation, and therefore are not sensitive to timing mis-

matches due to process variations. QDI design is inherently highly reliable. This is proven by first silicon successes such as: i) the Caltech MiniMIPS [19], ii) the Cornell 3D FPGA [4], and iii) the Achronix Speedster [23]. In addition, a high-performance FPGA from Cornell was demonstrated to operate correctly for a wide range of voltages, .13V to 2.3V, and a wide range of temperatures, 77K to 400K in a 180 nm process[5].

1.2.3 Fine-Grain Pipelining

Modern QDI design produces fine-grain pipelines with minimal effort. This style of pipelining allows designers to create high-throughput systems where conventional synchronous solutions struggle. For example, the Achronix Speedster can run at 1.5 GHz [20], which is three times faster than leading commercial synchronous FPGAs. Much of this speed advantage is due to the buffering in the interconnect, which is easier to accomplish in asynchronous logic due to the lack of retiming constraints.

1.3 Relaxed Quasi Delay-Insensitive Circuits

Relaxed quasi delay-insensitive (RQDI) circuits aim to maintain much of the robustness of QDI circuits while reducing area and power consumption. Rather than check every transition in a computation, RQDI circuits use a set of highly conservative timing assumptions to simplify logic. Although many timing assumptions have been proposed to simplify asynchronous logic, RQDI timing assumptions have two unique benefits: i) they have large timing margins of 300% or more, and ii) they do not increase circuit latencies.

1.3.1 Contributions

In this thesis we present the following circuit techniques to improve area, power consumption, and/or performance:

- **HCHB Template:** We define a new circuit template that reduces the logic needed to generate enable/acknowledge signals by applying an easily satisfied timing assumption.
- **Voltage Scaling:** We implement voltage scaling in two ways. One, we design efficient voltage converters that operate on data channels to support multiple voltage domains. Two, we present a circuit template that operates with its forward path (data logic) in a nominal voltage domain and its return path (enable/acknowledge) in a lower voltage domain, thus keeping latency constant.
- **Two Phase Circuits:** We propose an efficient two phase buffer and protocol converters for global communication and static switching networks, which are particularly important in FPGAs [27].
- **RQDI FPGA:** We apply the above techniques to an asynchronous FPGA.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 reviews asynchronous circuit design. Chapter 3 defines our conservative timing assumptions and presents an RQDI logic template. Chapter 4 introduces a two phase circuits for static routing and their associated protocol converters. Chapter 5 introduces circuits that perform voltage scaling. Chapter 6 presents a case study involving an asynchronous FPGA and we conclude in Chapter 7.

CHAPTER 2

BACKGROUND

2.1 QDI

Quasi delay-insensitive circuits are designed by decomposing a high level description of an asynchronous system into production rules (pull-up and pull-down networks) through numerous steps [17]. For our purposes, we will focus on handshaking expansions (HSE) and pre-established circuit templates.

2.1.1 Four Phase Dual Rail Protocol

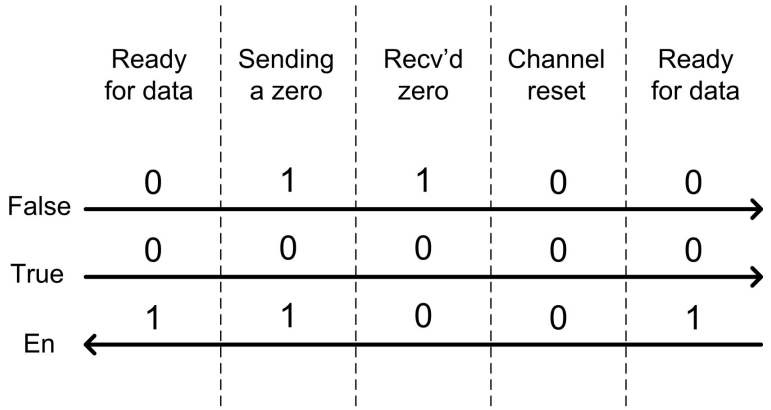

Figure 2.1: Four phase dual rail protocol.

The basic QDI primitive for a bit of data is a set three wires, shown in Figure 2.1. Two of these wires encode data in dual rail form and the third wire is used as an enable (the more natural inverted version of an acknowledge). After reset, both data rails are low and enable is high. A high enable signal indicates that the following stage of logic is ready to receive new data. At this time, one of the data rails may go high. If the false rail goes high, a zero is being sent. If the true rail goes high, a one is being sent. When the data has been received, the enable goes

low and the data rails reset. Finally, the enable may go high again.

2.1.2 Simple Buffer

A commonly used QDI circuit is the weak condition half-buffer (WCHB) shown in Figure 2.2 and described with the following HSE:

$$*[[R^e \wedge L^f \longrightarrow R^f\uparrow [] R^e \wedge L^t \longrightarrow R^t\uparrow]; L^e\downarrow;$$
$$[\neg R^e \wedge \neg L^f \wedge \neg L^t]; R^f\downarrow, R^t\downarrow; L^e\uparrow]$$

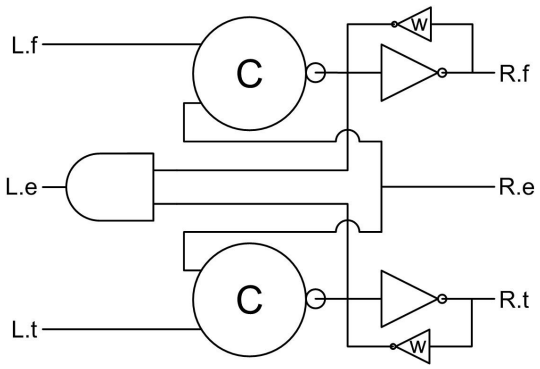

Figure 2.2: A WCHB.

The WCHB takes a dual-rail (four-phase) input and produces a dual rail output. There are two distinct phases: i) an evaluation phase (the first line of the HSE) where inputs arrive and the output becomes valid, and ii) a reset phase (the second line of the HSE) where the inputs and output reset. It is considered a half-buffer because it takes a pair of them to store a single data token. It has a latency (time it takes to propagate a token) of two transitions and a cycle time (time from receiving a token until it can receive another) of ten transitions. The WCHB contains two c-elements, circles marked with a 'C', which are state holding elements. The charge on state holding elements is kept with staticizers, which are weak feedback inverters (on the order of 10x weaker than the gate itself). Staticizers will not be explicitly drawn in the remainder of this thesis.

2.1.3 Logic Template

A WCHB is handy for buffers, but the precharge half-buffer (PCHB) is preferred for buffered logic [15]. The PCHB template for two inputs and one output is shown in Figure 2.3 and the HSE for the PCHB is as follows:

$$*[[R^e \wedge L^f \longrightarrow R^f\uparrow [] R^e \wedge L^t \longrightarrow R^t\uparrow]; L^e\downarrow;$$
$$[\neg R^e]; R^f\downarrow, R^t\downarrow; [\neg L^f \wedge \neg L^t]; L^e\uparrow]$$

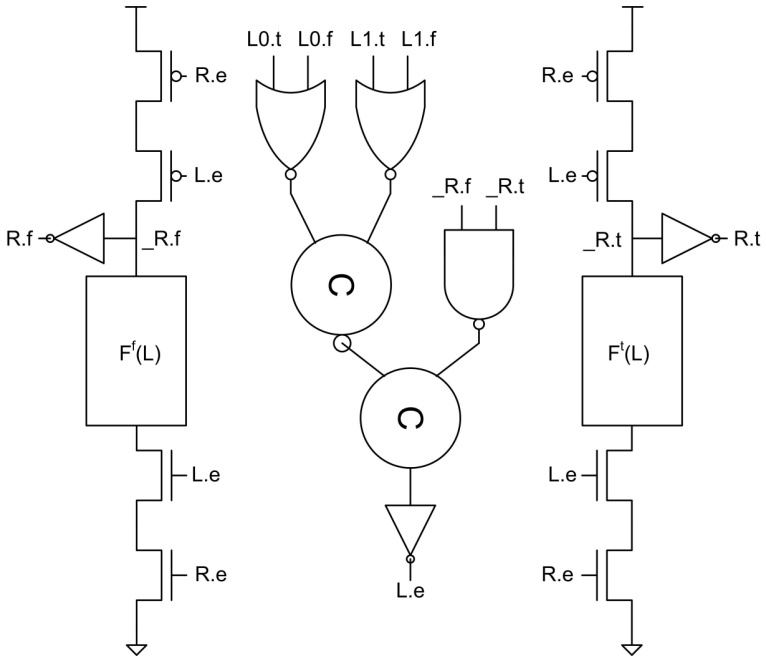

Figure 2.3: A two input and one output PCHB template.

Similar to dual rail domino logic [29], QDI logic computes the true and false rails for each operation. Input validity/neutrality and output input/neutrality can be detected with simple AND or OR gates. The PCHB reshuffling arranges the validity and neutrality checks so that they overlap the evaluation and reset of the computation. The PCHB template has a latency of two transitions and a cycle time of 14 transitions. The main difference from the WCHB is that the neutrality of the inputs is detected on $L^e\downarrow$ rather than $R\downarrow$. As a result, input neutrality can be detected in multiple transitions without impacting latency, which allows for a

greater number of inputs. In a WCHB, neutrality is detected in the pull-up stack of the data rails. This limits the number of inputs possible in a WCHB because more inputs means more series PMOS transistors in the pull-up stack. Generally, we limit ourselves to three series PMOS in any stack. Any more and the rise time will be poor and charge sharing in dynamic nodes becomes problematic.

An extension to the PCHB is the PCEHB. In the PCEHB, R^e and L^e are combined in a separate c-element. This improves the latency by reducing the number of series PMOS and NMOS in the data rail stacks by one. However, the PCEHB increases the cycle time by four transitions (a non-inverting c-element is two transitions and it needs to be set and reset in one cycle) making the total 18 transitions.

2.2 Asynchronous Circuit Performance

2.2.1 Cycle Time

The cycle time of a circuit can be defined as the time from when it receives a token until it can receive another token. The cycle time of a linear pipeline is equal to the cycle time of its slowest stage. Figure 2.4 shows a linear pipeline fed with an ideal token source. As tokens flow through the pipeline, they will spread until there is a single token for every *cycle time*/*latency* buffers. If we observe the output of the pipeline, a token will appear once every cycle time. Thus, the throughput of the pipeline is 1/*cycle time*. In high performance design, a reasonable cycle time target is 18 transitions.

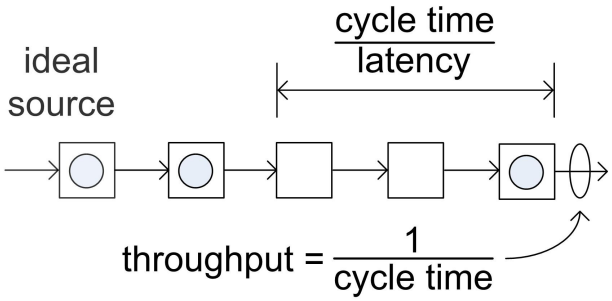

Figure 2.4: A linear pipeline fed with an ideal source.

2.2.2 Latency

The latency of a circuit can be defined as time from when a token arrives at the input until a token is produced at the output (assuming that the following stage is ready for a token). In a linear pipeline, throughput is solely dependant on cycle time. However, complex systems are composed of pipelines with loops and reconvergent paths where latency may limit throughput. A common (and grave) mistake made in high performance asynchronous design is to put too much emphasis on optimizing cycle time and not enough emphasis on optimizing latency.

Figure 2.5 show a pipeline that contains a loop that limits throughput. In order for loops to run at optimal throughout they must contain a sufficient number of tokens. The number of tokens in a loop remains fixed during normal operation. Therefore, these tokens must be inserted into the loop at system reset and are appropriately named initial tokens. It is not always possible to place enough initial tokens in a loop to run at full throughput. In these cases, the only way to improve throughput is to minimize the latency.

2.2.3 Slack Matching

The simplest way to understand the importance of slack matching is to examine a pipeline with mismatched slack. An example of mismatched slack is shown in

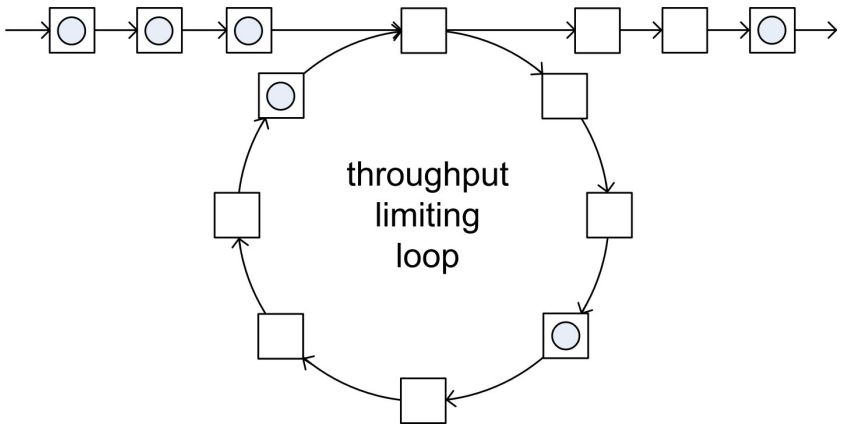

Figure 2.5: A throughput limiting loop.

Figure 2.6. In this figure, we show how tokens flow through two paths over the course three cycles. For simplicity, latency and cycle time are equal to one cycle. In Cycle 0, tokens can not exit the bottom path because Stage J requires tokens from both top and bottom paths to proceed. After two cycles, in Cycle 2, Stage J can produce a new token. However, during these two cycles, Stage F was unable to produce new tokens because there wasn't enough slack in the bottom path. Stage F is one again able to produce a new token in Cycle 3. In this example, adding two buffers to the bottom path would double the throughput of the pipeline.

2.3 Timing Margins

Process variations result from systematic and random effects. Traditionally, the systematic effects dominate. This leads to intra-die variations that can be addressed by binning die based on speed and leakage. Unfortunately, the ITRS[1] predicts a sharp increase in random effects as we move through the 22 nm process node. In the face of random effects, even adjacent devices can behave differently. If the random component become significant, then binning at the die level becomes less effective because many die will contain slow devices. Each design style has

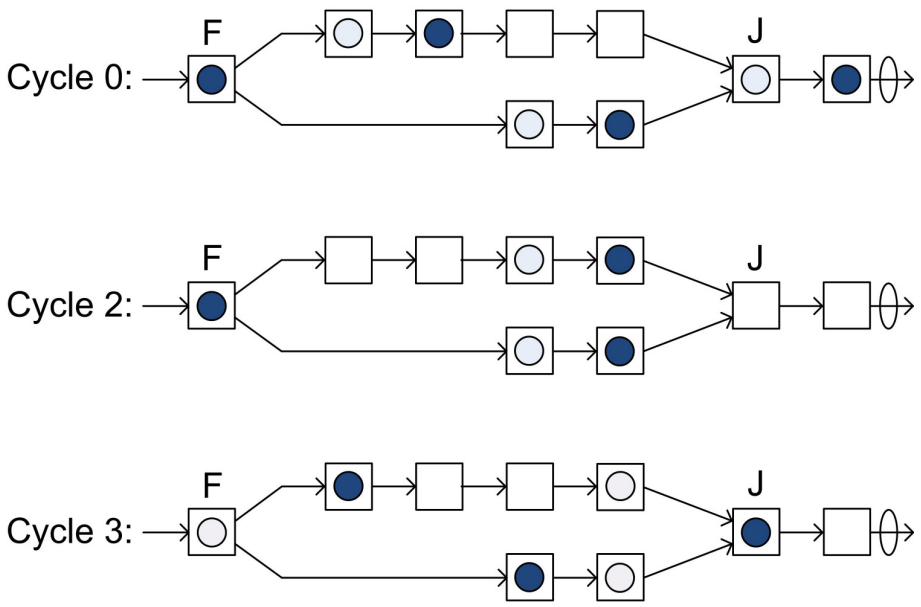

Figure 2.6: The impact of mismatched slack. Stage F represents a fork and Stage J represents a join.

their own methods of adding timing margins to address process variations.

2.3.1 Clocked Designs

A typical stage of synchronous logic is shown in Figure 2.7. The amount of logic contained between flops is chosen based on the target operating frequency and some timing margin. In a stage of synchronous logic, the cycle time and latency are both a single clock cycle. An advantage of this scheme is the ability to adjust the clock frequency externally, in order to increase the timing margin. If all delays are relative to the clock, then the frequency can be tailored based upon the process variations exhibited by a particular die. However, this scheme may be suboptimal in the face of random process variations. If the clock frequency is lowered to accommodate a few slow stages, then the latency and cycle time of the other stages will be unnecessarily increased.

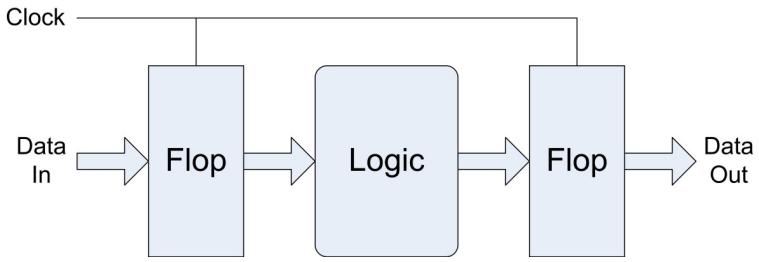

Figure 2.7: A stage of synchronous logic.

2.3.2 Micropipelines

Micropipelines encode data using bundled data [25]. Bundled data is a single rail scheme (each bit of data is represented with a single wire) that contains a valid signal and an acknowledge signal. Rather than bitwise encoding of data and timing information, like dual rail, bundled data encodes the timing for a group of bits (bundle) in a single valid wire. The timing for each stage, and some built in timing margin, is implemented by a fixed delay element, as shown in Figure 2.8.

This scheme is inferior to a clocked scheme because the delay can not be adjusted externally. A more practical solution would be to add some programmable delay that can be controlled by an external signal. This delay would likely have to be applied chip-wide due to limited pins and limited ability to pinpoint delay faults on a modern chip. With the addition of a programmable delay, micropipelines become equivalent to the synchronous case in terms of adjusting timing margins.

2.3.3 QDI

With QDI logic, timing margins are unnecessary. Timing information is encoded atomically with data. Completion of any operation can be determined by examining the data rails. In the face of random process variations, QDI logic will automatically adjust the timing of the slower stages and leave the faster/nominal

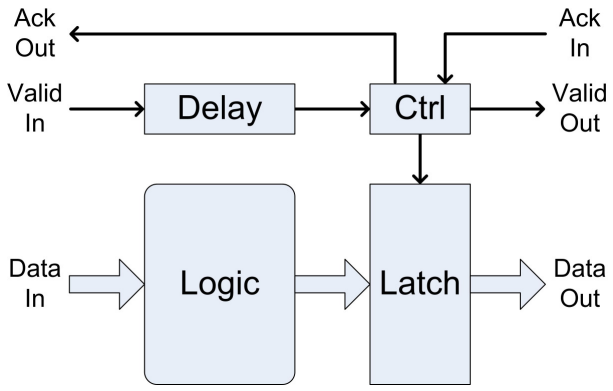

Figure 2.8: A micropipeline.

stage unaffected.

2.4 Circuit Guidelines

In general, we try to stick to the following guidelines for high performance asynchronous logic:

1. Avoid three transition or less cutoff paths so that signals are full swing.
2. Keep the latency of each stage to two transitions.
3. Keep the cycle time of each stage within 18 transitions.
4. Dynamic nodes cannot be directly shared between stages. These signals must be buffered first.
5. Limit PMOS to three in series and NMOS to five in series.
6. The output of all state holding logic is staticized (held by weak feedback).

Guideline 3 may be flexible depending on the application. We often find that even with a worst case cycle time of 18 transitions the frequency of the system is limited by the latency of loops with a suboptimal number of tokens. For this reason we place greater priority on optimizing latency over cycle time. Guideline 4 is meant

to prevent bit flips on dynamic nodes. Dynamic nodes are more susceptible to crosstalk because there are times that they are only driven by weak feedback.

Guideline 5 is important for two reasons. First, too many devices (especially PMOS) in series results in excessively slow transitions. At this point, it is faster to break the stage into multiple transitions. Second, this many series devices leads to large internal capacitances which results in charge sharing problems.

CHAPTER 3

RQDI

3.1 Relaxed QDI Logic

QDI circuits are quite robust in terms of process variations and design tolerances. In this work, we expose a timing assumption used in staticizers for QDI logic and apply it to other parts of circuits. Our goal is to optimize circuits with respect to area and power while maintaining the robustness of QDI. The resulting circuits are no longer strictly QDI. We refer to them as relaxed QDI (RQDI).

3.1.1 Half Cycle Timing Assumption

The WCHB, PCHB, and PCEHB circuit templates (and QDI circuits in general) are highly tolerant of process variations because each up and down transition is sensed. The only timing assumption allowed in QDI design is the isochronic fork assumption [18]. This timing assumption states that the difference in delay between branches of a wire is insignificant compared to the gate delays of the logic reading their values.

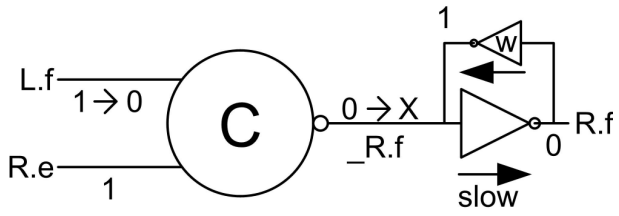

Figure 3.1: An error that can occur in QDI logic without the half cycle timing assumption.

Upon closer inspection, however, there is a second timing assumption that is quite common in QDI circuits. Observe the false rail of a WCHB shown in Figure 3.1. In order for this circuit to work properly a timing assumption is

made with respect to its staticizer. Let us assume that the inverter driving R^f is incredibly slow. $\_R^f\downarrow$ has fired, due to $L^f\uparrow$ and $R^e\uparrow$, but $R^f\uparrow$ has not. When $L.f\downarrow$ fires, the c-element becomes state holding and the only active current is the weak feedback. Even though the inverter is weak, it can flip $\_R^f$ because there is no opposing current. The resulting error is due to an actual analog problem and not an isochronic fork.

To avoid such timing errors with staticizers we introduce the half cycle timing assumption. The half cycle timing assumption (HCTA) is a local timing assumption (internal to a process) that assumes a small amount of logic (one or two transitions) will always switch within one half cycle of a process. With cycle times of 10-18 transitions, this assumption has a timing margin of 2.5x-4.5x. In addition, during the half cycle communication occurs across the channels where wires tend to be longer than wires internal to a process. Transitions across these wires will be slower, making the half cycle even longer compared to the two transition logic.

In QDI, the HCTA is only needed to guarantee the correct operation of staticizers. We can reduce logic and design new valid circuits by extending the HCTA for general use. We refer to the resulting logic as relaxed quasi-delay insensitive (RQDI). RQDI logic has a robustness similar to QDI logic because they both use the same timing assumptions and have the same timing margins.

3.1.2 HCTA Timing Margin

Table 3.1 compares the timing margins across different circuit families. QDI and RQDI exhibit extremely large timing margins without any impact on their latency or cycle time. With cycle times of 10 to 18 transitions, the up to two transitions of an HCTA have a timing margin of 250% to 450%. In synchronous logic, timing margins are increased by reducing the clock frequency. This will simultaneously

increase both latency and cycle time. Similarly, the timing margin in bundled data logic is increased by increasing the delay in the control logic.

Table 3.1: Timing margins associated with various circuit families. Symbols m, L, and C are the timing margin factor, latency, and cycle time respectively.

Circuit Family	**Timing Margin**	**Tradeoff**
Synchronous	m	mL, mC
Bundled Data (two-phase)	m	mL, mC
Bundled Data (four-phase)	m	mL, $2mC$
QDI (staticizers)	$2.5x - 4.5x$	none
RQDI	$2.5x - 4.5x$	none

To illustrate the large timing margins that can be achieved with an HCTA, observe the loop in Figure 3.2. Each stage of this pipeline has a two transition latency and a ten transition cycle time. There is a five stage loop with a single token. This pipeline is balanced in the sense that as the token makes one trip around the loop, the two input stage would be ready to process the token.

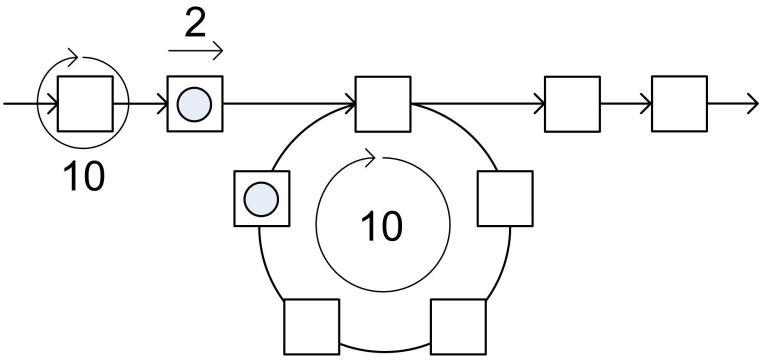

Figure 3.2: A loop that demonstrate the impact of timing margins on throughput. Each stage has a cycle time of ten transitions and a latency of two transitions.

Figure 3.3 graphs the frequency versus the timing margin for both HCTA and forward path timing margins for the pipeline in Figure 3.2. Each transition is assumed to be 50 ps, which is close to an average transition in a 65 nm process. In this example, an HCTA can have a timing margin of up to 250% without any impact on the throughput. For a forward path timing assumption, such as the one used in synchronous logic and micropipelines, this large timing margin will result

in an operating frequency that is almost four times slower than nominal.

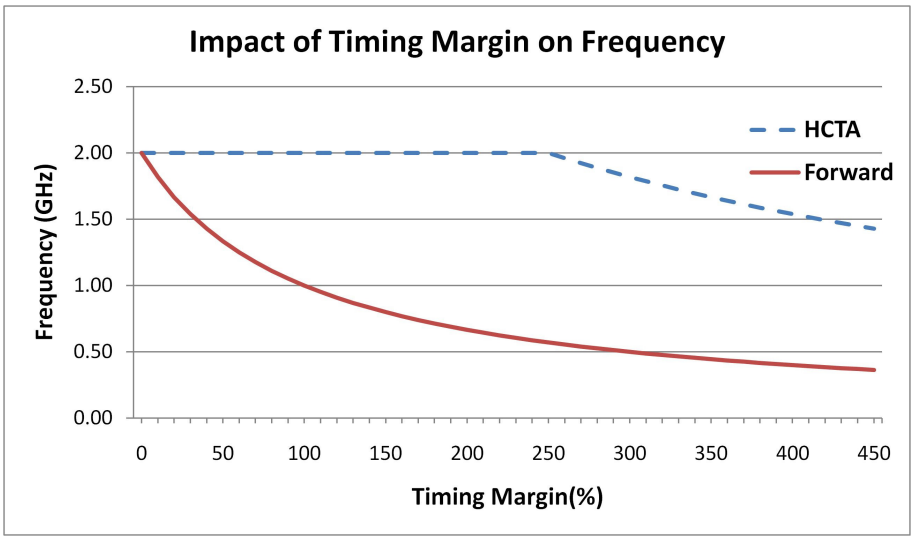

Figure 3.3: The impact of timing margins on frequency for an HCTA timing assumption and forward path assumptions.

3.1.3 HCHB Circuit Template

We can reduce the logic needed to compute the neutrality in logic templates by applying the HCTA. Consider the following HSE for the half cycle half-buffer (HCHB):

$$*[([R^e \wedge L^f \longrightarrow R^f\uparrow [] R^e \wedge L^t \longrightarrow R^t\uparrow]; L^e\downarrow;), \{N\downarrow\};$$
$$[\neg R^e], ([\neg L^f \wedge \neg L^t]; N\uparrow); R^f\downarrow, R^t\downarrow; L^e\uparrow]$$

We've introduced a variable N, for neutrality, with the intention of only sensing the $N\uparrow$ transition. N detects the neutrality of L and it can be implemented as the nor of L^f and L^t. $N\downarrow$ can fire at the beginning of the evaluation phase (first line of HSE) when L becomes valid, but doesn't need to fire until the beginning of the reset phase (second line of HSE) before $R^e\downarrow$ arrives, a half cycle later. We make the assumption that $N\downarrow$ will fire before the second half of the cycle and add no logic to detect this transition.

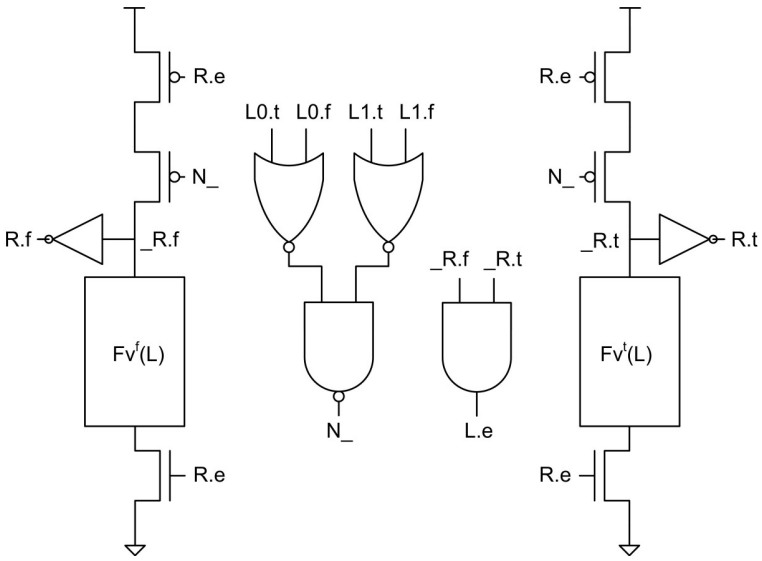

Figure 3.4: A two input and one output hchb template.

Applying the half cycle timing assumption results in the HCHB template shown in Figure 3.4. Validity and neutrality are checked in the data rails similar to the WCHB. This reduces the logic for L^e and gets rid of one series NMOS in the pull down stack. It takes two transitions to detect the neutrality of the inputs. This does not affect the two transition latency of the circuit, but makes the cycle time 14 transitions. An additional requirement of the HCHB over the PCHB is that the pull down networks of the data rails need to wait for all the inputs to become valid before firing. In some cases the pull down stacks already wait for validity. In other cases the pull down stacks need to be augmented to wait for input validity.

Figure 3.5 shows the false rails of a PCHB and HCHB and2 process and their corresponding transistor widths. The false rail of the HCHB has been extended to wait for the validity of both $L0$ and $L1$. The HCHB pull down stack has two more transistors than the PCHB, but area saved elsewhere in the circuit more than makes up the difference. In some circuits, e.g. a full adder, the pull down stack will already guarantee input validity with no additional effort. In circuits with many inputs, the validity can be checked in a separate single rail pseudo output.

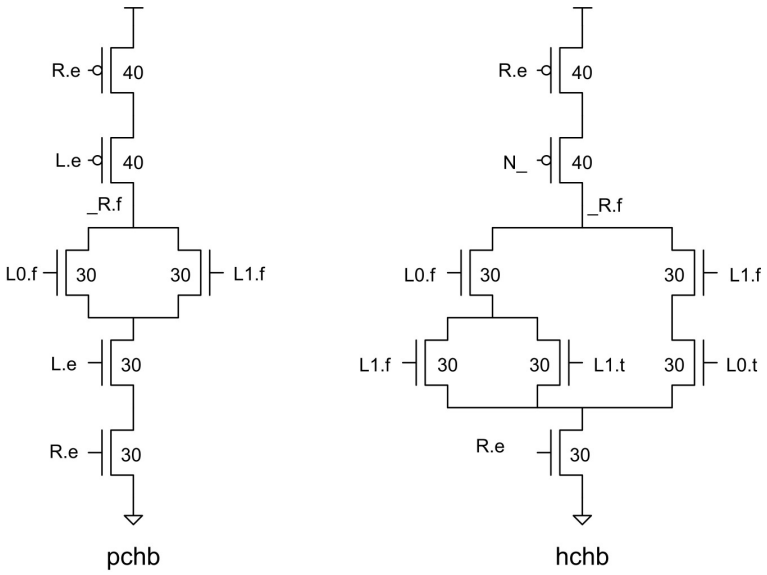

Figure 3.5: The false rail stacks of an and2 process for a PCHB(left) and a HCHB(right). The numbers are the transistor widths in lambda units (half minimum gate length).

However, the biggest area and power savings occur when validity can be checked in the pull down stacks of the data rails.

3.1.4 RQDI Testing Challenges

A nice feature of QDI circuits is that they can be made to deadlock in the presence of stuck-at faults [10, 13]. In some cases, RQDI circuits do not strictly maintain this property. For example, if $N_{\_}$ in Figure 3.4 is stuck at zero, then the buffer could reset prematurely. This type of fault may be hard to detect because failure depends on the timing of R^e versus the input data rails on reset. One potential solution is to check $N_{\_}$ in L^e. However, this would increase the area and the cycle time. Another solution is to test the circuit at multiple voltages in an attempt to induce the error. Ultimately, one may be willing to accept this additional risk. The stuck-at fault model is an abstraction, and as a result it does not model all possible faults [9]. The $N_{\_}$ stuck-at-zero fault could also have been modeled as a

transistor-stuck-on fault. In this case, the QDI circuit has the same shortcomings as the RQDI circuit.

3.1.5 RQDI and Other Asynchronous Circuit Families

Mousetrap [24] and GasP [26] are very similar to Micropipelines [25]. Mousetrap is essentially bundled-data with two-phase handshakes. The control logic (timing), generates signals that: i) trigger the internal latching, ii) acknowledge the previous stage, and iii) send a request to the next stage. These types of pipelines add a large amount of latency to the computation due to the timing margin (see Table 3.1). GasP requires transistor-level delay matching to function correctly, and thus uses riskier timing assumptions than RQDI. Single-track [6] circuits use bidirectional wires for their dual data rails to implement acknowledges. This circuit family violates one of our principle circuit guidelines of not sharing dynamic nodes between stages. In addition, it is unclear how to build static multiplexers on bidirectional wires. Static multiplexers are essential in asynchronous FPGAs and exist in nearly every stage. Static multiplexers are trivial to build in RQDI (and QDI).

3.2 Results

3.2.1 Setup

All simulations are done with HSpice using model files for a 65 nm process. Wire capacitances are approximated by adding a 4fF capacitance to each output node. This amount of capacitance is typical of short wires based on our observations of extracted layout in this technology. Gates are sized to have the drive strength of an inverter with its PMOS width set to 20 lambda units and its NMOS width

set to 10 lambda units (lambda is defined as half the minimum gate length). All power and energy numbers are based on total dissipated power.

Table 3.2: Benchmark circuits used in evaluation. Note: these are dual-rail pipelined circuits.

Name	Inputs	Outputs	Description
and2	2	1	and gate
or2	2	1	or gate
xor2	2	1	exclusive or
fa	3	2	full adder
benc	3	2	booth encoder

The benchmark circuits used in our evaluations are listed in Table 3.2. Latency and cycle time numbers reported represent the worst case. We measure the worst case by switching the data rail with the slower stack. For example, the true rail in the and2 circuit has one extra series NMOS transistor, therefore we exercise that stack in its simulations. The area reported is the total transistor area of a circuit (the sum of $width * length$ of each transistor).

3.2.2 HCHB Template

The latency of the five benchmark circuits for the PCHB, PCEHB, and HCHB templates are shown in Figure 3.6. On average, the latency of the PCEHB is 6% less than the other circuit templates. The PCEHB is generally lower latency because R^e and L^e are combined in a separate c-element, rather than in the data rail stacks. The HCHB has a similar latency to the PCHB except and2 and or2 circuits where it's 6% slower. The pull down stacks in these circuits were augmented to wait for input validity, which makes them slower.

Figure 3.7 shows a comparison of the total transistor area across the benchmark circuits. An interesting result is that the PCEHB is slightly smaller than the PCHB. Once again, this is attributed to its simpler data rail transistor stacks. The HCHB is about 15% smaller than the PCEHB template and 20% smaller

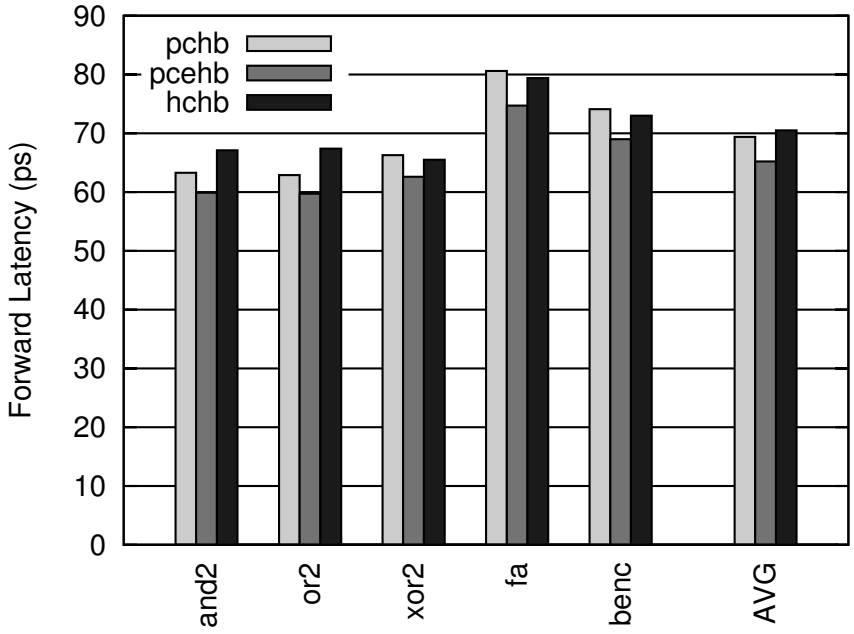

Figure 3.6: Forward latency of benchmark circuits across PCHB, PCEHB, and HCHB templates.

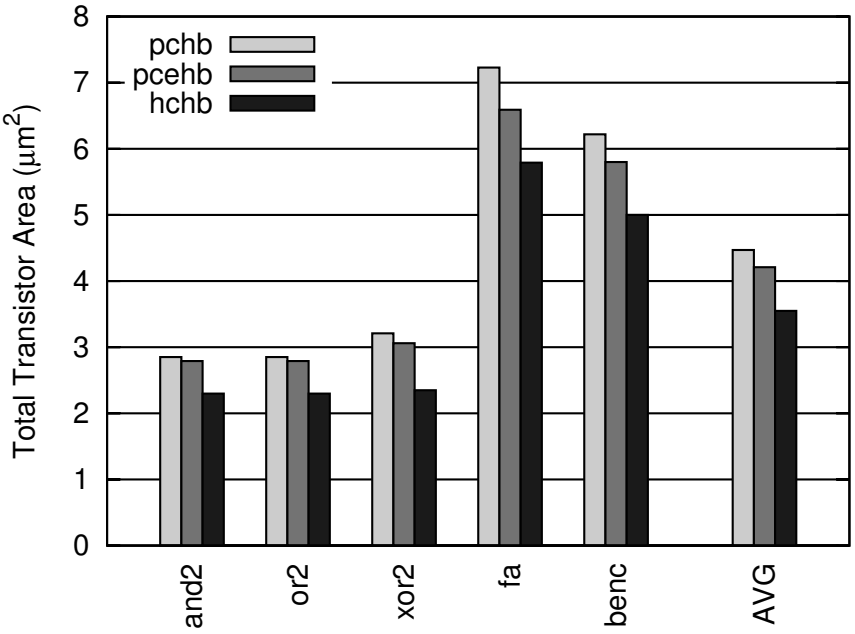

Figure 3.7: Total transistor area of benchmark circuits across PCHB, PCEHB, and HCHB templates.

than the PCHB template on average. This is a result of the simplified detection of input neutrality possible with the half cycle timing assumption.

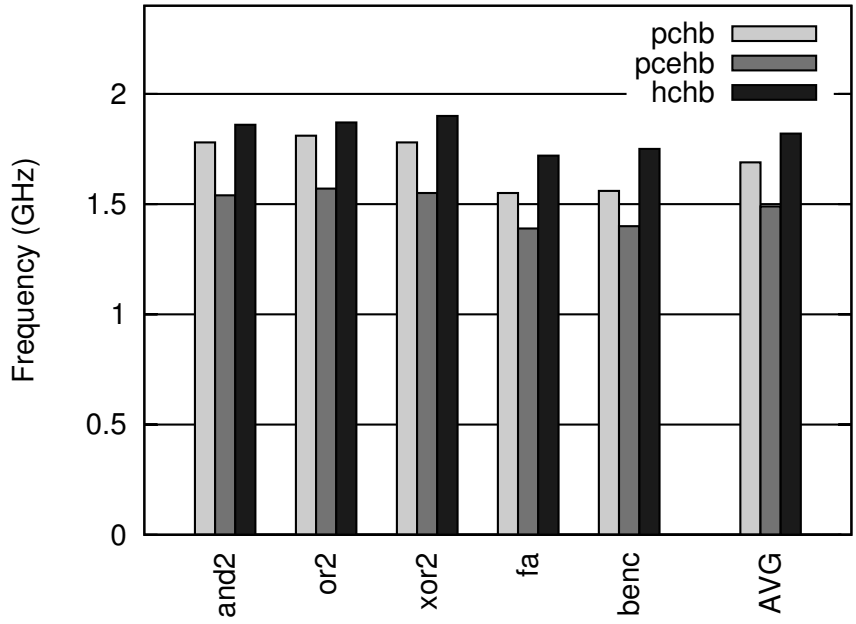

Figure 3.8: Frequency of benchmark circuits across PCHB, PCEHB, and HCHB templates.

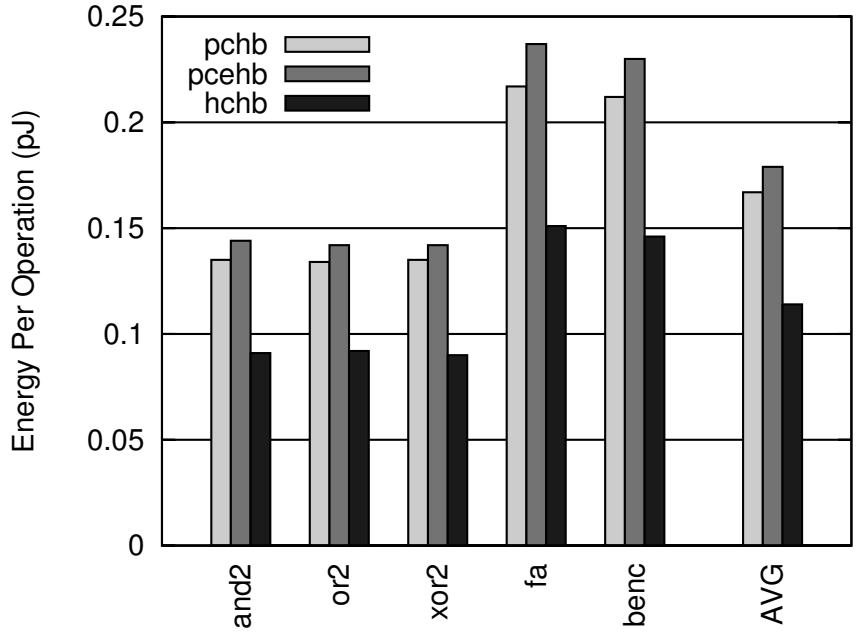

Figure 3.9: Energy per operation of benchmark circuits across PCHB, PCEHB, and HCHB templates.

The HCHB template is consistently higher frequency than the other templates across all five benchmark circuits, as seen in Figure 3.8. On average, the HCHB is

7% higher frequency than the PCHB. The PCEHB has an 18 transition cycle time and the HCHB and PCHB both have a 14 transition cycle time. However, the HCHB is higher frequency because many of its transitions, especially those that detect input neutrality, are simpler. This suggests that HCHB can use even less area because we can use smaller transistors for these fast transitions to match the frequency of the PCHB.

The energy per operation (or per cycle) of the benchmark circuits is reported in Figure 3.9. The HCHB template consistently uses less energy than the PCHB and PCEHB templates across all five benchmarks. The HCHB template consumes 32% and 36% less energy on average than the PCHB and PCEHB templates respectively. This is a great result because it is accompanied by significant area savings, a slight frequency improvement, and a negligible latency penalty.

CHAPTER 4

RQDI TWO-PHASE CIRCUITS

Two phase handshake protocols are often suggested to reduce power and increase frequency in asynchronous circuits. The main difficulty with two-phase protocols is that they are very inefficient in performing logic functions, as has been noted by others [21]. However, a simple two-phase buffer with similar characteristics to a WCHB (two transition forward latency and ten or less transition cycle time) would be useful in two specific applications. The first, and most obvious, is global communication. The second application is static switching networks.

4.1 Two Phase Logic

4.1.1 Two Phase Protocol

There are two basic protocols used for two-phase handshakes. We will refer to the first protocol as the rail transition (RT) protocol, as shown on the left of Figure 4.1. With the RT protocol, you simply transition the rail that you want to send data. For example, we will send a '1' from state '00'. Both the true rail (the first digit) and the false rail (the second digit) are logic low in state '00'. To send a '1', we set the true rail high, which makes the current state '10'. To send another '1', we set the true rail low and the current state returns to '00'.

The second basic protocol is the level-encoded dual-rail (LEDR) protocol [2], as shown on the right side of Figure 4.1. Rather than two data rails, the LEDR protocol encodes a data signal and a repeat signal. The data rail is always set to the value of the current token. The repeat rail is toggled when the current token is the same value as the previous token. Unlike the RT protocol, the value of the most recently sent token can be inferred from the current state. For example, the

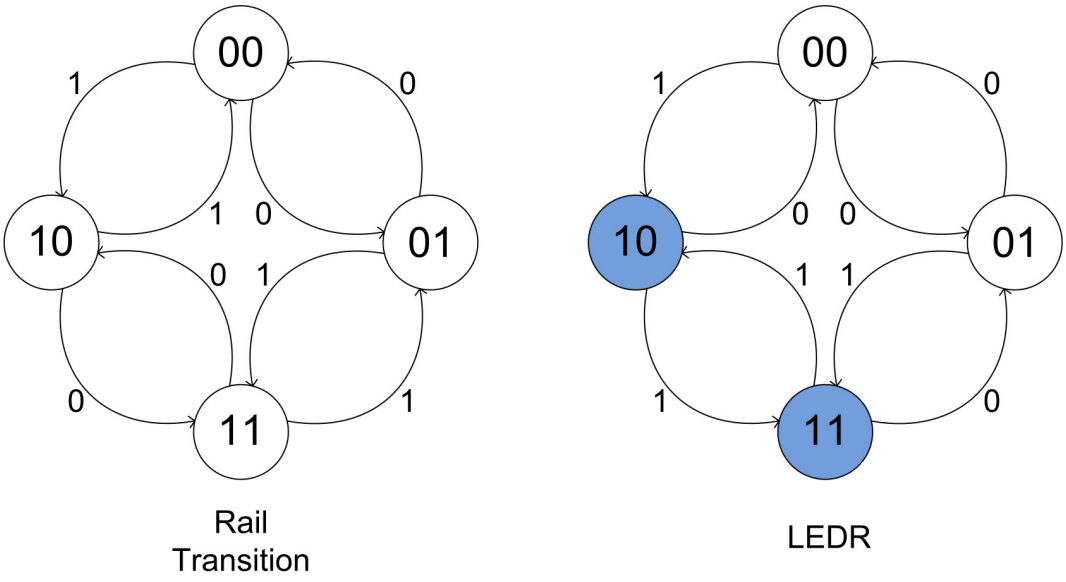

Figure 4.1: The rail transition and LEDR two-phase protocols.

two shaded states, '10' and '11', have most recently sent a '1'.

Which protocol is superior? The buffer that we will present in Section 4.1.3 works for both protocols. Single bit protocol converters are roughly the same size for both protocols. However, multi-bit conversions are cheaper with LEDR because the current state can be determined without examining the previous state. In addition, readily knowing the current token value makes debugging two-phase circuits easier. For these reason, we will focus on LEDR circuits in this thesis. We have explored RT based two-phase circuits in previous work [14].

4.1.2 XOR Gates

XOR gates are used extensively in two-phase logic. They are necessary to detect transitions in encodings that do not return to zero (reset). XOR gates are not valid QDI circuits due to the use of both non-inverted and inverted version of inputs. However, XOR gates are proper RQDI circuits. Under RQDI timing assumptions, XOR gates do not create short circuit paths or become state holding. Without

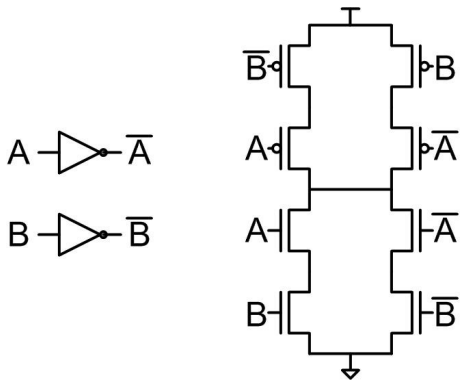

Figure 4.2: A valid RQDI XOR gate.

this, two-phase circuits would be infeasible.

Figure 4.3 shows the two representative sets of inputs for the XOR gate. It is essential that for each state the circuit is never state holding and always non-interfering. The former requires that there is a path to V_{dd} or GND and the latter requires that there is never a path to both V_{dd} and GND. In the figure, we step through each transition of the circuit. There is a period of time where each input and its inverted version are the same value because of the delays through the inverters. In spite of this, there is always exactly one path to V_{dd} or GND in every state. In the bottom case, the output of the gate may change before the output of an inverter does. However, we can assume that the inverter output will be ready before the next set of inputs arrive due to the HCTA (in some cases the assumption is on a full cycle).

4.1.3 Buffer

A simple dataless two-phase buffer can be represented by the following HSE:

$$*[[L = R^e]; R := L; L^e := \neg R]$$

The resulting circuit is a c-element and an inverter, shown in Figure 4.4. The circuit becomes more complicated when you incorporate data. The problem is

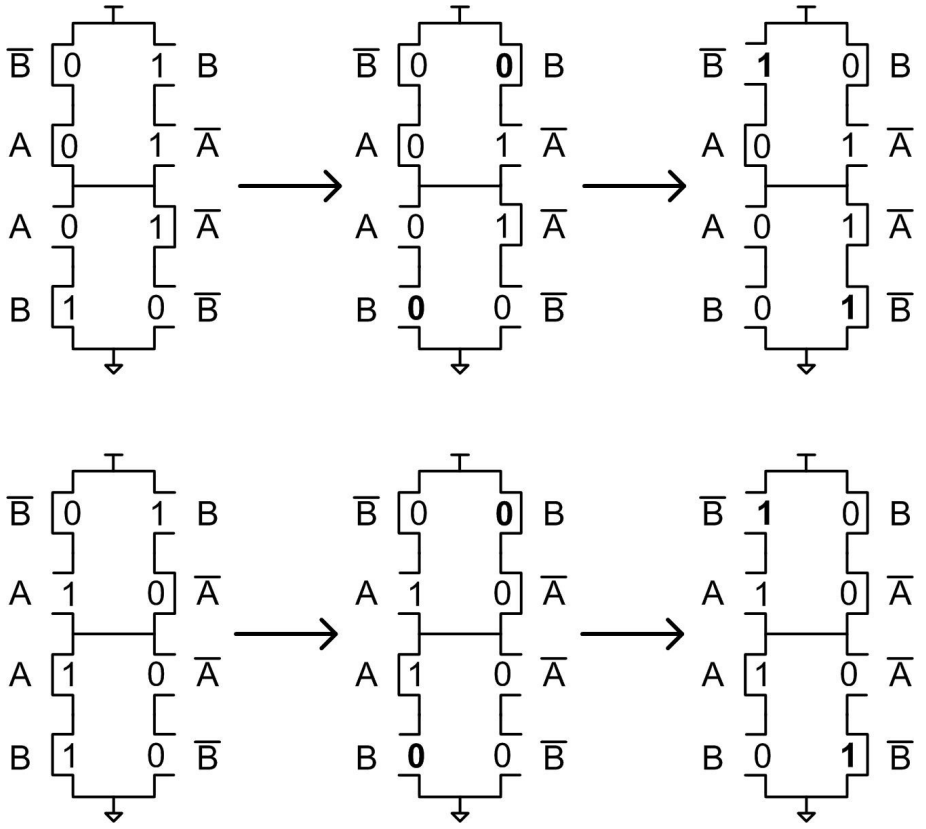

Figure 4.3: Two traces of an XOR gate for two sets of inputs. There is always exactly one path to V_{dd} or GND.

that R^e changes each cycle and each data rail needs to know which sense of R^e to wait for. We can introduce a state variable to track this, but it would be expensive to manage it. A better solution is for each rail to wait for the XOR of the opposite rail with R^e, instead of R^e alone. The idea is that if the opposite rail caused R^e to change then the output of the XOR will be unchanged since both of its inputs have switched (in reality the output will switch and then switch back). The HSE for the two-phase data buffer is:

$$
\begin{aligned}
*[[&L^f = XOR(R^t, R^e) \longrightarrow R^f := L^f \\
\,[]\,&L^t = XOR(R^f, R^e) \longrightarrow R^t := L^t] \\
&L^e := XOR(R^f, R^t)]
\end{aligned}
$$

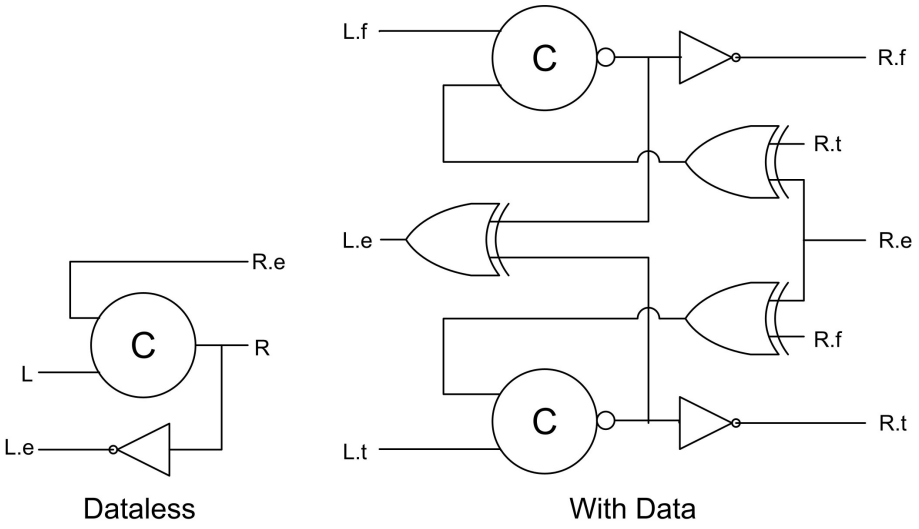

Figure 4.4: The HC2PFB buffer: dataless (left) and with data (right).

The HC2PFB (half cycle two-phase full-buffer) buffer is shown in Figure 4.4. Note, the HC2PFB works with both the LEDR protocol and RT protocol. The HC2PFB has a forward latency of two transitions and a cycle time of seven transitions. The pair of XOR gates that process R^e can be folded into the c-elements, but this makes the data rails more complex and increases the latency. The HC2PFB is 45% larger than the WCHB, however, since the HC2PFB has such a short cycle time the XOR gates can be undersized to reduce the area penalty. In addition, each HC2PFB can replace two stages of WCHBs because it can support twice the number of transitions in a cycle. The slack will remain the same because we are replacing two half-buffers with a full-buffer. When used in this fashion, the HC2PFB equivalent circuit is 15% smaller than the WCHB. This is an important result because the four-phase to two-phase and two-phase to four-phase converters are significantly larger than the buffer. (In Chapter 6, we use undersizing, solely, to reduce the area. The resulting additional slack improves performance in FPGAs.)

4.2 Converters

4.2.1 4:2 Converter

Consider the following HSE for the four-phase to two-phase converter:

$$
\begin{aligned}
*[[&L^f \wedge \neg en \longrightarrow R^d\downarrow, R^r\uparrow [] L^f \wedge en \longrightarrow R^d\downarrow, R^r\downarrow \\
&[] L^t \wedge \neg en \longrightarrow R^d\uparrow, R^r\downarrow [] L^t \wedge en \longrightarrow R^d\uparrow, R^r\uparrow]; L^e\downarrow; \\
&[\neg L^f \wedge \neg L^t]; L^e\uparrow; [R^e = en]; en := \neg R^e]
\end{aligned}
$$

The input channel, L, is a standard dual rail channel composed of a true rail, false rail, and an enable signal. The output channel, R, is an LEDR based channel composed of a data rail, repeat rail, and an enable signal. One of the key design requirements of both converters is that every signal be generated as early as possible. The reasoning behind this is that if the two-phase signals are slow, than the adjacent two-phase stage may be limited to a simple buffer with no switching logic. Insidiously, this results in increased area and latency beyond the converter itself. As a result of producing each signal as early as possible, a local enable variable, en, is needed to track the current parity of the circuit. Interestingly, it appears that both rails in R fire each cycle, but one of these firings is vacuous.

The four-phase to two-phase converter is shown in Figure 4.5. The physical implementation differs slightly from the above HSE. The en variable is implemented as a dual rail variable to avoid using back-to-back latches in the converter. Back-to-back latches introduce some non-RQDI timing constraints. The latency of the converter is three transitions in the worst case because of the need to invert L^f and L^t. The four-phase to two-phase converter is about $3x$ larger than a WCHB.

4.2.2 2:4 Converter

The following is the HSE for the two-phase to four-phase converter:

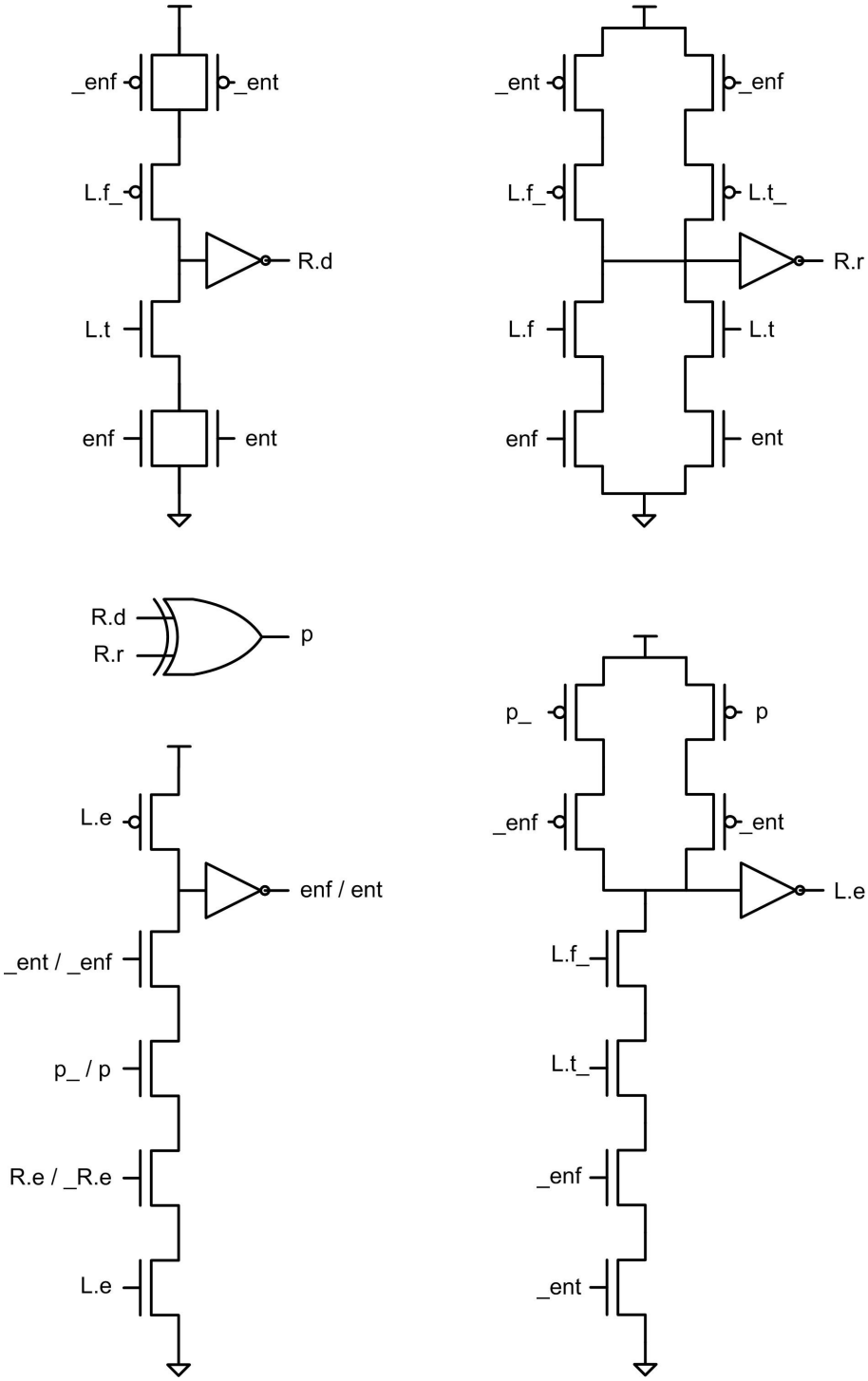

Figure 4.5: A four-phase to two-phase converter.

$$*[[\neg L^d \wedge L^r = en \longrightarrow R^f\uparrow [] L^d \wedge L^r \neq en \longrightarrow R^t\uparrow]; L^e := \neg en;$$
$$[\neg R^e]; R^f\downarrow, R^t\downarrow; [R^e]; en := L^e]$$

The two-phase to four-phase converter takes an LEDR input, L, and produces a dual rail output, R. The converter is shown in Figure 4.6. Similar to the four to two-phase converter, the enable signal is implemented as a dual rail variable to avoid the use of back-to-back latches. This converter also has a forward latency of three transitions due to inverting the input data rails. It is roughly $3.25x$ larger than the WCHB circuit.

4.3 Static Switching Networks

Static switching networks are especially important in FPGAs [27]. Logic clusters in FPGAs are surrounded by statically configured switching networks. In asynchronous FPGAs, these statically configured switching networks are built up out of the switches shown in Figure 4.7. Programming bits are set to select which set of input data rails are the input to the WCHB via the MUX. The output data rails of the WCHB fanout to other switches and the associated enables/acknowledges must be combined via a programmable c-element (depicted as pc in the diagram). We can replace the WCHB with a two-phase buffer and the switch will work without any further modification.

More than half the area of an asynchronous FPGA is devoted to routing. Two phase routing is attractive because it can reduce the switching in this area by a factor of two. In addition, there is potentially enough stages of routing to amortize the area and latency overhead associated with converting between protocols. It is important to note that there will not be frequency advantage when using two-phase circuits in this manner. The frequency is limited by the four-phase circuits

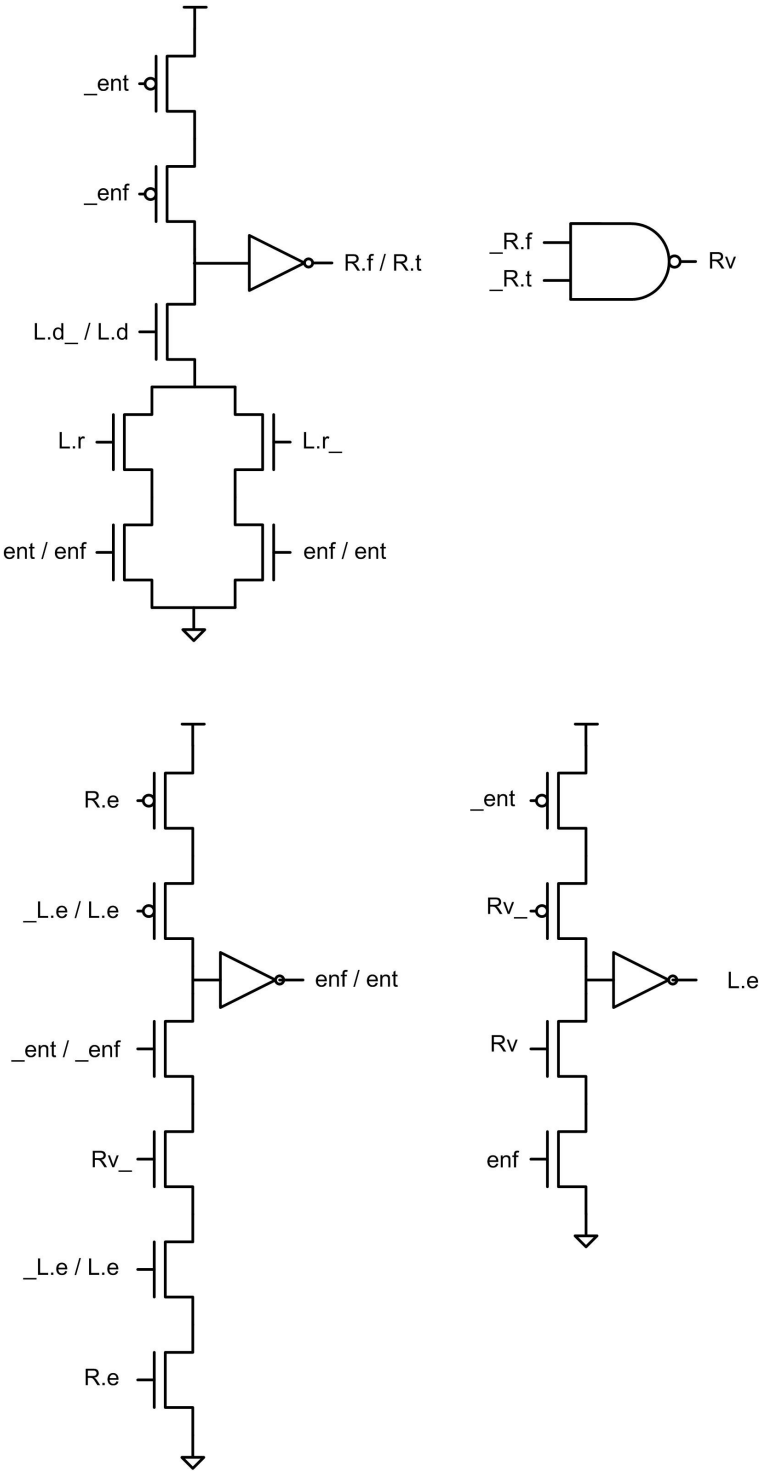

Figure 4.6: A two-phase to four-phase converter.

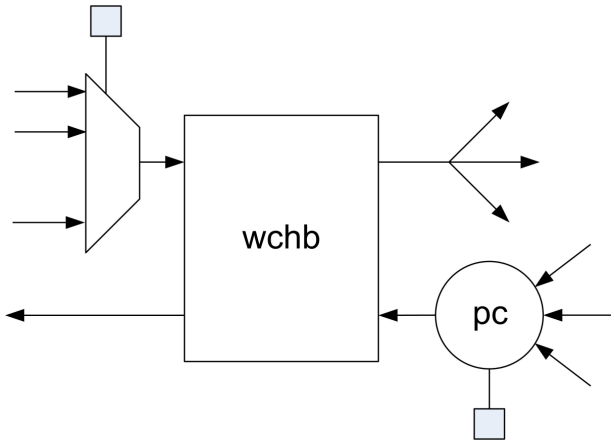

Figure 4.7: A statically programmed n-way switch.

performing the logic functions.

4.4 Results

4.4.1 Setup

In asynchronous FPGAs, programmable routing between logic clusters is made up of stages of the static switch (Figure 4.7). In these experiments, we make this routing use two-phase logic by replacing the WCHB with an HC2PFB. In fact, we replace two stages of the WCHB switch with one stage of the HC2PFB switch. This keeps the slack, latency, area, and cycle time roughly constant between the two implementations. There is some overhead incurred from the 4:2 and 2:4 phase converters at the input and output of the routing logic. We vary the number of two-phase pipeline stages between these converters and measure the area impact and energy reduction over the original WCHB version.

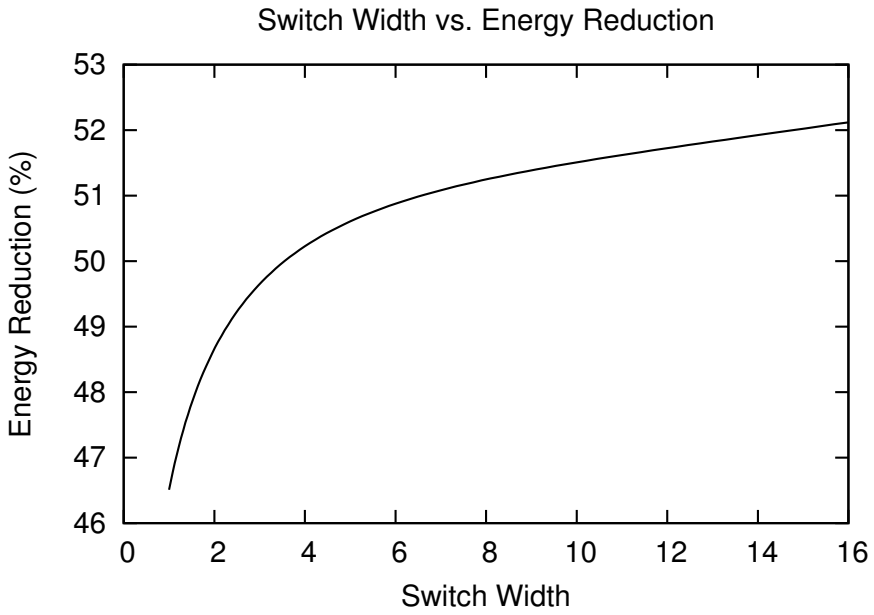

Figure 4.8: Energy reduction as switch width increases.

4.4.2 Two Phase Static Switching

Figure 4.8 shows the energy reduction in the two-phase static switch with increasing switch width. As the switch width increases, more static muxing and programmable c-elements are need to build the switch. As a result, more capacitance is switching each cycle. Intuitively, one would think that the maximum energy reduction would be 50%. However, each two-phase buffer replaces two four-phase buffers. Even in this configuration, the two-phase buffer is higher frequency. At a switch width of 16 there is over a 52% reduction in energy.

The main drawback to using the two-phase switch is the high cost of converting between two-phase and four-phase protocols. The two-phase buffer is about 15% smaller than the two four-phase buffers it replaces. The four-phase to two-phase converter is $3x$ larger than a WCHB and the two-phase to four-phase converter is $3.25x$ larger than a WCHB. Figure 4.9 tracks the area overhead of 20 stages of switches with varying widths. In an asynchronous FPGA, neighboring logic

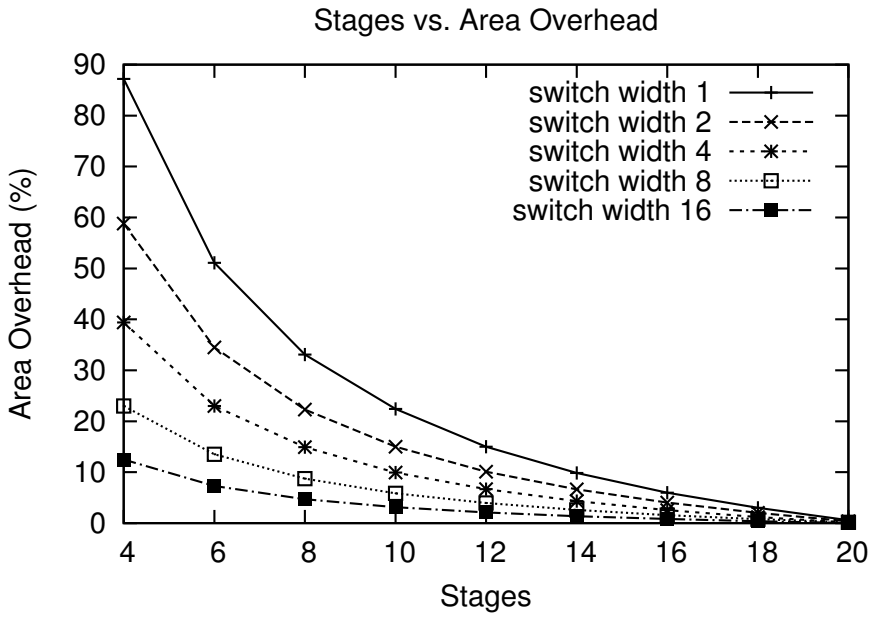

Figure 4.9: Area overhead as the number of stages increase.

clusters are typically separated by about 6 stages of 8-wide switches. This would put the area overhead at roughly 15%. However, we will see in Chapter 6 that only a fraction of the signals in one section of routing need access to the four-phase logic making the overhead much less than 15%.

CHAPTER 5

RQDI VOLTAGE SCALING

5.1 Voltage Scaling

A high level discussion on voltage scaling in asynchronous architectures can be found in [16]. Energy efficient pipelines are discussed in [28]. In this chapter we aim to facilitate voltage scaling in asynchronous circuits by: i)introducing a pair of efficient voltage converters, and ii) proposing a dual voltage circuit template that has constant latency. We focus on voltage scaling in pipelines where the throughput is limited by the architecture. Specifically, we consider token limited loops and reconvergent paths. The goal is to scale voltage in places where there is minimal impact on performance.

Typically, the equation for dynamic power consumption is given as follows:

$$P_{dynamic} = C{V_{dd}}^2 F \tag{5.1}$$

In this equation, C is the load capacitance, V_{dd} is the supply voltage, and F is the operating frequency of the circuit. We can simplify the relationship between V_{dd} and dynamic power via the following:

$$F \propto V_{dd} \tag{5.2}$$

$$P_{dynamic} \propto {V_{dd}}^3 \tag{5.3}$$

The time it takes to charge a capacitor is proportional to $1/V_{dd}$. Therefore, the frequency is proportional to V_{dd}. The resulting operating frequency for V_{dd} scaling in a 65 nm process for a typical asynchronous circuit is shown in Figure 5.1. We avoid scaling past .5 V (roughly twice the threshold voltage) in this technology because it results in extremely slow circuits. In addition, the power reduction

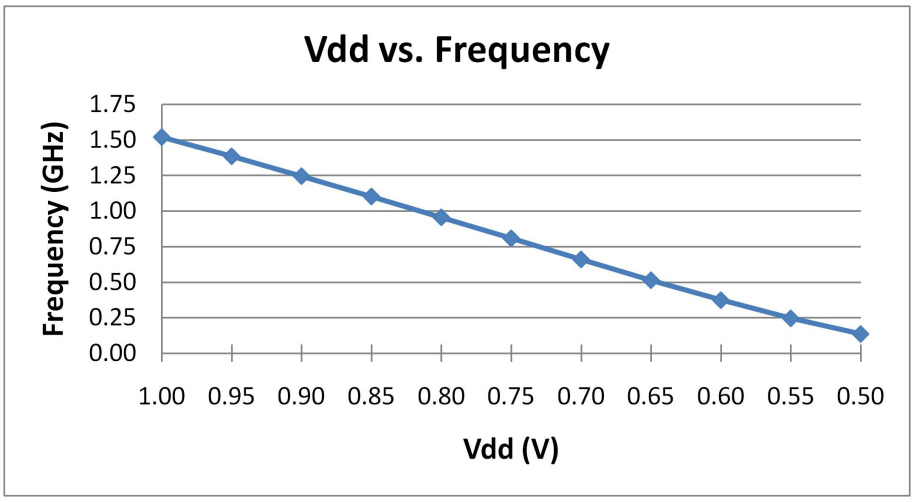

Figure 5.1: The operating frequency of a typical asynchronous circuit in a 65 nm process as its supply voltage is scaled.

at .5 V is already greater than 90 %. Due to the linear relationship of V_{dd} and frequency, dynamic power scales proportionally to ${V_{dd}}^3$.

When circuits are operating at peak throughput, a cubic reduction in power is possible at the cost of a linear reduction in frequency. However, when circuits are not operating at peak throughput, it is possible to reduce power in certain portions of the circuit with negligible impact on frequency. In these cases, the frequency has already been reduced due to some limitation in the architecture and as a result there is a linear reduction in power. That leaves a possible ${V_{dd}}^2$ reduction in power when reducing the supply voltage to match the already limited frequency. Figure 5.2 shows this additional reduction in power from voltage scaling for a typical asynchronous circuit in a 65 nm technology. Two common throughput limiting structures we will examine are token limited loops and reconvergent paths.

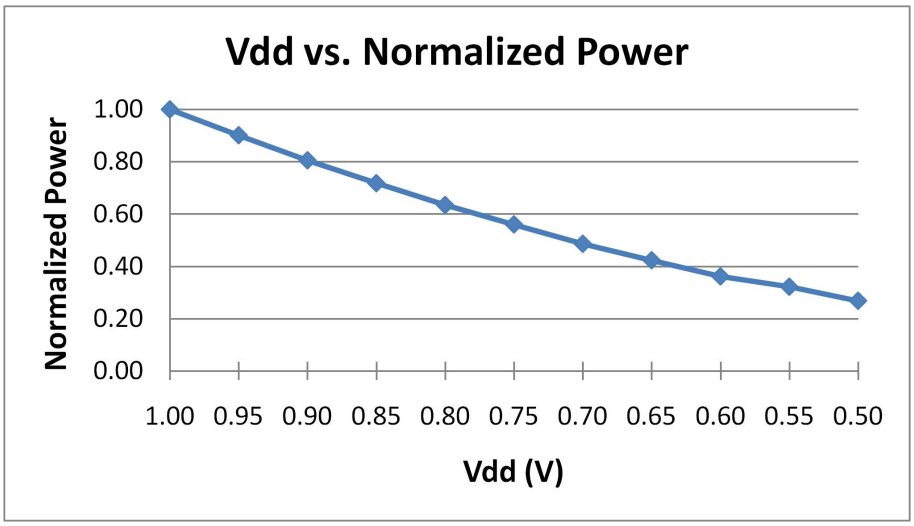

Figure 5.2: Normalized power reduction resulting from V_{dd} scaling in a typical asynchronous circuit in a 65 nm technology. Note, that this power reduction is in addition to the power already saved from operating at a reduced frequency.

5.2 Voltage Scaling and Throughput

The throughput of a pipeline, γ, is often defined as a function of slack per token, σ, with dynamic slack, r, and peak throughput, T [15]:

$$\gamma(\sigma) = \begin{cases} \frac{T}{\sigma r} & \sigma r \geq 1 \\ \frac{T(\sigma - 1)}{\sigma(1 - r)} & \sigma r \leq 1 \end{cases}$$

The $\sigma r \geq 1$ case occurs when the pipeline is token-limited. In other words, there aren't enough tokens to keep the pipeline running at full throughput. The throughput of this type of pipeline is limited by the total forward latency divided by the number of tokens. The $\sigma r \leq 1$ case occurs when the pipeline is hole-limited (a hole is a buffer absent a token). The throughput of this type of pipeline is limited by the total backward latency divided by the number of holes.

We consider pipelines composed of one of two types of buffers. For logic functions, we consider pipelines made from four-phase half-buffers. For routing, we consider pipelines made from two-phase full-buffers (all two-phase buffers are full-buffers). The throughput of four-phase(4h) and two-phase(2f) logic is computed

as follows:

$$4h\colon T = \min\left(\frac{k}{nl_f}, \frac{n-2k}{2nl_b}\right) \qquad 2f\colon T = \min\left(\frac{k}{nl_f}, \frac{n-k}{nl_b}\right) \tag{5.4}$$

In the above, k is the number of tokens, n is the number of pipeline stages, l_f is the forward latency, and l_b is the backward latency. Note that the token-limited case is the same in both full-buffers and half-buffers.

We envision two possible ways to scale voltage in asynchronous circuits. The first method is to simply scale V_{dd}. This increases both the forward and backward latencies of each stage in a pipeline. The second, and far more interesting, method is to scale the voltage of the enable (acknowledge) signals and their associated logic. This method keeps the forward latency fixed, but increases the backward latency of each stage.

Figure 5.3 shows the impact on throughput when applying each method of voltage scaling in a ten-stage half-buffer pipeline. The outermost triangle (solid line) is the throughput of the pipeline without any scaling (the left leg corresponds to the token-limited domain and the right leg corresponds to the hole-limited domain). The innermost triangle formed by the dotted and dashed lines represents the throughput when V_{dd} is reduced by 40 %. All points off of the base of the triangle have worse throughput than the nominal V_{dd} triangle. The triangle with a dashed right leg and solid left leg corresponds to the throughput of the pipeline when the enables are scaled by 40 %. Much of the left leg of this triangle is shared with the nominal V_{dd} case. This makes enable scaling ideal in cases where the pipeline is always token-limited.

5.2.1 Loops

Loops are ubiquitous in any reasonably complex design (an example of a loop is shown in Figure 2.5). The key to running loops at peak throughput is to have

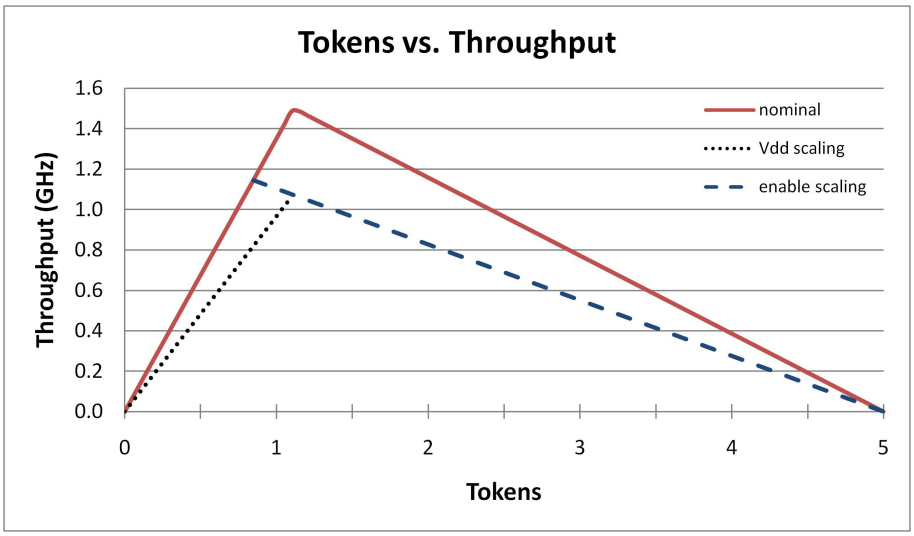

Figure 5.3: Impact of V_{dd} and enable scaling on throughput of a ten-stage half-buffer pipeline.

just enough tokens in the loop so that no stage is ever waiting for a token. This happens when $n = k(2l_f + 2l_b)/l_f$ for four-phase buffers and $n = k(l_f + l_b)/l_f$ for two-phase buffers. Loops are usually token-limited, rather than hole-limited. This is because adding initial tokens to a loop changes its meaning, while adding buffers does not. As a result, loops will often operate on the left leg of the solid triangle in Figure 5.3. This is an ideal case for enable scaling because the enable voltage can be reduced to some degree without any impact on the throughput.

It is not immediately clear if it is better to scale the voltage across the entire ring or to only scale the voltage for the enables. The enables can be scaled with little impact to throughput, but there is also less power to be saved using this technique. A common metric used to compare the energy efficiency of two circuits is $E\tau^2$, where E is energy and τ is the cycle time of the circuit. The goal is to minimize this value. $E\tau^2$ is attractive because to the first order it is independent of operating voltage because $E \propto {V_{dd}}^2$ and $\tau^2 \propto 1/{V_{dd}}^2$. Figure 5.4 compares $E\tau^2$ for V_{dd} scaling and enable scaling in a 14-stage ring with a single token. By this metric, enable scaling is clearly superior because it has a minima at .75 V while

V_{dd} scaling rises monotonically.

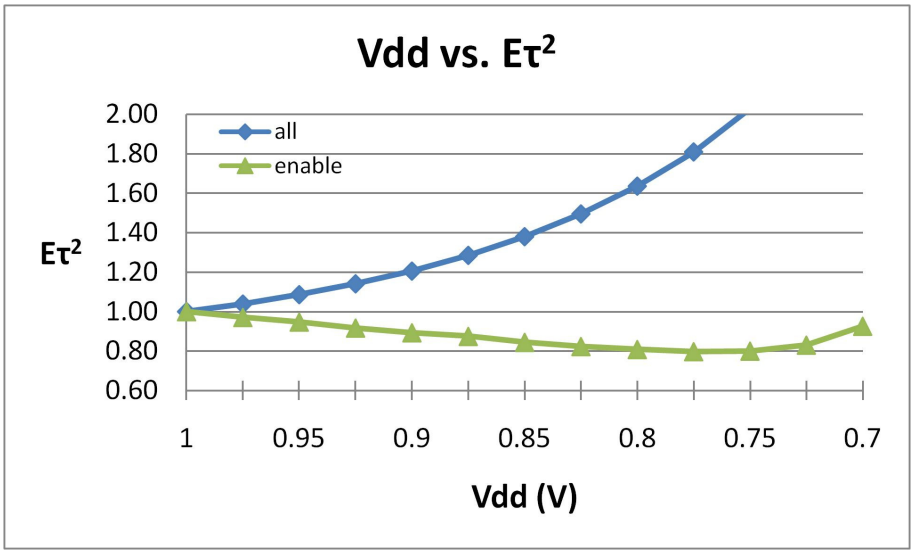

Figure 5.4: $E\tau^2$ for a 14-stage ring with a single token.

In order to apply enable scaling to loops, we need to choose the appropriate voltage for the enable signal that saves the most power without reducing throughput. This is achieved through a two-step process. The first step is to find the largest backward latency that will not impact the throughput of a token-limited loop. This occurs when the hole-limited domain intersects the loop's operating point on the token-limited domain. For four-phase half-buffers, we solve the following for $l_b{}'$, the increased backward latency:

$$\frac{k}{nl_f} = \frac{n-2k}{2nl_b{}'}$$

The $l_b{}'$ for four-phase(4h) and two-phase(2f) logic is as follows:

$$4h: l_b{}' = \frac{l_f(n-2k)}{2k} \quad 2f: l_b{}' = \frac{l_f(n-k)}{k} \tag{5.5}$$

The second step is to characterize the relationship between enable scaling and the backward latency of a specific buffer stage through analog circuit simulation (as is done in Section 5.4.1). The optimal value of $l_b{}'$ is then cross-referenced against these results to select the best voltage for the enable signal.

5.2.2 Reconvergent Paths

Reconvergent paths are another common structure found in asynchronous architectures. This type of structure is formed whenever a copy is made of a token and both the original and the copy are later used together in some computation. The composition of pipelines may run at full throughput if the amount of slack (buffering) on each pipeline is matched (an example of slack matching is shown in Figure 2.6). One of the main goals of an asynchronous designer is to make sure that slack is matched across parallel pipelines in a reconvergent path. However, this is a difficult goal to achieve in reconfigurable architectures where the length and composition of each pipeline are not known at design-time. This is a wide-spread problem in FPGAs, as we will see in the next chapter.

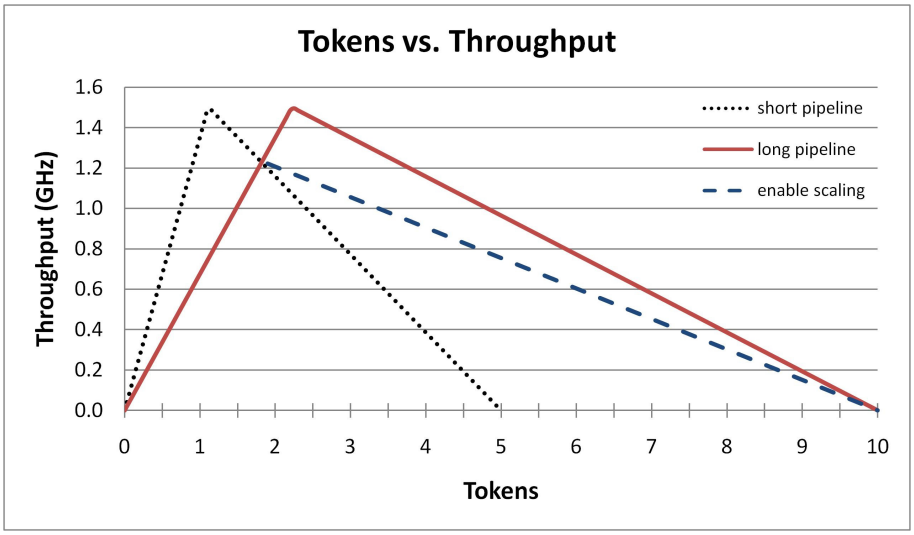

Figure 5.5: The throughput of the composition of short, ten-stage, and long, 20-stage, half-buffer pipelines. Enable scaling is shown for the long pipeline.

The throughput of a reconvergent path is the minimum of each pipeline's throughput [28]:

$$\gamma_{\|}(\sigma_a, \sigma_b) \quad = \quad \min(\gamma_a(\sigma_a), \gamma_b(\sigma_b))$$

The throughput of short, ten-stage, and long, 20-stage, half-buffer pipelines are

shown in Figure 5.5. The throughput of the composition of these pipelines is the overlapping triangle with a solid line on its left leg and a dotted line on its right leg. In the steady-state, the number of tokens in this structure is determined by the intersection of the token-limited domain of the long pipeline and the hole-limited domain of the short pipeline. Solving for the number of tokens, k, in four-phase and two-phase buffer pipelines results in the following:

$$4h\colon k = \frac{n_l n_s l_f}{2(n_s l_b + n_l l_f)} \qquad 2f\colon k = \frac{n_l n_s l_f}{n_s l_b + n_l l_f} \tag{5.6}$$

The variables n_l and n_s represent the number of stages in the long and short pipelines, respectively. Substituting these values back into Equation 5.4 results in the following throughput equations for half-buffers and full-buffers:

$$4h\colon T = \frac{n_s}{2(n_s l_b + n_l l_f)} \qquad 2f\colon T = \frac{n_s}{n_s l_b + n_l l_f} \tag{5.7}$$

Based on Figure 5.5, reconvergent paths are another ideal case for enable scaling. However, we would like to scale the enable on the long path only. Scaling the enable on both paths would hurt the throughput because it drops the intersection point on the left leg (solid line) of the throughput triangle. At the circuit level, this can be implemented using a virtual V_{dd}. (The overhead of this is minimal in programmable routing because there are already numerous programming bits per stage and this only adds one more.) As with rings, it is not clear if enable scaling the long path is the best method. Figure 5.6 compares $E\tau^2$ for three scaling schemes: i) V_{dd} scaling on both paths, ii) V_{dd} scaling in the long path, and iii) enable scaling in the long path. Once again, scaling V_{dd} everywhere has the worst $E\tau^2$ and rises monotonically. Scaling V_{dd} in the long path is somewhat better with an optimal voltage of .95 V. Enable scaling the long path yields the best $E\tau^2$ with an optimal voltage of .8 V.

As we did with loops, we would like to find the largest backward latency (for the long pipeline) that will not reduce throughput. For reconvergent paths, this occurs

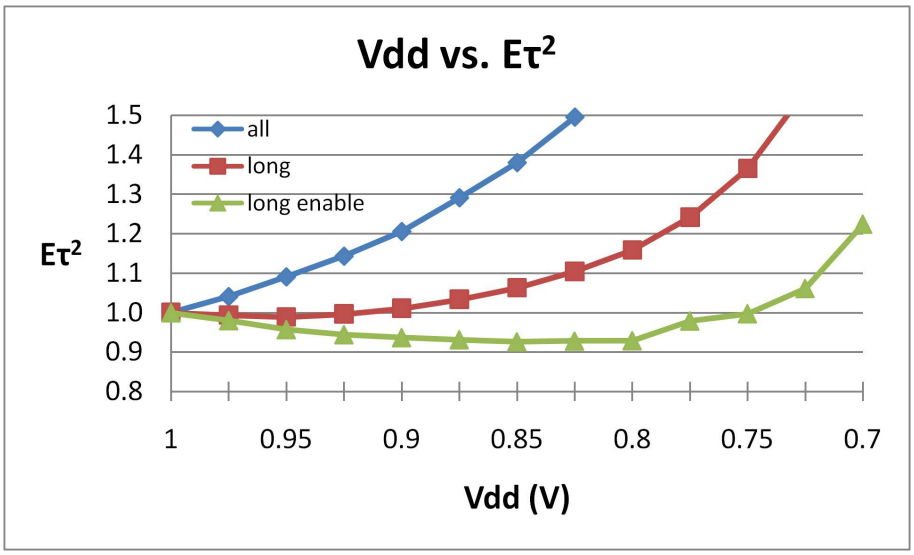

Figure 5.6: $E\tau^2$ for various voltage scaling schemes for a two-stage short pipeline and a ten-stage long pipeline.

when the hole-limited domain of the long pipeline intersects with the throughput. For four-phase half-buffer pipelines, we solve the following for $l_b{}'$:

$$T = \frac{n_l - 2k}{2n_l l_b{}'}$$

Substituting k with Equation 5.6 and T with Equation 5.7, yields the following equations for $l_b{}'$:

$$4h : l_b{}' = \frac{n_l l_f}{n_s} + l_b - l_f \qquad 2f : l_b{}' = \frac{n_l l_f}{n_s} + l_b - l_f \tag{5.8}$$

5.2.3 Reconvergent Paths with Initial Tokens

When there aren't any initial tokens, parallel pipelines in a reconvergent path each contain an equal number of tokens. Tokens enter/exit each pipeline simultaneously. However, if one of the pipelines is initialized with k_0 initial tokens, then it will always contain k_0 more tokens than the other pipeline. Initializing both pipelines with the same number of tokens is equivalent to not having any initial tokens. Approaching the steady-state, tokens will be added or removed from the pipelines

until they contain an optimal number of tokens. Therefore, we are only concerned with the difference between of the number of initial tokens in each pipeline. We define k_0 as difference between the number of initial tokens in each pipeline. We define k as the number of *shared tokens* between each pipeline (the number of tokens in the pipeline with fewer initial tokens). P_f and P_m are the pipelines with fewer initial tokens and more initial tokens, respectively. The number of stages in P_f and P_m are n_f and n_m.

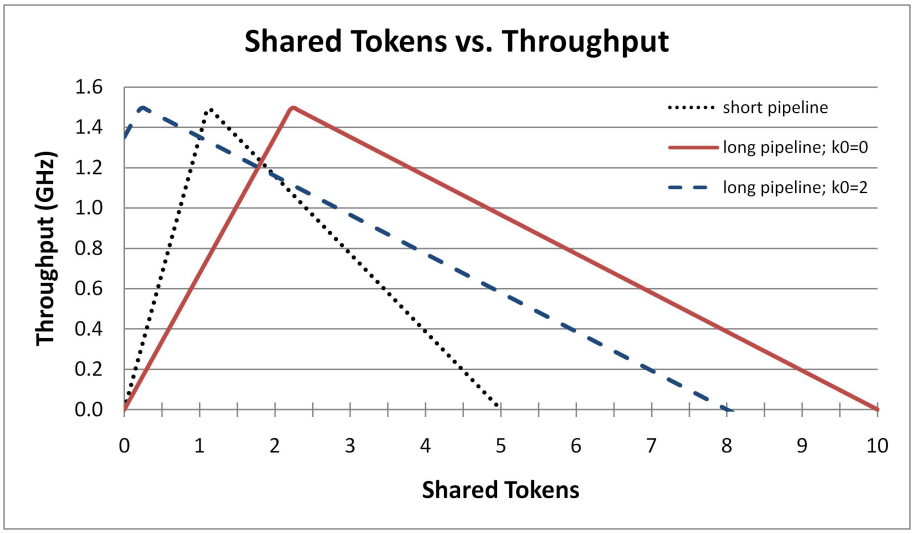

Figure 5.7: The throughput of the composition of short, ten-stage, and long, 20-stage, half-buffer pipelines. The throughput plot for the long pipeline is shifted left when two initial tokens are added (dashed line).

In relation to P_f, the throughput curve of P_m is shifted left as k_0 increases. The throughput for the pipeline with more initial tokens is:

$$\begin{aligned} 4h: T_m &= \min\left(\frac{k+k_0}{n_m l_f}, \frac{n_m - 2(k+k_0)}{2n_m l_b}\right) \\ 2f: T_m &= \min\left(\frac{k+k_0}{n_m l_f}, \frac{n_m - (k+k_0)}{n_m l_b}\right) \end{aligned} \tag{5.9}$$

In Figure 5.7, the longer pipeline is P_m. At $k_0 = 0$ the peak of long pipeline is to the right of the peak of the short pipeline and at $k_0 = 2$ the peak of the long pipeline is to the left of the peak of the short pipeline. This is significant because

the ordering of their peaks changes which legs of each throughput curve intersect to form the composite throughput curve. When $k_0 = 0$, we can compare n_f and n_m directly to determine which has an earlier peak. However, at $k_0 \neq 0$ the position of the peak corresponds to a pipeline with $n_m{}' = n_m - k_0\tau/l_f$ stages. Therefore, we compare n_f with $n_m{}'$ to determine the relative ordering of the peaks.

When $n_f \leq n_m{}'$ the throughput is limited by the intersection of the hole-limited domain of P_f and the token-limited domain of P_m. When $n_f \geq n_m{}'$ the throughput is limited by the intersection of the hole-limited domain of P_m and the token-limited domain of P_f. We solve for k at the intersection for each case:

$$\begin{aligned} 4h\colon k &= \begin{cases} \dfrac{n_f(n_m l_f - 2k_0 l_b)}{2(n_f l_b + n_m l_f)} & n_f \leq n_m{}' \\ \dfrac{n_f l_f(n_m - 2k_0)}{2(n_m l_b + n_f l_f)} & n_f \geq n_m{}' \end{cases} \\ 2f\colon k &= \begin{cases} \dfrac{n_f(n_m l_f - k_0 l_b)}{n_f l_b + n_m l_f} & n_f \leq n_m{}' \\ \dfrac{n_f l_f(n_m - k_0)}{n_m l_b + n_f l_f} & n_f \geq n_m{}' \end{cases} \end{aligned} \tag{5.10}$$

Substituting the above for k in Equation 5.9 yields the following equations for throughput:

$$\begin{aligned} 4h\colon T &= \begin{cases} \dfrac{n_f + 2k_0}{2(n_f l_b + n_m l_f)} & n_f \leq n_m{}' \\ \dfrac{n_m - 2k_0}{2(n_m l_b + n_f l_f)} & n_f \geq n_m{}' \end{cases} \\ 2f\colon T &= \begin{cases} \dfrac{n_f + k_0}{n_f l_b + n_m l_f} & n_f \leq n_m{}' \\ \dfrac{n_m - k_0}{n_m l_b + n_f l_f} & n_f \geq n_m{}' \end{cases} \end{aligned} \tag{5.11}$$

As we did in the previous subsection, we need to find the target $l_b{}'$ for enable scaling. When $n_f \leq n_m{}'$ we intersect the hole-limited domain of n_m with point (k, T) on the throughput graph. When $n_f \geq n_m{}'$ we intersect the hole-limited domain of n_f with point (k, T) on the throughput graph. This yields the following:

$$4h\colon l_b{}' = \begin{cases} \dfrac{n_f l_b + n_m l_f}{(n_f + 2k_0)} - l_f & n_f \leq n_m{}' \\ \dfrac{n_m l_b + n_f l_f}{(n_m - 2k_0)} - l_f & n_f \geq n_m{}' \end{cases}$$

$$2f: l_b{}' \;=\; \begin{cases} \dfrac{n_f l_b + n_m l_f}{(n_f + k_0)} - l_f & n_f \le n_m{}' \\ \dfrac{n_m l_b + n_f l_f}{(n_m - k_0)} - l_f & n_f \ge n_m{}' \end{cases} \tag{5.12}$$

5.3 Voltage Converters

In this section, we describe the RQDI circuits that are necessary to convert between voltage domains and scale enable signals.

5.3.1 Standard Voltage Converter

The standard low to high voltage converter is shown in Figure 5.8. The input signal *in* has a voltage range from GND to V_{ddl}. This signal is not directly used in a pull up network because if V_{ddl} is less than $V_{dd} - V_{th}$, then *in* cannot turn off PMOS transistors in the V_{dd} domain. Instead, *in* and its inverted version are fed into the pull down NMOS transistors of a cross coupled PMOS structure. When one of the NMOS transistors becomes active it begins to discharge its output node. A short circuit then exists between the NMOS and PMOS transistors. If the NMOS transistor is sized correctly, it will win the fight with the PMOS transistor and eventually both cross coupled nodes will switch. Higher voltage signals can be used freely in lower voltage domains, therefore a high to low converter is not needed.

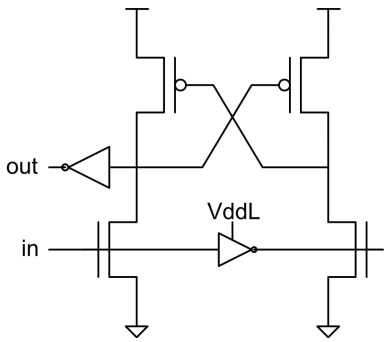

Figure 5.8: The standard voltage converter.

In asynchronous circuits, we will be converting voltage across channels rather than across simple signals. For dual rail codes, we have three signals to convert: the two data rails and the enable rail (acknowledge). In channels going from a lower to higher voltage domain, the data rails need to be converted to the higher voltage. In channels going from a higher to lower voltage domain, the enable rail needs to converted to the higher voltage.

5.3.2 High Voltage to Low Voltage Converter

The short circuit that occurs in the conventional voltage converter can be avoided by guarding the conversion with high voltage signals that are available in the handshake. The following is the HSE for high to low voltage converter:

$$*[([R^e \wedge L^f \longrightarrow R^f\uparrow [] R^e \wedge L^t \longrightarrow R^t\uparrow]; L^e\downarrow;), en\uparrow;$$
$$[R\_^e \wedge L\_^f \wedge L\_^t]; en\downarrow; R^f\downarrow, R^t\downarrow; L^e\uparrow]$$

This HSE is similar to the HCHB. The main difference is that we use inverted versions of the R^e, L^f, and L^t. Conveniently, the half cycle timing assumption allows us to use inverted versions of signals without having to check each transition on the inverted and non-inverted version of the signal (this is not possible in pure QDI circuits).

The high to low converter is shown in Figure 5.9. $R^e\downarrow$ is sensed in the en stack and $R^e\uparrow$ is sensed in the R^f and R^t stacks. The short circuit found in the conventional converter is avoided by guarding the stacks with L^f and L^t and their inverted versions. This only leaves the short circuit caused by the weak feedback that is common to all state holding gates. We can mitigate the impact of the weak feedback by adding a weak series PMOS for $R\_^e$ in en's feedback and R^e in R^f and R^t's feedback. An example of this is shown in the pull up feedback for en. When

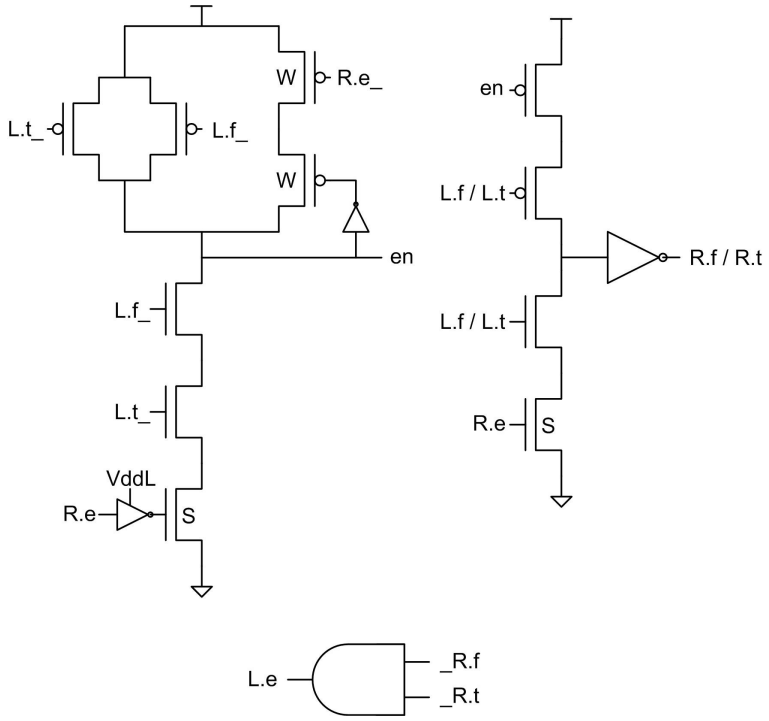

Figure 5.9: A pipelined high voltage to low voltage dual rail converter. The pull up feedback on the enable stack is explicitly shown.

$R\_^e$ goes to V_{DDL}, and $L\_^f/L\_^t$ are high, the bottom transistor partially turns on. At the same time, the top weak PMOS partially turns off which reduces the weak short circuit current. The transistors marked with a S are made strong by using a low threshold voltage transistor and over sizing it.

5.3.3 Low Voltage to High Voltage Converter

The low voltage to high voltage converter takes two low-swing data rails and one full-swing enable as inputs. It outputs two full-swing data rails and one low-swing enable. The HSE is the same as high voltage to low voltage converter. The main difference is that the strong inverters are used for the data rail inputs and the pull-up logic for each stack changes slightly, as shown in Figure 5.10.

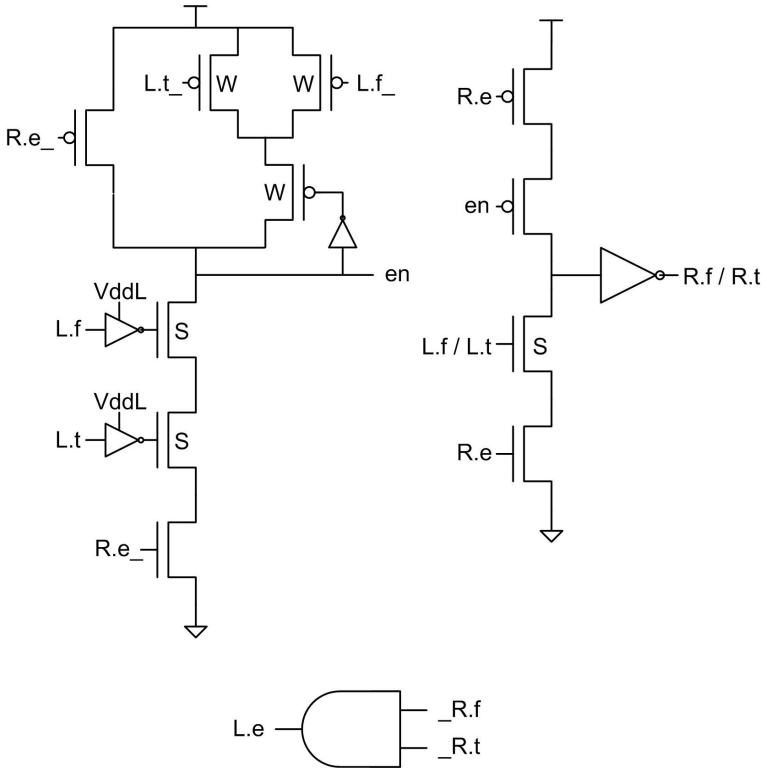

Figure 5.10: A pipelined low voltage to high voltage dual rail converter. The pull-up feedback on the enable stack is explicitly shown.

5.3.4 DVHB Circuit Template

Simply scaling the voltage in an asynchronous circuit will increase both its cycle time and latency. The latency can be kept constant by keeping the logic in the forward path in the high voltage domain and moving only the logic in the return path to the low voltage domain. In other words, the data rail stacks use V_{dd} and the enable/acknowledge logic uses V_{ddl}. A voltage conversion is needed whenever the data rail logic uses a signal from the enable logic.

To minimize the forward latency, we start with a PCEHB reshuffling:

$$*[[R^e]; en\uparrow; [L^f \longrightarrow R^f\uparrow [] L^t \longrightarrow R^t\uparrow]; L^e\downarrow;$$
$$[\neg R^e]; en\downarrow; R^f\downarrow, R^t\downarrow; [\neg L^f \wedge \neg L^t]; L^e\uparrow]$$

Both the R^e and L^e signals are in the low voltage domain. The PCEHB reshuffling

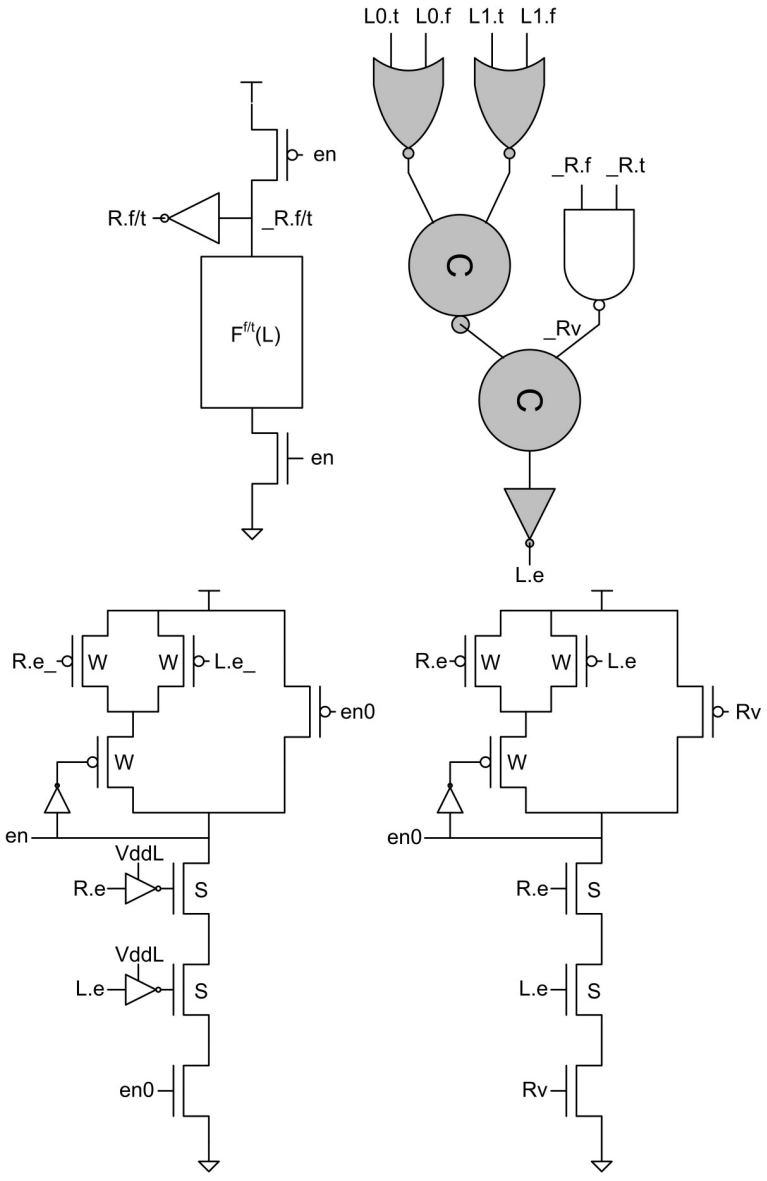

Figure 5.11: The DVHB circuit template. The shaded logic is in a lower voltage domain.

allows us to remove these signals from the data rails and do the voltage conversion in the logic for en. Similar to the voltage converters in the previous subsection, we need to find a high voltage signal to use as a guard in the en logic. The only choice is to use the validity of the data rails, Rv. The dual voltage half buffer circuit (DVHB) is depicted in Figure 5.11. The shaded logic is in the low voltage domain. The voltage conversion occurs in the $en0$ and en stacks. We use the same technique as before to lessen the weak feedback during the conversion.

Although the shaded logic seems to be a small part of the circuit, this logic switches every cycle compared to the data rails where only one rail switches per cycle. In addition, by making L^e and R^e low voltage we have reduced the switching in the channels where wires tend to be longer and more capacitive. Moreover, the en stack, $en0$ stack and the data rails only have one series PMOS transistor which helps to limit their output capacitance (the weak feedback PMOS transistors are minimum size and do not contribute greatly to the output capacitance).

5.4 Results

5.4.1 DVHB Template

In order to choose the correct voltage for enable scaling, we characterize the backward latency for a DVHB buffer in the target process (65 nm) using Hspice. We assume that we can accurately adjust the voltage by increments of $.05V$. Figure 5.12 shows the backward latency, $l_b{}'$, as we scale the enable signal from $1V$ to $.5V$. It also shows the power reduction resulting from enable scaling from $1V$ to $.5V$. Note, the power reduction displayed is on top of the reduction that results from operating at the reduced frequency.

Although we have reversed the x-axis, $l_b{}'$ follows $f(x) = 1/x$. This is as ex-

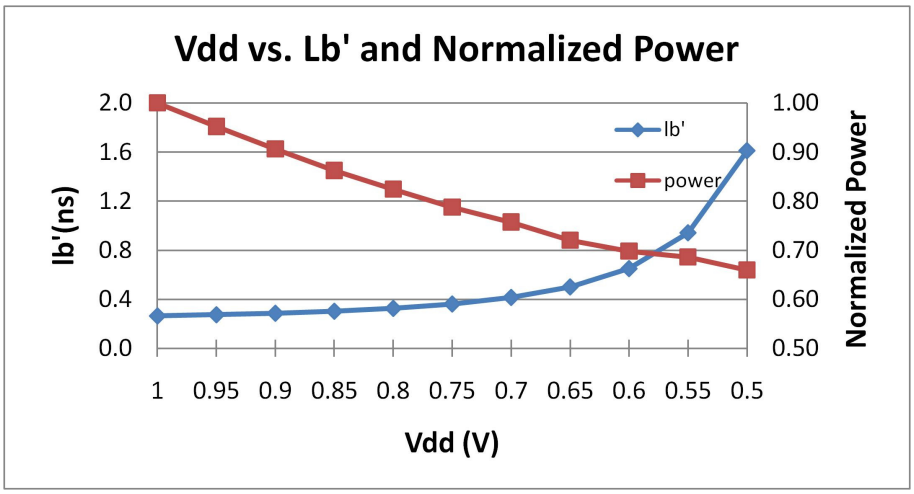

Figure 5.12: Relationship of $l_b{}'$ to normalized power during enable scaling for a DVHB buffer.

pected because the delay is proportional to $1/V_{dd}$. As a result, $l_b{}'$ increases slowly through about $.75V$ and then more quickly as we approach $.5V$. This means that small increases in $l_b{}'$ initially result in large voltage drops for the enable signal, but after about $.75V$ it takes larger increases in $l_b{}'$ to see addition drops in voltage.

5.4.2 Loops

To evaluate the potential power savings in loops, we simulate a token ring with a 1-3 tokens and 20-48 stages of half-buffers. The results are shown in Figure 5.13. When there are 20 stages in the ring and $k_0 = 2$, the pipeline is running at full throughput and there is no power savings. As the number of ring stages are increased, the throughput becomes more token-limited and enable scaling is used to save power.

The $k_0 = 1$ pipeline is always token-limited and power can be save across the whole range of ring stages. Power only scales to about 70% for two reasons: i) only roughly half the nodes in the ring run at the lower voltage, and ii) $l_b{}'$ is a $f(x) = 1/x$ function. The $k_0 = 3$ pipeline can not use enable scaling until it

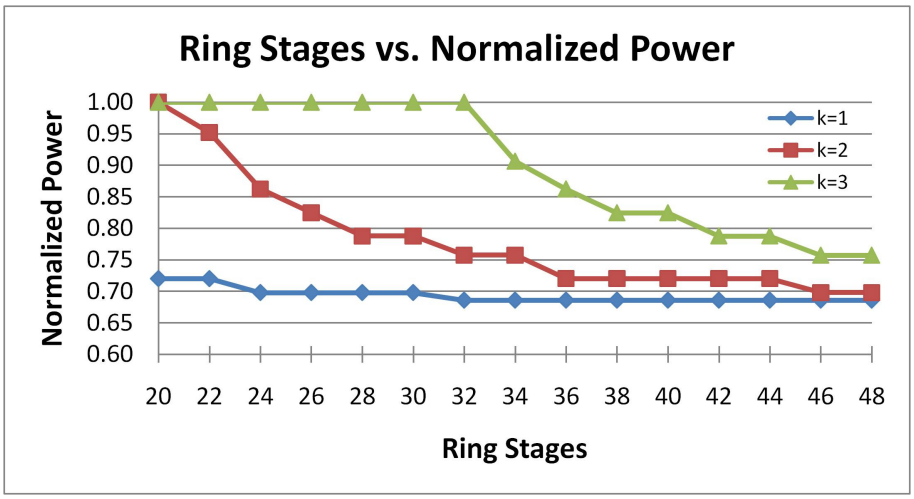

Figure 5.13: Normalized power usage of loops built from four-phase half-buffer pipelines with one, two and three tokens, k.

contains 34-stages because it is hole-limited up until that point.

5.4.3 Parallel Pipelines

To evaluate the potential power savings in reconvergent paths, we simulate a 2-20 stage short pipeline with no initial tokens and a 20-stage long pipeline with 0-3 initial tokens. At $k_0 = 0$ with 20 stages in the short pipeline, the composition runs at peak throughput. As we decrease the number of stages in the short pipeline, the throughput decreases and enable scaling reduces power. In this composition, we only scale the enable in the long pipeline. At 12 stages, the composition runs at 90% of peak throughput and there is a 10% power reduction. At 2 stages, the composition runs at 40% of peak throughput and has a power reduction of about 30%.

For $k_0 = 1$ at 20-12 stages, the long pipeline is effectively shorter than the short pipeline because of the initial token (recall that $n' = n - k_0\tau/l_f$). In this region, power savings are small because we are scaling the enable on the short pipeline. This remains true until the number of stages in the short pipeline drops to about

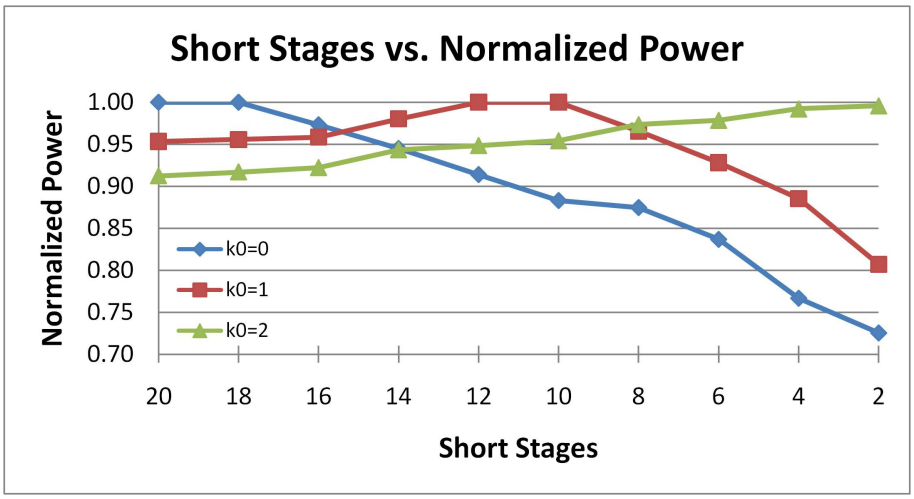

Figure 5.14: Normalized power usage of two parallel pipelines built from four-phase half-buffers. The longer pipeline has 20 stages and it contains zero, one, or two initial tokens, k_0.

11 (at this point the composition runs at peak throughput). For the composition with $k_0 = 2$, the short pipeline is effectively longer than the long pipeline for the whole simulation. The power reduction decreases because we are removing stages from the effectively longer pipeline. The power reduction for this composition is never larger than 10%. This is due to enable scaling on the short pipeline that only contains 50% to 10% of the total number of pipeline stages.

CHAPTER 6

CASE STUDY: FPGA

6.1 Architecture

The RQDI circuits described in the preceding chapters are most effective when applied to asynchronous FPGAs. In this section, we present a simple FPGA architecture with hardware modifications to support RQDI two-phase circuits and RQDI voltage scaling.

6.1.1 Overview

A high-level overview of the asynchronous FPGA is shown in Figure 6.1. The FPGA is composed of configurable logic blocks (CLB) connected together through an array of switch boxes (SB). In a synchronous FPGA, the SB is build from simple multiplexors. In an asynchronous FPGA architecture, the SB is made of buffered switches, similar to the one in Figure 4.7. This buffering in the interconnect allows asynchronous FPGAs to run at much high frequencies than synchronous FPGAs. However, this buffering also greatly increases the size of the SB and increases its power consumption. The FPGA is programmed through a chip-wide distributed memory (not shown).

6.1.2 Baseline Routing

Each switch box has 32 inputs and 32 outputs in each direction, as shown in Figure 6.2. The switch box is disjoint, i.e., the horizontal and vertical routes only connect at 32 switch points along their diagonal. As a result, an input can only exit the switch box on the same track. At each switch point, an input may change

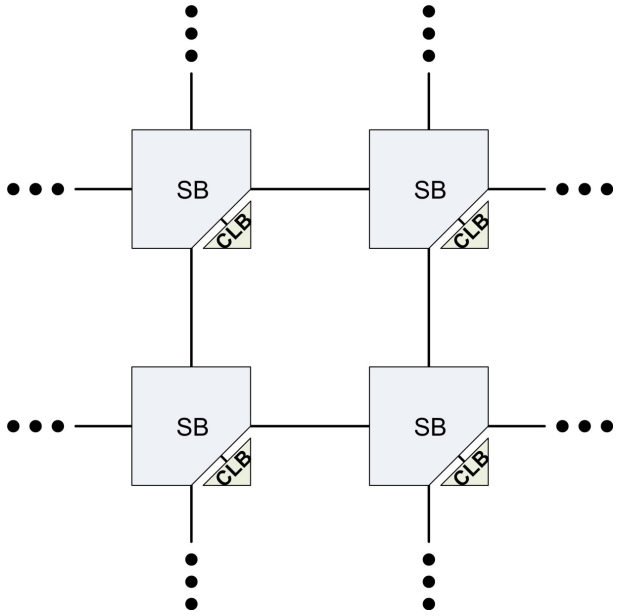

Figure 6.1: An asynchronous FPGA fabric composed of switch boxes (SB) and configurable logic blocks (CLB).

directions or enter the CLB. In order for an input to change tracks, it must enter and exit the CLB, burning CLB resources. The switch point can handle up to two different inputs and copy them out to any combination of outputs. Each switch point contains two four-input to four-output (4:4) switches (Figure 4.7).

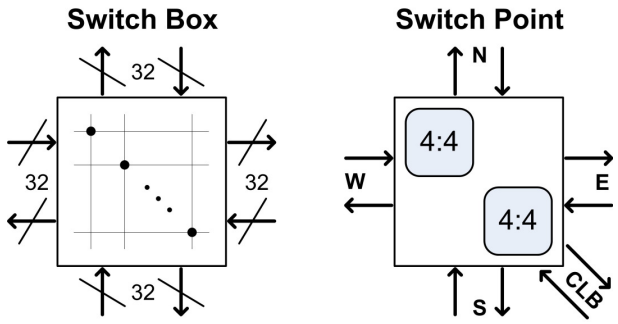

Figure 6.2: A 32 x 32 disjoint switch box made from 32 switch points.

It is often the case that outputs of a CLB are inputs to another CLB that is several tiles away, rather than adjacent to their tile. Due to this, it is unnecessary for each track to stop at every SB. To take advantage of these cases, different

length wire segments are often used to reduce the area of SBs and decrease the latency through a track. Figure 6.3 shows the three types of wire segments used in the target architecture. For instance, a hex segment connects SBs that are six tiles apart. In the target architecture, there are 12 singles, 12 doubles and 8 hexes.

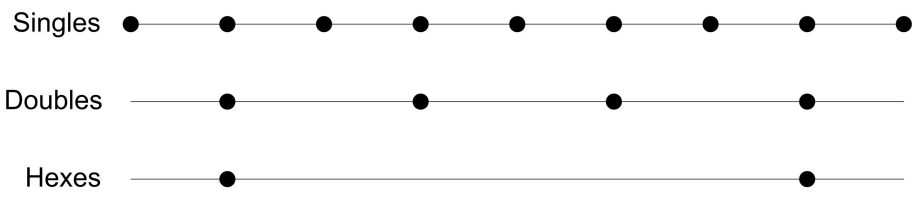

Figure 6.3: The three types of routing segments used in this FPGA.

6.1.3 Routing Enhancements

We have shown, in Chapter 4, the large power saving potential of two-phase routing. Unfortunately, two-phase buffers are significantly larger than four-phase buffers. There are two ways to implement two-phase routing for a minimal area impact: i) replace two four-phase stages with a single two-phase stage, and ii) replace a single four-phase stage with a single two-phase stage, but undersize the logic on the backward path to mitigate the area overhead. Typically, the second option would have a larger area impact, but it provides a compelling performance advantage.

When four-phase half-buffers are replaced with two-phase full-buffers, the amount slack in the pipeline is doubled. This alters the throughput curve of the pipeline, as shown in Figure 6.4. For hole-limited domain, the throughput of the pipeline is much larger using two-phase circuits. This is a very important result because it mitigates the performance impact of unbalanced reconvergent paths. Recall, the throughput of reconvergent paths is limited by the token-limited domain of the

longer pipeline and the hole-limited domain of the shorter pipeline. Two-phase circuits also improve the throughput of the less frequent hole-limited loops.

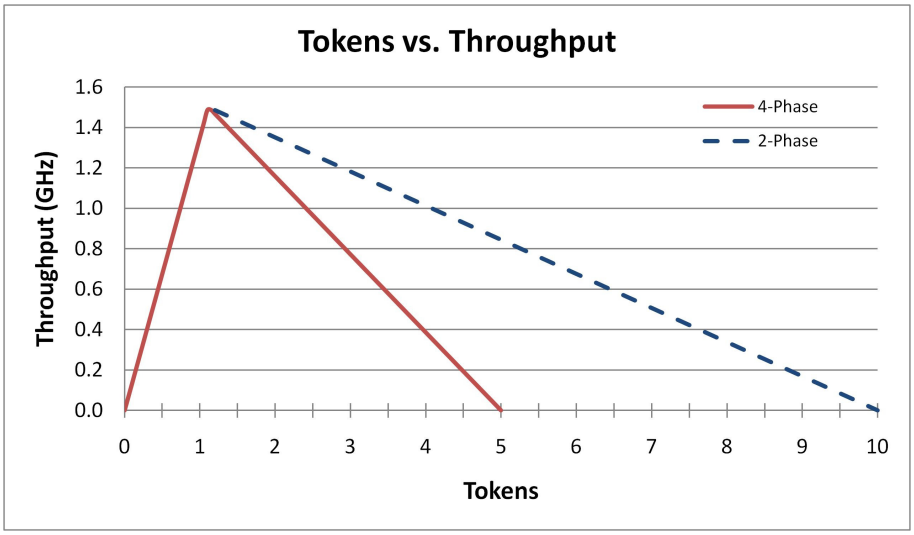

Figure 6.4: Throughput improvement in hole-limited domain from using two-phase routing.

In addition to being two-phase, each switch has additional logic to support enable scaling (voltage scaling on enable signals). Figure 6.5 shows the resulting two-phase low-power switch point. The shaded logic can be configured to use the nominal voltage, V_{dd}, or a lower voltage , V_{ddl}. The programmable c-element labeled PC2V has a built-in voltage converter. Although it may appear that this switch point would be much larger than a typical four-phase 4:4 switch, it is less than 10 % larger. The reason is that all of the logic on the backward path, namely the XOR and PC gates, can be downsized by more than 50 % and the circuit will still run at the same frequency of a four-phase 4:4 switch. This style of enable scaling hardware allows each 4:4 switch to be enable-scaled individually. There is a single low voltage V_{ddl} available on the chip, but each 4:4 switch can be programmed to use V_{ddl} or the nominal V_{dd}. The V_{ddl} voltage source is set globally.

Only a small amount of logic in the two-phase 4:4 switch runs at a lower voltage. However, most of the capacitance in the circuit resides on the external wires (the

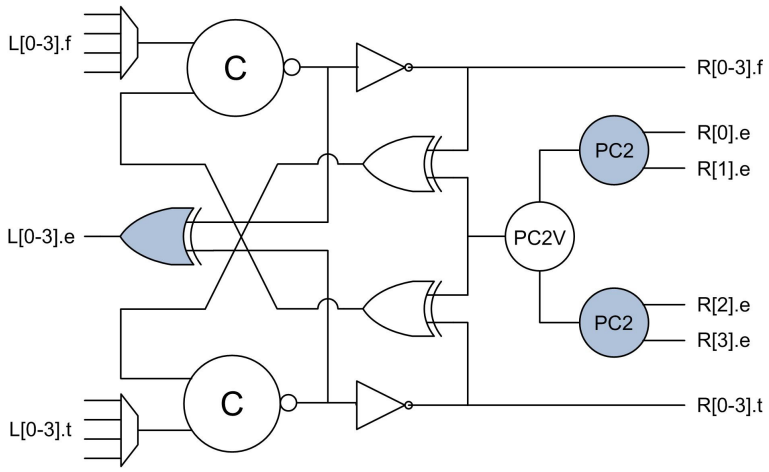

Figure 6.5: The 4:4 low-power switch used in the low-power switch point. The shaded logic can be configured to use a lower V_{dd}.

two data rails and the enable). The capacitance on each external wire is 20 fF, 40 fF, and 120 fF for singles, doubles, and hexes, respectively. This dwarfs the less than 5 fF capacitances seen on the internal nodes. The lower-voltage enable rail and one of the nominal-voltage data rails switch each cycle. Therefore, the amount of capacitance that runs at a lower voltage is nearly 50%.

6.1.4 Configurable Logic Block

The CLB for the target FPGA architecture is shown in Figure 6.6. The CLB contains a logic core surrounded by an input and output connection boxes. The connection boxes allow any input or output of the logic core to connect to any switch point in the SB. Inside the logic core are four four-input lookup tables (LUT4). The LUT4 outputs one of its 16 preprogrammed values based upon the four inputs. There are four LUT4s, therefore the logic core has a total of 16 inputs and 4 outputs. To support two-phase routing, the only alterations needed is the addition of phase converters for each input and output of the logic core.

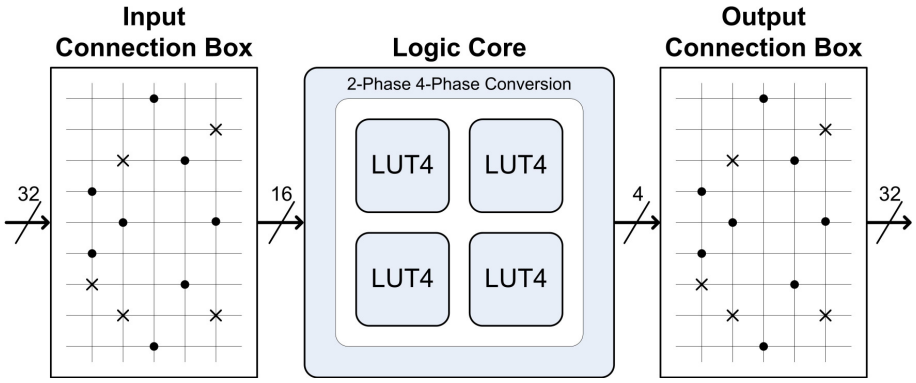

Figure 6.6: The CLB contains input/output connection boxes, four LUTs, and phase converters.

6.2 Timing Analysis

User designs that run on high-speed asynchronous FPGAs rarely operate at peak throughput for two reasons: i) it is not possible to balance all reconvergent paths and loops with limited routing resources, and ii) the designs may not be pipelined enough to take full advantage of the underlying architecture. If the designs are not operating at peak throughput, then power is being wasted when running the FPGA at the nominal voltage. As we have shown in Chapter 5, enable scaling, when applied correctly, can reduce power without impacting performance. By applying some timing analysis to user designs, we can determine the amount of enable scaling possible at each 4:4 switch. With this information, we can determine the optimal global V_{ddl} and which 4:4 switches are to be configured to use it.

6.2.1 Operating vs. Potential Throughput

For the purposes of timing analysis, we can construct a graph representation of user designs that have been mapped to the FPGA. Each node in the graph represents a buffer stage in the FPGA, e.g., LUTs and switches. From this graph, we can identify throughput limiting loops and reconvergent paths. The operating

throughput, T_O, of all the nodes in the graph is limited by:

1. the least throughput node anywhere in the graph
2. the least throughput loop anywhere in the graph
3. the least throughput reconvergent path anywhere in the graph

Although each node is limited by T_O, they may have a much higher potential throughput, T_P. Figure 6.7 shows a graph with three potential throughput limiting structures, i.e., Loop 1, Loop 2, and RCP 1. The T_O of the graph is limited by the structure with the least throughput. The T_P of the shaded node may be higher than T_O because it is only affected by the throughput of Loop 1. If $T_P > T_O$, then some about of enable scaling is possible. The precise amount of enable scaling can be determined by methods describe in the previous chapter. Specifically, finding the backward latency, $l_b{}'$, that makes $T_P = T_O$ and cross-referencing $l_b{}'$ with a characterization of the underlying circuit implementation, similar to that shown in Figure 5.12.

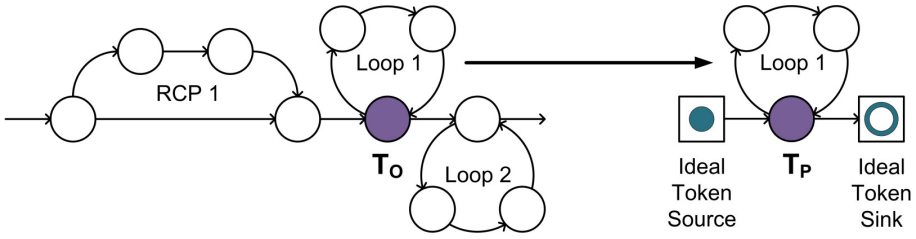

Figure 6.7: The operating frequency of the shaded node is limited by the least throughput structure of the two loops and reconvergent path.

6.2.2 Determining V_{ddl}

Once we know the minimal enable voltage for each node, V_{ddli}, determining the global V_{ddl} is straightforward. V_{ddl} can be anywhere from 1 V to .5 V at increments of 50 mV. Therefore, there are only 11 possible values for V_{ddl}. For each possible

V_{ddl}, we compare V_{ddl} to the V_{ddli} of each node. If $V_{ddl} < V_{ddli}$, then no power savings are possible in this node. If $V_{ddl} \geq V_{ddli}$, then we can lookup the associated power reduction for the underlying circuit with an enable operating at V_{ddl} (similar to Figure 5.12). We choose a global V_{ddl} that maximizes the overall power reduction.

6.3 Results

6.3.1 Setup

Table 6.1 highlights the most important architectural parameters of the target asynchronous FPGA. This architecture is designed to support all of the benchmarks listed in the following subsection.

Table 6.1: Target FPGA architectural parameters.

FPGA	
Fabric Maximum Frequency	1.5 GHz
Process Technology	65 nm
Switch Box	32 x 32 Disjoint Network
Wire Segments	12 Singles, 12 Doubles, and 8 Hexes
Logic Core	4 4-input LUTs
Array Size	48 x 48
Place and Route	VPR

6.3.2 Benchmarks

The benchmarks used in our evaluations are listed in Table 6.2. These are 8 of the 20 MCNC LGSynth93 benchmarks. Only ten of the MCNC LGSynth93 benchmarks are pipelined and two were excluded because they ran at less than 100 MHz. The standard way synchronous designs are mapped to an asynchronous architecture is to convert all flops in the design into initial tokens. The only additional hardware needed to support this is a configurable initial token on the output of each LUT.

Table 6.2: The eight MCNC LGSynth93 Benchmark circuits used in evaluations.

Name	Array Size	LUT Count
bigkey	36 x 36	1707
clma	47 x 47	8383
diffeq	20 x 20	1497
dsip	36 x 36	1370
elliptic	31 x 31	3604
frisc	30 x 30	3556
s38584.1	41 x 41	6447
tseng	17 x 17	1047

6.3.3 Area Estimates

Table 6.3 lists the area estimates for main components for the baseline FPGA and the low-power version. These area estimates are determined by comparing the total diffusion area of the sized netlist for each component against the post-layout area of similar circuits in this technology. Overall, the low-power FPGA is only about 12 % larger than the baseline FPGA.

The largest area increase occurs in the low-power CLB, which is about 36 % larger than the baseline CLB. However, in practice this area increase would be much less. Typically, the logic core would contain a number of full-adders which would help to amortize the cost of the phase converters. In addition, the logic core could be altered to use two-phase bundled-data, which would be more compatible with the two-phase routing. Converting from LEDR to two-phase bundled-data is much cheaper than converting from LEDR to QDI.

6.3.4 Power and Performance

Figure 6.8 shows the operating frequency of each benchmark with four-phase (baseline), two-phase, and two-phase enable-scaled routing. None of the benchmarks come within 40 % of the peak frequency of the underlying architecture. This is partly due to the fact that a synchronous place and route tool, VPR, was used to map the designs to the FPGA. (An academic asynchronous place and route

Table 6.3: Area estimates for FPGA circuits.

Circuit	Area
Switch Point	245 μm^2
Switch Box	7840 μm^2
LUT4	185 μm^2
Input CBOX	265 μm^2
Output CBOX	65 μm^2
CLB	1070 μm^2
Baseline FPGA	**2.05 mm^2**
Low-Power Switch Point	265 μm^2
Low-Power Switch Box	8480 μm^2
4:2 Converter	35 μm^2
4-wide 2:4 Converter	60 μm^2
Low-Power CLB	1450 μm^2
Low-Power FPGA	**2.29 mm^2**

tool does not exist yet.) However, even if an asynchronous place and route were used, these benchmarks do not contain enough pipelining to run near the peak frequency of the technology. Synthesis at a much higher level would be required to take full advantage of the high-speed asynchronous FPGA. Even at these speeds, some of these benchmarks may be 2-3 x faster than they would be running on a synchronous FPGA.

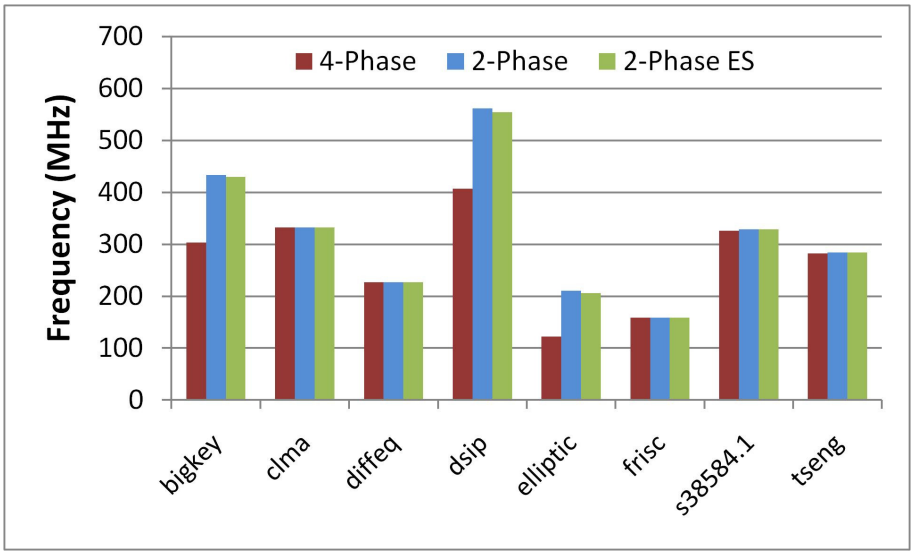

Figure 6.8: Operating frequency of each benchmark for four-phase, two-phase, and two-phase enable-scaled routing.

Moving from four-phase routing to two-phase routing results in a 40 % per-

formance improvement in bigkey and dsip, and a 70 % performance improvement in elliptic. These designs were limited by either a reconvergent path or a hole-limited loop. Two-phase circuits double the slack in the routing, which drastically improves the throughput in these structures. Due this performance increase, the power reduction in these benchmarks from two-phase routing is much less than the other five benchmarks, as shown in Figure 6.9. The bigkey benchmark has a 15% power decrease, dsip has a 30 % power decrease, and elliptic has a 3 % increase. There is a 40 % power decrease in the remaining benchmarks. The full 50 % power decrease is never seen because the slight area increase from using two-phase circuits makes the wires in the routing a bit longer and more capacitive. However, even a 40 % power reduction is quite large.

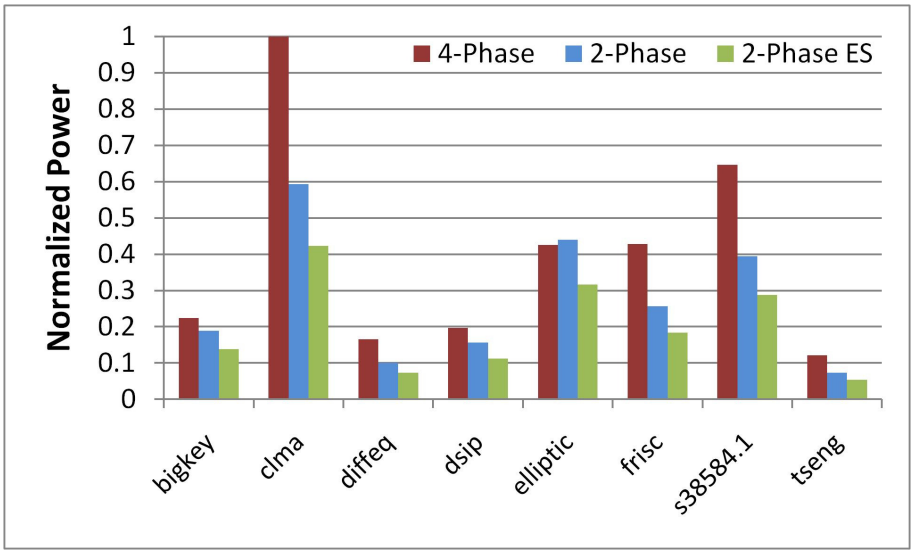

Figure 6.9: Normalized power consumption of each benchmark for four-phase, two-phase, and two-phase enable-scaled routing. All benchmarks are normalized to the clma benchmark.

Enable scaling provides an additional 28 % power reduction across all benchmarks. The choice of V_{ddl} for clma and s38584.1 is .55 V and .5 V is used for all the other benchmarks. The power reductions are close to the theoretical 35 % power reduction possible from enable scaling down to these operating frequencies. This

occurs because a high percentage of switches can be enable scaled. The structures that prevent enable scaling, such as being on the short path of a reconvergent path or on a hole-limited loop, are rare. In addition, two-phase routing fixes some of these structures and prevents them from limited enable scaling. Although elliptic sees a total power reduction of only 25 % because of its 70% performance increase, the other benchmarks experience a power reduction of 40% - 60%.

CHAPTER 7

CONCLUSIONS

We have presented a class of circuits that are derived by starting with quasi delay-insensitive circuits and applying a conservative timing assumption, namely the half cycle timing assumption. We refer to these as relaxed quasi delay-insensitive circuits. We used these circuits to help reduce power consumption in a few ways. First, we developed the half cycle half buffer (HCHB) circuit template that reduces the amount logic needed to generate enable/acknowledge signals. The HCHB template reduces area by 15% and energy by 32% on average across our benchmark circuits. Second, we presented a two phase buffer for use in global communication and static switching networks. This buffer was shown to reduce energy in static switches by over 50%. Third, we showed how to fold voltage converters into the HCHB buffer. We also proposed the dual voltage half buffer (DVHB) to allow voltage scaling on the enable/acknowledge logic (return path) while keeping the data logic (forward path) in a high voltage domain to maintain a constant forward latency.

As a case study, we designed an asynchronous FPGA using RQDI two-phase circuits and RQDI voltage scaling circuits. For eight of the MCNC LGSynth93 benchmarks, RQDI two-phase circuits provide up to a 70 % performance improvement and up to a 40 % power reduction. The RQDI voltage scaling circuits provide an additional 30 % power reduction across these benchmarks. The total power reduction is up to 60 %.

In conclusion, RQDI circuits are an important first step to mitigate the power increases expected from technology scaling in the deep submicron. Although, even more aggressive power saving techniques will likely be necessary in future technologies. In addition, RQDI circuits should be given serious consideration in all future

asynchronous FPGA designs due to their low overhead, large power reductions, and large performance improvements.

BIBLIOGRAPHY

[1] The international technology roadmap for semiconductors. `http://public.itrs.net`.

[2] Mark E. Dean, Ted E. Williams, and David L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (ledr). In *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*, pages 55–70, Cambridge, MA, USA, 1991. MIT Press.

[3] Chris Edwards. The wrong answers. `http://kn.theiet.org/magazine/issues/0911/the-wrong-answers-0911.cfm`, June 2009.

[4] David Fang, Christopher LaFrieda, Song Peng, , and Rajit Manohar. A 3-tier asynchronous fpga. In *Proceedings of the 23rd International VLSI/ULSI Multilevel Interconnection Conference*, 2006.

[5] David Fang, John Teifel, and Rajit Manohar. A high-performance asynchronous fpga: Test results. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

[6] M. Ferretti and P. Beerel. Single-track asynchronous pipeline templates using 1-of-n encoding. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 1008, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Michael K. Gowan, Larry L. Biro, and Daniel B. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *35th Design Automation Conference*, pages 726–731, 1998.

[8] Ben Hardwidge. Ibm technology alliance reveals 28nm chip plans. `http://www.bit-tech.net/news/hardware/2009/04/17/ibm-reveals-28nm-technology-plans/1`, April 2009.

[9] Charles Hawkins, Ali Keshavarzi, and Jaume Segura. A view from the bottom: Nanometer technology ac parametric failures why, where, and how to detect. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, 0:267, 2003.

[10] Pieter Johannes Hazewindus. *Testing delay-insensitive circuits*. PhD thesis, Pasadena, CA, USA, 1992.

[11] Rick Hodgin. Intels manufacturing prowess skips some 45nm cpus, goes straight to 32nm. http://www.geek.com/articles/chips/intels-manufacturing-prowess-skips-some-45nm-cpus-goes-/straight-to-32nm-20090618, June 2009.

[12] Mark Horowitz. Scaling, power and the future of cmos. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, page 23, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Christopher LaFrieda and Rajit Manohar. Fault detection and isolation techniques for quasi delay-insensitive circuits. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 41, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Christopher LaFrieda and Rajit Manohar. Reducing power consumption with relaxed quasi delay-insensitive circuits. In *ASYNC '09: Proceedings of the 2009 15th IEEE Symposium on Asynchronous Circuits and Systems (async 2009)*, pages 217–226, Washington, DC, USA, 2009. IEEE Computer Society.

[15] Andrew Matthew Lines. Pipelined asynchronous circuits. Master's thesis, California Institute of Technology, 1996.

[16] R. Manohar and M. Nystrom. Implications of voltage scaling in asynchronous architectures. Technical report, Cornell Computer Systems Lab CSL-TR-2001-1013, April 2001.

[17] Alain J. Martin. Compiling communicating processes into delay-insensitive vlsi circuits. *Distributed Computing*, 1(4), 1986.

[18] Alain J. Martin. Programming in vlsi: from communicating processes to delay-insensitive circuits. pages 1–64, 1990.

[19] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The design of an asynchronous mips r3000 microprocessor. In *Advanced Research in VLSI*, pages 164–181, 1997.

[20] Clive Maxfield. New 1.5 ghz fpgas shipping now! http://www.pldesignline.com/210601830, September 2008.

[21] Amitava Mitra, William F. McLaughlin, and Steven M. Nowick. Efficient asynchronous protocol converters for two-phase delay-insensitive global com-

munication. In *ASYNC '07: Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 186–195, Washington, DC, USA, 2007. IEEE Computer Society.

[22] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[23] Kevin Morris. Fast. very, very fast. `http://www.fpgajournal.com/articles_2008/20080916_fast.htm`, September 2008.

[24] Montek Singh and Steven M. Nowick. Mousetrap: high-speed transition-signaling asynchronous pipelines. *IEEE Trans. Very Large Scale Integr. Syst.*, 15(6):684–698, 2007.

[25] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, 1989.

[26] Ivan Sutherland and Scott Fairbanks. Gasp: A minimal fifo control. In *ASYNC '01: Proceedings of the 7th International Symposium on Asynchronous Circuits and Systems*, page 46, Washington, DC, USA, 2001. IEEE Computer Society.

[27] John Teifel and Rajit Manohar. Highly pipelined asynchronous fpgas. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 133–142, New York, NY, USA, 2004. ACM.

[28] John Teifel, Rajit Manohar, David Fang, Clint Kelly, and David Biermann. Energy-efficient pipelines. In *ASYNC '02: Proceedings of the 8th International Symposium on Asynchronus Circuits and Systems*, page 23, Washington, DC, USA, 2002. IEEE Computer Society.

[29] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective (3rd Edition)*. Addison Wesley, 3 edition, May 2004.