Period and Glitch Reduction Via Clock Skew Scheduling, Delay Padding and GlitchLess

by

Xiao Dong

B.A.Sc., The University of British Columbia, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia

(Vancouver)

September, 2009

Abstract

This thesis describes PGR, an architectural technique to reduce dynamic power via a glitch reduction strategy named GlitchLess, or to improve performance via clock skew scheduling (CSS) and delay padding (DP). It is integrated into VPR 5.0, and is invoked after the routing stage. Programmable delay elements (PDEs) are used as a novel architecture modification to insert delay on flip-flop (FF) clock inputs, enabling all optimization steps to share it, avoiding multiple architecture modifications. This thesis investigates the trade-off between power and performance, and finding an appropriate compromise considering process variation and timing uncertainties.

To facilitate realistic power estimates, a popular activity estimator, ACE, is modified with a new model to estimate glitching power, taking into account the analog behavior of glitch pulse width reduction as it travels along FPGA routing tracks. We show that the original glitch estimation method can underestimate glitching power by up to 48%, and overestimate by up to 15%.

In terms of performance, an average of 15% speedup can be achieved via CSS alone, or up to 37% for individual circuits. Although delay padding only benefits a few circuits, the average improvement of those circuits is an additional 10% of the original period, or up to 23% for individual circuits. In addition, GlitchLess is performed on both the original VPR and post-CSS solutions. On average, 16% of glitching power can be eliminated, or up to 63% for individual circuits.

Contents

List of Tables

List of Figures

Acknowledgements

First of all, I would like to thank my supervior Guy Lemieux, for his unwaivering support and guidence for two years. Without him, I would not be writing this thesis. I am very grateful for his time and patience.

Thanks to Roozbeh Mehrabadi of the SOC lab for his timely technical support when CAD tools stray from their expected behavior. I would also like to thank my lab-mates, Usman Ahmed, Scott Chin, Darius Chiu, Chris Chou, David Grant, Julien Lamoureux, Paul Teehan, and Mark Yamashita for their help when I was stuck on various stages of my research.

My last but not the least thanks goes to my parents for standing by me all these years, for their encouragement when I was down, for helping me to be a successful student, and for teaching me to be a better person.

Chapter 1

Introduction

1.1 Motivation

Power and performance are two very important issues in FPGA design. FPGA applications typically consume more power per operation, and run at slower speeds than their ASIC counterparts, due to circuitry needed for programmability.

There is much research effort addressing these two topics. On the performance front, two popular techniques are retiming and clock skew scheduling (CSS). The former method changes the positions of sequential elements (SEs) to shorten the effective critical path while maintaining functionality [21]. This work has also been applied to FPGAs [9, 25, 29]. One disadvantage of retiming is that the skew achievable may not be finely adjustable because the number of locations a SE can be placed is limited. Instead, CSS achieves period reduction by assigning intentional clock skews to SEs [11, 14] rather than moving them physically, and has also been applied to FPGAs [28, 30, 36]. Compared to retiming, the skews required by CSS may be realized using programmable delay elements (PDEs) that can be finely adjusted to provide many levels of quantized skew.

Total dynamic power consumption is significant for FPGAs due to large capacitive loading on the programmable interconnect. Recent advances in process technology have seen a decreasing trend in the rate of increase of dynamic power versus static power. However, total dynamic power still accounts for about 50% of total power [5]. In this

thesis, our use of the term "dynamic power" shall exclude clock network power; where appropriate, it will be considered separately. Dynamic power arises from two kinds of logic transitions produced by combinational logic building blocks in FPGAs called look-up tables (LUTs): functional and glitch. The former causes the data to be different at the end of a clock period, a result of user logic functions. The latter results from input data signals arriving at different times during the period, causing the output to fluctuate before settling down. Several existing examples to reduce glitching power include techniques at the architecture level [19], or at the CAD level during technology mapping [7] and routing [12].

1.2 Objectives

The purpose of this thesis is to achieve performance and power optimization with a single architecture change. This change increases FPGA area, but it does not alter the existing place-and-route algorithms and it makes only small netlist changes after routing is complete. This approach is called PGR, for period and glitch reductions. CSS and GlitchLess [19] are chosen as a basis of this work because these two methods can be applied to the proposed architecture. The algorithm will try to reduce period and power within the limitations of the architecture and the final routing solution. The PDE proposed in [19] is used here to provide discrete delays.

To reduce period to the lowest possible value without violating timing requirements, the CSS algorithm iteratively determines if there exists a set of quantized skews that can satisfy a given clock period, based on a set of constraints given by the properties of the circuit. Once the improved period has been determined, a technique called delay padding (DP) is used to relax the constraints used when solving the CSS problem, possibly achieving even further period reduction. In addition, process variation can cause signals to arrive earlier

or later than desired, and PGR takes this into account and allocates extra timing margins.

With a traditional zero-skew clock network, the departure time of all SEs is synchronized to the active clock edge. Skews on SE clocks change the departure times by varied amounts, causing downstream nodes' arrival times to change. This affects the amount of glitching present in a circuit, usually increasing it. In either case, an accurate tool is needed to determine the amount of glitching activity on each node. A node is either a SE, or a combinational LUT. An existing tool called ACE [20] is one such tool. ACE uses a threshold to determine whether a glitch does not propagate at all or propagates indefinitely until the next node. One goal of this work is to model the analog behavior of the gradual decrease in width of a narrow glitch as it travels along FPGA interconnect, and to determine the relationship between glitch pulse width and power consumption. More accurate glitch estimation leads to GlitchLess making better decisions on which node to reduce glitches, based on the amount of glitching power its output produces.

In this thesis, the original GlitchLess concept is used with a different implementation. Previously, delay elements added to node inputs needed to be very precise to eliminate glitching. This can be difficult with increasing process variations, which adds delay uncertainty. The new approach is much more resistant to variation because it prevents the output from transitioning until after the last expected input has arrived.

One final objective is to restrict the number of PDEs available in the architecture in an effort to save area. The PGR algorithm will try to reduce the number of PDEs used and the impact on optimization results via a two-pass approach. The first pass will assume every SE has access to a PDE. The second pass arranges SEs to share a reduced number of PDEs, and the sharing schedule is decided based on the results from the first pass.

1.3 Contributions

This thesis makes the following contributions summarized from the last section:

1. A unified architecture change, shared by CSS, delay padding and glitch reduction, is proposed which avoids the need for multiple architecture modifications. For glitch reduction, GlitchLess is applied using a different implementation than previous work.

2. Integrated delay padding scheme with CSS further optimizes performance. Past work [16, 18, 22, 33] uses either LP or graph algorithms to improve CSS. However, these techniques apply only to ASICs, and assume padded delays are continuous. However, FPGAs must use PDEs, and a PDE can only provide discrete delays. We adapt the algorithms to use discrete delays as well as margin for process variation.

3. PGR uses the same physically realizable architectural change to reduce power and increase performance. CSS, delay padding and glitch reduction techniques are combined with VPR 5.0 [23] into a single executable. This is important for getting a final result that considers both delay and power at the same time.

4. An improvement on vector based activity estimation [20] is proposed, taking into account the analog behavior of glitch pulses that travel along routing tracks. The resulting glitching power estimation is therefore more realistic.

The central theme of this work highlights the major difference of this work: previous related research has focused purely on either performance or power. Our work shows performance optimization adds to power, while 100% glitch reduction is not possible without impacting performance. Therefore it is important to achieve an appropriate compromise between the two. Furthermore, better PDE designs are motivated by putting PDE power overhead in perspective with total dynamic power consumption before and after glitch reduction: while

there is potential for good savings, a power-efficient PDE is crucial to the attractiveness of glitch reduction.

Part of the work done in this thesis has been accepted as a conference paper [13].

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 introduces basic concepts, including a brief overview of FPGA architecture, CSS, DP and activity estimation techniques, and GlitchLess. Chapter 3 describes the modifications to ACE. Chapter 4 describes the architecture changes and gives a brief overview of the algorithm. A detailed algorithm description is presented in Chapter 5. Chapter 6 gives detailed results and discussion, and Chapter 7 concludes the thesis and presents possible future work.

Chapter 2

Background

This chapter first presents the basics of a synchronous circuit, followed by a discussion of the clock skew scheduling (CSS) technique to improve performance. Past solutions and optimizations of CSS applied to Field-Programmable Gate Arrays (FPGAs) are described in detail. Delay padding, a useful extension of CSS that currently applies only to ASICs, is presented. The second part of the chapter presents concepts related to dynamic power reduction via glitch elimination. Discussion on activity estimation, power calculation and GlitchLess [19] is presented.

2.1 Synchronous Circuit and Clock Skew

A synchronous circuit is made up of blocks of combinational logic in between pairs of sequential elements (SEs) (denoted by R_i) connected by a common, periodic clock source. An example is shown in Figure 2.1. Each cycle, an incoming clock edge triggers R_j, which releases a data signal that travels through a block of combinational logic, and the computation result is stored in R_k. These two SEs and the combinational path form a local data path, or a pipeline stage. The entire circuit may be referred to as a pipeline, with each incoming data signal moving from one stage of the pipeline to the next according to the clock. In research literature, a common type of SE used is the positive clock edge triggered D flip flop (FF). Another type of SE is a flow-through latch, where changes at

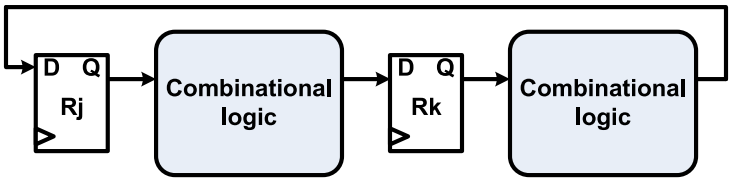

Figure 2.1: Example Synchronous Circuit

the input are immediately transferred to the output for the duration of the duty cycle. To determine the duty cycle of the latch's clock input, more timing constraints are needed to satisfy both clock edges of the duty cycle, resulting in a more complex problem. Therefore, FFs will be used throughout the thesis.

An important figure of merit of synchronous circuitry is the maximum operating frequency or, equivalently, the minimum clock period. A smaller period in between FFs requires a smaller data delay from FF output to input. Theoretically, the minimum period (P) is the maximum local data path delay (D_{max}) in the circuit plus the setup time required for stable register operation (T_{setup}). This is known as a setup-time constraint (Eq. 2.1).

$$P \geq T_{setup} + D_{max} \tag{2.1}$$

A violation of Eq. 2.1 will result in a *zero-clocking condition* [14] or a setup-time violation, where data from FF_i reaches FF_j too late relative to the next clock edge, and no new data is clocked to the next pipeline stage.

The clock is distributed in the circuit as a tree network, and it can limit the theoretical performance of synchronous circuits. It is often the largest net in the circuit, connecting to every FF in the circuit. Therefore, a clock signal may have to travel a long distance to reach FFs that are far away. Consequently, the clock's load capacitance due to wire length is often the greatest of all nets. These factors can cause a difference in the arrival time, or clock skew, of the clock signal to FFs at different parts of the circuit [15, 27]. Usually,

circuit designers take great efforts to reduce this clock skew as much as possible.

After accounting for skew at individual FFs, the resulting setup-time constraint is:

$$T_j - T_i \geq T_{setup} + D_{max}(i,j) - P \tag{2.2}$$

where T_i and T_j are clock arrival time at FF_i and FF_j, respectively. $D_{max}(i,j)$ is the maximum combinational delay between FF_i and FF_j. Clock skew gives rise to another possible circuit failure called the double-clocking condition, which is a type of hold-time violation:

$$T_i - T_j \geq T_{hold} - D_{min}(i,j) \tag{2.3}$$

where T_{hold} is the register's hold time, and $D_{min}(i,j)$ is the minimum combinational delay between FF_i and FF_j. Hold time violation occurs when the next data reaches FF_j too early relative to its next clock edge (due to clock skew), thereby overwriting the old data it was to capture. This will result in the old and new data being clocked into a stage during the same clock cycle, hence the name *double-clocking* [14].

Finally, process variation also makes it difficult to control device parameters such as channel length, width, dopant concentrations and gate thickness. Therefore, the threshold voltage and delay of logic gates may vary from chip to chip [8] or even between different gates on the same chip. This adds uncertainty into the above setup/hold time constraint equations. Often, this uncertainty is modeled by adding a timing margin (also called a guard band) M to the equations.

2.2 Clock Skew Scheduling

In 1990, Fishburn [14] proposed to use clock skew as a resource for improving performance, instead of treating it as an unavoidable burden. For example, consider the circuit in Figure 2.2. Assuming zero setup/hold times, a zero-skew clock network means the circuit has a minimum period of 14ns. If a skew of 4ns is applied to FF_B, the circuit is able to operate at a minimum period of 10ns. This effect can be viewed as time borrowing by shortening the effective delay of long paths, at the expense of increased delay for short paths. Indeed, the path from FF_B to FF_C now has an effective delay of 10ns from the clock edge.

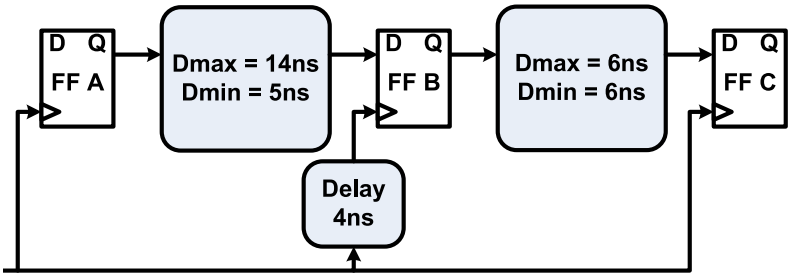

Figure 2.2: Intentional Clock Skew

The optimization problem is:

Minimize P, subject to:

$$|T_i| < P$$

$$T_j - T_i \geq T_{setup} + D_{max}(i,j) - P$$

$$T_i - T_j \geq T_{hold} - D_{min}(i,j)$$

This is a Linear Programming (LP) problem, and can be solved by an LP solver.

An issue with this scheme is process variation. In Figure 2.2, a 10ns period puts both

local paths on the verge of violating the setup time constraint. The uncertainty in gate delays and clock skews can cause zero clocking to occur. To fix this, a fixed amount of slack, or safety margin, is added to all local paths, at the expense of increased P [11]. The final LP problem is as follows:

Given P, maximize M, subject to:

$$|T_i| < P$$

$$T_j - T_i \geq T_{setup} + D_{max}(i,j) - P + M$$

$$T_i - T_j \geq T_{hold} - D_{min}(i,j) + M \tag{2.4}$$

The safety margin compensates for process variation and allows T_i, T_j and the path delays to vary by M in total without violating the constraints.

2.2.1 Solving CSS With Graph Theory

The CSS problem can be solved more efficiently using graph theory [10], and is demonstrated by [11]. In [10], a difference constraint is defined as a linear inequality in the form $x_j - x_i \leq b$, where b is a constant. For a set of difference constraints given in Eq. 2.4, a directed graph $G(V, E)$, called the constraint graph, may be constructed where vertex v_i corresponds to T_i, and edge weights correspond to the right hand side of the constraint equations. It is given that the values of v_i form a solution set for Eq. 2.4 if there are no negative weight cycles in $G(V, E)$.

To find the minimum achievable period, a binary search is performed between upper and lower bounds:

$$P_{max} = \max_{q=(i,j)\in G(V,E)} \{T_{setup} + D_{max}(i,j) + M\}$$
$$P_{min} = \max_{q=(i,j)\in G(V,E)} (\{T + T_{hold} + D_{max}(i,j) - D_{min}(i,j) + 2M\} \quad (2.5)$$

where q is an edge in the constraint graph, P_{max} is the largest local path delay in the circuit plus the safety margin, and P_{min} is determined from equating the setup/hold constraints in Eq. 2.4 such that both are satisfied simultaneously.

In each binary search iteration, the set of constraints in Eq. 2.4 is tested using graph theory. The goal is to determine if there exists a suitable clock schedule to satisfy a given P. A single-source shortest path algorithm can be used on $G(V, E)$ to find a solution to each T_i such that Eq. 2.4 for each pair of FF is satisfied, and that no negative weight cycles occur. The Bellman-Ford algorithm [10] is a suitable algorithm for this problem. A virtual vertex v_o is connected to primary input/output nodes to transform the graph into a single-source graph, and all other vertices contain T_i, the shortest-path weights from v_o.

Within the bound given by Eq. 2.5, the Bellman-Ford algorithm is used for each iteration of the binary search, until the bounds are ε apart from each other, as defined by the user.

1: **while** $(P_{max} - P_{min}) > \epsilon$ **do**
2: $P = (P_{min} + P_{max})/2$
3: **if** $G(V, E)$ has a positive cycle **then**
4: $P_{max} = P$;
5: **else**
6: $P_{min} = P$;
7: **end if**
8: **end while**

Figure 2.3: Binary Search for Optimum Period (from [11])

2.2.2 Solving CSS with Permissible Range

Given a pair of setup/hold constraints for a local path, the permissible range is the range of values ($T_{skewij} = T_i - T_j$) for which the setup/hold constraints remain satisfied. Neves and Friedman [24] solves the CSS problem by using a binary search to determine the permissible range of all local paths, subject to user-specified minimum value. For reconvergent paths (two or more local paths with a common source and sink) or cycles (feedback loops that begin and end at the same FF), the intersection of the permissible ranges is taken to form the effective permissible range. T_{skewij} is then chosen to be the middle of the effective permissible range. This approach leaves as much safety margin as possible on either side of the chosen skew to tolerate the unknown process variation effects.

2.2.3 Exact CSS Solution

The authors of [4, 32] state that the CSS problem can be solved by finding the minimum mean cycle in the constraint graph. The authors of [32] gives the complete algorithm description for the solution. A Z-cycle is defined as a cycle containing at least one hold-time constraint type edge. Finding the minimum period is equivalent to finding the minimum Z-cycle in the constraint graph. The work claims a polynomial runtime complexity.

Furthermore, the authors of [4] point out that it is much more expensive to generate the constraint graph (referred to as the sequential graph in [4]) than it is to compute the minimum mean cycle. They proposed an algorithm that solves the CSS problem while extracting only part of the graph, starting at the most timing-critical region of the circuit. The work claims that the runtime is reduced to 5.8% of the original, and that only 20% of the sequential circuit needs to be extracted.

In this thesis, the binary search method in [11] is used instead of the exact CSS solution method, since it is easier to adjust post-CSS skews to discrete delays (required by the

programmable nature of FPGAs) in the Bellman-Ford framework, and it is also easier to implement.

2.3 CSS for FPGAs

The discussion so far only considers CSS for ASICs. FPGA technology is reprogrammable alternative to ASICs. They allow fast turnaround time, and they are very popular for rapid prototyping and numerous other low-volume applications. This section presents a brief overview of the FPGA architecture and CAD design flow, followed by existing CSS techniques for FPGAs.

2.3.1 FPGA Architecture

The basic building block of an FPGA is called a Basic Logic Element (BLE). Each BLE contains a k-input, single-output Look Up Table (LUT) and a FF as shown in Figure 2.4. The purpose of the LUT is to implement an arbitrary logic function of up to k inputs. If the multiplexer is selected to bypass the FF, the BLE will be used as combinational logic. Otherwise, the BLE will act like the end of a pipeline stage in sequential logic.

BLEs are grouped together into logic clusters [6], which are also referred to as configurable logic blocks (CLB). A CLB may contain N BLEs grouped together as shown in

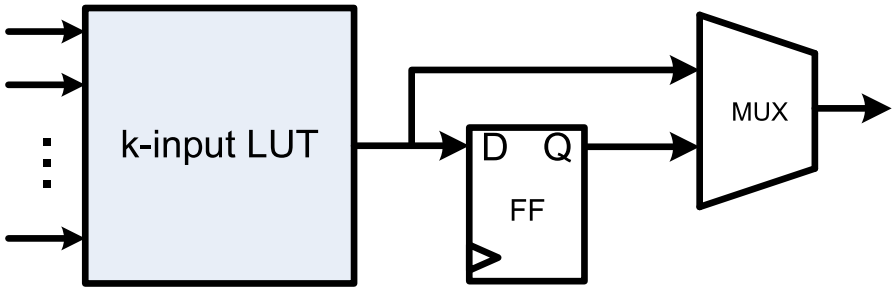

Figure 2.4: Basic Logic Element

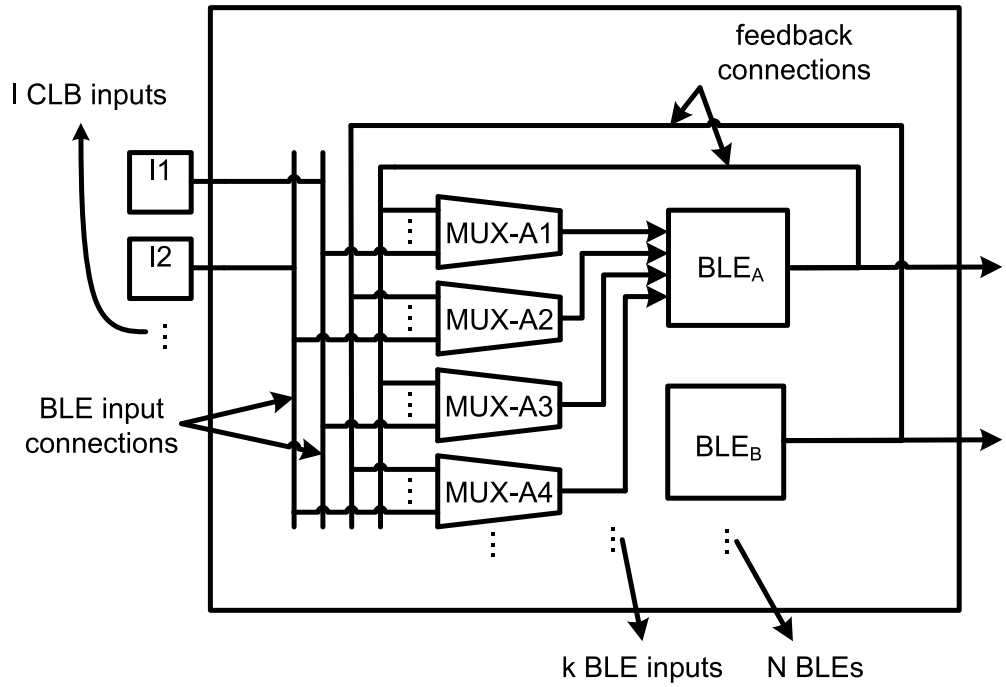

Figure 2.5: Configurable Logic Block

Figure 2.5, sharing I distinct inputs. A BLE inside the CLB can choose either one of the CLB inputs, or any feedback signal from one of the BLE outputs in the same cluster via the fast local routing. Circuit components that are closely connected with each other can take advantage of the fast local routing to improve overall speed [6, 35].

A generic FPGA is made of CLBs (L block) grouped together in a grid like fashion, shown in the top part of Figure 2.6. CLBs are surrounded by programmable routing fabric channels, which are illustrated in more detail in the bottom part of Figure 2.6. The versatility of an FPGA comes from the ability of every CLB being able to arbitrarily connect to any other CLB via the programmable switches (S block) and connections (C block). The number of wires in the routing channel is called the channel width.

The components in a FPGA are also susceptible to process variation. However, circuits implemented in a FPGA operate at much slower frequency than ASICs, making the pro-

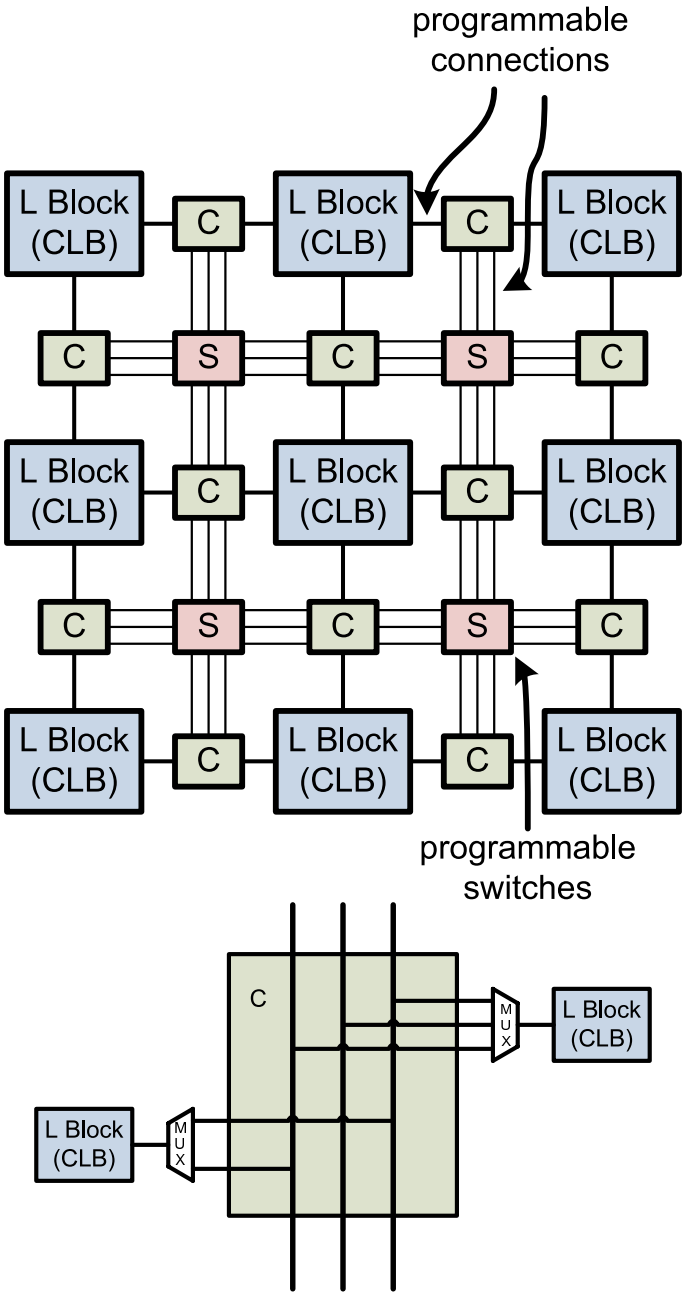

Figure 2.6: An Example FPGA

cess variation effect less significant. However, FPGAs are becoming faster with each new manufacturing process shrink, making process variation a more noticeable issue.

In this thesis, unless otherwise mentioned, the following major architectural parameters are used:

- k (LUT size): 4 and 6
- N (CLB size): 10
- I (Inputs per CLB): 22 for k=4 and 33 for k=6
- F_C input (fraction of routing wires each CLB input pin can connect to): 0.2
- F_C output (fraction of routing wires each CLB output pin can connect to): 0.1
- Segment Length (number of CLBs spanned by a routing wire): 4
- Switch Type: uni-directional buffered MUXes.
- Rmetal (resistance of routing wire per CLB length): 56.122Ω [3] for 65nm technology, based on an estimated 125μm CLB length
- Cmetal (capacitance of routing wire per CLB length): 21.13425fF [3] for 65nm technology, 125μm CLB length
- Other detailed information are obtained from the iFAR repository [2]

2.3.2 FPGA CAD Flow

To transform a circuit design onto a FPGA, the first step, technology mapping, maps user designed logic gates into k-input LUTs and FFs. A clustering algorithm then packs these into CLBs with N BLEs. Closely connected LUTs are usually packed into the same CLB

to take advantage of the fast local routing. The result of this step is a netlist file that describes which LUTs and FFs are inside each CLB. This file is used for the next step, placement, to map the CLBs onto physical locations on the FPGA chip. The VPR tool [6] is a very popular tool in academic research. VPR 5.0 is the latest version. It is used in this thesis and shall be simply referred to as VPR. It uses a placement technique called simulated annealing, which starts with a random placement CLBs scattered on the FPGA. Then, two random CLBs' positions are swapped, the cost of the placement is recalculated, and the swap is kept if the cost is lower. If the swap causes a cost increase, it may also be kept depending on a probability (called the temperature of the anneal process) that slowly decreases with time. This process is repeated until the temperature reaches a low point specified by the user. During annealing, the placement cost is usually a function of critical path delay and the interconnect area the circuit requires. The last step is called routing, which connects CLBs together via the routing resource. Two common metrics optimized by the router are channel width and critical path delay. VPR uses an architecture file that contains information described in the previous section to do placement and routing.

In this thesis, timing-driven placement and routing are used, with a fixed channel width of 104 as specified in the architecture files obtained from [2]. Clock skew scheduling is performed after routing, the last stage of the regular FPGA CAD flow.

2.3.3 Clock Skew Scheduling Techniques for FPGAs

Singh and Brown [28] use multiple global clock lines (L in total) in the FPGA to distribute skews to FFs, as shown in Figure 2.7. The work distributes multiple copies of the same clock with precisely ranged phase shifts, generated by on-chip PLLs. The optimization algorithm is similar to that presented in [14], but it must select a set of L distinct discrete skew values to realize the best possible schedule. Therefore, a discrete version of the

Bellman-Ford algorithm is used.

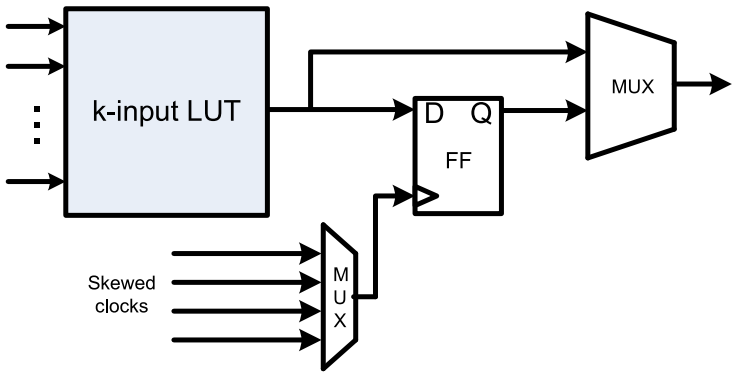

Figure 2.7: CSS with Skewed Global Clocks

An alternative approach by Yeh et al. [36] uses a single global H-tree with ribs on the H-tree for local routing as shown in Figure 2.8. The far right picture in Figure 2.8 shows the detailed local routing, where programmable delay elements (PDE) are inserted into branching points of the clock tree. Under this architecture, the clock signal goes through a trail of PDE nodes (from R to d in Figure 2.8) before arriving at each FF node. This levelized structure provides more choices for skew values than Singh and Brown's approach [28]. Since the max amount of delay a PDE can provide is fixed, an additional constraint must be satisfied when solving the optimization problem:

$$\Upsilon_{ij} \leq s_j - s_i \leq \Upsilon_{ij} + \zeta_i \tag{2.6}$$

Where Υ_{ij} is the interconnect delay between two PDE nodes or between a FF node and a PDE node at the end of the trail that provides its clock, $s_j$and s_i are clock arrival times of the two nodes, and ζ_i is the amount of delay provided by the PDE. For example, consider Figure 2.8. The clock arrival time of FF2 (s_i) is equal to the sum of PDE-d's arrival time (s_j), the delay it provides (ζ_d), and the delay of the wire (Υ_{ij}) between it and FF2.

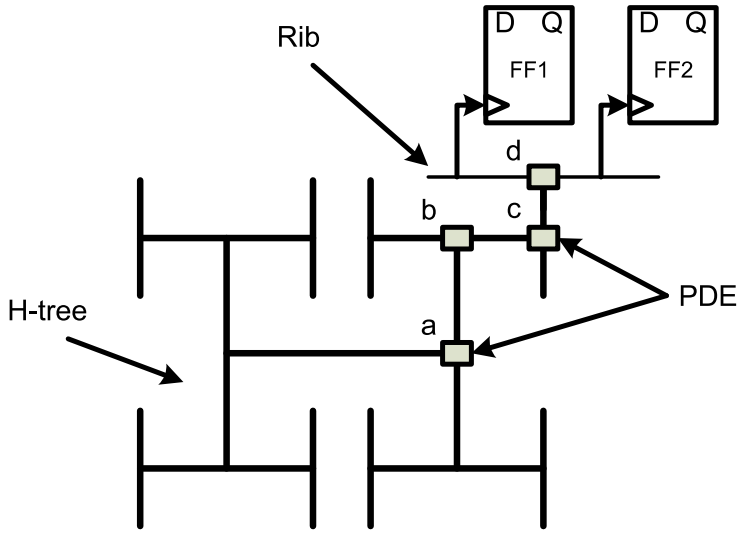

Figure 2.8: Global H-tree with Ribs for Local Routing

A third architecture uses the same spine and ribs clock network as [36], but inserts PDEs only at the local ribs [31]. Shown in Figure 2.9, this method produces 4 skewed version of the global clock for each row. In addition, a statistical timing model is used to express maximum and minimum path delays as Gaussian variables, and k is the user-defined uncertainty factor to account for path delay variations.

$$D_{max}(i,j) = \mu_{max} + k \cdot \sigma_{max}$$

$$D_{min}(i,j) = \mu_{min} - k \cdot \sigma_{min} \tag{2.7}$$

2.4 Delay Padding (DP) for CSS

The setup/hold constraints can limit the range of skews that can be assigned to SEs, and therefore the smallest obtainable period. In Eq. 2.2 and 2.3, larger D_{max} and smaller D_{min}

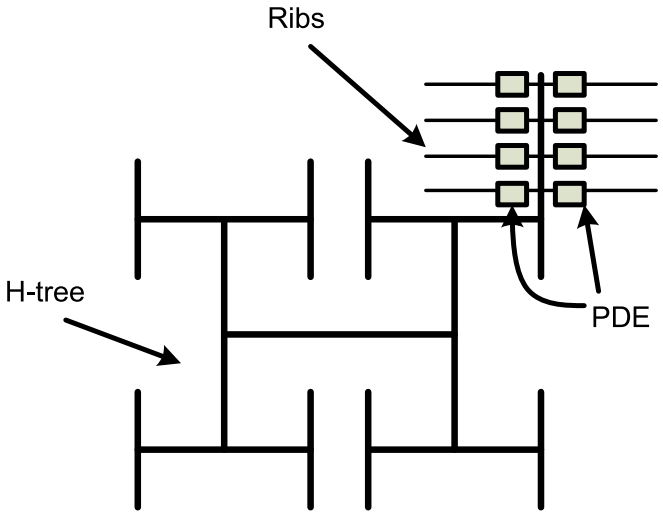

Figure 2.9: PDE Insertion at Local Ribs (from [31])

will decrease the permissible range of assigned skews. Nothing can be done to decrease $D_{max}(i,j)$, but an increase in $D_{min}(i,j)$ will widen the permissible range, allowing skew assignment to be more flexible. This short-path optimization effectively reduces hold time violations, allowing a smaller period. This section will describe several existing approaches for delay padding for ASICs. To our knowledge, delay padding has not yet been applied to FPGAs.

2.4.1 Delay Insertion using Linear Programming

Taskin and Kourtev [33] proposed to insert delays into signal paths. The authors show for reconvergent paths that are timing-critical, a smaller minimum period is achievable by decreasing the difference between the maximum and minimum path delays of each reconvergent path. A new set of setup/hold constraints are formulated in Eq. 2.8, where $D_{max}(i,j)$, $D_{min}(i,j)$ and I_{Mij}, I_{mij} define uncertainty bounds for each reconvergent path from FF_i to FF_j, and the delay provided by the inserted delay element, respectively.

Since there are three unknowns to solve for each constraint, Eq. 2.8 no longer forms a set of difference constraints that can be solved by graph theory, and LP is used assuming continuous skews and delays are available.

Minimize P, subject to:

$$I_{Mij} \geq I_{mij}$$

$$T_j - T_i \geq T_{setup} + D_{max}(i,j) - P + I_{Mij}$$

$$T_i - T_j \geq T_{hold} - D_{min}(i,j) - I_{mij} \tag{2.8}$$

2.4.2 Race Condition Aware (RCA) Clock Skew Scheduling

Huang and Nieh [17] uses an iterative method to find a clock skew and delay padding solution. The overall algorithm is in Figure 2.10.

1: (G_{DEL}, P_{RCA}, S_{DEL}) = Relax_Hold(G_{OCSS});
2: (G_{INS}) = Parameter_Assign(G_{DEL}, S_{DEL});
3: (G_{RCA}, S_{RCA}) = Parameter_Minimization(G_{INS}, P_{RCA});
4: **return** (G_{RCA}, P_{RCA}, S_{RCA});

Figure 2.10: RCA Algorithm (from [18])

The first part of the overall algorithm, Relax_Hold, is shown in Figure 2.11. The original circuit's constraint graph G_{OCSS} is the input. During iteration k, a set of skews ($S_{D(k)}$) and a period ($P_{D(k)}$) are determined using the binary search method in [11]. Then, the constraint graph is stripped of any critical hold-time edges (H-edges). A hold-time edge (essentially a hold-time constraint in Eq. 2.4), is a critical hold-time edge if both inequalities in Eq. 2.4 become equalities, forming a critical cycle. CSS is then performed again to determine a lower period, and the process is repeated until no further performance

1: k=0; $G_{D(k)} = G_{OCSS}$;
2: derive $S_{D(k)}$ and $P_{D(k)}$ with respect to $G_{D(k)}$;
3: **repeat**
4: obtain $G_{D(k+1)}$ by deleting all the actual critical H-edges in $G_{D(k)}$ with respect to $S_{D(k)}$;
5: derive $S_{D(k+1)}$ and $P_{D(k+1)}$ with respect to $G_{D(k+1)}$; k++;
6: **until** ($G_{D(k)} == G_{D(k)}$);
7: $G_{DEL} = G_{D(k-1)}$; $P_{RCA} = P_{D(k-1)}$; $S_{DEL} = S_{D(k-1)}$;
8: **return** (G_{DEL}, P_{RCA}, S_{DEL});

Figure 2.11: Detailed Relax_Hold Algorithm(G_{OCSS}) (from [18])

optimization is obtainable. At the end of Relax_Hold, the optimum set of skews (S_{DEL}), period (P_{RCA}) and constraint graph excluding deleted critical hold-time edges (G_{DEL}) are produced.

During Parameter_Assign, each deleted edge is given a padded delay of $padding = T_j - T_i - D_{min}(i,j) + T_{hold}$, the amount of delay required to satisfy the hold-time constraint. The result is G_{INS}, a constraint graph containing optimum skews and padded delays to satisfy the optimum period, P_{RCA}. The purpose of the last step, Parameter_Minimization, is to minimize the padded delays using binary search and graph theory. For each deleted edge, the padded delay is the binary search variable in the range $[0, T_j - T_i - D_{min}(i,j) + T_{hold}]$. During each iteration, the same set of constraints used during CSS is used again to determine the skews T_i and T_j. There are two differences to the CSS problem:

1. The objective is to minimize the padded delays with a fixed period, whereas the objective of CSS is to minimize the period.

2. Parameter_Minimization uses constraints from G_{INS}, whereas CSS in Relax_Hold uses G_{OCSS}. Although these two graphs have different edge weights and vertex (clock skew) values, the edges from these two graphs correspond to the same physical paths in the circuit.

The final output of the algorithm includes the period and skew schedule. One advantage of this approach is that it can be easily integrated with the graph theory based binary search approach in [11], which is more efficient when the skew values become quantized, and is easier to implement.

2.5 Activity Estimation

To obtain accurate power estimations, a good method to calculate activity is needed. The ACE tool [20] is one such approach. This section will summarize concepts related to activity, followed by a description of the ACE algorithm.

2.5.1 Terminology

There are three concepts that define the switching characteristics of a circuit. *Static probability* (P_1) is the probability that a signal is in the high (1) state. The *switching probability* (P_s) is the probability for a signal to change steady state value (0 to 1 or 1 to 0) at the end of a clock cycle. These transitions are a result of circuit operation, and are called *functional logic transitions*. *Switching activity* (A_S) is the probability of a signal going from 0 to 1 or 1 to 0 during each clock cycle. In this thesis, switching activity will be simply referred to as activity. For a logic gate, the activity of its output is the combined result of two kinds of logic transitions: functional and glitch. Glitching results from input data signals arriving at different times during the period, causing the output to fluctuate before settling down.

2.5.2 Simulation-Based Activity Estimation

ACE-2.0 [20] computes switching activities using logic simulation using pseudo-random input vectors and net delays generated by VPR. This is the most accurate method to obtain both functional and glitch activity for any arbitrary placement and routing solution

```
1: for all vector ∈ vector array do
2:     update_primary_inputs(circuit, vector);
3:     propagate_events(circuit);
4:     update_flip_flops(circuit);
5: end for
```

Figure 2.12: Main ACE Algorithm

produced for any FPGA architecture, and will be used as a basis in this thesis. The name "ACE" shall be used throughout this thesis to refer to this simulation-based technique.

The main ACE algorithm is shown in Figure 2.12. For every input vector that represents a clock cycle, the primary inputs are loaded, the entire circuit is evaluated, and FF outputs are updated for the next vector. The main simulation routine, propagate_events, is shown in Figure 2.13. It uses event-driven simulation, where each event represents a change (from 0 to 1 or 1 to 0) of an input signal to some node in the circuit. Events are queued in a list, sorted according to the event time relative to the start time of the cycle, and are examined in order by a loop (line 2). The event's signal value is used to evaluate the node's output value (line 4) via SIS [1], a logic synthesis tool. For example, if a 0 to 1 transition is detected, the time of the transition is compared to the time of the last transition from 1 to 0 (Time0(n), line 8). This detects the width of the most recent 1→0→1 pulse. If the pulse width is lower than a threshold (MIN_PULSE_WIDTH), then it is assumed that the pulse will get filtered out by a standard length-4 segment of routing track, and the toggle count for node n's output is decremented (line 9). In the case of a long pulse, the transition will be pushed onto the event queue (line 22).

2.6 Dynamic Power Calculation

Dynamic power is defined by $P = \alpha \times C \times V_{dd}^2 \times f$, where α is switching activity, C is capacitance, V_{dd} is supply voltage and f is operating frequency. For 65nm technology, V_{dd}

```
1: event = queue_pop (queue);
2: while event != NULL do
3:    n = event→fanout_node;
4:    value = evaluate_logic(n, event→value);
5:    if Value(n) != value then
6:       if value == 1 then
7:          //transition from 0→1
8:          if event→time > MIN_PULSE_WIDTH && event→time − Time0(n) <
          MIN_PULSE_WIDTH then
9:             Num_Transitions(n) −= 2;
10:         else
11:            Time1(n) = event→time;
12:         end if
13:      else
14:         //transition from 1→0
15:         if event→time > MIN_PULSE_WIDTH && event→time − Time1(n) <
          MIN_PULSE_WIDTH then
16:            Num_Transitions(n) −= 2;
17:         else
18:            Time0(n) = event→time;
19:         end if
20:      end if
21:      Value(n) = value;
22:      push_event (queue, n, event→time, value);
23:      Num_Transitions(n)++;
24:   end if
25: end while
```

Figure 2.13: ACE: propagate_events(circuit)

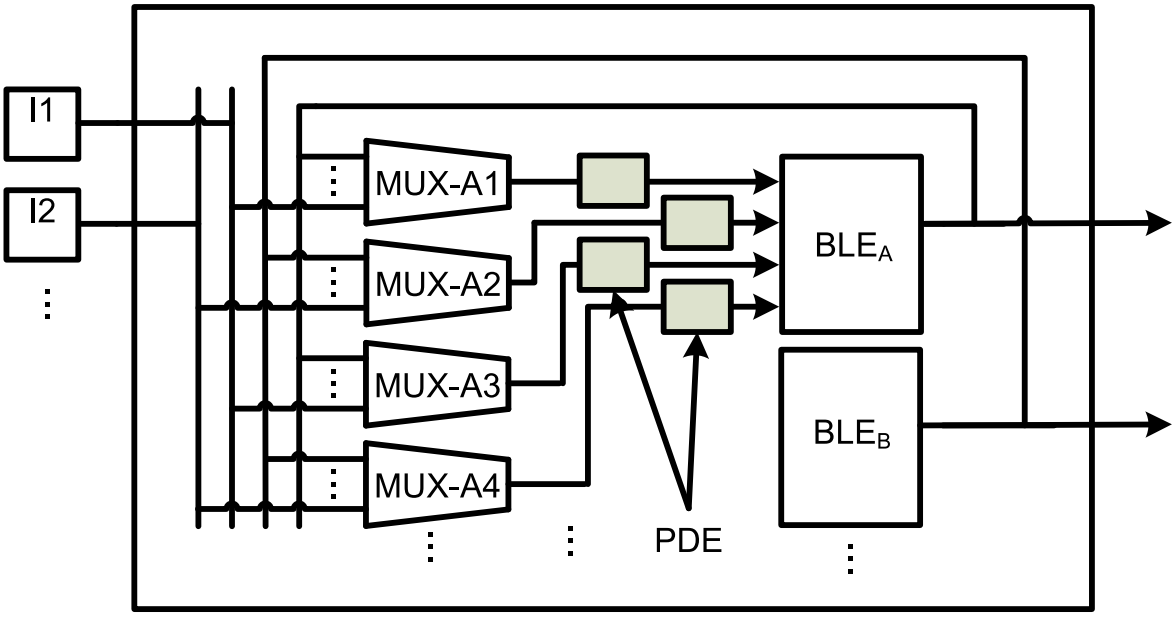

Figure 2.14: Architectural Modification for GlitchLess

is 1V. The power figure we will refer to in this work is the power per operation, namely $P_{op} = \alpha \times C$. A power unit P_{op} is defined as 1 femto-Farad of capacitance switching once per clock cycle ($\alpha = 1$).

2.7 Glitch Reduction

GlitchLess reduces glitching by delaying early arriving signals to prevent the output from fluctuating [19]. To realize this, PDEs are added to LUT inputs according to various schemes outlined in [19], and the basic technique in shown in Figure 2.14. Other work done to reduce glitching include [12], which uses routing techniques, and [7], which proposes a new glitch-driven technology mapping tool.

The first routine of the GlitchLess algorithm, calc_needed_delays, is shown in Figure 2.15. It does a timing analysis for the circuit based on the net delays produced by

VPR. The circuit is represented using a graph, with nodes representing LUTs and FFs, and edges representing the delay from node to node. For each node, a quantity called needed_delay is calculated for each fanin path that represents the amount of delay that must be added to the LUT input to ensure all input signals arrive at the same time. Then, config_LUT_input_delays function (Figure 2.16) will assign discrete delays to the LUT inputs. To account for variation, all delays are shortened by an amount d so critical path will not be increased.

```
1: for all node n ∈ circuit do
2:    //in topological order beginning from the primary inputs
3:    Arrival_Time(n) = 0.0;
4:    for all fanin f ∈ n do
5:       if Arrival_Time(f) + Delay(n, f) > Arrival_Time(n) then
6:          Arrival_Time(n) = Arrival_Time(f) + Fanin_Delay(n, f);
7:       end if
8:    end for
9: end for
10: for all node n ∈ circuit do
11:    //in topological order beginning from the primary inputs
12:    for all fanin f ∈ n do
13:       Needed_Delay(n, f) = Arrival_Time(n) − Arrival_Time(f) − Fanin_Delay(n, f);
14:    end for
15: end for
```

Figure 2.15: GlitchLess: calc_needed_delays(circuit) (from [19])

```
1: for all LUT n ∈ circuit do
2:    count = 0;
3:    for all fanin f ∈ n do
4:       if Needed_Delay(n, f)>min_in && Needed_Delay(n, f)≤max_in && count<num_in
         then
5:          Needed_Delay(n, f) −= min_in * floor(Needed_Delay(n, f)/min_in);
6:          count++;
7:       end if
8:    end for
9: end for
```

Figure 2.16: GlitchLess: config_LUT_delays(circuit, min_in, max_in, num_in) (from [19])

Chapter 3

Glitch Generation Modelling

This chapter presents a modified approach for glitch power estimation via vector simulation-based, event-driven activity calculation. The ACE framework is used as a basis. The purpose of the modification is to obtain a more accurate model for power estimation due to glitch generation. Throughout this thesis, all benchmarks are simulated with 5000 pseudo-random input vectors (clock cycles).

3.1 Introduction and Motivation

The ACE tool filters out fluctuations of very short pulse widths since the routing resource's parasitic capacitance can dampen them out. Originally, the maximum pulse width that can be filtered out by a single stage of length-4 routing segment was determined by HSPICE simulation. A glitch longer than this threshold is assumed to go on indefinitely, otherwise it is assumed to consume no power. Neither of these assumptions is true in reality: as long as the pulse width of a glitch is below a certain threshold (short glitch), it will be *gradually* filtered out after propagating down a certain number of wire segments. Glitches longer than the threshold may propagate indefinitely. To take this into account, we first simulate routing nets of various lengths in Cadence Spectre (an accurate alternative to HSPICE) to obtain glitch behavior, then modify ACE to determine a pulse width histogram, and finally combine the results with VPR to calculate power.

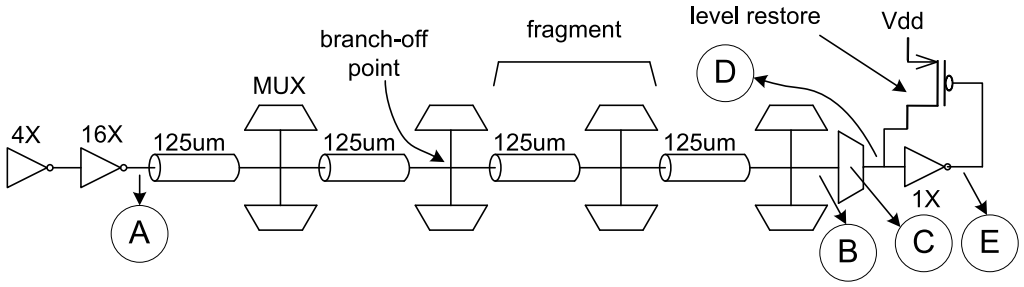

Figure 3.1: Length-4 Wire Stage Overall View

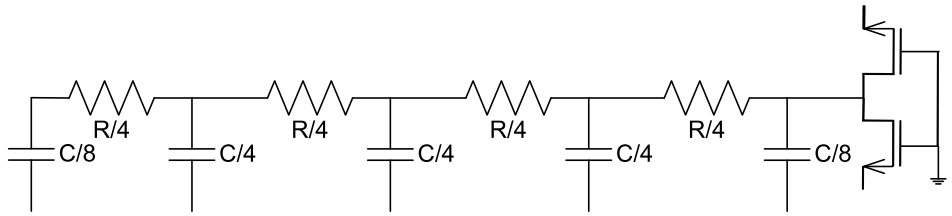

Figure 3.2: Fragment Detail View

3.2 Cadence Simulation

Cadence Spectre simulations are done for glitches of varying pulse widths travelling down various lengths of routing nets. A standard length-4 segment, called a stage, is modelled as in Figure 3.1, which contains driving buffers and 4 individual fragments of wire each spanning the length of one CLB. This CLB length is 125μm throughout the thesis. A pair of multiplexers are connected to the junction between fragments to indicate possible branch-off points. Each fragment is shown in Figure 3.2, where each multiplexer at the branch-off point is simulated as a minimum sized NMOS in cutoff mode. For accuracy, each fragment is modeled as a 4-piece π model [34].

A short glitch of a particular pulse width, travelling down a routing track will have its pulse width being gradually decreased by the routing resource's parasitic RC effect. Consider a 0-1-0 pulse travelling down a single stage: The rising edge needs a certain time for the output to rise to the supply voltage level because the capacitance need to charge

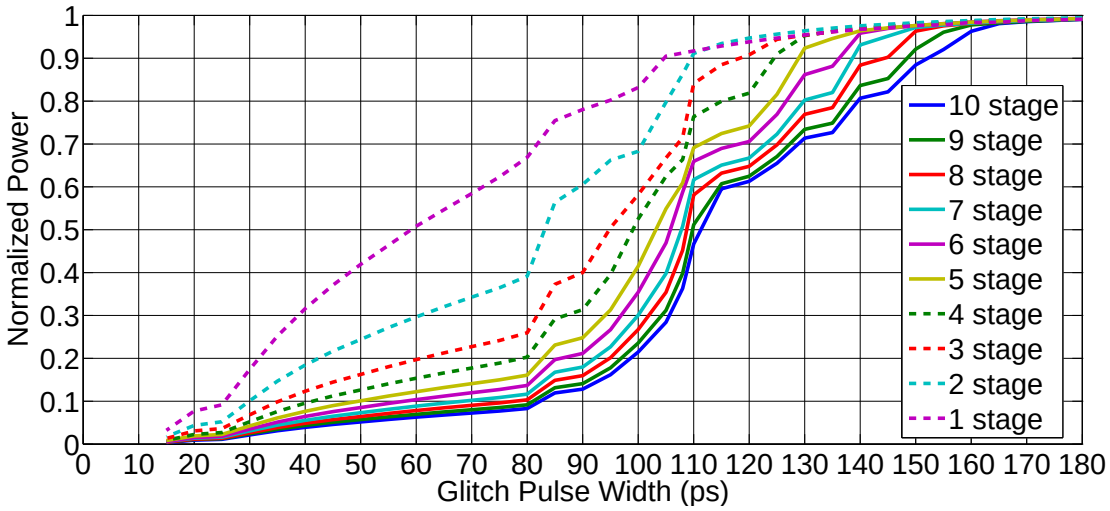

Figure 3.3: Segment Length to Power Lookup (65nm Technology)

up. If the pulse width is too short, the output may not have enough time to reach the supply voltage before the falling edge starts, which effectively reduces the peak voltage of the pulse created at the output.

As a short glitch travels down a routing track of n stages, the decreasing pulse width causes power consumption to decrease because the routing capacitance does not fully charge. The power consumed by a short glitch can be expressed as a percentage normalized to that consumed by a long glitch propagating down the same n stages. Simulation results for 1 to 10 stages are summarized in Figure 3.3. A converging trend is observed as the lines get closer together for increasing number of stages. Therefore, it is assumed that any net longer than 10 stages (wire segments) will behave the same as a 10-stage net.

To pass this information to VPR, a lookup table of percentages are created from Figure 3.3, whose x-axis is divided into bins. Each bin is 5ps wide. For each stage length, the power percentage for each bin is calculated as an average of its lower and upper boundary values. For example, in Figure 3.3, a 42ps pulse travelling down 1 stage will have a percentage of

(0.3+0.35)/2=32%. Any short glitch longer than 180ps consumes nearly the same power as a long glitch. Therefore, 180ps is used as the upper threshold to determine if a glitch can propagate indefinitely or not. Furthermore, any glitch shorter than 15ps consumes nearly zero power, so 15ps is used as the lower threshold to determine if a glitch should be ignored.

3.3 Glitch Binning Algorithm Description

The majority of the ACE program is unchanged from the last chapter, with the exception of the propagate_events routine. This section will detail the changes.

3.3.1 ACE Inputs

The program requires the following input files:

1. A BLIF file of the circuit to build circuit diagrams for timing analysis and logic evaluation.
2. A vector file containing pseudo-random input vectors for circuit primary inputs.
3. A net delay file produced by VPR, containing net delays for all nodes in the circuit, after routing is finished.

3.3.2 Event Propagation

Modifications to ACE include changes to the propagate_events routine to calculate glitch pulse widths, and to group glitches of different pulse widths into bins (for example, glitches ranging from 15ps to 20ps is bin #1, etc). The algorithm is outlined in Figure 3.5. Note the distinction between a *transition* and a *pulse*: a pulse (1-0-1 or 0-1-0) consists of two back-to-back transitions.

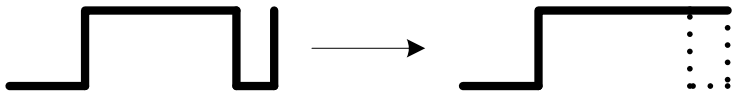

Figure 3.4: Glitch Filtering Effect

When a 0 to 1 transition is detected (line 6), and the width of the 1-0-1 pulse is below threshold (line 7), several things can happen. If the signal at the beginning of the cycle is 1, then this transition is the finishing edge of a pulse. Since the pulse width is too small, no action needs to be taken. If the signal at the beginning of the cycle is 0, then a 0-1-0 pulse precedes this transition. We need to "merge" it into the previous pulse as if this pulse has never happened, as shown in Figure 3.4. This is done by decrementing the glitch count (line 10 and 11).

If the 1-0-1 pulse width is above threshold, it is a glitch and should be accounted for, but only if the signal started with a value of 1 at the beginning of the cycle, because only then a 0-1 transition is the completing transition of a pulse. In other words, if the starting value is 0, then a 0-1 transition is always the starting edge of a pulse, so the glitch count should not be incremented yet. Lines 15 to 20 calculates the pulse width and increments the corresponding bin number. The case for a 0-1-0 pulse (starting from line 23) is similar.

3.3.3 ACE Output

ACE provides a single output text file containing the activity breakdown of all nodes in the circuit, with names taken from the BLIF file. For each node, P_1, P_S, A_S and total glitch activity summed over all nodes are printed, as well as the number of glitches in each bin.

```
1: event = queue_pop (queue);
2: while event != NULL do
3:   n = event→fanout_node;
4:   value = evaluate_logic(n, event→value);
5:   if Value(n) != value then
6:     if value == 1 then
7:       if event→time > MIN_PULSE_WIDTH && event→time − Time0(n) <
         MIN_PULSE_WIDTH then
8:         Num_Transitions(n) −= 2;
9:         if Prev_Value(n) == 0 then
10:          Num_Glitch(n)−−;
11:          Num_Glitch_Bin(n, Prev_Glitch_Bin(n))−−;
12:        end if
13:      else
14:        Time1(n) = event→time;
15:        if Prev_Value(n) == 1 then
16:          Num_Glitch(n)++;
17:          pulse_width = event→time - Time0(n);
18:          bin = (int)((pulse_width − MIN_PULSE_WIDTH)/BIN_WIDTH);
19:          Num_Glitch_Bin(n, bin)++;
20:          Prev_Glitch_Bin(n) = bin;
21:        end if
22:      end if
23:    else
24:      if event→time > MIN_PULSE_WIDTH && event→time − Time1(n) <
         MIN_PULSE_WIDTH then
25:        Num_Transitions(n) −= 2;
26:        if Prev_Value(n) == 1 then
27:          Num_Glitch(n)−−;
28:          Num_Glitch_Bin(n, Prev_Glitch_Bin(n))−−;
29:        end if
30:      else
31:        Time0(n) = event→time;
32:        if Prev_Value(n) == 0 then
33:          Num_Glitch(n)++;
34:          pulse_width = event→time - Time1(n);
35:          bin = (int)((pulse_width − MIN_PULSE_WIDTH)/BIN_WIDTH);
36:          Num_Glitch_Bin(n, bin)++;
37:          Prev_Glitch_Bin(n) = bin;
38:        end if
39:      end if
40:    end if
41:    Value(n) = value;
42:    push_event (queue, n, event→time, value);
43:    Num_Transitions(n)++;
44:  end if
45: end while
```

Figure 3.5: Modified ACE Routine: propagate_events(circuit)

3.4 Power Calculation

To calculate total dynamic power consumption, ACE output and Spectre simulation results are read into VPR as separate input files (detailed discussion about VPR modifications are presented in Chapter 4). For a glitch generated at the source node of a net in the circuit, the length and capacitance of the routing track for that net is determined with the VPR routing graph, the glitch activity for each bin is read from ACE, and the amount of glitching power can be calculated by multiplication of capacitance, glitch activity, and the percentage found in Fig 3.3 via indexing by net length and bin #.

There are other components that consume dynamic power, namely intra-CLB routing and MOSFETs that make up LUTs and SEs. The former is dominant because a feedback wire from LUT output to LUT input MUXes in the same CLB carries much more capacitance than the latter, which we neglect in our calculations.

3.5 Results and Discussion

The results from the original ACE, and those obtained from the new glitch binning algorithm, are compared in Table 3.1 for circuits produced by VPR. Units are P_{op} described in Chapter 2. All circuits are simulated using 5000 pseudo-random input vectors. A positive percentage difference means the original ACE underestimates glitching. The original ACE can underestimate glitching power as much as 48%, for k=4, and overestimate as much as 15% for k=6. Generally, original ACE underestimates glitch power for k=4 because arrival time differences for a smaller LUT tend to be smaller and get dropped (below threshold).

Our glitch power estimation can still be improved further. Glitch filtering creates two issues: glitch generation and propagation. The former is a glitch created at the output of a gate generated by the combined effect of its logic function and different input arrival times.

circuit	k = 4			k = 6		
	Bins	Original	% diff	Bins	Original	% diff
bigkey	913	471	48.4	560	629	-12.4
clma	3794	3407	10.2	2955	3303	-11.8
diffeq	136	129	4.9	63	58	6.7
dsip	698	512	26.6	574	557	3.2
elliptic	11607	10462	9.9	6408	6944	-8.3
frisc	1185	1096	7.5	1045	1088	-4.1
s298	5350	3906	27.0	4956	5585	-12.7
s38417	29292	19195	34.5	7036	8111	-15.3
s38584.1	10455	9246	11.6	4052	4395	-8.5
tseng	1334	1326	0.6	590	608	-3.2

Table 3.1: Glitch Power (P_{op}) of Original ACE and ACE with Binning

The latter is illustrated by the example in Figure 3.6. A glitch generated at the source node, BLE_A in CLB1, fans out to two sink nodes: BLE_B in CLB2 and BLE_C in CLB3. Short glitches become narrower as it travels along a routing segment, therefore the glitch will become narrower at the input of the fan out nodes. Also, the glitch travels a longer distance to BLE_C than it has to BLE_B, so the pulse width of the glitch at the input of BLE_C is narrower than that at the input of BLE_B. Therefore, the glitch pulse width at the output of BLE_C is narrower than that of BLE_B, and may become smaller than the lower threshold defined earlier. Proper glitch propagation modelling takes all of these into consideration.

Our work has a better estimate on glitch generation and power consumed by the routing that immediately follows a glitchy node, but it still lacks proper glitch propagation modelling. In particular, the new glitch binning algorithm does not shorten the width of the pulses as they are propagating through logic during the event-driven simulation. For a complete analysis, we need to account for the change of glitching activity on downstream nodes caused by glitch propagation, requiring VPR and ACE to be tightly integrated so

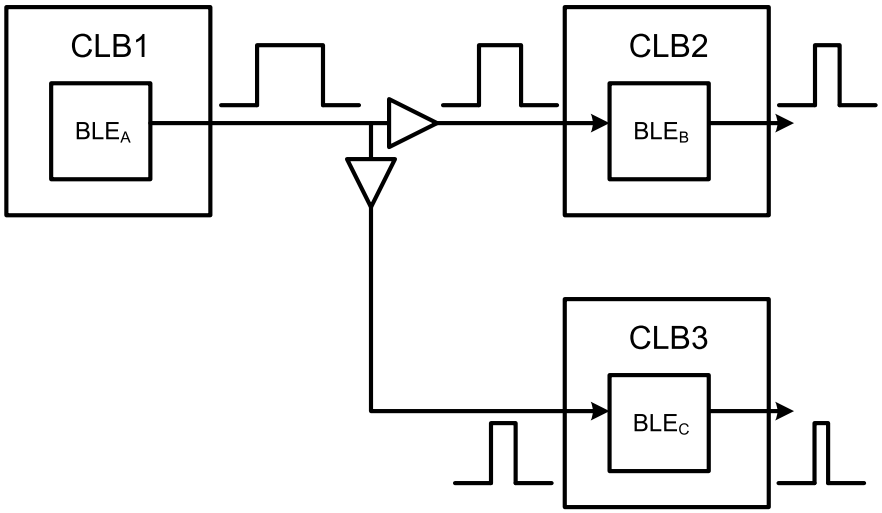

Figure 3.6: Glitch Propagation

logic evaluation and routing information can be obtained concurrently.

Chapter 4

Architecture and Algorithm Overview

A major contribution of this work is the proposal of a unified architecture change that can be shared by CSS, delay padding and GlitchLess, and the integrated tool flow. This section will detail this architecture as well as its adaptation by each of the 3 optimization steps. We assume that newer FPGAs, such as the Stratix III and Virtex 6, have 2 flip flops per LUT. A discussion of the tool flow, PGR algorithm and preliminary operations follow.

4.1 Architecture

4.1.1 Architecture for CSS and Delay Padding

Architecture changes are highlighted in Figure 4.1 with legends shown to distinguish optimization steps. CSS can be done by adding delay δ_A to FF_A. For delay padding, we use local rerouting within CLBs. The CLB input (solid arrow line) in Figure 4.1 goes to LUT_B originally. We reroute it (dash-dotted line) to unused FF_C in another BLE, then back to the original LUT. Properly adjusting the skew assigned to δ_C, any desired delay can be achieved provided there is enough slack for it.

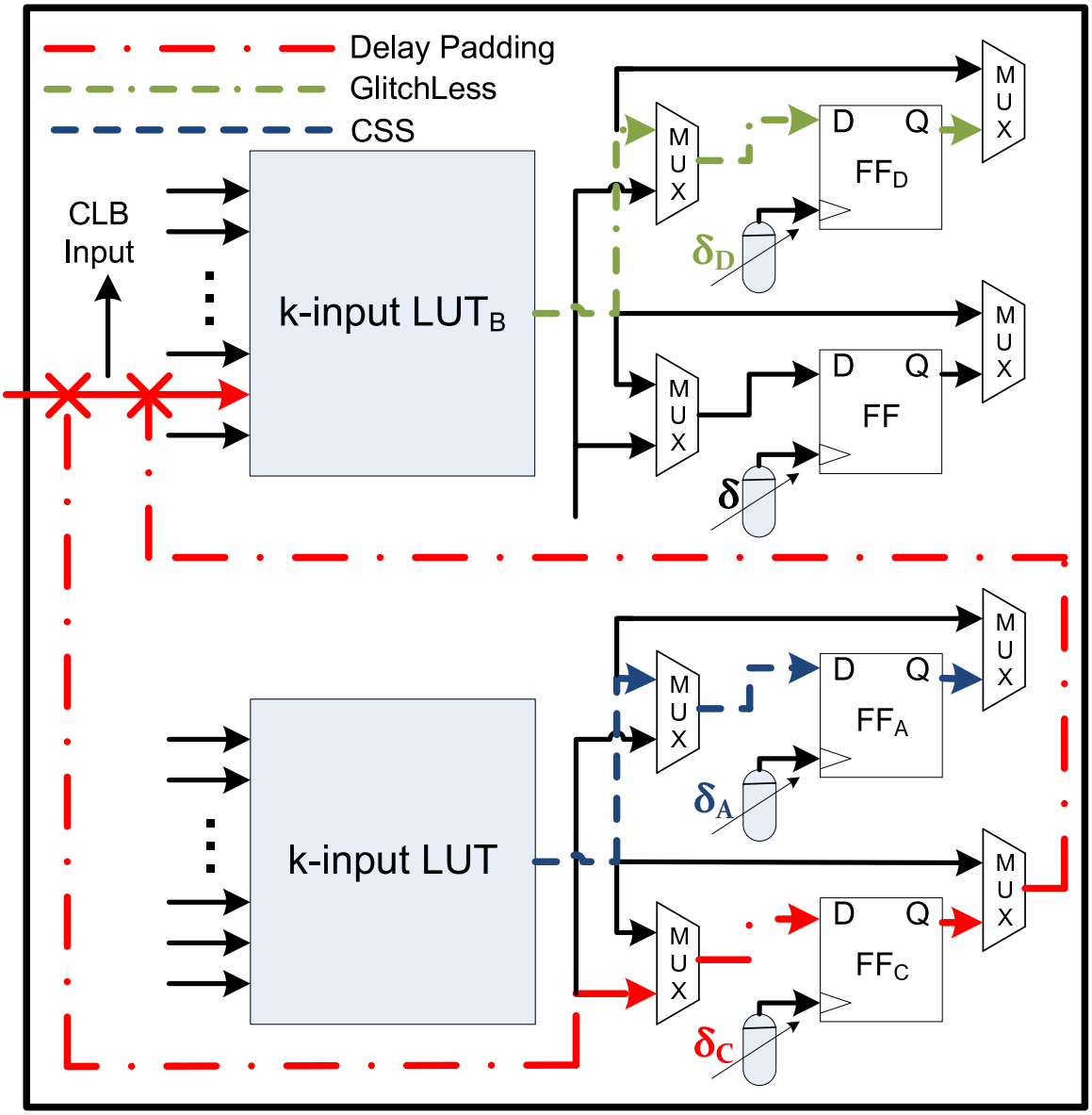

Figure 4.1: Unified Architecture Modification

4.1.2 Architecture for Glitch Reduction

To eliminate glitching on a combinational node, we use a circuit level architecture change different from that analyzed in [19]. Instead of inserting a PDE at LUT inputs, we achieve glitch reduction by directing the LUT output to FF_D, whose clock skew δ_D will be set to the latest arrival time of all LUT inputs plus a setup time and timing margin. The LUT output fluctuates, but the FF will block all glitches until the final functional evaluation is known. Our approach requires only one PDE to eliminate the glitching for each LUT, compared to k-1 PDEs for each LUT used in [19]. One disadvantage of this approach is the fact that clock has an activity of 1. Compared to PDEs inserted into the data lines with relatively low activity (say, between 0.05 and 0.2), this approach may introduce a significant power overhead. We will show how this affects the results in Chapter 6.

4.1.3 Alternative PDE-Sharing Architecture

In an effort to reduce area and power overhead, an alternative architecture is proposed where each FF in the CLB can select one of several PDEs shared by the PDE as shown in Figure 4.2. The number of PDEs available for sharing is an user-determined architecture parameter.

4.1.4 Interaction between CSS and GlitchLess

CSS and GlitchLess share the same architecture change, this is beneficial because applying CSS usually also increases glitching. In this section, we discuss how CSS affects glitching, and how can GlitchLess make use of the CSS architecture to reduce glitching.

Glitching can account for a large portion of dynamic power. Although it varies with every circuit, some correlation can be drawn between the amount of glitching and several circuit characteristics.

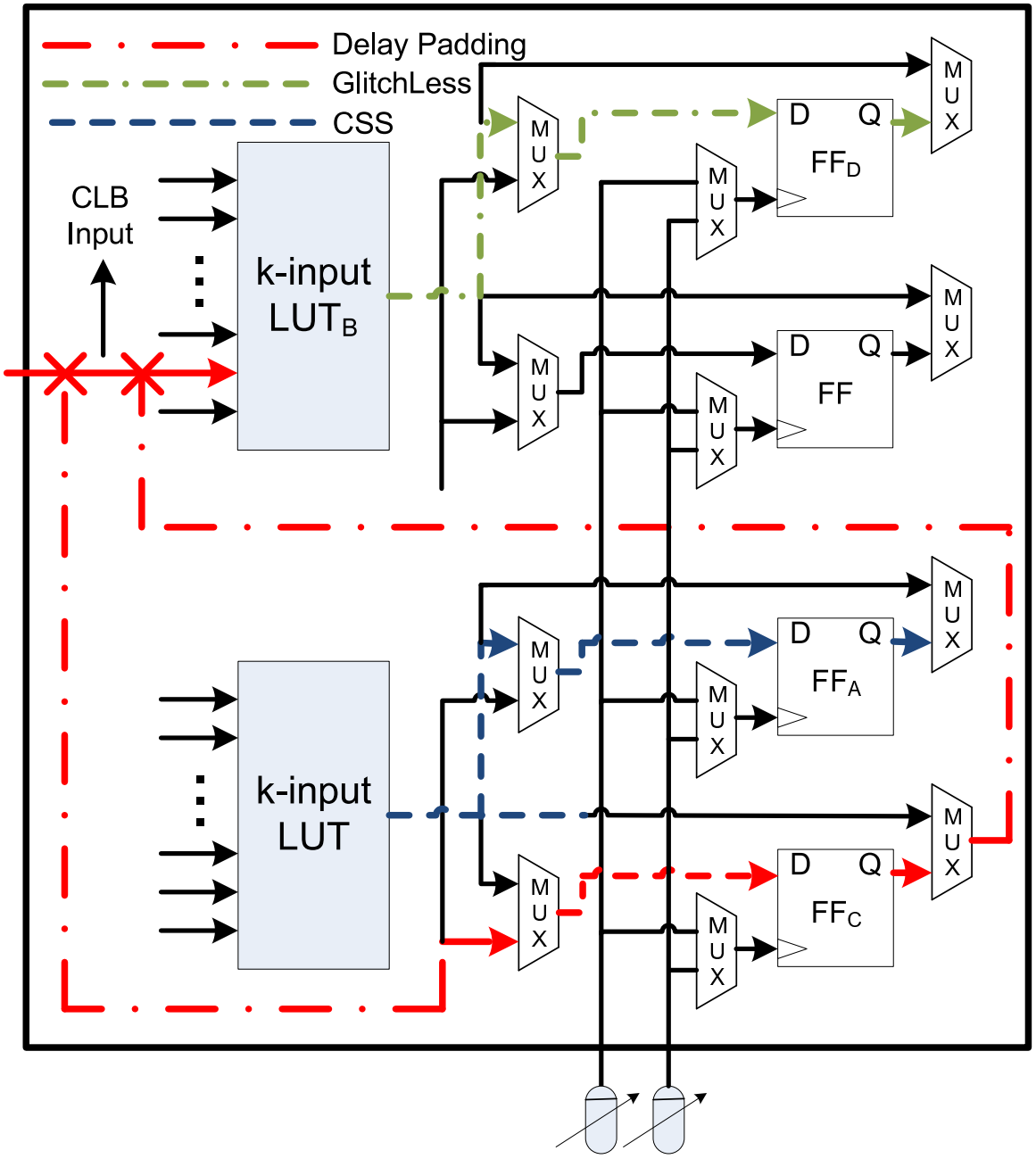

Figure 4.2: PDE Sharing Architecture

circuit	k = 4				k = 6			
	depth	%FF	$\frac{glitch_{preCSS}}{dynamicPower}$	$\frac{glitch_{postCSS}}{dynamicPower}$	depth	%FF	$\frac{glitch_{preCSS}}{dynamicPower}$	$\frac{glitch_{postCSS}}{dynamicPower}$
bigkey	5	13.1	1.9	4.2	5	32.4	1.6	2.3
clma	18	0.4	6.2	11.5	12	0.5	6.2	7.3
diffeq	16	25.2	3.4	2.7	10	43.4	2.1	1.6
dsip	5	16.4	1.3	1.7	5	32.6	1.4	1.8
elliptic	20	31.1	21.0	24.0	12	52.6	15.9	21.9
frisc	25	24.9	13.0	16.8	16	30.1	13.4	15.9
s298	17	0.4	21.7	29.5	13	0.6	23.2	28.2
s38417	13	22.8	26.8	28.3	9	43.2	14.4	24.4
s38584.1	11	19.7	7.8	9.4	9	29.4	4.0	5.57
tseng	15	36.8	15.3	15.1	10	48.2	8.4	11.9
average			11.8	14.3			9.1	12.1

Table 4.1: Sequential Circuit Characteristics and Glitching Power

In Table 4.1, we compare circuit depth and sequential element density of the ten biggest sequential MCNC circuits mapped to 4-LUTs and 6-LUTs to the percent of dynamic power that is due to glitching. The data shows when a circuit has high depth, and low percentage of nodes being sequential, the glitching is high. This is intuitive because FFs block glitching, and creates "flatter" circuits with less depth and less possibility for glitches to travel far. While glitching is insignificant for some circuits, there is motivation for glitch reduction for other circuits where the glitch power accounts for up to 30% of the dynamic power. Also, a 6-LUT architecture tends to have less glitching because fewer LUTs are needed to implement logic. This decreases inter-CLB routing, which has most of the capacitance.

CSS perturbs glitching. All FFs have the same signal departure time in zero-skew circuits, but skew assigned to SEs effectively delay that time, changing the amount of glitching created downstream. In Table 4.1, the preCSS and postCSS columns show the amount of dynamic power due to glitching before and after CSS with delay padding has

been performed, respectively. In most circuits, the amount of glitching increases by a fair margin after CSS. This further motivates the need for glitch reduction.

4.2 Algorithm Overview

The overall approach is illustrated in Figure 4.3. It offers three optimization choices. The first choice (choice "1") uses the original VPR placement and routing solution to generate a net delay file for ACE, which produces an activity file for PGR to do glitch reduction only. The resulting net delays are analyzed by ACE again to produce final activities, and the power analysis routine of PGR is used to determine power savings.

Alternatively, the place and route solution can be used directly by PGR to do CSS and delay padding, followed by ACE simulation (choice "2"), and power estimation of the clock scheduled circuit. The user may choose to use the activity file from the CSS solution to do further glitch reduction (choice "3"), followed by ACE to get power results.

Note that ACE needs to be run twice in choice 1 and 3, because node activities are dependent on the net delays, which changes after either CSS or GlitchLess. In CSS, different arrival times caused by skews to the FFs changes the downstream arrival time for combinational nodes, while GlitchLess delays the arrival time of combinational nodes in consideration of safety margins.

PGR is integrated with VPR into a single executable, and is invoked after VPR placement and routing. The required input files are summarized below.

1. Netlist, architecture, placement and routing files required for VPR.

2. BLIF file required for PGR's data structure building.

3. Activity file from ACE, for power estimation and GlitchLess.

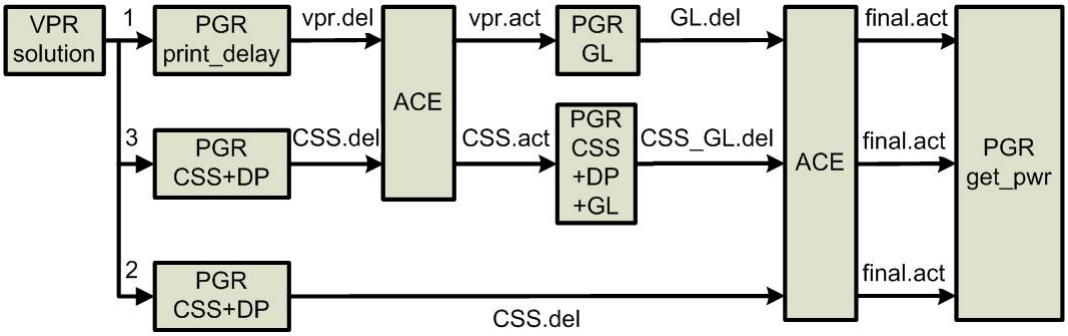

Figure 4.3: Top Level Algorithm

4. Power lookup file from Cadence simulations described in the previous chapter, for power calculations.

The output of the program is a delay file containing net delays between all nodes in the circuit. Depending on user choice, the delay file can be produced immediately after VPR finished, or after GlitchLess, CSS, or both. In addition, if CSS is performed, the program will output the final skew schedule produced as well as the corresponding period.

4.3 PGR Overview

The top-level PGR algorithm, shown in Figure 4.4, builds the necessary data structures for CSS and GlitchLess, and controls the main operation. This section explains some preliminary actions that build the circuit data structure..

4.3.1 Node Building Stage

The first step in the node building stage groups all nodes in the circuit into an array, such that every node is placed after all of its transitive fanins, which include all nodes in its fanin cone. The net delay information is obtained from VPR, and the BLIF input file is used to identify FFs. To avoid cycles, each FF is broken up into two nodes: A virtual

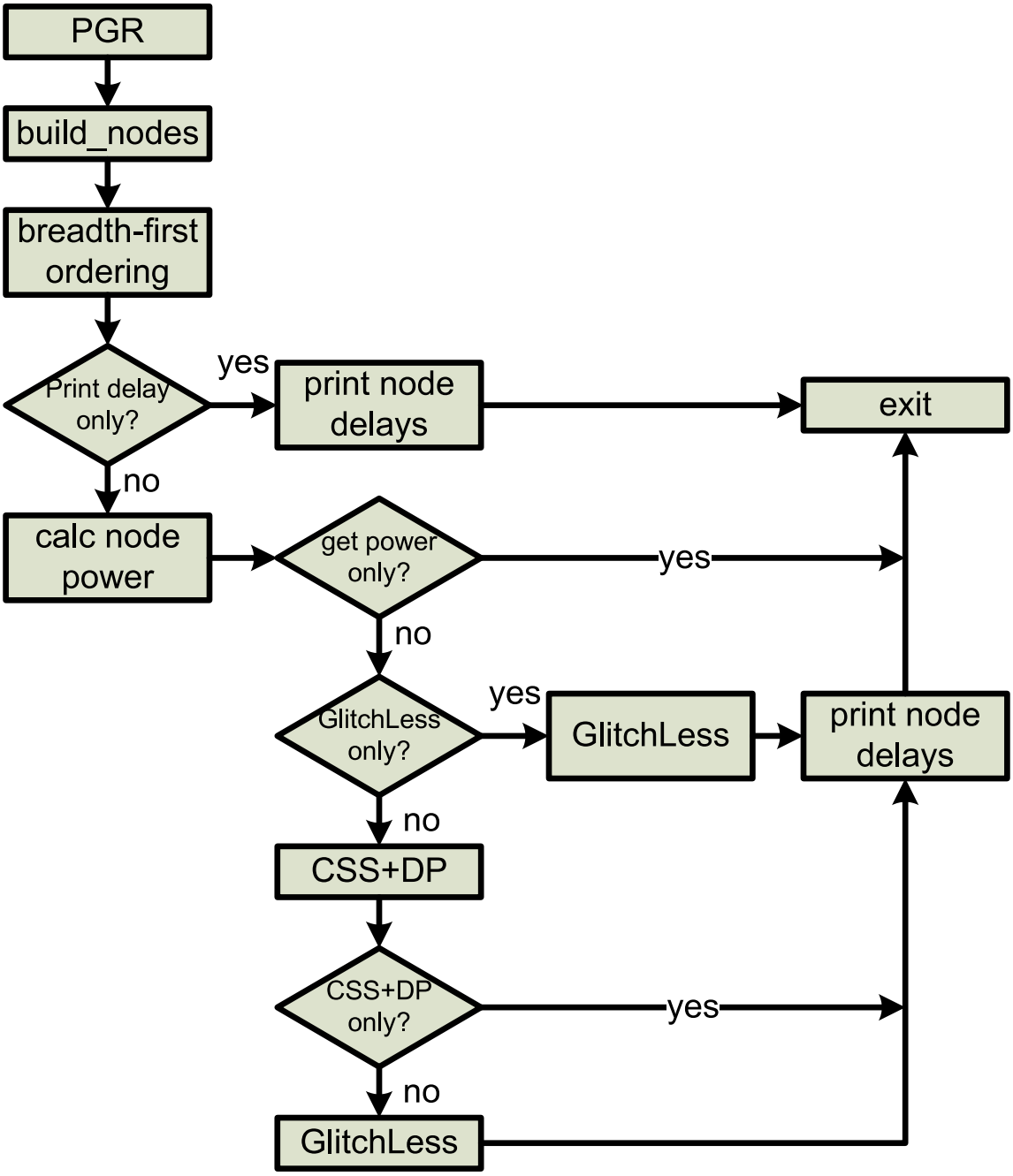

Figure 4.4: PGR Top Level Algorithm

output (has no fanout nodes) representing the D input, and a virtual input (has no fanin nodes) representing the Q output. This ordering of nodes is convenient for timing analysis, which is an important aspect of the program. For GlitchLess, a breadth-first ordering of the nodes is created for topological access. For CSS, a separate data structure is created that represents every FF as one node each to detect negative cycles when the Bellman-Ford algorithm is performed.

Special attention is paid to BLEs in sequential mode. One example is shown in Figure 4.5. In the BLIF file, three signals (named int2, int3 and int4) converge at a OR gate, whose output (named int5) goes directly to a FF. After placement and routing, the OR gate and the FF are absorbed into a single BLE operating in sequential mode, and the signal int5 disappears in the VPR netlist. The original circuit information needs to be preserved for correct interfacing with ACE, which does not have access to the VPR netlist. To accomplish this, each FF net is read from the BLIF file and compared to the VPR netlist to detect this situation. Three nodes are created in PGR, one for the OR gate, and two for the virtual source/sink pair.

In addition to routing, intra-CLB connections also add to net delays. However, the timing graph produced by routing only includes delays of inter-CLB routing. Therefore, these timing parameters are copied out of the architecture file and added in the PGR code.

4.3.2 Node Attribute Calculation

The next stage of preliminary operations is to assign each node its output capacitance, activity and power. Capacitance information is mainly obtained from the VPR routing graph for each net that fans out to inter-CLB routing. For muti-fanout nets, the algorithm calculates the length, in terms of length-4 segments, for each source to sink connection in the net. Intra-CLB capacitance is not included in VPR, and has to be calculated and

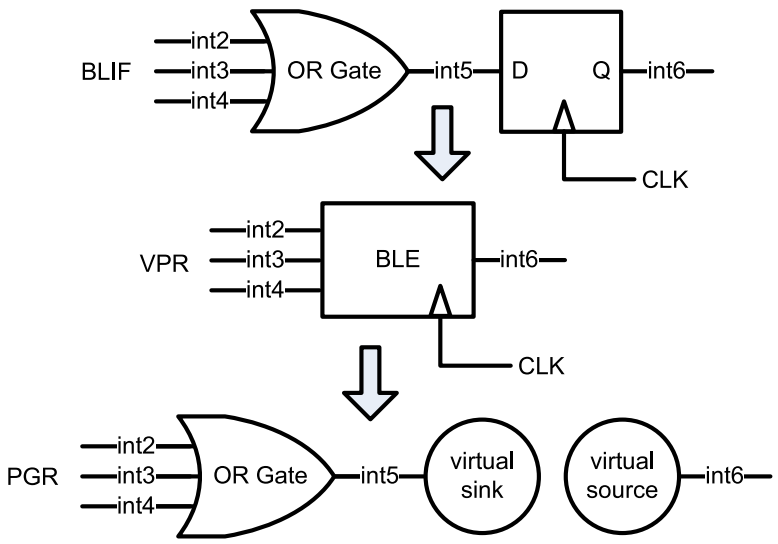

Figure 4.5: BLE in Sequential Mode

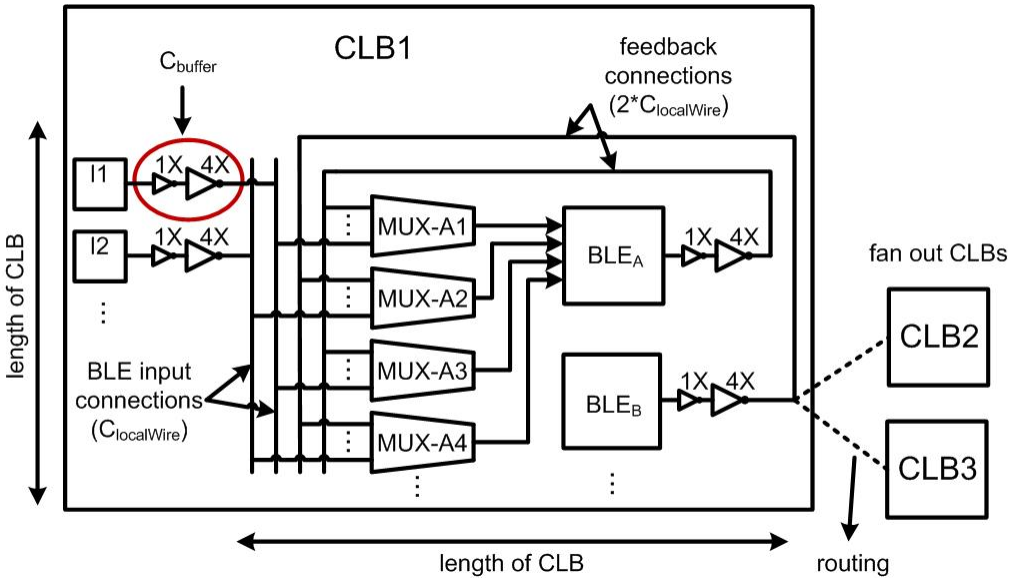

Figure 4.6: Calculating Intra-CLB Capacitance

added to the inter-CLB capacitance. Previous work [26] has a detailed model to compute intra-CLB capacitances. This thesis uses a simplified version of those calculations, and we believe it captures the most important parts of intra-CLB capacitance. Figure 4.6 illustrates how this is done. Each CLB input (I1, I2) connects with a 1X buffer followed by 4X buffer, with total capacitance C_{buffer}. The input then connects to a wire that runs the entire length of the CLB. This wire is named $C_{localWire}$, and is local type routing that has a capacitance less than the general routing resource in between CLBs. This capacitance is separately calculated using PTM models [3]. Each BLE output is connected to a wire with approximately twice the capacitance of $C_{localWire}$ because it needs to be routed back to the input MUXes in an L shape. Each CLB input and BLE feedback connects to half of the input MUXes, according to a 50% connectivity stated in [23]. Therefore the capacitance, named $C_{intraCLBnear}$, is calculated as follows:

$$C_{intraCLBnear} = C_{\text{buffer}} + 2 \cdot C_{localWire} + numMUX \cdot C_{1XNMOSdrain}$$

$$numMUX = inputsPerLUT \cdot numLUTsPerCLB \cdot CONNECTIVITY = 0.5 \cdot k \cdot N$$

In addition to the above equation, a BLE may fan out to inputs of other CLBs in the circuit via inter-CLB routing. A glitch generated by the BLE will propagate to these fan out CLBs, the buffers immediately after the CLB inputs, and the BLE input connections in the fan out CLBs. This capacitance is named $C_{intraCLBfar}$ and is calculated as follows:

$$C_{intraCLBfar} = C_{\text{buffer}} + C_{localWire} + numMUX \cdot C_{1XNMOSdrain}$$

After activities are read in from the activity input file from ACE, and power percentages are read from the power lookup file, the power consumed by each net is calculated with the following equation:

$$Power = C_{intraCLBnear} \cdot \sum_{i \epsilon bins} Glitch_i + \sum_{i \epsilon bins} (\sum_{j \epsilon fanouts} (C_{intraCLBfar} + C_{routing}) \cdot Glitch_i \cdot powerLookup[length_j][i])$$

where $C_{routing}$ is the capacitance of the routing trace between the BLE and the fanout CLB, $Glitch_i$ is the number of glitches that belong to i, the i^{th} bin, j is each fanout that connects to another CLB, and $length_j$ is the length of the routing track, in terms of length-4 segments, that connects to that CLB.

Chapter 5

Detailed Algorithm Description

This chapter discusses the two main routines of the PGR algorithm: CSS and GlitchLess. The interaction between these two modules and the overall algorithm is explained in detail.

5.1 CSS and DP Algorithm

The CSS algorithm uses a binary search technique combined with the Bellman-Ford algorithm to iteratively find the minimum period, then uses delay padding to further improve results. The procedure is repeated twice if there is a restriction on the number of PDEs available to each CLB, in an effort to reduce the period penalty while using as few PDEs as possible.

5.1.1 Motivation

CSS and DP have been explored in previous research, and a checklist of features they offer is compared against that of this thesis in Table 5.1. In the CSS-only area, both ASICs and FPGAs have been considered with variation modelling and solved using efficient graph algorithms. However, the work for FPGAs [28, 31, 36] does not consider delay padding. The work in [18, 33] considers both CSS and DP but is targeted to ASICs only. Variation modelling is included in [33], but it uses the less efficient LP method. In contrast, [18] uses more efficient graph algorithms, but lacks variation modelling. In addition, both of these

Feature		Sapatnekar [11]	Brown [28] / Sadowska [36] / Bazargan [31]	Kourtev [33]	Lu [18]	This Thesis
CSS	Platform	ASIC	FPGA	ASIC	ASIC	FPGA
	Delays	continuous	discrete	continuous	continuous	discrete
	Variation	yes	yes	yes	no	yes
DP	Platform			ASIC	ASIC	FPGA
	Delays			continuous	continuous	discrete
	Variation			yes	no	yes
Algorithm		graph	graph	LP	graph	graph

Table 5.1: Feature Comparison

approaches use a continuous delay model, which is not realizable with PDEs.

Our post-CSS delay padding algorithm offers the following novelties. It targets FPGAs, and it is aware of discrete delay steps, process variation margins that limit both the minimum and maximum delay that can be assigned to a node, and the possibility that delay padding may fail due to these margins. To the best of our knowledge, this is the first time delay padding has been applied to FPGAs.

5.1.2 Algorithm Description

The top-level CSS algorithm is shown in Figure 5.1 and described in more detail in Figure 5.2. The "CSS" box in Figure 5.1 refers to lines 6 to 14 in Figure 5.2, and the "DP" box refers to lines 15 to 21.

The first step uses the result of the node building stage (Section 4.3.1) to produce a circuit that contains CSS nodes to represent each FF, and edges representing difference constraints (Chapter 2.2.1). This is done in the build_CSS_nodes() function. Maximum and minimum path delays between each pair of CSS nodes is calculated using the method in [14]: the current source's arrival time is set to zero while all others' are set to infinity. After latest arrival time propagation, all sinks reachable from the source will have non-infinity

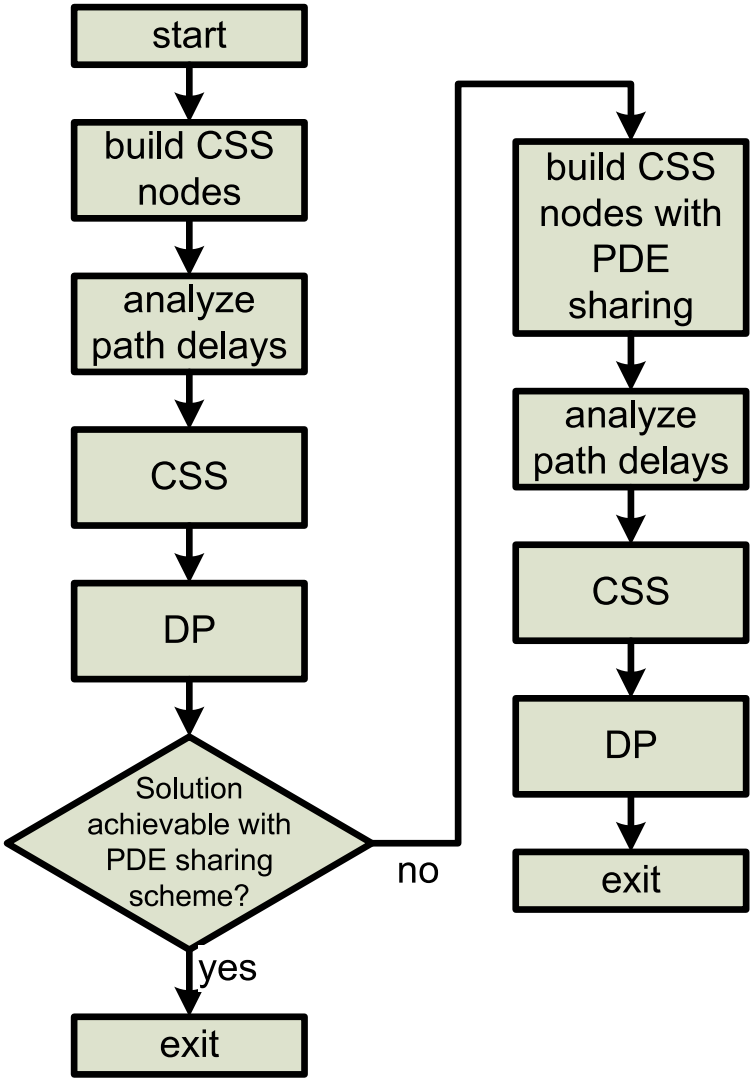

Figure 5.1: CSS and Delay Padding Flow Chart

```
1: //start two-pass procedure
2: for pass=0 to 1 do
3:    iteration = 0;
4:    build_CSS_nodes();
5:    analyze_path_delays();
6:    solution[iteration] = assign_skews(P_max, P_min);
7:    num_edges = find_crit_hold_edges(edges[iteration]);
8:    while num_edges > 0 do
9:       find_deleted_edge_nodes();
10:      recalculate P_max and P_min using recalc_binary_bounds();
11:      iteration++;
12:      solution[iteration] = assign_skews(P_max, P_min);
13:      num_edges = find_crit_hold_edges(edges[iteration]);
14:   end while
15:   while iteration > 0 do
16:      success = pad_delay(edges[iteration],solution[iteration]);
17:      if success then
18:         break;
19:      end if
20:      iteration−−;
21:   end while
22:   if pass==0 then
23:      if solution is achievable with number of PDEs available for sharing for all CLBs
         then
24:         break;
25:      end if
26:   end if
27: end for
```

Figure 5.2: CSS and Delay Padding Algorithm

arrival times representing D_{max}. A similar process is repeated to find D_{min} using negative infinity and earliest arrival time propagation. This is done in the analyze_path_delays() function.

The binary search method with Bellman-Ford algorithm in Figure 2.3 [11] is then applied inside the assign_skews() function. During each successful iteration in the binary search, where there are no negative cycles in the solution, each negative skew assigned by Bellman-Ford is adjusted to a positive value by adding the magnitude of the most-negative skew to the entire set of skews. Then, all skews are quantized, a procedure that snaps each skew to its nearest time unit that can be realized using a programmable delay element. This will cause a slight perturbation to the optimized period, and is calculated accordingly. The final period and skew schedule is stored in a solution array (line 6 in Figure 5.2).

After the initial CSS run, all critical hold-time edges found with the current skew schedule are removed and appended into a list of currently deleted edges [18] using find_crit_hold_edges(). The combinational LUTs on each critical edge are identified and put into arrays with find_deleted_edge_nodes(). The assign_skews() routine is run again with recalculated binary search bounds ignoring the deleted edges. The Bellman-Ford algorithm will ignore the hold-time constraint of the deleted edges, so their violation will not cause the negative weight cycle flag to go positive. This iterative process is repeated until no more critical edges can be found, and the end result contains the skews, period and deleted edges found with each iteration, all stored in arrays.

In the next step, the algorithm will attempt to pad delays for all deleted critical edges from the most current iteration. In case delay padding is not successful, additional attempts will be made for earlier iterations until a valid padding solution is achieved. The detailed delay padding algorithm is shown in Figure 5.3. For each deleted edge, the algorithm traces the combinational path from the source to sink in a topological breadth-first order. When

each node is visited, it tries to delay the signal as late as possible without violating setup constraints. In other words, the padded delay must be smaller than the available slack, which is calculated on line 6 of Figure 5.3, and is equal to the required time of the node "n" we are trying to delay (n→required) minus the latest signal arrival time of the fanin node (fanin→arrival) and the path delay between the fanin node and n (fanin_delay(n, fanin)).

To add delay to node n, its skew is first set to the arrival time of the fanin node plus the setup time and variation margin, rounded up to the nearest discrete time unit specified by the parameter "PRECISION". The variation margin added here is to compensate for an early arriving clock, which could cause the data signal to be late. In the case the clock arrives late, the data path has to have enough slack to compensate for it (line 9). This check is done with line 10: a node is skipped in the delay padding process if there is not enough slack to compensate for both late and early arriving clocks. To complete delay padding for the current edge, the quantity "delay" is incremented in quantized steps (lines 13 to 15), until either the node's available slack runs out, or the needed delay is satisfied. If padding one node is not enough to satisfy the hold-time constraint for the entire path, other nodes on the path are then considered for delay padding as well.

When delay padding for an edge finishes, check_other_paths() is used to check for other short delay paths (with the same source and sink) violated. Any such violations found are fixed in the same way, until setup and hold-time constraints for all combinational paths with the same source and sink are satisfied.

In this thesis, "PRECISION" and "MARGIN" in Figure 5.3 are chosen to be 0.1ns, or 1 discrete step, so that each PDE has at 0.1ns of uncorrelated variation tolerance between its assigned skew and path delay, for early and late clock signals. The margin M for CSS is 0.2ns, allowing T_i and T_j (Eqs. 2.2,2.3) to each shift 0.1ns away from each other, for example. In reality, timing margins should be a percentage of the magnitude of the delay,

```
1: success = 1;
2: for all edge "iedge" in deleted edges list do
3:    needed_delay = calculate_needed_delay(iedge);
4:    for all node "n" on deleted edge "iedge" do
5:       analyze_timing();
6:       available_slack = n→required - fanin→arrival - fanin_delay(n, fanin);
7:       skew = roundup(fanin→arrival + Ts + MARGIN + fanin_delay(n, fanin), PRE-
         CISION); //for early arriving clock
8:       delay = skew − fanin→arrival − fanin_delay(n, fanin);
9:       needed_slack = delay + MARGIN; //for late arriving clock
10:      if needed_slack>available_slack then
11:         continue;
12:      end if
13:      while (delay<needed_delay && needed_slack<max_padding) do
14:         increment skew, delay and needed_slack by PRECISION
15:      end while
16:      needed_delay −= delay;
17:      if needed_delay ≤ 0 then
18:         edge_done = 1; break;
19:      end if
20:   end for
21:   if edge_done == 1 then
22:      check_other_paths();
23:   else
24:      success = 0;
25:   end if
26: end for
27: if !success then
28:    roll_back_delays();
29: end if
30: return success;
```

Figure 5.3: Detailed Delay Padding Algorithm, pad_delay()

so that longer paths have more room for error, while smaller paths can have smaller margins and take up less slack. The margins used here may not be sufficient for larger delays, and further analysis with variable timing margins will be included in future work.

5.1.3 Variable PDE-per-CLB Restriction

When there is a restriction on the number of PDEs each CLB can have, a two-pass CSS approach is used to reduce the performance degradation due to decreased PDE availability, and the power consumption overhead. The number of PDEs available to each CLB is an user specified command line parameter.

In the first CSS pass, the algorithm in Figure 5.2 is performed assuming there is no PDE restriction. At the end of the assign_skews() and try_pad_delay() routines, the algorithm will determine the number of distinct skews needed in each CLB for CSS and DP. For example, if a CLB has six FFs with the skew set {1.2, 1.2, 1.6, 1.6, 1.6, 1.8}, and three combinational LUTs delay padded with the skews {1.2, 2.3, 3.0}, then this CLB needs five distinct skews {1.2, 1.6, 1.8, 2.3, 3.0}, three of which are needed for CSS ({1.2, 1.6, 1.8}) and three for DP ({1.2, 2.3, 3.0}).

At the end of the first pass, the necessity of the second pass is determined by comparing the required number of PDEs in each CLB to the number of PDEs available. If there exist any CLBs with more skews needed than the number of PDEs available, a second pass is needed. In the above example, if there are less than five PDEs to share, then the second pass is necessary.

In the second pass, the build_CSS_nodes() function at line 4 in Figure 5.2 differs from the first pass, where each FF is treated as a distinct node in the CSS graph. The situation for each CLB in the second pass can be one of two cases, illustrated using the above example.

In the first case, there are enough PDEs to satisfy the number of skews needed by CSS alone. This number is 3 in the example, for skew values of 1.2, 1.6 and 1.8. There are 2 FFs with skew 1.2. They will be "fused" together to form a single node in the CSS graph for the second pass. The same applies to the 3 FFs with skew 1.8. In the end, the 3 PDEs become 3 CSS nodes, and each may contain more than one FF.

In the second case, there are not enough PDEs to satisfy all needed skews. In the example, if there are fewer than 3 PDEs to share, we need a sharing schedule that distributes the six FFs to the 3 PDEs, such that the impact on the period is minimum. An easy way to do this is a random assignment, which has been implemented for this thesis. This may not be the optimum sharing schedule having the minimum impact on period. However, if there are only a few CLBs that need the treatment in the second case, the circuit graph used for CSS in the second pass will not be too different from that in the first pass.

After CSS is done with the fused nodes, delay padding is done as usual. After the skew to be assigned to a node "n" is decided (line 15 in Figure 5.3), a check is performed to see if there is a pre-existing PDE that offers the same skew. If there are no existing PDEs that can provide the required skew, a free PDE will be used. If there are no more free PDEs available, node "n" will be skipped and the algorithm will try to pad delays for the next node.

The last step of the algorithm assigns the final skew schedule to each FF in the circuit, and updates PDE usage information for each CLB.

5.1.4 Interface with GlitchLess

After CSS and DP are done, skews from the graph structure used to do CSS must be added to the array of nodes built earlier (Section 4.3.1). Since each FF is split into a virtual source and a virtual sink, the skew is added to the virtual source to signify the change of signal

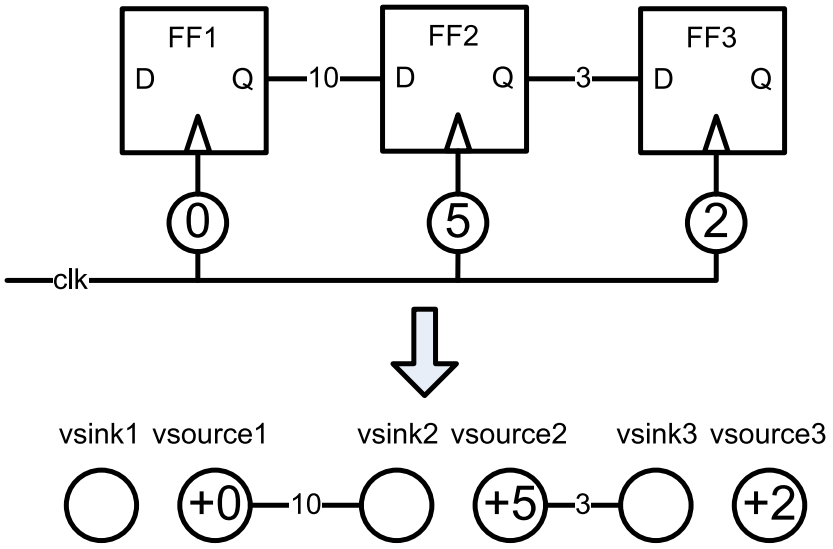

Figure 5.4: Adding Skew to Circuit Timing

departure times, as shown in Figure 5.4.

After the skews have been added, static timing analysis changes. In the example in Figure 5.4, the skew of 5 time units is added to the local path between FF2 and FF3. this will lengthen the arrival time of vsink3 to 8 instead of the 3 units of combinational delay. In a zero-skew circuit, the required time at vsink3 is the period. Since the skew of 2 at FF3 delays the departure time, the signal has more time to arrive, so the required time is period plus 2. If the period is 7, then the slack at vsink3 is required time - arrival time = 7+2-8 = 1.

5.2 Glitch Reduction Algorithm

Regardless of whether CSS and DP has been done before, the GlitchLess algorithm uses static timing analysis to determine the feasibility of adding PDEs in combinational paths.

```
1: for all levels in timing graph do
2:   rank_nodes(&list, threshold);
3:   for all node "n" in list do
4:     skew = roundup(n→arrival + Ts + MARGIN, PRECISION); //for early arriving
       clock
5:     needed_slack = skew − n→arrival + MARGIN; //for late arriving clock
6:     if needed_slack < n→slack && PDE is available then
7:       for all fanin "f" of node "n" do
8:         needed_delay = n→arrival − f→arrival − fanin_delay(n, f);
9:         fanin_delay(n, f) += needed_delay + needed_slack;
10:      end for
11:      analyze_timing();
12:      update_pde_usage();
13:    end if
14:  end for
15: end for
```

Figure 5.5: Glitch Reduction Algorithm

5.2.1 Motivation

One significant difference of this work compared to [19] is added consideration of process variation. In [19], all added delays are shortened by an amount d so that variation will not increase the critical path. This may result in increased glitching power in practise, since narrow pulses are not "zero power" as assumed previously. The approach in this thesis eliminates this issue by stopping all glitching at the FF input, and only clocks through the data when the last signal has arrived.

5.2.2 Algorithm Description

The algorithm is shown in Figure 5.5. The circuit is traversed in a breadth-first fashion so that the extra delays assigned upstream can be included and assumed to be fixed when calculating delay requirements for downstream nodes. A user-defined threshold parameter is specified to filter out nodes with small glitch activity. In each level (line 1), all com-

binational nodes are ranked according to their glitching power, so that nodes with high loading get priority during PDE delay assignment. For each node "n" that is eligible for GlitchLess, the skew is set to the node's latest signal arrival time (n→arrival), plus the setup time and timing margins (line 4 in Figure 5.5). Similar to delay padding, a check is done to make sure node "n" has enough slack in case the clock signal arrives late (lines 5 and 6).

When there is a restriction on the number of PDEs per CLB, the PDE checker (line 6, Figure 5.5) first checks to see if the needed skew can be provided by any of the CSS or DP skews, if there is no existing PDE to satisfy the skew, it then checks if a free PDE is available.

After GlitchLess is applied to each combinational node "n", static timing analysis is done on the circuit to recalculate arrival/required times, and the available number of PDEs for the CLB where "n" resides in is decremented if a free PDE is used up. (line 11 and 12, Figure 5.5).

Chapter 6

Experimental Results and Discussion

Of the largest 20 MCNC circuits, the 10 sequential circuits are used as benchmarks, and they are simulated for k = 4 and 6, N = 10, and I = 22 and 33, respectively. A technology of 65nm is used. All results are normalized with respect to the original solution found using VPR 5.0. Architecture files are from the iFAR repository [2], and routing resource capacitance and resistance values are calculated from the PTM website [3] assuming CLBs are 125μm x 125μm in size. All circuits are simulated using timing-driven placement and routing with a channel width of 104, which is sufficient to route all of these circuits. Area estimation is done with a channel width of 200, which represent the capacity of today's low-cost FPGAs. Activity estimation is produced with the modified ACE described in Chapter 3, simulated with 5000 pseudo-random input vectors.

6.1 CSS-Only Results

6.1.1 Performance

In Table 6.1 and 6.2, we demonstrate the period reduction (as a percentage of original critical path) we are able to obtain from CSS and delay padding, for k=4 and 6, respectively. The design flow follows choice "2" in Figure 4.3. The "20 PDE" column represents the

architecture where we allow each FF to have its own PDE, and the other columns represent the case where we fix the number of PDEs that are shared by FFs in each CLB. The CSS column represent CSS-only results, and the CSS+DP column represent CSS+DP results. When the result for CSS and CSS+DP are the same, the 2 columns are merged into one cell. The benchmarking for k=4 stops at 6 PDEs per CLB, and at 7 PDEs per CLB for k=6 because further increasing the number of PDEs per CLB does not incur any additional performance benefit.

When each FF has its own PDE, average speedups of 13% (k=4) and 17% (k=6) are obtained for CSS alone. Delay padding further improves 4 circuits for both k values. For *elliptic*, *frisc* and *tseng*, delay padding is very helpful. CSS alone reduces period by up to 37.7% for *dsip*, and CSS+DP reduces period by up to 32.1% for *tseng*.

When FFs share PDEs, the general trend is the period gets shorter as more PDEs become available. For k=4, the circuits that never benefit from DP do not differ in performance regardless of the number of PDEs available. This is likely because these circuits generally have lower percent of FFs (Table 4.1). This is mostly true for the k=6 case as well.

For the circuits that benefit from DP, the k=4 case sees the *dsip* circuit not having any difference across all configurations, possibly owing to the fact that it has the least FF density of the four circuits. For the k=6 case, *elliptic* is the circuit with the highest FF density, and the other three circuits do not show much difference in performance as the number of PDEs is increased. In general, the opportunity for delay padding becomes apparent only when more PDEs are available for sharing. More PDEs gives skew assignment more flexibility, and makes it more probable for critical edges to occur. For circuits that do not benefit from DP, one PDE per CLB is generally enough for the best solution possible.

There are some instances when more PDEs are available for sharing, but the perfor-

circuit	1 PDE		2 PDE		3 PDE		4 PDE		5 PDE		6 PDE		20 PDE	
	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP
bigkey	90.7		92.2		92.2		92.2		92.2		92.2		92.2	
clma	91.2		91.2		91.2		91.2		91.2		91.2		91.2	
diffeq	78.2		78.3		78.3		78.3		78.3		78.3		78.3	
dsip	74.0	72.6	73.6	72.5	73.6	72.5	73.6	72.5	73.6	72.5	73.6	72.5	73.6	72.5
elliptic	98.4		98.2	98.0	96.8	95.5	93.7	88.7	91.1	88.7	91.1	68.1	91.1	68.1
frisc	87.4		78.6	79.8	78.3	78.1	78.6	76.9	78.6	76.9	78.6	74.4	78.6	74.4
s298	99.7		99.7		99.7		99.7		99.7		99.7		99.7	
s38417	97.4		98.4		97.1		97.7		97.5		97.5		97.5	
s38584.1	88.5		88.5		87.7		87.7		87.7		87.7		87.7	
tseng	97.8		79.2		79.2	73.9	79.2	77.0	79.2	67.9	79.2	67.9	79.2	67.9
geomean	89.9	89.7	87.3		87.0	86.1	86.8	85.7	86.5	84.6	86.5	82.2	86.5	82.2

Table 6.1: CSS and DP Results, % of Original Period, k=4

circuit	1 PDE		2 PDE		3 PDE		4 PDE		5 PDE		6 PDE		7 PDE		20 PDE	
	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP	CSS	CSS+DP
bigkey	65.8	65.8	65.8		65.6		65.6		65.6		65.6		65.6		65.6	
clma	96.9	96.9	96.9		96.9		96.9		96.9		96.9		96.9		96.9	
diffeq	96.6	96.6	89.2		84.8	83.3	84.8	83.3	84.8	83.3	84.8	83.3	84.8	83.3	84.4	83.3
dsip	72.9	72.9	62.6		62.7		62.3		62.3		62.3		62.3		62.3	
elliptic	98.8	98.8	97.3		98.8		95.8	96.9	88.5	85.4	88.5	85.4	88.5	74.5	88.5	74.5
frisc	92.9	92.9	82.0	81.8	81.3	77.7	81.3	73.6	81.3	68.2	81.3	68.2	81.3	68.2	81.3	68.2
s298	100	100	100		100		100		100		100		100		100	
s38417	99.3	99.3	99.9		99.3		98.3		98.3		97.9		97.9		97.9	
s38584.1	96.7	96.7	88.1		88.1		88.1		88.1		88.1		88.1		88.1	
tseng	88.5	88.5	88.9		76.6	76.2	76.6	74.7	76.6	73.1	76.6	73.1	76.6	73.1	76.6	73.1
geomean	90.0	90.0	86.0		84.3	83.8	83.9	82.9	83.3	81.0	83.2	81.0	83.2	79.9	83.2	79.9

Table 6.2: CSS and DP Performance, % of Original Period, k=6

mance is slightly worse than the case with fewer PDEs. An example is *frisc* at k=4 for CSS only, from 3-PDEs to 4-PDEs. The difference in skew schedule is caused by a difference in how the PDEs are shared. In other words, the optimum period before the skews are quantized are the same, and the slight difference in skew schedule due to different PDE sharing schemes causes the final period to differ. The difference is less than one percent.

For CSS+DP, an example is *tseng* at k=4, between the 3-PDE and 4-PDE columns. In this case, the ability to complete delay padding differs when PDEs are shared. When there are three PDEs for sharing, the skew schedule may help DP to share PDEs better than when there are four PDEs for sharing.

There are also cases where the CSS+DP result is worse than the CSS-only result, for example, *frisc* at k=4 and 2 PDEs per CLB. While the CSS+DP skew schedule is produced by the circuit graph without all the critical hold edges, the CSS only solution is produced by the graph with those edges. Therefore, the PDE sharing arrangements during the second pass are different, and this results in the final optimum period being different. This difference is also less than one percent.

6.1.2 Power Overhead

Table 6.3 and 6.4 show the power overhead, normalized to the power of the original circuit. The first two columns represent power overhead for the architecture where each FF has its own PDE, and expressed as a percentage. The "no PDE" column excludes the power of the PDEs themselves, but includes the power fluctuation due to changes in activity. The increase averages 3-4% for both LUT sizes. When the PDEs' own power is included, as shown in the "PDE" column, the total power overhead of the scheme becomes apparent. The PDE designed in [19] is used with 6 stages of delay. As a result of more FETs needed as well as progressively larger FETs needed to provide large delays, each PDE adds roughly

45 units of power (45fF, activity of 1).

The power overhead is compared to that of the approach used in [28], which uses 4 phase-shifted global clocks. Since [28] does not report power usage, we estimate the power of 3 additional global clocks as follows. For each global clock, the placement size (rows $\times$ columns of CLBs) is obtained from VPR, and it is assumed that CLBs are laid over a spine-and-ribs clock network. For example, if a circuit is 10×10 CLBs, then the power due to one extra clock is $C_{clk} = 10 \cdot C_{rib} + C_{spine} + C_{clb}$, and $C_{rib} = 10 \cdot C_{int}$, $C_{spine} = 10 \cdot C_{global}$, and $C_{clb} = 10 \times 10 \cdot C_{local} \cdot \%_{FF}$. C_{global}, C_{int} and C_{local} are the capacitance of the wire that spans the length of 1 CLB for global, intermediate and local type routing wire, calculated from the interconnect model and typical wire dimensions from [3]. $\%_{FF}$ is the percent of the CLBs that contain an active flip-flop. The power overhead of 3 more global clocks is then $3 \cdot C_{clk}$. This is shown in the 4-clk column. Bigger circuits with fewer user FFs incur less overhead with our approach, while small circuits with more SEs favor the approach in [28].

Power overhead can be decreased by using fewer PDEs as shown in the other columns. In these cases, the power overhead further includes the capacitance of the wires needed to deliver each delayed signal into the CLB. This overhead is around 18fF for a 125μm segment of local routing, which is relatively small compared to the PDE's capacitance.

For both LUT sizes, all circuits and all configurations of PDE sharing, the power overhead is smaller than that with the original 1-PDE-per-FF approach and the 4-clk approach due to multiple FFs sharing the same PDE in many CLBs. For some circuits, the power overhead increases as more PDEs became available and are used for sharing, but the performance does not necessarily increase. For example, *diffeq*'s period is the same (k=4) in all configurations (Table 6.1), but the power overhead is always increasing (Table 6.3). This demonstrates the fact that there are many skew schedules that can satisfy a

circuit	no PDE	PDE	4-clk (estimated)	1 PDE	2 PDE	3 PDE	4 PDE	5 PDE	6 PDE
bigkey	103	162	287	149	163	163	163	163	163
clma	106	119	206	114	119	119	119	119	119
diffeq	99.3	636	567	358	474	529	534	534	534
dsip	100	153	264	144	146	146	146	146	146
elliptic	104	235	193	153	168	175	181	181	210
frisc	105	618	557	319	408	431	452	459	482
s298	111	114	168	114	114	114	114	114	114
s38417	102	168	167	123	133	150	154	156	157
s38584.1	102	156	165	131	141	146	147	147	147
tseng	99.8	391	332	237	274	298	292	329	329
geomean	103	224	261	169	188	196	198	201	205

Table 6.3: Power after CSS and DP, % of Original Dynamic Power, k = 4

circuit	no PDE	PDE	4-clk (estimated)	1 PDE	2 PDE	3 PDE	4 PDE	5 PDE	6 PDE	7 PDE
bigkey	101	182	348	162	172	177	179	179	179	179
clma	101	118	191	118	118	118	118	118	118	118
diffeq	99.6	812	475	367	450	546	578	586	590	590
dsip	100	172	316	154	158	161	161	161	161	161
elliptic	108	275	209	151	164	167	170	195	196	226
frisc	103	736	571	332	411	447	476	539	545	545
s298	107	111	156	111	111	111	111	111	111	111
s38417	113	261	199	151	172	182	200	207	220	224
s38584.1	102	174	178	134	146	151	152	152	153	153
tseng	104	463	384	264	286	338	336	370	370	370
geomean	104	260	276	178	194	206	212	220	222	226

Table 6.4: Power after CSS and DP, % of Original Dynamic Power, k = 6

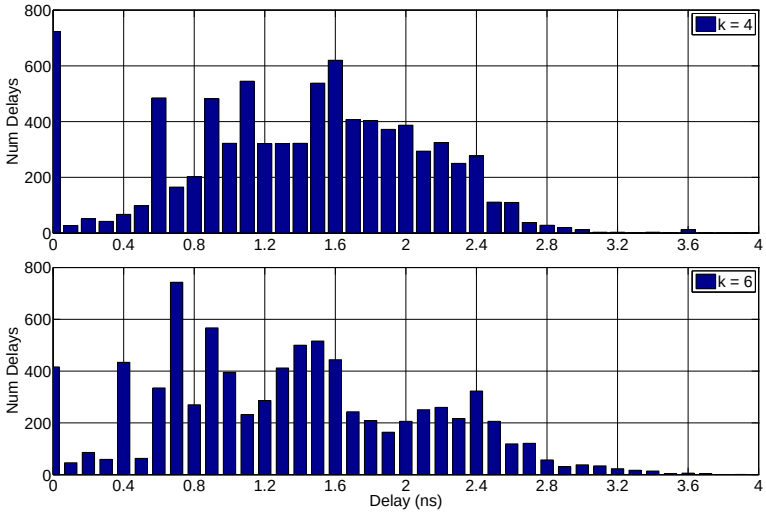

Figure 6.1: Skew Histogram

given period in the decision version of the CSS problem. Those that can achieve the same period with fewer number of distinct skews are better in terms of power. Running each benchmark across a range of configurations, one can determine what is the best sharing scheme for individual circuits.

A cumulative histogram of the total number of PDEs used for each discrete delay across all circuits is shown in Figure 6.1. The data is relatively spread out, indicating that PDEs use a large number of different delays.

6.1.3 Area Overhead

Area calculation is done with the model used in [19] and a channel width of 200. With 1 PDE assigned to each FF, 20 PDEs are needed per CLB, and the overhead is 11.7% for k=4 and 7.6% for k=6. This is the maximum area overhead with our approach. With PDE sharing, the area over decreases drastically. For example, a 4 PDE configuration will cut the overhead by nearly 5x, taking into account the area of 20 multiplexers for every FF.

6.2 GlitchLess Only Results

In Figure 6.2, normalized glitching power is plotted against the ranking threshold percentage, where the node with the maximum glitching power in each circuit is defined as a threshold of 1.0. The design flow follows choice "1" in Figure 4.3. PDE overhead is not added. The lines represent glitch power savings as the threshold is gradually decreased and more nets are de-glitched. The trend is rather gradual for threshold values larger than 20%, and this makes sense since there are only a few nodes with high glitching power consumption. As the threshold is lowered below 20%, more nodes became eligible for skew assignment and the rate of savings increases more dramatically. The square and circle lines show that even at a threshold of 0, total glitch elimination is not possible because nodes on the critical path are not considered for de-glitching. When these nodes are considered, the PDEs result in an increase of the critical path, allowing all glitching to be removed, as the triangle and cross lines point out.

The difference in power savings between the cases when there is only 1 PDE per CLB, and when every FF has its own PDE is almost negligible for threshold values greater than 0.2. This is shown in Figure 6.3 by the additional curves. This is because the few nodes that have glitching power higher than a threshold of 0.2 are very likely to be spread out among many CLBs, instead of concentrating inside a single CLB. However, as the threshold is further decreased, more nodes in every CLB will become eligible for GlitchLess, and the savings increases. Lastly, an additional configuration where there are 2 PDEs given to each CLB for GlitchLess, is shown to have negligible difference from the square and circle lines.

The impact of increasing the critical path is shown in Figure 6.4. Again, PDE overhead is omitted. The Y-axis is a product of the dynamic power and the period, normalized to that of the original circuit placed and routed by VPR. The square lines excludes nodes on the critical path, so period stays the same as they reflect potential power savings. However,

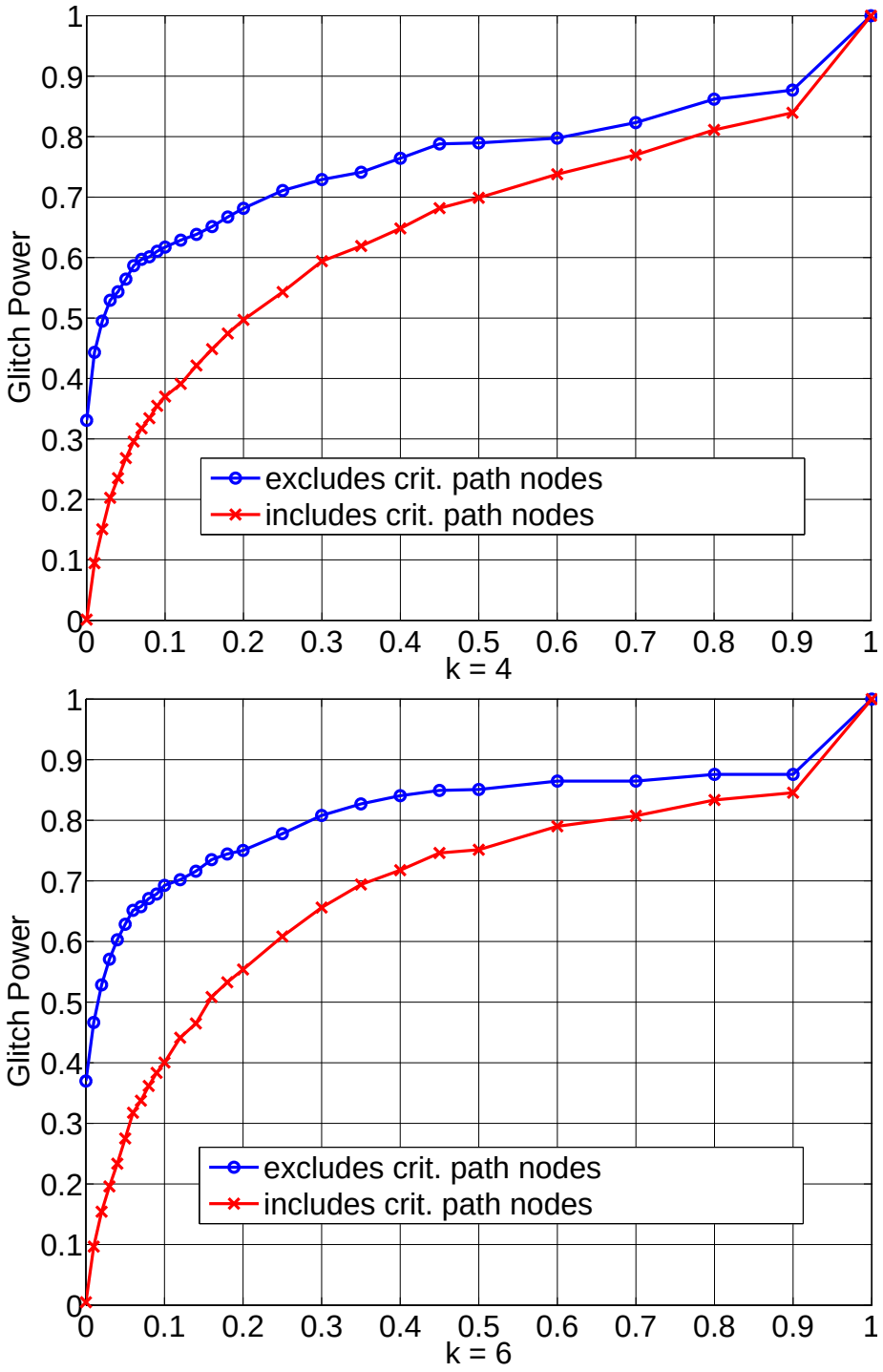

Figure 6.2: Glitch Power Reduction, 1 PDE per FF

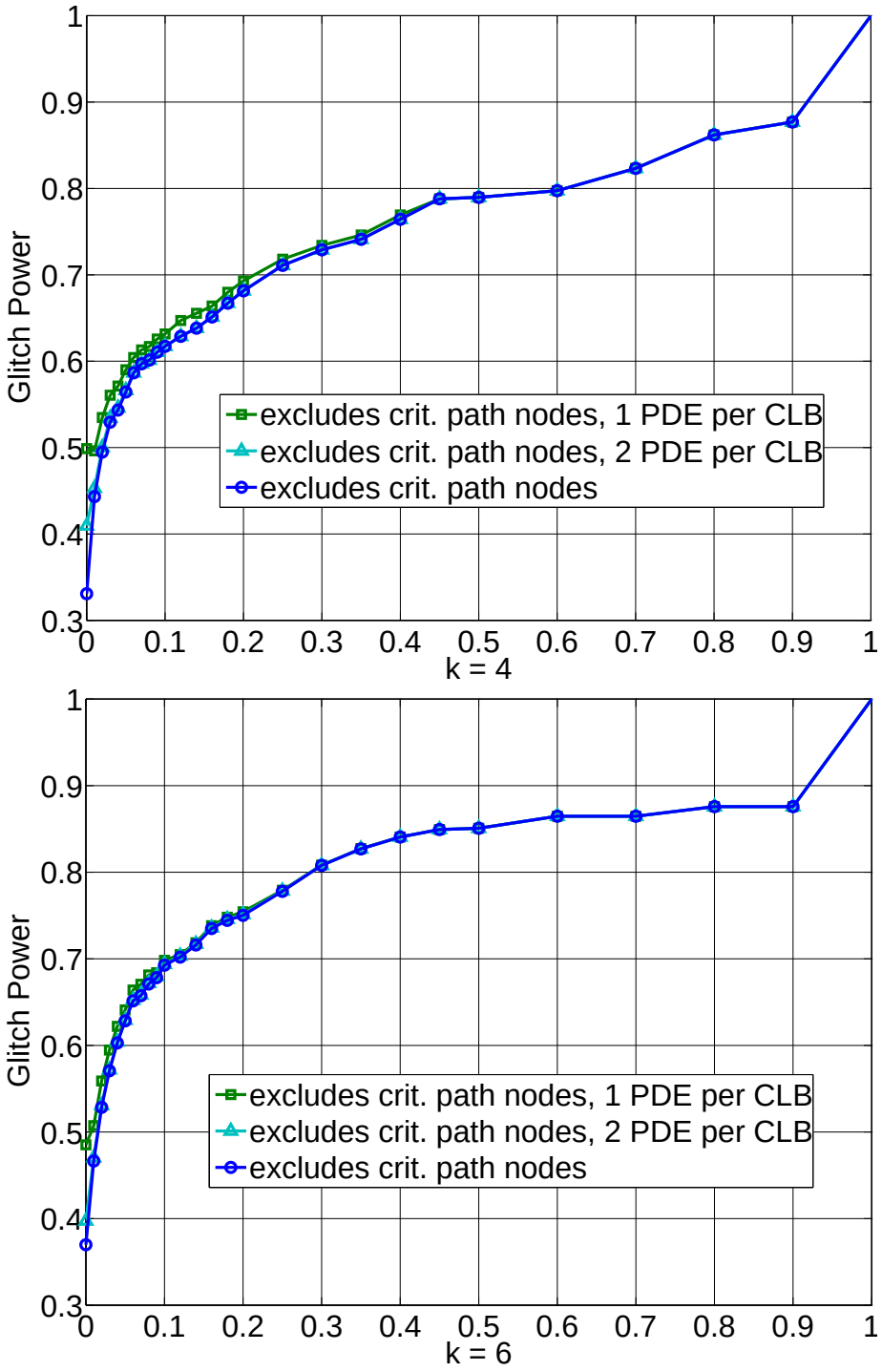

Figure 6.3: Glitch Power Reduction, PDE Sharing

it is clear from the other curve that the additional power savings from doing GlitchLess on critical path nodes is not worth the performance impact it incurs, as the power-period product goes above 1.0, and is never better than the case where critical path is not allowed to increase.

Figure 6.5 shows the power-period product plots when PDE overhead is added, and Figure 6.6 shows the number of PDEs used on a log scale. Roughly a third of the savings can be obtained using less than 10 PDEs on average, for both k=4 and 6, confirming that there are indeed very few nodes with high glitching power. When the critical path is allowed to increase, more PDEs are needed. As the threshold is lowered below 20%, the number of PDEs used increases dramatically in all three cases, and the savings from the diminishing rate of return of each added PDE is over-compensated by the PDE's own power overhead. As a result, savings at low thresholds are gone. It is interesting to see that for threshold values below 0.3, the 1-PDE-per-CLB configuration yields slightly better power savings because it limits the number of PDEs used. The 2-PDE-per-CLB configuration needs twice as many PDEs to be inserted into the architecture, but only improves the averages by a negligible amount for k=4, and not at all for k=6. This means having 1 PDE per CLB is good enough for the benchmark suite used for this thesis.

The best savings (P_{final} for 1-PDE-per-FF, $P_{final1PDE}$ for 1-PDE-per-CLB) for each circuit, where period is not allowed to increase, is presented in Table 6.5. The limit (P_{limit}) refers to the best possible savings achievable (assuming 100% glitch elimination with no PDE overhead), and the percentage of the limit achieved is also shown (%). Some circuits cannot be improved, so their limiting case is close to 100% because glitching is only a small fraction of total dynamic power. There are several cases (shown in bold) where the result from 1-PDE-per-CLB is better than 1-PDE-per-FF due to multiple nodes sharing the same PDE. Overall, the technique is able to remove on average 13% to 20% of glitching power,

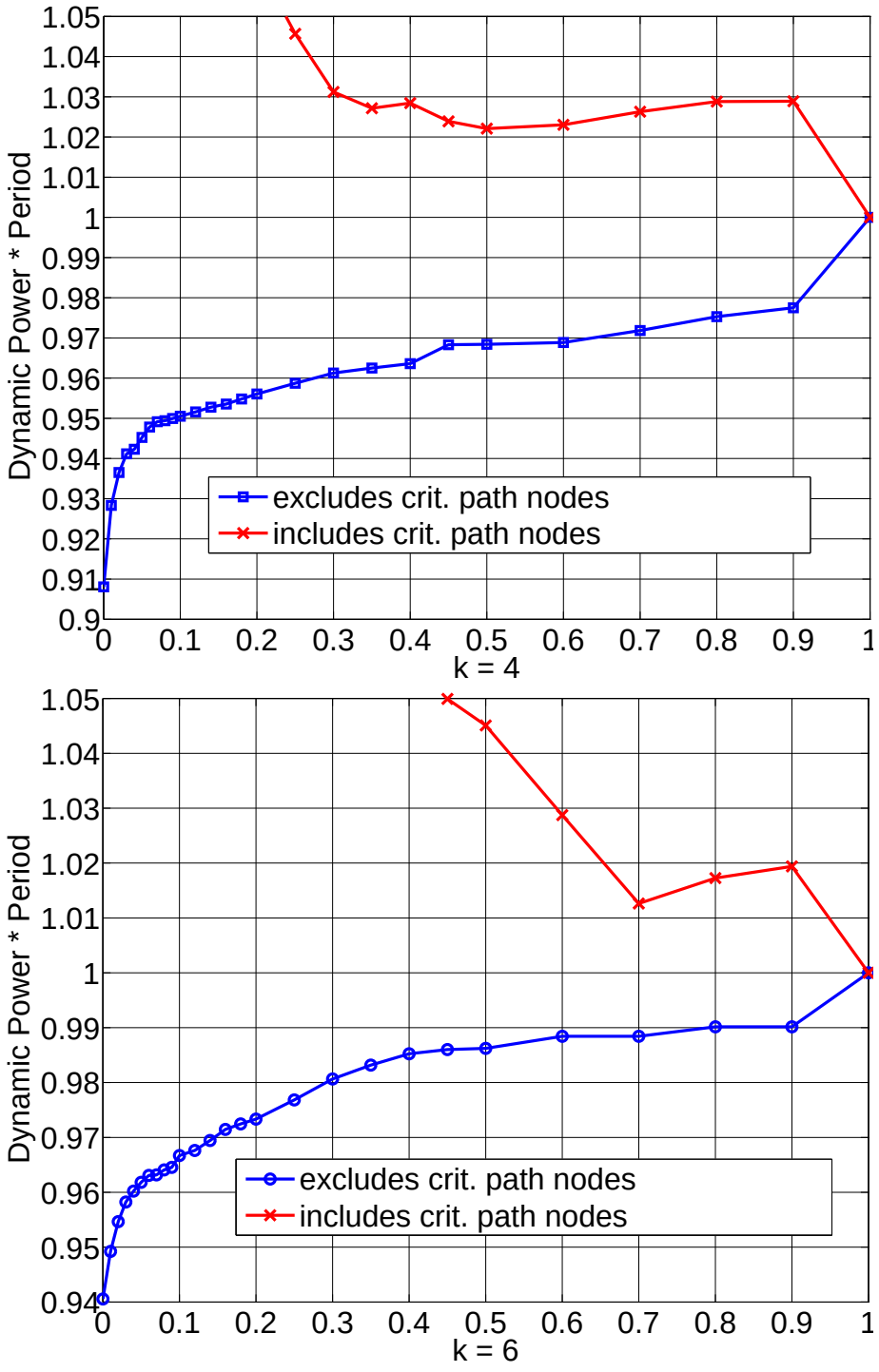

Figure 6.4: Power Plot Without PDE Overhead

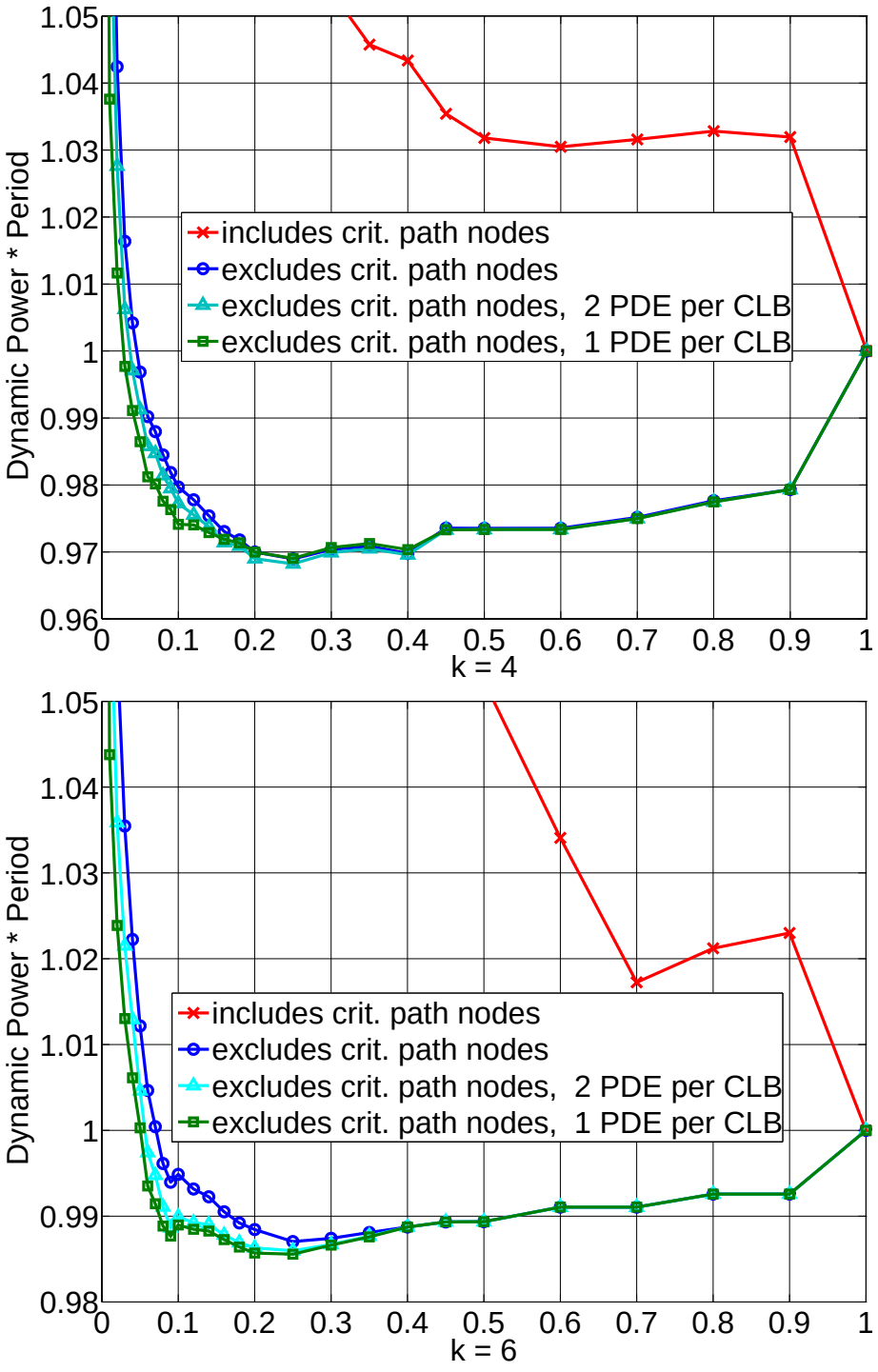

Figure 6.5: Power Plot With PDE Overhead

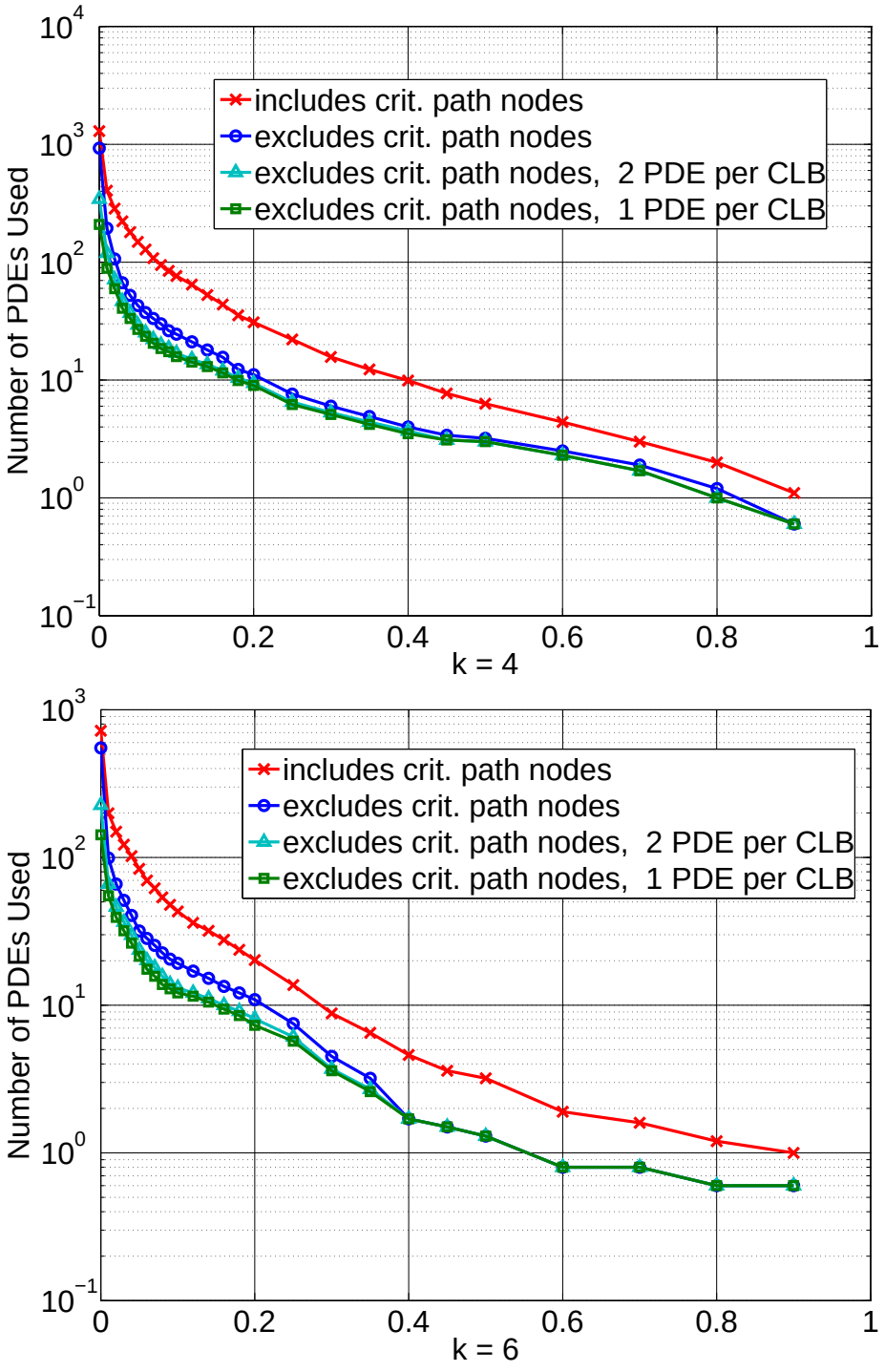

Figure 6.6: PDE Usage for GlitchLess

circuit	k = 4							k = 6						
	P_{final}	%	$P_{final1PDE}$	%	$P_{final2PDE}$	%	P_{limit}	P_{final}	%	$P_{final1PDE}$	%	$P_{final2PDE}$	%	P_{limit}
bigkey	100	0	100	0	100	0	98.2	100	0	100	0	100	0	98.4
clma	98.0	32.3	98.2	29.0	98.1	30.6	93.8	98.6	23.0	98.3	**27.9**	98.4	26.2	93.9
diffeq	100	0	100	0	100	0	96.6	100	0	100	0	100	0	97.9
dsip	100	0	100	0	100	0	98.8	100	0	100	0	100	0	98.6
elliptic	86.7	63.3	87.5	59.5	86.7	63.3	79.0	92.2	49.1	92.2	49.1	92.0	50.3	84.1
frisc	94.6	41.5	94.6	41.5	94.6	41.5	87.0	95.1	36.6	95.1	36.6	95.1	36.6	86.6
s298	100	0	100	0	100	0	78.3	98.3	7.3	98.3	7.3	98.3	7.3	76.8
s38417	85.7	53.4	85.7	53.4	85.7	53.4	73.2	98.8	8.3	96.6	**23.6**	97.2	19.4	85.6
s38584.1	99.5	6.4	98.6	**17.9**	98.6	17.9	92.2	99.6	10.0	99.6	10.0	99.6	10.0	96.0
tseng	99.6	2.6	99.6	2.6	99.6	2.6	84.8	100	0	100	0	100	0	91.6
average		20.0		20.4		20.9			13.4		15.4		15.0	
geomean	96.9		96.3		96.2		87.7	98.2		98.0		98.0		90.7

Table 6.5: Dynamic Power after GlitchLess

or up to 63% for individual circuits. The $P_{final2PDE}$ column represents the savings for the 2-PDE-per-CLB scheme, and the performance is almost identical to the 1-PDE-per-CLB scheme.

6.3 GlitchLess Savings After CSS+DP+GL Run

Choice "3" in Figure 4.3 offers the complete flow. This section shows the effectiveness of GlitchLess with prior CSS perturbation on glitching activity.

Table 6.6 provides GlitchLess results after CSS+DP+GlitchLess are run with 4 PDEs per CLB. This particular scheme is chosen because 4 PDEs per CLB is enough for most circuits to reach the same period reduction obtainable via the 1-FF-per-PDE scheme. Of the 4 PDEs, GlitchLess is allowed to use at most 1 free PDE, since in the last section it is shown using additional PDEs does not result in additional savings. To compare against GlitchLess only results in the last section, the power overhead of CSS and DP PDEs are not included to be consistent (CSS and DP are not done for GlitchLess only tool flow). For example, if a CLB has 2 PDEs used for CSS and 1 PDE used for GlitchLess, the dynamic power numbers presented in Table 6.6 only takes into account the 1 PDE used for GlitchLess and glitching power generated by all nodes inside the CLB.

In Table 6.6, the $P_{initial}$ column shows the power due to functional and glitching activity (excluding CSS/DP PDEs) after CSS+DP is performed. The P_{final} column shows the power after GlitchLess, including functional/glitching activity and GlitchLess PDE overhead. The P_{limit} represent the best possible savings, similar to Table 6.5.

The average percent of glitching power eliminated is 18% and 15% for k=4 and 6. This is on par with the corresponding averages from the GlitchLess results (20% and 13%).

circuit	k = 4				k = 6			
	$P_{initial}$	P_{final}	%	P_{limit}	$P_{initial}$	P_{final}	%	P_{limit}
bigkey	102.5	102.5	0	98.2	100.8	100.8	0	98.4
clma	105.9	100.5	44.6	93.8	101.2	98.9	31.5	93.9
diffeq	99.3	99.3	0	96.6	99.5	99.5	0	97.9
dsip	100.4	100.4	0	98.8	100.4	100.4	0	98.6
elliptic	104.3	89.0	60.5	79.0	100.9	93.5	44.0	84.1
frisc	105.2	99.3	32.4	87.0	103.6	99.0	27.1	86.6
s298	111.1	110.8	0.9	78.3	106.9	105.6	4.3	76.8
s38417	102.1	91.5	36.7	73.2	111.0	103.2	30.7	85.6
s38584.1	101.8	101.2	6.2	92.2	101.7	101.2	8.8	96.0
tseng	102.5	102.5	0	84.8	104.0	104.0	0	91.6
average			18.1				14.6	
geomean	103.6	101.8		87.7	102.9	101.7		90.7

Table 6.6: Dynamic Power after Full Run, Excluding CSS+DP PDE Overhead

6.4 Summary of Results

This chapter detailed the performance and power improvements for doing CSS+DP only and for GlitchLess only. CSS alone reduces period by 13% and 17% on average for k=4 and 6, respectively, and up to 37.7% for individual circuits. Delay Padding further benefits four circuits for both LUT sizes, and the average improvement is 18% and 20% of the original period. The average period reduction for all of the circuits that benefit from DP is an additional 10% of the original period on top of the CSS only results, and up to an additional 23% of the original period of individual circuits.

Power overhead due to PDEs used for CSS and DP is significant. The power overhead after CSS and DP are done, expressed as a percentage of the original power, is 124% and 160% for k=4 and 6, respectively.

PDE sharing helps to reduce the power overhead, but reduces the improvement on period. Usually, the circuits that have lower FF density benefit more from PDE sharing without having to give up performance improvements. For the circuits that benefit from

delay padding, PDE sharing takes away the flexibility to assign every FF in the CLB with a different skew, and the penalty on performance improvements is more severe. On average, a 1-PDE-per-CLB scheme gives on average 10% period reduction for both k values, and the power overhead is 69% and 78% for k=4 and 6, respectively. To retain all of the performance improvements, 6 and 7 PDEs need to be available for sharing for k=4 and 6. The power overhead in this case is 105% and 126%. The power overhead of a previous approach that used 4 global clocks to provide skews is estimated to be 161% and 176% for k=4 and 6, and is on the same order of magnitude as our approach.

For GlitchLess only, the percentage of glitching power that can be reduced averages at 70% and 60% for k=4 and 6. This translates to an average dynamic power reduction of 9% and 6% without PDE power overhead. In order to achieve 100% glitching elimination, the critical path must be allowed to increase, but this performance penalty is not worth the extra power benefit. Including PDE overhead, 13% to 20% of glitching power can be reduced on average, and up to 63% for individual circuits. The power savings for the 1-PDE and 2-PDE per CLB schemes are almost identical to the 1-PDE-per-FF scheme.

For full run results including CSS, DP and GlitchLess, average glitch elimination of 18% and 15% is achieved for k=4 and 6, and this is on par with the averages obtained in the GlitchLess only case.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis presented an architecture change and associated tool flow to consider both power and performance optimization. A programmable delay element (PDE) added to each FF clock input can be used to satisfy CSS, delay padding and GlitchLess simultaneously. The combined CSS and DP algorithm uses iterative binary search to repeatedly relax hold-time constraints and reduce the period. CSS alone can improve period by an average of 13% and 17% of the original period for k=4 and 6, respectively. With CSS and DP, the average improvement is 18% and 20% of the original period. The average period reduction for the circuits that benefit from DP is 10% of the original period. When there is a restriction on the number of PDEs in each CLB, it is found that a maximum of 7 PDEs per CLB is sufficient to achieve the same result as when each FF has access to a dedicated PDE. However, most circuits that cannot benefit from DP achieve maximum optimization with just one or two PDEs per CLB.

PDE power consumption overhead with CSS is significant and is an estimated average of 124% (k=4) and 160% (k=6) of the original circuit's dynamic power. This is on par with previous work that used 4 global clocks to provide skews. When there is a restriction on the number of PDEs each CLB can share, the power overhead can be reduced to an average of 70% for both LUT sizes when each CLB is restricted to share just 1 PDE.

For glitch reduction, the modified GlitchLess traverses the circuit in a depth-first fashion, assigning PDEs only to nodes with glitching power that is higher than user specified threshold. On average, it can reduce 13% to 20% of glitching power, and up to 63% for individual circuits. The average is low because, for several circuits, nodes with high glitching power are on critical paths and cannot tolerate increased delays. Also, the PDEs consume significant power, making it impractical to apply GlitchLess to every circuit node. Finally, for a full run, including CSS, DP and Glitchless, the average percentage of glitching eliminated is 15% to 18%, and this is on par with the averages obtained in the GlitchLess only case.

To provide accurate glitch estimation, the modified ACE algorithm uses simulation results from Cadence combined with glitch pulse-width calculations and binning. The results showed that the original method can underestimate glitching by as much as 48% while overestimating it by up to 15%.

7.2 Future Work

The MCNC circuits used in this thesis are small circuits that are more than 15 years old, and take up a small area of at most 30 x 30 CLBs. Larger and more modern circuits will provide a better measurement of the effectiveness of CSS, DP and GlitchLess. Since most modern circuits contain memory (RAM) and digital signal processing (DSP) functional blocks, the nature of modern circuits is significantly different. Therefore, once modern circuits are obtained, one will need to change ACE to simulate RAM/DSP blocks and to modify the data structures in PGR to include multi-output nodes.

Chapter 6 showed that PDE overhead incurs a large power overhead with CSS and DP, and it limits GlitchLess performance. Therefore, it is important to have a power-efficient PDE to drive down the cost of implementing CSS and DP.

In terms of algorithm design, PGR can potentially be improved in the following areas.

7.2.1 CSS

During CSS, the quantization process after each iteration of the binary search adjusts each skew to the nearest delay value. This minimizes the difference between the adjusted skew and the original (continuous) skew. However, a pair of skews (for example, x_j and x_i) may be adjusted in a way such that the period needs to be increased to satisfy the setup constraints. An alternative way to do skew adjustment is a random "snapping" of skews. For example, a skew of 0.2345 would be randomly adjusted to 0.2 or 0.3, with a discretization level of 0.1. This approach should be implemented to compare to the existing approach.

During the second pass of CSS+DP when CLBs are restricted to PDE sharing, the FFs are randomly distributed to PDEs for CLBs that needed more distinct skew values in the first pass than the number of available PDEs. This simple solution is easier to implement but may not produce the best impact on the period. There may be other sharing schemes that can even further reduce the power overhead while minimizing the impact on period. For example, the number of PDEs required may be reduced by extending the optional two-pass approach into a multi-pass algorithm, regardless of whether the number of PDEs per CLB satisfies the optimum skew schedule from the first pass or not. The reason to do many passes is based on the fact that there are many clock schedules that can satisfy the same period. After each pass, the algorithm can check each CLB to see if there are PDEs that have the same skew. If found, the FFs that share these PDEs can be further combined into a "larger" node in the CSS graph, and the CSS+DP procedure is repeated. This may be more beneficial as the number of PDEs shared by each CLB increases.

7.2.2 Glitch Estimation

When a node is assigned a GlitchLess PDE, its arrival time changes due to timing margins, and its output activity changes due to glitch elimination. This will affect the glitching created on downstream nodes. Since PGR and ACE are separate, standalone tools, it is not feasible to update activity estimation after GlitchLess is done on every node. Therefore, it is beneficial to integrate ACE into the PGR/VPR framework, so that not only glitch propagation can be done (Chapter 3.5), but glitch estimation can be updated more frequently to provide better ranking for each logic level of the circuit.

Bibliography

[1] Sequential Interactive Synthesis (SIS) abstract page, 1994. `http://embedded.eecs.berkeley.edu/research/sis/abstract.html`.

[2] iFAR - intelligent FPGA architecture repository, 2008. `http://www.eecg.utoronto.ca/vpr/architectures/`.

[3] Predictive technology model (PTM) website, 2008. `http://www.eas.asu.edu/~ptm/`.

[4] Christoph Albrecht. Efficient incremental clock latency scheduling for large circuits. In *Conference on Design, Automation and Test in Europe*, pages 1091–1096, Munich, Germany, 2006. European Design and Automation Association.

[5] Altera Corporation. Achieving low power in 65-nm Cyclone III FPGAs, 2007. `http://www.altera.com/literature/wp/wp-01016.pdf`.

[6] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[7] Lei Cheng, Deming Chen, and Martin D. F. Wong. GlitchMap: an FPGA technology mapper for low power considering glitches. In *Design Automation Conference*, pages 318–323, San Diego, California, 2007. ACM.

[8] Seung Hoon Choi, Bipul C. Paul, and Kaushik Roy. Novel sizing algorithm for yield improvement under process variation in nanometer technology. In *Design Automation Conference*, pages 454–459, San Diego, CA, USA, 2004. ACM.

[9] Jason Cong and Chang Wu. FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits. In *Design Automation Conference*, pages 644–649, Anaheim, California, United States, 1997. ACM.

[10] T. H. Cormen. *Introduction to Algorithms*. MIT Press, 2001.

[11] R.B. Deokar and S.S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 407–410 vol.1, 1994.

[12] Quang Dinh, Deming Chen, and Martin D.F. Wong. A routing approach to reduce glitches in low power FPGAs. In *International Symposium on Physical Design*, pages 99–106, San Diego, California, USA, 2009. ACM.

[13] Xiao Patrick Dong and Guy Lemieux. PGR: Period and glitch reduction via clock skew scheduling, delay padding and GlitchLess. In *International Conference on Field-Programmable Technology (FPT)*, 2009.

[14] John P. Fishburn. Clock skew optimization. *IEEE Trans. Comput.*, 39(7):945–951, 1990.

[15] Eby G Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proc. IEEE*, 89:665—692, 2001.

[16] Shih-Hsu Huang, Chun-Hua Cheng, Chia-Ming Chang, and Yow-Tyng Nieh. Clock period minimization with minimum delay insertion. In *Design Automation Conference*, pages 970–975, San Diego, California, 2007. ACM.

[17] Shih-Hsu Huang and Yow-Tyng Nieh. Clock skew scheduling with race conditions considered. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4):45, 2007.

[18] Shih-Hsu Huang, Yow-Tyng Nieh, and Feng-Pin Lu. Race-condition-aware clock skew scheduling. In *Design Automation Conference*, pages 475–478, Anaheim, California, USA, 2005. ACM.

[19] J. Lamoureux, G. Lemieux, and S. Wilton. GlitchLess: dynamic power minimization in FPGAs through edge alignment and glitch filtering. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(11):1521–1534, 2008.

[20] J. Lamoureux and S.J.E. Wilton. Activity estimation for Field-Programmable gate arrays. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2006.

[21] Charles Leiserson and James Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, June 1991.

[22] Chuan Lin and Hai Zhou. Clock skew scheduling with delay padding for prescribed skew domains. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 541–546, 2007.

[23] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, and Jonathan Rose. VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 133–142, Monterey, California, USA, 2009. ACM.

[24] J.L. Neves and E.G. Friedman. Optimal clock skew scheduling tolerant to process variations. In *Design Automation Conference*, pages 623–628, 1996.

[25] Peichen Pan and C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. *ACM Trans. Des. Autom. Electron. Syst.*, 3(3):437–462, 1998.

[26] Kara Poon. Power estimation for field-programmable gate arrays. Master's thesis, University of British Columbia, Vancouver, BC, 2002.

[27] P. Sedcole, J.S. Wong, and P.Y.K. Cheung. Modelling and compensating for clock skew variability in FPGAs. In *International Conference on Field-Programmable Technology (FPT)*, pages 217–224, 2008.

[28] Deshanand P. Singh and Stephen D. Brown. Constrained clock shifting for field programmable gate arrays. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 121–126, Monterey, California, USA, 2002. ACM.

[29] Deshanand P. Singh and Stephen D. Brown. Integrated retiming and placement for field programmable gate arrays. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 67–76, Monterey, California, USA, 2002. ACM.

[30] S. Sivaswamy and K. Bazargan. Statistical generic and chip-specific skew assignment for improving timing yield of FPGAs. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 429–434, 2007.

[31] Satish Sivaswamy and Kia Bazargan. Statistical analysis and process variation-aware routing and skew assignment for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 1(1):1–35, 2008.

[32] A. Takahashi and Y. Kajitani. Performance and reliability driven clock scheduling of sequential logic circuits. In *Asia and South Pacific Design Automation Conference*, pages 37–42, 1997.

[33] Baris Taskin and Ivan S. Kourtev. Delay insertion method in clock skew scheduling. In *International Symposium on Physical Design*, pages 47–54, San Francisco, California, USA, 2005. ACM.

[34] Paul Teehan. Reliable high-throughput FPGA interconnect using source-synchronous surfing and wave pipelining. Master's thesis, University of British Columbia, Vancouver, BC, 2008.

[35] Mark Yamashita. A combined clustering and placement algorithm for FPGAs. Master's thesis, University of British Columbia, Vancouver, BC, 2007.

[36] Chao-Yang Yeh and Malgorzata Marek-Sadowska. Skew-programmable clock design for FPGA and skew-aware placement. In *ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays*, pages 33–40, Monterey, California, USA, 2005. ACM.