

Modeling and Reduction of Dynamic Power in Field-Programmable Gate Arrays

by

Julien Lamoureux

B.Sc., The University of Alberta, 2001

M.A.Sc., The University of British Columbia, 2003

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

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia

November 2007

© Julien Lamoureux, 2007

Abstract

Field-Programmable Gate Arrays (FPGAs) are one of the most popular platforms for implementing digital circuits. Their main advantages include the ability to be (re)programmed in the field, a shorter time-to-market, and lower non-recurring engineering costs. This programmability, however, is afforded through a significant amount of additional circuitry, which makes FPGAs significantly slower and less power-efficient compared to Application Specific Integrated Circuits (ASICs).

This thesis investigates three aspects of low-power FPGA design: switching activity estimation, switching activity minimization, and low-power FPGA clock network design. In our investigation of switching activity estimation, we compare new and existing techniques to determine which are most appropriate in the context of FPGAs. Specifically, we compare how each technique affects the accuracy of FPGA power models and the ability of power-aware CAD tools to minimize power. We then present a new publicly available activity estimation tool called ACE-2.0 that incorporates the most appropriate techniques. Using activities estimated by ACE-2.0, power estimates and power savings were both within 1% of results obtained using simulated activities. Moreover, the new tool was 69 and 7.2 times faster than circuit simulation for combinational and sequential circuits, respectively.

In our investigation of switching activity minimization, we propose a technique for reducing power in FPGAs by minimizing unnecessary transitions called glitches. The technique involves adding programmable delay elements at inputs of the logic elements of the FPGA to align the arrival times, thereby preventing new glitches from being generated. On average, the proposed technique eliminates 87% of the glitching, which reduces overall FPGA power by 17%. The added circuitry increases the overall FPGA area by 6% and critical-path delay by less than 1%.

Finally, in our investigation of low-power FPGA clock networks, we examine the tradeoff between the power consumption of FPGA clock networks and the cost of the constraints they impose on FPGA CAD tools. Specifically, we present a parameterized framework for describing FPGA clock networks, we describe new clock-aware placement techniques, and we perform an empirical study to examine how the clock network parameters affect the overall power consumption of FPGAs. The results show that the techniques used to produce a legal placement

can have a significant influence on power and delay. On average, circuits placed using the most effective techniques dissipate 9.9% less energy and were 2.4% faster than circuits placed using the least effective techniques. Moreover, the results show that the architecture of the clock network is also important. On average, FPGAs with an efficient clock network were up to 12.5% more energy efficient and 7.2% faster than other FPGAs.

Table of Contents

Abstract	i
List of Tables	vi
List of Figures	vii
List of Abbreviations	ix
Acknowledgments.....	x
Co-Authorship Statement.....	xi
Chapter 1	1
1.1 Motivation.....	1
1.2 FPGA Power Minimization	2
1.2.1 FPGA Architecture and Power Dissipation	2
1.2.2 FPGA Power Modeling.....	5
1.2.3 Low-Power FPGA Architecture	5
1.2.4 Low-Power FPGA CAD	6
1.2.5 Low-Power Techniques used in Commercial FPGAs	7
1.3 Focus and Contributions of this Thesis.....	9
1.3.1 Switching Activity Estimation for FPGAs	10
1.3.2 Switching Activity Minimization in FPGAs.....	13
1.3.3 FPGA Clock Networks	14
1.4 Thesis Organization	16
References for Chapter 1.....	17
Chapter 2	21
2.1 Introduction	21
2.2 Background	23
2.2.1 Terminology.....	23
2.2.2 Activity Estimation	23
2.2.3 Existing Activity Estimation Tools.....	26
2.3 Empirical Study on Activity Estimation for FPGAs.....	26
2.3.1 Accuracy and Speed of Probabilistic-Based Activity Estimation.....	26
2.3.2 FPGA Power Modeling.....	30
2.3.3 Power-Aware CAD for FPGAs	32
2.4 ACE-2.0: A New Activity Estimator	33
2.4.1 Phase 1	34
2.4.2 Phase 2	35
2.4.3 Phase 3	40
2.4.4 ACE-2.0 Results	41
2.5 Conclusions and Future Work.....	42
References for Chapter 2.....	45
Chapter 3	47
3.1 Introduction	47
3.2 Background	49
3.2.1 Switching Activity Terminology	49
3.2.2 Process, Voltage, and Temperature Variation	49
3.2.3 Existing Glitch Minimization Techniques	50

3.3 Glitching in FPGAs	50
3.3.1 Switching Activity Breakdown	51
3.3.2 Pulse Width Distribution	51
3.3.3 Power Dissipation of Glitches	52
3.3.4 Potential Power Savings	53
3.4 Proposed Glitching Elimination Techniques	54
3.4.1 Glitch Elimination Techniques	54
3.4.2 Architectural Alternatives	55
3.4.3 Programmable Delay Element	58
3.4.4 CAD Algorithms	59
3.4.5 PVT Variation Techniques	63
3.5 Experimental Framework	67
3.5.1 Switching Activity Estimation	67
3.5.2 Area, Delay, and Power Estimation	68
3.5.3 Architecture Assumptions and Benchmark Circuits	69
3.6 Scheme Calibration	69
3.6.1 Scheme 1 Calibration	69
3.6.2 Scheme 2 Calibration	71
3.6.3 Scheme 3 Calibration	71
3.6.4 Scheme 4 Calibration	74
3.6.5 Scheme 5 Calibration	75
3.6.6 Summary	76
3.7 Results	77
3.7.1 Area Overhead	77
3.7.2 Power Overhead	78
3.7.3 Delay Overhead	78
3.7.4 Overall Power Savings (Assuming No Variation)	79
3.7.5 Overall Power Savings (Assuming Variation)	80
3.8 Conclusions and Future Work	81
References for Chapter 3	83
Chapter 4	85
4.1 Introduction	85
4.2 Background and Previous Work	87
4.2.1 Background	87
4.2.2 Previous Work	90
4.3 Parameterized FPGA Clock Network Framework	91
4.3.1 Clock Sources	91
4.3.2 Global and Local Clock Regions	92
4.3.3 First Network Stage	92
4.3.4 Second Network Stage	94
4.3.5 Third Network Stage	95
4.4 Clock-aware FPGA CAD	96
4.4.1 Clock-Aware Placement	96
4.5 Clock-Aware Placement Results	101
4.5.1 Experimental Framework	101
4.5.2 Placement Results	104
4.6 Clock Network Architecture Results	107

4.6.1 Clocks per Logic Block (W_{lb})	107
4.6.2 Clocks per Rib (W_{rib})	110
4.6.3 Global Channel Width (W_{global}).....	112
4.6.4 Number of Clock Regions (nx_region, ny_region)	113
4.7 Conclusions and Future Work.....	116
References for Chapter 4.....	118
Chapter 5	119
5.1 Summary and Contributions	119
5.2 Relationship between Contributions	124
5.3 Future Work	125
References for Chapter 5.....	129
Appendix A.....	130
Appendix B	134

List of Tables

Table 2.1: Accuracy of speed of switching activity estimation techniques.	29
Table 2.2: Power estimates using ACE, Sis, and simulation.	31
Table 2.3: Power savings using each activity estimator.....	33
Table 2.4: ACE-2.0 results.....	42
Table 2.5: Power estimates using ACE-2.0.....	43
Table 2.6. Power savings using ACE-2.0.....	44
Table 3.1. Breakdown of Switching Activity.....	51
Table 3.2. FPGA power with and without glitching.	53
Table 3.3: Delay insertion parameters.....	57
Table 3.4: Summary of programmable delay element values.....	76
Table 3.5: CLB area overhead (excluding global interconnect).	77
Table 3.6: Overall area overhead.	77
Table 3.7: Average power overhead (%)	78
Table 3.8: Average delay overhead.....	79
Table 3.9: % Glitch elimination of each scheme.	80
Table 3.10: Overall power savings.....	80
Table 4.1: Clock network parameters.	95
Table 4.2. Benchmark circuit names and attributes.	103
Table 4.3. Techniques used by each placer.....	104
Table 4.4. Overall energy per clock cycle.....	105
Table 4.5. Impact of clock constraints for each placer.....	106
Table 4.6: Logic utilization vs. number of clocks per logic block (W_{lb}).....	108
Table 4.7. Critical-path delay and energy vs. number of clocks per logic block (W_{lb}).....	109
Table 4.8. Overall energy per cycle vs. the number of clocks per rib (W_{rib}).	111
Table 4.9. Clock energy, routing energy, and critical-path delay vs. clocks per rib (W_{rib}).....	112
Table 4.10. Energy and delay vs. global channel width (W_{global}).....	113
Table 4.11. Overall energy vs. the number of regions ($n_x_{region} \times n_y_{region}$).....	116
Table 4.12: Clock energy, routing energy, and critical-path delay vs. clocks per rib (W_{rib}).....	116
Table 5.1: Breakdown of the three contributions.....	123
Table 5.2: Summary of the key results.....	124
Table A.1. Benchmark Circuit Attributes.	132
Table B.1. Downloading instructions of research tools.	134

List of Figures

Figure 1.1: Conceptual FPGA models.....	3
Figure 1.2: Breakdown of dynamic power consumption in Xilinx Spartan-3 FPGAs [9].	4
Figure 2.1: (a) Spatial correlation (b) Temporal correlation.....	25
Figure 2.2: Example of unrolling next state logic.....	25
Figure 2.3: Activity comparison framework.....	27
Figure 2.4: Power modeling framework.	30
Figure 2.5: Power-aware CAD framework.....	32
Figure 2.6: ACE-2.0 pseudo code.....	34
Figure 2.7: Simplified simulation pseudo-code.	35
Figure 2.8: Example of a partially collapsed network.	37
Figure 2.9: Example of BDD pruning.....	38
Figure 2.10: Correlation vs. execution time.....	39
Figure 2.11: BDD pruning pseudo code.	40
Figure 3.1. Pulse width distribution of glitches.	52
Figure 3.2. Normalized power vs. pulse width.	52
Figure 3.3. Removing glitches by delaying early-arriving signals.	54
Figure 3.4. Delay insertion schemes	55
Figure 3.5. Programmable delay element.	58
Figure 3.6. Pseudo-code for calculating the delay needed to align the inputs.....	59
Figure 3.7. Pseudo-code for assigning delays in Scheme 1.....	60
Figure 3.8. Pseudo-code for assigning delays in Scheme 2.....	60
Figure 3.9. Pseudo-code for assigning additional delays in Scheme 3.....	61
Figure 3.10. Pseudo-code for assigning additional delays in Scheme 4.....	62
Figure 3.11. Pseudo-code for assigning additional delays in Scheme 5.....	62
Figure 3.12: Example that inserted delays need to scale with the remaining delays.....	63
Figure 3.13: Schematic of the previous programmable delay element [9].	64
Figure 3.14: Rise and fall times of delay element from [9] considering process variation	65
Figure 3.15: Rise times of the programmable delay element considering process variation.....	66
Figure 3.16: Minimum Transistor Equivalents (MTE) area model [19].	68
Figure 3.17. Glitch elimination vs. minimum LUT input delay for Scheme 1.....	70
Figure 3.18. Glitch elimination vs maximum LUT input delay for Scheme 1.	70
Figure 3.19. Glitch elimination vs. number of input delay elements per LUT for Scheme 1.....	71
Figure 3.20. Glitch elimination vs. maximum LUT input delay for Scheme 3.....	72
Figure 3.21. Glitch elimination vs. number of input delay elements per LUT for Scheme 3.....	73
Figure 3.22: Glitch elimination vs. minimum LUT output delay for Scheme 3.....	73
Figure 3.23: Glitch elimination vs. maximum LUT output delay for Scheme 3.....	74
Figure 3.24: Glitch elimination vs. number of input delay elements per LUT for Scheme 4.	75
Figure 3.25: Glitch elimination vs. number of bank delay elements for Scheme 5.....	76
Figure 3.26: Glitch elimination and power savings vs. β	81
Figure 4.1: Example clock distribution networks.....	88
Figure 4.2: Commercial FPGA clock networks.....	88
Figure 4.3: Multiplexer structure of Stratix II clock networks.	89
Figure 4.4: Control block for local and global clock signals.....	89

Figure 4.5: Two examples of a 7×5 crossbar network.	90
Figure 4.6: Example FPGA with 3×3 clock regions.	92
Figure 4.7: Connections from the periphery to the center of regions.	93
Figure 4.8: Global clock connections from the center of the FPGA to the center of regions.	94
Figure 4.9: Second stage of the clock network.	94
Figure 4.10: Third stage of the clock network.	95
Figure 4.11. Pseudo-code description of the static clock resource assignment technique.	99
Figure 4.12. Pseudo-code description of the dynamic clock resource assignment technique. ...	100
Figure 4.13. Pseudo-code description of the routine used to reassign clock nets.	100
Figure 4.14: Example of buffer sizing and sharing in the clock network.	102
Figure 4.15: Clock network area vs. W_{lb}	109
Figure 4.16: Clock network area vs. W_{rib}	111
Figure 4.17: Clock network area vs. $W_{global} + W_{local}$	113
Figure 4.18: Clock network area vs. number of clock regions.	114
Figure 4.19: Clock network area vs. number of clock regions (with adjusted W_{local}).	115

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BLE	Basic Logic Element
CAD	Computer-Aided Design
CLB	Configurable Logic Block
DLL	Delay Locked Loop
IC	Integrated Circuit
ISCAS	International Symposium on Circuits and Systems
FF	Flip-flop
FPGA	Field Programmable Gate Array
LE	Logic Element (same as BLE)
LUT	Lookup Table
MCNC	Microelectronics Centre of North Carolina
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MTE	Minimum Transistor Equivalents
NMOS	n-channel MOSFET transistor
PLL	Phase Locked Loop
PMOS	p-channel MOSFET transistor
PVT	Process, Voltage, and Temperature
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random Access Memory
TSMC	Taiwan Semiconductor Manufacturing Company
URL	Uniform Resource Locator
VPR	Versatile Place and Route

Acknowledgments

I would first like to thank my academic advisor, Steve Wilton. Throughout my Ph.D., Steve has consistently provided sound guidance, technical advice, and opportunities to present my research and make new research contacts. Moreover, I thank him for always being available, enthusiastic, and professional. I consider myself extremely fortunate to have had the opportunity to work with Steve throughout my graduate studies.

I would also like to thank the other members of Steve's research group: Usman Ahmed, Nathalie Chan King Choy, Scott Chin, Zion Kwok, Andrew Lam, Cindy Mark, Brad Quinton, and Andy Yan; as well as the other members of the SoC research group, especially: Marwa Hamour, Behnoosh Rahmatian, and Pedram Sameni for their valuable suggestions, insightful discussions, and for providing a friendly work environment that I looked forward to coming to every day. Special thanks are also due to Guy Lemieux for his collaboration in our glitch minimization study and for his valuable suggestions in general.

I would like to thank Altera and Xilinx for giving me the opportunity to present my research and for the constructive comments and suggestions provided by their employees. Particular thanks are due to Vaughn Betz, Paul Laventis, and Dave Lewis of Altera for the feedback they gave regarding activity estimation and clock distribution networks in FPGAs, and to Jason Anderson and Tim Tuan of Xilinx for the feedback they gave regarding activity estimation and glitch minimization.

Finally, I greatly appreciate the financial support for this project, which was provided by the Altera Corporation, Micronet R & D, and the Natural Sciences and Engineering Research Council of Canada (NSERC). Without their support, this research would not have been possible.

Co-Authorship Statement

Each of the three contributions described in this thesis have been submitted for publication. The first contribution entitled: “Fast activity estimation for power modeling and optimization in FPGAs,” has been submitted to the ACM Transactions on Design Automation of Electronic Systems and was co-authored by Julien Lamoureux and Steve Wilton. The study was designed collaboratively by Julien Lamoureux and Steve Wilton. Julien conducted the research, analyzed the data, and prepared the manuscript under the guidance of Dr. Steve Wilton.

The second contribution entitled: “Glitchless: dynamic power minimization in FPGAs through edge alignment and glitch filtering,” has been submitted to the IEEE Transactions on Very Large Scale Integration Systems and was co-authored by Julien Lamoureux, Dr. Guy Lemieux and Dr. Steve Wilton. The study was designed collaboratively by Julien Lamoureux, Dr. Guy Lemieux, and Dr. Steve Wilton. Julien conducted the research, analyzed the data, and prepared the manuscript under the guidance of Dr. Guy Lemieux and Dr. Steve Wilton.

Finally, the third contribution entitled: “On the interaction between power and flexibility of FPGA clock networks,” has been submitted to the ACM Transactions on Reconfigurable Technology and Systems and was co-authored by Julien Lamoureux and Dr. Steve Wilton. The study was designed collaboratively by Julien Lamoureux and Dr. Steve Wilton. Julien conducted the research, analyzed the data, and prepared the manuscript under the guidance of Dr. Steve Wilton.

Chapter 1

Introduction

1.1 Motivation

Field-Programmable Gate Arrays (FPGAs) are integrated circuits that can be programmed to implement any digital circuit. The main difference between FPGAs and conventional fixed logic implementations, such as Application Specific Integrated Circuits (ASICs), is that the designer/customer programs the FPGA on-site (in the field). For fixed logic implementations, the designer must create a layout mask and send it to a foundry to be fabricated. Creating a layout is labour-intensive and requires expensive CAD tools and experienced engineers. Moreover, fabrication takes several weeks and costs over a million dollars for the latest process technologies. This expense is compounded when bugs are found after fabrication and the chip must be re-fabricated. Using an FPGA instead of a fixed logic implementation eliminates these non-recurring engineering (NRE) costs and significantly reduces time-to-market. This makes FPGAs ideal for prototyping, debugging, and for small to medium volume applications.

On the other hand, FPGAs are slower and less efficient than fixed implementations due to the added circuitry that is needed to make them flexible. Programmable switches controlled by configuration memory occupy a large area in the FPGA and add a significant amount of parasitic capacitance and resistance to the logic and routing resources. Because of this, FPGAs are approximately 3 times slower, 20 times larger, and 12 times less power efficient compared to ASICs [1]. The area overhead, combined with research and development costs, increases the per-unit cost of FPGAs, which makes them less suited for high-volume applications. Moreover, the speed and power overhead precludes the use of FPGAs for high-speed or low-power applications. These limitations motivate research of new more efficient FPGA architectures and CAD techniques.

Many studies have focused on reducing the speed and area overhead of FPGAs. Important advancements include cluster-based logic blocks [2], which improve speed by grouping the basic logic elements of the FPGA into clusters with faster local interconnect; embedded memories [3], which reduce the speed and area overhead for applications with storage requirements; and embedded ALUs [4], which reduce the speed and area overhead for applications that perform

arithmetic operations. A significant number of studies have also focused on faster, more area-efficient programmable routing resources [5]. Important advancements have also been made to the FPGA CAD tools, which are used to map applications onto the programmable fabric of the FPGA. The Versatile Place and Route (VPR) tool, described in [2], yielded significant improvements in performance by improving on the existing clustering, placement, and routing algorithms. As another example, logic-to-memory mapping tools, described in [6]-[8], improved the area efficiency of FPGAs with embedded memories by packing parts of the application into unused memories before mapping the rest of the application into logic elements.

In recent years, however, the focus of the research has been shifting to lowering power consumption. As CMOS process technology scales down, the power density continues to increase due to higher chip operating frequencies, higher total interconnect capacitance per chip, and increasing leakage. Indeed, the International Technology Roadmap for Semiconductors (ITRS) has identified low-power design techniques as a critical technology need [8]. This, combined with improved reliability, lower operating and cooling costs, and the ever-growing demand for low-power portable communications and computer systems, is motivating new low-power techniques. This is especially true for FPGAs, which dissipate significantly more power than fixed-logic implementations. This thesis therefore focuses on minimizing power in FPGAs.

1.2 FPGA Power Minimization

In this section, we first review the structure of an FPGA, focusing on what makes FPGAs power-hungry. We then describe previous work in power modeling for FPGAs, low-power FPGA architectures, and low-power Computer-Aided Design (CAD) algorithms. We also identify which of these techniques are being used in current commercial FPGAs and CAD tools.

1.2.1 FPGA Architecture and Power Dissipation

FPGAs are made up of a large number of logic blocks, which implement the logic part of digital circuits, and a configurable routing fabric, which implements the connections between the logic blocks, as illustrated in Figure 1.1 (a). Modern FPGAs also have embedded fixed logic components such as memories and arithmetic logic units. These embedded components are typically implemented so as to occupy the same area as the programmable logic tiles and are often arranged in rows or columns as illustrated in the Figure 1.2 (b).

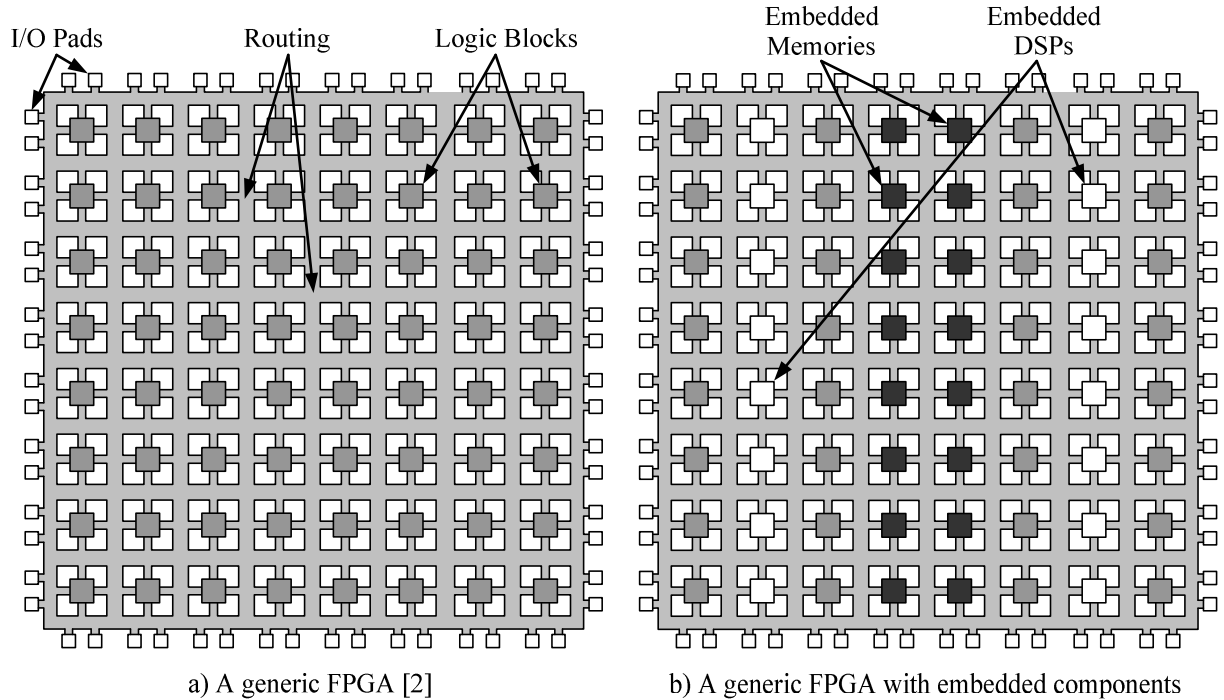


Figure 1.1: Conceptual FPGA models.

Flexibility is achieved in two ways. First, the function implemented by each logic element is configurable. Typically, FPGAs use lookup-tables to implement logic; the function implemented by each lookup-table can be configured using a set of memory cells (called *configuration bits*) [2]. The use of a lookup-table means that the logic element can implement any function of its inputs. Each logic element also typically contains a flip-flop which can be optionally used to register the lookup-table output.

The second way flexibility is achieved is through the use of a programmable routing fabric. In typical FPGAs, there are two levels of routing: logic elements are grouped into clusters (typically between 4 and 10 logic elements are in each cluster) and the clusters are connected using programmable routing [2]. This programmable routing fabric consists of fixed prefabricated metal tracks connected using programmable switches [5]. These programmable switches are controlled by configuration bits.

Together, the configurable logic elements and the configurable routing fabric make the FPGA flexible enough to implement virtually any digital circuit. However, as indicated earlier, this flexibility comes at the cost of increased power. As in all integrated circuits, FPGAs dissipate static and dynamic power. Static power is dissipated when current leaks from the power supply to ground through transistors that are in the “off-state” due to sub-threshold

leakage (from source to drain), gate-induced drain leakage, and gate direct-tunneling leakage. Dynamic power is dissipated when capacitances are charged and discharged during the operation of the circuit. Figure 1.2 shows the breakdown of static and dynamic core power in Xilinx Spartan-3 FPGAs [9], excluding embedded components. In these results, dynamic power accounted for 62% of the total power (and static power for 38%).

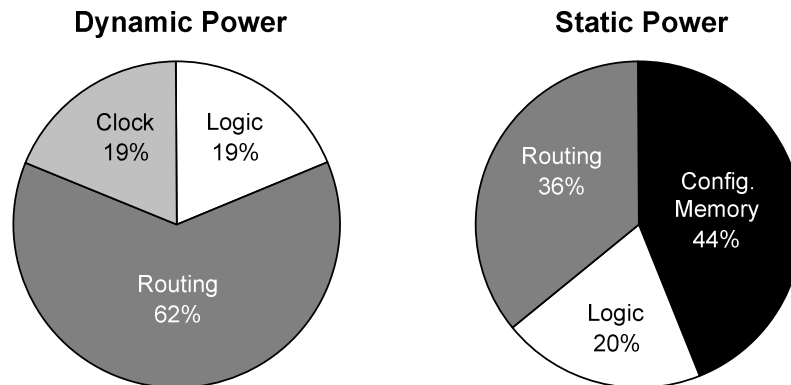


Figure 1.2: Breakdown of dynamic power consumption in Xilinx Spartan-3 FPGAs [9].

The extra circuitry that provides flexibility to an FPGA affects both the static and dynamic power dissipated by the FPGA. First consider static power. As described above, an FPGA contains a large number of configuration bits, both within each logic element and in the programmable routing used to connect logic elements. Each of these configuration bits dissipates static power. An ASIC does not contain any such programming bits, and thus would consume significantly less static power than an FPGA. In addition, since the lookup-tables consume significantly more transistors than the corresponding logic in an ASIC, the static power dissipated in the logic is larger in an FPGA than in an ASIC.

An FPGA's flexibility also implies more dynamic power dissipation than a non-programmable solution. In both an ASIC and FPGA, connections between gates are associated with some amount of parasitic capacitance due to the metal wire used to implement the connection as well as the driver and driven transistors. However, as described above, a connection in an FPGA also contains a large number of programmable switches. These switches significantly increase the parasitic capacitance on each wire segment and charging and discharging this parasitic capacitance consumes dynamic power.

For all the reasons outlined above, an FPGA is significantly less power-efficient than an ASIC. A recent study found that, for typical applications, an FPGA consumes 12 times more

power than the corresponding ASIC [1]. There has been much research addressing this; a summary of this research is outlined in the next subsections.

1.2.2 FPGA Power Modeling

A number of different FPGA power models have recently been presented in the literature. In [10], a detailed and flexible FPGA power model that estimates dynamic and static power of the logic, routing, and clock network for generic parameterized FPGA architectures is described. For static power, the model uses a first-order analytical technique which calculates subthreshold leakage based on transistor size and various technology-specific parameters. For dynamic power, the model extracts transistor-level capacitance information for each node of the application from the VPR CAD tool after it has completed routing. The average switching activity of each node is determined using a vectorless activity estimation tool called ACE. The study also examines how various FPGA architecture parameters affect overall power consumption. A similar FPGA power model, described in [11], also estimates dynamic and static power of cluster-based FPGAs but it calculates the power for each clock cycle using simulated activity information instead of vectorless average activities. This power model was also enhanced to support FPGAs with a programmable supply voltage in [12] and programmable threshold voltages in [13]. In [14]-[16], high-level FPGA power models that use macro-models to estimate power were described. These models characterize the power consumption of various FPGA components, such as adders, multipliers, and programmable logic, for low-power high-level synthesis or design space exploration.

1.2.3 Low-Power FPGA Architecture

A number of studies have also investigated low-power architecture design for FPGAs. Energy-efficient FPGA routing architectures and low-swing signaling techniques to reduce power were described in [17],[18]. In [19], a new FPGA routing architecture that utilizes a mixture of hardwired and traditional programmable switches was proposed. Both static and dynamic power were minimized by reducing the number of configurable routing elements in the routing fabric. In [20], a novel FPGA routing switch that can operate in high-speed, low-power, or sleep mode to reduce dynamic power in non timing critical logic during normal operation and standby power when the logic in use was presented. Similarly, in [21], power-gating was applied to the switches in the routing resources to reduce static power and duplicate routing resources (that use

either high or low Vdd) were used to reduce dynamic power. In [22], the design of energy-efficient modules for embedded components in FPGAs was described. Power was reduced by optimizing the number of connections between the module and the routing resources (based on the requirements of the embedded component) and by using reduced supply voltage circuit techniques.

1.2.4 Low-Power FPGA CAD

Large FPGAs have millions of programmable switches which are configured using a series (flow) of CAD tools. The FPGA CAD flow generally has five stages: high-level synthesis, technology mapping, clustering, placement, and routing. High-level synthesis converts a high-level description of the application to a netlist of logic gates and flip-flops. Technology mapping converts this netlist to a netlist of FPGA logic elements (LEs), which typically consist of lookup tables (LUTs) and flip-flops. Clustering packs the LEs into logic clusters; each cluster typically consists of between 8 and 10 LEs. Placement then assigns a physical location to each logic cluster. Finally, routing determines how to implement the connections between the logic clusters within the routing resources of the FPGA. These CAD tools can also have a significant impact on power consumption since they affect how directly the circuit elements of the application are connected. In this section, we review previous work on low-power CAD algorithms.

Low-power high-level synthesis algorithms for FPGAs were presented in [23],[24]. In [23], power is reduced by minimizing the total power of the operations and the size of the multiplexers that connect them. The algorithm described in [24] targets dual-Vdd FPGAs and minimizes power by assigning low-Vdd to as many operations as possible given resource and timing constraints. Several low-power technology mapping algorithms for FPGAs were presented in [25]-[31]. In general, power is minimized by absorbing as many high-activity nodes as possible when the gates are packed into LUTs and/or by minimizing node-duplication, which tends to increase the amount of interconnect between the LUTs. Low-power clustering techniques have been described in [32]-[34]. In these algorithms, power is minimized by absorbing as many small (low fanout) and high-activity nets as possible when the LUTs are packed into clusters (logic blocks). Absorbing small nets tends to reduce the total number of inter-cluster nets (which dissipate the most power) and absorbing high-activity nets reduces dynamic power since the local routing within clusters dissipates less power than the global routing between clusters. The low-power place and route techniques described in [35]-[38]

minimize power by reducing the distance between logic blocks connected by high-activity wires (during placement) and by routing high-activity wires as directly as possible (during routing). In [39], leakage power is minimized by choosing low-leakage LUT configurations. Power-aware algorithms for mapping logical memory to the physical FPGA embedded memories was described in [40]. The algorithms minimize dynamic power consumed by the embedded memories by evaluating a range of possible mappings and selecting the most power-efficient choice. Finally, in [41], the technology mapping, clustering, placement, and routing stages of the FPGA CAD flow were all made power-aware to determine which stages are most suited for reducing power and if the savings of the individual stages are cumulative. The study found that the power savings of the individual CAD stages were mostly cumulative, with a combined power-delay savings of 26%.

1.2.5 Low-Power Techniques used in Commercial FPGAs

The latest devices from FPGA vendors such as Altera and Xilinx incorporate a number of low-power architecture and CAD techniques from the literature along with many of the low-power device-level technologies from manufacturing. In this subsection, we review some of these techniques.

At the device level, Altera and Xilinx both utilize triple gate oxide technology, which provides a choice of three different gate thicknesses, to tradeoff between performance and static power [42],[43]. In earlier technologies, only two thicknesses were available. Transistors with thicker oxide were used for the large, higher voltage tolerant transistors in the I/O blocks and the thinner ones were used everywhere else. The new medium thickness oxide transistors provide slightly less performance than thin oxide transistors, but leak significantly less. In the latest FPGAs, these are used in the configuration memory and the switches that are controlled by this memory. Since the configuration memory remains static during the operation of the device, the oxide thickness does not affect the performance of the corresponding switches. To reduce dynamic power, FPGA vendors use a low-k dielectric between metal layers, which reduces the parasitic capacitance. This, in addition to smaller device geometries, reduces the average node capacitance and, correspondingly, dynamic power. Dynamic power of the core of the FPGAs can be reduced further by lowering the supply voltage because dynamic power has a quadratic relationship (CV^2f) with the supply voltage. Xilinx reduced the core supply voltage from 1.2V (in the Virtex 4) to 1.0V (in the Virtex 5), which reduces core power significantly. Similarly, the

core supply voltage of Altera Stratix III FPGAs can be selected (by the user) to be either 1.1V, for high performance, or 0.9V, for lower power.

Altera and Xilinx have also made a number of architecture-level changes to their latest devices to reduce static and dynamic power. Both vendors have recently increased the size of the LUTs within the logic blocks [42],[43]. By increasing the size of the basic logic elements, from 4-input LUTs to 6 and 7-input LUTs, both static and dynamic power are reduced since more logic is implemented within each LUT and less routing is needed between the LUTs. This reduces power since LUTs are implemented using smaller transistors (compared to transistors in the routing resources), which leak less and dissipate less dynamic power. Both vendors have also modified their routing architectures to increase the number of neighbouring logic blocks that can be reached in only one or two hops (each routing segment used counts as one hop). Using more 1-hop routes reduces the average capacitance of the routes, which improves both power and performance. Other architecture-level features that reduce overall power are the embedded memories, adders, and multipliers. Although each of these functions can be implemented using the programmable logic fabric, implementing them as a fixed-function embedded block is more power-efficient since they do not have extra circuitry to make them flexible and can be turned off when they are not used.

Finally, a number of low-power techniques have also been incorporated into the commercial FPGA CAD tools. Detailed power models have been integrated within the Altera Quartus II [44] and Xilinx ISE CAD tools [45]. Both vendors provide a spreadsheet utility to make early power predictions before the design is complete and a detailed power model that can be used when the design is complete. Early power estimates are based on estimated resource usage, I/O types, clock requirements, clock frequencies, and environmental conditions. The detailed power models provide estimates after the application has been placed, routed, and simulated. The estimations from the detailed power models are more accurate than those from the early power models, since detailed capacitance, leakage, and switching activity information is known for each node in the application circuit. In the case where simulation results are not available, only basic probability-based (vectorless) activity estimation is available and the accuracy of the power estimates is significantly reduced. This is especially true for sequential circuits. Power-aware CAD techniques have also been incorporated into the commercial CAD flows. In Quartus II, power is minimized during technology mapping, placement, and routing by

minimizing the capacitance of high-activity signals using techniques similar to those described in the previous section. Power is also minimized by optimizing the mapping to the embedded memories, as described in [40], and, similarly, by optimizing the mapping to the embedded DSP blocks. In ISE, power is also minimized during placement and routing (but not technology mapping) by minimizing the capacitance of high-activity signals. Dynamic power is further minimized by strategically setting the configuration bits within partially used (some inputs are not used) LUTs to minimize switching activity. Both CAD tools also ensure that all unused logic blocks, embedded blocks, routing resources, and clock network resources are turned off to save power.

Combining all of these techniques, Altera reports that Stratix III FPGAs are over 50% more power efficient than Stratix II FPGAs [44]. Similarly, Xilinx reports that Virtex-5 FPGAs consume over 35% less dynamic power than Virtex-4 FPGAs, with even greater savings when embedded components are used [43]. Xilinx also points out that low-leakage techniques were already incorporated in their Virtex-4 FPGAs, resulting in 70% lower static power consumption compared to competing FPGAs.

1.3 Focus and Contributions of this Thesis

Although significant improvements have been made in recent years, a number of important FPGA power issues still remain. One important issue is the dynamic power that is wasted by unnecessary toggling in the routing resources of the FPGA. This occurs when LUT input values transition at different times (due to uneven delays) and produce spurious transitions at the output of the LUT. Another important issue is the design of efficient clock networks. FPGA clock networks dissipate a significant amount of power since they toggle every clock cycle and, like other parts of the FPGA, have a significant amount of parasitic capacitance since they need to be flexible enough to implement any application. Furthermore, the clock networks impose constraints on the CAD tools, which can reduce the efficiency of the implementation. In this thesis we address three different issues related to dynamic power in FPGAs: two concern estimating and minimizing the amount of toggling in the routing resources of the FPGA, and the other concerns FPGA clock networks. In the following subsections, we outline the three contributions.

1.3.1 Switching Activity Estimation for FPGAs

Although leakage is becoming more important in new processes, dynamic power is still the main source of power dissipation in current FPGAs; a study that examined power dissipation in a commercial 90nm FPGA found that dynamic power accounts for 62% of total power [9]. The dynamic power dissipated in an FPGA is directly proportional to *switching activity*, which is the average number of transitions of all nodes in the circuit per unit time. The first contribution of this thesis is *an improved technique for estimating the switching activity of all nodes within a circuit*. In addition, we investigate the *impact that the accuracy of the switching activity estimation has on the ability to obtain accurate power estimates from the FPGA power models and power savings from the power-aware FPGA CAD tools*. In this subsection, we describe the motivation for the work, outline the technical achievements, and discuss the work's significance. Details of this contribution are given in Chapter 2 of this thesis.

As described in the previous section, power reduction in FPGAs requires both power-aware CAD tools and accurate power estimation models. An accurate method of estimating switching activity is a key part of this. More specifically, switching activity estimation is important for two reasons:

1. Detailed FPGA power models require accurate switching activity information to produce accurate and useful power estimates. As an example, consider an application designer who is deciding how many pipelining stages should be used in his or her design. Increasing the number of pipelining stages increases the number of flip-flops in the design, but reduces the average switching activity, because flip-flops filter out glitches. This tradeoff makes it unclear which implementation is more efficient. Using a detailed FPGA power model with accurate switching activity information, the designer can simply compare each implementation and choose the one that dissipates the least amount of power.
2. Power-aware FPGA CAD tools require accurate switching activity information to minimize power. In general, the power-aware tools minimize the capacitance of high-activity connections. Therefore, the power savings depend on the accuracy of the switching activities. As an example, consider a power-aware router that reduces power by routing high-activity nets as directly as possible to minimize capacitance. If the activities are not accurate, the router may optimize the wrong wires and produce an inefficient implementation.

The most straight-forward gate-level activity estimation technique is to perform a circuit simulation while counting the number of transitions that occur on each node. This approach is very accurate, but it has two limitations. First, simulation is slow since a large number of input vectors need to be simulated in order to obtain meaningful results. Second, the activity results are directly related to the input vector patterns that are used to drive the simulation. This pattern dependency is a serious problem since realistic input vectors are rarely available until after the application has been designed; power estimates for alternative functional blocks are often required when the rest of the application has not yet been specified and little is known about the nature of the input.

Another gate-level activity estimation technique is to calculate the activity of each gate based on the input activities and Boolean function of the gate. This *probabilistic* (or *vectorless*) approach is faster than simulation and does not require input vectors (only input activities), but is less accurate than simulation. The activities are less accurate since they generally do not take timing information and signal correlation into account. A number of techniques that consider these issues have been proposed; however, there is always a tradeoff between speed and accuracy. Therefore, the challenge is to determine how much accuracy is required and use the fastest techniques that achieve this.

Current commercial FPGA power models rely mostly on simulated activities, and provide only basic probabilistic activity estimation when simulation results are not available [44],[45]. Similarly, academic FPGA power models rely either on simulated activities, as in [11], or basic probabilistic activity estimation, as in [10]. The ACE activity estimator, from [10], uses the Transition Density model, from [46], for estimating the activities; the analytical glitch filter, from [47], for modeling physical RC filtering that occurs in the circuitry; and a simple iteration technique to determine the activities at the output of flip-flops. Although this approach is accurate for small purely combinational designs, the accuracy quickly degrades for large or sequential designs since it does not adequately model the effects of the circuit delays and signal correlation caused by sequential feedback. Consequentially, this inaccuracy introduces significant error into the power estimates and reduces the power savings of the FPGA CAD tools.

In Chapter 2 of this thesis, we investigate probabilistic activity estimation techniques in the context of FPGAs. We make two key contributions. First, we determine how accurate the gate-

level techniques must be to obtain accurate power estimates from the FPGA power models and high power savings from the power-aware FPGA CAD tools. We find that existing techniques are sufficiently accurate for combinational circuits, but for sequential circuits (circuits containing flip-flops), the existing estimation techniques do not provide enough accuracy to guide power-aware CAD tools to a low-power solution.

Second, we present a new activity estimation tool that combines a number of new and existing techniques. Our tool is different than existing techniques in two ways:

1. To capture spatial correlation (correlation between the activities of different nodes with the circuit), existing techniques "collapse" logic circuits before calculating the activities of nodes. This can lead to long run times, since the collapsed logic representation can be large. We employ a novel "pruning" technique in which we reduce the size of the collapsed logic representation at only a very small (and bounded) cost in accuracy.
2. To capture temporal correlation (correlation between the activities of the same node in different cycles), existing techniques either unroll the sequential circuit, requiring increased run-time and storage, or use approximate equations that take temporal correlation into account. We simplify this by identifying sequential feedback loops, and using logic simulation to estimate the activities of nodes within these loops.

We encapsulated our techniques into a new tool called ACE2.0, and have made this tool publicly available to the research community. Our new algorithm results in power estimates and power savings that are both within 1% of results obtained using simulated activities. Moreover, the new tool is 69 times faster than simulation for combinational circuits and 7.2 times faster than simulation for sequential circuits.

These results are significant. One of the key limitations of current academic and industrial power estimation tools for FPGAs is activity estimation. Our work directly addresses this. The algorithm is fast and accurate enough that, if extended to consider embedded blocks, it could immediately be integrated into commercial FPGA power estimation tools. As described earlier, this would lead to better power estimates, and would help power-aware CAD tools find low power solutions. This, in turn, may open the use of FPGAs to new applications, and help FPGA customers create more power-efficient designs.

1.3.2 Switching Activity Minimization in FPGAs

The second contribution of this thesis is the proposal and evaluation of an architectural enhancement that can be used to minimize the glitching activity within an FPGA. In Section 1.2.1, we discussed techniques to reduce power by minimizing the distance between logic elements connected by high-activity wires. Here, we explain how to reduce power by actually minimizing the switching activity of the wires. In this subsection, we introduce the concept of glitching activity, and briefly describe our architectural enhancement. Chapter 3 of the thesis contains details of this contribution.

The switching activity of a node has two components. The first component consists of functional transitions, which are transitions that need to occur to perform the computation. The second component consists of glitches, which are transitions that occur unnecessarily when the values at the inputs of a logic element arrive at different times. As an example, consider an exclusive-or gate with two inputs. If the two inputs both change from 1 to 0, ideally the output would remain at 0. However, if the two inputs change at slightly different times, the output may change to 1 for a short time. This actually causes two unnecessary transitions: one transition from 0 to 1, and one from 1 to 0. These glitches can dissipate a significant amount of power. In Chapter 3 of this thesis, we show that glitching accounts for 31% of the switching activity and 23% of the total power in FPGAs.

To reduce the number of glitches that are produced at the output of a gate, we propose an enhancement to the logic block architecture of an FPGA. Specifically, we propose adding configurable delay elements to each (or some of) the inputs to each lookup-table. After placement and routing, detailed information about the arrival times of signals to each lookup-table is known. Using these arrival times, it is possible to determine whether a glitch may occur at the output. Each delay element can then be programmed to eliminate as many of these glitches as possible.

The fact that our delay elements are configurable is key. Delay insertion has been used previously to minimize glitching in ASICs [48]. In ASICs, the problem is much simpler because the arrival times can be estimated before fabrication. Thus, in an ASIC, the delay elements can provide fixed delays. In an FPGA, because we do not know the arrival times before fabrication, it must be possible to configure each delay element to take on an arbitrary delay after fabrication.

The ability of our architecture to eliminate glitches depends on several factors. First, the resolution of the delay element as well as the maximum delay that can be added to each input is important. In Chapter 3, we vary these parameters, and determine the impact on the overhead and the ability to reduce glitching activity. In addition, the position of these delay elements within the logic block is important. We present and evaluate several schemes for positioning these delay elements.

On average, the proposed technique eliminates 87% of the glitching, which reduces overall FPGA power by 17%. The added circuitry increases the overall FPGA area by 6% and critical-path delay by less than 1%.

A 17% reduction in power is significant. Moreover, our proposal can be applied to all commercial FPGAs, and requires no changes to the CAD flow or the rest of the architecture. The gains are roughly independent of those that can be obtained using process enhancement techniques. However, there may be some overlap in these gains with those that can be obtained using a power-aware CAD flow, since by reducing the activity of high-activity signals, there may be less "low-hanging fruit" available for the power-aware CAD flow.

1.3.3 FPGA Clock Networks

The third contribution of this thesis is an architecture study for flexible clock networks on an FPGA, along with an experimental comparison of various placement algorithms that obey constraints imposed by an FPGA clock network.

Over the last 20 years, significant advancements have been made and FPGAs are now being used to implement large commercial applications for digital signal processing, communication, medical imaging, and scientific computing. These large systems often require more than one clock domain. As an example, consider a communications application connected to several I/O ports. Each port might have its own clock, meaning the circuitry connected to each port must be controlled by a separate clock. FPGA vendors support such applications through the use of complex clock networks that are flexible enough to support a wide range of applications, yet have low skew and are power-efficient.

Power efficiency is one of the critical concerns in clock network design. The clock network has a significant impact on power since it connects to each flip-flop on the FPGA in a flexible way and toggles every clock cycle. As illustrated in Figure 1.2, the clock network in a current 90 nm FPGA accounts for 19% of dynamic power dissipation. Moreover, depending on

how flexible it is, the clock network can impose constraints on the placement of logic within the FPGA; these constraints may, in turn, affect power dissipation in other parts of the FPGA. In current FPGAs, the FPGA is divided into regions in which a limited number of different user clocks can be supported (by the underlying clock network). For applications with many clock domains, these constraints can affect how efficiently applications can be implemented.

In Chapter 4 of this thesis, we examine the tradeoff between the flexibility of FPGA clock networks and overall power consumption. Three important contributions are made. First, we present a parameterized framework to describe FPGA clock networks. The framework allows us to describe a wide range of different clock networks and includes key features of the latest commercial FPGA clock networks. The challenge in creating such a framework is to make it flexible enough to describe as many different "reasonable" clock network architectures as possible, and yet be as concise as possible. In our model, we describe a clock network using seven parameters.

Second, we present and compare a number of new clock-aware placement techniques, to determine which is most effective. The clock network imposes hard constraints on the placement of modules within the FPGA. The new placement algorithm obeys these constraints, while minimizing power and maximizing speed and routability. One of the challenges is that the hard constraints imposed by the clock network depends on how the clock network is used. Often, user clocks can be mapped to global or local clock distribution resources. Thus, the placement tool must simultaneously assign the user clocks to the physical clock resources and assign the circuit blocks to physical locations on the FPGA. In Chapter 4, we consider several techniques for combining these seemingly disparate algorithms, and evaluate each in terms of its ability to find a legal solution and the quality of the solution it finds. This is an important problem; we show that, on average, circuits placed using the most effective techniques dissipate 15% less energy and are 3% faster than circuits placed using the least effective techniques. Despite that, current academic research into FPGA placement algorithms rarely considers the clock network during optimization; our results suggest that this is a major limitation.

Finally, we use our framework and the clock-aware placement algorithm to experimentally investigate the architecture of the clock network itself. All clock networks we consider have flexibility; however, the amount of flexibility varies, as does the manner in which this flexibility is provided. The results of these experiments provide insight into what makes a good FPGA

clock distribution network. Again, our results show that the clock network optimization is important; on average, FPGAs with an efficient clock network are up to 9% more energy efficient and 5% faster than other FPGAs.

We have implemented our placement algorithm into the popular VPR CAD tool [2], and have integrated VPR's architecture description format to include our clock network parameters. This has been made publicly available. The significance of this work is thus two-fold: (1) we provide information that will help FPGA vendors provide more efficient clock networks, and (2) we provide a new framework for architectural exploration that will help guide future researchers.

1.4 Thesis Organization

This thesis follows the *manuscript-based format* for PhD theses at the University of British Columbia. In this format, each chapter is a complete, unaltered, published or submitted paper, complete with introduction, conclusion, and references. The three chapters that make up this thesis are submitted journal papers. Preliminary versions of the material in each submitted journal paper has been published in conference papers, as referenced below.

The first contribution, Activity Estimation, is in Chapter 2. This chapter compares various switching activity estimation techniques to determine which are most suitable for FPGA models and CAD tools and then describes a new tool which combines the most suitable techniques. Part of this work was previously published in [49] and the complete work has been submitted for review in [50]. The second contribution is in Chapter 3. This chapter describes a new switching activity minimization technique for FPGAs which involve adding programmable delay elements within FPGAs to align arrival times so as to eliminate glitching. Part of this work has been published in [51] and the complete work has been submitted for review in [52]. The third contribution is in Chapter 4. This work examines the tradeoff between power and flexibility of FPGA clock networks. Parts of this work have been published in [53] and [54], and the complete work has been submitted for review in [55]. Finally, Chapter 5 summarizes the thesis contributions and conclusions and provides suggestions for future work.

Chapter 1 References

- [1] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays, pp. 21-30, 2006.
- [2] V. Betz., J. Rose, and A. Marquardt, "Architecture and CAD for deep-submicron FPGAs," Kluwer Academic Publishers, 1999.
- [3] S.J.E. Wilton, J. Rose, and Z.G. Vranesic, "Architecture of centralized field-configurable memory," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 97-103, 1995.
- [4] S. Hong and S.-S. Chin, "Reconfigurable embedded MAC core design for low-power coarse-grain FPGA," IET Electronics Letters, Vol. 39, Issue 7 (April), pp. 606-608, 2003.
- [5] G. Lemieux and D. Lewis, "Design of interconnection networks for programmable logic," Springer (formerly Kluwer Academic Publishers), 2004.
- [6] S. J. E. Wilton, "SMAP: heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 171-178, 1998.
- [7] J. Cong and S. Xu, Technology mapping for FPGAs with embedded memory blocks, in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 179-188, 1998.
- [8] International Technology Roadmap for Semiconductors, "International technology roadmap for semiconductors 2005," 2005.
- [9] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90nm low-power FPGA for battery-powered applications", Proc. ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 3-11, 2006.
- [10] K.K.W. Poon, S.J.E. Wilton, A. Yan, "A detailed power model for field-programmable gate arrays," ACM Trans. on Design Automation of Electronic Systems (TODAES), Vol. 10, No. 2, pp. 279-302, April 2005.
- [11] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 175-184, 2003.
- [12] F. Li, Y. Lin, and L. He, "FPGA power reduction using configurable dual-Vdd," Design Automation Conference (DAC), pp. 735-740, 2004.
- [13] F. Li, Y. Lin, L. He, and J. Cong, "Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 42-50, 2004.
- [14] T. Jiang, X. Tang, and P. Banerjee, "Macro-models for high level area and power estimation on FPGAs," ACM Great Lakes Symp. on VLSI GLSVLSI, pp. 162-165, 2004.
- [15] A. Reimer, A. Schulz, and W. Nebel, "Modeling macromodules for high-level dynamic power estimation of FPGA-based digital designs," Intl. Symp. on Low Power Electronics and Design (ISLPED), pp. 151-154, 2006.
- [16] D. Chen, J. Cong, Y. Fan, and Z. Zhang, "High-level power estimation and low-power design space exploration for FPGAs," Asia South Pacific Design Automation Conference, pp. 529-534, 2007.

- [17] V. George, H. Zhang, and J. Rabaey, "The design of a low energy FPGA," Intl. Symp. on Low Power Electronics and Design, pp. 188-193, 1999.
- [18] M. Meijer, R. Krishnan, and M. Bennebroek, "Energy efficient FPGA interconnect design," Design and Test in Europe (DATE), pp. 1-6, 2006.
- [19] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozargzadeh, "HARP: hard-wired routing pattern FPGAs," Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 21-29, 2005.
- [20] J.H. Anderson and F.N. Najm, "A novel low-power FPGA routing switch," IEEE Custom Integrated Circuits Conference (CICC), pp. 719-722, 2004.
- [21] Y. Lin, F. Li, and L. He, "Routing track duplication with fine-grained power-gating for FPGA interconnect power reduction," Asia South Pacific Design Automation Conference, pp. 645-650, 2005.
- [22] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," Intl. Symp. Low Power Electronics and Design, pp. 155-160, 1999.
- [23] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architecture," Low Power Electronics and Design, pp. 134-139, 2003.
- [24] D. Chen, J. Cong, and J. Xu, "Optimal module and voltage assignment for low-power," Asia South Pacific Design Automation Conference, pp. 850-855, 2005.
- [25] A.H. Farrahi and M. Sarrafzadeh, "FPGA technology mapping for power minimization," Intl. Workshop on Field-Programmable Logic and Applications (FPL), pp. 167-174, 1994.
- [26] M.J. Alexander, "Power optimization for FPGA look-up tables", ACM Intl. Symp. on Physical Design (ISPD), pp. 156-162, 1997.
- [27] C-C. Wang and C-P Kwan, "Low power technology mapping by hiding high-transition paths in invisible edges for LUT-based FPGAs", IEEE Intl. Symp. on Circuits and Systems, pp. 1536-1539, 1997.
- [28] Z-H. Wang, E-C. Liu, J. Lai, and T-C. Wang, "Power minimization in LUT-based FPGA technology mapping," ACM Asia South Pacific Design Automation Conference, pp. 635-640, 2001.
- [29] J. Anderson and F.N. Najm, "Power-aware technology mapping for LUT-based FPGAs," IEEE Intl. Conference on Field-Programmable Technology (FPT), pp. 211-218, 2002.
- [30] H. Li, S. Katkoori, and W.-K. Mak, "Power minimization algorithms for LUT-based FPGA technology mapping", ACM Trans. on Design Automation of Electronic Systems (TODAES), Vol. 9, Issue 1, pp. 33-51, 2004.
- [31] D. Chen, J. Cong, F. Li, and L. He, "Low-power technology mapping for FPGA architectures with dual supply voltages," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 109-117, 2004.
- [32] A. Singh, G. Parthasarathy, and M. Marek-Sadowski, "Efficient circuit clustering for area and power reduction in FPGAs," ACM Trans. on Design Automation of Electronic Systems (TODAES), Vol. 7, Issue 4, pp. 643-663, 2002.
- [33] D. Chen and J. Cong, "Delay optimal low-power circuit clustering for FPGAs with dual supply voltages," Intl. Symp. on Low Power Electronics and Design (ISLPED), pp. 70-73, 2004.

- [34] H. Hassan, M. Anis, A. El Daher, and M. Elmasry, "Activity packing in FPGAs for leakage power reduction," Design Automation and Test in Europe (DATE), pp. 212-217, 2005.
- [35] K. Roy, "Power-dissipation driven FPGA place and route under timing constraints", IEEE Trans. on Circuits and Systems, Vol. 46, Issue. 5, pp. 634-637, 1999.
- [36] N. Togawa, K. Ukai, M. Yanagisawa, and T. Ohtsuki, "A simultaneous placement and global routing algorithm for FPGAs with power optimization", Asia Pacific Conference on Circuits and Systems, pp. 125-128, 1998.
- [37] B. Kumthekar, and F. Somenzi, "Power and delay reduction via simultaneous logic and placement optimization in FPGAs," Design Automation and Test in Europe Conference (DATE), pp. 202-207, 2000.
- [38] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays, pp. 51-58, 2004.
- [39] J.H. Anderson, F.N. Najm, and T. Tuan, "Active leakage power optimization for FPGAs," ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays, pp. 33-41, 2004.
- [40] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware RAM mapping for FPGA embedded memory blocks," Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 189-198, 2006.
- [41] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware CAD Algorithms for FPGAs," IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 701-708, 2003.
- [42] Altera Corp., "Quartus II handbook," Vol. 3, Chapter 10, May 2007.
- [43] Xilinx Inc., "Power Consumption in 65nm FPGAs", February, 2007.
- [44] Altera Corp., "Quartus II handbook," Vol. 2, Chapter 9, May 2007.
- [45] Xilinx Inc., "Optimizing FPGA power with ISE design tools," Xcell Journal, Issue 60, pp. 16-19, 2007.
- [46] F. Najm, Transition density: A new measure of activity in digital circuits, IEEE Trans. Computer-Aided Design, Vol. 12, Issue 2, pp. 310-323, 1993.
- [47] F. Najm, Low-pass filter for computing the transition density in digital circuits, IEEE Trans. Computer-Aided Design, Vol. 13, Issue 9, pp. 1123-1131, 1994.
- [48] A. Raghunathan, S. Dey and N. K. Jia, "Register transfer level power optimization with emphasis on glitch analysis and reduction", IEEE Trans. CAD, Vol. 18, No. 8, pp. 1114-1131, 1999.
- [49] J. Lamoureux and S.J.E. Wilton, "Activity Estimation for Field-Programmable Gate Arrays," in the International Conference on Field-Programmable Logic and Applications (FPL), pp. 87-94, 2006.
- [50] J. Lamoureux and S.J.E. Wilton, "Fast Activity Estimation for Power Modeling and Optimization in FPGAs," submitted for review, ACM Trans. on Design Automation of Electronic Systems (TODAES), in 2007.
- [51] J. Lamoureux, G. Lemieux, and S.J.E. Wilton, "Glitchless: An Active Glitch Minimization Technique for FPGAs," in the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 156-165, 2007.

- [52] J. Lamoureux, G. Lemieux, and S.J.E. Wilton, "GlitchLess: Dynamic Power Minimization in FPGAs Through Edge Alignment and Glitch Filtering," submitted for review, IEEE Trans. on VLSI, 2007.
- [53] J. Lamoureux and S.J.E. Wilton, "FPGA Clock Network Architecture: Flexibility vs. Area and Power," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 101-108, 2006.
- [54] J. Lamoureux and S.J.E. Wilton, "Clock-aware placement for FPGAs," to appear in the Intl. Workshop on Field-Programmable Logic and Applications (FPL), 2007.
- [55] J. Lamoureux and S.J.E. Wilton, "On the Tradeoff between Power and Flexibility of FPGA Clock Networks," submitted for review, ACM Trans. on Reconfigurable Technology and Systems, 2007.

Chapter 2

Fast Activity Estimation for Power Modeling and Optimization in FPGAs¹

2.1 Introduction

Advancements in process technology, programmable architecture, and computer-aided design (CAD) tools are allowing circuit designers to implement increasingly larger applications on FPGAs. The main advantages of using FPGAs, instead of Application Specific Integrated Circuits (ASICs), are a faster time-to-market and the ability to quickly prototype and modify the application since layout and fabrication are not required. On the other hand, FPGAs are slower and dissipate significantly more power than ASICs because of the overhead circuitry that is needed to make FPGAs programmable. Therefore, the key challenge in FPGAs is to minimize this speed and power overhead while keeping the design effort low.

The CAD tools, which are used to configure the programmable logic and routing switches of the FPGA, can have a significant impact on speed and power of an application. As an example, the power-aware FPGA CAD flow described in [1], which minimizes the length (and therefore capacitance) of high-activity wires, reduces the power-delay product of applications implemented on an FPGA by 26% on average.

An important part of the CAD tools is power modeling. With an accurate power model, a designer can determine if their implementation meets specification or if a different implementation is more efficient. Moreover, the power-aware CAD flow described in the previous example requires accurate switching activity information to choose which wires should be optimized to reduce power. This chapter focuses on the how to obtain switching activities in the context of FPGAs.

Numerous activity estimation techniques have been proposed in the literature [2]-[11]. These techniques vary widely in terms of speed and accuracy. For FPGAs, the techniques need to be accurate enough for the power-aware CAD tools to minimize power and for the power

¹ A version of this chapter has been submitted for publication. J. Lamoureux and S.J.E Wilton, Fast activity estimation for power modeling and optimization in FPGAs, ACM Transactions on Design Automation of Electronic Systems, 2007.

models to produce power estimates that can be used to determine if an implementation meets specifications. At the same time, the techniques need to be fast enough so that it does not burden the design cycle. In other words, the time it takes to estimate the switching activities should not take longer than the other phases of the FPGA CAD flow. In this chapter, we compare existing activity estimation techniques to determine which are most suitable for FPGAs. Specifically, we answer the following questions:

1. How does the accuracy of the switching activity estimates affect the accuracy of the final power estimates in FPGAs? If the accuracy of the power estimate is not overly sensitive to the accuracy of the activity estimates, then a faster technique may be sufficient.
2. How does the accuracy of the switching activity estimates affect the ability of power-aware CAD tools to minimize power in FPGAs? Knowing how sensitive the power-aware CAD tools are to the accuracy of the activities will help to determine what techniques are most appropriate.
3. Should different activity estimation techniques be used for different applications? Estimating activities is more difficult when circuits have many state storage elements (flip-flops). Perhaps accurate techniques should be used for circuits with many flip-flops and faster techniques can be used otherwise.

Once these questions have been answered, we present a new activity estimation tool called ACE-2.0, which was developed based on the answers to the three questions above. The tool is compatible with the Versatile Place and Route [12] and the power model from [13], and is publicly available. A preliminary version of this work has appeared in [14]; however, this chapter expands on the results to better illustrate which techniques are most appropriate for FPGAs.

This chapter is organized as follows. Section 2.2 summarizes existing activity estimation techniques and describes an existing power model and power-aware CAD flow for FPGAs. Section 2.3 then examines the accuracy of the power model and the performance of the power-aware CAD flow when various activity estimation techniques are employed. In doing so, it answers the three questions listed above. Section 2.4 first describes a new activity estimation tool called ACE-2.0, which incorporates the techniques found suitable in Section 2.3, and then compares the new activity estimation tool to an existing activity estimation tool called ACE-1.0.

Finally, Section 2.5 summarizes our conclusions and proposes future work. Instructions for downloading our new tool are provided in the appendix.

2.2 Background

This section begins by introducing some terminology that is used in this chapter to describe switching activity information. It then provides background on the different techniques that can be used to obtain switching activity information. Finally, it describes a number of existing activity estimation tools.

2.2.1 Terminology

Switching activity information is required when estimating static and dynamic power dissipated by integrated circuits. Three statistical values are commonly used to describe the switching behavior of circuit wires: static probability, switching probability, and switching activity. The static probability of a wire x , denoted $P_I(n)$, is the probability that the value of that wire is logic high. This value is used in static power calculations to determine how long transistors are in a particular leakage state. The switching probability of a wire n , denoted $P_s(n)$, is the probability that the steady-state value of that wire will transition from high(low) to low(high) from one clock cycle to the next. Finally, the switching activity of a wire n , denoted $A_s(n)$, refers to the average number of times that the wire will transition from high to low or from low to high during a clock period. Although its steady-state value can only change once per cycle, a wire can actually transition multiple times before reaching steady-state since the arrival times at the inputs of a logic gate may be different, $P_s(n) \leq A_s(n)$. These spurious transitions are called glitches. Switching activities are used in dynamic power calculations.

2.2.2 Activity Estimation

There are several techniques that can be used to estimate switching activities. These techniques, which vary significantly in terms of accuracy and speed, can be categorized into two general groups: simulation-based and probability-based.

Simulation-based Activity Estimation

Simulation-based activity estimation can be employed at various levels of abstraction, which include: switch-level, gate-level, and block-level. In this chapter, we only consider gate-level simulation since it provides switching activity estimates for every wire in the netlist of the

application, which are needed by the power-aware FPGA CAD algorithms and detailed FPGA power models. Block-level simulation, which considers larger blocks such as adders, multipliers, memories, and state machines, is not considered since they do not provide switching activity estimates for each wire in the netlist. On the other hand, switch-level simulation, which simulates transitions at the transistor level, is detailed enough but is significantly slower than gate-level simulation.

Gate-level simulation-based activity estimation involves simulating a Boolean network, consisting of logic gates and flip-flops, while keeping track of transitions in order to determine P_I , P_s , and A_s for each node in the network. During simulation, the values at the output of the gates are determined from the values at the input of those gates each time an input changes. Gate-level simulation is a well studied problem and much effort has been placed on improving its speed [3],[10],[11]. Despite many innovations, gate-level simulation for large system-level applications can still take as long as days. Moreover, simulation requires input vectors, which are often not available when designing a new application.

Probability-based Activity Estimation

Probability-based (or vectorless) activity estimation is typically faster than simulation because it involves a one-time calculation for each node of a circuit. The most straightforward way of calculating $P_I(n)$, $P_s(n)$ and $A_s(n)$ is to visit the nodes one at a time, starting from the primary inputs. For each node, the Lag-one model from [8] and the Transition Density model from [2] can be used to calculate $P_I(n)$, $P_s(n)$ and $A_s(n)$, based on $P_I(f)$, $P_s(f)$ and $A_s(f)$ of each fan-in node f of node n , and the function implemented by node n . These quantities can then be used to estimate the power using standard power models [13].

Probability-based estimates are less accurate than simulation-based estimates for two reasons. First, they typically ignore wire and gate delays, which are the cause of glitching activity. Techniques to take these delays into account have been described in [4]; however, these techniques are not considered in this chapter since delay information may not be available at design time. Second, they typically ignore both spatial and temporal correlation between signals. Spatial correlation occurs when the logic value of a wire depends on the value of another wire. Spatial correlation can occur between primary inputs when the input data has correlation or between internal nodes when gates fan-out to multiple gates and later reconverge, as illustrated in Figure 2.1 (a). Temporal correlation occurs when the value of a wire depends on previous

values of that same wire. This can also occur at the primary inputs or within sequential circuits which have feedback, as illustrated in Figure 2.1 (b). From [15], ignoring temporal correlation introduces between 15% and 50% error, and ignoring spatial correlation introduces between 8% and 120% error.

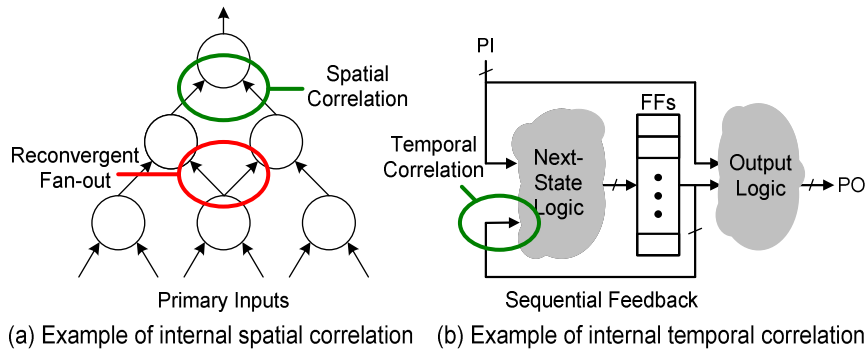


Figure 2.1: (a) Spatial correlation (b) Temporal correlation.

Probability-based techniques that consider spatial correlation have been described in previous works [4],[8]. Instead of estimating the activity of gates locally (as standard probability-based techniques do), the error introduced by reconvergent fan-out can be eliminated by recursively collapsing nodes with its predecessors (combine the Boolean logic of the current gate with the Boolean logic of the fanin gates) until the inputs of the node consist only of primary inputs. The switching activity at the output of the node is then calculated as before (using some probability-based calculation). Typically, a binary decision diagram (BDD) is used to represent this Boolean logic, since this representation is efficient for large nodes [16].

Techniques that consider temporal correlation have also been described in previous works. In [6]-[8], internal temporal correlation is captured by “unrolling” next state logic (see Figure 2.2) and iteratively estimating switching activities until the activities converge. By unrolling the next logic ∞ times, all temporal correlation can be captured. Although unrolling ∞ times is infeasible, [8] found that 3 times produced good results.

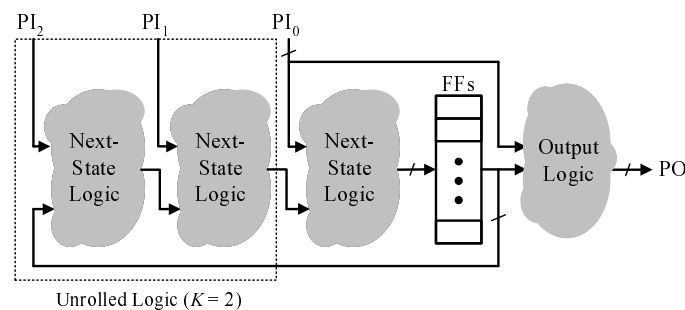


Figure 2.2: Example of unrolling next state logic.

2.2.3 Existing Activity Estimation Tools

A number of existing activity estimation tools that employ the probability-based and simulation-based techniques described above are publicly available [7],[11],[13]. The activity estimation tool for FPGAs, described in [11], is a statistical-based activity estimator, which performs gate-level simulation of randomly generated input vectors until a user-defined degree of certainty can be statistically guaranteed.

The ACE-1.0 tool, described in [13], is a probability-based activity estimator which estimates P_1 , P_s , and A_s for combinational and sequential circuits. The static and switching probabilities are calculated using the techniques in [9], and switching activities are calculated using the Transition Density model [2] and the analytical low-pass filter model described in [5]. For circuits with sequential feedback, an iterative technique is used to update the switching probabilities at the output of flip-flops using the simplifying expressions $P_1(Q) = P_1(D)$ and $P_s(Q) = 2*P_1(D)*(1-P_1(D))$ as described in [9]. ACE is fast but inaccurate for large and/or sequential circuits.

The Sis [7] activity estimator (called Sis-1.2 in this chapter) estimates P_1 and P_s , but does not estimate A_s . For circuits with sequential feedback, static and switching probabilities are calculated using the iterative solution described in [8]. Sis-1.2 is only accurate for circuits that do not have a large glitching component. Furthermore, the estimator can become very slow for circuits with large BDD representations such as multipliers [16].

2.3 Empirical Study on Activity Estimation for FPGAs

This section examines the questions described in the introduction regarding the effect of switching activities on power models and power-aware CAD tools for FPGAs. This is important: the activity estimation techniques described in Section 2.2 are general since they do not target any specific implementation; without using the activities in the context of FPGAs, it is impossible to determine with confidence which techniques are most suitable.

2.3.1 Accuracy and Speed of Probabilistic-Based Activity Estimation

Before examining how the accuracy of the activities effect the accuracy of the power model and the ability of the power-aware CAD tools to minimize power, we begin by comparing the accuracy and speed of the existing activity estimation tools to highlight the strengths and weaknesses of the techniques they employ. Specifically, we compare the ACE-1.0 and Sis-1.2

tools. This comparison provides insight when we examine the effect that the activities have on the power model and CAD tools later on.

Experimental Framework

Figure 2.3 illustrates how the activities are obtained for each activity estimator. Twenty large MCNC [17] benchmarks circuits and one ISCAS [18] multiplier circuit (C6288) are used to empirically determine the speed and accuracy of the each activity estimation method. The multiplier is included since it is known to be a challenging circuit for activity estimation. For each circuit, static probabilities and switching probabilities are pseudo-randomly determined for each primary input of every circuit. A custom vector generator routine is then used to generate pseudo-random test vectors that match these static and switching probabilities. The pseudo-random static and switching probabilities are used to drive ACE-1.0 and Sis-1.2, and the corresponding vectors are used to drive the Verilog XL® gate-level simulator. Finally, circuit delay information that is needed for the simulation is obtained using the Versatile Place and Route (VPR) tool for FPGAs [12]. Each experiment is executed on a 1.7GHz Intel® Celeron® processor, with 500 MB of memory.

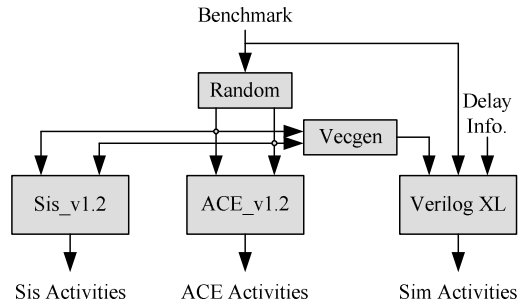


Figure 2.3: Activity comparison framework.

Accuracy Metrics

Three metrics are used to examine accuracy. The first is *average relative error*, which is zero for perfectly accurate estimates. For each circuit, this measurement takes the average of the magnitude of the relative error of every node within that circuit, as described in the following expression:

$$average\ relative\ error(circuit) = \frac{1}{N} \cdot \sum_{n \in circuit} \left| \frac{A_{s_{est}}(n) - A_{s_{sim}}(n)}{A_{s_{sim}}(n)} \right| \quad (2.1)$$

where N is the number of nodes in the circuit. The second measurement is the *activity ratio*, which divides the sum of the estimated activities by the sum of the simulated activities, as described by the following expression:

$$\text{activity ratio}(\text{circuit}) = \frac{\sum_{n \in \text{circuit}} A_{s_{est}}(n)}{\sum_{n \in \text{circuit}} A_{s_{sim}}(n)} \quad (2.2)$$

where an activity ratio of one is ideal. Finally, the third accuracy measurement is r^2 correlation, which measures the quality of a least squares fitting to the exact data, as described by the expression below [21]. Intuitively, an r^2 value of 1 indicates that the estimated results match the simulated results perfectly and an r^2 value of 0 indicates that the estimated results are completely unrelated to the simulated results.

$$r^2 = \frac{\left(\sum_{n \in \text{circuit}} A_{s_{sim}}(n) A_{s_{est}}(n) - N \cdot \overline{A_{s_{sim}}} \cdot \overline{A_{s_{est}}} \right)^2}{\left(\sum_{n \in \text{circuit}} A_{s_{sim}}(n)^2 - N \cdot \overline{A_{s_{sim}}}^2 \right) \left(\sum_{n \in \text{circuit}} A_{s_{est}}(n)^2 - N \cdot \overline{A_{s_{est}}}^2 \right)} \quad (2.3)$$

Although the metrics 2.2 and 2.3 give a better measure of accuracy, each measurement has a purpose. The *average relative error* gives an intuitive feel for how much error is occurring for individual nodes; however, the measurement does not give insight to the nature of the error. As an example, the *average relative error* is the same if an estimator tends to overestimate or underestimate activities by a factor of two. The *activity ratio*, on the other hand, does indicate if the estimator tends to over or under estimate activities but is not a good measure of accuracy since the error of one activity can cancel out the error of another activity. Finally, r^2 is useful for measuring the fidelity of the activities. In other words, it measures how well the estimator can determine if one node has a higher activity than another node. Intuitively, fidelity is important for power-aware CAD tools, which need to determine which wires to optimize.

Results

Table 2.1 compares the accuracy and run-time of ACE-1.0 and Sis-1.2. Many observations can be made from the results. First, the table shows that some results are missing for Sis-1.2 due to insufficient memory or extremely long execution time. This problem occurred in all but the two smallest sequential circuits (dsip and bigkey) and in one of the combinational circuits (C6288),

which is a multiplier. The likely origin of the problem is the use of BDDs, which tend to grow exponentially in size with the number of logic inputs [19] and are inefficient at representing a number of common circuits, including multipliers [16]. ACE-1.0 was able to produce results for all the circuits; however, the accuracy was low for sequential circuits with a correlation of 0.47 and activity ratio of 2.37.

Table 2.1: Accuracy of speed of switching activity estimation techniques.

Benchmark Circuits	# Gates	# Flip-Flops	Simulation	Sis-1.2				ACE-1.0			
			Run-Time (s)	r ²	Avg. Rel. Err.	Act. Ratio	Speed up	r ²	Avg. Rel. Err.	Act. Ratio	Speed up
alu4	2732	0	89	0.86	0.03	1.34	127	0.89	0.03	1.33	9.3
apex2	3165	0	68	0.86	0.02	1.53	57	0.86	0.01	1.22	5.2
apex4	2195	0	34	0.94	0.01	1.20	57	0.97	0.01	1.24	4.7
C6288	1820	0	315	-	-	-	-	0.58	0.60	0.47	35
des	2901	0	206	0.65	0.06	1.00	172	0.84	0.05	1.16	16
ex1010	8020	0	144	0.94	0.00	0.93	60	0.91	0.01	1.70	1.5
ex5p	1779	0	44	0.94	0.02	1.12	110	0.97	0.02	1.20	8.5
misex3	2557	0	56	0.91	0.02	1.38	80	0.94	0.01	1.21	6.2
pdc	8408	0	156	0.88	0.01	1.16	60	0.95	0.01	1.16	1.6
seq	2939	0	79	0.93	0.02	1.13	99	0.94	0.02	1.31	6.6
spla	7438	0	154	0.91	0.01	0.94	67	0.98	0.01	1.28	2.1
Comb. Avg.	3996	0	122	0.88	0.02	1.17	89	0.89	0.07	1.21	8.8
bigkey	2979	224	150	0.64	0.10	1.23	18	0.17	0.16	1.74	10
clma	14235	33	254	-	-	-	-	0.11	0.15	6.92	0.7
diffeq	2544	395	56	-	-	-	-	0.66	0.01	0.72	3.7
dsip	2531	224	141	0.80	0.08	1.57	16	0.90	0.03	1.16	13
elliptic	5464	1470	147	-	-	-	-	0.73	0.02	0.88	1.7
frisc	6002	992	148	-	-	-	-	0.81	0.01	1.08	1.6
s298	4268	8	73	-	-	-	-	0.65	0.02	1.55	2.4
s38417	13307	1463	730	-	-	-	-	0.13	0.20	1.85	2.1
tseng	1858	404	20	-	-	-	-	0.04	0.15	5.40	2.0
Seq. Avg.	5910	579	191	0.72	0.09	1.40	17	0.47	0.08	2.37	4.1

Secondly, it can be seen that the accuracy of ACE-1.0 and Sis-1.2 are similar for the combinational circuits, with correlations of 0.88 and 0.89, and average relative errors of 2% and 7%, respectively. These results are likely acceptable for power-aware CAD tools; however, the activity ratio indicates that both estimators tend to overestimate activities, which may translate to power overestimations.

Finally, the table shows the run-time of each estimator. The average Sis-1.2 run-times are not comparable to the other tools since Sis-1.2 did not find a solution for many of the circuits. By inspection of the circuits that did pass, however, we observed that the tool is relatively efficient compared to simulation. ACE-1.0, on the other hand, is somewhat slower with average

run-times that are approximately 8.8 and 4.1 times faster than simulation for combinational and sequential circuits, respectively.

2.3.2 FPGA Power Modeling

Comparing activities directly is useful since it reveals how accurate the activities are, if there are any trends such as over or underestimation, and if accuracy depends on circuit types. However, in order to determine the level of accuracy needed in the context of FPGAs, this subsection examines how the activities affect detailed power measurements of circuits implemented on FPGAs.

Experimental Framework

Figure 2.4 illustrates the experimental framework used to compare the effect of the activities on power estimates. The framework employs the VPR tool to place and route the benchmarks circuits and then uses the FPGA power model described in [13] to estimate the power of each implementation. All experiments target island-style FPGAs implemented in a 0.18 μ m TSMC CMOS process.

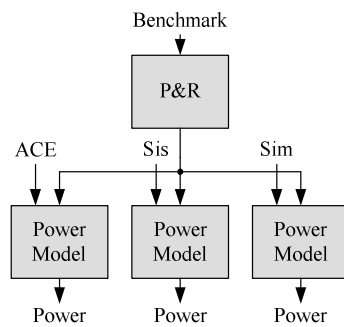


Figure 2.4: Power modeling framework.

Three different power estimates are obtained for each benchmark circuit. In each case, the same power model is used but the activities are produced using ACE-1.0, Sis-1.2, and gate-level simulation.

Results

Table 2.2 compares power estimates obtained using the activities from ACE-1.0 and Sis-1.2 to the power estimates obtained using simulated activities.

Table 2.2: Power estimates using ACE, Sis, and simulation.

Benchmark Circuits	Sim.	Sis-1.2		ACE-1.0	
	Power	Power	% Diff	Power	% Diff
alu4	1.02	1.39	35.6	1.40	37.0
apex2	0.94	1.25	33.7	1.25	33.8
apex4	0.69	0.77	11.9	0.75	9.3
C6288	0.85	-	-	1.05	22.8
des	1.73	2.02	16.5	2.04	17.8
ex1010	1.49	-	-	1.58	5.7
ex5p	0.42	0.51	22.5	0.50	19.7
misex3	0.83	1.03	24.3	1.03	24.4
pdc	2.50	2.69	7.7	2.70	8.1
seq	1.03	1.22	18.1	1.22	18.4
spla	1.50	1.62	7.6	1.62	7.4
Comb. Avg.	1.18	1.39	19.8	1.38	18.6
bigkey	0.78	0.92	16.9	1.66	112
clma	1.68	-	-	6.80	304
diffeq	0.20	-	-	0.81	296
dsip	0.60	0.85	42.9	1.44	139
elliptic	0.67	-	-	2.23	231
frisc	0.80	-	-	2.12	164
s298	0.49	-	-	1.57	220
s38417	2.95	-	-	7.96	170
tseng	0.19	-	-	0.83	344
Seq. Avg.	0.93	0.88	29.9	2.82	220

The table shows that both activity estimators tend to overestimate power as predicted by the *activity ratio* metric in Section 2.3.1. The most severe cases occur when ACE-1.0 is used for sequential circuits. This trend follows from the observation made in the previous section regarding the overestimation of the switching activities. Clearly, another technique should be used for sequential circuits. The ACE-1.0 results suggest that the filter function employed to reduce glitching is not accurate enough in this context. The Sis-1.2 results were unexpected since Sis-1.2 estimates switching probabilities which ignores glitching. The reason why Sis-1.2 still overestimates power is that it uses static probabilities (P_1) for primary inputs nodes and assumes that switching probabilities (P_s) are equal to $2*P_1*(1-P_1)$, which is the upper bound. The pseudo-randomly generated activities used in our framework chooses probabilities between 0 and $2*P_1*(1-P_1)$, which is more realistic. As an example, a wire which is 0 half of the time and 1 the other half does not necessarily toggle every clock cycle (as is inherently assumed in Sis-1.2). Thus, on average, the Sis-1.2 activity estimator assumes higher activities at the primary inputs of each circuit, which lead to overestimation throughout the circuit.

2.3.3 Power-Aware CAD for FPGAs

This section examines how the accuracy of the activity estimation techniques affects power-aware CAD tools that use switching activities to minimize the power of circuits implemented on FPGAs. Intuitively, the most important characteristic of the activities being used to minimize power is fidelity since the tools must know which wires to optimize for power.

Experimental Framework

Figure 2.5 illustrates the framework used to compare the effect of the activities on the power-aware version of VPR from [1], by implementing each benchmark three times using activities generated using ACE-1.0, Sis-1.2, and gate-level simulation. The power of all three implementations of the circuit is then estimated using the FPGA power model from [13] and activities obtained using simulation.

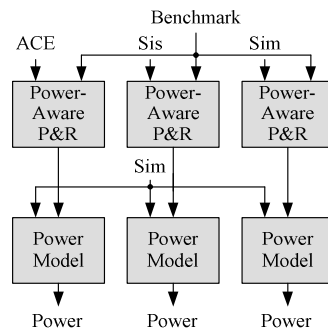


Figure 2.5: Power-aware CAD framework.

Results

Table 2.3 reports the power savings obtained by the power-aware version of VPR using the three different activity estimators. Intuitively, the greatest power savings should be obtained when power-aware VPR is guided by the simulated activities since they are exactly the same as the activities used to estimate power.

On average, the power-aware CAD flow reduces power by 9.8% for combinational circuits and 14.3% for sequential circuits when driven by the simulated activities. Although no average is available for Sis-1.2, careful inspection reveals power savings achieved using activities from Sis-1.2 and ACE-1.0 are approximately 9%, which is relatively good. This follows from the good correlation results measured in Section 2.3.1. For sequential circuits; however, the savings using ACE-1.0 activities are only 5.8% compared to 14.3%, which is not acceptable. These results again suggest that more accurate techniques must be employed to improve the performance of power-aware CAD tools.

Table 2.3: Power savings using each activity estimator.

Benchmark Circuits	Power Savings (%)		
	Simulation	Sis-1.2	ACE-1.0
alu4	10.5	9.5	9.3
apex2	14.5	13.9	12.3
apex4	11.6	10.9	10.0
C6288	7.9	-	8.2
des	9.0	9.1	9.8
ex1010	5.3	1.9	-1.4
ex5p	8.8	8.1	8.6
misex3	15.1	14.5	14.7
pdcc	2.1	0.0	2.5
seq	17.3	17.0	16.4
spla	5.4	5.5	5.6
Comb. Avg.	9.8	9.0	9.0
bigkey	34.4	22.8	21.8
clma	2.8	-	1.3
diffcq	8.5	-	-13.7
dsip	19.6	7.6	13.8
elliptic	10.6	-	-10.4
frisc	14.1	-	7.9
s298	3.4	-	-0.6
s38417	7.2	-	6.8
tseng	21.3	-	20.9
Seq. Avg.	14.3	-	5.8

2.4 ACE-2.0: A New Activity Estimator

The previous section served to highlight the strengths and weaknesses of existing activity estimation tools in the context of FPGAs. Specifically it found that the technique used in Sis-1.0 for circuits with sequential feedback does not scale well enough for larger circuits and the technique in ACE-1.0 was inaccurate, causing significant error in power estimates and degraded power savings from the power-aware CAD tool. The previous section also showed that the Transition Density model combined with a low-pass analytical filter were somewhat accurate but still caused power model to overestimate power in large circuits.

Using this information, this section describes a new activity estimation tool called ACE-2.0, which addresses these weaknesses. Figure 2.6 outlines the ACE-2.0 algorithm, which has three phases. The first phase begins by simulating static and switching probabilities for logic within sequential feedback loops, if there are any. The second phase then employs the Lag-one model, described in [8], to calculate static and switching probabilities for the remaining logic that has not been simulated. Finally, the third phase calculates switching activities using a novel probability-based technique that considers glitching. Each phase is described below.

```

ACE-2.0 (network, vectors, activities) {
  // Phase 1
  feedback_latch = find_feedback_latches (network);
  feedback_logic = find_feedback_logic (feedback_latches);
  if (vectors == NULL) vectors = gen_vectors (activities);
  simulate_probabilities (feedback_logic, vectors);

  // Phase 2
  foreach node n ∈ network
    if (Status(n) != SIMULATED) {
      bdd = get_partially_collapsed_and_pruned_bdd (n);
      Static_Prob(n) = calc_static_prob (bdd);
      Switch_Prob(n) = calc_switch_prob (bdd);
    }
  }
  // Phase 3
  foreach node n ∈ network {
    bdd = get_local_bdd(n);
    Switch_Act(n) = calc_switch_activity (bdd);
  }
}

```

Figure 2.6: ACE-2.0 pseudo code.

2.4.1 Phase 1

This phase determines the static and switching probability for logic and flip-flops within sequential feedback loops in a circuit. The previous section demonstrated that sequential feedback is the greatest source of error and long execution times for existing tools. The existing probability-based techniques proposed in the literature use BDDs or involve solving systems of non-linear equations. In either case, the techniques are not feasible for large circuits. Thus, the solution used within ACE-2.0 is to use a simplified form of simulation.

Two simplifications are made to improve the efficiency of the simulation. The first simplification is to simulate only the logic within sequential feedback loops. In most circuits with sequential feedback, the logic within feedback loops accounts for only a fraction of the circuit. Simulating this logic produces accurate probabilities within the feedback loops which can then be used when calculating probabilities for the remaining logic in the circuit.

The second simplification is to simulate switching probabilities instead of switching activities. In circuits with many levels of logic or with many exclusive-or gates, such as C6288, glitching accounts for a significant proportion of all transitions. As opposed to simulating activities, which involves processing each transition in each gate for every cycle, simulating probabilities only involves processing the final value of each gate for every cycle. The simulation routine used in ACE-2.0 is described in Figure 2.7.

```

simulate_probabilities (feedback_logic, vectors) {
  foreach vector  $\in$  vectors {
    update_primary_inputs (vector);
    evaluate_logic (feedback_logic);
    update_flip_flops ();
  }
}

evaluate_logic (logic) {
  foreach node n  $\in$  logic
    if (get_status_of_inputs (n) == NEW) {
      value = evaluate (n);
      if (value != Value(n)) {
        Value(n) = value;
        Status(n) = NEW;
        Num_transitions(n)++;
      } else {
        Status(n) = OLD;
      }
    } else {
      Status(n) = OLD;
    }
  }
  if (Value(n) == 1) Num_ones(n)++;
}
}

```

Figure 2.7: Simplified simulation pseudo-code.

Each cycle of the simulation begins by updating the values of the primary inputs with the next input vector. If vectors are not supplied, ACE-2.0 pseudo-randomly generates vectors which match the specified input activities. Once the input values are specified, the routine determines the output value of each gate in the feedback logic in topological order from the primary inputs to the primary outputs. Finally, at the end of each cycle, the routine updates the value at the output of each flip-flop based on the input value. To further improve performance, the evaluate logic routine only evaluates when at least one fanin of the gate has changed.

2.4.2 Phase 2

Although ACE-2.0 uses simulation to obtain static and switching probabilities for logic and flip-flops within sequential feedback loops, switching probabilities are also required for logic and flip-flops not within sequential feedback loops. These remaining probabilities are calculated using the Lag-one model [8], which produces exact switching probabilities (assuming that inputs are not correlated).

For a Boolean function f , the Lag-one model can be calculated by summing probabilities over all pairs of input states $\{x_i, x_j\}$ such that $f(x_i) = \overline{f(x_j)}$. Intuitively, you can think of an input

state as a single row of the truth table representation of function f . Explicitly, the switching probability P_s can be calculated using the following expression:

$$P_s = \sum_{x_i \in X_1} \left[P_1(x_i) \cdot \sum_{x_j \in X_0} P_s(x_i, x_j) \right] + \sum_{x_i \in X_0} \left[P_1(x_i) \cdot \sum_{x_j \in X_1} P_s(x_i, x_j) \right] \quad (2.4)$$

where X_l is the set of input states $x_i \in X_l$ such that $f(x_i) = 1$, X_0 is the set of input states $x_j \in X_0$ such that $f(x_j) = 0$, $P(x_i)$ is the probability that the current input state is x_i , and $P(x_i, x_j)$ is the probability that the input state will be x_j at the end of a clock cycle given that the input state was x_i at the beginning of the clock cycle.

The probability that the current input state is x_i can be determined by taking the product of the static probabilities for each input literal, as expressed below:

$$P_1(x_i) = \prod_{k=1}^n P(f_k, x_i[k]) \quad | \quad P(f_k, x_i[k]) = \begin{cases} P_1(f_k) & x_i[k] = 1 \\ 1 - P_1(f_k) & x_i[k] = 0 \end{cases} \quad (2.5)$$

where n is the number of inputs that fan into function f , f_k is the function of the k^{th} input that fans into f , and $x_i[k]$ is the value of the k^{th} literal of input state x_i .

Similarly, the probability that the input state will change from x_i to x_j can be determined by taking the product of the switching probabilities for each input literal, as expressed below:

$$P_s(x_i, x_j) = \prod_{i=k}^n P(f_k, x_i[k], x_j[k]) \quad (2.6)$$

$$P(f_k, x_i[k], x_j[k]) = \begin{cases} P_{0 \rightarrow 1}(f_k) & x_i[k] = 0, x_j[k] = 1 \\ 1 - P_{0 \rightarrow 1}(f_k) & x_i[k] = 0, x_j[k] = 0 \\ P_{1 \rightarrow 0}(f_k) & x_i[k] = 1, x_j[k] = 0 \\ 1 - P_{1 \rightarrow 0}(f_k) & x_i[k] = 1, x_j[k] = 1 \end{cases} \quad (2.7)$$

where $P_{0 \rightarrow 1}(f_k)$ is the probability that f_k will transition from 0 to 1 and $P_{1 \rightarrow 0}(f_k)$ is the probability that f_k will transition from 1 to 0. These probabilities can be determined with the following expressions:

$$P_{0 \rightarrow 1}(f_k) = \frac{P_s(f_k)}{2 \cdot (1 - P_1(f_k))} \quad \text{and} \quad P_{1 \rightarrow 0}(f_k) = \frac{P_s(f_k)}{2 \cdot P_1(f_k)} \quad (2.8)$$

The most efficient implementation of the Lag-one model involves using a BDD. However, as observed in Section 2.3, using BDDs become infeasible for large circuits because of the

exponential relationship between BDD size and the number of inputs. As a solution, ACE-2.0 combines the *partial collapsing* technique described in [20] with *BDD pruning* as an approximation to improve the speed of the calculation.

Partial Collapsing

The activity estimation techniques described in [7],[8] suffer in that they require collapsing a large number of nodes. During activity estimation, each node is collapsed with all of its predecessors. Although this takes into account spatial correlation within the circuit, the technique is infeasible for large circuits. In [20], an alternative is proposed; rather than collapsing each node with all its predecessors, only smaller portions of the logic are collapsed, as shown in Figure 2.8. This results in smaller BDDs, leading to faster activity estimation time. The cost of doing this is that not all spatial correlation can be captured. There is a tradeoff between the amount of partial collapsing (the size of each collapsed node) and the error introduced by not taking into account all spatial correlation.

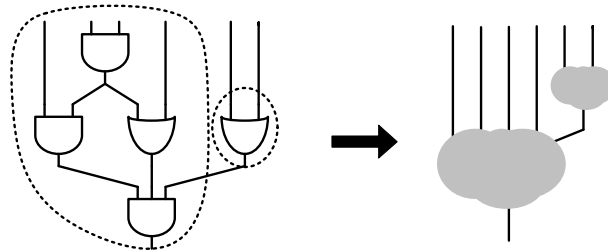


Figure 2.8: Example of a partially collapsed network.

The `partial_collapse` routine collapses nodes with only some (as opposed to all) of its predecessors such that the BDD representation of the collapsed logic contains at most `max_nodes` nodes. The `max_nodes` parameter can be used to tradeoff accuracy with run-time.

BDD Pruning

An alternative to partial collapsing is BDD pruning. By pruning low-probability branches of a BDD, the size of a BDD can be significantly reduced with minimal impact on the accuracy of the activity estimation. In logic synthesis applications (which also use BDDs to represent logic nodes), such pruning is not possible since it would be unacceptable to change the behavior of the circuit. For activity estimation, however, the BDDs can be pruned since some degree of approximation is acceptable. Although pruning BDDs is not a new idea, pruning BDDs to improve the execution time of activity calculations is novel.

The proposed BDD pruning technique involves removing BDD branches with probabilities smaller than a pruning threshold probability, min_prob . An example is illustrated in Figure 2.9. In the example, the static probability of the function's four inputs are $P_1(a)=0.8$, $P_1(b)=0.9$, $P_1(c)=0.7$, and $P_1(d)=0.5$, and the probability of each branch is shown next to each node. The grey nodes are pruned from the BDD since their probabilities are smaller than $min_prob = 0.05$.

After a branch is pruned, it must be replaced by either a '0' or a '1' terminal for the BDD to remain valid. A naïve approach is to arbitrarily use a '0' or a '1' terminal; however, this produces more error later on when the BDDs are used to calculate activities. A better approach is to replace the branch with a '0' terminal when the value at the output of the function is more likely to be '0' ($P_{branch} < 0.5$) and a '1' terminal when the value is more likely to be '1' ($P_{branch} \geq 0.5$), given that you are at that BDD node.

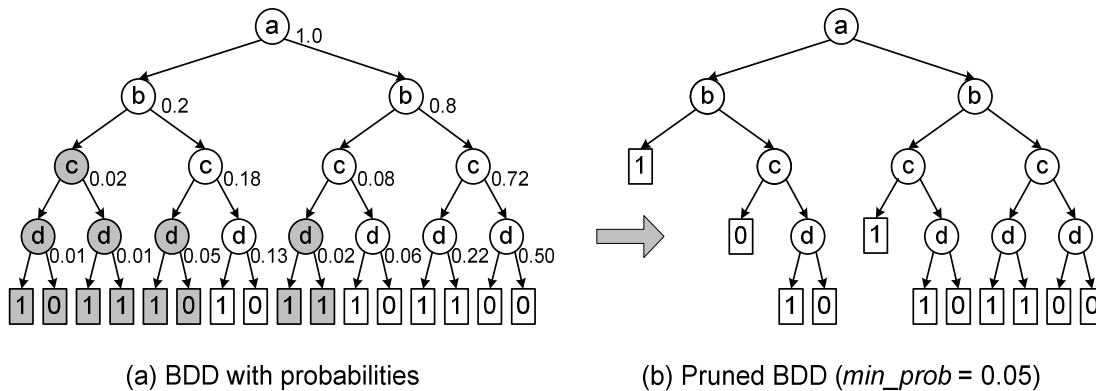


Figure 2.9: Example of BDD pruning.

The threshold probability parameter, min_prob , controls the tradeoff between accuracy and execution time. Increasing min_prob increases pruning and results in smaller BDDs, faster execution times, but reduced accuracy.

Combining Partial Collapsing and BDD Pruning

Partial collapsing and BDD pruning can be combined. In Figure 2.11, the BDD generated by the *partial_collapse* routine is then pruned using the *bdd_prune* routine. The resulting routine has two parameters: max_size and min_prob . Intuitively, the max_size parameter limits the BDD size of the collapsed node and the min_prob parameter controls the amount of pruning that occurs on the BDD representation of the collapsed node. The best min_prob value depends on the size of the BDD to be pruned. Pruning a very large BDD with a large min_prob value would significantly decrease accuracy. Conversely, pruning a small BDD with a small min_prob value would be ineffective. By sweeping the max_size parameter between 0 and 1280, and the min_prob

parameter between 0.0005 and 0.008, a simple expression that produces good results was obtained:

$$\min_prob_{good} = \frac{1}{2 \cdot \max_size} \quad (2.9)$$

Using the above expression, the switching probabilities were calculated and compared to those obtained with partial collapsing.

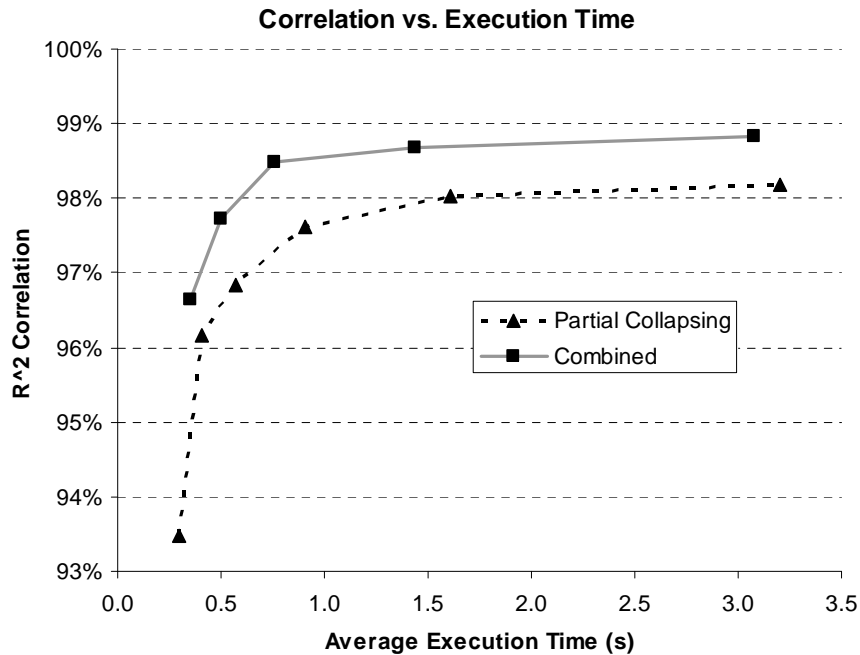


Figure 2.10: Correlation vs. execution time.

Figure 2.10 illustrates the tradeoff between the correlation and execution time of the switching probability estimations. The x-axis is the average execution time of the switching probability estimation per benchmark circuit and the y-axis is the relative error of the switching probability estimates averaged over every node in every benchmark circuit. The points on the graph correspond to various BDD size and BDD pruning threshold parameter values. Two useful observations can be made from this figure. The first observation is that the performance when partial collapsing and BDD pruning are combined is significantly improved when compared to partial collapsing on its own. In order to obtain a correlation of 98%, the combined approach is approximately 3 times faster than collapsing approach. The second observation relates the appropriate configuration of the BDD size and BDD pruning parameters. The figure illustrates that accuracy of both approaches quickly improve up to a certain point and then level

off. For the combined approach, the best results were obtained *max_size* and *min_prob* values of 50 and 0.01, respectively.

```

get_partially_collapsed_pruned_bdd (n, max_size, min_prob) {
  bdd = partially_collapse (n, max_size); // see [20]
  pruned_bdd = bdd_prune (bdd, 1.0, min_prob);
  return (pruned_bdd);
}

bdd_prune (bdd, prob, min_prob) {
  if (bdd is a '0' or '1' terminal) return (bdd);

  if (prob < min_prob) {
    if (Prob(bdd) < 0.5) {
      return ('0'); // replace branch with '0'
    } else {
      return ('1'); // replace branch with '1'
    }
  }
  }
  n = literal (bdd);
  true_prob = prob · P1(n);
  false_prob = prob · (1.0 – P1(n));
  true_bdd = bdd_prune (bdd->>true, true_prob, min_prob);
  false_bdd = bdd_prune (bdd->>false, false_prob, min_prob);
  bdd' = build_bdd (n, true_bdd, false_bdd);
  return (bdd');
}

```

Figure 2.11: BDD pruning pseudo code.

2.4.3 Phase 3

The final phase of the ACE-2.0 algorithm addresses the issue of accurately and efficiently modeling the glitch component of switching activities. This subsection introduces the novel switching activity calculation that we used. The calculation is a simple generalization of the Lag-one model, yet performs well compared to existing techniques.

A transition at the output of a gate is normally caused by a transition occurring at a single input of that gate; however, transitions can also occur (or be canceled out) when two or more inputs transition at nearly the same time. Consider a two-input XOR gate with inputs A and B. If A is '0' and B transitions from '0' to '1', this transition will probably cause a transition at the output. Similarly, a second transition, this time of input 'A', will probably cause a second transition at the output. However, these two input transitions might not cause a transition at the output if they happen at the same (or nearly the same) time since the glitch generated by these two input transition may be filtered out by the resistance and capacitance of the gate. In other words, the amount of glitching that occurs depends on the minimum pulse width of the gate.

The new calculation adds the notion of minimum pulse width to the Lag-one model. Explicitly, the switching activity A_s is calculated using the following expressions:

$$A_s(f) = \frac{T}{\tau} \cdot P_s(f) \quad (2.10)$$

$$P_{0 \rightarrow 1}(f_k) = \frac{P_s(f_k)}{2 \cdot (1 - P_1(f_k))} \cdot \frac{\tau}{T} \quad \text{and} \quad P_{1 \rightarrow 0}(f_k) = \frac{P_s(f_k)}{2 \cdot P_1(f_k)} \cdot \frac{\tau}{T} \quad (2.11)$$

where T is the amount of time that separates the arrival times of the earliest and the latest input signals of node f and τ is some period of time (less than or equal to T) that represents the minimum pulse width of that node. Note carefully that this differs slightly from our earlier implementation, described in [14], where T is the maximum delay from the primary inputs to the output of the function f . The newer implementation is more exact since glitches are only generated during this window of time.

The switching probabilities (P_s) are still calculated using equation (2.4) but that $P_{0 \rightarrow 1}$ and $P_{1 \rightarrow 0}$ are calculated using (2.11) instead of (2.8). Intuitively, the calculation determines the switching probability during period $\tau < T$ (assuming that the input arrival times are normally distributed) and then multiplies this probability by T/τ (the number times that period τ occurs during T).

It is interesting to note that when $\tau = T$ the new switching activity model reduces to the Lag-one model (which ignores glitching) since output transitions caused by any number of input transitions are equally weighted. Conversely, when $\tau \rightarrow 0$, the model reduces to the Transition Density model (which overestimates glitching) since only single input transitions carry any weight in the calculation. The most accurate results are obtained when τ is set to approximately the physical delay of the gate.

2.4.4 ACE-2.0 Results

Table 2.4 compares the activities obtained using ACE-2.0 to the activities from simulation. The table shows that the estimated activities are very accurate for both combinational and sequential circuits. In particular, the correlation for combinational circuits is 0.97, which is close to ideal; and the correlation is for sequential circuits is 0.86, which is significantly better than ACE-1.0, which was 0.47 in Table 2.1. Moreover, the last column in the table shows that ACE-2.0 is 69 times faster than simulation for combinational circuits and 7.2 times faster than simulation for sequential circuits.

Table 2.4: ACE-2.0 results.

Benchmark Circuits	ACE-2.0 Activities vs. Simulated Activities			
	R ²	Act. Rat.	Avg. Rel. Error	Speedup
alu4	0.93	1.02	0.01	68
apex2	0.97	0.98	0.00	97
apex4	0.99	0.90	0.00	57
C6288	0.93	1.33	0.28	45
des	0.98	0.89	0.02	33
ex1010	0.99	0.96	0.00	51
ex5p	0.98	0.91	0.01	88
misex3	0.95	0.96	0.01	80
pdc	0.97	0.89	0.00	47
seq	0.97	0.94	0.01	99
spla	0.99	0.87	0.00	96
Comb. Mean.	0.97[†]	0.97[†]	0.03[*]	69[*]
bigkey	0.63	1.52	0.09	11
clma	0.92	0.94	0.00	5.2
diffeq	0.75	0.80	0.01	6.0
dsip	0.99	0.97	0.01	12
elliptic	0.90	0.69	0.01	6.1
frisc	0.95	0.77	0.00	6.3
s298	0.91	0.99	0.01	6.1
s38417	0.89	1.07	0.03	8.8
tseng	0.76	1.23	0.01	2.5
Seq. Mean.	0.86[†]	1.00[†]	0.02[*]	7.2[*]

[†] Calculated using the geometric mean.

^{*} Calculated using the arithmetic average.

Table 2.5 compares FPGA power estimates when estimated activities are used. The table shows that ACE-2.0 activities produce very accurate power estimates, with less than 1% error compared to simulation. Finally, Table 2.6 compares the power savings obtained when estimated activities are used to drive power-aware FPGA CAD tools. The table shows that ACE-2.0 activities produce power savings that closely match those obtained using simulated activities.

2.5 Conclusions and Future Work

This chapter examined various activities estimation techniques in order to determine which are most appropriate for use in the context of FPGAs. It found that existing probability-based activity estimation tools were either too slow or too inaccurate for circuits with sequential feedback; causing inaccurate power estimations and reduced power savings in FPGA CAD tools. It also found that using fully collapsed logic to calculate probabilities is not feasible for large circuits because of execution time. Finally, it found that calculating switching activities using

the Transition Density and the associated low-pass filter caused the power model to overestimate power.

Given the above findings, a new activity estimation tool called ACE-2.0 that incorporates the techniques found most suitable was described. The new tool begins by calculating static and switching probabilities for every node in the circuit. For circuits with sequential feedback, a simplified simulation technique is used for the feedback logic and the Lag-one model is used for the remaining logic. To improve the speed of the Lag-one calculation with only a slight loss of accuracy, BDD sizes were reduced using partial collapsing and BDD pruning. Once the static and switching probabilities are obtained, ACE-2.0 employs a novel probability-based technique to calculate the switching activities.

Finally, the new tool was validated in the context of FPGAs. Using activities estimated by ACE-2.0, power estimates and power savings were both within 1% of results obtained using simulated activities. Moreover, the new tool was 69 and 7.2 times faster than simulation for combinational and sequential circuits, respectively.

Table 2.5: Power estimates using ACE-2.0.

Benchmark Circuits	Sim.	Sis-1.2		ACE-1.0		ACE-2.0	
	Power	Power	% Diff	Power	% Diff	Power	% Diff
alu4	1.02	1.39	35.6	1.40	37.0	1.02	-0.2
apex2	0.94	1.25	33.7	1.25	33.8	0.94	-0.1
apex4	0.69	0.77	11.9	0.75	9.3	0.69	-0.4
C6288	0.85	-	-	1.05	22.8	0.86	1.6
des	1.73	2.02	16.5	2.04	17.8	1.72	-0.6
ex1010	1.49	-	-	1.58	5.7	1.49	-0.3
ex5p	0.42	0.51	22.5	0.50	19.7	0.42	0.0
misex3	0.83	1.03	24.3	1.03	24.4	0.83	-0.2
pdc	2.50	2.69	7.7	2.70	8.1	2.50	0.0
seq	1.03	1.22	18.1	1.22	18.4	1.02	-0.5
spla	1.50	1.62	7.6	1.62	7.4	1.50	0.0
Comb. Avg.	1.18	1.39	19.8	1.38	18.6	1.18	-0.1
bigkey	0.78	0.92	16.9	1.66	112	0.78	0.1
clma	1.68	-	-	6.80	304	1.68	0.2
diffeq	0.20	-	-	0.81	296	0.20	0.2
dsip	0.60	0.85	42.9	1.44	139	0.60	0.1
elliptic	0.67	-	-	2.23	231	0.67	0.1
frisc	0.80	-	-	2.12	164	0.80	0.2
s298	0.49	-	-	1.57	220	0.51	4.0
s38417	2.95	-	-	7.96	170	2.96	0.3
tseng	0.19	-	-	0.83	344	0.19	0.2
Seq. Avg.	0.93	0.88	29.9	2.82	220	0.94	0.6

Table 2.6. Power savings using ACE-2.0.

Benchmark Circuits	Power Savings (%)			
	Simulation	Sis-1.2	ACE-1.0	ACE-2.0
alu4	10.5	9.5	9.3	10.8
apex2	14.5	13.9	12.3	14.6
apex4	11.6	10.9	10.0	11.2
C6288	7.9	-	8.2	9.6
des	9.0	9.1	9.8	9.7
ex1010	5.3	1.9	-1.4	-2.0
ex5p	8.8	8.1	8.6	8.5
misex3	15.1	14.5	14.7	14.1
pdc	2.1	0.0	2.5	1.3
seq	17.3	17.0	16.4	17.0
spla	5.4	5.5	5.6	1.8
Comb. Avg.	9.8	9.0	9.0	8.8
bigkey	34.4	22.8	21.8	33.8
clma	2.8	-	1.3	2.8
diffeq	8.5	-	-13.7	11.2
dsip	19.6	7.6	13.8	19.6
elliptic	10.6	-	-10.4	9.0
frisc	14.1	-	7.9	15.8
s298	3.4	-	-0.6	5.2
s38417	7.2	-	6.8	7.2
tseng	21.3	-	20.9	21.4
Seq. Avg.	14.3	-	5.8	14.0

Chapter 2 References

- [1] J. Lamoureux, S.J.E. Wilton, On the Interaction Between Power-Aware Computer-Aided Design Algorithms for Field-Programmable Gate Arrays, *Journal of Low Power Electronics (JOLPE)*, Vol. 1, Issue 2, pp. 119-132(14), Aug. 2005.
- [2] F. Najm, Transition density: A new measure of activity in digital circuits, in *IEEE Trans. Computer-Aided Design*, Vol. 12, Issue 2, pp. 310-323, 1993.
- [3] R. Burch, F. Najm, P. Yang, T. Trick, A Monte Carlo approach to power estimation, in *IEEE Trans. on VLSI Systems.*, Vol. 1, Issue 1, pp. 63-71, 1993.
- [4] C.Y. Tsui, M. Pedram, A.M. Despain, Efficient estimation of dynamic power consumption under a real delay model, in *IEEE Intl. Conf. Computer-Aided Design (ICCAD)*, pp. 224-228, 1993.
- [5] F. Najm, Low-pass filter for computing the transition density in digital circuits, in *IEEE Trans. Computer-Aided Design*, Vol. 13, Issue 9, pp. 1123-1131, 1994.
- [6] C.Y. Tsui, M. Pedram, A.M. Despain, Exact and approximate methods for calculating signal and transition probabilities in FSMs, in *ACM/IEEE Design Automation Conference (DAC)*, pp. 18-23, 1994.
- [7] J. Monteiro, S. Devadas, A methodology for efficient estimation of switching activity in sequential logic circuits, in *ACM/IEEE Design Automation Conference (DAC)*, pp. 12-17, 1994.
- [8] R. Marculescu, D. Marculescu, M. Pedram, Switching activity analysis considering spatiotemporal correlations, in the *IEEE Intl. Conf. Computer-Aided Design (ICCAD)*, pp. 294-299, 1994.
- [9] Q. Wu, M. Pedram, and X. Wu, A note on the relationship between signal probability and switching activity, in the *Proc. of the Asia and South Pacific Design Automation Conf.*, pp. 117-120, 1997.
- [10] J.N. Kozhaya, F. Najm, Accurate power estimation for large sequential circuits, in *IEEE Intl. Conf. Computer-Aided Design (ICCAD)*, pp. 488-493, 1997.
- [11] E. Todorovich, M. Gilabert, G. Sutter, S. Lopez-Buedo, and E. Boemo, A tool for activity estimation in FPGAs, in the *Intl. Conf. on Field-Programmable Logic (FPL)*, pp. 340-349, 2002.
- [12] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, Kluwer Academic Publishers, 1999.
- [13] K.K.W Poon, S.J.E Wilton, A detailed power model for field-programmable gate arrays, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, April 2005, Vol. 10, Issue 2, 2005, pp. 279-302.
- [14] J. Lamoureux, S.J.E. Wilton, Activity estimation for Field-Programmable Gate Arrays, in the *Intl. Conf. on Field-Programmable Logic (FPL)*, pp. 87-94, 2006.
- [15] P.H. Schneider, S. Krishnamoorthy, Effects of correlation on accuracy of power analysis – an experimental study, *ACM/IEEE Intl. Symp. of Low Power Electronics and Design (ISLPED)*, pp. 113-116, 1996.
- [16] K.D. Lamb, Use of binary decision diagrams in the modeling and synthesis of binary multipliers, in the *IEEE International ASIC Conference and Exhibit*, pp. 159-162, 1996.

- [17] S. Yang, Logic Synthesis and optimization benchmarks, Version 3.0, Tech. Report, Microelectronics Center of North Carolina, 1991.
- [18] M.C. Hansen, H. Yalcin, J.P. Hayes, Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering, IEEE Design & Test of Computers, Vol. 16, Issue 3, July-Sept. 1999, pp. 72-80.
- [19] Sheldon B. Akers, Binary Decision Diagrams, IEEE Transactions on Computers, C-27(6): pp. 509–516, June 1978.
- [20] B. Kapoor, Improving the accuracy of circuit activity measurement, ACM Design Automation Conference, pp. 734-739, 1994.

Chapter 3

GlitchLess: Dynamic Power Minimization in FPGAs through Edge Alignment and Glitch Filtering¹

3.1 Introduction

Field-Programmable Gate Arrays (FPGAs) are integrated circuits consisting of general purpose logic, routing, I/O, and clock resources that can be configured to implement any digital circuit. Smaller process technologies, more efficient programmable architectures, and smarter computer-aided design (CAD) tools are allowing increasingly larger and faster systems to be implemented on FPGAs. Despite these advancements in speed and area, power dissipation has continued to increase to the point where it has become one of the main challenges when implementing large applications using FPGAs. FPGAs dissipate significantly more power than application specific integrated circuits (ASICs) because of the added circuitry needed to make them programmable.

There are two types of power dissipation in integrated circuits: static and dynamic. Static (or leakage) power is dissipated when current leaks between the various terminals of a transistor, while dynamic power is dissipated when individual circuit nodes toggle. A study that examined power dissipation in a commercial 90nm FPGA found that dynamic power accounts for 62% of total power [1].

There are a number of ways to reduce power in FPGAs. Techniques that can be used at the physical level include lowering the supply voltage, to reduce dynamic power [2], or increasing the threshold voltage, to reduce leakage power [3]. At the circuit level, devices can be sized less aggressively for speed to minimize capacitive loading and therefore dynamic power [4]. At the architecture level, careful power management [5] and clock network design are also helpful [6]. Even at the CAD level, grouping logic with high-activity connections close together reduces dynamic power [7]. A summary of techniques that can be used to minimize power is described in [8].

¹ A version of this chapter has been submitted for publication. J. Lamoureux, G.G. Lemieux, and S.J.E. Wilton, Glitchless: dynamic power minimization in FPGAs through edge alignment and glitch filtering, IEEE Transactions on Very Large Scale Integration Systems (VLSI), 2007.

This chapter introduces a new circuit-level technique that reduces dynamic power in FPGAs by actively minimizing the number of unnecessary transitions called glitches or hazards. The proposed technique involves adding programmable delay elements within the programmable logic structures of an FPGA, called configurable logic blocks (CLBs), to programmably align the arrival times of early-arriving signals to the inputs of the lookup tables (LUTs) and to filter out glitches generated by earlier circuitry. Furthermore, the proposed technique can be used in collaboration with other low-power techniques and requires little or no modifications to the existing FPGA routing architecture or CAD flow, since it is applied after routing.

In theory, the proposed technique could be used to eliminate all the glitching within FPGAs thereby saving significant amounts of power. In practice, however, we must trade-off these power savings with the area and speed overhead incurred by the additional circuitry that is required. Fortunately, there is not a significant impact on circuit speed (other than increased parasitic capacitances), since only the early-arriving signals need to be delayed. However, the programmable delay elements do consume chip area, so we should expect a modest increase in the area of the programmable logic. This tradeoff between glitch reduction (and hence power), area, and speed will be quantified in this chapter. Specifically, this chapter examines the following questions:

1. How should the programmable delay elements be connected within the CLBs? The programmable delay elements could conceivably be connected to the inputs or outputs of each CLB or they could be connected to the inputs of the LUTs within the CLBs.
2. How many programmable delay elements are needed within each CLB? Intuitively, adding more programmable delay elements to the CLBs eliminates more glitches since more signals can be aligned; however, it also increases the area overhead.
3. How flexible should the programmable delay elements be? The more flexible each delay element is, the better it will be able to align the arrival times of signals. However, there is a tradeoff between this flexibility and the area overhead of the added circuits.
4. Does the delay insertion technique work when there is process, voltage, and/or temperature (PVT) variation? PVT affects both the delay of the existing FPGA logic and the delay of the programmable delay elements. Special measures must be taken to ensure that the delay insertion technique can tolerate variation well enough to eliminate glitches without introducing timing violations.

A preliminary version of this work appeared in [9]; however, this chapter expands the work by introducing an additional delay insertion scheme which reduces the area overhead and by proposing new techniques and a new programmable delay element that tolerates PVT variation more effectively than in the previous work.

This chapter is organized as follows. Section 3.2 presents terminology used in this chapter to describe glitching and PVT variation and summarizes existing techniques that can be used to minimize glitching. Section 3.3 examines how much glitching occurs within FPGAs. Section 3.4 presents the delay insertion schemes that are proposed in this chapter. Section 3.5 describes the experimental framework that is used to estimate power savings and area and delay overhead. Section 3.6 describes how each scheme is calibrated and Section 3.7 presents the overall power savings and the corresponding overhead. Finally, Section 3.8 summarizes the results and presents our conclusions.

3.2 Background

3.2.1 Switching Activity Terminology

There are two types of transitions that can occur on a signal. The first type is a functional transition, which is necessary in order to perform a computation. A functional transition causes the value of the signal to be different at the end of the clock cycle than at the beginning of the clock cycle. In each cycle, a functional transition occurs either once or the signal remains unchanged. The second type of transition is called a glitch (or a hazard) and is not necessary in order to perform a computation. These transitions can occur multiple times during a clock cycle.

3.2.2 Process, Voltage, and Temperature Variation

Process variation refers to the variability of device geometries, interconnect geometries, dopant concentrations, and dielectric properties. This variability is inherent to the semiconductor processes used during fabrication. Similarly, *voltage variation* refers to the variability of the power supply and *temperature variation* refers to variability of the temperature of the surrounding environment. PVT variations can either be *die-to-die* (in which various dies have different properties) or *within-die* (in which circuit elements on the same chip have different properties). In either case, these variations can affect both the timing and leakage power of the devices.

3.2.3 Existing Glitch Minimization Techniques

Several techniques have been proposed to minimize glitching. For example, CAD techniques including logic decomposition [10], loop folding [11], high-level compiler optimization [12], technology mapping [7],[13] and clustering [7] have been proposed to minimize switching activity. These techniques can eliminate some of the glitching, but typically incur area and delay penalties as they reorganize the structure of the circuit. Other approaches involve relocating flip-flops [14] or inserting additional flip-flops (pipelining) [15] to reduce the combinational path length. These techniques can also eliminate some of the glitching; however, significant power savings require additional flip-flops which increases the latency of the circuit. The *gate freezing* technique described in [16] eliminates glitching by suppressing transitions until the freeze gate is enabled. This technique is suitable for fixed implementations since it can be applied to selected gates with high glitch counts. However, the technique is less suitable for FPGAs since the applications implemented on FPGAs are not known until after fabrication, meaning it is difficult to determine, at fabrication time, where the extra circuitry should be added. Finally, the delay insertion technique described in [17] minimizes glitching in fixed logic implementations by aligning the input arrival times of gates using fixed delay elements. In this chapter, we propose a similar technique that targets FPGAs. Aligning edges in an FPGA is considerably more complex than doing so in an ASIC, since in an FPGA, the required delay times are not known when the chip is fabricated. This means the delays must be programmable; if not managed carefully, the overhead in these programmable delay elements can overwhelm any power savings obtained by removing glitches.

3.3 Glitching in FPGAs

This section begins with a breakdown of functional versus glitching activity to determine how much glitching occurs within FPGAs. It then examines the width of typical glitches and determines how much power is dissipated by a single glitch. Finally, it indicates how much power could be saved if glitching could be completely eliminated. These statistics are important, not only because they help motivate our work, but also because they provide key numbers (such as typical pulse width) that will be needed in Section 3.6 when the delay insertion schemes that are proposed in Section 3.4 are calibrated.

3.3.1 Switching Activity Breakdown

Table 3.1 reports the switching activities for a suite of benchmark circuits implemented on FPGAs. These activities are gathered using gate-level simulation of a post place-and-route implementation for a set of benchmark circuits (see Section 3.5 for more details). Gate-level simulations provide the functional and total activity; the glitching activity is computed as the difference between these two quantities. In general, the amount of glitching is greater in circuits with many levels of logic, circuits with uneven routing delays, and circuits with exclusive-or logic. As an example, an unpipelined 16-bit array multiplier (C6288) implemented on an FPGA has four times more glitch transitions than functional transitions.

Table 3.1. Breakdown of Switching Activity

Circuit	Logic Depth	Activity	Functional Activity	Glitch Activity	% Glitch
C1355	4	0.32	0.23	0.09	27.5
C1908	10	0.26	0.17	0.09	34.6
C2670	7	0.27	0.21	0.06	22.2
C3540	12	0.42	0.23	0.19	45.2
C432	11	0.26	0.18	0.08	29.3
C499	4	0.34	0.23	0.11	31.9
C5315	10	0.40	0.25	0.15	36.7
C6288	28	1.56	0.29	1.27	81.1
C7552	9	0.39	0.23	0.16	42.0
C880	9	0.23	0.19	0.05	19.8
alu4	7	0.08	0.07	0.01	13.1
apex2	8	0.05	0.04	0.01	13.7
apex4	6	0.04	0.03	0.01	32.3
des	6	0.27	0.17	0.10	36.8
ex1010	8	0.03	0.01	0.02	52.9
ex5p	7	0.17	0.08	0.09	51.0
misex3	7	0.06	0.05	0.01	20.9
pdc	9	0.03	0.02	0.01	31.8
seq	7	0.05	0.04	0.01	16.0
spla	8	0.05	0.03	0.02	42.7
Geomean	8.1	0.024	0.019	0.047	30.8

3.3.2 Pulse Width Distribution

In FPGAs, glitches are generated at the output of a LUT when the input signals transition at different times. The *pulse width* of these glitches depends on how uneven the input arrival times are. Intuitively, we would expect FPGA glitches to be wider than ASIC glitches, since signals are often routed using non-direct paths due to the limited connectivity of FPGA routing resources. Figure 3.1 plots the pulse width distribution of the C6288 benchmark circuit. The

graph shows that the majority of glitches have a pulse width between 0 and approximately 10 ns. Although this range varies across our benchmark circuits, we have found that the shape of the distribution is similar for every circuit.

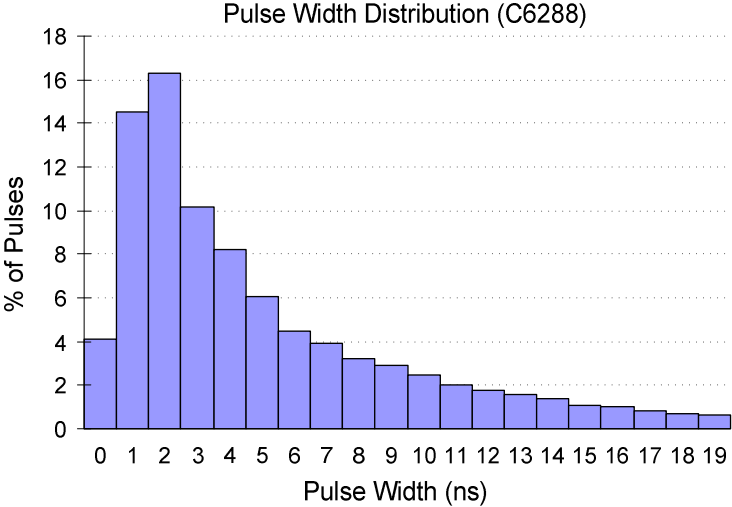


Figure 3.1. Pulse width distribution of glitches.

3.3.3 Power Dissipation of Glitches

The parasitic resistance and capacitances of the routing resources filter out very short glitches. To measure the impact of this, HSPICE was used to build a profile of power with respect to pulse width. Figure 3.2 illustrates the relative power dissipated when pulses with widths ranging from 0 to 1ns are applied to an FPGA routing track that spans four CLBs. A 180nm process was assumed.

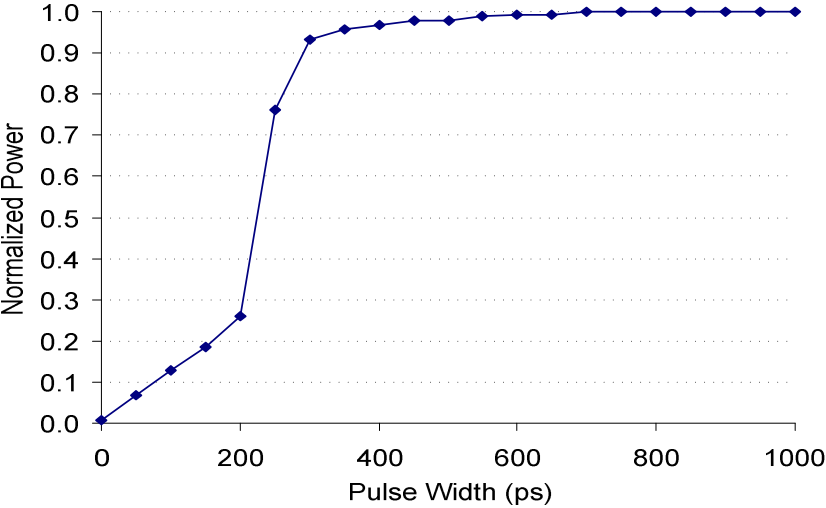


Figure 3.2. Normalized power vs. pulse width.

The graph shows that pulses less than or equal to 200 ps in duration are mostly filtered out by the routing resources. Pulses that are longer than 300 ps in duration dissipate approximately the same amount of power as longer pulses. Thus, if the input signals of a gate arrive within a 200 ps window, the glitching of that gate is effectively eliminated.

3.3.4 Potential Power Savings

Table 3.2 reports the average total power dissipated by circuits when implemented on an FPGA. The second column reports the power of the circuits in the normal case, when glitching is allowed to occur. The third column reports the power in the ideal case, when glitching is completely eliminated and there is no overhead. The fourth column shows the percent difference between the two power estimates; this number indicates how much power could be saved if glitching was completely eliminated without any overhead. Depending on the circuit, the potential power saving ranges between 4% and 73%, with average savings of 22.6%. These numbers motivate a technique for reducing glitching in FPGAs.

Table 3.2. FPGA power with and without glitching.

Circuit	Power (mW) (glitching)	Power (mW) (no glitching)	% Difference
C1355	9.5	6.7	28.8
C1908	6.2	4.9	21.1
C2670	21.5	18.6	13.4
C3540	21.3	14.6	31.7
C432	4.6	3.8	17.1
C499	8.7	5.7	34.6
C5315	34.7	26.8	22.8
C6288	41.6	11.2	73.1
C7552	39.9	29.8	25.5
C880	5.8	5.3	9.6
alu4	39.2	37.8	3.6
apex2	41.2	39.4	4.3
apex4	24.5	22.0	10.1
des	88.2	72.4	17.9
ex1010	51.4	41.9	18.4
ex5p	29.7	21.4	28.1
misex3	41.6	38.3	8.1
pdcc	35.8	31.0	13.3
seq	38.3	36.0	6.1
spla	45.5	35.8	21.4
Geomean	24.3	18.8	22.6

3.4 Proposed Glitching Elimination Techniques

This section describes the technique proposed in this chapter to eliminate glitching. It begins by describing the proposed technique and discusses other possible techniques as well. It then presents five variations (or schemes) of the proposed technique, which employ delay elements in different locations within the FPGA logic blocks. It then describes the programmable delay element that is used to align the arrival times and the CAD algorithms that are used to configure these programmable delay elements. Finally, it describes techniques that can be used to make programmable delay insertion more tolerant to PVT variation.

3.4.1 Glitch Elimination Techniques

Our proposed technique involves adding programmable delay elements within the CLBs of the FPGA. Within each CLB, the programmable delay elements are configured to delay early-arriving signals so as to align the arrival times on each LUT input to eliminate glitching. The technique is shown in Figure 3.3; by delaying input c , the output glitch is eliminated. Note that the overall critical-path of the circuit is not increased since only the early-arriving inputs are delayed.

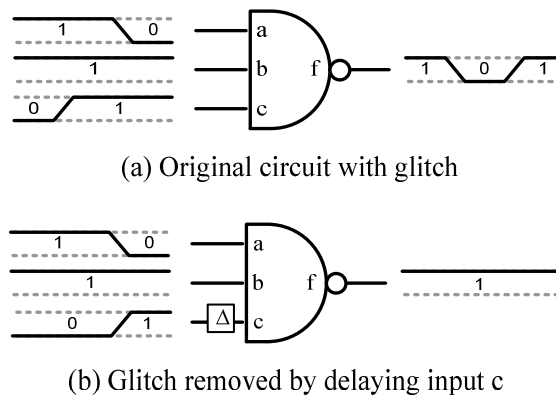


Figure 3.3. Removing glitches by delaying early-arriving signals.

Another technique that we considered involved modifying the placement and routing algorithms to be glitch-aware. By placing CLBs at even distances from common sources and/or routing connections to balance arrival-times, the amount of glitching could likely be reduced. The inherent problem with this approach is that it is difficult to balance arrival-times by making the late-arriving fanins faster since the CAD algorithms have already been optimized for to minimize critical-path delay. The other alternative is to balance arrival-times by making the early-arriving signals slower. This approach; however, would not minimize power as efficiently

as the proposed technique since the routing resources, which would effectively be used to add delay early arriving signals, dissipate more dynamic power than the proposed programmable delay element, which uses a large resistance (as opposed to capacitance) to delay signals.

3.4.2 Architectural Alternatives

We consider five alternative schemes for implementing the delay insertion technique; the schemes differ in the location of the delay elements within the CLB. Figure 3.4 (a) illustrates the baseline CLB. A CLB consists of LUTs, flip-flops, and local interconnect. The LUTs and FFs are paired together into Basic Logic Elements (BLEs). Three parameters are used to describe a CLB: I specifies the number of input pins, N specifies the number of BLEs and output pins, and K specifies the size of the LUTs. The local interconnect allows each BLE input to choose from any of the I CLB inputs and N BLE outputs. Each BLE output drives a CLB output. The five schemes we consider for adding delay elements to a CLB are illustrated in Figure 3.4 (b-f). Each of which is described below.

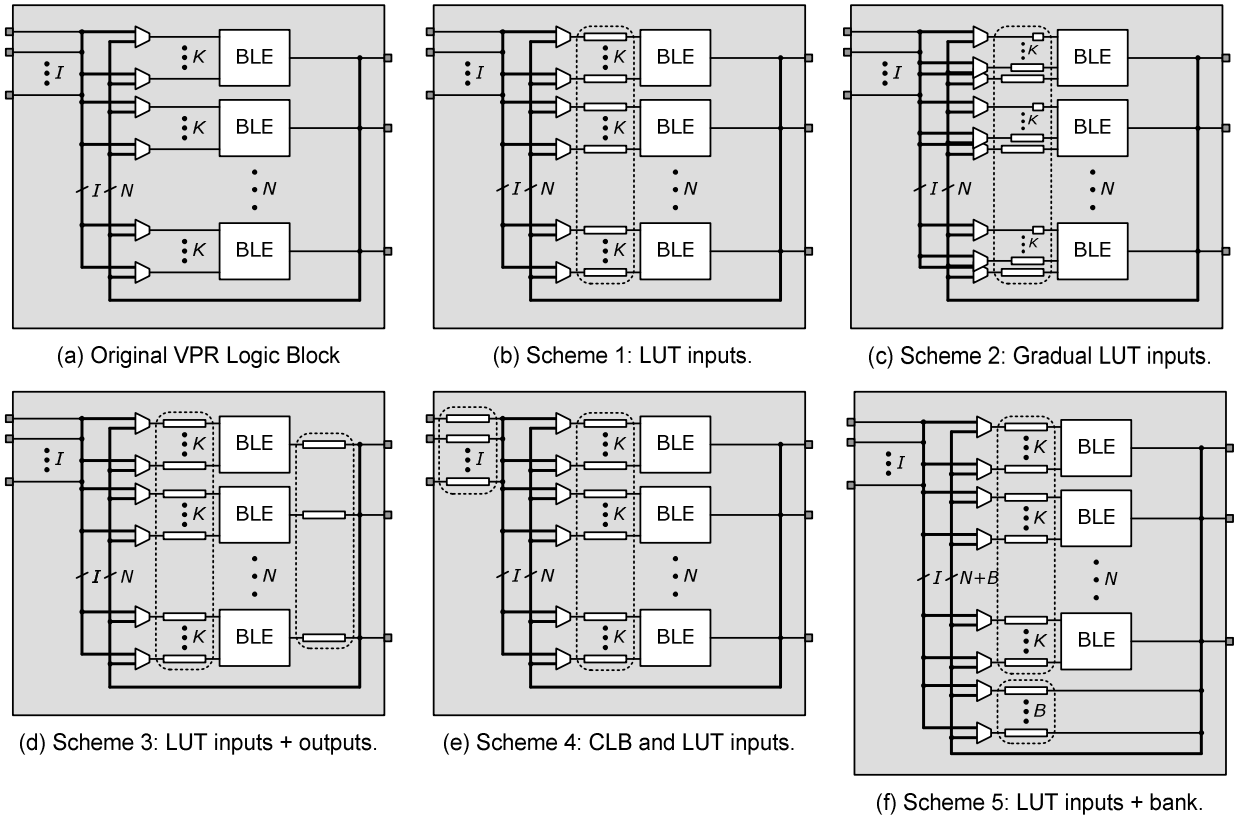


Figure 3.4. Delay insertion schemes

In Scheme 1, the programmable delay elements are added at the input of each LUT, as shown in Figure 3.4 (b). This architecture allows each LUT input to be delayed independently. We describe the architecture using three parameters: min_in , max_in , and num_in . The min_in parameter specifies the precision of the delay element connected to the LUT inputs. Intuitively, more glitching can be eliminated when min_in is small since the arrival times can be aligned more precisely. On the other hand, there is more overhead when min_in is small since each programmable delay element requires more stages to provide the extra precision. The max_in parameter specifies the maximum delay that can be added to each LUT input. Intuitively, more glitching can be eliminated when max_in is large since wider glitches can be eliminated. However, there is more overhead when max_in is large. Finally, the num_in parameter specifies how many LUT inputs have a programmable delay element, between 1 and K (the number of inputs in each LUT). Increasing num_in reduces glitching but increases the overhead. In Section 3.6, we quantify the impact of these parameters on the power, area, and delay of this scheme.

The disadvantage of Scheme 1 is that, since some inputs need very long delays for alignment, large programmable delay elements are required. Since num_in delay elements are needed for every LUT, this technique has a high area overhead if num_in is large. In Scheme 2, shown in Figure 3.4 (c), the programmable delay elements are in the same location as Scheme 1; however, the maximum delay of the elements is gradually decreased for each LUT input (by a factor of 0.5). Intuitively, the arrival times of the inputs most likely vary with one another, therefore the area overhead can be reduced by reducing the maximum delay of some of the delay elements without a significant penalty on glitch reduction. The same parameters used to describe Scheme 1 are used to describe Scheme 2, with max_in specifying the maximum delay of the largest delay element.

In Scheme 3, shown in Figure 3.4 (d), additional programmable delay elements are added to the outputs of LUTs (we refer to these new delay elements as *LUT output delay elements*). With this architecture, a single LUT output delay element could be used to delay a signal that fans out to several sinks, potentially reducing the size and the number of delay elements required at each LUT input. We describe the LUT output delay elements using two parameters, min_out and max_out , which specify the minimum and maximum delay of the output delay elements. The LUT input delay elements are described using the same parameters as Scheme 1.

Scheme 4, shown in Figure 3.4 (e), is another way to reduce the area required for the LUT input delay elements. In this scheme, additional delay elements, which we call *CLB input delay elements*, are added to each of the I CLB inputs. Since there are typically fewer CLB inputs than there are LUT inputs in a CLB, this could potentially result in an overall area savings. The parameters min_c and max_c specify the minimum and maximum delay of the CLB input delay elements. We assume every CLB input has a delay element, in order to maintain the equivalence of each CLB input.

Finally, Scheme 5, shown in Figure 3.4 (f), reduces the size of the LUT input delay elements by adding a bank of delay elements which can programmably be used by any LUT in the CLB. We refer to these delay elements as *bank delay elements*. Signals that need large delays can be delayed by the bank delay elements, while signals that need only small delays can be delayed by the LUT input delay elements. In this way, the LUT input delay elements can be smaller than they are in Scheme 1. These bank delay elements are described using two additional parameters: max_b and num_b . The max_b parameters specify the maximum delay of the bank delay elements and the num_b parameter specifies the number of programmable delay elements in the bank. Note that we assume that the minimum delay of the bank delay element is equal to the maximum delay of the LUT input delay element since only one of delay elements needs to add precision.

The parameters used to describe each scheme are summarized in Table 3.3 below. The area and delay overhead for each scheme, as well as their ability to reduce glitches, will be quantified in Section 3.6 and Section 3.7.

Table 3.3: Delay insertion parameters

Scheme	Parameter	Meaning
All	min_in	Min delay of LUT input delay element
	max_in	Max delay of LUT input delay element
	num_in	# of LUT input delay elements / LUT
3	min_out	Min delay of LUT output delay element
	max_out	Max delay of LUT output delay element
4	min_c	Min delay of CLB input delay element
	max_c	Max delay of CLB input delay element
5	max_b	Max delay of bank delay element
	num_b	# of bank delay elements / CLB

3.4.3 Programmable Delay Element

Figure 3.5 illustrates an example of the programmable delay element used in each of the proposed delay insertion schemes. The programmable delay element uses techniques similar to those described in [18], however, this programmable delay element has a longer maximum delay and evenly spaced delay increments. The circuit has multiple delay stages (5 in this example), each consisting of two transmission gates and an SRAM cell. Each stage has a fast and a slow mode, which is controlled by the value stored in that SRAM cell. In the slow mode, the signal must pass through the slow transmission gate, consisting of pass-transistors with long channel lengths (for high resistance). In the fast mode, the signal is allowed to pass through fast a transmission gate consisting of a minimum sized transistor. By approximately doubling the resistance of each successive stage, the circuit can be configured using n bits to produce one of 2^n different delay values with even increments. Specifically, the circuit can be configured to produce any delay $\Delta \in \{k, \tau + k, 2\tau + k, 3\tau + k, \dots, (2^n - 1)\tau + k\}$, where k is the delay produced by the (non-zero) bypass resistances and the inverters and τ is the minimum delay increment. As an example, τ in the figure below is the difference between delay of going through the bypass and the low-resistance path in the final (rightmost) stage and the delay of going through the bypass and the high-resistance path in the final stage. Note that the binary delay approach used in this circuit is more efficient than a straight-forward linear arrangement of equal-delay elements since it requires significantly less multiplexing to select the needed delay.

In addition to n delay stages, the programmable delay element has a 2-to-1 multiplexer and a buffer. The multiplexer is required to bypass the first $n-1$ stages when a very small delay is needed. Without this, the minimum delay of the circuit (k) would be too large. The buffer consists of two inverters with long channel lengths to minimize short-circuit (or crowbar) power.

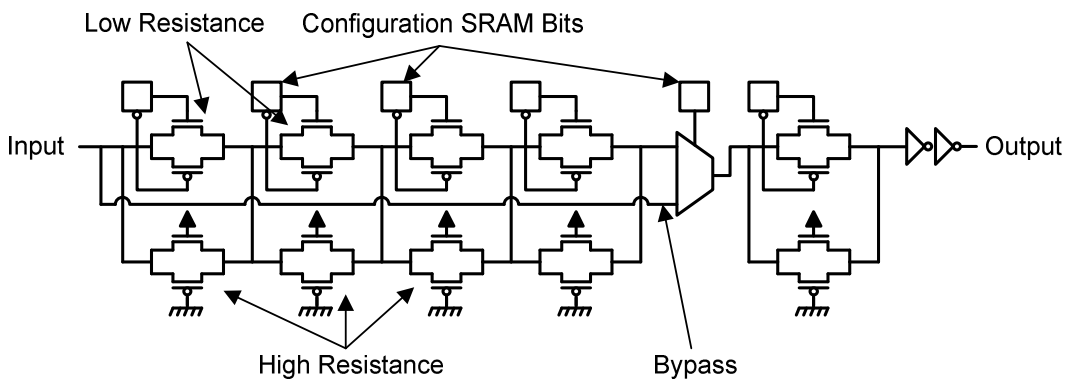


Figure 3.5. Programmable delay element.

This is the circuit we use to obtain the area, power, and delay overhead for the proposed delay insertion technique. The programmable circuit produces the required delays and careful consideration was taken to minimize the area and power dissipation of the circuit. That being said, there are likely other circuit-level techniques that can be used to align input edges and filter glitches that may be even more efficient. Our main goal is to validate the overall technique and to give a reasonable account of the tradeoffs between power savings and area/delay overhead.

3.4.4 CAD Algorithms

This section describes the algorithms used to determine the configuration of each programmable delay element. This configuration occurs after placement and routing, when accurate delay information is available.

Regardless of the architecture scheme used, a quantity *Needed_Delay* is first calculated for each LUT input. This quantity, which indicates how much delay should be added to the LUT input so that all LUT inputs transition at the same time, is calculated using the algorithm in Figure 3.6. Note that the propagation delay from the input to the output of a LUT can be different for each LUT inputs. These uneven fanin delay are accounted for when the arrival times are calculated. Specifically, the *Fanin_Delay(n,f)* value includes the delay of the fanin connection plus the delay between the input pin to the output of the node.

```

calc_needed_delays (circuit) {
  foreach node n ∈ circuit {
    // in topological order beginning from the primary inputs
    Arrival_Time(n) = 0.0;
    foreach fanin f ∈ n
      if (Arrival_Time(f) + Fanin_Delay(n, f) > Arrival_Time(n))
        Arrival_Time(n) = Arrival_Time(f) + Fanin_Delay(n, f);
  }
  foreach node n ∈ circuit {
    // in topological order beginning from the primary inputs
    foreach fanin f ∈ n
      Needed_Delay(n, f) = Arrival_Time(n) - Arrival_Time(f) -
        Fanin_Delay(n, f);
  }
}

```

Figure 3.6. Pseudo-code for calculating the delay needed to align the inputs

The next step is to implement a delay as close to *Needed_Delay* as possible for each LUT input. Since, in all but the first scheme, signals can be delayed in more than one way, there is more than one way to implement the needed delay. The technique used is different for each scheme.

The algorithm used to calculate the configuration of each LUT input delay element in Scheme 1 is shown in Figure 3.7. In this case, there is only one way to insert delays, so the algorithm is straightforward. Note that the precision of the delay elements (*min_in*) and the number of delay elements attached to each LUT (*num_in*) will affect how closely the inserted delays match the desired values (determined using the algorithm of Figure 3.6).

```

scheme1 (circuit, min_in, max_in, num_inl) {
  config_LUT_input_delays (circuit, min_in, max_in, num_in);
}
config_LUT_input_delays (circuit, min_in, max_in, num_in) {
  foreach LUT n ∈ circuit {
    count = 0;
    foreach fanin f ∈ n {
      if (Needed_Delay(n, f) > min_in && Needed_Delay(n, f) ≤ max_in && count < num_in) {
        Needed_Delay(n, f) = Needed_Delay(n, f) – min_in * floor(Needed_Delay(n, f) / min_in);
        count++;
      }
    }
  }
}

```

Figure 3.7. Pseudo-code for assigning delays in Scheme 1.

The algorithm for Scheme 2, shown in Figure 3.8, is similar to the algorithm for first scheme except that it begins by sorting the delay elements and the fanins based on delay. Both are sorted to ensure that the fanins that need small delays use the smaller delay elements, which leaves the larger delay elements to the fanins that need larger delays.

```

scheme2 (circuit, min_in, max_in, num_inl) {
  config_gradual_LUT_input_delays (circuit, min_in, max_in, num_in);
}
config_gradual_LUT_input_delays (circuit, min_in, max_in, num_in) {
  foreach LUT n ∈ circuit {
    // make list of available delay elements
    delay = max_in;
    for (i=num_in-1; i>0; i--) {
      delay_elements[i] = delay;
      delay = delay / 2.0;
    }
    sort_fanins_by_needed_delay (n); // smallest first
    foreach fanin f ∈ n {
      if (Needed_Delay(n,f) < min_in) continue;
      for (i=0; i<num_in; i++) {
        if (delay_elements[i] >= Needed_Delay(n,f)) {
          Needed_Delay(n, f) = Needed_Delay(n, f) – min_in * floor(Needed_Delay(n, f) / min_in);
          delay_elements[i] = -1.0; // mark delay element is used
        }
      }
    }
  }
}

```

Figure 3.8. Pseudo-code for assigning delays in Scheme 2.

The algorithm for Scheme 3 is shown in Figure 3.9. This algorithm first visits each LUT in topological order from the inputs to the outputs and determines the minimum delay needed by all the fanouts of that LUT. It then configures the output delay element to match this delay and then updates the needed delay value of each fanout. It then configures the input delays as in Scheme 1.

```

scheme3 (circuit, min_in, max_in, num_in, min_out, max_out) {
  config_output_delays (circuit, min_out, max_out);
  config_LUT_input_delays (circuit, min_in, max_in, num_in);
}
config_output_delays (circuit, min_out, max_out) {
  foreach LUT n ∈ circuit {
    min = max_out;
    foreach fanout f ∈ n {
      if (Needed_Delay(f, n) < min) {
        min = Needed_Delay(f, n);
      }
    }
    if (min ≥ min_out) {
      foreach fanout f ∈ n {
        Added_Delay(f, n) = min_out * floor(min / min_out);
        Needed_Delay(f, n) = Needed_Delay(f, n) - Added_Delay(f, n);
      }
    }
  }
}

```

Figure 3.9. Pseudo-code for assigning additional delays in Scheme 3.

Scheme 4, which incorporates programmable delay elements at the CLB inputs and LUT inputs, uses the algorithm described in Figure 3.10 to configure the CLB input delay elements and then uses the algorithm described in Figure 3.7 to configure the LUT input delays. The algorithm visits each CLB input and determines the minimum delay needed by the LUT inputs that are driven by that input. It then configures the CLB input delay element to match the minimum delay and updates the needed delay of the affected LUT inputs to reflect the change.

Finally, Scheme 5, which incorporates a bank of programmable delay elements in addition to those at the LUT inputs, uses the algorithm described in Figure 3.11 to configure the bank of delay elements. The algorithm visits each CLB in the circuit and configures the bank circuits to delay signals that need to be delayed by more than max_in and smaller or equal to max_b . When the algorithm finds a signal that needs a delay that is greater than max_in , it calculates the amount of delay that it can add to a signal (by a delay element in the bank) and then updates the needed delay to reflect the change for the subsequent LUT input algorithm. The count variable is used to limit the number of bank delay elements that are used for each CLB. After the bank

delay elements have been configured, the algorithm from Figure 3.7 is used to configure the LUT input delay elements.

```

scheme4 (circuit, min_in, max_in, num_in, min_clb, max_clb)
{
  config_CLB_input_delays (circuit, min_clb, max_clb);
  config_LUT_input_delays (circuit, min_in, max_in, num_in);
}
config_CLB_input_delays (circuit, min_clb, max_clb) {
  foreach CLB c ∈ circuit {
    foreach input i ∈ c {
      min = max_clb;
      foreach fanout f ∈ i {
        if (f ∈ c && Needed_Delay(f, i) < min) {
          min = Needed_Delay(f, i);
        }
      }
      if (min ≥ min_clb) {
        foreach fanout f ∈ i {
          Added_Delay(f, i) = min_clb * floor(min / min_clb);
          Needed_Delay(f, i) =
            Needed_Delay(f, i) - Added_Delay(f, i);
        }
      }
    }
  }
}

```

Figure 3.10. Pseudo-code for assigning additional delays in Scheme 4.

```

scheme5 (circuit, min_in, max_in, num_in, max_b, num_b)
{
  config_bank_delays (circuit, max_in, max_b, num_b);
  config_LUT_input_delays (circuit, min_in, max_in, num_in);
}
config_bank_delays (circuit, max_in, max_b, num_b) {
  foreach CLB c ∈ circuit {
    count = 0;
    foreach LUT n ∈ c {
      foreach fanin f ∈ n {
        /* Note: min_b == max_in */
        if (Needed_Delay(n, f) > max_in &&
            Needed_Delay(n, f) ≤ max_in + max_b &&
            count < num_b)
        {
          Added_Delay(n, f) = max_in *
            floor(Needed_Delay(n, f) / max_in);
          Needed_Delay(n, f) =
            Needed_Delay(n, f) - Added_Delay(n, f);
          count++;
        }
      }
    }
  }
}

```

Figure 3.11. Pseudo-code for assigning additional delays in Scheme 5.

3.4.5 PVT Variation Techniques

PVT variations can have a significant impact on circuit delay, which is problematic for the proposed delay insertion technique. Our technique requires accurate estimates of the path delays and the inserted delays in order to align the arrival times. If the estimates are not accurate, and the delay elements are not configured properly, they may be ineffective at reducing glitches or increase the critical-path delay. Techniques for minimizing the effect of both die-to-die and within-die PVT variation on the proposed delay insertion technique are described below.

Die-to-Die Variation

Die-to-die variation occurs when circuits on different chips have different delay properties. A common practice used by FPGA vendors to deal with variation is *speed binning*, which involves grouping a product based on the maximum speed of that product. Because of PVT variation, some FPGAs are faster than other FPGAs. Grouping the FPGAs into different speed bins allows the vendors to sell FPGAs with different speed grades. This practice tends to reduce die-to-die variation for FPGAs within each speed bin and makes the proposed technique more feasible.

Although speed-binning can help reduce the die-to-die variations, this may not be sufficient to provide the accuracy required to obtain significant power savings. Within a speed grade, we can tolerate variations if the programmable delay element is designed to react the same way as the existing FPGA logic and routing resources. As an example, consider an input signal that arrives 1ns before the slowest input under normal conditions, as illustrated in Figure 3.12 (a). In order to eliminate glitches, the corresponding programmable delay element would be configured to add 1ns to that input. Now, consider some variation that causes that same input to arrive only 0.5ns before the slowest input (see Figure 3.12 (b)). In this case, adding 1ns would be too much and possibly cause a timing violation. However, if the programmable delay element is affected the same way as the remaining circuitry, the added delay would actually be 0.5ns, producing the desired effect.

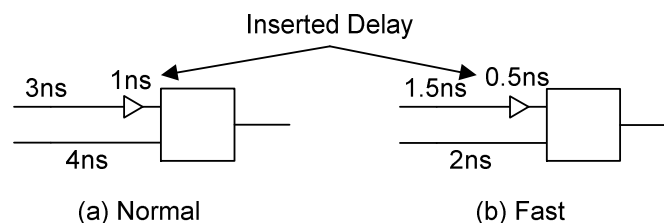


Figure 3.12: Example showing that the inserted delays need to scale with the remaining delays.

For this to be effective, PVT variation must affect the delay of the programmable delay element in the same way as the existing FPGA routing and logic circuitry. In the remainder of this section, we show that this is *not* true in the delay element presented in prior work [9] but that it is partially true in the delay element presented in Section 3.4.3.

First, consider the delay element proposed in [9]. The circuit, which is illustrated in Figure 3.13, is composed of two inverters. The first inverter has programmable pull-up and pull-down resistors to control the delay of the circuit. The second inverter has large channel lengths to minimize short-circuit power. The pull-up and pull-down resistors of the first inverter have n stages. Each stage has a resistor and a bypass transistor controlled by an SRAM bit. The resistor in each stage consists of a pass-transistor that is only partially turned on (though biasing) to produce a large resistance.

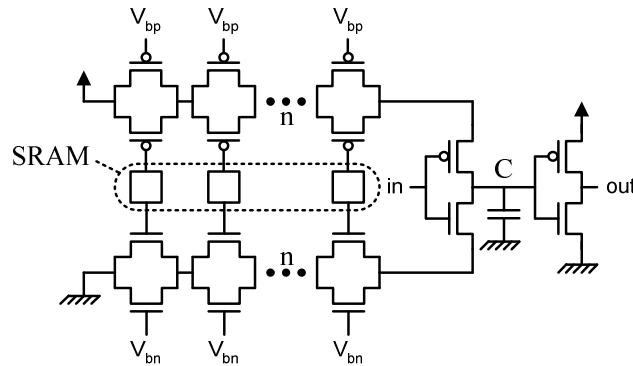


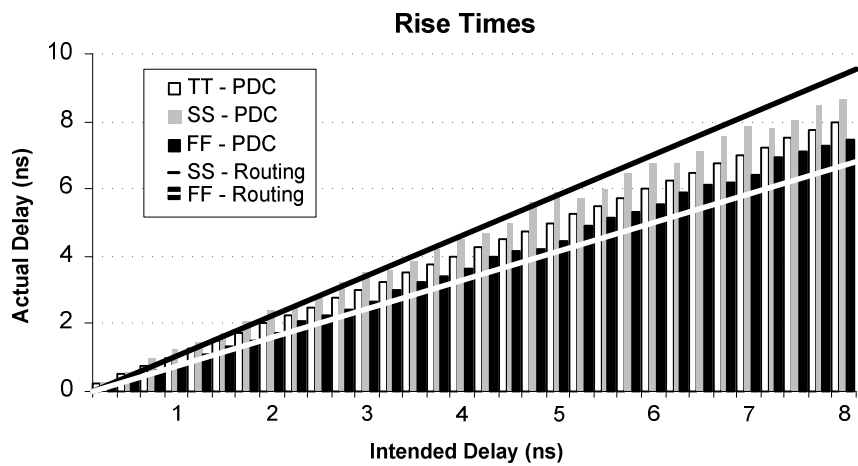
Figure 3.13: Schematic of the previous programmable delay element [9].

The circuit does not tolerate variation for two reasons. First, the circuit uses gate biasing to produce a large resistance which tends to react differently to variation compared to the remaining FPGA circuitry. Second, the rise and fall times of the circuit become unbalanced when there is variation because the NMOS and PMOS transistors react differently to variation. This is less of a concern in conventional buffers and logic gates which also use PMOS pull-up networks and NMOS pull-down networks, since the effect is averaged out when the inverters or logic gates are cascaded.

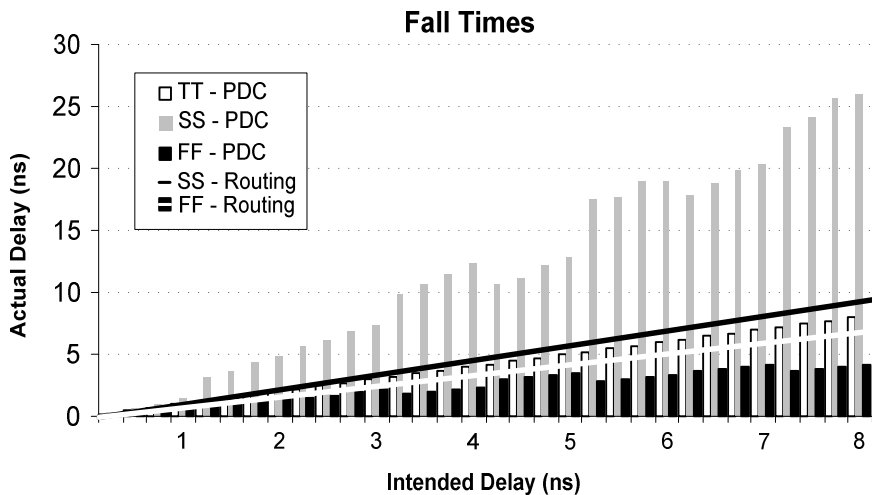
To illustrate these effects, Figure 3.14 shows the rise and fall times of the programmable delay element for every possible delay configuration. For the black, white, and grey bars, the X-axis represents intended delay and the Y-axis represents actual delay. Results from three experiments are shown. The white bars are the delays of the programmable delay element simulated in HSPICE assuming typical-typical (TT) process parameters. Similarly, the grey and

black bars are the delays assuming slow-slow (SS) and fast-fast (FF) process parameters, respectively.

In addition to the programmable circuit delays, the graphs also include lines that show the effect of process variation on the delay of the existing FPGA routing resources. For the black and white lines, the x-axis represents the delay of the existing FPGA routing resources when typical-typical (TT) process parameters are assumed and the y-axis represents the delay of the same resources when other process parameters are assumed. Specifically, the black line (SS-Routing) indicates the delay of the FPGA routing assuming SS process parameters and the white line (FF-Routing) indicates the delay of the FPGA routing assuming FF parameters.



(a) Rise times



(b) Fall times

Figure 3.14: Rise and fall times of delay element from [9] considering process variation

The two graphs highlight the drawbacks described above. As shown in the first graph, the rise times are not as affected by process variations as the FPGA routing circuitry is. As shown in the second graph, however, the fall times are significantly affected by process variations. On average, the fall time assuming the FF process corner is 47% faster than TT values, while the average fall time assuming the SS process corner is 137% slower.

Now consider the programmable delay element described in this chapter (in Section 3.4.3). In this circuit, NMOS and PMOS transistors were used in parallel in order to average out their response to variation. The rise times of the new delay circuit are shown in Figure 3.15. Similar results were obtained for the fall time. On average, the actual delays are 19% faster and 26% slower for the FF and SS process corners, respectively. The response of the new delay circuit still varies more than the response of the FPGA routing resources since interconnect does not vary as much as devices; however, the new delay circuit responds significantly better than the previous delay circuit and is quite suitable for this application.

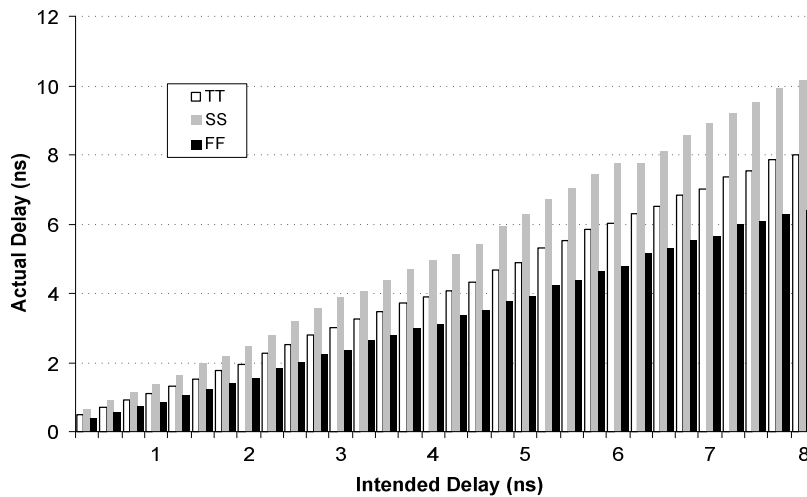


Figure 3.15: Rise times of the programmable delay element considering process variation
Within-Die Variation

In the case of within-die variation, speed binning and proportional scaling may not be sufficient. Since the inputs of a LUT can come from any part of the chip, within-die variation can affect the delay of one input differently from another input. Although most connections are local (since the FPGA clustering, placement, and routing tools minimize the routing distance between connections), within-die variation is still a problem for large nets that span the entire chip.

A naïve solution for within-die variation is to reconfigure the programmable delay elements of each FPGA individually based on the PVT variation for that FPGA. This solution,

however, is impractical since it is difficult to obtain PVT variation information for individual FPGAs and it would be time consuming to reconfigure each FPGA with different delays.

Another, more practical solution, is to pessimistically reduce the delay added by each programmable delay element. We first determine, D , the delay that would be inserted by the programmable delay element using static timing analysis assuming no PVT variation. Then, if the nature of the expected variations are known, we can estimate the approximate worst-case impact of the variation, d . We then configure the programmable logic element to insert the delay $D-d$. This ensures that the delay inserted by the delay element does not increase the overall delay of the circuit. However, it also means that the actual delay that is inserted may be shorter than the delay that is needed to eliminate the glitch. This will impact the glitch power eliminated; however, even in cases where the glitch is not eliminated, the width of the glitch will be shorter than it would otherwise be. These shorter pulses are then more likely to be filtered out by other delay elements that are downstream.

Note that a more complete approach to this technique would involve using statistical timing analysis to determine the maximum delays that can safely be added without increasing the critical path delay. However, statistical timing analysis is not supported within our current experimental framework. Nonetheless, the results for this static approach, which are presented in Section 3.7.5, still serve to demonstrate the tradeoff between the power savings and the uncertainty introduced by PVT variation.

3.5 Experimental Framework

This section describes the experimental framework that is used to obtain the switching activity information and the FPGA area, delay, and power estimates that are presented in Section 3.3, 3.6 and 3.7.

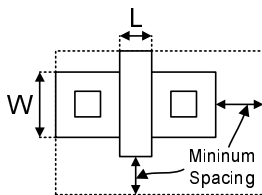
3.5.1 Switching Activity Estimation

The switching activities are obtained by simulating circuits at the gate level and counting the toggles of each wire. The simulations are driven by pseudo-random input vectors and circuit delay information from the VPR place and route tool [19]. To capture the filtering effect of the FPGA routing resources and of the programmable delay elements, the simulator uses the *inertial delay* model. Furthermore, to replicate an FPGA routing architecture consisting of length 4 routing segments, the VPR delays are divided into chains of 300 ps delays which is the approximate delay of one length 4 routing segment..

3.5.2 Area, Delay, and Power Estimation

Area, delay, and power estimates are obtained from the VPR place and route tool and HSPICE simulations. VPR is used to model the existing FPGA circuitry and HSPICE is used to model the added delay element circuitry.

The VPR models are detailed, taking into account specific switch patterns, wire lengths, and transistor sizes. After generating a specified FPGA architecture, VPR places and routes a circuit on the FPGA and then models the area, delay, and power of that circuit. VPR models area by summing the area of every transistor in the FPGA, including the routing, CLBs, clock network, and configuration memory. The area of each transistor is approximated using the Minimum Transistor Equivalents (MTE) metric from [19], which calculates the layout area occupied by a minimum sized transistor plus the minimum spacing as illustrated in Figure 3.16. The model was augmented slightly in this chapter to consider transistors with longer than minimum channel length. Expression (3.1) models the layout area of a transistor with respect to its channel width (W) and Expression (3.2) models the area with respect to its length (L). The models were derived by observing the relative area increase when either W or L is increased. The expressions differ slightly since the minimum width of a transistor accounts for approximately one half of the y-component of the layout area, whereas the minimum length accounts for approximately one fifth of the x-component of the layout area.



$$Area(W) = \frac{1}{2} + \frac{W}{2 \cdot W_{min}} \quad (3.1)$$

$$Area(L) = \frac{4}{5} + \frac{L}{5 \cdot L_{min}} \quad (3.2)$$

Figure 3.16: Minimum Transistor Equivalents (MTE) area model [19].

The delay and power are modeled after routing occurs, when detailed resistance and capacitance information can be extracted for each net in the benchmark circuit. The Elmore delay model is used to produce delay estimates and the FPGA power model described in [20] is used to produce power estimates. The power model uses the VPR capacitance information and externally generated switching activities to estimate dynamic, short-circuit, and leakage power. Note, however, that the leakage power estimates for both the existing FPGA circuitry and the programmable delay elements do not account for PVT variation (typical process, voltage, and temperature are assumed).

3.5.3 Architecture Assumptions and Benchmark Circuits

We gathered results for three LUT sizes: 4, 5, and 6 inputs. In all cases, we assumed that each CLB contains 10 LUTs and that the CLBs have 22, 27, and 33 inputs for architectures with 4, 5, and 6 input LUTs, respectively; these values are similar those used in current commercial devices [20][21]. In each case, we assume that the crossbar that programmably connects the CLB inputs and LUT outputs to the LUT inputs with each CLB is fully populated as described in [19]. Furthermore, for routing, we assumed two segmented routing fabrics, one consisting of buffered length 1 and another of length 4 routing segments and a channel width that is 20% wider than the minimum channel width (a separate value was found for each benchmark). Since the results were similar for both segment lengths, only the length 4 results are presented in Section 3.6 and 3.7 unless stated otherwise.

In each experiment, we used 20 combinational benchmarks including the 10 largest combinational circuits from the MCNC and ISCAS89 benchmark suites. Before placement and routing, each circuit is mapped to LUTs using the Emap technology mapper [7] and packed into clusters using the T-VPack clusterer [19].

3.6 Scheme Calibration

Before we examine the overall power savings and area and delay overhead of the delay insertion technique, we need to find suitable values for the parameters of each scheme (listed in Table 3.3). In each case, the value is chosen so as to eliminate as much of the glitching as possible, while minimizing the area and delay overhead.

3.6.1 Scheme 1 Calibration

We first consider the *min_in* parameter, which defines the minimum delay increment of the programmable delay element at the inputs of the LUTs. Intuitively, a smaller delay increment reduces glitching but increases area. Figure 3.17 shows how much glitching is eliminated for minimum delay increments ranging between 0.1 and 3.2ns. To isolate the impact of the *min_in* parameter, the graph assumes that every LUT input has a programmable delay element with an infinite maximum delay (*max_in* is ∞ and *num_in* is K).

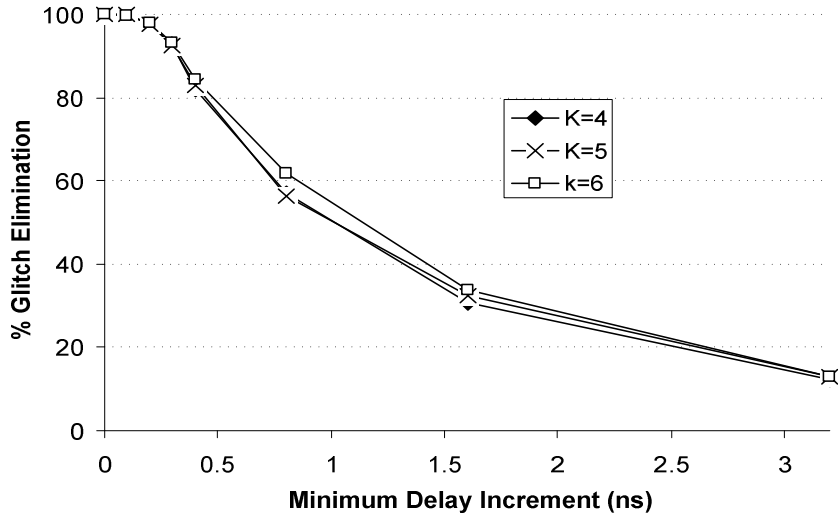


Figure 3.17. Glitch elimination vs. minimum LUT input delay for Scheme 1.

The graph illustrates that most of the glitching can still be eliminated when the minimum delay increment is 0.25ns. This corresponds to the fact that narrow glitches are filtered away by the routing resources and that the majority of glitches have a width greater than 0.2ns, as described in Section 3.3. The same conclusion holds for FPGAs that use 4, 5, or 6 input LUTs.

The second parameter, denoted *max_in*, defines the maximum delay of the programmable delay element at the inputs of the LUTs. Intuitively, increasing the maximum delay reduces glitching but increases area. Figure 3.18 shows how much glitching is eliminated as a function of the maximum delay. The graph illustrates that over 90% of the glitching can be eliminated when the maximum delay of the programmable delay element is 8ns. This corresponds with Figure 3.1, which illustrates that the majority of glitches have a width that is less than 10ns.

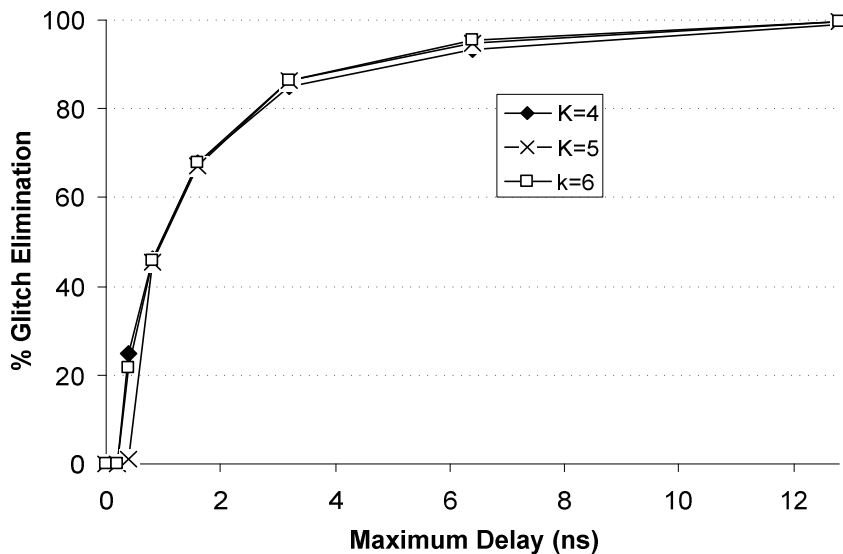


Figure 3.18. Glitch elimination vs maximum LUT input delay for Scheme 1.

Finally, num_in defines the number of LUT inputs that have a programmable delay element. Intuitively, increasing the number of inputs with delay elements reduces glitching since the arrival times of more inputs can be aligned. Figure 3.19 shows how much glitching is eliminated when the number of inputs with programmable delays is varied. The graph assumes that the minimum delay increment is $1/\infty$ and the maximum delay is ∞ .

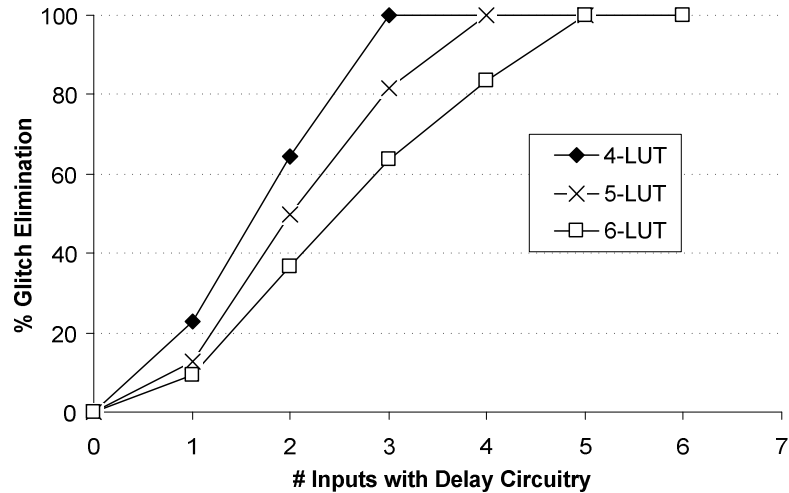


Figure 3.19. Glitch elimination vs. number of input delay elements per LUT for Scheme 1.

The graph illustrates that each LUT should have a programmable delay element on every input minus one ($K-1$). Intuitively, adding delay circuitry to every input is not necessary since each LUT has at least one input that does not need to be delayed (the slowest input). However, adding fewer than $K-1$ delay elements significantly reduces the amount of glitching that can be eliminated.

3.6.2 Scheme 2 Calibration

Scheme 2 has the same three parameters as Scheme 1 and the same values are used for each parameter. Specifically, num_in is $K-1$, min_in is 0.25ns, and max_in is 8ns. However, to minimize overhead, the maximum delay of the LUT input delay elements (max_in) is gradually decreased by half (or by 1 delay stage) per LUT input. As an example, the maximum delay values for a 4-input LUT would be 8ns, 4ns, and 2ns.

3.6.3 Scheme 3 Calibration

Scheme 3 has five parameters, namely: min_in , max_in , num_in , min_out , and max_out . The first three parameters control the delay elements at the inputs of the LUTs; the last two parameters control the delay elements at the output of the LUTs. Although the min_in , max_in ,

and num_in parameters where already calibrated for Scheme 1, they must be recalibrated for Scheme 3 since the output delay elements change how much delay is needed by LUT input delay elements. Intuitively, however, the value of the min_in parameter can be reused since the LUT input delays are still used to perform the final alignment of each signal.

The max_in and num_in are both recalibrated assuming min_out is infinitely precise ($1/\infty$) and max_out is ∞ . Figure 3.20 shows the glitch elimination for max_in from 0 to 12ns assuming again that min_in is $1/\infty$ and num_in is K . The results are similar to those in Scheme 1 except that some glitching is eliminated when max_in is 0 since the output delay elements are aligning some of the inputs. Again, most of the glitching can be eliminated when max_in is set to 8ns.

Figure 3.21 shows glitch elimination with respect to num_in . As before, the graph assumes that min_in is $1/\infty$ and max_in is ∞ . The graph shows that more glitching is eliminated using fewer LUT input delay elements when the output delays are used. In Scheme 2, most of the glitching can be eliminated when num_in is $K-2$.

The remaining output delay element parameters are calibrated assuming min_in is 0.25ns, max_in is 8.0ns, and num_in is $K-2$. Figure 3.22 shows the glitch elimination for min_out from 0 to 3.2ns assuming that max_out is ∞ and Figure 3.23 shows the glitch elimination for max_out from 0 to 12ns assuming that min_out is $1/\infty$. The graphs illustrate that a 0.25ns and 8ns are also suitable for min_out and max_out , respectively.

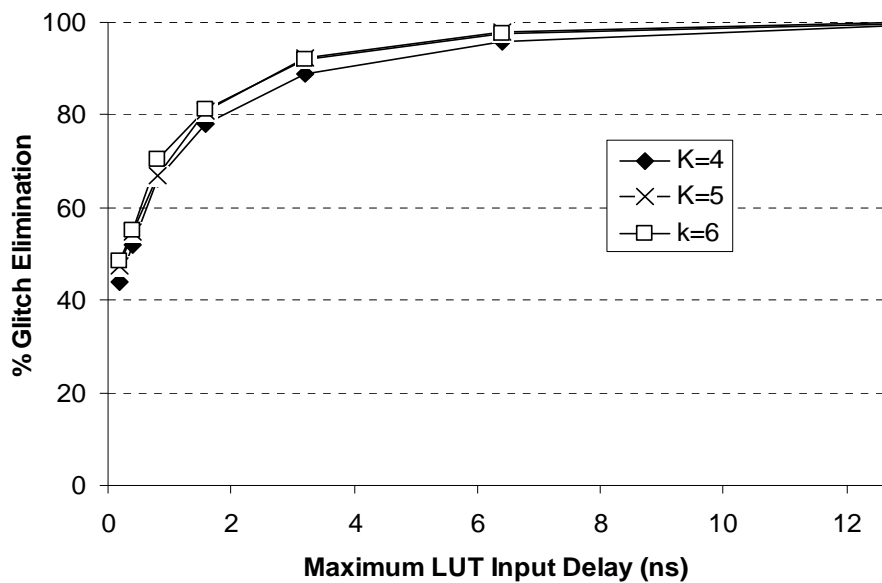


Figure 3.20. Glitch elimination vs. maximum LUT input delay for Scheme 3.

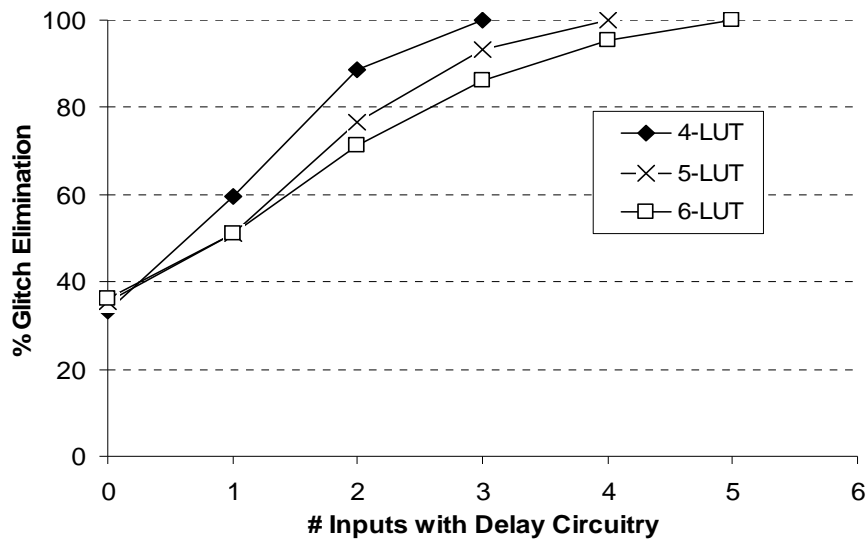


Figure 3.21. Glitch elimination vs. number of input delay elements per LUT for Scheme 3.

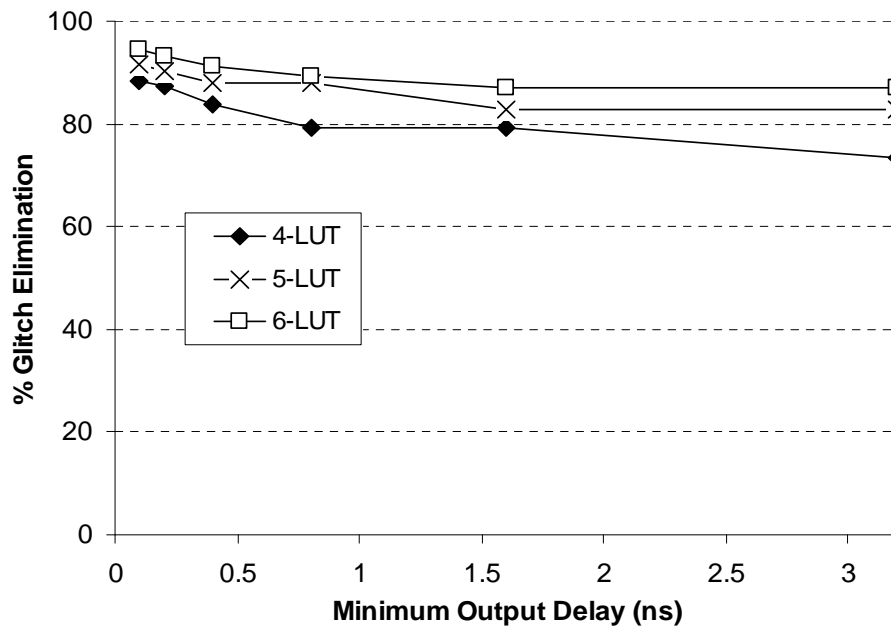


Figure 3.22: Glitch elimination vs. minimum LUT output delay for Scheme 3.

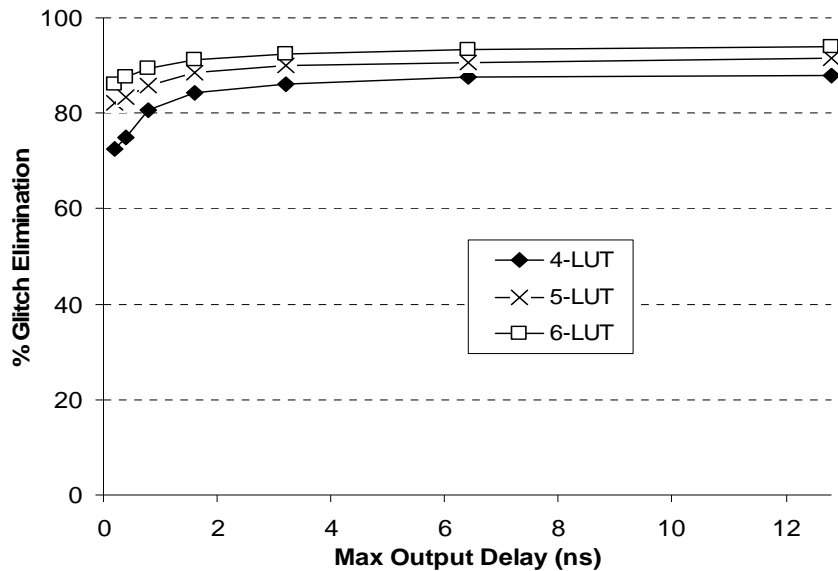


Figure 3.23: Glitch elimination vs. maximum LUT output delay for Scheme 3.

3.6.4 Scheme 4 Calibration

Scheme 4 has five parameters, namely: min_in , max_in , num_in , min_c , and max_c . The first three parameters control the delay elements at the inputs of the LUTs; the last two parameters control the delay elements at the input of the CLBs. The min_in , max_in , and num_in parameters were again recalibrated to account for the affect of the CLB input delay elements. The same procedure used in Scheme 2 was used. The results for min_in and max_in were similar to the previous cases, which indicated that 0.25ns and 8ns, respectively, were suitable.

The results for num_in , which are plotted in Figure 3.24, were different than in the previous cases. To isolate the impact of num_in , the graph assumes that min_in is $1/\infty$, max_in is ∞ , min_c is $1/\infty$, and max_c is ∞ . The results indicate that num_in should be 1, 2, and 2, for 4, 5, and 6-LUTs, respectively. Intuitively, fewer LUT input delay elements are needed since the CLB input delay elements account for most of the delay. Only in cases where the CLB inputs fanout to multiple LUTs within that CLB and those fanouts need different delays are the LUT input delay elements required.

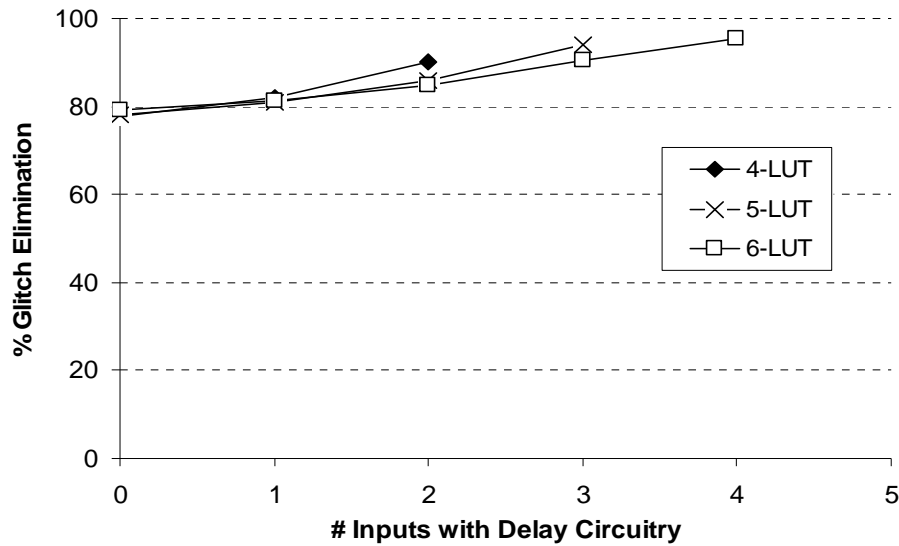


Figure 3.24: Glitch elimination vs. number of input delay elements per LUT for Scheme 4.

3.6.5 Scheme 5 Calibration

Finally, Scheme 5 has five parameters, namely: min_in , max_in , num_in , max_b , and num_b . The first three parameters control the delay elements and the inputs of the LUTs; the last two parameters control the bank of delay elements in the CLB. The bank of programmable delay elements are only used for signals that need more delay than can be added by the LUT input delay elements, therefore this scheme uses the same min_in and num_in values as Scheme 1: 0.25ns and K-1, respectively. Suitable values for max_in and max_b were found empirically to be 4.0ns and 8.0ns, respectively. Finally, Figure 3.25 shows glitch elimination with respect to the number of bank delay elements per CLB (num_b) assuming min_in is 0.25ns, num_in is K-1, max_in is 4ns, and max_b is 8ns. The results show that 4 is a suitable value for num_b for CLBs with 10 LUTs. Note that a larger value for num_b is not chosen because of the large area penalty (incurred within the local CLB interconnect) that is associated with added more bank delay elements.

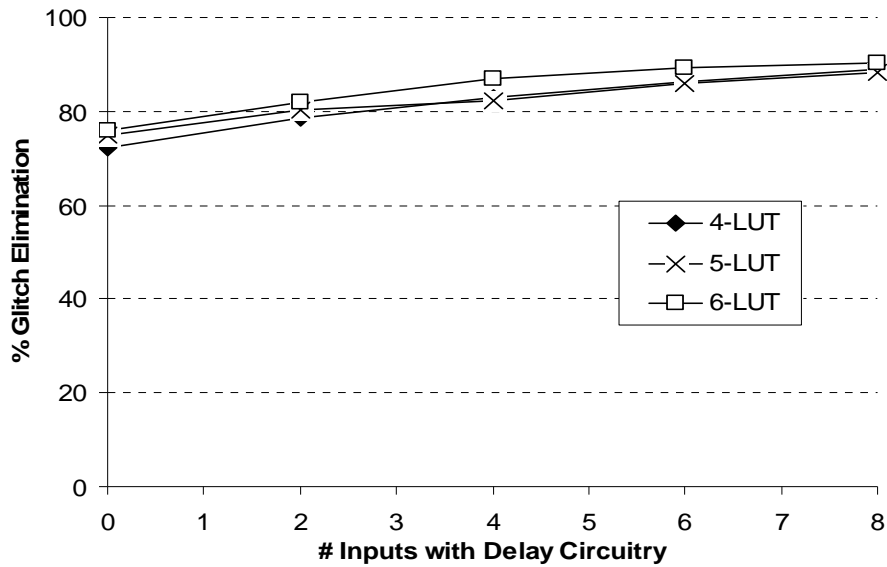


Figure 3.25: Glitch elimination vs. number of bank delay elements for Scheme 5.

3.6.6 Summary

Table 3.4 summarizes the values that were selected for each of the five delay insertion schemes. The first two columns specify the scheme number and the programmable delay element location. The third and fourth columns specify the minimum delay increment and the maximum delay of the programmable delay element at that location. The fifth column specifies the corresponding number of delay stages needed to implement the programmable delay element. Finally, the sixth column specifies the number of programmable delay elements needed per LUT (rows 2, 3, 5, and 7) and per CLB (rows 4, 6, and 8).

Table 3.4: Summary of programmable delay element values.

Scheme	Location	Delay Increment (ns)	Max. Delay (ns)	# Stages	# Circuits
1	LUT Inputs	0.25	8	5	K-1
2	LUT Inputs	0.25	8, 4, 2, ...	5, 4, 3, ...	K-1
3	LUT Inputs	0.25	8	5	K-2
	CLB Outputs	0.25	8	5	N
4	LUT Inputs	0.25	8	5	1, 2, 2
	CLB Inputs	0.25	8	5	$K(N+1) / 2$
5	LUT Inputs	0.25	4	4	K-1
	Bank	4.0	8	1	4 (N=10)

3.7 Results

This section presents the overall results. It begins by presenting the area, delay, and power overhead of each delay insertion scheme. It then presents the overall power savings assuming there is no PVT variation. Finally, it presents the overall power savings assuming there is PVT variation.

3.7.1 Area Overhead

The area overhead is determined by summing the area of the added delay circuitry in each CLB. This area includes the area of the delay elements and the added configuration memory. Table 3.5 reports how much area is needed in the CLBs and Table 3.6 reports the percent area overhead, taking the CLB and routing area into account. More precisely, the percent area overhead is calculated by dividing the total area occupied by the added programmable delay circuitry by the total area occupied by the FPGA logic and routing resources, which we determined using VPR.

Table 3.5: CLB area overhead (excluding global interconnect).

LUT Size	Original CLB Area (MTE)	CLB Area Overhead (MTE)				
		Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	6938	2460	2020	2460	2568	3184
5	10361	3280	2430	3280	3368	3808
6	15228	4100	2720	4100	4282	4494

Table 3.6: Overall area overhead.

LUT Size	Overall Area Overhead (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	8.0	6.6	8.0	8.4	10.4
5	7.6	5.3	7.6	7.8	8.8
6	6.7	4.4	6.7	7.0	7.3

The tables show that Scheme 2 has the lowest area overhead, followed by Schemes 1, 3, and 4, and finally Scheme 5 has the highest overhead. Scheme 5 requires the most area because of the large multiplexers needed to select which CLB input or LUT output uses the bank delay elements. Schemes 1, 3, and 4 have a similar area overhead since they use the same size delay elements and roughly the same number of them. Scheme 2 has the lowest area overhead since it uses smaller delay elements. The tables also show the area overhead decreases as the LUT size

increases. This occurs since the area of the LUTs and multiplexers increases exponentially with K , while the area of the delay elements only increases linearly.

3.7.2 Power Overhead

Even if all the glitches could be eliminated, the programmable delay elements still dissipate power. This overhead is modeled by summing the power dissipated by the added circuitry in each CLB of the FPGA using the expression below.

$$P_{overhead} = \frac{\sum_{n \in dnodes} E_{toggle} \cdot \alpha(n)}{T_{crit}} + P_{static} \quad (3.3)$$

In the expression, $dnodes$ is the set of nodes in the circuit that are connected to a programmable delay element, E_{toggle} is the energy dissipated by one programmable delay element during one transition, $\alpha(n)$ is the switching activity of the delayed node n , and T_{crit} is the critical path delay of the circuit. The energy and leakage power of the programmable delay element are determined using HSPICE, the switching activity is determined using gate-level simulation, and the critical-path delay is determined using the VPR place and route tool. Note that the leakage power estimates assume typical process, voltage, and temperature conditions.

Table 3.7 reports the average overhead power (as a percentage) dissipated by the added delay circuitry for each scheme. The power of the remaining FPGA circuitry is calculated using the power model described in [21].

Table 3.7: Average power overhead (%)

LUT Size	$P_{overhead} / (P_{overhead} + P_{FPGA}) * 100$				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	0.94	0.79	1.02	1.16	0.97
5	0.97	0.84	1.12	1.28	0.99
6	1.02	0.94	1.14	1.10	0.93

The table shows that the power overhead is approximately 1% for all the schemes and that Scheme 2 has the lowest power overhead.

3.7.3 Delay Overhead

Although the delay elements are programmed to only add delay to early arriving edges, a small delay penalty may be incurred even if the delay element is bypassed because of parasitic resistance and capacitance. To model delay overhead, HSPICE is used to determine the parasitic

delay incurred by the delay element. The critical-path delay of each circuit is then recalculated, taking these parasitic delays into account. Finally, the overhead is calculated by comparing the new critical-path delay to the original critical-path delay.

Table 3.8 reports the average delay overhead for each scheme. Schemes 1, 2, and 4 have the smallest overhead since both have *fast-paths* with no delay elements (no parasitics) to slow down the critical-path. There is still a slight overhead because some circuits have LUTs with more than one critical (or nearly critical) path input. When this occurs, the parasitics slows down one of these nearly critical paths. Schemes 3 and 4 have a larger overhead, since neither scheme offers a *fast-path* for critical-path connections. Specifically, the parasitic capacitance of the programmable delay elements at the output of the CLBs for Scheme 3 and at the inputs of the CLBs for Scheme 4 imposes a small delay on any signal that bypasses them (see Figure 3.5).

Table 3.8: Average delay overhead.

LUT Size	Average Delay Overhead (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	0.21	0.19	2.4	2.3	0.21
5	0.13	0.14	2.2	2.1	0.13
6	0.14	0.15	2.1	1.9	0.14

3.7.4 Overall Power Savings (Assuming No Variation)

Table 3.9 presents the average glitch elimination for each scheme:

Table 3.9: % Glitch elimination of each scheme.

Scheme1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
91.8%	87.3%	83.3%	81.8%	85.4%

Table 3.10

presents the corresponding overall power savings. Both tables indicate that Scheme 1 produces the best results, with 91.8% glitch elimination and overall power savings of 18.2%. The power savings are close to the ideal savings of 22.6%. Note also that the results in both tables are for FPGAs with 4-input LUTs and length 4 routing segments; the results for 5 and 6-input LUTs and for FPGAs with length 1 routing segments were similar. As an example, using Scheme 1 for FPGAs the 6-input LUTs and length 1 routing segments reduced glitching by 92.9% and the overall power by 16.8%. The power savings for larger LUTs are slightly smaller because there tends to be less glitching to begin with since the netlists have fewer levels of logic.

Table 3.9: % Glitch elimination of each scheme.

Scheme1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
91.8%	87.3%	83.3%	81.8%	85.4%

Table 3.10: Overall power savings.

Circuit	Power Saving (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
C135	25.4	25.4	25.0	25.0	25.8
C1908	18.1	17.5	18.4	16.1	17.0
C2670	11.6	11.4	11.3	10.2	11.7
C3540	27.5	25.4	22.9	23.5	26.3
C432	13.0	11.0	10.7	10.6	10.6
C499	31.8	31.8	30.9	32.3	32.4
C5315	18.2	16.8	16.2	16.0	17.9
C6288	52.1	41.3	43.2	40.0	46.1
C7552	22.6	21.0	18.9	19.7	22.3
C880	7.2	6.5	6.5	8.0	7.1
alu4	2.5	2.5	2.4	3.3	2.7
apex2	3.6	3.6	3.2	3.8	3.6
apex4	9.5	9.5	9.1	9.4	9.3
des	15.1	14.9	12.1	14.2	14.4
ex1010	16.8	16.8	16.4	16.5	15.9
ex5p	23.8	23.3	23.4	21.5	25.0
misex3	7.6	7.6	7.3	7.3	7.2
pdc	11.1	10.8	10.1	10.7	11.3
seq	5.3	5.2	5.9	5.7	5.6
spla	20.3	20.1	19.8	20.0	20.2
Average	18.2	16.8	16.3	16.2	17.4

3.7.5 Overall Power Savings (Assuming Variation)

The results presented in the previous sections assumed no PVT variation. The following results present the overall power saving when the technique described in Section 3.4.5 is applied to cope with the timing uncertainty introduced by PVT variation. Specifically, we repeated the experiments from Section 3.7.4, using the same delay element parameter values as before, but we reduced the delay inserted by each delay element by a factor β . We varied β from 0.7 (meaning each delay element is programmed to provide a delay of 70% of the value predicted assuming no process variations) to 1.0 (which is the same as the results in Section 3.7.4). Figure 3.26 shows the results. In this figure, β is shown on the X-axis. The lower line indicates the amount of glitching removed compared to the case when programmable delay elements are not used. As the results show, when β is 0.7, the glitch savings are reduced to 56% (compared to

91% when process variations are not considered). The upper line shows the resulting decrease in power; as expected, the power reduction is proportional to the number of glitches removed. Overall, these results indicate that the delay insertion technique still works when the added delays are reduced, but with diminished glitch and power savings as the timing uncertainty increases.

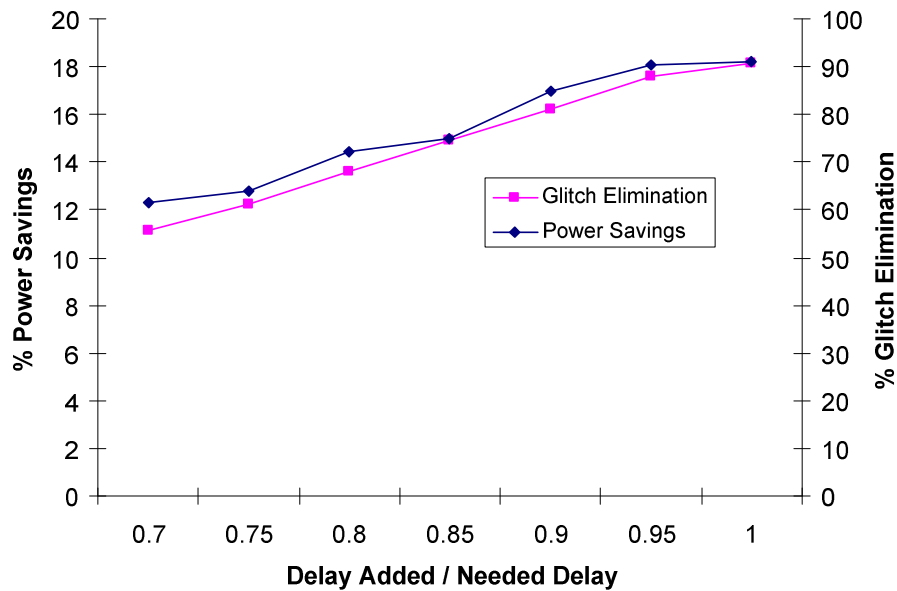


Figure 3.26: Glitch elimination and power savings vs. β .

3.8 Conclusions and Future Work

This chapter proposed a glitch elimination technique to minimize dynamic power in FPGAs. The technique involves adding programmable delay elements within the CLBs of the FPGA to align the edges on each LUT input and filter out existing glitches, thereby reducing the number of glitches on the output of each LUT. Five alternative schemes were considered for implementing this technique. Scheme 1, which uses programmable delay elements on $K-1$ inputs of each LUT, produced the greatest power savings, reducing power by 18.2%. However, Scheme 2, which uses $K-1$ delay elements that gradually decrease in size, produced similar power savings but has a lowest area overhead. On average, Scheme 2 of the proposed technique eliminates 87% of the glitching, which reduces overall FPGA power by 16.8%. The added circuitry increases overall area by 6.6% and critical-path delay by less than 1%.

There are a number of interesting issues that were not fully explored in this chapter that merit further research. First, a more complete approach to the proposed delay insertion technique would involve using statistical timing analysis to determine the maximum delays that

can safely be added without increasing the critical-path delay. Second, research into the feasibility of the proposed technique in newer process technologies which tend to dissipate more leakage power is also needed. Finally, further research of circuit-level techniques that can be used to align the arrival times may yield lower overhead, increased power savings, and/or improved PVT tolerance. As an example, a self-calibrating delay element that tunes itself to the latest arriving transition of a LUT (relative to the clock) would be ideal since it would be more tolerant to variation. Furthermore, this delay element could be used to gate all the early arriving inputs or suppress output transitions until the last input arrives. This technique may reduce area since it requires only one delay element per LUT.

Chapter 3 References

- [1] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, A 90nm low-power FPGA for battery-powered applications, Proc. ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 3-11, 2006.
- [2] C.T. Chow, L.S.M Tsui, P.H.W. Leong, W. Luk, and S.J.E. Wilton, Dynamic voltage scaling for commercial FPGAs, Proc. IEEE Intl. Conf. on Field-Programmable Technology (FPT), pp. 173-180, 2005.
- [3] A. Kumar and M. Anis, "Dual-Vt FPGA design for subthreshold leakage tolerance", Proc. Intl. Symp. on Quality Electronic Design (ISQED), pp. 735-740, 2006.
- [4] R.R. Rao, D. Blauw, D. Sylvester, C.J. Alpert, and S. Nassif, An efficient surface-based low-power buffer insertion algorithm, Proc. ACM Intl. Symp. on Physical Design, pp. 86-93, 2005.
- [5] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and T. Tuan, A dual-Vdd low power FPGA architecture, Proc. Intl. Conf. on Field-Programmable Logic and Applications (FPL), pp. 145-157, 2004.
- [6] J. Lamoureux and S.J.E. Wilton, FPGA clock network architecture: flexibility vs. area and power, Proc. ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 101-108, 2006.
- [7] J. Lamoureux and S.J.E. Wilton, On the interaction between power-aware computer-aided design algorithms for field-programmable gate arrays, Journal of Low Power Electronics (JOLPE), Vol. 1, No. 2, pp. 119-132, 2005.
- [8] J.H. Anderson, Power optimization and prediction techniques for FPGAs, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Toronto, 2005.
- [9] J. Lamoureux, G.G. Lemieux, and S.J.E. Wilton, Glitchless: an active glitch minimization technique for FPGAs, Proc. ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 156-165, 2007.
- [10] J. C. Monteiro and A. L. Oliveira, Finite state machine decomposition for low power, Proc. Design Automation Conference (DAC), pp. 758-763, 1998.
- [11] D. Kim and K. Choi, Power conscious high-level synthesis using loop folding, Proc. Design Automation Conference (DAC), pp. 441-445, 1997.
- [12] M. Kandemir et al, Influence of compiler optimizations on system power, IEEE Trans. VLSI Systems, Vol. 9, No. 6, pp. 801-804, 2001.
- [13] D. Chen, J. Cong, F. Li, and L. He, Low-power technology mapping for FPGA architectures with dual supply voltages, Proc. Intl. Symp. on Field-Programmable Gate Arrays (FPGA), pp. 109-117, 2004.
- [14] J.C. Monteiro, S. Devadas and A. Ghosh, Retiming sequential circuits for low power, Proc. Design Automation Conference (DAC), pp. 398-402, 1993.
- [15] S. Wilton, S.-S. Ang and W. Luk, , The impact of pipelining on energy per operation in field-programmable gate arrays, Proc. Intl. Conf. on Field-Programmable Logic and its Applications (FPL), pp. 719-728, 2004.
- [16] L. Benini et al, Glitch power minimization by selective gate freezing, IEEE Trans. VLSI Systems, Vol. 8, No. 3, pp. 287-298, 2000.

- [17] A. Raghunathan, S. Dey and N. K. Jha, Register transfer level power optimization with emphasis on glitch analysis and reduction, *IEEE Trans. CAD*, Vol. 18, No. 8, pp. 1114-1131, 1999.
- [18] M. Maymandi-Nejad and M. Sachdev, A digitally programmable delay element: design and analysis, *IEEE Trans. on VLSI*, Vol. 11, No. 5, pp. 871-878, 2003.
- [19] V. Betz., J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, Kluwer Academic Publishers, 1999.
- [20] Altera, *Stratix III device handbook 1*, November 2006.
- [21] Xilinx, *Virtex-5 user guide*, February 2007.
- [22] K.K.W. Poon, S.J.E. Wilton, A. Yan, A detailed power model for field-programmable gate arrays, *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, Vol. 10, No. 2, pp. 279-302, April 2005

Chapter 4

On the Tradeoff between Power and Flexibility of FPGA Clock Networks¹

4.1 Introduction

With advancements in process technology, programmable logic architecture, and computer-aided design, field-programmable gate arrays (FPGAs) are now being used to implement and prototype large system-level applications. These applications often have several clock domains. In order to support applications with multiple clock domains, FPGA vendors incorporate complex clock distribution circuitry within their devices [1]-[4].

Designing a suitable clock distribution network for an FPGA is significantly more challenging than designing such a network for a fixed function chip such as an Application-Specific Integrated Circuit (ASIC). In an ASIC, the location and skew requirements of each domain are known when the clock network is designed. In an FPGA, however, a single clock network that works well across many applications must be created. When the FPGA is designed, the number of clock domains the user will require, the clock signals that will be generated, the skew requirements of each domain, and where each domain will be located on the chip are all unknown. This forces FPGA vendors to create very complex yet flexible clock distribution circuitry.

This flexibility has a significant area and power overhead. Power is a particular concern because the clock signals toggle every clock cycle and are connected to a large number of the flip-flops. Previous studies have indicated that in a typical FPGA, 19% of the total FPGA power is dissipated in the clock network [5]. The more flexible the clock network, the more parasitic capacitance on the clock nets, and the more routing switches traversed by each clock signal; this leads to increased power dissipation. Clearly, FPGA vendors must carefully balance the flexibility of their clock distribution networks and the power dissipated by these networks.

¹ A version of this chapter has been submitted for publication. J. Lamoureux and S.J.E. Wilton, On the interaction between power and flexibility of FPGA clock networks, ACM Transactions on Reconfigurable Technology and Systems, 2007.

The clock distribution network also has an impact on the ability of the computer-aided design (CAD) tools to minimize the power and maximize the clock frequency of a user circuit. FPGA clock networks typically are not flexible enough to supply any clock signal to any flip-flop. As we will discuss in this chapter, typical networks allow only a subset of the clock signals to reach particular regions of the FPGA. This imposes additional constraints on the placement algorithm, as well as the clustering algorithm that groups logic elements into clusters. If the clock network is not flexible enough, these constraints could result in increased power dissipation and delay in a user circuit. Again, this balance must be considered by FPGA vendors as they design their clock distribution networks.

This chapter investigates the tradeoff between clock network flexibility and the power and speed of FPGAs. In particular, it makes the following contributions:

1. We present a parameterized framework that describes a family of FPGA clock networks and encompasses the salient features of commercial FPGA clock networks. Such a framework is important as it allows us to reason about and explore clock networks.
2. We present new clock-aware placement techniques that satisfy the placement constraints imposed by the clock network. As described above, the topology of the clock distribution architecture implies constraints on where logic blocks can be placed, and on what logic blocks can be packed together into clusters. For a given clock distribution architecture, our placement algorithm finds a legal placement solution that meets these constraints. As a secondary goal, the algorithms try to find a solution that uses the clock distribution network efficiently. For example, it may be possible to group together logic blocks such that parts of each clock network can be "turned off" or remain unused, thereby saving significant power. We consider several placement algorithms and compare their ability to meet the placement constraints as well as to minimize power by using the clock network efficiently. Our algorithms are implemented into the Versatile Place and Route (VPR) tool [6], and are flexible enough to target an FPGA with any clock distribution network that fits within our parameterized framework.
3. We examine how the architecture of the clock network and the placement algorithm used affects the overall power, area, and delay of the FPGA. We consider both the cost of clock

network itself and the impact of the constraints imposed by the clock network. In doing so, we identify the key parameters in our clock framework that have the most impact.

Together, these contributions provide insight into what makes a good FPGA clock distribution architecture.

Early versions of the parameterized clock network framework and clock-aware placement techniques were presented in [7] and [8], respectively. In this chapter, we combine and expand these studies. Specifically, in this chapter the assumptions we make regarding buffer sizing and buffer sharing within the clock networks have been revisited to reflect current low-skew design techniques. Moreover, the results have been expanded by including the impact of the clock network constraints on the clustering stage of the FPGA CAD flow. Finally, the empirical study has been made more concrete by analyzing a greater number of multiple clock-domain benchmark circuits.

This chapter is organized as follows. Section 4.2 provides background on clock networks and previous work related to FPGA clock networks and CAD. Section 4.3 describes a parameterized framework used to describe and compare different FPGA clock networks. Section 4.4 describes new clock-aware placement techniques for FPGAs. Section 4.5 examines which clock-aware placement techniques perform the best in term of power and speed. Section 4.6 then examines how FPGA clock networks affect overall FPGA power, area, and speed. Finally, Section 4.7 summarizes our conclusions.

4.2 Background and Previous Work

This section provides background on clock networks and describes previous work related to FPGA clock networks and clock-aware CAD.

4.2.1 Background

The primary goal when designing a clock network for any digital circuit (ASICs and FPGAs alike) is to minimize clock-skew, and the secondary aim is to minimize power and area. Many low-skew and low-power techniques for ASIC clock networks have been described in the literature. Buffered trees are the most common strategy for distributing clock signals [9]. Buffered trees have a root (clock source), a trunk, branches, and leaves (flip-flops), and are driven by buffers at the trunk and/or along the branches of the tree, as illustrated in Figure 4.1 (a). Another buffered-tree approach uses symmetry to minimize clock-skew, as illustrated in

Figure 4.1 (b). Symmetric buffered trees utilize a hierarchy of symmetric H-tree or X-tree structures to make the path from the source to each register the same length.

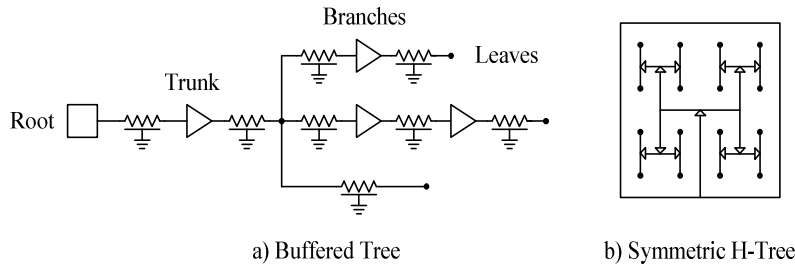


Figure 4.1: Example clock distribution networks.

Commercial FPGAs

Commercial FPGAs, such as the Actel ProASIC3 [1], the Altera Stratix III [3], and the Xilinx Virtex 5 [4] devices, support multiple local and global clock domains. In each device, the FPGA is divided into regions (see Figure 4.2). The Statix III has four quadrants that can be further subdivided into four sub-regions (per quadrant), the ProASIC3 has four quadrants, and the Virtex 5 has fixed size regions that are 20 rows high and span half the width of the FPGA. The Altera Stratix III has 16 global clock networks, which can be connected to all the flip-flops on the FPGA; and 22 local clock networks in each of the four quadrants, which can be connected to any of the flip-flops within that quadrant. Similarly, the Actel ProASIC3 has 6 global clock networks and 3 local clock networks per quadrant and the Xilinx Virtex 5 has 32 global clock networks and 10 local clock networks.

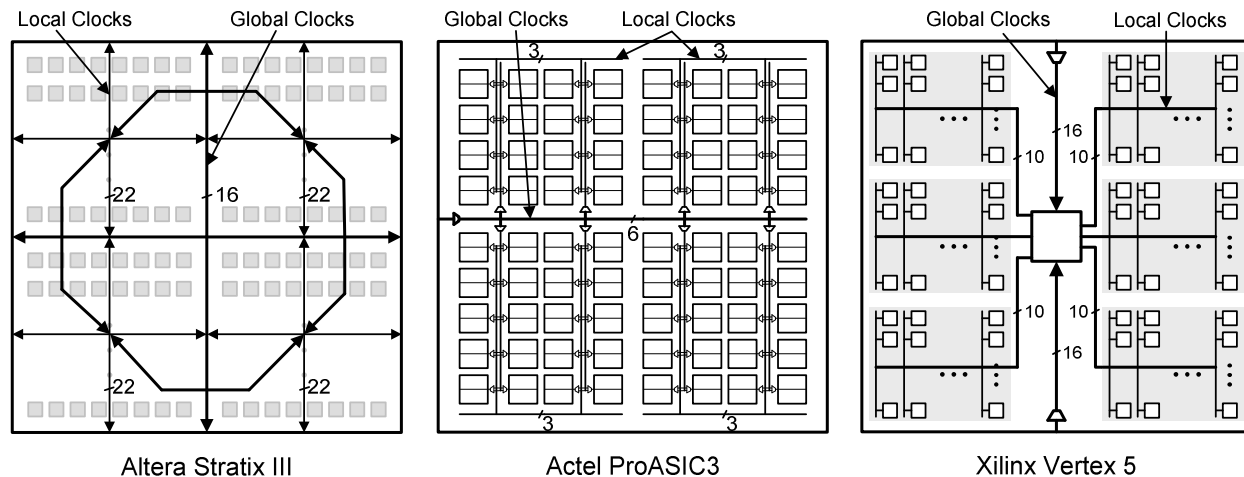


Figure 4.2: Commercial FPGA clock networks.

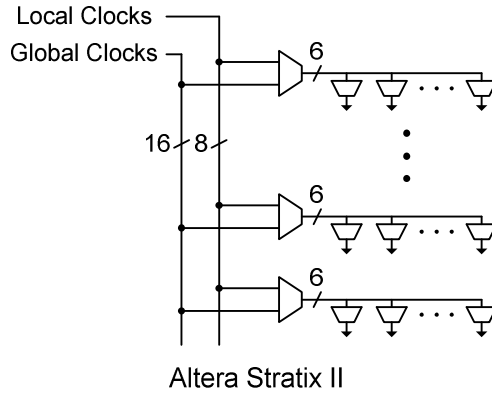


Figure 4.3: Multiplexer structure of Stratix II clock networks.

The circuitry that drives the clock networks is similar for each of the three devices. As shown in Figure 4.4, the local and global clock networks are driven by control blocks that select the clock signal and dynamically enables or disables the clock to reduce power consumption when the clock signal is not being used. The clock networks can be driven by an external source, an internal source, or by clock management circuitry which multiplies, divides and/or shifts an external source.

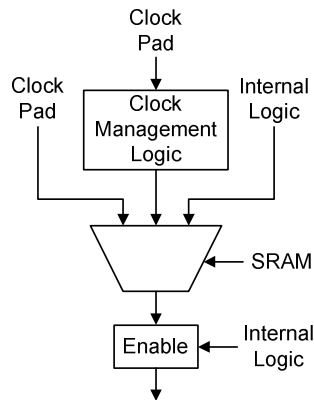


Figure 4.4: Control block for local and global clock signals.

Full Crossbar and Concentrator Networks

In this chapter, we will employ both full crossbar and concentrator crossbar networks as building blocks. An $n \times m$ single-stage crossbar is a single stage network that connects n inputs to m outputs, as shown in Figure 4.5. Figure 4.5 (a) shows a full crossbar network in which each output can be driven by any input. Such a crossbar requires $n \cdot m$ switches. Figure 4.5 (b) shows a concentrator network, in which any m -element set of the n input signals can be mapped to the m outputs (without regard for the order of the outputs). A concentrator built this way contains $m \cdot (n - m + 1)$ switches [10]. Concentrator crossbars are most efficient when m is close to n (i.e. the

network is close to square) or when m is very small (close to 1). A perfectly square crossbar concentrator only has n switches. As the crossbar concentrator becomes more rectangular, the number of switches approaches that of a full crossbar. Full and concentrator crossbars can also be implemented using fanin-based switches (multiplexers followed by buffers).

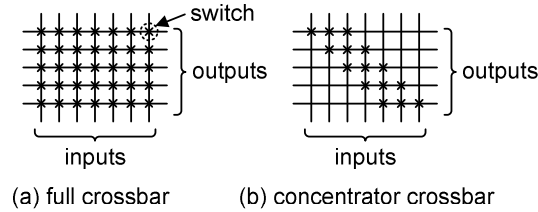


Figure 4.5: Two examples of a 7x5 crossbar network.

4.2.2 Previous Work

The existing literature on FPGA's has mostly assumed simplified FPGA clock architectures. The study described in [11] examines the design of energy efficient FPGA's. For clock networks, the study proposes that edge triggered flip-flops are used within logic blocks to reduce the power dissipated by the clock network, since this reduces the toggle rates by a factor of two. In [12],[13], FPGA power models that assume simple H-tree clock networks with buffers inserted at the end of each branch are described. In both models, clock networks span the entire FPGA and are implemented using dedicated (non-configurable) resources. In [12], additional buffers are inserted to separate long wire segments in order to minimize clock skew. Neither model is parameterized since the intent was only to include the power dissipated by the clock networks to make the power estimates somewhat realistic.

Clock-Aware CAD

Clock-aware placement has been considered in several studies related to ASICs. As an example, the quadratic placer described in [14] minimizes clock-skew by biasing the placement to evenly disperse clocked cells. As a result, the clock tree generated after placement is more balanced. Another technique, described in [15], minimizes clock-skew using a placement algorithm to optimally size and place clock buffers within the generated clock tree. Although useful for ASICs, these techniques are not applicable to FPGAs which have fixed (but configurable) clock networks.

FPGA Clock CAD

Only a few CAD studies have considered FPGA clock networks. In [16], clock-skew is minimized during placement by balancing the usage in each branch of the programmable clock network. This technique, however, is not necessary in recent FPGAs since the clock inputs to the logic blocks are buffered, which makes clock delay independent of usage. In [17], dynamic clock management is applied to FPGAs to minimize power. The technique involves dynamically slowing clock frequencies when the bandwidth demand decreases. This technique can also be used in conjunction with dynamic voltage scaling to further reduce overall power. Finally, the T-VPack clustering tool, described in [6], is clock-aware because it limits the number of clock signals that are used within each cluster, as specified by the user. Although the clusterer supports circuits with multiple clocks, the studies focus on FPGAs with only one clock.

4.3 Parameterized FPGA Clock Network Framework

This section presents a parameterized framework for describing FPGA clock networks. The framework can be used to describe a broad range of clock networks and encompasses the salient features of the clock networks used in current and future FPGAs. This framework is important since it provides a basis for comparing new FPGA clock networks and clock-aware CAD techniques.

The framework assumes a clock network topology with three stages. The first stage programmably connects some number of clock sources (clock pads, PLL/DLL outputs, or internal signals) to the center of some number of clock regions. The second stage programmably distributes the clocks to the logic blocks within each region. The third stage connects the clock inputs of each logic block to the flip-flops within that logic block. This topology is described in more detail in the following subsections.

4.3.1 Clock Sources

The source of the user clocks can be external, from a dedicated input pad, or internal, generated by a phase-locked loop (PLL), a delay-locked loop (DLL), or even the core of the FPGA. In all cases, we assume that these clock sources are distributed evenly around the periphery of the FPGA core. In our model, the number of potential clock sources is denoted n_{source} , meaning there are $n_{source}/4$ potential clock sources on each side of the FPGA.

4.3.2 Global and Local Clock Regions

The clock network has both local and global resources. The global clock resources distribute large user clocks across the entire chip (but not necessarily to every logic block, as described in Section 4.3.3). The local clock resources, on the other hand, distribute the smaller user clocks to individual regions of the FPGA. Although it is possible to distribute a large user clock to the entire chip by stitching together several local clocks, this would be less efficient than using a global clock.

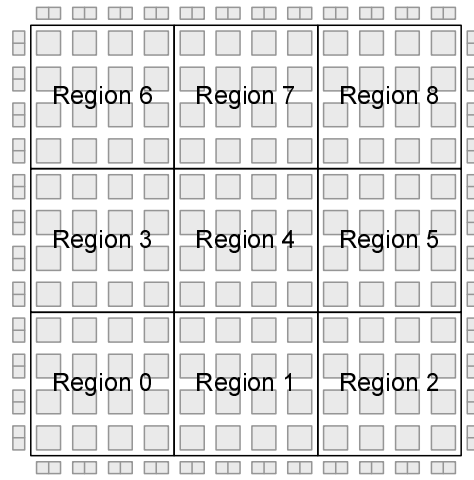


Figure 4.6: Example FPGA with 3×3 clock regions.

To support local clocks, the FPGA fabric is broken down into a number of regions, each of which can be driven by a different set of local clocks (the same local clock can also be connected to more than one region). The number of regions in the X-dimension is denoted by nx_region , and the number of regions in the Y-dimension is denoted by ny_region . The total number of regions is thus $nx_region*ny_region$. As an example, Figure 4.6 shows an FPGA in which nx_region and ny_region are both 3, which produces 9 clock regions.

4.3.3 First Network Stage

The first stage of the clock network programmably connects the clock sources on the periphery of the chip to the center of each region. The first stage consists of two parallel networks: one for the global clocks and one for the local clocks. We denote the total number of global clock signals as W_{global} . The global clock network selects $W_{global}/4$ signals from each of the $nsource/4$ potential clock sources on each side, as shown in Figure 4.7. This selection is done using a concentrator network on each side of the chip (see Section 4.2.1). The use of a concentrator network guarantees that any set of $W_{global}/4$ signals can be selected from the

$n_{source}/4$ sources on each side. This architecture does not, however, guarantee that any set of W_{global} signals can be selected from the n_{source} potential sources; it would be impossible to select more than $W_{global}/4$ signals from any side. Relaxing this restriction would require an additional level of multiplexing (or larger concentrators and longer wires), which we do not include in our model.

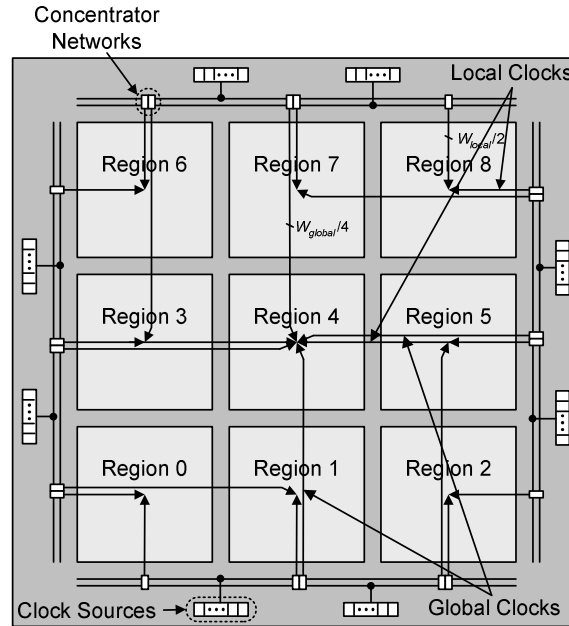


Figure 4.7: Connections from the periphery to the center of regions (local clocks) and the center of the FPGA (global clocks).

All W_{global} signals are routed on dedicated wires to the center of the chip. Since the sources all come from the periphery of the chip, the connection between the sources and the center of the chip will introduce little or no skew. In an architecture in which some sources come from inside the chip, the drivers can be sized such that skew is minimized. From the center of the chip, all W_{global} clocks are distributed to the center of all regions using a spine-and-ribs distribution network with n_{y_region} ribs, as shown in Figure 4.8. Although an H-tree topology would have a lower skew, it is more difficult to mesh such a topology onto a tiled FPGA with an arbitrary number of rows and columns.

There is one local clock network per region. We denote the number of local clocks per region as W_{local} . The W_{local} signals are selected from the two sides of the FPGA that are closest to the region, as shown in

Figure 4.7 (if the number of regions in either dimension is odd, the selection of the “closest” side is arbitrary for regions in the middle). Half of the W_{local} signals are selected from

the sources on each of the two sides using a concentrator network. The use of a concentrator network guarantees that any set of $W_{local}/2$ signals can be selected from the $n_{source}/4$ potential sources on two sides. Driver sizing can be used to minimize the skew among the clocks connected to each region. Skew between regions is not as important, since global clocks will likely be used if a clock is to drive multiple regions.

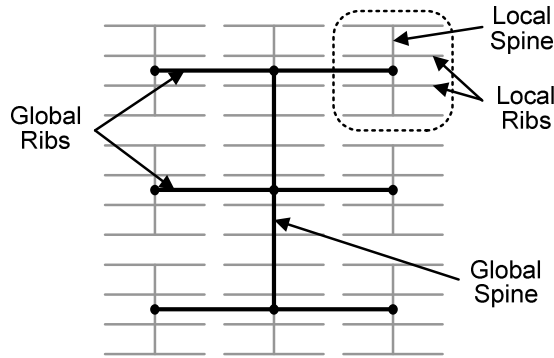


Figure 4.8: Global clock connections from the center of the FPGA to the center of regions.

4.3.4 Second Network Stage

The second network stage programmably connects clock signals from the center of each region to the logic blocks within that region. There is one of these networks for each region.

The input to each second stage network consists of W_{global} global clocks and W_{local} local clocks from the first stage. These clocks are distributed using a spine-and-ribs topology as shown in Figure 4.9. The spine contains $W_{global}+W_{local}$ wires. In each row, a concentrator network is used to select any set of W_{rib} clocks from the spine. These clocks are distributed to the logic blocks in that row through a local rib. Each logic block in the row connects to the rib through another concentrator network; the concentrator is used to select any set of W_{lb} clocks from the rib.

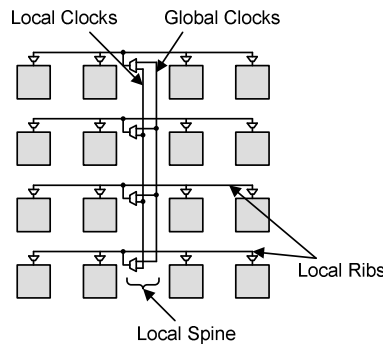


Figure 4.9: Second stage of the clock network.

4.3.5 Third Network Stage

Finally, the third network stage programmably connects the W_{lb} logic block clocks to the N logic elements within the logic block. This is illustrated in Figure 4.10. In order to provide flexibility to the clustering tool, we assume that the clock pins of the logic block are connected to the clock pins of the logic elements (within that logic block) using a full-crossbar network, such that each of the N logic elements can be clocked by any of the W_{lb} logic block clocks.

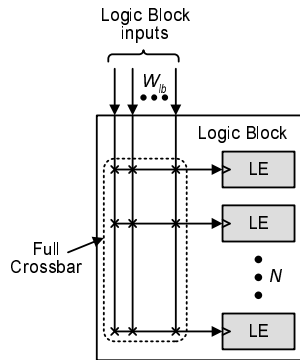


Figure 4.10: Third stage of the clock network.

Table 4.1 summarizes the parameters of the FPGA clock network framework and includes corresponding values that are used in the empirical studies described in this Section 4.5 and 4.6 of this chapter. These values are similar to those used in current commercial devices [1]-[4].

Table 4.1: Clock network parameters.

Parameter	Meaning	Baseline Value	Range in this Chapter
nx_region	Number of clock regions in the X dimension	2	1 - 4
ny_region	Number of clock regions in the Y dimension	2	1 - 4
$nsource$	Number of potential clock sources (one quarter of these are on each side)	128	128
W_{global}	Number of global clocks that can be routed to the fabric at one time	52	0-52
W_{local}	Number of local clocks that can be routed to each region at one time	52	52
W_{rib}	Number of clocks that can be routed to all logic blocks in a single row of a region at one time.	10	6-14
W_{lb}	Number of clocks that can be connected to each logic block.	2	1-10

4.4 Clock-aware FPGA CAD

The previous section presented a parameterized framework to describe FPGA clock networks. This section describes new clock-aware placement techniques for finding a legal placement solution that satisfies all of the placement constraints imposed by the clock network.

4.4.1 Clock-Aware Placement

Placement can have a significant impact on power, delay, and routability. Placing logic blocks on a critical-path close together minimizes delay and placing logic blocks connected by many wires close together improves power and routability. Power can be further minimized by assigning more weight to connections with high toggle rates, as described in [18]. For applications with many clock domains, however, constraints that limit the number of clock nets within ribs (W_{rib}) and within a region (W_{local}) can interfere with these optimizations.

To investigate how clock networks constraints affect the power, delay, and routability during placement, the T-VPlace algorithm from [6] was enhanced to make it clock-aware. T-VPlace is based on simulated annealing. The original cost function used in T-VPlace has two components. The first component is the *wiring cost*, which is the sum of the bounding box dimensions of all nets (not including clock nets). The second component is the *timing cost*, which is a weighted sum of the estimated delay of all nets. The cost of a swap is then:

$$\Delta C = \lambda \cdot \frac{\Delta \text{Timing Cost}}{\text{Previous Timing Cost}} + (1 - \lambda) \cdot \frac{\Delta \text{Wiring Cost}}{\text{Previous Wiring Cost}} + \quad (4.1)$$

where *PreviousTimingCost* and *PreviousWiringCost* terms in the expression are normalizing factors that are updated once every temperature change, and λ is a user-defined constant which determines the relative importance of the cost components.

Three enhancements were made to T-VPlace in order to make it clock-aware. First, a new term was added to the existing cost function to account for the cost of the clock. Second, a new processing step that determines which user clock nets use local clock network resources and which clock nets use global clock network resources was added. Third, the *random* initial placement approach (used in the original placer) was replaced with a new routine that finds a legal initial placement. Each enhancement is described below.

New Cost Function

The new cost function is the same as the original cost function, except that it has a new term to account for the cost of using clock network resources. Intuitively, the new term in the cost function minimizes the usage of clock network resources, which minimizes clock power and is key for finding legal placements. The new cost function is described by the following expression:

$$\Delta C = \lambda \cdot \frac{\Delta \text{Timing Cost}}{\text{Previous Timing Cost}} + (1 - \lambda) \cdot \frac{\Delta \text{Wiring Cost}}{\text{Previous Wiring Cost}} + \gamma \cdot \frac{\Delta \text{Clock Cost}}{\text{Previous Clock Cost}} \quad (4.2)$$

Like the two original terms, the new term is normalized by the cost of the clock during the previous iteration and is weighted by the factor γ . Experimentally, we have found that $\gamma = 0.3$ produces good results.

Two different cost functions were considered for the clock term. The first, called the *standard* cost function, is based on the amount of clock network resources needed to route all the user clock nets. Specifically, the cost function counts the number of clocks used in each rib, local region, and global region. Moreover, the cost of each resource type (rib, local, or global) is scaled by a constant (K_{rib} , K_{local} , or K_{global} , respectively) to reflect the capacitance of those resources.

Although straight-forward, the standard cost function can be short-sighted when large (high fanout) user clock nets occupy more than one region. Unless all the pins are moved out of a region, the cost of occupying that region does not change.

The second cost function, called the *gradual* cost function, changes when large nets are partially moved. The function scales the cost of adding an LE to a region based on how many other LEs in that region are connected to the same user clock net as that LE. Specifically, the incremental cost of adding an LE that uses clock net i to region j is described by the following expression.

$$\text{Clock Cost}(i, j) = K_j \cdot \frac{\text{max_pins}(i, j) - \text{pins}(i, j) + 1}{\text{max_pins}(i, j)}$$

where,

(4.3)

$\text{pins}(i, j) \equiv$ number of pins of net i in region j

$\text{max_pins}(i, j) = \min(\# \text{ pins of clock net } i, \# \text{ LEs per region } j)$

In the expression, K_j is the weight factor or region j , which is either K_{rib} , K_{reg} , or K_{global} , depending on what type of region is being considered. By summing the cost of all the clock nets in every region, the total clock cost reduces to:

$$\text{Clock Cost} = \sum_{i \in \text{clock nets}} \left(\begin{array}{l} K_{rib} \cdot \sum_j^{\text{ribs}} \text{pins}(i,j) \cdot \left[1 - \frac{\text{pins}(i,j) - 1}{2 \cdot \text{max_pins}(i,j)} \right] + \\ K_{reg} \cdot \sum_j^{\text{regions}} \text{pins}(i,j) \cdot \left[1 - \frac{\text{pins}(i,j) - 1}{2 \cdot \text{max_pins}(i,j)} \right] + \\ K_{global} \cdot \text{pins}(i) \cdot \left[1 - \frac{\text{pins}(i) - 1}{2 \cdot \text{pins}(i)} \right] \end{array} \right) \quad (4.4)$$

Intuitively, both cost functions encourage swaps that reduce the amount of clock network resources that are used. In the second cost function, however, the cost of moving a logic block to a region is smaller when other logic blocks in that region are connected to the same clock net. The goal of the gradual cost function is to prevent large clock nets from spreading out to more regions than necessary. However, since the function does not reflect the actual cost of the clock as directly as the first cost function, the overall results may not be minimized as intended. An empirical study, described in Section 4.5, was performed to determine which cost function is most suitable.

Clock Resource Assignment

The second enhancement determines which user clock nets should use the local clock network resources and which user clock nets should use the global clock network resources. Although this decision could be left to the user, it is more appropriate for the tool to make the assignment since it is convenient and the user may not be familiar with the underlying architecture of the clock network.

Global clock network resources are more expensive than the local clock network resources in terms of power since they are routed to the center of the FPGA before spanning to the center of the clock regions. Depending on the clock network and application, global clock network resources may also be in short supply. Therefore, global clock networks should be reserved for large nets that do not fit within local regions or for nets that are inherently spread out.

We consider two approaches for assigning global clock network resources. The first approach assigns global resources statically, based on the size (fanout) of the clock nets. The advantage of this approach is that it is quick and easy. The disadvantage is that it overlooks

smaller nets that require global resources because they are inherently spread out. The second approach assigns global resources dynamically during placement based on how spread out the clock nets are. Figure 4.11, Figure 4.12, and Figure 4.13 describe each assignment technique in more detail.

The static assignment routine, described in Figure 4.11, begins by determining how many clock nets use global clock network resources based on the number of clock nets in the application, the number of global clock network resources available, and a user-defined margin. The margin, which we set to 0.5, effectively relaxes the W_{local} placement constraint and makes finding a legal solution easier. The routine then assigns the global resources to the clock nets with the greatest number of pins (highest fanout).

```

assign_global_resources_statically () {
    num_global_min = max( 0, num_clock_nets - num_local_regions * num_local_clocks );
    num_global_safe = min( num_clock_nets, num_global_clocks, num_global_min +
                          num_clock_nets * MARGIN );

    /* biggest to smallest by fanout */
    sorted_clock_nets = sort( clock_nets, num_pins );

    for ( iclk = 0; iclk < num_global_safe; iclk++ )
        use_global[sorted_clock_nets[iclk]] = TRUE;
}

```

Figure 4.11. Pseudo-code description of the static clock resource assignment technique.

The dynamic assignment technique described in Figure 4.12 is applied during placement. Initially, all the user clock nets are assigned to use local clock network resources. Then, during the simulated annealing routine, some clock nets are *reassigned* to use global resources if the placer can not find a legal placement. Specifically, the placer reassigns clock nets when the cost of the clock stops decreasing and the placement is still not legal. When clock nets are reassigned, the temperature is increased to allow the placer to adapt to the change. The pseudo-code in Figure 4.12 is based on T-VPlace, which is described in [6].

The clock nets are *reassigned* using the routine described in Figure 4.13. The routine begins by calculating how spread out each clock net is by calculating the *locality distance*, which counts how many clock pins would need to move in order to make the clock net local. After sorting each net by locality distance, it then assigns global clock network resources to half of the remaining local clock nets that have the highest locality distance.

```

placement_with_dynamic_assign () {

    assign_local_resources_to_all_clock_nets ();
    cost = random_placement();
    T = initial_temperature();

    while ( exit_criterion () == False ) {
        count = swap_count = 0;
        old_cost = cost;
        while ( inner_loop_criterion () == False) {
            if (try_swap (t, cost) == 1) {
                swap_count++;
            }
            count++;
        }
        if ( is_placement_legal() == False && swap_count < count / 2 && old_cost <= cost) {
            reassign_clocks ();
            t = t * 5; /* increase temperature */
        }
        t = update_temp ();
    }
}

```

Figure 4.12. Pseudo-code description of the dynamic clock resource assignment technique.

```

reassign_clock () {
    num_not_local = 0;
    for (iclk=0; iclk<num_clock_nets; iclk++) {
        if (use_global[iclk] == FALSE) {
            locality_dist = calc_locality_distance (iclk);
            locality_dist[iclk] = locality_dist;
            if (locality_dist > 0) {
                num_not_local++;
            }
        }
    }
    /* sort biggest to smallest distance*/
    sorted_clock_nets = sort ( clock_nets, locality_dist );

    num_to_reassign = min( (num_not_local + 1) / 2, num_global_available )
    for (iclk=0; iclk<num_to_reassign; iclk++) {
        use_global[sorted_clock_nets[iclk]] = TRUE;
    }
}

```

Figure 4.13. Pseudo-code description of the routine used to reassign clock nets.

Legalization

The final enhancement needed to make the placer clock-aware is to produce a placement that is legal. Legalization ensures that the number of different clock nets used in every region is less than or equal to the number of clock resources available in that region.

We consider two approaches. The first approach finds a legal solution before placement. A legal solution is found using simulated annealing with the timing and wiring cost components of the cost function turned off (leaving only the clock cost component turned on). If a legal

placement is found, the actual placement is then performed with all three cost components turned on but only allowing legal swaps.

The second approach involves legalizing during the actual placement. The algorithm starts with a random placement and then uses simulated annealing with the timing, wiring, and clock costs turned on. To gradually legalize the placement, the clock cost component is modified to severely penalize swaps that make the placement either illegal or more illegal. In other words, illegal swaps are allowed but they have a higher cost. Explicitly, we multiplied the cost of using a rib, spine, or global routing resource that is not available by a constant value, called *Illegal_Factor*. In our experiments, we found 10 to be a suitable value for *Illegal_Factor*.

4.5 Clock-Aware Placement Results

This section begins by describing the experimental framework that is used in this chapter and then compares the clock-aware placement techniques that were described in the previous section.

4.5.1 Experimental Framework

The same empirical framework is used in Section 4.5 and Section 4.6. A suite of benchmark circuits is implemented on a user-specified FPGA architecture using standard academic FPGA CAD tools. The CAD tools consist of the Emap technology mapper [18], the T-VPack clusterer [6], the VPR placer (with clock-aware enhancements), and the VPR router [6]. Finally, the power, area, and delay of each implementation are modeled using VPR for area and delay and the power model from [13], which has been integrated into VPR.

The VPR models are very detailed, taking into account specific switch patterns, wire lengths, and transistor sizes. After generating a user-specified FPGA architecture, VPR places and routes a circuit on the FPGA and then models the power, area, and delay of that circuit. The area is estimated by summing the area of every transistor in the FPGA, including the routing, CLBs, and configuration memory. The delay is estimated using the Elmore delay model and detailed resistance and capacitance information obtained from the router. The power is modeled using the capacitance information from the router and externally generated switching activities to estimate dynamic, short-circuit, and leakage power. In this chapter, the switching activities, which are required by the power model, are obtained using gate-level simulation and pseudo-random input vectors.

We enhanced the VPR power and area models to account for the parameterized clock network described in Section 4.3. The following buffer sharing and sizing assumptions were used to model the clock networks:

1. Each switch is implemented using a transmission gate controlled by an SRAM cell. The transmission gate consists of one minimum sized NMOS transistor and one 2X PMOS transistor in parallel.
2. Shared buffers are used to drive all periphery, spine, rib, and CLB clock network wires.
3. Large cascaded buffers with four stages (1X, 4X, 16X, and 64X) are used to drive the periphery, spine, and rib wires and smaller cascaded buffers with three stages (1X, 4X, and 16X) are used to drive the CLB clock wires.
4. Large repeaters (64X) are used to drive very long wires and are spaced by 5mm.
5. Unused clock networks are turned off to reduce power consumption.

An example of clock network switches of buffers is illustrated in Figure 4.14. Note that these buffer sizing and sharing assumptions differ from the assumptions described in our earlier work [7] in which the buffers were smaller but not shared, and the repeaters spacing was significantly smaller, separated by only one tile ($\sim 125\mu\text{m}$). The new assumptions reduce the power, area, and skew of the clock network by reducing the number of buffers and repeaters and the overall amount of parasitic capacitance.

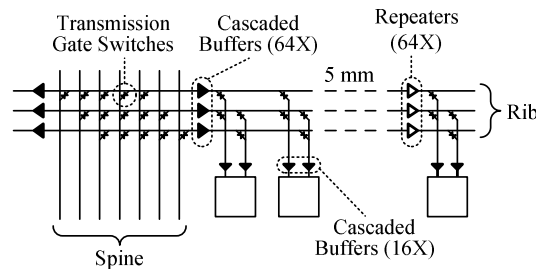


Figure 4.14: Example of buffer sizing and sharing in the clock network.

Benchmark Circuits

In order to investigate new FPGA clock network architectures and clock-aware CAD tools, benchmark circuits with multiple clock domains are needed. Existing academic benchmark circuits, however, are small and have only one clock. Moreover, since developing system-level circuits with multiple clock nets is labor intensive and expensive, commercial vendors are reluctant to release their intellectual property.

As a solution, we combine a number of single domain benchmark circuits to form larger multi-domain circuits that resemble system-level circuits from a place and route viewpoint. The technique uses Rent’s Rule [19] to determine the number of primary inputs and outputs the combined circuit should have and inserts additional flip-flops between the clock domains as an interface. Using this technique and benchmark circuits from MCNC, several large benchmark circuits with multiple clock domains were generated. Table 4.2 lists the new benchmark circuits and provides additional details regarding the circuit size and number of clock domains. Note also that the benchmark names are intended to provide some insight regarding the size of the clock domains. As an example, 1lrg40sml consists of one large circuit and 40 small circuits. See Appendix A for more information regarding the benchmark circuits.

Table 4.2. Benchmark circuit names and attributes.

Name	# LUTs	# FFs	# Clock Domains
10lrg	14320	4780	10
10lrg_reordered	14320	4780	10
15med	5710	2361	15
1lrg40sml	9452	3179	41
2lrg4med80sml	15289	9119	86
30med	5210	1874	30
30mixedsize	12537	3399	30
30seq20comb	16398	2984	50
3lrg50sml	10574	4433	53
40mixedsize	12966	4578	40
4lrg4med16sml	11139	5026	24
4lrg4med4sml	8873	3868	12
50mixedsize	16477	5681	50
5lrg35sml	11397	3646	40
60mixedsize	15246	5217	60
6lrg60sml	15838	8037	66
70s1423	13440	7308	70
70sml	9301	2398	70
lotsaffs	7712	3755	19
lotsaffs2	9309	4567	32

FPGA Architecture and Assumptions

All experiments target island-style FPGAs implemented in a 180nm TSMC process. We use the baseline FPGA architecture described in [6], which consists of logic blocks with 10 logic elements each and a segmented routing fabric with length 4 wires. For the clock network, we assume the baseline clock architecture described in Table 4.1, which is similar to current commercial architectures.

For each experiment, the size of the FPGA is determined by the size of the benchmark circuit. Specifically, the size is determined by finding the minimum number of FPGA tiles ($nx \times ny$) which is greater or equal to the number of logic blocks in the benchmark circuit. For the general routing (not clock routing), the channel width is selected by finding the minimum routable channel width and then adding 20% to that channel width. These assumptions serve to model the case when FPGAs are highly utilized.

4.5.2 Placement Results

To compare the techniques for each of the three enhancements, we implemented eight different clock-aware placers (one for each possible combination). The eight placers are described in Table 4.3. As an example, the first placer (Placer 1) uses the standard function for the clock term in the cost function, the static approach to assign global clock network resources, and the pre-placement approach to legalize the placement. Note that Placer 3 and Placer 7 assign global clock network resources dynamically during the pre-placement, while Placer 4 and Placer 8 assign global clock network resources dynamically during the actual placement since there is no pre-placement in the latter implementations.

Table 4.3. Techniques used by each placer.

Placer	Cost Function	Assignment Technique	Legalization Technique
Placer 1 (P1)	Standard	statically	pre-placement
Placer 2 (P2)	Standard	statically	during placement
Placer 3 (P3)	Standard	dynamic	pre-placement
Placer 4 (P4)	Standard	dynamic	during placement
Placer 5 (P5)	Gradual	statically	pre-placement
Placer 6 (P6)	Gradual	statically	during placement
Placer 7 (P7)	Gradual	dynamic	pre-placement
Placer 8 (P8)	Gradual	dynamic	during placement

Table 4.4 presents the overall energy per cycle dissipated by each benchmark circuit when implemented by the eight different clock-aware placers. The average is calculated using the geometric mean (rather than the arithmetic mean) to ensure that each benchmark circuit contributes evenly to the average, regardless of its size. Moreover, the averages only include benchmark circuits that were successfully implemented by every placer to make the results comparable.

Table 4.4. Overall energy per clock cycle.

Benchmark	Overall Energy (nJ)								
	Orig.	P1	P2	P3	P4	P5	P6	P7	P8
10lrg	6.63	-	-	8.03	6.45	6.74	6.34	6.42	6.34
10lrg_reordered	6.98	-	-	6.74	6.59	6.59	7.19	6.50	6.46
15med	2.91	2.99	3.06	3.07	3.79	2.89	2.89	2.86	2.84
1lrg40sml	4.5	4.60	4.65	4.66	4.52	4.48	4.49	4.41	4.69
2lrg4med80sml	12.3	12.4	15.4	11.6	11.6	11.7	13.1	11.1	11.8
30med	2.68	2.73	2.80	2.65	2.60	2.67	2.60	2.54	2.58
30mixedsize	7.24	7.03	7.90	6.97	7.12	6.54	6.60	6.48	6.47
30seq20comb	6.48	6.43	6.52	6.45	6.37	6.85	6.85	6.22	6.02
3lrg50sml	5.42	5.50	5.46	5.17	5.11	5.15	5.14	4.96	4.84
40mixedsize	7.79	8.16	9.77	7.90	8.02	7.63	7.84	7.46	7.58
4lrg4med16sml	6.34	6.59	6.71	6.69	6.41	6.33	6.60	6.20	6.37
4lrg4med4sml	4.49	4.63	4.69	4.69	4.43	4.49	4.39	4.48	4.37
50mixedsize	9.98	10.35	10.35	10.01	10.25	9.59	9.77	9.56	9.59
5lrg35sml	6.25	6.69	6.96	6.31	6.43	6.10	6.76	6.15	5.93
60mixedsize	9.56	10.4	10.1	10.0	9.49	9.09	9.29	10.04	8.85
6lrg60sml	12	13.3	15.1	12.8	12.4	11.6	11.8	11.3	11.3
70s1423	11	-	-	10.8	10.8	-	-	10.1	10.7
70sml	5.07	5.23	5.40	4.92	4.86	5.11	4.89	4.79	4.77
lotsaffs2	3.73	3.52	3.60	3.46	3.33	3.47	3.73	3.39	3.30
lotsaffs	2.32	2.30	2.28	2.27	2.14	2.24	2.28	2.31	2.14
Geomean	5.73	5.87	6.15	5.74	5.71	5.58	5.70	5.49	5.43
% Diff.	0	2.5	7.4	0.2	-0.2	-2.5	-0.5	-4.2	-5.1

A number of observations can be drawn from the table. First, in some experiments, the placer failed to find a legal solution. Even when the placer starts by looking for a legal solution, there is still no guarantee that a legal placement will be found. In this experiment, placers P1 and P2 failed to find a legal placement for the 10lrg, 10lrg_reordered, and 70s1423 benchmark circuits and placer P5 and P6 failed to find a legal placement for 70s1423. These placers failed because they use the static global assignment approach to decide whether clocks use local or global clock network resources. In these cases, more global clock network resources needed to be allocated. In terms of energy, the placers that use the static approach (P1, P2, P5, and P6) also performed worse than the corresponding placers (P3, P4, P7, and P8) that assign global clock network resource dynamically. Intuitively, assigning the global clock dynamically works better since more information is available when the resources are being assigned. Specifically, the placer can determine which nets are spread out over more than one region.

Another notable observation is that the gradual clock cost function produced placements that were more energy efficient than the standard cost function. On average, placers P5 to P8 were between 3.5% and 5.4% more energy efficient than placers P1 to P4, respectively. To

examine this further, Table 4.5 compares the average energy of the clock and general purpose routing resources and the average critical-path delay of each clock-aware placer to that of the original non-clock-aware placer from VPR. Note that the original placer ignores clock nets and any of the associated placement constraints. Therefore, the placement solutions produced by the original placer are not legal. This table serves to highlight the impact that the clock constraints have on each of the placers.

Table 4.5. Impact of clock constraints for each placer.

Placer	Clock Energy (nJ)	% Diff	Routing Energy (nJ)	% Diff	Tcrit (ns)	% Diff
Orig	2.28	0.0	2.05	0.0	51.9	0.0
P1	1.90	-17	2.58	26	52.7	1.5
P2	1.92	-16	2.81	37	53.4	2.8
P3	1.81	-21	2.54	24	53.0	2.1
P4	1.81	-21	2.50	22	52.1	0.2
P5	2.12	-6.9	2.07	1.2	52.3	0.6
P6	2.12	-7.0	2.19	7.1	51.6	-0.7
P7	1.97	-14	2.13	3.9	52.8	1.7
P8	1.97	-14	2.08	1.8	52.0	0.0

The table demonstrates that the clock constraints can have a significant impact on the energy dissipated by the clock networks and general purpose routing resources. As expected, the clock-aware implementations dissipate significantly less clock power than the original (non clock-aware) implementations since the clock-aware placers minimize clock usage. On the other hand, the table shows that the clock-aware implementations dissipate more power within the general purpose routing resources. This is especially true for placers P1 to P4, which use the standard clock cost function. This effect is reduced for placers P5 to P8, which use the gradual clock cost function.

In terms of speed, the clock-aware placement techniques only have a small impact with average critical-path delay variations of approximately 1% in either direction.

Finally, from the results, it appears that Placer 8 is the best overall clock-aware placer with the lowest overall energy per cycle and the second fastest average critical-path delay. Specifically, Placer 8 is 5% more energy efficient than the original VPR placer (which does not produce legal placements) and 12.5% more energy efficient than Placer 2, the least efficient clock-aware placer.

4.6 Clock Network Architecture Results

In the previous section, we compared different clock-aware placement techniques. In this section we use the best clock-aware placer from the previous section (Placer 8) to compare how different clock network parameters affect overall power, area, and delay. Specifically, we consider four clock network parameters: W_{lb} , W_{rib} , W_{glb} , and $nx(y)_{region}$. For each experiment, we vary one parameter at a time. Our goal is not to exhaustively measure the cost of every possible combination of parameters, but rather to examine the impact that each clock network constraint has on the overall power, area, and critical-path delay of system-level applications.

4.6.1 Clocks per Logic Block (W_{lb})

We first consider the flexibility within the logic blocks by varying W_{lb} , the number of clocks per logic block. Intuitively, the larger this number, the simpler the task for the clustering algorithm (since the constraint on the number of clocks per logic block is not as severe); however, the larger W_{lb} , the more power-hungry the clock network will be. Specifically, we would expect that limiting the number of clocks per logic block would reduce the packing efficiency of the clusterer since this limits which blocks can be packed together. To examine this, we packed the benchmark circuits and varied the number of clock signals per logic block between 1 and 3. Table 4.6 presents the results.

The table shows that increasing the number of clocks per logic block increases packing efficiency. However, the efficiency only increases by 1.2% when W_{lb} is increased from 1 to 3. The main reason for the small impact is that the efficiency is already close to 100% when W_{lb} is limited to 1, which leaves little room for improvement.

We next consider how W_{lb} affects the area of the clock network. Figure 4.15 shows a breakdown of the clock network area relative the overall FPGA area. Note that the area for some parts of the clock network is not visible because the area they occupy is insignificant. The graph varies W_{lb} from 1 to 10 and uses the baseline value from Table 4.1 for the remaining parameters. As shown, the area due to the logic block to logic element (LB-LE) switches increases significantly as W_{lb} increases. Recall that a full crossbar network was assumed within the logic block. Therefore, increasing W_{lb} by one adds one extra switch for every logic element in the FPGA. Note also that the clock network has a greater impact on power than it does on area.

Table 4.6: Logic utilization vs. number of clocks per logic block (W_{lb}).

Benchmark	# BLEs	$W_{lb} = 1$		$W_{lb} = 2$		$W_{lb} = 3$	
		# CLBs	Packing Efficiency (%)	# CLBs	Packing Efficiency (%)	# CLBs	Packing Efficiency (%)
10lrg	15778	1587	99.4	1583	99.7	1582	99.7
10lrg_reordered	15780	1587	99.4	1582	99.7	1582	99.7
15med	6650	673	98.8	669	99.4	667	99.7
1lrg40sml	10600	1078	98.3	1064	99.6	1062	99.8
2lrg4med80sml	20810	2124	98.0	2092	99.5	2085	99.8
30med	6301	644	97.8	634	99.4	632	99.7
30mixedsize	14087	1427	98.7	1418	99.3	1415	99.6
30seq20comb	17708	1805	98.1	1788	99.0	1781	99.4
3lrg50sml	11832	1212	97.6	1189	99.5	1188	99.6
40mixedsize	14982	1518	98.7	1502	99.7	1500	99.9
4lrg4med16sml	12844	1297	99.0	1287	99.8	1285	100.0
4lrg4med4sml	10146	1022	99.3	1016	99.9	1016	99.9
50mixedsize	18676	1889	98.9	1875	99.6	1870	99.9
5lrg35sml	12714	1293	98.3	1277	99.6	1272	100.0
60mixedsize	17775	1808	98.3	1789	99.4	1784	99.6
6lrg60sml	19765	2011	98.3	1983	99.7	1980	99.8
70s1423	15899	1622	98.0	1605	99.1	1598	99.5
70sml	10363	1071	96.8	1049	98.8	1043	99.4
lotsaffs	10276	1046	98.2	1031	99.7	1029	99.9
lotsaffs2	8321	839	99.2	837	99.4	835	99.7
Average	-	-	98.5	-	99.5	-	99.7

The area due to the rib to logic block (RIB-LB) switches increases as W_{lb} increases from 1 to 6, and then decreases as W_{lb} increases from 7 to 10. This is because the number of switches in a concentrator network is smallest when the number of outputs is either very small or close to the number of inputs. Because of this, the incremental area cost of increasing W_{lb} when W_{lb} is more than half of W_{rib} (the number of clocks in each rib) is small.

Note that preliminary clock network area results presented in [7] differ from the area results in this chapter since we use different buffer sizing and buffer sharing assumptions. Although the conclusions we draw are similar, the area overhead of the clock network is smaller when the new assumptions are modeled compared to when the preliminary assumptions are modeled. As an example, the area of the baseline clock network described in Table 4.1 accounts of 4.0% of the overall area (in this chapter) and 7.2% (in [7]).

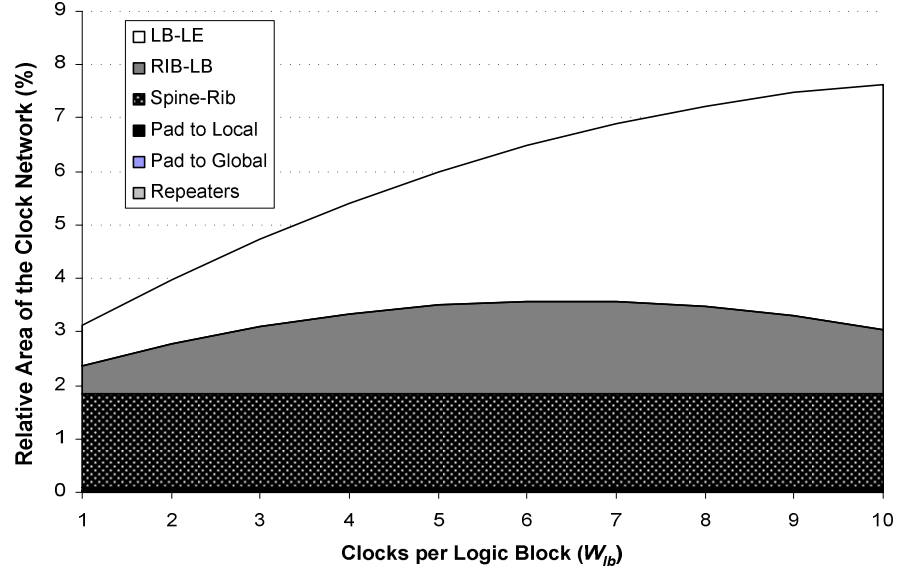


Figure 4.15: Clock network area vs. W_{lb} .

Finally, we consider how the W_{lb} constraint affects the overall energy per clock cycle and the critical-path delay. Table 4.7 summarizes the results for each circuit when the W_{lb} is varied from 1 to 3.

Table 4.7. Critical-path delay and energy vs. number of clocks per logic block (W_{lb}).

Benchmark	Energy per cycle (nJ)			T_{crit} (ns)		
	$W_{lb}=1$	$W_{lb}=2$	$W_{lb}=3$	$W_{lb}=1$	$W_{lb}=2$	$W_{lb}=3$
10lrg	6.29	6.34	6.60	65.6	67.3	67.9
10lrg_reordered	6.29	6.46	6.60	69.9	69.2	72.2
15med	2.71	2.84	2.86	67.3	73.2	71.4
1lrg40sml	4.69	4.69	4.47	71.2	71.2	70.8
2lrg4med80sml	9.84	11.84	11.84	35.5	34.9	34.9
30med	2.38	2.58	2.63	32.9	33.4	32.2
30mixedsize	6.03	6.47	6.66	42.1	39.9	39.8
30seq20comb	6.64	6.02	6.46	70.4	72.5	72.5
3lrg50sml	4.62	4.84	5.26	35.6	35.0	35.5
40mixedsize	6.99	7.58	9.47	65.3	71.0	69.6
4lrg4med16sml	5.99	6.37	6.60	70.6	75.3	76.2
4lrg4med4sml	4.33	4.37	4.46	36.3	35.1	35.8
50mixedsize	9.20	9.59	9.83	64.7	68.3	70.0
5lrg35sml	5.54	5.93	6.07	69.1	73.0	74.4
60mixedsize	8.22	8.85	8.85	40.7	40.4	40.4
6lrg60sml	10.56	11.31	11.81	71.1	70.0	69.2
70s1423	9.01	10.72	10.72	29.3	32.3	32.3
70sml	4.79	4.77	4.82	64.3	70.5	69.7
lotsaffs2	3.06	3.30	3.48	36.8	36.3	36.2
lotsaffs	2.10	2.14	2.16	36.3	35.2	34.6
Geomean	5.39	5.70	5.81	53.8	55.0	55.0
% Diff.	-5.45	0.00	1.84	-2.2	0.0	0.0

The results indicate that increasing the number of clocks per logic block actually increases (rather than decreases) the overall energy and critical-path delay. Intuitively, increasing W_{lb} should decrease energy and delay since the clusterer can pack logic elements more efficiently when logic elements with different clocks can be packed together. However, both tend to increase due to cases when flip-flops from different clock domains are packed in the same logic block (during clustering) and the corresponding clock domains are placed in different regions of the FPGA (during placement). When this occurs, flip-flops end up being placed far apart, which leads to increased routing demand and critical-path delay. This issue could likely be resolved after placement by moving or duplicating logic elements that are causing the routing and delay increase, as described in [20]; however, this feature is not supported in our experimental CAD flow.

4.6.2 Clocks per Rib (W_{rib})

We next consider the impact of varying the number of wires in each rib within a region. Intuitively, the placement tool has to ensure that the total number of clocks used by all logic blocks lying on a single rib is no larger than W_{rib} . The higher this value, the easier the placement task becomes, however, a larger value of W_{rib} means the clock network will be larger and consume more power.

Figure 4.16 illustrates the clock area when W_{rib} is varied from 6 to 14 and the baseline values from Table 4.1 are used for the remaining parameters. The figure shows that the clock network area increases significantly as W_{rib} increases. This is due to an increase in the number of the rib-to-logic block (Rib-LB) switches and spine-to-rib (Spine-Rib) switches. The area of remaining clock network connections remains unchanged and is mostly due to logic block-to-logic element (LB-LE) switches.

Table 4.8 presents the overall energy per cycle when W_{rib} is varied from 6 to 14. The results from this table show that the overall energy decreases significantly (6%) when W_{rib} is increased. In most cases, the majority of the savings are gained when W_{rib} is increased by 1 or 2 tracks beyond the minimum W_{rib} that produces a legal solution. Correspondingly, Table 4.9 shows the impact of the W_{rib} parameter on routing energy and the critical-path delay and shows that the overall energy increase originates from the routing energy increase and that the impact on delay from the W_{rib} constraints is not as significant as the impact from the W_{lb} constraint.

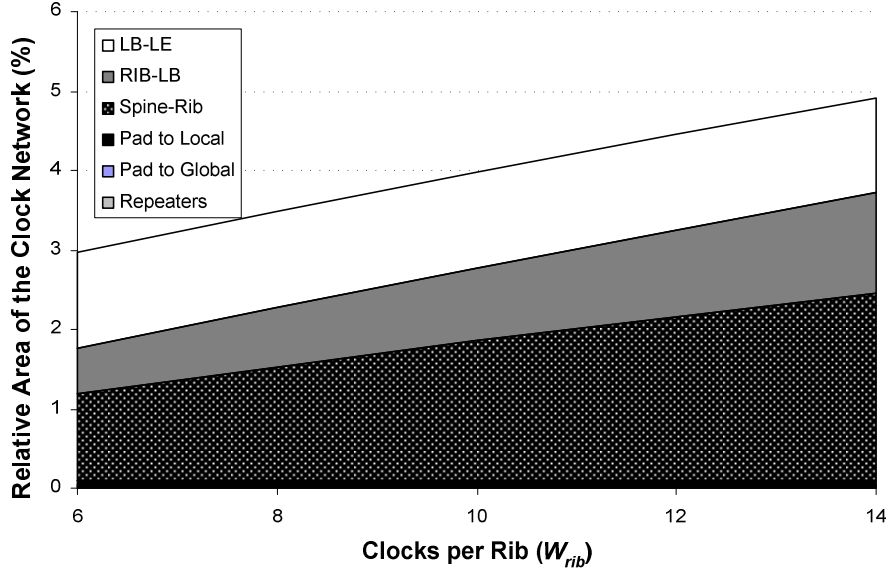


Figure 4.16: Clock network area vs. W_{rib} .

Table 4.8. Overall energy per cycle vs. the number of clocks per rib (W_{rib}).

Benchmark	Energy per cycle (nJ)								
	$W_{rib}=6$	$W_{rib}=7$	$W_{rib}=8$	$W_{rib}=9$	$W_{rib}=10$	$W_{rib}=11$	$W_{rib}=12$	$W_{rib}=13$	$W_{rib}=14$
10lrg	6.80	6.44	6.37	6.33	6.34	6.33	6.34	6.35	6.34
10lrg_reordere	6.49	6.34	6.69	6.46	6.46	6.47	6.46	6.46	6.46
15med	3.03	2.83	2.84	2.78	2.84	2.89	2.86	2.83	2.84
1lrg40sml	5.83	4.43	4.45	4.40	4.69	4.25	4.83	4.39	4.31
2lrg4med80sm	-	-	-	-	11.84	11.57	11.50	11.38	11.39
30med	2.60	2.55	2.55	2.57	2.58	2.55	2.54	2.56	2.50
30mixedsize	6.37	6.41	6.51	6.51	6.47	6.35	6.38	6.39	6.44
30seq20comb	7.33	7.33	7.33	7.14	6.02	5.99	5.99	7.10	5.97
3lrg50sml	4.89	5.02	4.89	4.92	4.84	4.85	4.85	4.86	4.75
40mixedsize	8.59	7.54	7.56	7.55	7.58	7.94	7.98	7.47	7.53
4lrg4med16sm	6.37	6.40	6.42	6.28	6.37	6.25	6.17	6.23	6.35
4lrg4med4sml	4.44	4.36	4.38	4.43	4.37	4.41	4.41	4.42	4.42
50mixedsize	9.68	9.52	9.54	9.49	9.59	9.50	10.56	9.57	9.60
5lrg35sml	5.99	5.96	6.62	6.04	5.93	6.26	6.01	5.97	6.10
60mixedsize	8.86	9.01	8.96	8.88	8.85	8.91	8.94	8.91	9.09
6lrg60sml	-	-	11.95	11.35	11.31	11.22	11.15	11.29	11.34
70s1423	-	-	11.42	11.21	10.72	10.38	10.26	10.27	10.25
70sml	4.84	4.84	4.75	5.07	4.77	4.98	4.67	4.69	4.88
lotsaffs2	3.32	3.30	3.36	3.32	3.30	3.34	3.36	3.39	3.34
lotsaffs	2.41	2.21	2.17	2.19	2.14	2.22	2.22	2.23	2.22
Geomean	5.33	5.13	5.17	5.12	5.07	5.08	5.12	5.10	5.06
% Diff.	5.1	1.2	2.0	1.2	0.0	0.4	1.1	0.7	-0.1

Table 4.9. Clock energy, routing energy, and critical-path delay vs. clocks per rib (W_{rib}).

W_{rib}	Clock Energy (nJ)	% Diff	Routing Energy (nJ)	% Diff	T_{crit} (ns)	% Diff
6	1.74	-0.8	2.25	12.8	54.0	0.1
7	1.75	-0.5	2.06	3.2	54.0	0.1
8	1.76	0.0	2.08	4.5	54.3	0.5
9	1.76	0.1	2.04	2.6	53.9	-0.1
10	1.76	0.0	1.99	0.0	54.0	0.0
11	1.76	0.1	2.01	0.8	53.6	-0.6
12	1.75	-0.2	2.04	2.6	54.0	0.1
13	1.76	0.1	2.02	1.3	53.9	-0.2
14	1.75	-0.1	1.99	-0.2	53.9	-0.2

4.6.3 Global Channel Width (W_{global})

In this section, we consider the impact of changing the number of clock wires in the spine of the regions ($W_{local} + W_{global}$) on area, power, and delay. Intuitively, a wider spine means more clocks can be distributed within a given region, which makes placement easier but increases the area and power consumption of the clock network.

Figure 4.17 illustrates the clock network area when $W_{local} + W_{global}$ is varied from 40 to 120, keeping a 1:1 ratio between W_{local} and W_{global} (matching the baseline architecture in Table 4.1). The graph shows that increasing number clock wires in the spine does increase the area the clock network, but not as significantly as did increasing the W_{lb} and W_{rib} parameters. Most of this area increase comes from the increase in the number of spine-to-rib (Spine-Rib) switches. The number of switches between the clock sources and the spine (Pad to Local and Pad to Global) also increases, but this area is negligible, and is not apparent in the figure.

Table 4.10 presents the average overall energy, routing energy, and critical-path delay when the number of global clocks (W_{global}) is varied between 0 and 40 clocks and the baseline values are used for the remaining parameters.

Results when W_{global} is 0 and 4 are not shown since legal placements could not be found for every circuit. This emphasizes the importance of providing enough global clock network resources. The table also shows that (as long as a legal solution can be found) the number of global clock network resources only has a small impact on the overall energy and critical-path delay. Intuitively, the impact on overall energy is small since most of the power dissipated by the clock network is dissipated in the ribs and the logic blocks.

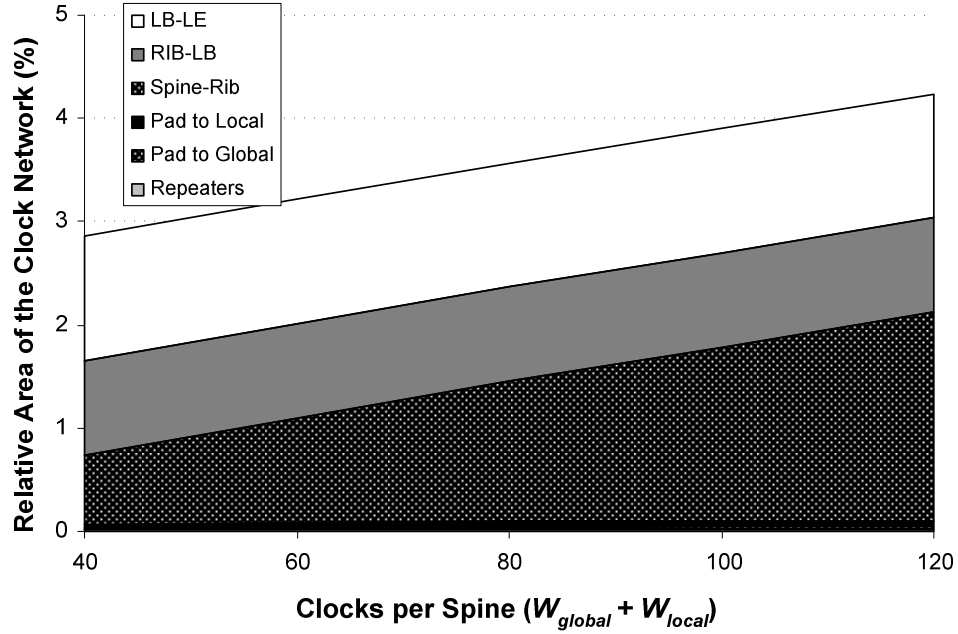


Figure 4.17: Clock network area vs. $W_{global} + W_{local}$.

Table 4.10. Energy and delay vs. global channel width (W_{global}).

W_{global}	Overall Energy (nJ)	% Diff	Routing Energy (nJ)	% Diff	T_{crit} (ns)	% Diff
52	5.30	0.0	2.07	0.0	54.8	0.0
8	5.28	-0.3	2.07	0.1	55.1	0.7
12	5.26	-0.6	2.05	-1.0	54.8	0.2
16	5.29	-0.1	2.07	0.2	54.7	-0.1
20	5.28	-0.2	2.06	-0.3	54.7	-0.1
24	5.29	-0.1	2.07	-0.3	54.8	0.0
28	5.29	-0.2	2.06	-0.3	54.7	0.0
32	5.29	-0.1	2.07	0.0	54.8	0.0
36	5.29	-0.1	2.07	0.0	54.8	0.0
40	5.29	-0.1	2.07	0.0	54.8	0.0

4.6.4 Number of Clock Regions (nx_region, ny_region)

Finally, this section examines how the number of clock regions affects power, area, and delay. As described in Section 4.3.2, we assume the FPGA is broken down into $nx_region \times ny_region$ regions, each of which contains its own set of local clocks. A clock network with many clock regions is suitable for implementing applications with many clock domains. In the best case, we can map each domain of a user circuit to a single region of the FPGA. Intuitively, this would save power because each clock would only be routed to logic blocks within the region that uses it.

Figure 4.18 shows a breakdown of the clock network when the number of clock regions is 1, 4, 9, and 16 and the baseline values from Table 4.1 are used for the remaining parameters. The figure illustrates that the increasing the number of clock regions increases the area fairly significantly and that most of the area increase is due to the increase in the number of spine-to-rib (Spine-Rib) switches.

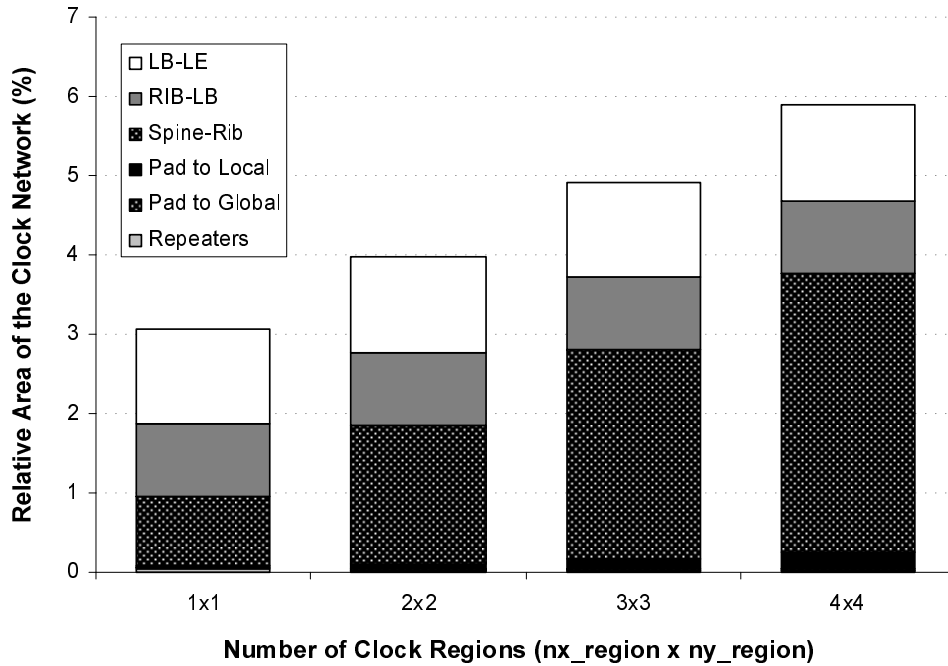


Figure 4.18: Clock network area vs. number of clock regions.

Note, however, that as we increase the number of regions and therefore reduce their size, the number of clock domains of the user circuit that are mapped to each region generally decreases. In the case where each user clock can be placed within one clock region, the number of clocks required in each is inversely proportional to the number of regions. To examine this effect, Figure 4.19 shows a breakdown of the clock network when the number of clock regions is 1, 4, 9, and 16, W_{local} is 64, 16, 7, and 4, W_{global} is 16, and the baseline values from Table 4.1 are used for the remaining parameters. Figure 4.19 shows that the area of the clock network does not increase significantly when the number of clock regions is increased, as long as the number of clocks in the spines is roughly inversely proportional to the number of clock regions.

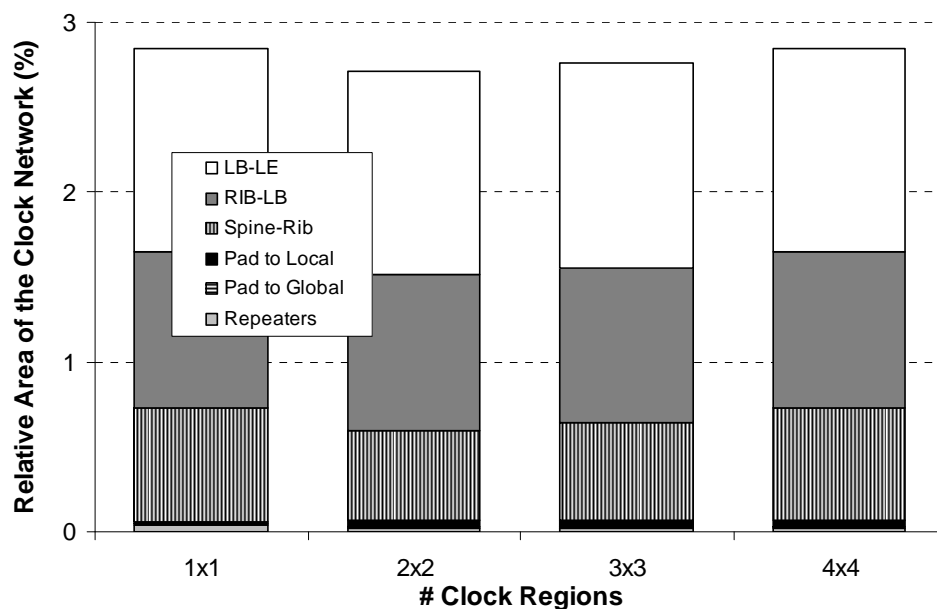


Figure 4.19: Clock network area vs. number of clock regions (with adjusted W_{local}).

Finally, we consider how the number of clock regions affects the overall energy. Table 4.11 presents the overall energy for FPGAs with 1, 4, 9, and 16 clock regions. Again, the baseline values from Table 4.1 are used for the remaining parameters. Intuitively, increasing the number of clock regions increases the amount of circuitry needed to physically implement the local connections between the clock sources on the periphery of the FPGA to center of each region. At the same time, however, increasing the number of clock regions decreases the size of each region, which reduces the number of tracks used within the clock regions and ribs.

The table shows that increasing the number of clock regions significantly reduces overall energy, with average savings of 12.1% when a clock network with 16 regions is used instead of the baseline clock network, which has 4 regions. Moreover, the table also shows that reducing the number of clock regions from 4 to 1 increases overall energy by over 30% and makes finding a legal placement more difficult, with 4 failed placements. In each of these cases, the placer failed to find a legal placement since the demand for rib tracks increased beyond the baseline value.

Table 4.11. Overall energy vs. the number of regions ($nx_region \times ny_region$).

Benchmark	Energy (nJ)			
	1 (1x1)	4 (2x2)	9 (3x3)	16 (4x4)
10lrg	8.21	6.34	5.98	5.63
10lrg_reordered	8.42	6.46	6.15	5.76
15med	3.51	2.84	2.66	2.58
1lrg40sml	-	4.69	3.98	3.91
2lrg4med80sml	-	11.8	10.2	9.51
30med	3.22	2.58	2.38	2.27
30mixedsize	8.06	6.47	5.97	5.79
30seq20comb	8.46	6.02	5.61	5.39
3lrg50sml	-	4.84	4.38	5.24
40mixedsize	9.41	7.58	6.84	6.77
4lrg4med16sml	8.02	6.37	5.78	5.58
4lrg4med4sml	5.63	4.37	4.11	3.90
50mixedsize	12.6	9.59	8.59	8.29
5lrg35sml	7.69	5.93	5.62	5.24
60mixedsize	-	8.85	8.02	7.82
6lrg60sml	-	11.3	10.2	10.2
70s1423	-	10.7	9.46	9.30
70sml	-	4.77	4.36	4.30
lotsaffs	4.57	3.30	2.97	2.79
lotsaffs2	3.03	2.14	1.96	1.87
Geomean (all circuits)	-	5.70	5.23	5.01
% Diff	-	0.0	-8.3	-12.1
Geomean (complete circuits)	6.40	4.92	4.55	4.35
% Diff	-30.1	0.0	7.6	11.7

Table 4.12: Clock energy, routing energy, and critical-path delay vs. clocks per rib (W_{rib}).

# Regions	Clock Energy (nJ)	% Diff	Routing Energy (nJ)	% Diff	T_{crit} (ns)	% Diff
4 (2 × 2)	2.07	0.0	2.17	0.0	55.0	0.0
9 (3 × 3)	1.54	-26	2.23	2.9	54.4	-1.1
16 (4 × 4)	1.36	-35	2.19	1.0	54.8	-0.4

4.7 Conclusions and Future Work

This chapter presented a new framework for describing FPGA clock networks, described new clock-aware placement techniques, and examined how the FPGA clock networks and the constraints they impose on the CAD tools affect the overall power, area, and speed of FPGAs.

The framework, which can be used to describe a wide range of FPGA clock networks, is key in this research since it provides a basis for comparing new FPGA clock networks and clock-aware CAD techniques. The challenge in creating such a framework was to make it flexible

enough to describe as many different "reasonable" clock network architectures as possible, and yet be as concise as possible. In our model, we describe clock networks using seven parameters.

After describing the framework, new clock-aware placement techniques that satisfy the placement constraints imposed by the clock network were described. Several useful clock-aware placement techniques were found. Specifically, we found that simulated annealing can be used to satisfy the placement constraints imposed by the clock network (to legalize the placement). Moreover, we found that the cost function used to minimize the usage (and therefore power) of clock network is important. Specifically, we introduced a gradual cost function which produced implementations that are 5% more energy efficient than those produced using a standard costing approach. Finally, for FPGAs with local and global clock network resources, we found that using a dynamic approach to assign global resources to clock nets improved the likelihood of producing legal placements.

Using these clock-aware techniques, an empirical study was then performed to examine the impact of the constraints imposed by the clock network. A number of important observations were made. First, the clock network should be more flexible near the clock sources and less flexible near the logic elements in order to minimize power. The results showed that adding flexibility near clock sources (near the pads) only slightly increases area and has little impact on overall energy, but makes finding legal placements significantly more straightforward. Moreover, adding flexibility near the logic elements significantly increases area, has little effect on overall energy, and can have a negative impact on critical-path delay. Another important observation is that dividing FPGA clock networks into smaller regions only slightly increases area, but significantly reduces overall energy when implementing applications with many clock domains. On average, FPGAs that have clock networks with 2x2 clock regions dissipated 12.5% more power than FPGAs that have clock networks with 4x4 clock regions for the benchmark circuits in our study.

Chapter 4 References

- [1] Actel. 2007. *ProASIC3 flash family FPGAs datasheet: device architecture* (Jan).
- [2] Altera. 2005. *Stratix II Device Handbook 1*, Chapter 2 (March).
- [3] Altera. 2006. *Stratix III device handbook 1*, Chapter 6 (Nov).
- [4] Xilinx. 2007. *Virtex-5 user guide*, Chapter 2 (Feb).
- [5] Tuan, T., Kao, S., Rahman, A., Das, S., and Trimmerger, S. 2006. A 90nm low-power FPGA for battery-powered applications. *ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*. 3-11.
- [6] Betz, V., Rose, J., and Marquardt, A. 1999. Architecture and CAD for deep-submicron FPGAs. *Kluwer Academic Publishers*.
- [7] Lamoureux, J., and Wilton, S.J.E. 2006. FPGA clock network architecture: flexibility vs. area and power. *ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*, 101-108.
- [8] Lamoureux, J., and Wilton, S.J.E. 2007. Clock-aware placement for FPGAs. To appear in the *Intl. Conf. on Field-Programmable Logic and Applications (FPL)*.
- [9] Friedman, E.G. 2001. Clock distribution networks in synchronous digital integrated circuits. *Proc. of the IEEE* 89, 5 (May). 665-692.
- [10] Nakamura, S., and Masson, G.M. 1982. Lower bounds on crosspoints in concentrators. *IEEE Trans. on Computers* 31, 12. 1173-1178.
- [11] George, V., Zang, H., and Rabaey, J. 1999. The design of a low-energy FPGAs. *Intl. Symp. on Low-Power Electronic Design (ISLPED)*. 188-193.
- [12] Li, F., Chen, D., He, L., and Cong, J. 2003. Architecture evaluation for power-efficient FPGAs. *Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*. 175-184.
- [13] Poon, K.K.W., and Wilton, S.J.E. 2005. A detailed power model for field-programmable gate arrays. *ACM Trans. on Design Automation of Electronic Systems (TODAES)* 10, 2 (April). 279-302.
- [14] Natesan, V., and Bhatia, D. 1996. Clock-skew constrained cell placement. *Intl. Conf. on VLSI Design*. 146-149.
- [15] Edahiro, M., and Lipton, R.J. 1996. Clock buffer placement algorithm for wire-delay-dominated timing model. *Great Lakes Symp. on VLSI*. 143-147.
- [16] Zhu, K., and Wong, D.F. 1997. Clock skew minimization during FPGA placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits* 16, 4 (April). 376-385.
- [17] Brynjolfson, I., and Zilic, Z. 2000. Dynamic clock management for low-power applications in FPGAs. *IEEE Custom Integrated Circuits Conference (CICC)*. 139-142.
- [18] Lamoureux, J., and Wilton, S.J.E. 2005. On the interaction between power-aware computer-aided design algorithms for field-programmable gate arrays. *Journal of Low Power Electronics (JOLPE)* 1, 2. 119-132.
- [19] Landman, B.S., and Russo, R.L. 1971. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Computers* C-20, 12. 1469-1479.
- [20] Schabas, K., and Brown, S.D. 2003. Using logic duplication to improve performance in FPGAs. *ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*, 136-142.

Chapter 5

Conclusion

This chapter begins by summarizing the contributions made in this thesis, it then discusses how the contributions relate to each other, and finally it points out the main limitations of each work and proposes future work.

5.1 Summary and Contributions

This dissertation has made three contributions to the research area of low-power FPGA design. The first contribution addresses FPGA power estimation and the next two contributions address FPGA power reduction. The contributions and key results made in each chapter of this dissertation are summarized below and listed in Table 5.1 and Table 5.2, respectively.

Chapter 2 addresses power estimation for FPGAs. In particular, it addresses the problem of estimating the switching activities of the wires within user-circuits that are implemented on FPGAs. These activities are needed by detailed power models that use the activities to calculate total FPGA power dissipation and for power-aware FPGA CAD tools, which minimize power by implementing high-activity wires using more power-efficient FPGA resources. Therefore, the accuracy of the activity estimates directly affects the accuracy of the power estimates and the savings that can be achieved by the power-aware CAD tools. On the other hand, the activity estimation also needs to be fast, so as not to burden the FPGA design flow, and should be *vectorless*, since input vectors are usually not available when user-circuits are being designed. Thus the main challenge is to find vectorless activity estimation techniques with the right balance between accuracy and speed in the context of the FPGAs.

To find the right balance between accuracy and speed, Chapter 2 begins by comparing a number of existing activity estimation techniques to determine which are most suitable. Specifically, an empirical study is performed to determine how the accuracy of the switching activities affects (1) the accuracy of FPGA power models and (2) the power savings that can be achieved by power-aware FPGA CAD tools. This is important since, to our knowledge, it is the first study that examines how accurate switching activities really need to be in the context of

FPGAs. Based on this study, the techniques that produce the most accurate power estimates and the greatest power savings in a reasonable amount of time are then combined into a new activity estimation tool called ACE-2.0 which is shown to be significantly faster than simulation-based activity estimation and more accurate than existing academic probability-based activities estimation tools and current commercial FPGA tools.

The ACE-2.0 tool incorporates a number of new and existing probabilistic and simulation-based techniques. The Lag-one model, described in [1], is used to calculate the *switching probabilities* (which are needed before calculating switching activities) for wires in all the combinational parts and some of the sequential parts of the user-circuit. If a circuit has sequential feedback (there is cycle in the sequential network), the switching probabilities in those parts are pre-calculated (prior to the remaining parts) by simulating automatically generated pseudo-random input vectors (with activities matching those specified by the user). This is faster than full simulation since only part of the circuit is simulated and switching probabilities are easier to simulate than switching activities. Once all the switching probabilities are known, the switching activities are then calculated using a novel generalization of the Lag-one model which accounts for glitches caused by uneven arrival times. By combining various techniques, the new tool produces accurate activity values for all different types of circuits, including sequential circuits with feedback loops. In commercial tools, this is only possible by performing a full simulation which is significantly slower than probabilistic estimation.

ACE-2.0 incorporates two additional techniques to further speed up the estimation. The first is an existing technique, described in [2], which involves partially collapsing the logic (as opposed to not collapsing or fully collapsing the logic) before performing the switching probability calculation. Fully collapsing the logic eliminates calculation error which occurs when there is a logical correlation between different wires of the user-circuit. However, fully collapsing logic also increases run-time exponentially with circuit size. By partially collapsing the logic, most of the error caused by signal correlation can be eliminated without exponentially increasing run-time. The second speed up technique is a novel technique which involves pruning the data structure that is used to represent the logic of the user-circuit during the probability calculation. Both techniques significantly improve the speed of the estimation with little impact of the overall accuracy. Using activities from ACE-2.0, power estimates and power savings are both within 1% of results obtained using activities from simulation. Moreover, the

new tool is 69 and 7.2 times faster than simulation for combinational and sequential circuits, respectively.

The second contribution of this thesis, which is described in Chapter 3, is a new technique which minimizes power by reducing the amount of unnecessary toggling (glitching) that occurs within the logic and routing resources of an FPGA. The technique involves adding programmable delay elements throughout the programmable fabric of the FPGA. After an application has been placed and routed on the FPGA, these delay elements are configured to align the arrival times at the input of each logic element in the FPGA to prevent glitches from being generated (by uneven arrival times). The main challenges of implementing this technique in commercial FPGAs are area and delay overhead, which are incurred by the added delay circuitry; and process, voltage, and temperature variation, which make aligning the arrival times more difficult.

A number of different implementations of the programmable delay insertion technique are considered to find an implementation with a low area and delay overhead. Specifically, the study compares five different schemes which incorporate the delay elements in different parts of the FPGA logic blocks. For each scheme, the number and flexibility of the programmable delay elements is also varied to find the implementation with the highest power savings and lowest area/delay overhead. The study finds that connecting the programmable delay elements to K-1 inputs of each K-input lookup tables produces the best results. Moreover, using delay elements with gradually decreasing sizes on each lookup table reduces area with little impact on the power savings. On average, the proposed technique eliminates 87% of the glitching, which reduces overall FPGA power by 17%. The added circuitry increases the overall FPGA area by 6% and critical-path delay by less than 1%. Furthermore, the proposed technique requires little or no modifications to the existing FPGA routing architecture or CAD flow, since it is applied after routing. This technique and variations of this technique are currently being evaluated by Xilinx Inc. for possible use in future FPGA devices.¹

The third contribution of this thesis, which is described in Chapter 4, is a study of low-power FPGA clock networks. In general, clock distribution networks dissipate a significant amount of power since they toggle every clock cycle. This is especially important in FPGA clock networks, which require added circuitry to make them flexible enough to implement the

¹ Jason Anderson, Xilinx Inc, private communication, July 2007.

clocks of arbitrary user-circuits. In addition to the power dissipated by the clock network itself, clock networks also affect the overall power consumption since they impose constraints on how the logic elements of the user circuit can be arranged within the FPGA. If the clock network is not flexible enough, this can lead to an inefficient implementation.

The FPGA clock network study has three phases. First, a new parameterized framework for defining the architecture of programmable clock networks is described. This parameterization is essential for describing and comparing new FPGA clock networks. The challenge in creating such a framework is to make it flexible enough to describe as many different "reasonable" clock network architectures as possible, and yet be as concise as possible. In our model, we describe a clock network using 7 parameters.

In the second phase, new clock-aware placement techniques that satisfy constraints imposed by the underlying FPGA clock network and reduce power by minimizing the usage of clock network resources are described. Specifically, the VPR placement tool [3] is enhanced by adding (1) a new term to the cost function that accounts for clock network usage (2) a new initial placement routine that uses simulated annealing (considering only clock usage in the cost function) to find a legal placement and (3) a new resource assignment routine that decides whether user clocks use local or global clock network resources. Two different techniques are considered for each of these three enhancements. To compare the techniques, eight different clock-aware placers (one for each possible combination) are implemented. The results show that the techniques used to produce a legal placement can have a significant influence on power and delay. On average, circuits placed using the most effective techniques dissipate 5.1% less energy and are as fast as circuits placed using the original VPR placer which does not produce legal placements. Specifically, the study found that simulated annealing can be used to satisfy the placement constraints imposed by the clock network (to legalize the placement). Moreover, the study found that the cost function used to minimize the usage (and therefore power) of clock network is important. In particular, the gradual cost function produced implementations that are more energy efficient than those produced using a standard costing approach. Finally, for FPGAs with local and global clock network resources, the results show that using a dynamic approach to assign global resources to clock nets improved the likelihood of producing legal placements.

Finally, the third phase is an empirical study that examines the affect of changing the flexibility at various levels of programmable clock networks on power, area, and delay. The study shows that FPGAs with an efficient clock network dissipate significantly less power than other FPGAs. The results show that clock networks should have more flexibility near the clock sources (on the periphery) and less flexibility near the logic elements in order to minimize power and area. Moreover, the results show that adding flexibility near clock sources only slightly increases area and has little affect on overall energy, but makes finding legal placements significantly more straightforward. Furthermore, adding flexibility near the logic elements significantly increases area, increases overall energy, and can even have a negative impact on critical-path delay. Another important observation is that dividing FPGA clock networks into smaller regions only slightly increases area, but significantly reduces overall energy when implementing applications with many clock domains. On average, FPGAs that have clock networks with 2x2 clock regions dissipated 9.4% more power than FPGAs that have clock networks with 4x4 clock regions for the benchmark circuits in our study.

Table 5.1: Breakdown of the three contributions.

Research Area	Contributions
Activity Estimation	<ul style="list-style-type: none"> - first study to examine how accurate switching activities need to be in the context of FPGAs - combine probabilistic and simulation-based techniques to handle different circuit types - new generalization of the Lag-one model to account for glitching - new BDD pruning technique - new vectorless activity estimation tool called ACE-2.0
Glitch Minimization	<ul style="list-style-type: none"> - new glitch minimization technique for FPGAs - examined 5 different programmable delay insertion schemes to minimized area/delay overhead - proposed techniques to cope with process variation
Clock Distribution Networks	<ul style="list-style-type: none"> - new parameterized framework of FPGA clock networks - first academic clock-aware placement tool - clock network architecture study

Table 5.2: Summary of the key results.

Research Area	Key Results
Activity Estimation	<ul style="list-style-type: none"> - the Lag-one model can be generalized to model glitching - BDD pruning speeds up activity estimation with little affect on accuracy - power model estimates and power-aware CAD savings are both within 1% compared to simulation - 69X and 7.2X speedup vs. simulation
Glitch Minimization	<ul style="list-style-type: none"> - programmable delay elements can be used to remove most of the glitching in FPGAs - reduced glitching by 87% - reduced power by 17% with an area overhead 6% and less than 1% speed overhead
Clock Distribution Networks	<ul style="list-style-type: none"> - clock-aware placement techniques can have a significant impact on overall power (5.1% savings compared to baseline VPR) - simulated annealing can be used to legalize a placement with clock network constraints - assigning clock network resources dynamically improves the chances of finding a legal placement - clock networks should be more flexible near the sources and less flexible hear the flip-flops to reduce power - dividing clock networks with a rib-and-spine topology into smaller regions significantly reduces overall power (in our study, clock architectures with 16 regions are 11.7% more efficient than 4 regions)

5.2 Relationship between Contributions

Each of the contributions that have been described in this thesis is related to reducing power consumption in FPGAs. This section discusses the relationship that exists between some of the contributions.

First, consider the relationship between switching activity estimation, described in Chapter 2, and glitch minimization, described in Chapter 3. Glitching is an important part of switching activity and one of the main challenges in activity estimation is accurately modeling glitching. In Chapter 2, glitching is modeled using a generalization of the Lag-one model, which uses circuit delay information to determine the likelihood that glitches will be generated by uneven arrival times. This is directly related to the technique used in Chapter 3, which minimizes glitching by inserting delays to align the arrival times. Using ACE-2.0, the glitch minimization effect of aligning the arrival times is captured, which is not true of earlier activity estimation tools.

Second, consider the relationship between switching activity estimation and the clock network study, described in Chapter 4. Activity estimation techniques are not required for clocks since the clocks simply toggle twice every cycle (unless they are turned off) and the clock frequency is defined by the designer and the critical-path delay of the application.

Finally, consider the relationship between the glitch minimization technique and the clock network study. The glitch minimization technique reduces the power dissipated within the general purpose logic and routing resources of the FPGA but it does not affect the power dissipated within the clock network resources, which do not have glitching. However, the flexibility of the clock network can affect how efficiently the remaining logic and interconnect of the application can be implemented. The impact that the clock network has on the efficiency of the remaining circuitry becomes less significant when the switching activities of the application are reduced (using glitch minimization) since less power is dissipated within these other resources. However, this point is subtle and would not need to be considered when designing FPGA clock networks.

5.3 Limitations and Future Work

This section points out the main limitations of the work described in this thesis and proposes future work to address these limitations. These limitations and the corresponding future work are summarized in Table 5.3.

The FPGA activity estimation work described in Chapter 2 has three notable limitations that merit further attention. The first limitation is that input data correlation is not considered in the study. In the first part of the work, which involves an empirical study to compare existing activity estimation techniques to find which are most suitable for FPGA CAD tools, the experimental framework employs pseudo-random input vectors rather than real input vectors. The pseudo-random input vectors are neither spatially nor temporally correlated. Real input vectors, however, can be correlated and this correlation affects switching activities within the application. Moreover, the new FPGA activity estimation tool described in the second part does not incorporate techniques to model the effect of correlated input vectors (although correlation within the circuit is modeled). Therefore, to improve the overall accuracy of the tools and the validity of the results, the effect of correlated inputs vector should be modeled as in [5] and realistic input vectors should be used to validate the estimated activities.

The second limitation of the work described in Chapter 2 is that it does not address system-level issues that can affect switching activities. System-level applications consist of many subcomponents which perform different tasks; during the operation of these applications, subcomponents are often idle as they wait for information from other subcomponents or for an external request. To model this system-level behavior, the activity estimation tool should support circuit hierarchy and allow the user to specify the percentage of time that each subcomponent is active (not idle). Although straight-forward, this type of system-level support would significantly improve the accuracy of power estimates for large applications with potentially idle subcomponents.

Finally, the third limitation of the activity estimation study is the use of software-based power modeling instead of physical power measurements. The power model used in the study uses FPGA architecture and circuit assumptions from the VPR CAD tool. These assumptions have been used in many academic studies and, in most cases, are representative enough of current FPGAs to draw conclusions regarding new architectures or CAD techniques. For switching activity estimation, however, the circuit-level assumptions may have a more significant impact. Specifically, the switch patterns, buffer sizing, and process technology within a device have a direct impact on the amount of glitching that is generated (by uneven arrival times) and the amount of glitching that is filtered out the (by the resistance and capacitance of the routing resources). These effects are difficult to capture in simulation. Thus, to further validate ACE-2.0, the next step in this research would be to measure power directly by taking physical measurements of the device and comparing these to the power estimates.

The FPGA glitch minimization technique described in Chapter 3 has two main limitations. The technique, which involves adding programmable delay elements within FPGAs to eliminate glitching by aligning arrival times, is susceptible to process, voltage, and temperature (PVT) variation and has not been validated in hardware. PVT variation can affect the delay characteristics of the programmable circuitry and the proposed delay elements within the FPGA. This delay variation makes it more difficult to align arrival times within the application and can cause timing violations if proper measures are not taken to ensure the added delays are not larger than expected. CAD-level techniques for dealing with local and global PVT variation are described in Chapter 3; however, these techniques are costly in terms of power savings. As an alternative, there are likely circuit-level techniques which may provide a better solution for

dealing with variation. Ideally, the delay elements would be self-calibrating. In other words, each delay element would align itself with the arrival time of the latest input signal on that LUT. Although a self-calibrating delay element would likely be larger than the delay element proposed in this thesis, the additional area could be amortized over the $K-1$ inputs of each LUT by using the latching technique described above. A number of other circuit-level techniques are likely possible.

The second limitation of the glitch elimination study is that the power savings have not been validated in hardware. As in Chapter 2, the VPR-based power model from [1] and activities estimated using gate-level simulation are used to estimate the power savings of the proposed technique. This model is very detailed and has been used in many other FPGA studies; however, the amount of power dissipated by glitches depends strongly on the circuit-level implementation of the logic and routing resources within a device. Therefore, the next step in this research is to measure power saving directly by taking physical measurements of a hardware implementation. Although fabricating a complete FPGA with programmable delay elements is not feasible in an academic context, a proof of concept approach that involves implementing sample programmable logic and routing resources with and without programmable delay elements on silicon would be feasible and adequate for validating this technique.

Finally, the main limitation of the FPGA clock network study described in Chapter 5 is that it does not consider how embedded components such as memories and arithmetic logic affects the clock network. As an example, memory blocks are typically embedded in rows or columns amid the programmable logic blocks of the FPGA. During placement, the algorithm must keep track of these locations and ensure that logic blocks are not placed in memory locations and vice versa. Like the clock network constraints, the embedded memory constraints would likely also have an impact on the overall efficiency of the implementation. Moreover, when these features are combined, the constraints they impose would likely interact and make finding a legal and efficient placement even more challenging. Thus, the next step in this research is to incorporate our clock-aware framework and CAD techniques with new place-and-route techniques that support these other modern FPGA features and constraints.

Table 5.3: Summary of Limitations and Future Work

Research Area	Limitations	Future Work
Activity Estimation	<ul style="list-style-type: none">- ACE-2.0 does not model correlated primary input vectors- assumes input vectors with pseudo-random switching activities- does not support system-level behavior of applications (with subcomponents that can be idle)	<ul style="list-style-type: none">- obtain realistic input vectors- incorporate correlation technique from [5] for primary inputs- add system-level support to ACE-2.0 that allow user to specify idle time
Glitch Minimization	<ul style="list-style-type: none">- process variation coping techniques reduces power savings- power savings have not been physically measured on an FPGA	<ul style="list-style-type: none">- new circuit-level techniques for circuit elements- implement sample FPGA circuit with delay elements on silicon and physically measure power savings
Clock Distribution Networks	<ul style="list-style-type: none">- clock-aware placement techniques do not consider constraints imposed by other FPGA components such as memory and arithmetic logic	<ul style="list-style-type: none">- incorporate clock-aware framework and CAD technique with new place-and-route tool that considers embedded components

Chapter 5 References

- [1] R. Marculescu, D. Marculescu, M. Pedram, Switching Activity Analysis Considering Spatiotemporal Correlations, in the IEEE Intl. Conf. Computer-Aided Design (ICCAD), pp. 294-299, 1994.
- [2] B. Kapoor, Improving the Accuracy of Circuit Activity Measurement, ACM Design Automation Conference, pp. 734-739, 1994.
- [3] V. Betz., J. Rose, and A. Marquardt, Architecture and CAD for deep-submicron FPGAs, Kluwer Academic Publishers, 1999.
- [4] K.K.W. Poon, S.J.E. Wilton, A. Yan, A detailed power model for field-programmable gate arrays, ACM Trans. on Design Automation of Electronic Systems (TODAES), Vol. 10, No. 2, pp. 279-302, April 2005.
- [5] R. Marculescu, D. Marculescu, M. Pedram, Efficient power estimation for highly correlated input streams, ACM/IEEE Design Automation Conference, pp. 628-634, 1995.

Appendix A

Benchmark Circuits with Multiple Clock Domains

Benchmark circuits with multiple clock domains are used in the empirical clock network study described in Chapter 5. Existing academic benchmark circuits typically have only one clock domain since developing large system-level circuits is labor intensive and commercial vendors are reluctant to release their intellectual property. As a solution, we combine a number of single domain benchmark circuits to form larger multi-domain circuits. To connect the circuits, we begin by determining how many external pins the meta-circuit should have using Rent's rule [1]. Rent's rule relates the number of pins (N_p) to the number of gates (N_g) in a logic design using the following expression:

$$N_p = N_{PI} + N_{PO} = K_p \cdot N_g^\beta$$

where β is the Rent's constant and K_p is a proportionality constant. The values of the two constants vary for different circuit types. β is lower for circuits with pin counts that do not increase quickly when the size of the circuit increases, such as static memory. K_p is lower for circuits with fewer pins in general. In [1], an empirical study found that β varies between 0.12 and 0.63, and K_p varies between 0.82 and 6 for chip-level designs. For circuits implemented on gate arrays, β was 0.5 and K_p was 1.9. These are the values used in this chapter.

The number of primary inputs (N_{PI}) and primary outputs (N_{PO}) of the meta-circuit are then determined from the following constraints:

1. There must be at least one sink for each source.

$$N_I + N_{PO} \geq N_O + N_{PI}$$

$$N_I + N_{PO} \geq N_O + (N_P - N_{PO})$$

$$N_{PO} \geq \lceil (N_O - N_I + N_P) / 2 \rceil$$

2. The number of primary inputs cannot be greater than the number of input IP inputs (N_I).

$$N_{PI} \leq N_I$$

$$(N_P - N_{PO}) \leq N_I$$

$$N_{PO} \geq N_P - N_I$$

3. The number of primary inputs must be greater than zero.

$$N_I > 0$$

$$N_P - N_{PO} > 0$$

$$N_{PO} < N_P$$

Therefore,

$$\text{MAX}(\lceil (N_O - N_I + N_P) / 2 \rceil, N_P - N_I) \leq N_{PO} < N_P$$

This expression bounds the number of primary outputs. The lower bound is defined by constraint 1 and 2, and the upper bound is defined by constraint 3. Assuming the lower bound is smaller than the upper bound, a valid solution is to choose the middle between the two bounds as shown below.

$$N_{PO} = (\text{MAX}(\lceil (N_O - N_I + N_P) / 2 \rceil, N_P - N_I) + N_P) / 2$$

$$N_{PI} = N_P - (\text{MAX}(\lceil (N_O - N_I + N_P) / 2 \rceil, N_P - N_I) + N_P) / 2$$

After determining N_{PI} and N_{PO} , all the sinks and sources of the meta-circuit are listed in arrays as illustrated below.

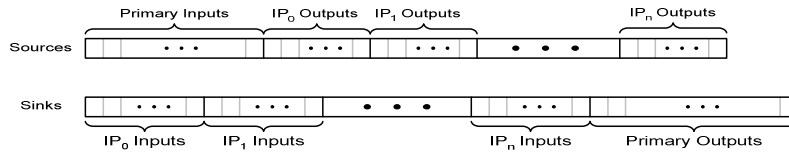


Figure A.1: Stitch connection scheme.

Finally, each sink is assigned a source using the algorithm in Figure A.2. Two synchronizing flip-flops are inserted between connections from IP outputs to IP inputs since the IP cores use different clocks.

```

j = 0;
for (i = 0; i = NI + NPO; i++) {
    sink = sinks[i];
    source = sources[j % (NO + NPI)];
    connections{sink} = source;
}

```

Figure A.2: Pseudo-code of benchmark circuit stitching algorithm.

The above technique was used to combine MCNC benchmarks into larger multiple clock domain benchmark circuits. Table A.1 summarizes the attributes of each benchmark circuit used in this thesis. Column 2 specifies the total number of LUTs each benchmark circuit; columns 3 to 5 specify the number of flip-flops in the sub-circuits, between the sub-circuits, and in total, respectively; columns 6 and 7 specify the number of primary inputs in the sub-circuits and in the benchmark circuit, respectively; and columns 8 and 9 specify the number of primary outputs in the sub-circuits and in the benchmark circuit, respectively. A list of the specific sub-circuits used in each benchmark circuit is available on the internet (see Appendix B for URL).

Table A.1. Benchmark Circuit Attributes.

Benchmark Circuit	# LUTs	# FFs			# PIs		# POs	
		Sub-circuit	Interface	Total	Sub-circuit	External	Sub-circuit	External
10lrg	14320	3316	1464	4780	831	101	887	158
10lrg_reordered	14320	3316	1464	4780	831	101	887	158
15med	5710	1367	994	2361	587	91	568	90
1lrg40sml	9452	2317	862	3179	569	138	559	138
2lrg4med80sml	15289	3403	5716	9119	3059	245	1704	174
30med	5210	492	1382	1874	795	112	592	92
30mixedsize	12537	1343	2056	3399	1163	150	993	133
30seq20comb	16398	1342	1642	2984	992	191	803	165
3lrg50sml	10574	2861	1572	4433	925	156	904	155
40mixedsize	12966	2758	1820	4578	1071	161	941	146
4lrg4med16sml	11139	3278	1748	5026	1000	126	1006	133
4lrg4med4sml	8873	2598	1270	3868	739	107	721	105
50mixedsize	16477	3685	1996	5681	1177	179	1155	177
5lrg35sml	11397	2222	1424	3646	771	60	940	231
60mixedsize	15246	1731	3486	5217	1900	198	1479	168
6lrg60sml	15838	3673	4364	8037	2388	218	1554	165
70s1423	13440	5180	2128	7308	1260	251	420	124
70sml	9301	1134	1264	2398	801	170	735	162
lotsaffs	7712	3123	632	3755	367	51	477	162
lotsaffs2	9309	3609	958	4567	597	118	616	138

Appendix A References

- [1] B.S. Landman and R.L. Russo, "On a pin versus block relationship for partitions of logic graphs," IEEE Trans. on Computers C-20, 12, pp. 1469-1479, 1971.

Appendix B

Instructions for Downloading Source Code

Table B.1 provides the URL for downloading source code and other research tools presented in this thesis. Each tool has a website where the tool can be downloaded using a standard web browser.

Table B.1. Downloading location of research tools.

Research Tool	Location
ACE-2.0 Activity Estimator	http://www.ece.ubc.ca/~julienl/activity.htm
Multiple Clock Domain Benchmark Circuits	http://www.ece.ubc.ca/~julienl/benchmark.htm
Clock-Aware VPR Placer	http://www.ece.ubc.ca/~julienl/clock.htm