Power Estimation for Diverse Field Programmable Gate Array Architectures

by

Jeffrey Goeders

BASc, University of Toronto, 2010

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

Master of Applied Science

in

THE FACULTY OF GRADUATE STUDIES
(Electrical and Computer Engineering)

The University Of British Columbia
(Vancouver)

October 2012

Abstract

This thesis presents a new power model, which is capable of modelling the power usage of many different field-programmable gate array (FPGA) architectures.

FPGA power models have been developed in the past; however, they were designed for a single, simple architecture, with known circuitry. This work explores a method for estimating power usage for many different user-created architectures. This requires a fundamentally new technique. Although the user specifies the functionality of the FPGA architecture, the physical circuitry is not specified. Central to this work is an algorithm which translates these functional descriptions into physical circuits. After this translation to circuit components, standard methods can be used to estimate power dissipation.

In addition to enlarged architecture support, this model also provides support for modern FPGA features such as fracturable look-up tables and hard blocks. Compared to past models, this work provides substantially more detailed static power estimations, which is increasingly relevant as CMOS is scaled to smaller technologies. The model is designed to operate with modern CMOS technologies, and is validated against SPICE using 22 nm, 45 nm and 130 nm technologies.

Results show that for common architectures, roughly 73% of power consumption is due to the routing fabric, 21% from logic blocks and 3% from the clock network. Architectures supporting fracturable look-up tables require 3.5-14% more power, as each logic element has additional I/O pins, increasing both local and global routing resources.

Preface

The work presented in this thesis will be published in the following conference proceedings:

Jeffrey Goeders and Steven Wilton. VersaPower: Power Estimation for Diverse FPGA Architectures. In *International Conference on Field Programmable Technology*, December 2012. Accepted. (Poster Presentation)

Portions of this publication are used in all chapters of this thesis. I was solely responsible for the code development of this work, as well as performing the necessary experiments. I am the primary author of this publication, and wrote the majority of the paper. I collaborated with my supervisor, Steve Wilton, in designing this work, and he provided instruction and guidance throughout the development. He also aided in revising and editing the above paper.

Table of Contents

List of Tables

List of Figures

Acronyms

ASIC application-specific integrated circuit.

BLE basic logic element.

CAD computer-aided design.

CLB configurable logic block.

FPGA field-programmable gate array.

HDL hardware description language.

ITRS International Technology Roadmap for Semiconductors.

LUT look-up table.

PLD programmable logic device.

PTM Predictive Technology Model.

RTL register-transfer level.

SPICE Simulation Program with IC Emphasis.

VLSI very-large-scale integration.

VPR Versatile Place and Route.

VTR Verilog-to-Routing.

Acknowledgments

First and foremost, I would like to thank my family. My wife, Jessie, for her encouragement to complete this degree, and her support and patience during busy times. My parents, for their examples, and their emphasis on the value of education.

This work would not be possible without the guidence, instruction and example of my supervisor, Steve Wilton. He is always generous with his time, providing ideas, editing papers, and answering questions.

I would also like to thank my labmates for their help and suggestions: Kyle Balston, Assem Bsoul, Stuart Dueck, and Eddie Hung, as well as others who answered questions, and gave advice: Jason Luu, Guy Lemieux, Jonathan Rose, and the VTR team.

I would also like to thank NSERC and Altera for funding this research.

Chapter 1

Introduction

1.1 Motivation

Power dissipation has become a first-class concern in the development of new integrated circuits. For the past 9 years, the International Technology Roadmap for Semiconductors (ITRS) has identified power consumption as one of the top three challenges facing semiconductor development [1], and in the latest report [2], it states that power management will continue to be a grand challenge in the foreseeable future. In the past, dynamic power was the primary concern, growing rapidly as circuit operating frequencies increased. However, as transistor technologies have scaled down, static power from leakage currents has become equally important. In fact, the ITRS predicts that in the long-term, static power increases will lead to a major industry crisis, threatening the survival of CMOS technologies [2].

The ITRS classifies devices into three types: 1) high-performance, 2) cost-performance, and 3) portable, or battery-powered. Power is a major design factor in all of these categories. In high-performance, such as desktop processors, the power must not exceed the rate at which heat can be removed from the device. In cost-performance, the goal is to reduce the energy cost per computation; for example, in server farms [3]. In portable applications, an area of rapid growth, power is minimized in order to extend battery life. In addition, the ITRS cites the need to reduce global energy usage as a motivating factor in reducing power consumption of electronic devices [1].

This work focuses on power consumption of one type of integrated circuit, field-programmable gate arrays (FPGAs). FPGAs are a type of user programmable computer chip. They contain many programmable logic blocks that can be used to implement circuits with hundreds of thousands of logic gates. These logic blocks are surrounded by a vast network of configurable routing segments. Together, the logic and routing allow FPGAs to implement almost any type of digital circuit.

This flexibility has made FPGAs a popular choice in many different circuit applications, including both high-performance [4] and cost-performance [5, 6] scenarios. However, the flexibility comes at a cost; the generic logic and routing in an FPGA have a large overhead, requiring a larger circuit and more power than an application-specific integrated circuit (ASIC). One study found that FPGAs require over 10 times the power of an equivalent ASIC [7].

The power usage of an FPGA depends on two main factors: 1) the FPGA architecture, which includes how the FPGA is designed and which CMOS tech-

nology is used, and 2) the user circuit, including how the circuit is mapped to the FPGA resources. This mapping of the user circuit to the FPGA is performed by computer-aided design (CAD) tools. Recent years have seen numerous techniques for creating power-efficient FPGA architectures [8, 9], power-aware CAD algorithms [10, 11], and low-power applications [12]. As the capacities of FPGAs continue to grow, the importance of power efficient operation will only increase.

In order to evaluate new FPGA architectures, or new CAD algorithms, researchers need a customizable CAD flow that supports experimental architectures. Furthermore, if researchers want to investigate how these architectures and algorithms affect power dissipation, an accurate power model, typically integrated into the CAD tools [13, 14], is required. Although vendor tools can quickly estimate the power dissipation of an application on an existing FPGA, they cannot be used to estimate the power of novel architectures or new low-level mapping algorithms.

Versatile Place and Route (VPR) [15], an academic, open source, FPGA CAD tool has become the most popular tool used in the academic community to test experimental FPGA architectures and CAD algorithms. When VPR was first released in 1997, it supported only basic FPGA architectures. Over time, new FPGA architectures and algorithms have been developed by industry and academia, and many have been integrated into VPR.

Throughout these years, there have been power models that have been developed, which integrate with VPR. These include the Poon model [13], the Jamieson model [14], and the Li model [16]. These models use probabilistic power estimation [17], as opposed to simulation, with a switch-level abstraction of the FPGA

circuit. This allows for fast power estimations that are sufficiently accurate to evaluate architectural trade-offs. These models were integrated with VPR 4.3 and VPR 5.0, which supported only simple FPGA architectures with a handful of configuration parameters.

Recently, a new version of VPR has been developed, VPR 6.0. This new version of VPR is a significant advance over its predecessors; among other improvements, it includes an overhaul of the types of FPGA architectures than can be supported. The tool now supports an architecture description language that users can leverage to test custom FPGA architectures, with support for complex logic blocks. Users can define a hierarchy of block types, which can be used to describe traditional FPGA architectures, as well as more modern features, such as fracturable look-up tables (LUTs). In addition, user-defined heterogeneous hard-blocks, such as memories and multipliers, are now supported. One demonstration of functionality included the evaluation of a floating point unit within an FPGA architecture [18].

This new tool opens the door for research into many more types of FPGA architectures. However, its powerful architecture language requires a much more flexible power model than any previously developed. The past power models, while flexible enough to support different lookup-table sizes, cluster sizes, and interconnect topologies, are not able to estimate the power dissipation for *most* architectures that will be studied using the new CAD flow. This new CAD flow is the door to the investigation of much more exotic architectures than ever before,

yet without an accompanying flexible power model, this potential will not be fully realized.

Another issue with past power models is that they are outdated in the assumptions they make regarding CMOS technology. The power estimations made in those models were targeted to technologies ranging in the hundreds of nanometres. However, today's technologies range in the tens of nanometres, and many of the modelling techniques used in the past models are not accurate at this level.

1.2 Contributions of this Work

Although FPGA power models have been created in the past, they were designed for a single, simple architecture, with known circuitry. This work explores a method for estimating power usage for *many different* user-created architectures. This requires a fundamentally new technique. Although the user specifies the functionality of the architecture, the actual circuitry is not specified. These functional descriptions of FPGA architectures must be translated into physical circuits. After this translation to circuit components, power estimation can be performed.

We have implemented, verified, and used this new approach to FPGA power modelling as follows:

1. We have devloped a power model, integrated into the VPR 6.0 CAD flow, which is capable of providing power estimations for *all* architectures supported by the tool. In addition, we have added detailed static power estimation, and support for fracturable LUTs, hard blocks, and modern CMOS

Feature	Poon/ Jamieson Models	Li Model	This Work
Architectures Supported	Traditional	Traditional	User-designed
CMOS Technologies	180 nm	100 nm	22-130 nm
Fracturable LUTs	No	No	Yes
Hard Blocks	No	No	Yes
Static Power	Worst-case	Worst-case	Detailed
Autosizing of buffers and interconnect	No	No	Yes
Transistor Properties	User-provided	User-provided	Automatic

Table 1.1: Comparison of this work with past power models.

processes. Table 1.1 provides a comparison of this model to past FPGA power models.

2. The power estimations of the model are verified against SPICE simulations. Dynamic power estimates are within 20% and static power estimates are within 5%.

3. The model is used to investigate power characteristics of different FPGA architectures. This includes a breakdown of power beween FPGA components, and a study of the power characteristics of fracturable LUTs.

1.3 Challenges

In undertaking these research goals, there are two major challenges that exist. First, the new model must be flexible enough to process *any* architecture that

can be described using VPR 6.0's architecture description language. Providing such flexibility, while maintaining accuracy and ease-of-use is a significant challenge. The limited coverage of previous models meant that it was reasonable to ignore some FPGA components that did not contribute greatly to the overall power, such as local interconnect buffers, local wire capacitance, and internal multiplexer nodes. However, to accurately cover the much enlarged design space, all of these components must be accurately modelled.

Secondly, in past models, the most detailed estimations were performed for the dynamic switching power, where switch-level estimation was performed on every transistor. However, other contributers to power were estimated in less detail. Subthreshold leakage was calculated using a simple worst-case estimate, and short-circuit power was simplified to be 10% of dynamic power. However, when modelling transistors into the tens of nanometres, these secondary power components begin to play a greater role in the overall power dissipation. More detailed estimation methods are necessary to obtain acceptable levels of accuracy.

1.4 Overview of Results

This thesis includes a new power model, designed to work with architectures in versions 6.0 (and higher) of the VPR tool suite. The model is validated against 22 nm, 45 nm and 130 nm technologies. When compared to SPICE circuit simulations, the estimates of our model were within 20% for dynamic power estimations and within 5% for static power estimations.

Once verified, we use the model to study the power characteristics of different architectures. In the first of two experiments we test the power breakdown between major FPGA components for the three different technologies. Results show that for a 45 nm 6-LUT, 10 LUTs per CLB architecture, 73% of power usage is due to the routing fabric, 21% due to logic blocks and 3% due to the clock network (single clock). In the second experiment we study the effect of fracturable LUTs on overall power usage. Of particular interest is the fact that modifying the architecture to support fracturable LUTs increases power consumption by 3.5-14%.

1.5 Thesis Organization

The thesis is organized as follows: Chapter 2 provides background information on FPGAs, CAD tools, power estimation techniques, and power models. Chapter 3 provides an overview of our power model and details the architecture generator. The architecture generator creates the entire FPGA circuitry from the user-supplied architecture description. Chapter 4 details the low-level power modelling, which describes how power estimation is performed once the FPGA circuitry is known. Chapter 5 provides verification of the model, as well as experiments that test the power characteristics of different FPGA architectures. Chapter 6 concludes the document.

Chapter 2

Background

This chapter provides background information on FPGAs, their architecture, and their associated CAD tools. It also outlines different power estimation techniques, including descriptions of the estimation methods used by past power models.

2.1 FPGAs

Programmable logic devices (PLDs) are electronic components that are programmed by the user to implement a digital circuit. Unlike fixed logic devices, which are manufactured for a specific function, PLDs are standard, off-the-shelf parts, that can be used for a wide range of functions. PLDs offer many advantages over fixed logic, such as shorter design times, lower non-recurring costs, and in some devices the ability to be reprogrammed [19]. Figure 2.1 shows a taxonomy of PLDs. This work focuses on power estimation for one type of PLD, SRAM-based field-programmable gate arrays (FPGAs).

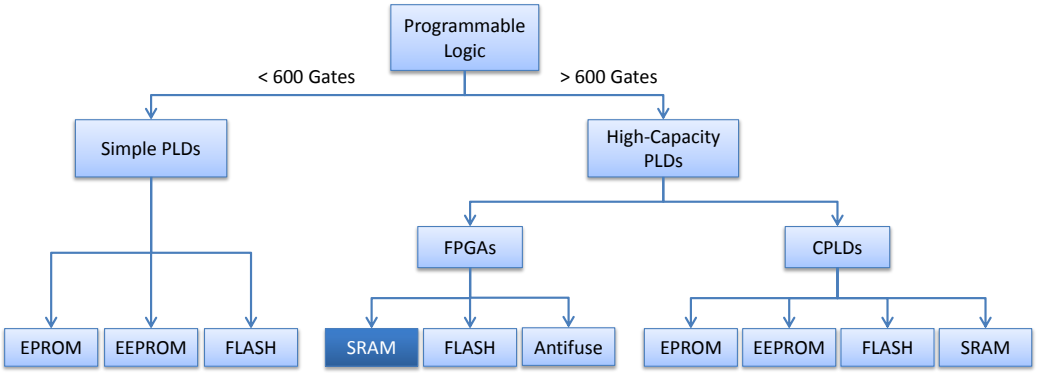


Figure 2.1: Taxonomy of PLD devices, from [20].

Field-programmable gate arrays (FPGAs) are the largest devices in the PLD family, containing thousands to millions of logic gates [20]. FPGAs consist of many fined-grained logic elements, surrounded by a very large segmented routing network. This design makes FPGAs highly flexible, and capable of implementing virtually any digital circuit. Most FPGA architectures are programmed by configuring a set of SRAM bits, which control the logic and routing of the chip [21]. This allows the FPGA to be reprogrammed as needed. The size, flexibility, and reprogrammability has led to the use of FPGAs in many applications, such as ASIC prototyping, image processing, internet infrastructure, medical devices, automotive, and others.

2.1.1 Basic Architectures

The first FPGAs were introduced by Xilinx Inc. in 1985. Early FPGA architectures contained three main components: 1) logic blocks, 2) routing, and 3) I/O blocks [22]. Logic blocks are responsible for performing the actual computation,

such as arithmetic or logical functions. Routing allows for data to be moved between logic blocks, and I/O blocks allow for data to be moved on and off the FPGA chip [21]. The following sections describe the logic blocks and routing in greater detail.

2.1.1.1 Logic Blocks

The basic functional unit within the FPGA is the look-up table (LUT). A k-input LUT has 2^k configuration bits, which are used to store the values of a k-input truth table. The inputs to the LUT control a multiplexer which chooses between the stored values (Figure 2.2). A k-input LUT can implement any k-input logic function, since any logic function can be represented in truth table form [21]. Typically, LUTs are paired with a flip-flop, which saves state, and allows for implementation of sequential circuits. The LUT and flip-flop, together with a multiplexer to select between the two outputs, are referred to as a basic logic element (BLE) (Figure 2.3).

Multiple BLEs are combined to form a logic block, also known as a configurable logic block (CLB). FPGA architectures differ in the number of BLEs per CLB. For example, the Altera Stratix V architecture uses 10 BLEs per block [23], and the Xilinx Virtex 7 architecture uses 8 BLEs per block [24]. The logic block also contains routing structures that connect the input and output pins of the logic block to the BLEs, as well as connecting the BLEs to each other. These connections are referred to as local interconnect, and are often implemented as a single

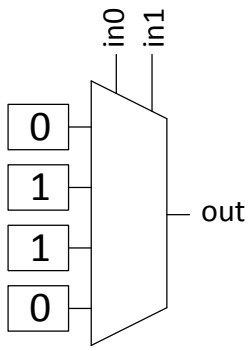


Figure 2.2: 2-input LUT, implementing the XOR function.

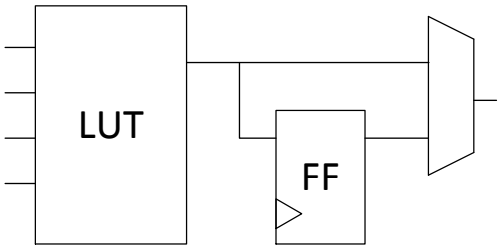


Figure 2.3: Basic logic element (BLE).

crossbar [25]. Figure 2.4 shows a logic block that contains 4 BLEs, with a local interconnect crossbar.

2.1.1.2 Routing

Most FPGAs employ an island style architecture, where the logic blocks are arranged in a grid, and are surrounded by many horizontal and vertical routing segments. Modern FPGAs usually use unidirectional routing segments, while older FPGAs used bidirectional routing [26, 27]. The routing segments are connected to each other through a *switch box*, and the logic blocks connect to the routing channels through *connection boxes* [28]. Figure 2.5 provides an illustration of a

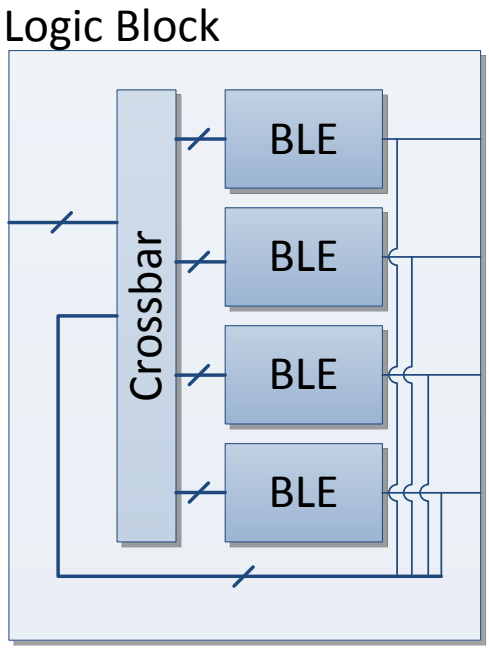


Figure 2.4: Logic block containing 4 BLEs.

simple, hypothetical FPGA architecture. Commercial FPGAs vary in their implementation details, and sometimes the connection boxes and switch boxes are combined into a single structure.

Switch boxes are located at the intersection of vertical and horizontal routing segments. They contain programmable switches that allow each wire segment to connect to multiple other wire segments, in order to route signals throughout the FPGA. The topology of these connections depends on the FPGA architecture [29, 30]. In addition, the architecture may be designed to contain longer wire segments that bypass some switch boxes [26]. Figure 2.5 shows an architecture than contains a combination of length-1 and length-2 wire segments.

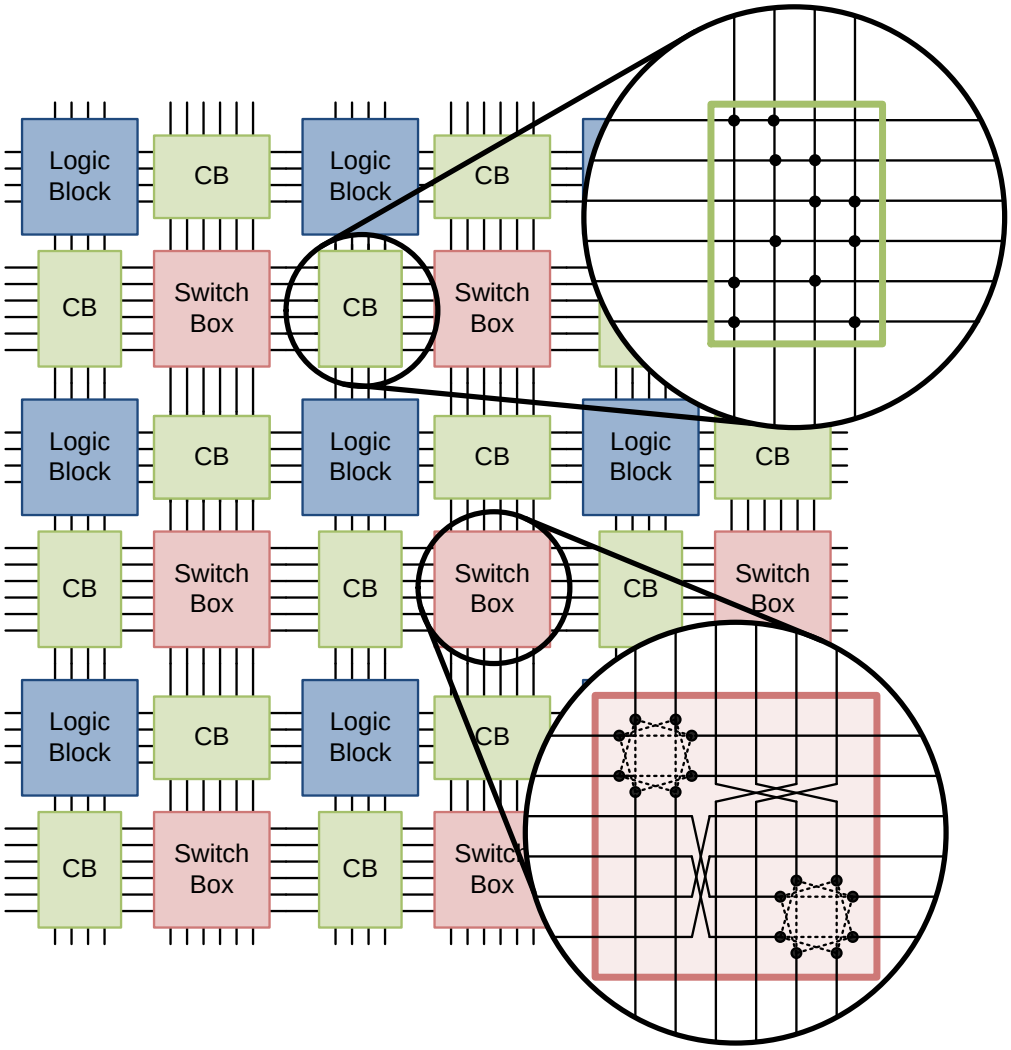


Figure 2.5: Island-style FPGA architecture, containing logic blocks, switch boxes and connection boxes. This architecture contains both length-1 and length-2 routing segments.

Connection boxes provide the connection between the routing segments and the logic blocks pins. Each logic block pin connects to some of the neighbouring routing channels, as illustrated in Figure 2.5. Real-world architectures may contain routing channels that are hundreds of segments wide, so it is not feasible for each pin to connect to all of the channels [27]. The pattern of connections is determined by the FPGA vendor. Each connection is programmable, and can be either enabled or disabled when configuring the FPGA.

2.1.2 Modern Architectures

Modern FPGA are much more complex than described above. They have evolved from a homogeneous architecture, as described previously, to heterogeneous architectures that contain many different types of blocks, and blocks with complex features.

2.1.2.1 Heterogeneous FPGAs

In heterogeneous architectures, some logic blocks are replaced by memories and multipliers, to more efficiently implement certain functions (Figure 2.6) [27]. There may be multiple types of logic blocks throughout the FPGA; for example, some blocks may contain larger or smaller LUTs [31]. Additional hard cores may also be built into the FPGA to accelerate some functions, such as ethernet interfaces or SerDes functions [23]. Some FPGAs also contain built-in processors. As an example, the Altera Cyclone 5 SoC FPGAs contain an ARM Cortex-A9 core [32].

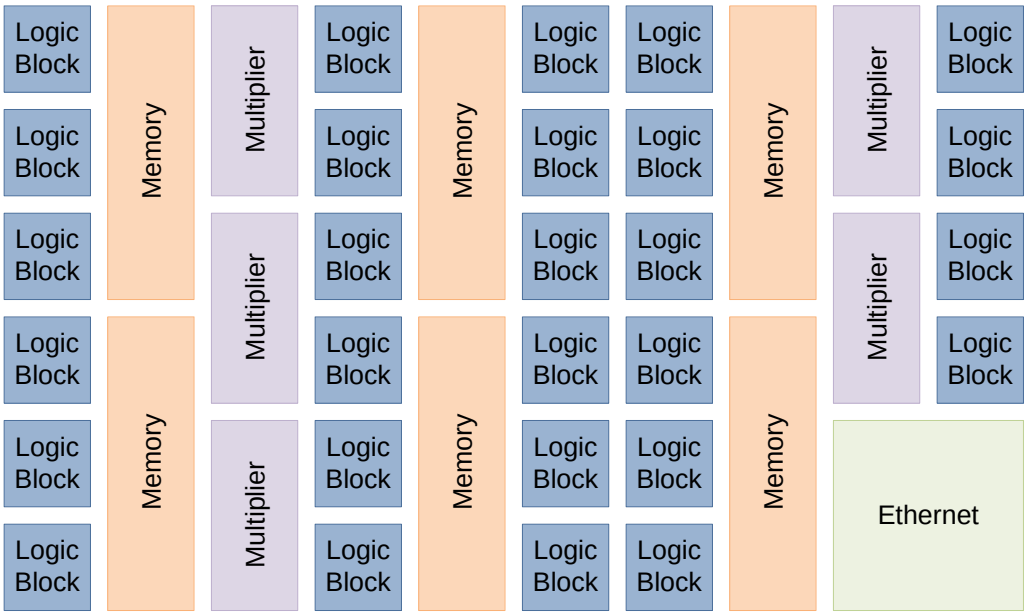


Figure 2.6: Heterogeneous FPGA architecture (routing not shown).

2.1.2.2 Complex Logic Blocks

Modern logic blocks contain additional features, such as carry-chain logic and fracturable logic. Carry-chain logic provides support for accelerating arithmetic and logical operations [33]. Fracturable logic allows LUT hardware to be split so that LUTs can implement two smaller logic functions, instead of a single large function. This is accomplished by allowing the truth table to be shared between two logic functions, splitting the multiplexer, and providing an extra output. Figure 2.7 illustrates the modifications to enable fracturable LUTs. This work provides power estimations of FPGA architectures containing fracturable LUTs; however, carry-chains are not yet supported.

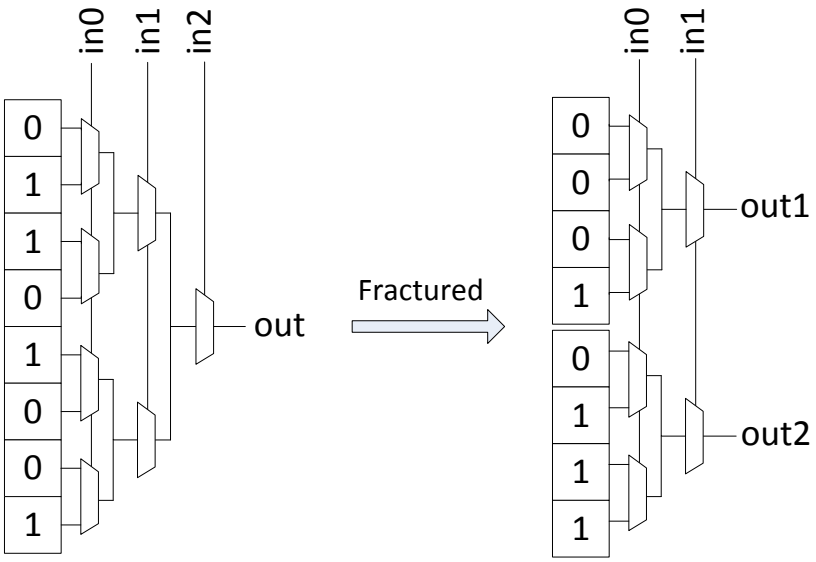


Figure 2.7: Fracturing of a LUT. On the left is a 3-input LUT, implementing a 3-input XOR function. On the right, the same hardware is used to implement two 2-input functions. Out1 implements the AND function, and out2 implements the OR function.

2.2 FPGA Computer Automated Design (CAD)

Modern FPGAs contain millions of configuration bits, making it impractical for designers to manually implement their circuits. Instead, computer-aided design (CAD) tools are used, which perform the steps required to translate a hardware description language (HDL) circuit to a set of configuration bits used to program the FPGA [34].

2.2.1 CAD Flow Steps

Figure 2.8 illustrates the steps of the FPGA CAD flow, and the following briefly describes each step.

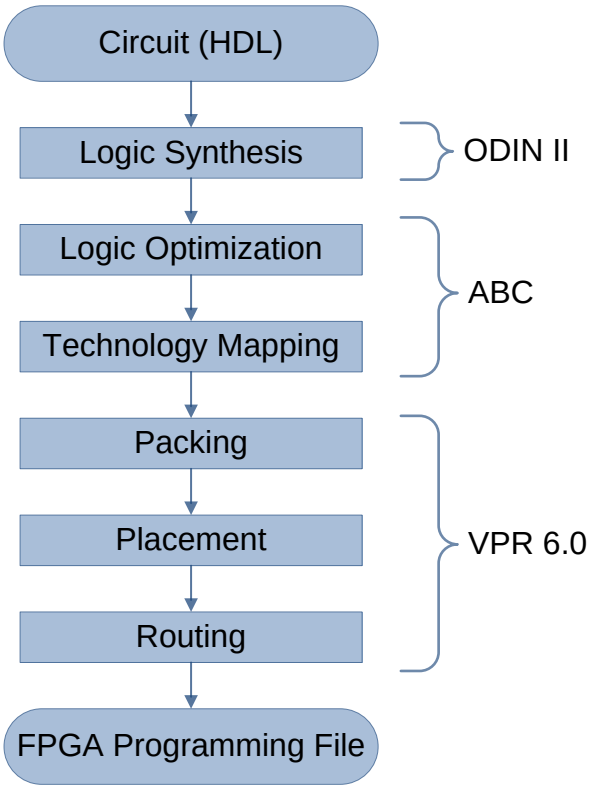


Figure 2.8: FPGA CAD flow [34], with associated tools [18].

- *Logic Synthesis* converts a circuit specified in register-transfer level (RTL) format, such as Verilog or VHDL, to a gate-level representation [35].
- *Logic Optimization*, often called technology-independent optimization, removes redundant logic and simplifies logic where possible [36].
- *Technology Mapping* converts the logic functions to fit within the LUTs. For example, if the FPGA architecture uses 4-input LUTs, all logic functions must be transformed into functions with four or less inputs.
- *Packing* groups the LUTs together into logic blocks.

- *Placement* decides where each logic block should be placed on the FPGA.
- *Routing* determines the configuration of the routing switches to provide connections between the necessary logic blocks pins.

Upon completion of this flow, the CAD tools will have determined the configuration bits for the LUTs, the local interconnect switches, and the global interconnect switches. These configuration bits are combined into a file called a *bitstream*, which is used to program the FPGA.

2.2.2 Verilog-to-Routing (VTR)

FPGA power models are typically integrated into CAD tools because the power estimation is dependent on placement and routing information [13, 14, 16]. In addition, integration of CAD and power estimation allows for the development of power-aware CAD algorithms, such as in [10].

The power model presented in this work is integrated with Verilog-to-Routing (VTR), a recently developed, academic, open-source FPGA CAD suite [18]. VTR is a collection of tools that are used to perform the full CAD flow. ODIN II [37] is used for logic synthesis, and converts a Verilog circuit to a gate-level sum-of-products representation. Next, ABC [38] performs logic optimization using logic balancing and refactoring algorithms [39], followed by technology mapping using the WireMap algorithm [40]. Finally, VPR 6.0 performs packing, placement and routing. Packing uses the AAPack algorithm [41], placement uses a simulated annealing algorithm [15, 42], and routing is performed using the Pathfinder

algorithm [43]. Together, these tools allow researchers to test experimental FPGA architectures that are not supported by commercial tools.

Past FPGA CAD tools, such as VPR 4.3 [15] and VPR 5.0 [44], supported basic homogeneous FPGA architectures with some user-supplied parameters to configure the architecture. The architecture space supported by VTR is dramatically larger. It supports heterogeneous architectures, hard blocks and complex logic blocks. The architecture is specified through a new *architecture description language*, which users can leverage to provide a much more detailed specification of the FPGA architecture. In the past users were restricted to the traditional paradigm for logic block design; however, this new language allows users to fully customize logic blocks. The language supports a hierarchical model, where logic blocks are comprised of entities. Each entity can instantiate child entities, with custom interconnect between parent and children. This allows for exotic FPGA architectures with arbitrary complexity.

2.3 Power Estimation Techniques

Power estimation techniques for very-large-scale integration (VLSI) circuits can be classified in two ways: 1) the level of abstraction, and 2) the method of estimation, either *simulation* or *probabilistic* [17, 45, 46].

2.3.1 Abstraction Levels

Figure 2.9 provides the different circuit abstraction levels at which power estimation can be performed [45]. More detailed circuit abstractions allow for more

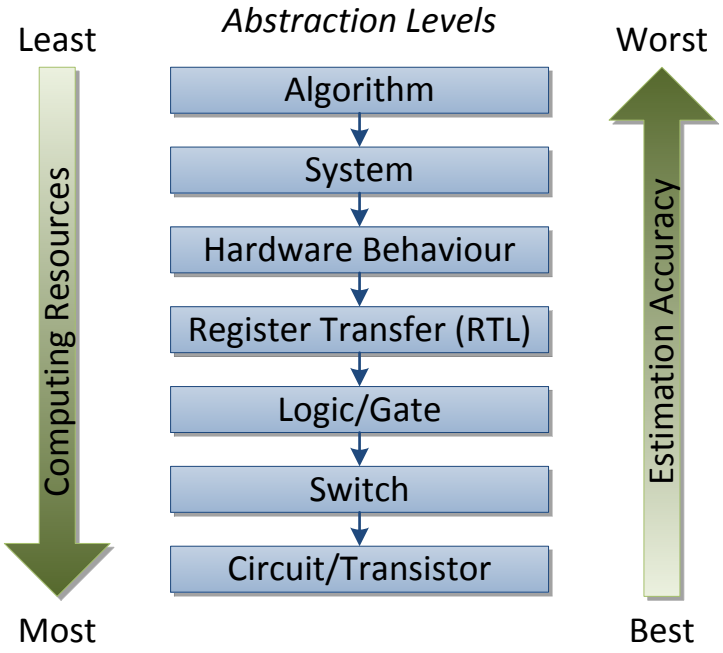


Figure 2.9: Circuit abstraction levels used for power estimation, and impacts on accuracy and resources.

accurate power estimations, but at the cost of increased computation. Likewise, higher abstractions allow for quicker estimations, but with reduced accuracy. Generally, power estimation tools operate at the gate-level or lower, as it is difficult to obtain sufficient accuracy at higher levels [46].

The most commonly used abstraction levels are *gate-level*, *switch-level*, and *circuit-level*. Circuit-level estimation requires detailed transistor models in order to determine nonlinear voltages and currents. Switch-level simplifies the circuit to a collection of nodes and transistors, where each node is modelled as a capacitance to ground, and each transistor is modelled as an on/off switch with finite

resistance. Gate-level further abstracts the design, grouping transistors into logic cells, such as NAND, NOR, etc [47].

It is possible to model FPGA hadware at the lowest abstraction levels. The GILES project [48] provides a tool for generating complete transistor layouts based on VPR architectures. Although this level of detail allows for the most accurate power estimations, it is rarely used for large circuits due to computational requirements. For large circuits, such as FPGAs, it is more feasible to perform power estimation at the switch-level or gate-level.

2.3.2 Simulation-Based Power Estimation

Power estimation through simulation is performed by providing a set of inputs to the circuit, simulating the circuit for a predetermined number of clock cycles, and measuring the power. Although simulation can provide accurate power estimations, it suffers from being computationally intensive and being highly dependent on circuit inputs, known as *strong input-pattern dependence*.

Simulation Program with IC Emphasis (SPICE) is the de facto tool for circuit-level simulation and power analysis, and provides very accurate estimations. However, the level of detail of SPICE leads to long run times as circuits increase in size, making SPICE unsuitable for very large circuits, such as FPGAs [46]. Other approaches have performed simulation at higher abstraction levels, such as switch-level or gate-level, to reduce the amount of computation [49–51].

However, even if computation can be reduced, the issue remains that the power dissipation is highly dependent on the inputs provided to the circuit; inputs that

are more active will cause the circuit to consume more power. In order to provide accurate estimation, realistic input vectors are required. It may be difficult to generate realistic input vectors, especially if the circuit is still under design [17]. One solution to this problem is to use a statistical approach, which employs a Monte Carlo simulation with randomly generated input vectors [52, 53]. This technique involves repeatedly generating random input vectors, performing gate-level simulation, and measuring the power, until the result reaches a certain accuracy and confidence level.

The statistical Monte Carlo method provides acceptably accurate estimates, without substantial computation; however, the estimate is only of the total power. It does not provide visibility to the gate level, or even groups of gates [17]. This internal visibility is a desired property as it allows designers to determine the power requirements of different architectural components. In [54], the Monte Carlo method was modified to add internal visibility, but the increase to run time was significant.

2.3.3 Probabilistic Power Estimation

The other method of power estimation is the probabilistic approach. The major advantage over simulation is that it does not require actual input vectors, only statistical properties of the inputs, and thus is characterized as being *weakly input-pattern dependent*. The behaviour of the inputs is characterized using the following two properties [17]:

1. The Signal Probability, P_1, is the long-term probability that a signal is logic-high. For example, a clock signal with a 50% duty cycle will have $P_1(clk) = 0.5$.

2. The Transition Density (or switching activity), A_S, is the average number of times the signal will switch during each clock cycle. For example, a clock has $A_S(clk) = 2$.

Once the user provides the signal probability and transition density for the inputs, activity estimation tools such as [13, 37, 55–57] can be used to determine these properties for the internal nets. These tools use simulation, static analysis, or a combination of both to determine these properties. Once these properties are known for all nets, power estimation can be performed. For example, switching power is directly proportional to the transition density, and leakage currents are dependent on the signal probabilities [58].

Although the probabilistic method is faster than simulation, and less input dependent, it is generally less accurate [45]. The signal properties used by the probabilistic method assume certain statistical behaviour, which may not be true of the actual input signals. The first assumption is that all signals are independent, referred to as *spatial independence* [17]. In reality, signals may be correlated; for example, two signals may never be logic high at the same time. Another assumption is that signal values are independent from one clock cycle to the next, referred to as *temporal independence* [17]. These behaviours cannot be captured in the signal probability or transition density metrics.

Although the probabilistic approach is less accurate than simulation, it requires much less computation resources, making it possible to use more detailed abstraction levels [45]. This allows for finer-grained power details, with less run time.

2.4 FPGA Power Estimation Tools

There have been three major power models developed for use with academic FPGA CAD tools. They are outlined in the following sections, including a description of the approach used in this work.

2.4.1 The Poon Power Model

The Poon power model [13] was developed by Kara Poon at the University of British Columbia in 1999. The tool was designed to work in conjunction with the popular academic CAD tool VPR 4.3. It supports a traditional homogeneous architecture with bidirectional routing, as described in Section 2.1.1. It includes architecture parameter support for LUT suze, LUTs per logic block, and interconnect topology. The model uses a probabilistic approach to power estimation and switch-level circuit abstraction. This work also includes ACE-1.0 [13], a tool that provides activity estimations of signals.

The Poon power model calculates switching power of all transistors in the FPGA, based on the switching density of the signals. Short-circuit power is assumed to be 10% of dynamic power. Subthreshold leakage is calculated using the equation in [59]. A worst-case is assumed for subthreshold leakage where half of

all CMOS transistors, and all pass-transistors, are assumed to be leaking. Other types of static power, such as gate leakage, are ignored.

2.4.2 The Jamieson Power Model

The Jamieson model [14] was developed by Peter Jamieson in 2009. This is a modified version of the Poon model, designed to work in conjunction with VPR 5.0. The main difference is that it supports more modern routing topologies, the most significant being unidirectional routing. Also, this model uses ODIN II [37] to perform activity estimation, instead of ACE-1.0. The model uses the same techniques as the Poon model to calculate the dynamic and static power dissipation, and supports the same architectural parameters.

2.4.3 The Li Model

The Li model [16] was developed by Fei Li, et. al. in 2005, and was designed to work in conjunction with VPR 4.3. Like the other models, it supports a traditional architecture with basic parameters. Like the others, this model uses a probabilistic approach to power estimation. Switch-level abstraction is used for the routing and clock network. To reduce computation, a higher level abstraction is used for logic blocks, where macro-modelling is used to estimate power dissipation of LUTs. Like the other models, this model uses the signal activity to calculate switching power. Short-circuit power estimation is more detailed, and is calculated by using SPICE to simulate buffers of various sizes. A similar simulation approach is used for subthreshold leakage estimation.

2.4.4 The Estimation Technique of this Work

This power model presented in this work also uses a probabilistic approach. This was chosen as it allows for fine-grained estimation without the heavy computation requirements of simulation. Although not as accurate as simulation, it is sufficient to evaluate trade-offs during architecture design [46]. The majority of this work uses switch-level abstraction, although in some cases more detailed modelling is used. Since VTR supports arbitrary user-designed logic blocks it is not feasible to use higher abstraction levels, as was done in the Li model.

The major difference between this model and past models is the type of architectures supported. Past models only supported a traditional homogeneous architecture, with limited parameters. This model supports many new features such as heterogeneous architectures, hard blocks—such as memories and multipliers—and complex logic blocks that contain fracturable LUTs. Furthermore, while the past tools supported only a few architecture parameters, the architecture engine in VTR allows for custom designed blocks with arbitrary complexity. This architecture flexibility makes this model fundamentally different than past power models. The following chapters explain this new power model.

Chapter 3

System Design and Architecture Generation

This chapter begins by providing an overview of the new power model, and an explanation of how the model is integrated into the VTR CAD flow. The remainder of the chapter details the architecture generator. This module is the largest piece of the power model. It generates the entire FPGA circuitry based on the user-described architecture.

3.1 Power Model Overview

The power model is integrated into VTR, an academic experimental FPGA CAD flow [18]. This flow was designed to allow researchers to investigate novel FPGA architectures and the associated CAD algorithms. New FPGA architectures (such as new logic block or routing structures) can be described using an architecture

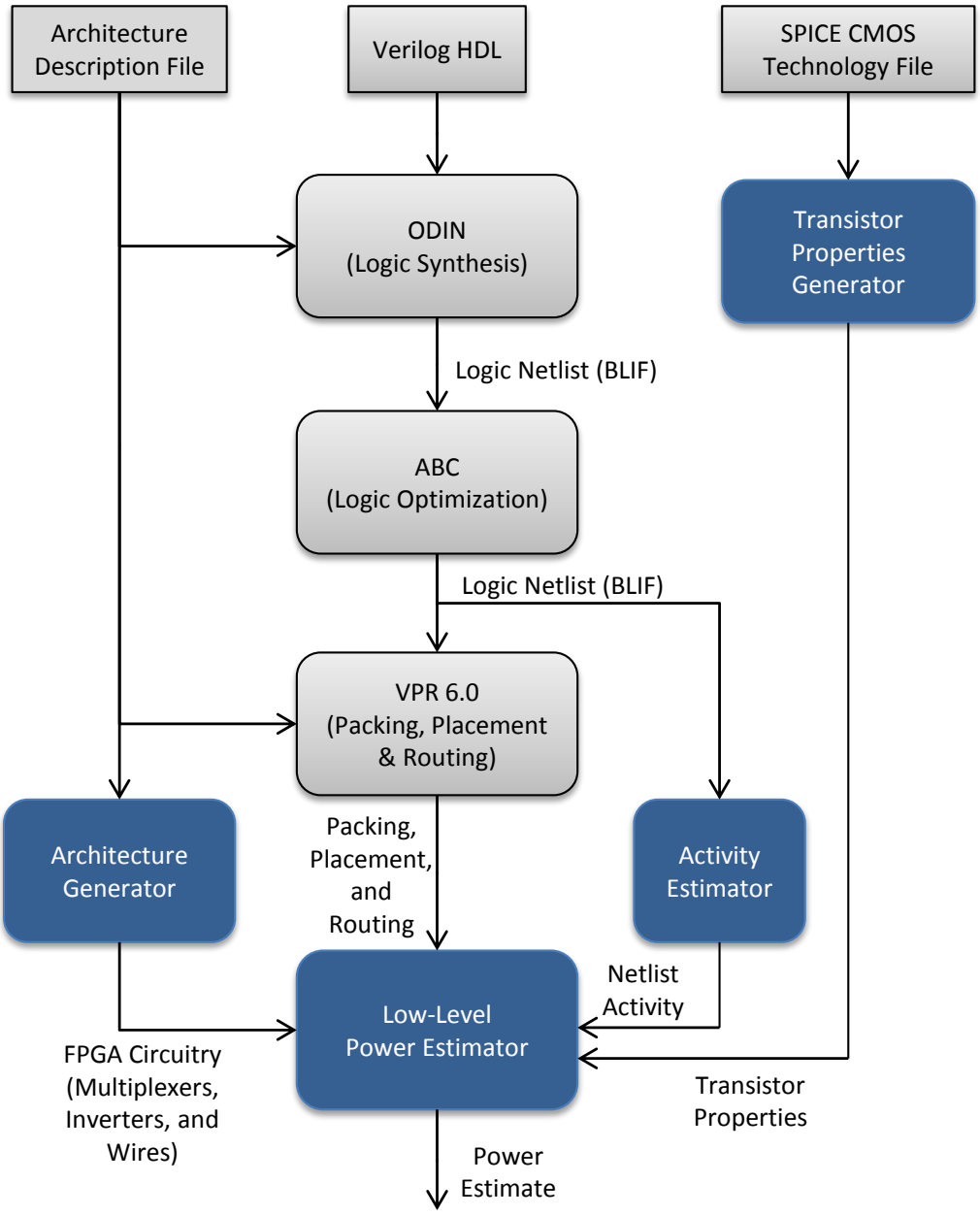


Figure 3.1: Modifications to the VTR flow for power estimation.

description language, and the CAD tools will automatically adapt to the new architecture. The original VTR flow contains detailed area and delay models which can be used to evaluate the efficiency of benchmark circuits implemented on the proposed architectures. Since the code is open-source, designers can also modify the CAD tools themselves to investigate the effectiveness of new CAD techniques, applied to novel or existing FPGA architectures.

The current VTR flow uses ODIN II [37] for Verilog synthesis, ABC [38] for logic optimization and VPR 6.0 for packing, placement and routing. Our power model adds multiple components which are integrated into the VTR flow as shown in Figure 3.1. Table 3.1 provides a listing of the model components, their purpose, and the relevant sections in this thesis.

Module	**Function**	**Section**
Architecture Generator	Generates FPGA circuitry	Section 3.2
Low-Level Power Estimator	Estimates dynamic and static power	Section 4.1
Activity Estimator	Estimates behaviour of the user circuit	Section 4.2
Transistor Properties Generator	Extracts transistor properties from SPICE simulation	Section 4.3

Table 3.1: System modules.

The first step in estimating power usage is to determine the actual FPGA circuitry. This is a difficult task, since VTR's architecture description language allows the user to create arbitrary architectures, with no limit on complexity. This task is performed by the architecture generator, which reads the architecture de-

scription file provided by the user, and generates the circuitry of the entire FPGA. This circuitry is a collection of inverters, simple multiplexers and wires, and includes details of the transistor sizes and wire lengths. All major components such as flip-flops, LUTs, switch boxes, etc. are decomposed into these basic circuit elements.

The FPGA power usage is also dependent on the behaviour of the user-provided circuit. The activity estimator analyzes the user circuit to determine its behaviour, which includes the transition density (A_S) and the signal probability (P_1) for each net. The power usage also depends on how the user circuit is implemented in the FPGA circuitry. These details are contained in the packing, placement and routing information that is output by VPR.

Power estimation is also heavily dependent on the chosen CMOS technology. The transistor properties generator uses SPICE simulation to automatically extract the necessary transistor properties, which includes transistor sizes and capacitances, leakage currents, and other characteristics.

Once the information on the FPGA circuitry, the user circuit, and the transistor properties are obtained, the actual power estimation can be performed. The low-level power estimator uses standard equations to calculate the dynamic and static power of each inverter, multiplexer and wire in the FPGA circuitry. It details how much each component type, such as switch boxes, flip-flops and crossbars, contribute to the overall power.

Our implementation combines VPR 6.0, the low-level power estimator, and the architecture generator into a single executable. However, in this document

they are described as separate entities to clearly outline their roles in the overall power estimation.

The remainder of this chapter will describe the architecture generator module. The other modules will be detailed in the following chapter.

3.2 The Architecture Generator

Complete power estimation requires transistor-level details of the entire FPGA circuitry in order to make power estimations. This includes the size and connections of every transistor, as well as wire length of all interconnect. Since there are millions of transistors in a typical experimental architecture, it is infeasible for the user to provide this information directly. The architecture generator analyzes the architecture description file supplied by the user, and determines the entire FPGA circuitry. All of the FPGA components (switch boxes, connection boxes, LUTs, flip-flops, etc.) are decomposed into inverters, simple multiplexers, and wires. The size of each transistor, and the length of each wire, is calculated automatically.

Generating this information is challenging. The architecture language provided in VPR 6.0 is extremely flexible, allowing the user to specify a wide variety of architectures. CLBs can be arbitrarily complex, consisting of a hierarchal collection of interconnect and logic elements (including facturable LUTs). The global interconnect can consist of segments of different lengths connected using different patterns. Transistor sizes, which have a first-order affect on power, de-

pend on the physical length and fanout of segments, which depends on, among other things, the number of transistors in various components of the architecture.

VPR 6.0 contains an architecture generation engine which estimates many of these low-level quantities based on the user-described architecture. However, the existing engine is limited to generating parameters needed for timing analysis and area estimation. Power estimation requires a much more complete set of quantities; buffers cannot be abstracted as delay elements, and accurate buffer sizes are needed, even for those buffers not on the critical path. This work provides a significantly enhanced architecture generation engine that provides the information needed to make accurate dynamic and static power estimates.

It is important to note that previous models in [13, 14] also contain an architecture generation engine; however, the limited architectural space supported by earlier versions of VPR means that the generator was far less flexible. Thus, these previous generators are not suitable for our purpose.

Section 3.2.1 describes the methods used to decompose the global routing into inverters, multiplexers and wires. Section 3.2.2 describes this decomposition for the logic blocks, and Section 3.2.3 for the clock network. Transistor sizes for these components depend on the physical dimensions of different entities in the FPGA. Section 3.2.4 details the algorithm we use to approximate these dimensions.

3.2.1 Global Routing

A significant portion of the transistors in an FPGA are used to implement the flexible global routing network. This network consists of switch boxes and connection

boxes, as described in Section 2.1. The switch boxes and connection boxes are comprised of buffers and multiplexers, the sizes and topologies of which depend on architectural parameters supplied by the user.

3.2.1.1 Switch Boxes

Switch boxes, which lie at the intersection of horizontal and vertical channels in an FPGA, are responsible for driving the global routing wires. There may be tens to hundreds of global wires incident to each switch box. Each global wire is driven by a multiplexer followed by a buffer [26]. The buffer drives a wire, which spans one or more grid tiles of the FPGA. At the end of the wire, is a fan-out to other switch boxes or connection boxes [60]. This is illustrated in Figure 3.2, and is referred to in this thesis as a switch box component, as many of these comprise a single switch box. The power usage of a switch box component is comprised of the power dissipation in the multiplexer (See Section 3.2.1.3), the buffer (See Section 3.2.1.4), and the wire.

The size of the multiplexer is determined by the number of fan-ins of the routing channel. These fan-ins can come from other global wires, or from outputs of logic blocks. The buffer size is a function of the capacitance it drives, which is composed of the wire capacitance and the input capacitance of the fan-out. The wire capacitance is a function of the wire length, which is determined using the methods in Section 3.2.4.

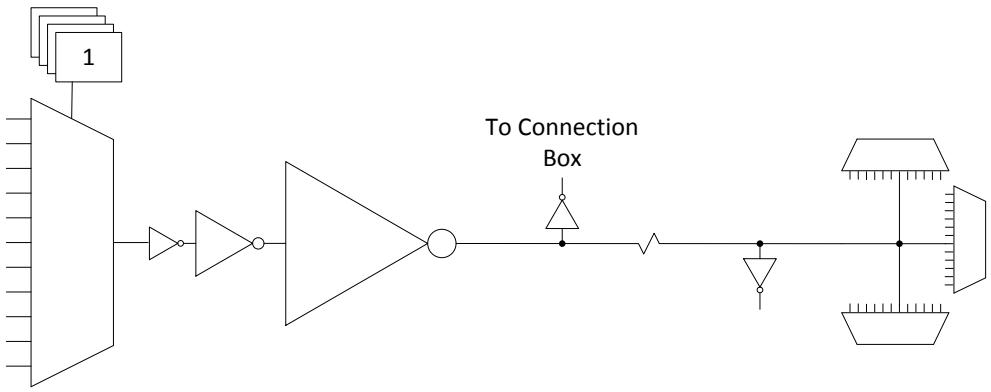


Figure 3.2: Switch box component, comprised of a multiplexer connected to a buffer, driving a wire. The wire fans out to connection boxes and other switch boxes.

3.2.1.2 Connection Boxes

Connection boxes provide the interface between the routing channels and the logic blocks. They are comprised of buffers and multiplexers. There is one multiplexer for each input pin of the logic block. A buffer connects each routing track to the multiplexers, and is needed as the track may drive many pins.

Figure 3.3 provides an illustration of a simple connection box. In a real FPGA architecture, there would be many more routing tracks, and many more CLB input pins. This potentially causes large buffers and multiplexers.

The buffers are sized according to the capacitance they drive, which is determined by the number of multiplexers they connect to. Each routing track connects to a fraction of the CLB inputs, represented by Fc_{in} in the architecture description file. Likewise, the number of inputs to each multiplexer is equal to Fc_{in} multiplied by the number of routing tracks.

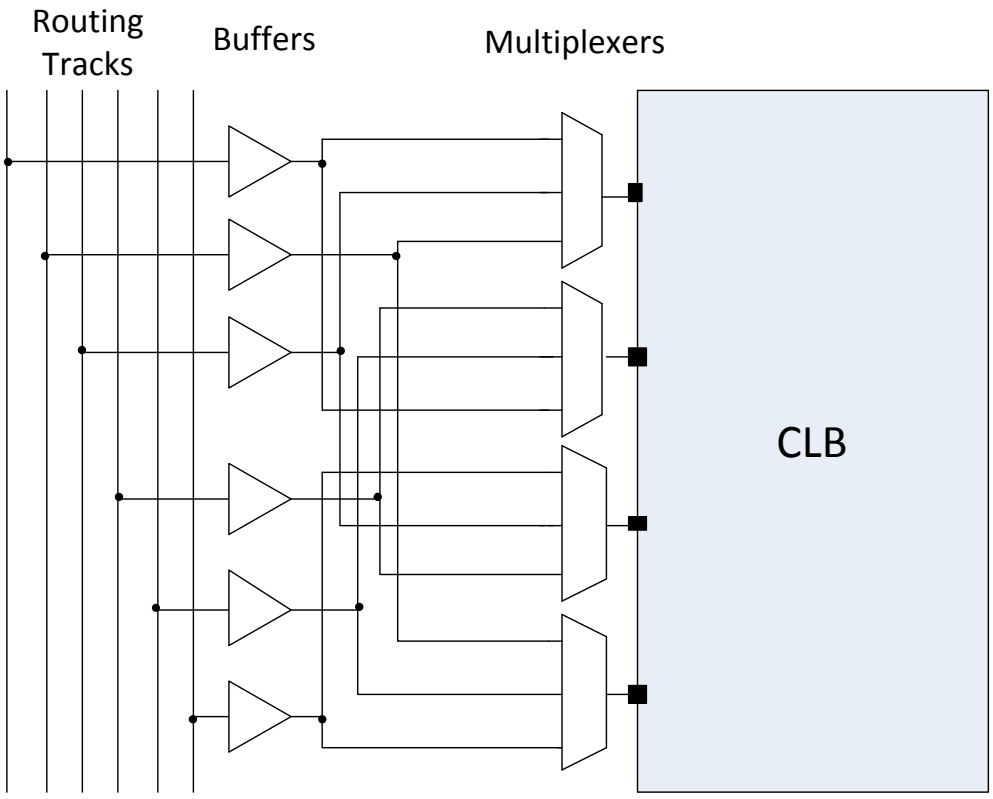


Figure 3.3: Sample of a connection box.

3.2.1.3 Multiplexers

The multiplexers in the switch boxes and connection boxes are built using NMOS pass transistor logic, using minimum sized transistors [60]. By default, two levels are used for any multiplexer with four or more inputs; however, the user can override the number of levels or the transistor sizes by modifying the architecture description file. Figure 3.4 shows a 4:1 two-level multiplexer.

Multi-level multiplexers are decomposed into a collection of single-level multiplexers. Figure 3.5 provides an illustration of this decomposition. It is conceiv-

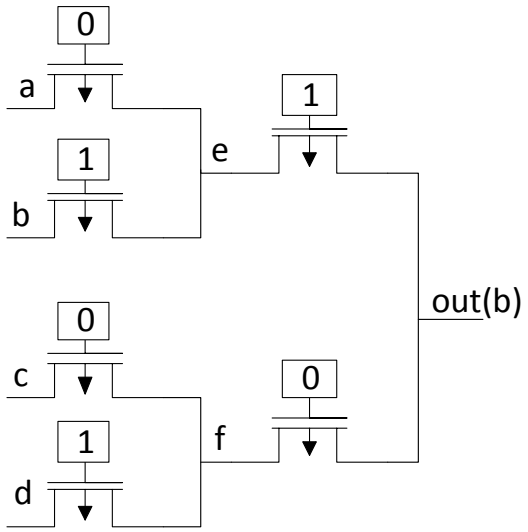


Figure 3.4: 4:1 2-level multiplexer.

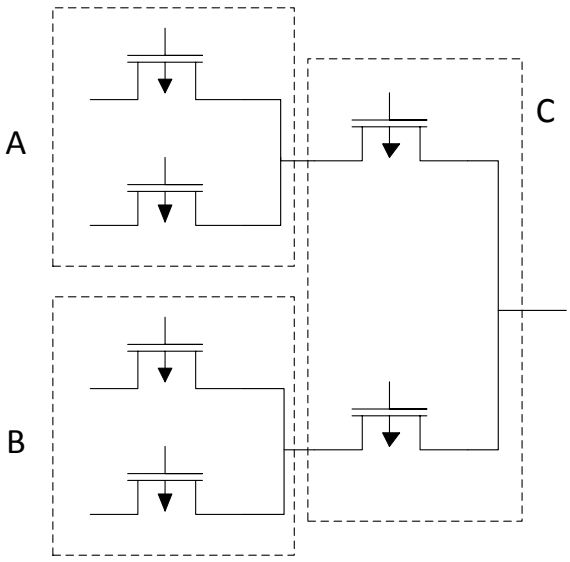


Figure 3.5: 4:1 2-level multiplexer, decomposed into single-levels.

able that some architectures may contain very large multiplexers, which could have many internal nodes, each having a large capacitance due to the number of connected transistors. Consider the multiplexer in Figure 3.4. Input *b* is selected, and any toggles on node *b* will also toggle nodes *e* and *out*. This was the extent of modelling of past power models [13, 14]. However, toggles on input *d* will still cause internal node *f* to toggle, consuming power. In Figure 3.5, although only multiplexers *A* and *C* belong to the activated path, multiplexer *B* will still consume power. For this reason, the architecture generator will include all three multiplexers in the generated FPGA circuitry.

3.2.1.4 Buffers

Buffers in the connection boxes and switch boxes are built using multiple stages of cascaded inverters, as illustrated in Figure 3.6. The first stage of the buffer is a sense stage with size $(W/L)_P = 1$, and $(W/L)_N = 2$. The sense stage is needed as the input to the buffer may receive a weak-1 due to pass transistor logic [60, 61]. The NMOS transistors in the remaining *N* stages of the buffer are sized according to Equation 3.1.

$$(W/L)_N = S^{(i/N)} \tag{3.1}$$

S is the size of the final stage of the buffer, *N* is the total number of stages, excluding the sense stage, and *i* is the index of the stage being sized, such that $i \in [1, N]$. The PMOS transistors are sized larger, according to the P/N ratio (See Section 4.3.4).

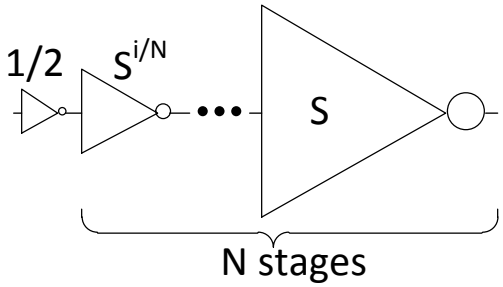


Figure 3.6: Multi-stage buffer, used in connection boxes and switch boxes.

The size of the final buffer stage, S, is found using Equation 3.2, which is based on the logical effort model [58].

$$S_{buf} = \frac{1}{4} * \frac{C_{Load}}{C_{INV}} \tag{3.2}$$

In this equation, C_{Load} is the total load capacitance driven by the buffer, and C_{INV} is the input capacitance of a minimum sized inverter. The number of stages is chosen using $N = \frac{logS}{log4}$, where N is rounded to the nearest integer [58]. This creates a buffer where the size from one stage to the next grows by a factor close to 4. This factor is commonly chosen when designing high-performance circuits, as it minimizes the delay. For power efficient circuits a larger factor may be used, resulting in slower, but smaller buffers [58]. The user may specify an alternative value for this factor in the architecture description file.

3.2.2 Complex Logic Blocks

VPR 6.0 supports user-defined logic blocks, which allow for a much larger range of architecture features than was possible in previous academic FPGA flows. This flexibility is obtained through a hierarchical description of the entities within each CLB. Each entity can:

- Instantiate multiple child entities, including children of different types.
- Define arbitrary interconnect between itself and children, or between multiple children.
- Operate in multiple modes. For example, a 6-LUT could operate as a single 6-LUT, or as two 5-LUTs.

Our model includes an algorithm which handles all of these features; any CLB that can be described using the new architcture language can be handled by our tool. Algorithm 1 provides pseudo code for the hierarchical power modelling algorithm. The algorithm employs a recursive function, which calculates the power usage for an entity. If the entity has children, the algorithm accounts for the power usage of the circuitry connecting the parent to children, as well as interconnect between children. The function is then called recursively for the child entities. If the entity does not have children it must be a primitive, such as a flip-flop, a LUT, or some other hard-block. For these types, the primitive handler function is called, which will determine the type of entity, and call the appropriate function. Each component of this algorithm will be further described below.

Algorithm 1 Calculate power dissipation of a CLB recursively:

calc_entity_power(*entity*):

```
1:  if entity has children then
2:      // This is a parent mode, determine mode of operation
3:      mode ⇐ entity.mode
4:      // Add interconnect power
5:      for each interc in mode do
6:          power ⇐ power + calc_interc_power(interc)
7:      end for
8:      for each child in mode do
9:          power ⇐ power + calc_entity_power(child)
10:     end for
11: else
12:     // This is a leaf node such as a LUT or Flip-Flop
13:     // Call the primitive handler
14:     power ⇐ power + calc_primitive_power(entity)
15: end if
16: return power
```

3.2.2.1 Intra-CLB Interconnect:

For each entity within a CLB, the architecture description file must specify the type of interconnect between itself and its children. This interconnect may be one of three types: *direct*, *many-to-one*, or *complete*. Figure 3.7 provides a diagram of a traditional logic block, and includes an example of each type of interconnect.

The architecture generator must decompose each interconnect into multiplexers with known transistor sizes, and wires with known length. The wire length depends on the physical spanning distance between parent and child entities, represented as L_{interc}. This distance is approximated using Equation 3.3. Figure 3.8 illustrates the derivation of this equation.

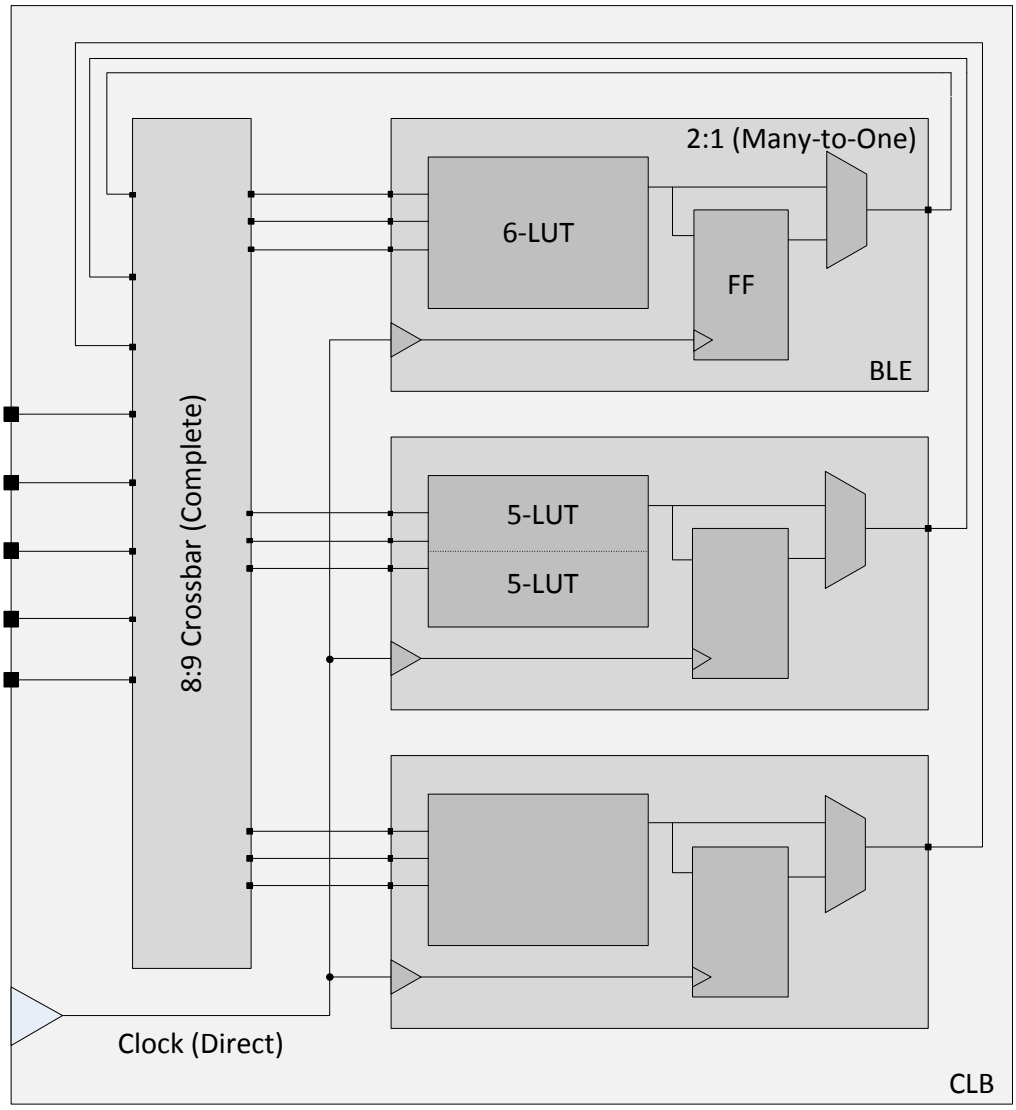


Figure 3.7: Types of local interconnect.

$$L_{interc} = 0.5 * (\sqrt{Area_{Parent}} - \sqrt{Area_{Children}}) \tag{3.3}$$

An m-input, n-output interconnect structure contains $(m+n)$ input/output wires, where the length of each wire is approximated as $1/2 \cdot L_{interc}$. This approximation is illustrated in Figure 3.9.

This approach provides a very rough approximation, and assumes a square topology with even spacing between the parent and child entities. In reality, this is unlikely, and interconnect may span much more, or much less than this value. Unfortunately, it is difficult to provide a more accurate estimation without knowing the transistor layout of the device, which is too complex to estimate automatically, and too burdensome to require from the user. In addition, this model only considers the wirelength of interconnect structures between parent and child entities. The wirelength within a primitive, such as a LUT or flip-flop, is ignored.

The following describes modelling techniques specific to each interconnect type:

Direct Interconnect This is a pure wire connection and requires no multiplexers. The connection contains a single source input, which drives one or more sinks. In Figure 3.7, the clock wires provide an example of this type of connection.

Many-to-One Interconnect This type is implemented as a single multiplexer. An example of this pattern is a multiplexer that chooses whether the BLE output is

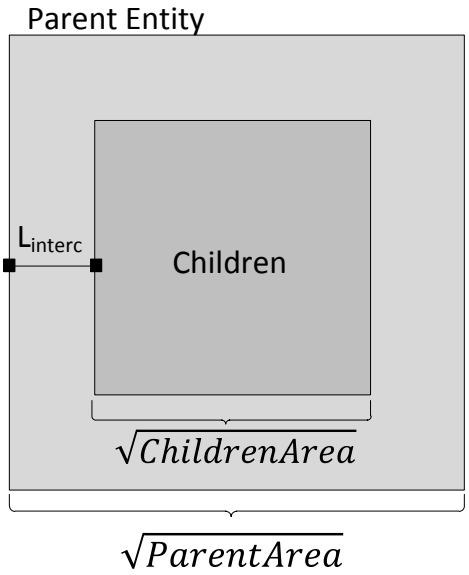


Figure 3.8: Local interconnect spanning distance, L_{interc}, between parent and child entities in a CLB.

driven by the LUT output or the flip-flop output, as shown in Figure 3.7. These potentially large multiplexers are decomposed into single-level multiplexers, in the same manner as the global routing multiplexers (see Section 3.2.1.3).

The architecture description file also supports the declaration of bus-based multiplexers, which can be used for coarse-grained architectures. An *m*-input many-to-one interconnect of *n*-wide buses is modelled as *n* instances of an *m*-input multiplexer.

Complete Interconnect This type describes a fully populated *m*-input, *n*-output crossbar. These crossbars are modelled as *n* number of *m*-to-1 multiplexers. Each multiplexer is modelled as previously described. Figure 3.7 provides an example

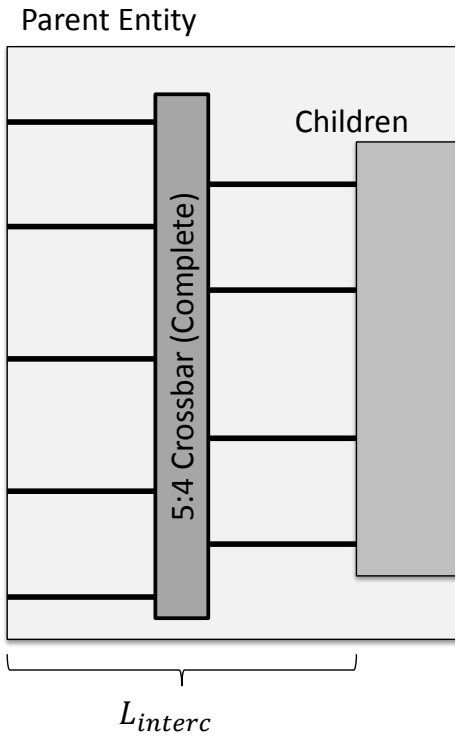


Figure 3.9: Wire length in a local interconnect structure. Shown here is a 5-input, 4-output complete interconnect, which results in 9 wires of length $1/2 \cdot L_{interc}$.

of an 8-to-9 crossbar connecting the inputs and outputs of the CLB to the inputs of the BLEs.

3.2.2.2 Look-up Tables (LUTs)

LUTs are assumed to be implemented as a network of two-input multiplexers, with level restorers used between every other stage [60]. Figure 3.10 provides an illustration of a 4-input LUT. Calculating the power dissipation of this structure requires the switching activity of internal nodes of the multiplexer. The activity estimator, which will be described in Section 4.2, determines the activity of each

net, but not the activity within the LUTs themselves. These internal activities are calculated as follows. The initial level of multiplexers is fed by SRAM bits, making P_1 (the long-term probability that a node is high) equal to the value of the SRAM cell. P_1 values are cascaded through the LUT levels using Equation 3.4 for each 2-input mulitplexer, where s is the select signal.

$$P_1(out) = (1 - P_1(s)) * P_1(in0) + P_1(s) * P_1(in1) \tag{3.4}$$

$$A_S(out) = (1 - P_1(s)) \cdot A_S(in0) + P_1(s) \cdot A_S(in1) + A_S(s) \cdot P(in0 \neq in1) \tag{3.5}$$

A_S values (the expected switching activity of each node) are calculated at the output of each 2-input multiplexer using Equation 3.5, which is comprised of three terms. The first two terms represent the case when the selected input toggles, causing the output to toggle. The third term, $A_S(s) \cdot P(in0 \neq in1)$, represents toggles on the output due to the selector toggling, which occurs only when the two inputs are of different logic value. The probability of the two inputs being different is calculated using Algorithm 2.

3.2.2.3 D Flip-Flop

A D Flip-Flop, contains a master and slave loop, each with two inverters and a two input multiplexer. Figure 3.11 provides an illustration. The inverters are assumed to be minimum sized, and the multiplexers are composed of two minimum size

Algorithm 2 Method of calculating the probability that the two inputs of a stage-n multiplexer in a LUT have different logic values.

$P_{total} = 0$
for all $\{in_1, in_2, ...in_{n-1}\} \mid in_i \in \{0,1\}$ **do**
  // Check if SRAM values are diffent
  if $SRAM(in_1, in_2, ..., 0) \neq SRAM(in_1, in_2, ...1)$ **then**
    P_{branch} = 1
    // Sum probability of each branch
    for $i = 1 \rightarrow (n-1)$ **do**
      $P_{branch} = P_{branch} \cdot P(input_i == in_i)$
    end for
    $P_{total} += P_{branch}$
  end if
end for

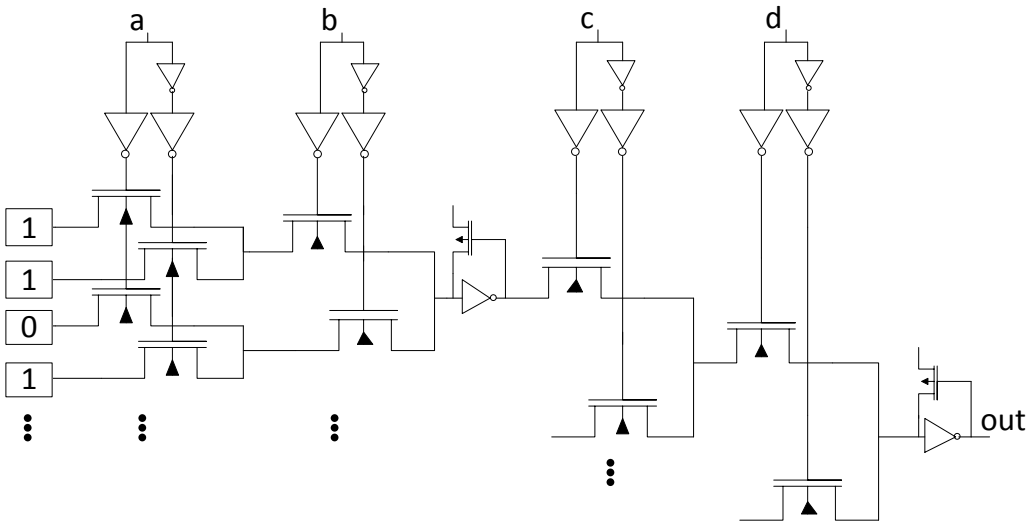


Figure 3.10: 4-input LUT.

transmission gates [58]. Like LUTs, the internal node activity must be calculated. The behaviour of the master loop is assumed to be $P_1 = P_1(D)$ and $A_S = (1 -$

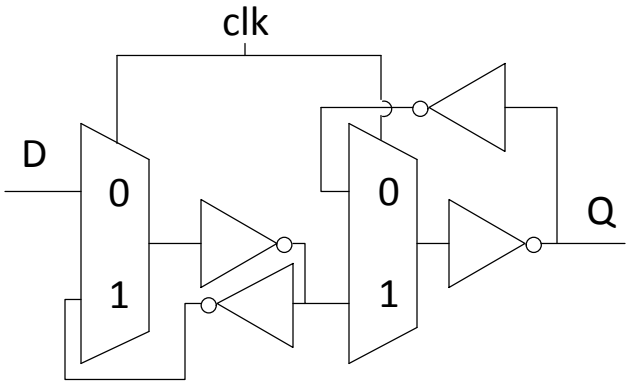


Figure 3.11: D Flip-Flop.

$A_S(clk)) * A_S(D)$. The slave loop behaviour is assumed to be $P_1 = P_1(Q)$ and $A_S = A_S(Q)$.

3.2.2.4 Undefined Hard Blocks (Black Boxes)

Our model does not estimate power of I/O pads, memories, multipliers, and other hard blocks designed by users. I/O pads have intentionally been excluded due to their complexity. If the user wishes to model memories, multipliers or other custom hard blocks, he/she must provide parameters for the block. This can be done in one of three ways:

1. Provide the absolute dynamic and static power of the block in the architecture file. This is the simplest, but least accurate approach, as power estimates are independent of the behaviour of input pins.

2. Provide the equivalent internal capacitance of the block in the architecture file. The power model will average the switching activity (A_S) across all

input pins to calculate dynamic power. Static power must still be specified as an absolute in the architecture file.

3. Provide a coded function that can be called by the primitive handler. The primitive handler will provide the signal probability (P_1) and switching activity (A_S) of the input pins. The user provided function should use this information to make more detailed power estimations, such as is done for LUTs and flip-flops.

3.2.2.5 Static Power of Unused Blocks

One step of Algorithm 1 is to determine the mode of operation for each entity in the CLB. For example, it is necessary to know whether the fracturable LUT is operating in fractured or non-fractured mode. However, if an entity is unused, it is not clear which mode it is operating in. This is needed to calculate static power of the unused entity. To handle this scenario, the power model allows the user to specify the default mode of operation in the architecture file. For example, the user can specify that when unused, LUTs should be modelled as the non-fractured type.

3.2.2.6 Limitations

There are several limitations in the way that the architecture description file specifies modes of operation. The file is designed to describe the *functional* behaviour of entities, and not the *actual hardware*. For example, functionally a 6-LUT can operate as two 5-LUTs. However, the actual hardware of the 6-LUT is not iden-

tical to the hardware of two distinct 5-LUTs. Unfortunately, there is no visibility within VTR to distinguish between a normal 5-LUT and a 5-LUT that is actually half of a 6-LUT. Thus when LUTs are fractured they are treated as multiple ordinary LUTs. This results in the input buffers to the LUTs being counted twice. Our testing shows that this discrepancy should be limited to a 1-2% overestimate in the overall power usage.

3.2.3 Clock Network

The clock network modelled is a four quadrant spine and rib design, as illustrated in Figure 3.12. The design is similar to the topology used in a Xilinx Virtex II Pro [62]. At this time VPR only supports a single clock; however, the power model contains infrastructure to model multiple clocks, provided that each clock is composed of the same topology as illustrated in the figure. Some FPGA multi-clock networks are more complicated, with different clocks connecting to different regions of the chip [62], but we do not model these more complex networks.

The model assumes that the entire spine and rib clock network will contain buffers separated in distance by the length of a grid tile. As in Section 3.2.1.4, the buffer is multi-stage with the final stage sized according to Equation 3.2. In this case, the load capacitance is assumed to be the next clock buffer, plus the capacitance of the wire connecting to it.

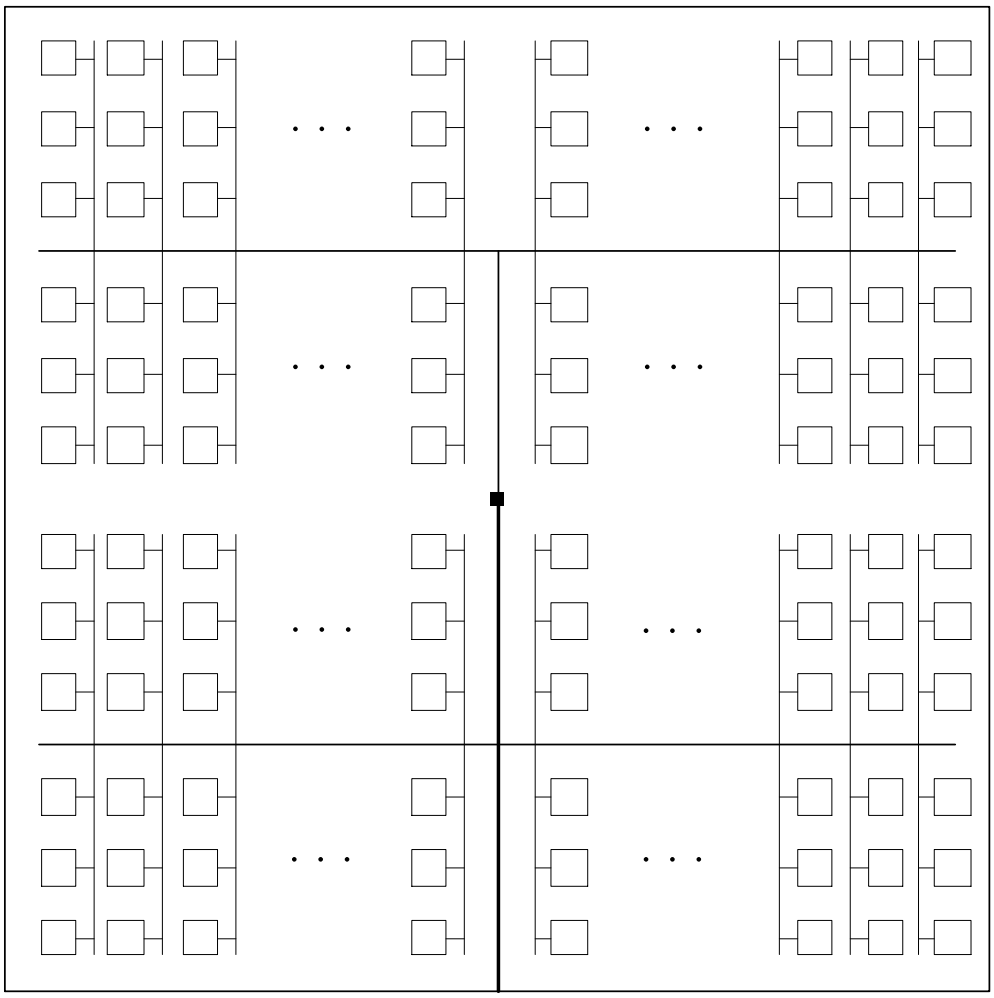


Figure 3.12: The clock network. Squares represent CLBs, and the wires represent the clock network.

3.2.4 Physical Size Estimation

One benefit of this power model is that all of the components are automatically sized. This includes routing buffers, clock buffers, and buffers within a CLB. The size of these buffers depends on the capacitance of the wire that is driven by the buffer. For this reason, the power model must be aware of the physical size of grid tiles in the FPGA, as well as the size of entities within CLBs, such as LUTs, flip-flops and multiplexers. The power model employs a transistor counting algorithm to determine the physical size of the various parts of the FPGA. This algorithm works with the architecture description file, so any user defined architecture is supported.

Algorithm 3 provides the method used to calculate the FPGA grid tile size, as well as the sizes of all CLB entities. Similar to the power estimation algorithm, a recursive function traverses the CLB entity hierarchy, counting transistors at each level. For entities that support multiple modes, the mode that requires the largest transistor area is assumed. If the user introduces custom hard-blocks in the design, they must also provide a function that returns the number of transistors in the block.

In our estimations we assumed the area of an SRAM cell to be equivalent to the layout area of 6 minimum-sized transistors [58]. The architecture description file contains a parameter that allows the user to override this value.

Algorithm 3 FPGA physical size estimation.

```
calc_FPGA_tile_length:
    transistor_cnt = cnt_entity_transistors(CLB)
    transistor_cnt = transistor_cnt + 2 * cnt_connection_box_transistors()
    transistor_cnt = transistor_cnt + cnt_switch_box_transistors()
    return sqrt(transistor_cnt * transistor_unit_area)

cnt_entity_transistors(entity):
    max_transistor_cnt ⇐ 0
    if entity has children then
        // Find the mode that requires the most transistors
        for each mode in entity.modes do
            transistor_cnt ⇐ 0
            for each interc in mode do
                transistor_cnt ⇐ transistor_cnt + cnt_interc_transistors(interc)
            end for
            for each child in mode do
                transistor_cnt ⇐ transistor_cnt + cnt_entity_transistors(child)
            end for
            max_transistor_cnt = max of (max_transistor_cnt, transistor_cnt)
        end for
    else
        // This is a leaf node such as a LUT or Flip-Flop
        // Call the leaf handler
        max_transistor_cnt ⇐ max_transistor_cnt + cnt_leaf_transistors(entity)
    end if
    // Store area of this entity for later use
    entity.area ⇐ max_transistor_cnt * transistor_unit_area
    return max_transistor_cnt
```

3.3 Summary

This chapter provided an overview of the components in our power model, and the roles they play in power estimation. The chapter also detailed the architecture generator, which is responsible for translating arbitrary FPGA architectures into

basic circuit components. This includes generating circuit information for the global routing fabric, the CLBs, and the clock network. All transistors and wire lengths in these circuits are automatically sized.

The architecture generator is able to process any user described architecture, regardless of complexity, and produce a collection of inverters, single-level multiplexers and wires. Once the FPGA circuitry has been reduced to these simple components, the other parts of the model are capable of estimating the dynamic and static power of each component. This power estimation is outlined in the next chapter.

Chapter 4

Power Estimation

This chapter describes the remaining components of our model: The low-level power estimator, the activity estimator, and the transistor properties generator.

The architecture generator translates the entire FPGA circuitry into a collection of inverters, multiplexers and wires, after which the low-level power estimator provides power estimates of these components. These estimates, described in Section 4.1, depend on the signal behaviour and transistor properties. Signal behaviours are determined by the activity estimator, as described in Section 4.2. Transistor properties are generated by running SPICE simulations and extracting the necessary values, as described in Section 4.3.

4.1 Low-Level Power Estimation

The power dissipated in the inverters, multiplexers and wires is comprised of both dynamic and static power. Dynamic power includes both switching power and

short-circuit power. Static power consists of subthreshold and gate leakage. Other sources of power are not significant and are ignored [58].

4.1.1 Switching Power

Switching power is the result of charging and dissipating energy stored in the capacitance of transistor nodes and wires, and is proportional to the frequency with which the nodes toggle. The switching power dissipated by every wire, as well as the source, drain, and gate of every transistor is estimated using Equation 4.1 [63].

$$P_{dyn} = \alpha C V_{swing} V_{DD} f \tag{4.1}$$

The parameters of the above equation are:

- α: The expected switching activity of the node. $\alpha = 1$ represents a node that toggles once, or switches twice, per clock cycle.
- C: The node or wire capacitance.
- V_{DD}: The supply voltage.
- V_{swing}: The swing voltage, which is the voltage range of the node or wire.
- f: The operating frequency of the circuit.

In our implementation, transistor node capacitances are provided by the transistor properties generator, which will be described in Section 4.3.1. Wire capacitances are calculated by multiplying the wire length provided by the architecture generator with the unit-length wire capacitances provided in the architecture description file. The activity factor, α, is calculated as $\alpha = A_S/2$, where A_S is the transi-

tion density from the activity estimator, which will be described in Section 4.2. The operating frequency, f, is provided by VPR 6.0. Typically, the swing voltage, V_{swing}, is equal to the supply voltage. However, in pass-transistor logic, there may be voltage drop since NMOS transistors transmit a weak logic-1 [61]. This voltage drop is estimated using techniques described in Section 4.3.5.

4.1.2 Short-Circuit Power

Short-circuit current occurs in CMOS logic during an input transition, as both the pull-up and pull-down networks are simultaneously enabled for a short period of time [58]. In the components handled by this model, short-circuit power occurs only in inverters. It is highly dependent on the speed of the input transition; slower input transitions lead to longer periods of short-circuit current. In cases where the inverter is driven by a pass-transistor multiplexer, the input voltage must be pulled up to V_{DD}, creating a slow input transition, and increased short-circuit power [64]. If the input multiplexer is large, the input capacitance will be larger, further slowing the input transition.

The model estimates the short-circuit power of all buffers as a factor of the switching power. This factor is extracted from SPICE, as described in Section 4.3.6. This factor depends on the size of the buffer, the type of logic driving the buffer, and the capacitance at the input.

4.1.3 Subthreshold Leakage Power

Subthreshold leakage current occurs in transistors that are operating in the cut-off region, but have a non-zero source-drain voltage (V_{ds}). The amount of leakage is highly dependent on V_{ds} [58]. Subthreshold leakage occurs in both inverters and multiplexers.

Inverters

In an inverter, the subthreshold leakage power, P_{st}, of the two transistors is:

$$P_{st,\,PMOS} = V_{DD} \cdot P_1 \cdot I_{st}((W/L)_P) \tag{4.2}$$

$$P_{st,\,NMOS} = V_{DD} \cdot (1 - P_1) \cdot I_{st}((W/L)_N) \tag{4.3}$$

I_{st} is the subthreshold leakage current of the transistor when $V_{ds} = V_{DD}$, which is a function of the transistor size. It is determined automatically through SPICE, as described in Section 4.3.2.

Multiplexers

For muliplexers, past power models [13, 14] used a simple worst-case analysis, where all transistors were assumed to be leaking. We use a different approach, analyzing the leakage behaviour of each transistor. In an m-input, single-level multiplexer, there will always be one input that is selected. The voltage drop (V_{ds}) across the selected transistor is small, and the subthreshold leakage can be ignored. However, the other $m-1$ transistors in the multiplexer may experience a more significant voltage drop and exhibit significant leakage [65]. The subthresh-

old leakage power of each of these transistors is:

$$P_{st} = V_{dd} \cdot P[V(i) \neq V(o)] \cdot I_{st}(V_{ds}) \tag{4.4}$$

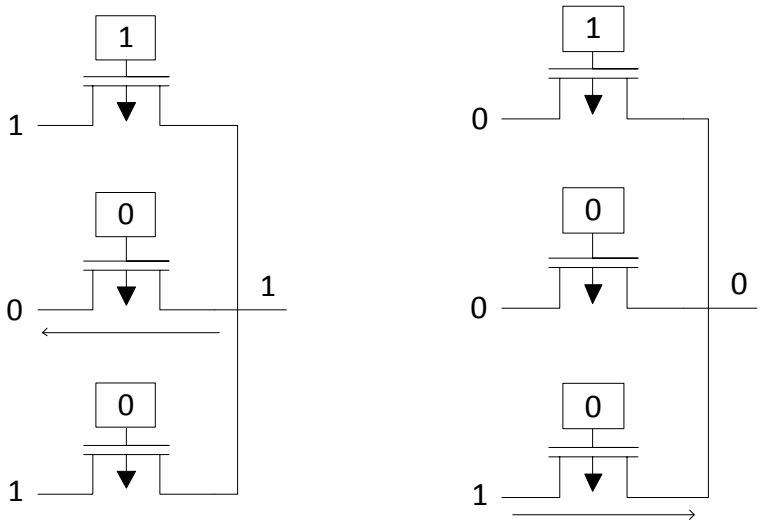
$P[V(i) \neq V(o)]$ is the probability that the logic-value of the input is different than the logic value of the output, and is determined using the signal probabilities (P_1 values). Figure 4.1 provides an example of a three-input multiplexer, and illustrates how the leaking transistors depend on the values of the inputs. I_{st} is the subthreshold current of a minimum-sized NMOS transistor, for a given voltage drop, V_{ds}. As mentioned earlier, the voltage drop across the multiplexer is a function of the size of the multiplexer, and the values of the inputs. SPICE simulations are used to determine the leakage current for various V_{ds} values, as described later in Section 4.3.2.

4.1.4 Gate Leakage Power

Gate leakage occurs when currents tunnel through the transistor gate to the source-drain channel. We found that gate leakage was only significant for large inverters. The gate leakage power (P_g) of the two transistors in the inverter is calculated as:

$$P_{g,\,PMOS} = V_{DD} \cdot (1 - P_1) \cdot I_g(W/L_P) \tag{4.5}$$

$$P_{g,\,NMOS} = V_{DD} \cdot P_1 \cdot I_g(W/L_N) \tag{4.6}$$



(a) Output→Input **(b)** Input→Output

Figure 4.1: Subthreshold leakage in a multiplexer, which can occur between input and output in either direction. Leaking transistor shown with an arrow.

In these equations, I_g is the gate leakage current of the transistor, and is a function of the transistor size. It is extracted from SPICE, as described later in Section 4.3.3.

4.2 Activity Estimation

The dynamic and static power estimates described in the previous section are dependent on the behaviour of the user-supplied circuit. For each net in the circuit, the activity estimator determines:

- The Transition Density, or switching activity, A_S.
- The Signal Probability, P_1.

These values were described in greater detail in Section 2.3.3

4.2.1 Algorithm

Previous power models have used ACE-1.0 [13] and ACE-2.0 [55] to generate these values. ACE-1.0 is fast, but is inaccurate for large and/or sequential circuits. ACE-2.0 provides an improved algorithm that better handles sequential circuits. Unfortunately, ACE-2.0 is built on the Berkeley SIS tool [66], which is obsolete. SIS has been superseded by the Berkeley ABC circuit tool [38], which is more robust, has better performance, and supports much larger circuits.

As part of this work, we have implemented the algorithms from ACE-2.0, using the ABC libraries. Our implementation consists of two phases: simulation and static analysis. Although ACE-2.0 used three phases, the second phase was an optimization, designed to improve run-time when estimating activity of large circuits. Because of the improved performance of ABC, the second phase is no longer necessary. The following describe the two phases of the activity estimator, which correspond to the first and third stages of ACE-2.0. These phases are described in greater detail in [55].

4.2.1.1 Stage 1: Simulation

In the first stage, the circuit is simulated for 5000 cycles. The user has the option of providing a vector of inputs, or they may specify P_1 and A_S for each of the input nodes. If neither is provided, the inputs are assumed to have the behaviour $P_1 = 0.5$ and $A_S = 0.2$. These are the same values used by ACE-1.0 and ACE-

2.0. The overall power of the circuit is highly dependent on activity of the inputs. Inputs with larger activity will directly increase the estimated dynamic power of the circuit.

During circuit simulation, the logic values of each register are monitored. This is used to calculate P_1 and A_S for each register in the circuit. The simulation phase is necessary as the static analysis method is not accurate when there are sequential loops present [55].

4.2.1.2 Stage 2: Static Analysis

The second stage is a static analysis of the behaviour of the combinational logic nodes between the registers. It determines the probability of changes on the inputs, and whether these changes will cause a change on the output. The algorithm also provides an estimation of circuit glitching, using a low-pass filter approach as described in [67].

4.2.2 Limitation: Black Boxes

ACE-2.0 was created to be used with older versions of VPR, which did not support heterogeneous FPGA architectures. The latest version of VPR allows for the specification of hard-blocks, such as memories, multipliers, or other user-defined functions. In the BLIF format, these functions are represented as black boxes, with no definition of their behaviour.

A major limitation of the activity estimator is that it does not consider the contents of the black boxes. It uses ABC to read the BLIF file, which removes

black boxes from the circuit. The outputs of the black box become primary inputs to the circuit, and the black box inputs become primary outputs of the circuit. The activity estimation then proceeds as usual. The outputs of the black box will be assigned activity values, just as if they were an input to the circuit. This results in activity information that does not reflect the functionality of the black box. It is possible to work around this limitation by providing realistic activity values for the black box outputs, just as can be done with primary inputs.

4.3 Transistor Properties Generator

Static and dynamic power calculations rely heavily upon the CMOS technology being used. In past power models [13, 14] users were responsible for providing many properties of the CMOS technology, such as capacitances, short-circuit behaviour, and other transistor parameters. The models used these parameters with equations to approximate transistor capacitances and leakage currents. There are two main disadvantages to this method. First, this technique does not scale well to modern CMOS technologies. The equations used to estimate leakage currents in older CMOS technologies are not accurate across modern technologies. Secondly, this method places a burden on the user to determine all of these parameters prior to using the power model.

Our model takes a different approach; it provides a script that performs multiple SPICE simulations, extracts relevant transistor characteristics, and writes them to a file. The user is only required to provide a transistor technology file, the operating voltage, and the operating temperature. The entire process takes about 5

minutes to execute. This process needs to be performed only once, and the file can be reused for subsequent executions of the power model. A new file needs to be created only when changing CMOS technologies, operating voltage, or temperature.

When performing SPICE simulations, the script uses the default parameters included in the technology file. In addition, parameters are provided for the source/drain areas and perimters (AS, PS, AD, PD). The width of the source and drain regions is assumed to be equal to the width of the transistor. The length of these regions is assumed to be 2.5 times the length of the transistor.

The following explains how the script determines relevant transistor characteristics for the CMOS technology.

4.3.1 Transistor Node Capacitances

Transistor nodal capacitances are determined for both PMOS and NMOS minimum-length transistors. The transistor sizes begin at the minimum width, and increase by 5% until reaching 2000 times the minimum width, resulting in data for over 150 different transistor sizes. For each size, a SPICE simulation is performed on a single transistor. The desired node (source, drain, or gate) is kept logic-high and the voltages to the other two nodes are varied. The capacitance of the desired node is measured and averaged across the simulation.

4.3.2 Subthreshold Leakage Current

Subthreshold leakage currents are required for two different scenarios:

1. For inverters, where transistor size varies, but V_{ds} is always equal to V_{DD}.

2. For multiplexers, where transistor size is always minimal, but V_{ds} varies.

For the first case, where transistor size varies, the process is similar to that described above. Over 150 different transistor sizes are simulated in SPICE, and the subthreshold leakage current is measured. For PMOS transistors the source and gate are set to V_{DD}, and the drain is set to ground. The voltages are reversed for NMOS transistors.

For the second case, where V_{ds} varies, a minimum-sized NMOS transistor is simulated. The gate is set to ground, the drain-source voltage is incremented from $1/2 V_{DD}$ to V_{DD}, and the subthreshold current is measured.

4.3.3 Gate Leakage Current

To determine gate leakages, a similar process was followed. SPICE simulation was performed for over 150 PMOS and NMOS transistor sizes. For PMOS transistors, the gate was set to ground, the source and drain were set to V_{DD}, and the current through the gate was measured. For NMOS transistors, the node voltages were reversed.

4.3.4 P/N Ratio Sizing

In order to properly model transistor sizes, the P/N ratio must be determined. The P/N ratio is the ratio of the width of the PMOS transistor to the width of the NMOS transistor in an inverter, and depends on the CMOS process. SPICE simulation is performed on a single inverter with a square input wave. The NMOS transistor is

sized such that $W/L = 1$, and the PMOS transistor is swept in size from $W/L = 1$ to $W/L = 5$, by increments of 0.05. The P/N ratio is chosen to minimize the difference in rise and fall delays, which is common design practise [58].

4.3.5 Multiplexer Voltage Drop

In pass-transistor multiplexers, when the output is logic-high, the output voltage will always be less than V_{DD}. This is because NMOS transistors transmit a weak logic-1 [58]. Our testing showed that the output voltage depends on both the size of the multiplexer, and the logic value of the non-selected inputs. Adding more transistors to the multiplexer, or having more inputs that are grounded, cause the output voltage to drop. This is because both scenarios decrease the equivalent resistance between the output and ground. In addition, since multiplexers may be stacked in series, the input voltage to a multiplexer may be less than V_{DD}, further decreasing the output voltage.

To determine the expected voltage drop across a multiplexer we simulated many different multiplexers, varying both the size and the input voltage. For a given size and input voltage, we simulate the multiplexer with all non-selected inputs set to ground, and then with all inputs set to the input voltage. The results are extracted and stored in a table. This allows the model to predict the output voltage of a multiplexer based upon 1) the size, 2) the input voltage, and 3) the expected logic-value of the inputs. Although this may seem excessively detailed, it is necessary to make accurate static power estimates. Subthreshold currents are

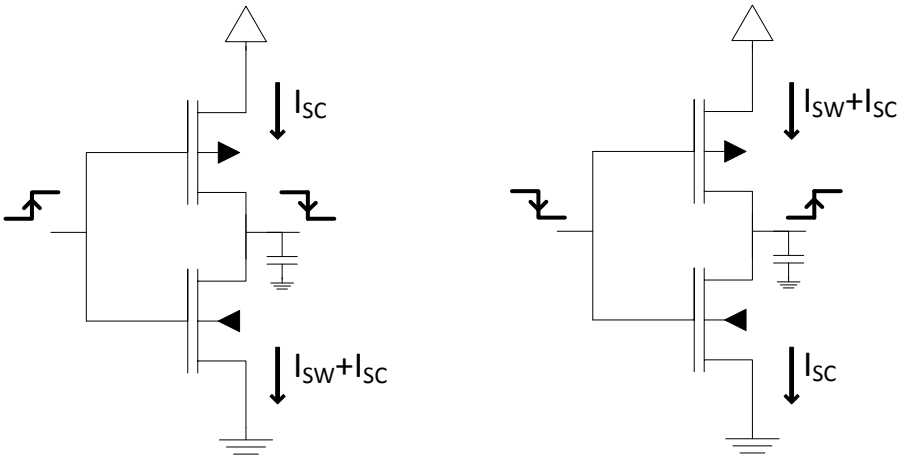


Figure 4.2: Short circuit currents in inverters during input transitions. The left-hand side shows an inverter experiencing a rising edge. The output capacitance discharges through ground (I_{SW}), and the short-circuit current (I_{SC}) enters from the supply and exits through ground. On the right-hand side, a falling edge input is shown, where the output capacitance in charged.

highly dependent on V_{ds}; even small changes to the multiplexer voltage drop can cause large changes in static power dissipation.

4.3.6 Short-Circuit Buffer Factor

As described in Section 4.1.2, CMOS buffers experience short-circuit currents during a switching of the input. The short-circuit currents are increased when the buffer is driven by a pass-transistor multiplexer, as it takes time for the input to be pulled up to V_{DD}. If the input multiplexer is large, the pull-up takes longer due to the increased capacitance.

To determine the effect of short-circuit power we used SPICE to automatically simulate buffers of various sizes, driven by either CMOS logic, or pass-transistor logic with various multiplexer sizes. We measured the currents from the supply, and to ground, during rising and falling edges. When the input to an inverter experiences a falling edge, the output experiences a rising edge, and the capacitance at the output is charged. The current from the supply includes both the current to charge the output, as well as the short-circuit current. The current to ground is only the short circuit current. A similar, but reverse scenario occurs on the rising edge of an input. Figure 4.2 illustrates the full behaviour. Using this technique we can isolate the short-circuit current, and represent it as a factor of the switching power.

4.4 Summary

This chapter detailed the low-level power estimator, activity estimator, and transistor properties generator. Together, these modules are able to estimate the static and dynamic power of every inverter, multiplexer and wire that make up the FPGA circuitry.

The activity estimator determines the behaviour of the user circuit, which directly influences the power estimates. The transistor properties generator uses SPICE simulations to extract characteristics of the CMOS process, such as transistor capacitances and leakage currents, which are essential in making accurate transistor-level power estimates. The low-level power estimator combines all of this information, and calculates the static and dynamic power of the entire FPGA.

This chapter, together with the previous chapter, contain all of the implementation details of our model. The following chapter explains how the tool is verified, as well as experimental results.

Chapter 5

Verification and Results

This chapter includes an evaluation of the accuracy of the power model estimations, compared to those obtained from SPICE simulations, as well as experiments performed using the model to estimate the impact of various FPGA architecture parameters on power. The first experiment (Section 5.3) provides a breakdown of power usage between major FPGA components for three CMOS processes. The second experiment (Section 5.4) studies the effect of fracturable LUTs on overall power usage.

5.1 Verification of Power Estimation

In order to verify that our model produces accurate power estimations, we compared the power estimates from our model to results from SPICE simulations for many different FPGA components.

5.1.1 Verification Procedure

SPICE circuits were created for various sizes of inverters and multiplexers, which are the basic blocks of the FPGA circuitry. To test the interaction between these basic components we also designed some larger components, which include multi-stage buffers, LUTs, flip-flops and switch box components.

The circuits were simulated in HSPICE [68] for one clock cycle, with a 5 ns period. For each component, we tested both high-activity and zero-activity scenarios. In the high-activity test, each input was toggled once during the clock cycle, such that $P_1 = 0.5$ (signal probability) and $A_S = 2$ (activity factor). The high-activity scenario verifies the accuracy of the dynamic power estimations, as these large activity values will cause the dynamic power to dominate the overall power usage. The second experiment simulates a zero-activity circuit. In this case, the inputs are held at logic-high and are unchanged for the duration of the simulation, thus $P_1 = 1$ and $A_S = 0$ for each input. This scenario tests the accuracy of the static power estimations, as there is no dynamic power when the inputs do not toggle.

The energy usage is estimated in SPICE by integrating the supply current over the simulation time, and the result is compared to the estimate provided by our power model. This comparison was performed for three different CMOS processes: 22 nm, 45 nm, and 130 nm. The CMOS technology files were obtained from the Berkeley Predictive Technology Model (PTM) [69], which are predictions of real-world technologies. Table 5.1 lists characteristics of these CMOS processes.

Process	V_{DD}	P/N Ratio	NMOS			PMOS		
			C_g (aF)	$C_{s/d}$ (aF)	I_{st} (nA)	C_g	$C_{s/d}$	I_{st}
22 nm	0.8	1.70	15	57	4.8	8	80	4.3
45 nm	1.0	1.75	53	118	2.3	28	182	0.5
130 nm	1.3	2.50	255	351	10.2	131	602	6.2

Table 5.1: CMOS process characteristics. All transistor values are shown for a minimum-sized transistor. I_{st} is the subthreshold leakage current when $V_{ds} = V_{DD}$. Parameters were extracted at 85° C.

5.1.2 Verification Results

Table 5.2 and Table 5.3 contain the results of the high-activity and zero-activity tests, respectively. Each table lists the energy usage obtained through SPICE simulations, and the error percentages of our model.

The results show that for all component types, the high-activity estimation is within 20% of SPICE results, which demonstrates the accuracy of the dynamic power estimations. In the case of the zero-activity scenario, the estimations are even more accurate, falling within 5% of the SPICE results. This demonstrates the high accuracy of the static power estimations.

In general, our model provides good estimates of both dynamic and static power usage for all component types. However, it should be noted that the total estimated FPGA power will still be significantly lower than if the power were measured for a physical FPGA. This is because the model does not incorporate all components of the FPGA, such as the I/O pads, the clock generator, and other specialized circuitry. Unlike power estimation algorithms integrated into commercial CAD tools, the purpose of our model is *not* to provide absolute FPGA power es-

Component Type	Size	22 nm		45 nm		130 nm	
		Act. (fJ)	% Est. Error	Act. (fJ)	% Est. Error	Act. (fJ)	% Est. Error
Inverter	1	0.16	-6	0.5	-7	4	-16
	8	1.06	-14	2.9	-12	26	-21
	16	2.08	-14	5.7	-13	52	-21
	32	4.15	-14	11.2	-13	102	-21
	64	8.25	-14	22.3	-13	205	-22
Multiplexer	4	0.40	+1	1.4	-7	8	-16
	8	0.74	+1	2.8	-6	14	-16
	12	1.01	+3	3.4	-6	19	-14
	16	1.34	+3	4.6	-8	25	-14
	20	1.61	+3	5.4	-6	30	-13
Multi-Stage Buffer	16	2.89	+3	8.0	+6	95	-15
	25	4.37	-1	12.5	+1	156	-17
	64	11.34	-3	31.9	-4	396	-21
LUT	2	1.59	+15	3.89	+17	28	0
	4	6.70	+18	17.12	+14	115	-7
	6	25.82	+20	67.56	+13	437	-10
D Flip-Flop	-	1.15	-8	3.7	-9	26	-10
Switch Box Component (Mux./Buffer)	4/9	3.73	+4	8.8	+4	77	-14
	8/9	4.32	-2	9.5	0	84	-15
	12/16	6.80	-9	15.0	-8	143	-19
	16/16	7.44	-12	15.6	-4	150	-19
	20/25	10.79	-15	22.9	-13	229	-20
	25/25	11.65	-18	23.8	-11	234	-19

Table 5.2: High-Activity energy estimation, versus SPICE simulation. *Act.* shows actual energy (femtojoules), obtained through SPICE simulation, for a single cycle (5 ns). *% Est. Error* shows the percentage difference between our model estimation and the actual energy, where positive percentages indicate overestimates. All inputs toggle twice during the cycle.

Component Type	Size	22 nm		45 nm		130 nm	
		Act. (fJ)	% Est. Error	Act. (fJ)	% Est. Error	Act. (fJ)	% Est. Error
Inverter	1	0.03	0%	0.01	0%	0.11	0%
	8	0.33	0%	0.05	0%	0.92	0%
	16	0.66	0%	0.11	0%	1.84	0%
	32	1.33	0%	0.21	0%	3.69	0%
	64	2.68	0%	0.42	0%	7.38	0%
Multiplexer	4	0.01	+6%	0.01	+4%	0.08	+3%
	8	0.02	+7%	0.02	+4%	0.15	+3%
	12	0.02	+5%	0.02	+3%	0.19	+2%
	16	0.03	+7%	0.03	+4%	0.22	+3%
	20	0.03	+6%	0.03	+4%	0.26	+2%
Multi-Stage Buffer	16	0.78	+1%	0.17	+2%	2.17	+3%
	25	1.18	+1%	0.24	+1%	3.28	+2%
	64	2.51	0%	1.16	0%	6.83	+1%
LUT	2	0.35	0%	0.12	+1%	1.16	+1%
	4	0.92	-1%	0.39	-4%	3.22	-2%
	6	2.26	0%	1.19	-1%	8.35	-2%
D Flip-Flop	-	0.10	0%	0.04	-2%	0.37	-2%
Switch Box Component (Mux./Buffer)	4/9	0.48	0%	0.12	-1%	1.41	-1%
	8/9	0.50	0%	0.14	-1%	1.52	-1%
	12/16	0.83	0%	0.21	-1%	2.44	-1%
	16/16	0.85	0%	0.22	-1%	2.51	-1%
	20/25	1.26	0%	0.30	-1%	3.66	-1%
	25/25	1.28	0%	0.31	-1%	3.73	-1%

Table 5.3: Zero-Activity energy estimation, versus SPICE simulation. *Act.* shows actual energy (femtojoules), obtained through SPICE simulation, for a single cycle (5 ns). *% Est. Error* shows the percentage difference between our model estimation and the actual energy, where positive percentages indicate overestimates. Inputs do not toggle.

timates. Instead, the purpose is to quantify relative improvements or degradation in power efficiency as architectural or CAD parameters are changed. The results show that the power model is scalable between small and large components, as well as between small and large transistor technologies. The scalability of the power model across different technologies and components makes it useful for evaluating trade-offs during FPGA architecture and CAD design.

Although our model is sufficiently accurate for architectural evaluations, some error still exists between estimates of our model, and the result from SPICE simulations. The following section provides explanations for these differences.

5.2 Sources of Estimation Error

The following outlines sources of error in our power model estimations, specifically in comparison to SPICE results for identical components.

5.2.1 Short-Circuit Current

The largest source of error is short-circuit currents, which are complex and difficult to estimate. All CMOS circuity experiences some short-circuit current; however, it is particularly significant when the CMOS logic is driven by pass-transistor logic. This pattern occurs in both switch boxes and LUTs. For example, switch box components (Figure 3.2) contain a multiplexer, made up of pass-transistor logic, that drives the input of a multi-stage buffer. The pass-transistor logic cannot produce a strong logic-1, so a weak PMOS transistor is added in a feedback configuration, which pulls the input to V_{DD}. However, this pull-up takes time,

resulting in a slow rising edge to the input and substantially larger short-circuit currents than if the buffer were driven by CMOS logic.

This behaviour can be seen in Table 5.2, which shows that the power usage of a switch box component is much larger than just the sum of the multiplexer and buffer that compose it. For example, at 22 nm, a 12-input multiplexer requires 1.0 fJ of energy per cycle, and a size-16 buffer requires 2.9 fJ. However, when combined, resulting in the buffer being driven by pass-transistor logic, they require 6.8 fJ of energy, an increase of 74%.

We do perform some estimation of this short circuit current, as explained in Section 4.1.2. This has allowed us to reduce the error from 50% to within 20%. More extensive short-circuit modelling would be required to improve accuracy further.

5.2.2 Transistor Node Capacitances

Another source of error is our method of estimating transistor node capacitances. In our power model they are extracted from SPICE simulation, as explained in Section 4.3.1. Our model uses a single value that represents the average transistor capacitance. In reality, transistor node capacitances are a function of the state of the transistor, and vary based on the voltages present at the other nodes of the transistor. We do not model this level of complexity, and this simplification is likely responsible for some of the error in dynamic power estimations.

5.2.3 Gate Leakage Currents

Our model estimates gate leakage currents only for CMOS inverters, as described in Section 4.1.4. We ignore gate leakage in multiplexers because 1) multiplexers are built using small transistors, so the gate leakage is minimal, and 2) variations in internal node voltages of multiplexers make accurate gate leakage difficult. This simplification accounts for the error in static power estimations of multiplexers, which is at most 7%.

5.3 Experiment 1: Component Breakdown

With the accuracy of the model verified, we now use it to study power characteristics of different architectures. The first experiment provides a breakdown of power usage between the major FPGA components, for different CMOS technologies.

5.3.1 Methodology

We executed the full VTR flow, and measured the power for the entire suite of VTR benchmarks. The architecture file used resembles the architecture of an Altera Stratix IV FPGA [70]. The architecture consists of 6-input LUTs arranged in CLBs, where each CLB contains 10 LUTs and has 33 inputs. The CLBs are connected using length-4 segments. The timing information is taken from the iFAR FPGA architecture repository [71], and interconnect capacitances are taken from the 2007 ITRS interconnect roadmap [72]. The transistor technologies used are 22 nm, 45 nm, and 130 nm PTM [69] models, as described in the previous section. Table 5.1 lists characteristics of these processes.

5.3.2 Results

Component	22nm	45nm			130nm
	Avg %	*Min %*	*Max %*	*Avg %*	*Avg %*
Routing	**83.8**	**61.9**	**87.2**	**72.7**	**68.2**
Switch Box	73.2	33.7	64.6	46.4	47.7
Connection Box	8.8	7.8	30.3	17.8	12.6
Global Wires	1.0	1.2	11.6	5.6	5.9
CLBs	**13.9**	**5.3**	**33.7**	**21.4**	**26.0**
LUTs	9.0	2.2	14.8	7.7	10.9
Flip-Flops	1.0	0.9	6.3	2.6	4.9
MUXs / Crossbars	3.1	0.7	11.4	5.7	6.4
Local Wires	1.2	1.9	13.1	7.0	7.3
Clock	**1.4**	**0.7**	**8.6**	**3.4**	**3.3**
Buffers	1.2	0.6	5.8	2.4	2.2
Wires	0.2	0.1	2.9	1.0	1.1

Table 5.4: Power breakdown by component type.

Table 5.4 provides the breakdown of power usage between major components for 22 nm, 45 nm and 130 nm technologies, where the results are averaged across all VTR benchmarks. Table 5.5 provides a breakdown for each benchmark circuit for the 45 nm technology. As evident from the results, the total power, and breakdown between components, is highly dependent on the benchmark circuit.

5.3.3 Analysis

The 45 nm results show that on average, 73% of the power consumption is due to the routing fabric, 21% from CLBs, and 3% from the clock network. The results show that routing is responsible for an increasing fraction of overall power as the

Benchmark Circuit	6-LUTs	Power (mW)	Routing % Total	CLBs % Total	Clock % Total
bgm	30089	78.6	70	28	2
blob merge	6016	10.3	73	23	4
boundtop	2921	6.6	67	27	6
ch intrinsics	413	2.1	66	26	9
diffeq1	434	2.2	72	25	3
diffeq2	277	2	79	18	4
LU8PEEng	21954	35.9	75	24	1
mcml	99700	109.3	75	24	1
mkDelayWorker32B	5580	32.8	83	13	5
mkPktMerge	226	18.2	87	5	8
mkSMAdapter4B	1977	5.5	72	22	6
or1200	2963	7.4	73	25	3
raygentop	2134	10.2	69	27	4
sha	2212	3.3	62	34	4
stereovision0	11462	36.9	64	29	7
stereovision1	10366	58.1	73	24	3
stereovision2	29849	227.4	82	17	1
stereovision3	174	0.9	52	36	12

Table 5.5: Power usage, and breakdown by circuit (45 nm).

technology is scaled down. From 130 nm to 22 nm the percentage of power due to the routing network grows from 68% to 84%. This behavior is due to the fact that the wire capacitance does not scale down at nearly the same rate as the transistor node capacitances. For example, the capacitance of routing segments from 45 nm to 22 nm drops by 57% due to shorter segments and lower wire capacitance per length. However, the input capacitance of a minimum sized inverter between 45 nm and 22 nm drops by 71%, significantly more than the drop in wire capaci-

tance. According to Equation 3.2, the switch box driver strength will need to be 50% larger, requiring almost double the number of equivalent transistors.

The percentage of power that is attributed to the clock network is very small. This is because the architecture contains only a single clock, wheras commercial architectures typically contain several clocks [23, 24]. At this point, VPR only supports single clock architectures. We expect that once VPR supports multiple clocks, and the architectures are modified to reflect this, the clock network power will increase substantially.

It should be noted that this architecture was optimized for the 45 nm technology. It is likely that a different architecture would be chosen for different CMOS technologies. For example, the segment lengths may be reduced at 22 nm to decrease the buffer sizes. Although this architecture may not be ideal for technologies other than 45 nm, the results are useful in illustrating the trends that occur between CMOS technologies. Understanding these trends is important when designing architectures for future technologies. Furthermore, both the transistor technologies and interconnect capacitances are based on predictive models, and real world technologies may be different.

5.4 Experiment 2: Fracturable LUTs

The second experiment explores the effect of fracturable LUTs on total power dissipation.

5.4.1 Methodology

The baseline architecture is the same 6-LUT design, as used in the previous experiment. We modified the baseline architecture to support fracturable LUTs, so that the 6-LUT can operate as two 5-LUTs. This allows two 5-input (or smaller) logic functions to be packed into each LUT. However, in order for two logic functions to be packed into a single LUT they must share some inputs. The parameter F_i indicates the number of inputs available to the LUT. For example, when $F_i = 7$, the two logic functions packed into the 5-LUTs must share three inputs.

The experiment tests F_i values of 6, 7 and 8. In addition, we test both scaling the number of CLB inputs up with F_i, and leaving the number of inputs constant at 33. Furthermore, this architecture can be modified to include either one or two flip-flops for each LUT. If only a single flip-flop is used, the two outputs of the fractured LUT are multiplexed before connecting to the flip-flop.

For this experiment we used the 45 nm technology model described in the previous section, as well as the same interconnect capacitances.

5.4.2 Results

Table 5.6 lists the results of the experiment. This table includes the baseline, non-fractured architecture, as well as architectures where we varied 1) the LUT input-sharing flexibility, F_i, 2) the number of CLB inputs, and 3) whether there is one or two flip-flops per LUT. The results show the change in total power compared to the baseline architecture, as well as the change in total number of CLBs, average power per CLB, average power per LUT, and total routing power.

F_i	CLB Inputs	Total Power	# CLBs	Power / CLB	Power / LUT	Routing Power
Non-fractured (Baseline)						
6	33	**13 mW**	726	3.8 nW	0.14 nW	9 mW
Fractured, *One* Flip-Flop per LUT						
6	33	**+4.6**%	-7.8%	+12.5%	+7.5%	+5.6%
7	33	**+9.5**%	-8.8%	+25.6%	+8.8%	+9.6%
8	33	**+14.3**%	-9.9%	+38.7%	+9.5%	+12.4%
Fractured, *Two* Flip-Flops per LUT						
6	33	**+3.5**%	-17.2%	+23.5%	+14.8%	+4.5%
7	33	**+6.8**%	-18.9%	+39.3%	+16.9%	+5.5%
8	33	**+10.9**%	-18.9%	+52.6%	+16.7%	+8.2%
Fractured, *One* Flip-Flop per LUT, CLB inputs scale with F_i						
7	39	**+9.7**%	-9.6%	+29.2%	+8.9%	+8.8%
8	44	**+14.3**%	-10.9%	+45.4%	+9.7%	+11.6%
Fractured, *Two* Flip-Flops per LUT, CLB inputs scale with F_i						
7	39	**+6.2**%	-19.2%	+43.2%	+16.7%	+4.4%
8	44	**+11.2**%	-19.6%	+61.3%	+17.5%	+5.9%

Table 5.6: Power of fracturable LUTs.

One might expect that fracturing the LUTs will cause the power usage to decrease, since the circuit will require fewer CLBs. However, the results show the opposite; adding the fracturable LUT feature actually increases power usage. The total power usage increases by 3.5-14% depending on the type of fracturable LUT architecture. Although the number of CLBs is decreasing, both the power consumed per CLB and the routing power is increasing.

5.4.3 Analysis

By adding the fracturable LUT feature, we can pack more logic functions into each CLB. Further modifying the architecture to increase either F_i and/or the number of CLB inputs will increase the likelihood of packing logic functions together. The more logic functions that can be packed together, the fewer CLBs are required to implement the circuit, allowing for a smaller FPGA. However, each of these modifications require changes to the hardware that increase the power demand.

Fracturing LUTs: By adding the fracturable LUT feature, each LUT will now have two output pins instead of one. This increases the CLB crossbar size, since all LUT outputs are fed back into the crossbar. It also doubles the number of CLB output pins, which leads to larger multiplexers in the routing switch boxes. Additionally, LUTs that contain two logic functions will consume more dynamic power than if they implemented just one of the logic functions. These behaviours are evident in the results, as adding the fracturable LUT feature increases the power per CLB, power per LUT, and global routing power.

Increasing F_i: Since the LUT inputs are fed by the CLB crossbar, increasing F_i will cause a linear increase in the crossbar size. The results show that although increasing F_i does reduce the number of CLBs by an additional 1 or 2%, the power per CLB increases rapidly.

Increasing CLB inputs: Each of the CLB inputs is fed into the CLB crossbar. Thus, increasing the number of inputs increases the crossbar size. Additionally, the connection boxes will need to be larger to accommodate the increased number of CLB input pins. The results show that increasing the number of CLB inputs reduces the number of CLBs by an additional 1%, but at the cost of a large increase to the power per CLB.

Flip-Flops per LUT: Adding a second flip-flop per LUT allows for many more logic functions to be packed together, at only the power cost of the second flip-flop. The results show that the increase in power over the baseline is a lower penalty than when only using one flop-flop.

The lowest power penalty for a fracturable LUT architecture is when there are two flip-flips per LUT, $F_i = 6$, and 33 CLB inputs. In this case the number of CLBs is reduced by 17% over the non-fractured architecture, at a power cost of only 3.5%.

5.5 Summary

This chapter outlined the verification of the power model, as well as experiments performed with the model. When comparing against SPICE simulations, the power model produces dynamic power estimates within 20%, and static power

estimates within 7% of SPICE simulations. The power estimates are accurate across a wide range of component sizes and CMOS technologies. This makes the power model suitable for evaluating trade-offs during architecture and CAD design.

Two experiments were performed using the model. In the first experiment, the VTR benchmarks were tested for a 6-input LUT, 10 LUTs per CLB architecture. The results show that on average, for a 45 nm technology, 73% of the power consumption is due to the routing fabric, 21% from CLBs, and 3% from the clock network. In the second experiment we modified the architecture to add fracturable LUTs. This resulted in a 3.5-14% increase in power consumption, depending on the type of architecture. The best fracturable LUT architecture reduced the number of CLBs by 17%, at only a 3.5% increase in power.

Chapter 6

Conclusions

A new power model has been developed which can provide power estimates for modern FPGA architectures, not supported by previous power models. This includes support for fracturable LUTs, hard-blocks, and user-defined logic blocks. It is designed to operate with modern CMOS technologies, ranging into the tens of nanometres. The model is integrated into VTR, the newest academic CAD flow. This allows researchers to test the power characteristics of new architectures, as well as new CAD algorithms.

Chapter 2 outlined FPGA architectures, their associated CAD tools, and power estimation techniques. Like past models, this model uses a probabilistic approach to power estimation. This allows for detailed estimation, without the high computation requirements of simulation.

Chapters 3 and 4 provided details of the new model. A new architecture generator is developed, which is capable of transforming arbitrary user-described

architectures into basic circuit components, comprised of inverters, multiplexers and wires. Once decomposed into basic components, the power estimation is performed. Estimates are made for dynamic power, consisting of switching and short-circuit power, as well as static power, which consists of subthreshold and gate leakage. These estimates depend on signal activities, determined using the ACE-2.0 [55] tool, and transistor characteristics, which are automatically extracted from SPICE simulations.

Chapter 5 provides verification of the model, and results of experiments. The model was verified against SPICE for a variety of circuit components, sizes and transistor technologies. Dynamic power estimations are within 20% of SPICE, and static power estimations are within 5%. This accuracy makes the model suitable for evaluating and comparing power requirements of different FPGA architectures. Once verified, we used the model to test power characteristics of common architectures. Results show that for a 45 nm 6-LUT, 10 LUTs per CLB architecture, 73% of power usage is due to the routing fabric, 21% due to logic blocks and 3% due to the clock network (single-clock). Results also show that fracturing LUTs increases power consumption by 3.5-14%. This is because fractured LUTs add additional pins to the logic block, increasing both local and global routing requirements.

6.1 Future Work

The power model presented in this work is built on the academic CAD suite, VTR. VTR is an actively developed project, and the developers are constantly

working to add architectural support for more modern FPGA features. Currently in development is support for carry chains and multiple clock networks. The power model will need to be updated to support these features, as well as any others that are developed in the future.

In [10], the CAD algorithms in VPR were modified to be power aware. This work could be updated to support this new model. The results would be interesting as some of the CAD algorithms have changed since the publication of [10]. Furthermore, VTR supports the full CAD flow, from synthesis to routing, while VPR only supported the last steps of the flow. This makes it possible to perform power optimizations at earlier stages, possibly reducing power even further.

Another area of development could be spatial or temporal power estimations. Currently, the model estimates only the average power dissipation. Temporal power estimation would expand power estimates into the time domain, giving information about how power requirements change during circuit operation. It would provide the minimum and maximum power, which are especially relevant for embedded applications. Spatial power estimation would provide details about power requirements for different areas of the FPGA. It could be used to isolate areas of high power, and with modifications to the CAD algorithms, spread power dissipation evenly across the chip, increasing reliability.

Most importantly, this work provides a method for other researchers to test their own ideas for architectures or CAD algorithms.

6.2 Summary of Contributions

In summary, this work has made the following contributions:

1. A new FPGA power model, capable of performing power estimates for all FPGA architectures supported in VPR 6.0. This includes features such as fracturable LUTs and hard blocks. In addition, this model provides a more detailed estimation of static power compared to previous models.

2. The model was verified against SPICE, achieving accuracy within 20% for dynamic and 5% for static power estimates.

3. The model was used to test power characteristics of different architectures, including a study showing that fracturable LUTs increase power by 3.5-14%.

4. The model will be publicly available, and included in the next release of VTR. This allows researchers worldwide to test new architectures and algorithms.

Bibliography

[1] International Technology Roadmap for Semiconductors. 2010 Update Overview. 2010.

[2] International Technology Roadmap for Semiconductors. 2011 Edition Executive Summary. 2011.

[3] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[4] Martin C. Herbordt, Tom VanCourt, Yongfeng Gu, Bharat Sukhwani, Al Conti, Josh Model, and Doug DiSabello. Achieving High Performance with FPGA-Based Computing. *Computer*, 40(3):50–57, March 2007.

[5] Seonil Choi, Ronald Scrofano, Viktor K. Prasanna, and Ju-Wook Jang. Energy-Efficient Signal Processing Using FPGAs. In *International Symposium on Field Programmable Gate Arrays*, pages 225–234, February 2003.

[6] Seonil Choi and Viktor K. Prasanna. Time and Energy Efficient Matrix Factorization using FPGAs. In *International Conference on Field Programmable Logic and Applications*, pages 507–519, September 2003.

[7] Ian Kuon and Jonathan Rose. Measuring the Gap Between FPGAs and ASICs. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, February 2007.

[8] Assem A.M. Bsoul and Steven J.E. Wilton. An FPGA Architecture Supporting Dynamically Controlled Power Gating. In *International Conference on Field Programmable Technology*, pages 1–8, December 2010.

[9] Fei Li, Yan Lin, Lei He, and Jason Cong. Low-power FPGA Using Pre-Defined Dual-Vdd/Dual-Vt Fabrics. In *International Symposium on Field-Programmable Gate Arrays*, pages 42–50, February 2004.

[10] Julien Lamoureux and Steven J. E. Wilton. On the Interaction between Power-Aware Computer-Aided Design Algorithms for Field-Programmable Gate Arrays. *Journal of Low Power Electronics*, 1(2):119–132, August 2005.

[11] Deming Chen, Jason Cong, and Yiping Fan. Low-Power High-Level Synthesis for FPGA Architectures. In *International Symposium on Low Power Electronics and Design*, pages 134–139, August 2003.

[12] Hoang Le and V.K. Prasanna. Scalable High Throughput and Power Efficient IP-Lookup on FPGA. In *International Symposium on Field-Programmable Custom Computing Machines*, pages 167–174, April 2009.

[13] Kara K. W. Poon, Steven J. E. Wilton, and Andy Yan. A Detailed Power Model for Field-Programmable Gate Arrays. *Transactions on Design Automation of Electronic Systems*, 10(2):279–302, April 2005.

[14] Peter Jamieson, Wayne Luk, Steve J.E. Wilton, and George A. Constantinides. An Energy and Power Consumption Analysis of FPGA Routing Architectures. In *International Conference on Field Programmable Technology*, pages 324–327, December 2009.

[15] Vaughn Betz and Jonathan Rose. VPR: A New Packing, Placement and Routing Tool for FPGA research. In *International Conference on Field Programmable Logic and Applications*, pages 213–222, September 1997.

[16] Fei Li and Lei He. Power Modeling and Characteristics of Field Programmable Gate Arrays. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(11):1712–1724, November 2005.

[17] Farid N. Najm. A Survey of Power Estimation Techniques in VLSI Circuits. *Transactions on Very Large Scale Integration (VLSI) Systems*, 2 (4):446–455, December 1994.

[18] Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Anderson.

The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *International Symposium on Field-Programmable Gate Arrays*, pages 77–86, February 2012.

[19] Steven R. Carlough, Pete M. Campbell, Samuel A. Steidl, Atul Garg, Cliff A. Maier, Hans J. Greub, John F. McDonald, and Matthew W. Ernest. *Programmable Logic Devices*. John Wiley & Sons, Inc., 2001.

[20] *CPLDs vs. FPGAs: Comparing High-Capacity Programmable Logic*. Altera, February 1995. Ver. 1.

[21] Scott Hauck and André DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon)*. Morgan Kaufmann, 2007.

[22] *XC3000 Series Field Programmable Gate Arrays (XC3000A/L, XC3100A/L)*. Xilinx, November 2007. Ver 3.1.

[23] *Stratix V Device Handbook, Volume 1: Device Interfaces and Integration*. Altera, June 2012. Ver 1.7.

[24] *7 Series FPGAs Overview*. Xilinx, May 2012. Ver 1.11.

[25] Guy Lemieux and David Lewis. Using Sparse Crossbars Within LUT Clusters. In *International Symposium on Field Programmable Gate Arrays*, pages 59–68, February 2001.

[26] Guy Lemieux, Edmund Lee, Marvin Tom, and Anthony Yu. Directional and Single-Driver Wires in FPGA Interconnect. In *International Conference on Field Programmable Technology*, pages 41–48, December 2004.

[27] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA Architecture: Survey and Challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, February 2008.

[28] Guy Lemieux and David Lewis. *Design of Interconnection Networks for Programmable Logic*. Springer, 2004.

[29] Herman Schmit and Vikas Chandra. FPGA Switch Block Layout and Evaluation. In *International Symposium on Field-Programmable Gate Arrays*, pages 11–18, February 2002.

[30] M. Imran Masud and Steven Wilton. A New Switch Block for Segmented FPGAs. In *Field Programmable Logic and Applications*, pages 274–281. August 1999.

[31] Jianshe He and J. Rose. Advantages of Heterogeneous Logic Block Architecture for FPGAs. In *Custom Integrated Circuits Conference*, pages 7.4.1–7.4.5, May 1993.

[32] *Cyclone V Device Overview*. Altera, June 2012. Ver 2.0.

[33] Scott Hauck, Matthew M. Hosler, and Thomas W. Fry. High-Performance Carry Chains for FPGAs. *Transactions on Very Large Scale Integration (VLSI) Systems*, 8(2):138–147, April 2000.

[34] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Springer, 1999.

[35] Andrew Rushton. *VHDL for Logic Synthesis*. Wiley, 2011.

[36] Rrobert K. Brayton, Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, 78(2):264–300, February 1990.

[37] Peter Jamieson, Kenneth B. Kent, Farnaz Gharibian, and Lesley Shannon. Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research. In *International Symposium on Field-Programmable Custom Computing Machines*, pages 149–156, May 2010.

[38] Alan Mishchenko. ABC: A System for Sequential Synthesis and Verification, 2009. URL http://www.eecs.berkeley.edu/alanmi/abc.

[39] Robert Brayton and Alan Mishchenko. ABC: An Academic Industrial-Strength Verification Tool. In *International Conference on Computer Aided Verification*, pages 24–40, July 2010.

[40] Stephen Jang, Billy Chan, Kevin Chung, and Alan Mishchenko. WireMap: FPGA Technology Mapping for Improved Routability. *Transactions on Reconfigurable Technology and Systems*, 2(2), June 2009.

[41] Jason Luu, Jason Helge Anderson, and Jonathan Scott Rose. Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and

Complex Interconnect. In *International Symposium on Field-Programmable Gate Arrays*, pages 227–236, February 2011.

[42] Scott Kirkpatrick, C. Daniel Gelatt, and Mario Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[43] Larry McMurchie and Carl Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *International Symposium on Field-Programmable Gate Arrays*, pages 111–117, February 1995.

[44] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, Kenneth Kent, and Jonathan Rose. VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling. *Transactions on Reconfigurable Technology and Systems*, 4(4):32:1–32:23, December 2011.

[45] Gary K. Yeap. *Practical Low Power Digital VLSI Design*. Springer, 1997.

[46] Gary K. Yeap. *Low Power VLSI Design and Technology*. World Scientific Pub Co Inc, 1996.

[47] Michael John Sebastian Smith. *Application-Specific Integrated Circuits*. Addison-Wesley Professional, 1997.

[48] Ian Kuon, Aaron Egier, and Jonathan Rose. Design, Layout and Verification of an FPGA Using Automated Tools. In *International Symposium on Field-Programmable Gate Arrays*, pages 215–226, February 2005.

[49] Robert Tjarnstrom. Power Dissipation Estimate by Switch Level Simulation. In *International Symposium on Circuits and Systems*, pages 881–884, May 1989.

[50] Thomas H. Krodel. PowerPlay-Fast Dynamic Power Estimation Based on Logic Simulation. In *International Conference on Computer Design: VLSI in Computers and Processors*, pages 96–100, October 1991.

[51] F. Dresig, P. Lanches, O. Rettig, and U.G. Baitinger. Simulation and Reduction of CMOS Power Dissipation at Logic Level. In *European Conference on Design Automation*, pages 341–346, February 1993.

[52] C.M. Huizer. Power Dissipation Analysis of CMOS VLSI Circuits by means of Switch-Level Simulation. In *European Solid-State Circuits Conference*, pages 61–64, September 1990.

[53] Richard Burch, Farid N. Najm, Ping Yang, and Timothy N. Trick. A Monte Carlo Approach for Power Estimation. *Transactions on Very Large Scale Integration (VLSI) Systems*, 1(1):63–71, March 1993.

[54] Michael G. Xakellis and Farid N. Najm. Statistical Estimation of the Switching Activity in Digital Circuits. In *Design Automation Conference*, pages 728–733, June 1994.

[55] Julien Lamoureux and Steven J.E. Wilton. Activity Estimation for Field-Programmable Gate Arrays. In *International Conference on Field Programmable Logic and Applications*, pages 1–8, August 2006.

[56] Tan-Li Chou, Kaushik Roy, and Sharat Prasad. Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching. In *International Conference on Computer-Aided Design*, pages 300–303, November 1994.

[57] Abhijit Ghosh, Srinivas Devadas, Kurt Keutzer, and Jacob White. Estimation of Average Switching Activity in Combinational and Sequential Circuits. In *Design Automation Conference*, pages 253–259, June 1992.

[58] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective (4th Edition)*. Addison Wesley, 2010.

[59] Sung-Mo (Steve) Kang and Yusuf Leblebici. *CMOS Digital Integrated Circuits Analysis & Design*. McGraw-Hill Science/Engineering/Math, 2002.

[60] Eddie Hung, Steven Wilton, Haile Yu, Thomas Chau, and Philip H.W. Leong. A Detailed Delay Path Model for FPGAs. In *International Conference on Field Programmable Technology*, pages 96–103, December 2009.

[61] Johnny Pihl. Single Ended Swing Restoring Pass Transistor Cells for Logic Synthesis and Optimization. In *International Symposium on Circuits and Systems*, pages 41–44, May 1998.

[62] Julien Lamoureux and Steven J. E. Wilton. FPGA Clock Network Architecture: Flexibility vs. Area and Power. In *International Symposium on Field-Programmable Gate Arrays*, pages 101–108, February 2006.

[63] David Hodges, Horace Jackson, and Resve Saleh. *Analysis and Design of Digital Integrated Circuits*. McGraw-Hill Science/Engineering/Math, 2003.

[64] Harry J.M. Veendrick. Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits. *IEEE Journal of Solid-State Circuits*, 19(4):468–473, August 1984.

[65] Keheng Huang, Yu Hu, Xiaowei Li, Bo Liu, Hongjin Liu, and Jian Gong. Off-Path Leakage Power Aware Routing for SRAM-based FPGAs. In *Design, Automation, and Test in Europe*, pages 87–92, March 2012.

[66] Ellen M. Sentovich, Kanwar J. Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical report, EECS Department, University of California, Berkeley, 1992.

[67] Farid N. Najm. Low-pass Filter for Computing the Transition Density in Digital Circuits. *Transactions on Computer-Aided Design*, 13(9): 1123–1131, September 1994.

[68] Synopsys. HSPICE - Accurate Circuit Simulation, July 2012. URL http://www.hspice.com.

[69] Yu Cao. Berkeley Predictive Technology Model, 2008. URL http://ptm.asu.edu/.

[70] *Stratix IV Device Handbook Volume 1: Device Interfaces and Integration*. Altera, September 2012. Ver. 4.6.

[71] Ian Kuon and Jonathan Rose. Automated Transistor Sizing for FPGA Architecture Exploration. In *Design Automation Conference*, pages 792–795, June 2008.

[72] International Technology Roadmap for Semiconductors. 2007 Edition Interconnect. Technical report, 2007.