Activity-based Power Estimation and Characterization of DSP and Multiplier Blocks in FPGAs

by

Nathalie Chan King Choy

B.A.Sc., University of Toronto, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT 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

October, 2006

Abstract

Battery-powered applications and the scaling of process technologies and clock frequencies have made power dissipation a first class concern among FPGA vendors. One approach to reduce power dissipation in FPGAs is to embed coarse-grained fixed-function blocks that implement certain types of functions very efficiently. Commercial FPGAs contain embedded multipliers and "Digital Signal Processing (DSP) blocks" to improve the performance and area efficiency of arithmetic-intensive applications. In order to evaluate the power saved by using these blocks, a power model and tool flow are required.

This thesis describes our development and evaluation of methods to estimate the activity and the power dissipation of FPGA circuits containing embedded multiplier and DSP blocks. Our goal was to find a suitable balance between estimation time, modeling effort, and accuracy. We incorporated our findings to create a power model and CAD tool flow for these circuits. Our tool flow builds upon the Poon power model, and the Versatile Place and Route (VPR) CAD tool, which are both standard academic experimental infrastructure.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Professor Steve Wilton, for teaching me so much these last two years and for providing many opportunities to interact with the FPGA community and to gain valuable teaching experience. I would like to thank everyone in the Socwilton, Soclemieux, and other System-on-Chip groups for their ideas, advice, and good company. Peter Jamieson (from the University of Toronto), Brad, Dipanjan, Julien, and Victor were incredibly helpful when I was getting started with all the tools. Roozbeh and Roberto were also great resources.

I am grateful to Altera and the Natural Sciences and Engineering Research Council of Canada for funding my project and to my defense committee for volunteering their time.

Thank you to Scott and Lesley for your emotional support during the academic rough patches and for knowing exactly what I was going through. Also thank you to Steph and Jess for your emotional support and encouragement during the non-academic rough patches, the roommate antics, and the entertaining adventures.

Finally, I would like to dedicate this work to my family, friends, and my dragon boat family (Swordfish, Roli, and UC Water Dragons crews) who kept me sane through a very intense last two years.

Chapter 1

Introduction

1.1 Motivation

Power dissipation has become increasingly important in the electronics industry. There are two reasons for this:

1. There is a growing number of portable electronic applications, which have battery life constraints

2. Process technologies in the nanometer scale and clock frequencies in the gigahertz range result in very hot chips and increased cooling costs.

These issues are relevant to both Application-Specific Integrated Circuits (ASICs) and to Field-Programmable Gate Arrays (FPGAs). With increasing mask costs for ASICs, FPGAs have become an attractive implementation alternative for low and medium volume production. However, FPGAs lag behind ASICs significantly in their power efficiency. The additional transistors required for programmability result in higher power dissipation per function. Leakage power is also dissipated in both the used and unused parts of FPGAs, and most commercial FPGAs do not have a sleep mode to reduce power in unused parts (The Actel IGLOO flash-based FPGA is an exception [6]).

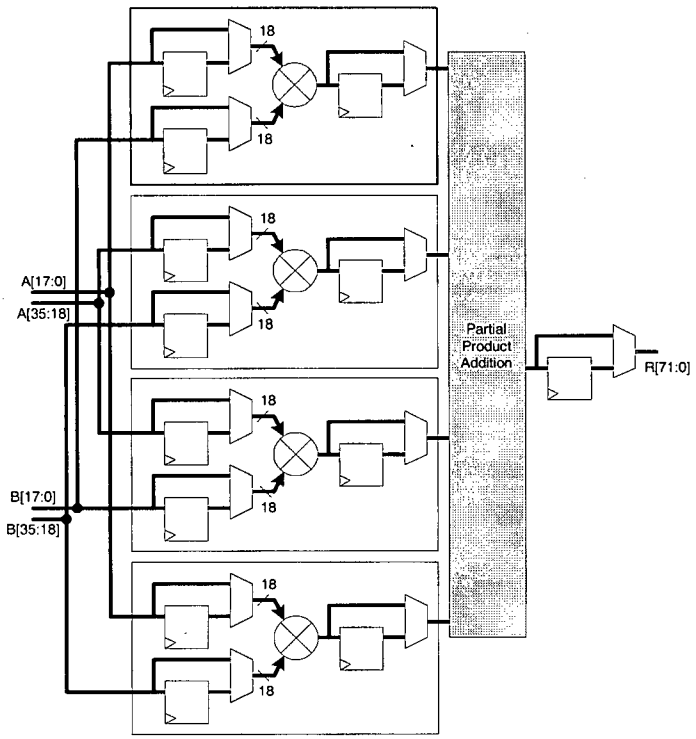

Figure 1.1: 18-bit x 18-bit multipliers combined for 36-bit x 36-bit multiplier

Over the past few years, significant advances have been made at the circuit level [7][8], architecture level [9], and CAD tool level [10][11][12]. Still, current FPGAs have been found to be, on average, 12 times less power efficient than ASICs [13].

A basic FPGA is composed of programmable logic elements (called Look-up Tables or *LUTs*), wire segments, and programmable connections/switches that are typically laid out in a regular pattern. The LUTs in FPGAs typically have 4 to 6 inputs and can implement any boolean function of these inputs. By configuring each LUT and connecting them together using the programmable interconnect, users can implement virtually any digital circuit. To reduce area and improve circuit speed, larger logic blocks are used (typically composed of a collection of 4-10 LUTs); they are called *clusters*.

One promising approach to reducing power in FPGAs at the architectural-level is to embed *coarse-grained fixed-function* blocks that implement certain types of functions very efficiently. Commercial FPGAs contain embedded multipliers and embedded "Digital Signal Processing (DSP) blocks" to improve the performance and area efficiency of arithmetic-intensive applications, such as DSP applications [14][15][16][17]. A simplified example of a DSP block is shown in Figure 1.1; in this diagram, four 18x18-bit multipliers are programmably connected to create a 36x36-bit multiplier, with optional registers at the multiplier inputs, outputs, and adder stage outputs. Other examples of DSP block configurations are adder/subtractors, multiply-accumulators, and multiply-adders. Arithmetic-intensive applications can be implemented in LUTs, but even if LUTs are enhanced with dedicated interconnect for fast carry chains [18] and arithmetic chains [15], significant performance, density, and power improvements can be obtained by implementing the arithmetic parts of the applications in the embedded blocks [19]. Reference [20] found that an average area savings of 55% and an average increase in clock rate of 40.7% could be obtained for floating point applications by embedding floating point DSP blocks.

There are several disadvantages of including DSP blocks in an FPGA fabric. If the blocks are not used (1) the area is wasted, and (2) the blocks will still dissipate leakage power. This makes it important to try to optimize the size and quantity of embedded resources included. However, it is difficult to determine what proportion of the FPGA area should be devoted to embedded blocks because it is difficult to determine what typical usage is for a device that can implement almost any digital circuit.

In order to meaningfully evaluate the power savings that can be obtained by using these blocks, an experimental flow that can estimate the power dissipated by FPGAs containing embedded DSP blocks is required. Commercial tools from FPGA vendors provide power estimation that includes these blocks; however, these tools are tailored for specific devices, and do not provide the flexibility needed to investigate alternative embedded block architectures.

The Versatile Place and Route (VPR) tool suite [21] with the Poon power model [22] has become standard experimental infrastructure to study FPGA architecture, CAD, and power issues (we will refer to the combination of VPR and the Poon power model as PVPR). The estimates from the power model can be used by FPGA architects to evaluate architectures for power efficiency, by FPGA users to make power-aware design decisions, and by FPGA CAD tool developers to create power-aware CAD algorithms. However, PVPR only supports homogeneous FPGA architectures containing LUTs and a routing fabric, and thus is not sufficient to examine architectures containing embedded DSP blocks.

1.2 Research Goals

The objective of this research is to create an enhanced FPGA CAD flow that can be used to evaluate the power dissipation of FPGAs containing embedded DSP and arithmetic blocks. We have imposed the following requirements:

- As PVPR is standard experimental infrastructure for academic FPGA architecture, CAD, and power studies, our power estimation of the DSP and multiplier blocks

must be compatible with the PVPR flow.

- As PVPR results are frequently used to perform architectural evaluations, our power estimation must be fast, to facilitate iterations over tens or hundreds of architectural alternatives and benchmark circuits.

- To facilitate iterations over many architectural alternatives, our method should aim to minimize the modeling effort required when introducing a new architecture for evaluation.

- In order for the power estimates to be meaningful, our method should aim to be as accurate as possible, within the constraints imposed by the previous requirements.

Fast power estimation techniques make use of average quantities, such as probabilities, to reduce runtime. Details are lost during this averaging, resulting in a runtime versus accuracy tradeoff. Similarly, reducing characterization effort typically results in the capturing of fewer details, again resulting in an effort versus accuracy tradeoff. As the requirements that we have set for our flow involve both fast estimation and low characterization effort, we expect that we will be sacrificing some accuracy. Therefore, we must find a suitable balance between speed, characterization effort, and accuracy.

1.3 Research Approach

As will be explained in Chapter 2, power estimation involves two steps: (1) activity estimation, and (2) power estimation using the activities.

Dynamic power dissipation is proportional to how often nodes in a circuit switch values (from logic-0 to logic-1, or logic-1 to logic-0). Activity estimation is the calculation of how much switching is expected at each node in the circuit. In implementing the activity estimation for FPGA architectures containing embedded DSP blocks, we came across the following challenge: it is not clear how to propagate activity calculations through a DSP block. Traditional activity estimation techniques propagate activities through small gates (LUTs) using transition probability or transition density models [23]; however, these approaches do not scale well to embedded blocks with tens (or even hundreds) of inputs.

The next step is to estimate the power dissipated by the FPGA implementing a user's circuit. In implementing algorithms to perform this estimation, we identified the following challenge: once the pin activities of each embedded DSP block are known, it is still not clear how to use them to estimate the power dissipated by the embedded DSP block. There are many possibilities, each providing a different trade-off between power estimation time, modeling effort when a new block is to be investigated, and accuracy. The second technical challenge is to choose a power estimation technique that provides the best balance between these factors.

In this thesis we treat each of these two challenges as a separate problem. For each problem, we develop and evaluate solutions. Then, we use the best of our solutions to each problem to create our experimental CAD flow for evaluating FPGA architectures containing DSP blocks.

1.4 Organization of Thesis

This thesis is organized as follows. Chapter 2 provides an overview of FPGA architectures, the FPGA CAD flow, power, and power estimation. It also describes previous work related to power estimation. Chapter 3 describes the existing PVPR CAD flow and how our DSP and arithmetic block power model fits into this framework. Chapter 4 describes our solutions and their evaluation for the problem of activity estimation. Chapter 5 describes our solutions and their evaluation for the problem of power estimation. Chapter 6 describes how we integrated our solutions from Chapters 4 and 5 to create a *Power Estimation Tool Flow* for evaluating FPGA architectures containing DSP blocks, presents results for two benchmark circuits, and compares the results to those obtained using gate-level simulation. Finally, Chapter 7 summarizes the conclusions and provides suggestions for future work.

Chapter 2

Background and Previous Work

2.1 FPGA Architectures

The most basic building blocks of FPGAs are programmable logic blocks, wire segments, and programmable connections/switches that are typically laid out in a regular pattern. By programming basic functions into each block and connecting them together using the programmable interconnect, users can implement virtually any digital circuit.

A number of architectures have been developed to optimize for criteria such as routability, area efficiency, and timing. Architectures are frequently classified based on their routing architecture because most of the area in an FPGA is devoted to routing. Reference [1] lists the three main categories of commercial FPGA architectures that were available when that book was written: island-style, row-based, and hierarchical. Today, platform-style FPGAs are available, which include coarse-grained embedded components.

This thesis will focus on FPGAs based on island-style architectures. Section 2.1.1 will introduce basic island-style architecture components, Section 2.1.2 will describe modern enhancements to the basic components, and Section 2.1.3 will describe platform-style FPGAs. The framework that we build upon is described in Chapter 3.

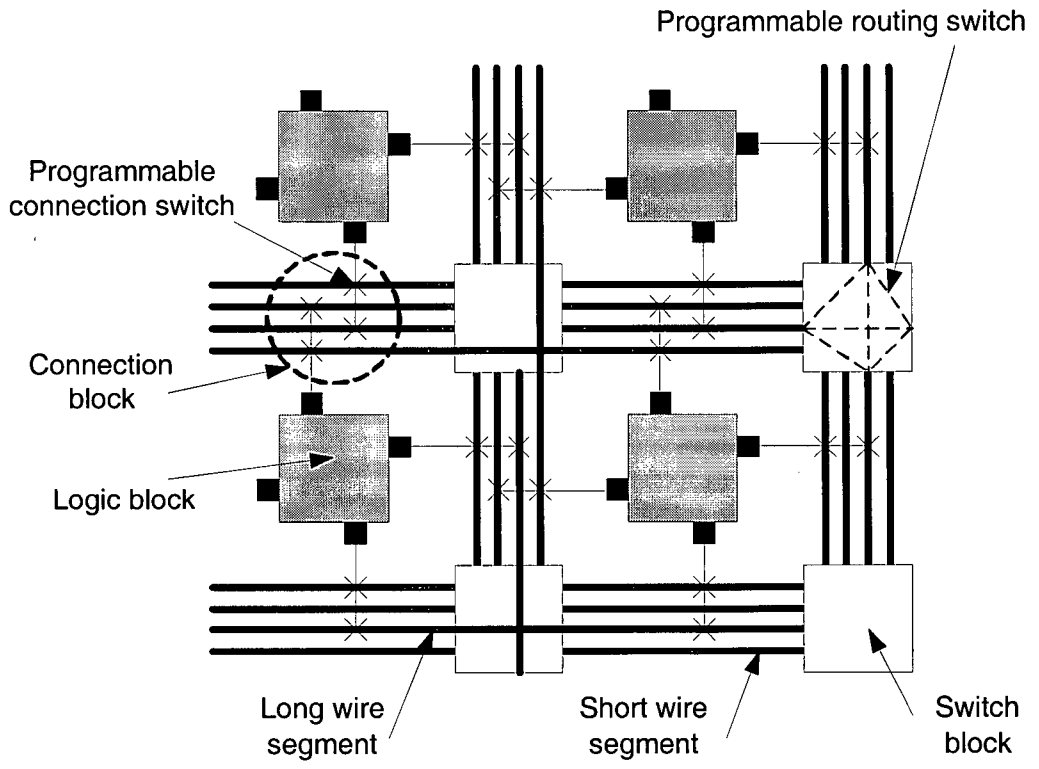

Figure 2.1: Island-style Architecture (from [1])

2.1.1 Island-style Architectures

Figure 2.1 shows an island-style architecture, where programmable logic blocks are "islands" in a "sea" of programmable routing fabric [5]. In the traditional island-style architecture, the logic blocks are clusters of look-up tables (LUTs). The routing fabric is composed of wires, programmable connection blocks, and programmable switch blocks.

Look-up Tables

Look-up tables (LUTs) are the most basic elements for implementing logic in an FPGA. Figure 2.2 shows a 2-input LUT. A K-input LUT (K-LUT) is a memory with 2^K bits, K

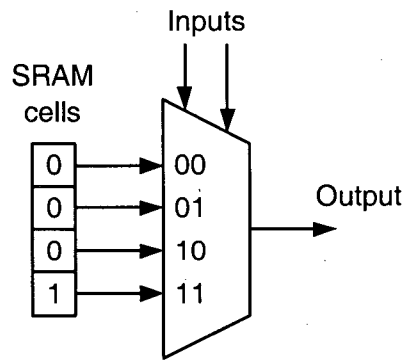

Figure 2.2: 2-input LUT (from [2])

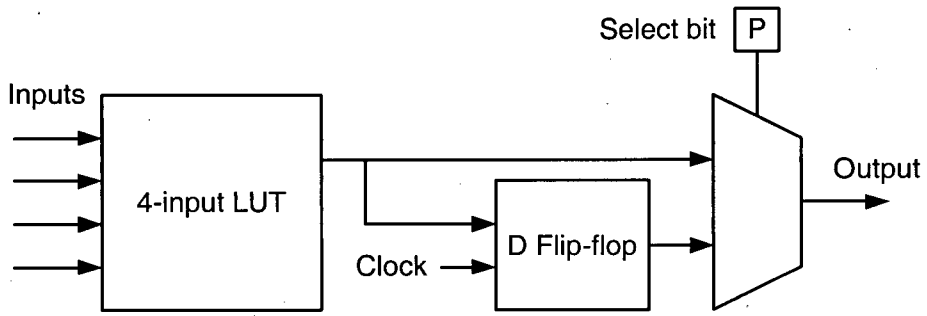

Figure 2.3: LUT with Flip-flop (from [2])

address inputs, and one output port. By setting the memory bits, this structure can be used to implement any combinational function of K inputs. Typical values for K are 4-6.

To implement sequential logic, a LUT is typically paired with a flip-flop to form a *Basic Logic Element (BLE)*, as shown in Figure 2.3. The output of the BLE is selected from either the registered or the un-registered version of the LUT output, depending on the select bit of the output multiplexer.

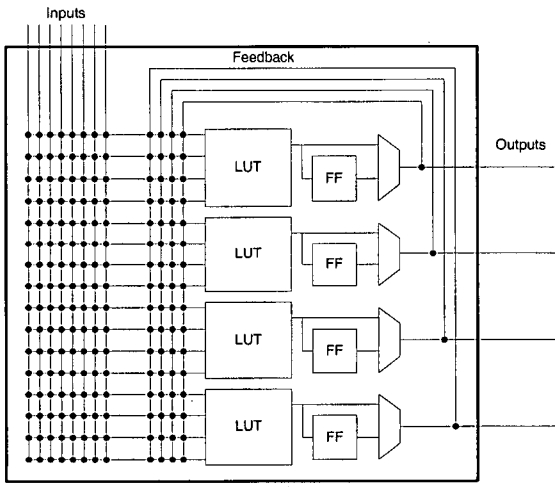

Figure 2.4: Cluster-based logic block

Cluster-based Logic Blocks

The use of larger logic blocks helps to increase circuit speed and reduces circuit area and CAD tool processing time. Unfortunately, LUT complexity grows exponentially with the number of inputs. The use of cluster-based logic blocks addresses this problem. Clusters are typically composed of a number of BLEs, internal cluster routing, and possibly specialized internal cluster connections, such as carry chains. Within a cluster, BLE inputs are typically connected to the cluster inputs and BLE outputs by a multiplexer-based crossbar [2].

An example of a cluster-based logic block is shown in Figure 2.4. It has 8 inputs, contains 4 BLEs, and has 4 outputs. The LUT inputs can be connected to either the cluster inputs or the BLE outputs via the internal routing. By grouping BLEs that share signals, placement and routing processing time is reduced because the number of inter-block connections is reduced. This internal routing is also shorter and faster than

inter-block connections, reducing circuit delay.

Routing

Logic blocks are connected together and to I/O resources using a routing "fabric" that is composed of:

- pre-fabricated metal tracks
- programmable switch blocks
- programmable connection blocks

Figure 2.1 shows the components of the routing fabric. The tracks are arranged in channels, typically horizontally and vertically, to form a grid pattern. Wires run along these tracks. The connection blocks connect the wires to the inputs and outputs of logic blocks adjacent to the channel. The switch blocks connect the horizontal and vertical wires together to form longer wires and make turns. It should be noted that, although we consider the switch blocks and connection blocks as separate entities, they are often not separate in the FPGA circuit layout.

The programmable switches in the switch blocks and connection blocks can be either buffered or unbuffered. Typically they are buffered to reduce delays in long connections; however, this increases routing area. Buffered switches can be either unidirectional or bidirectional. Modern FPGAs use unidirectional switches to get better delay optimization and a denser routing fabric [24]. Figure 2.5 shows examples of switches typically found in FPGAs: (a) unbuffered, (b) buffered unidirectional, and (c) buffered bidirectional.

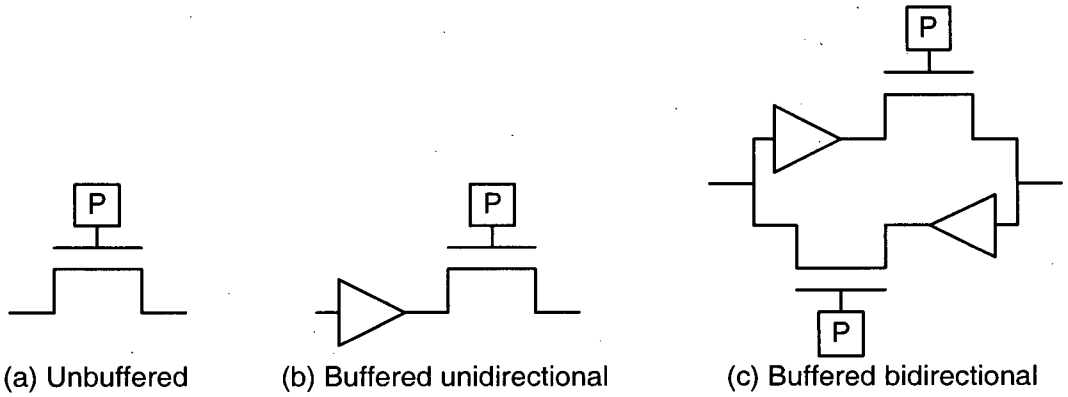

Figure 2.5: Programmable Switches

The routing wires inside an FPGA can be categorized as either (a) segmented local routing for connections between logic blocks, or (b) dedicated routing for global networks, such as clocks or reset/preset lines. The local routing segments can span a single logic block or multiple blocks; typically, not all segments are of the same length. The dedicated routing tracks are designed to ensure low-skew transmission. Commercial FPGAs typically also include phase-locked loops (PLLs) and delay-locked loops (DLLs) for clock de-skew on these dedicated lines. They also may include clock multiplexers and clock management circuitry to reduce power consumption by disabling unused parts of the clock network [25].

2.1.2 Enhanced LUT Architectures and Carry Chains

Commercial FPGAs contain improvements to the basic clustered K-LUT and interconnect FPGAs described in Section 2.1.1.

Adaptive Logic Module (ALM)

The Altera *Adaptive Logic Module (ALM)* is a BLE that can implement a single function of six inputs or two functions that share four inputs [15]. It is based on research that found that 6-LUTs have better area-delay performance than the typical 4-LUT, but can result in wasted resources if many of the logic functions do not require six inputs.

Configurable Logic Block (CLB)

The Xilinx Virtex-5 CLB is a BLE based on a 6-LUT with two outputs so that it can either implement a single function of six inputs or two functions of five (out of the same six) inputs [26]. As with the ALM, it is based on research showing that 6-LUTs have better performance, but may waste resources.

Carry chains

Carry chains are dedicated connections between logic blocks that aid in the efficient implementation of arithmetic operations. They also can be used in the efficient implementation of logical operations, such as parity and comparison. Fast carry chains are important because the critical path for these operations is often through the carry.

Figure 2.6 shows a simple carry chain architecture. Each 4-LUT in a BLE can be fractured to implement two 3-LUTs; this is sufficient to implement both the sum and carry, given two input bits (a and b) and a carry input. The carry out signal from one BLE would typically be connected to the carry in of an adjacent BLE using a fast dedicated connection. The Z input is used to break the carry chain before the first bit of

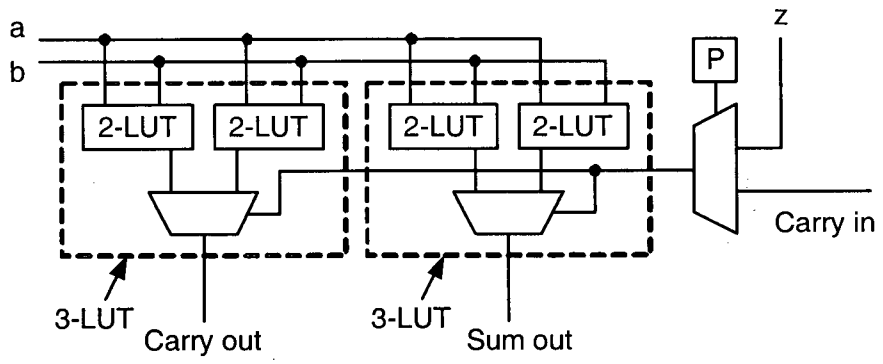

Figure 2.6: Carry Chain Connections to a 4-LUT

an addition.

More complex carry schemes have been published. In [18], carry chains based on carry select, variable block, and Brent-Kung schemes are described; the Brent-Kung scheme is shown to be 3.8 times faster than the simple ripple carry adder in Figure 2.6. The selection of a carry scheme by the FPGA architect involves weighing the area cost of a faster more complex carry chain against the performance benefits, because they are only beneficial when used. Actel and Xilinx devices include support for carry-lookahead and Altera devices include support for carry select capabilities.

2.1.3 Platform-style Architectures

With Moore's Law and the increase in the possible number of transistors on a die, FPGA manufacturers could afford to sacrifice some silicon area to application-specific circuits. In 2000, to increase the efficiency and density of designs containing components such as processors, large memories, and complex arithmetic circuits, FPGA manufacturers began to release Platform-style FPGAs containing dedicated circuitry for these parts.

In the context of FPGAs, *Intellectual Property (IP) cores* are typically reusable circuit modules or high-level building blocks that can be combined to create a system. These cores can be *soft* or *hard*. Soft cores are implemented in LUTs and hard cores are implemented as embedded ASIC blocks in the FPGA. Platform FPGAs contain hard IP cores, such as processors, large memory blocks, fast multipliers, and *Digital Signal Processing (DSP)* blocks. They also contain dedicated interconnect for fast communication between certain types of adjacent cores.

DSP blocks

To improve the density and address the performance requirements of DSP and other arithmetic-intensive applications, FPGA manufacturers typically include dedicated hardware multipliers in their devices. Altera Cyclone II and Xilinx Virtex-II/-II Pro devices contain embedded 18x18-bit multipliers, which can be split into 9x9-bit multipliers [27]. The Virtex-II/-II Pro devices are further optimized with direct connections to the Xilinx block RAM resources for fast access to input operands. Higher-end FPGAs include DSP blocks, which are more complex dedicated hardware blocks optimized for a wider range of DSP applications. Altera Stratix and Stratix II DSP blocks support pipelining, shift registers, and can be configured to implement 9x9-bit, 18x18-bit, or 36x36-bit multipliers that can optionally feed a dedicated adder/subtractor or accumulator [15]. Xilinx Virtex-4 XtremeDSP slices contain a dedicated 18x18-bit 2's complement signed multiplier, adder logic, 48-bit accumulator, and pipeline registers. They also have dedicated connections for cascading DSP slices, with an optional wire-shift, without having to use the slower

general routing fabric [27].

This inclusion of dedicated multipliers or DSP blocks to complement the general logic resources results in a heterogeneous FPGA architecture. Research has considered what could be gained from tuning FPGA architectures to specific application domains, in particular datapaths and DSP.

The work in [28] and [3] is tuned for datapath (arithmetic) circuits. Datapath circuits are often composed of regularly structured components, called *bit-slices*. The authors propose a multi-bit logic block that uses configuration memory sharing to exploit this regularity and save area. In typically-sized cluster-based logic blocks (containing 4 to 10 BLEs), configuration SRAM cells consume 39% to 48% of the total cluster area. If each cluster is used to implement a single bit-slice of the datapath circuit and adjacent clusters are used to implement adjacent bit-slices, the configuration memory for corresponding BLEs in the adjacent slices can be shared. A multi-bit logic block is illustrated in Figure 2.7, indicating resources that can share configuration memory. The authors also propose bus-based connections to exploit this regularity to achieve 14% routing area reduction for implementing datapath circuits. A disadvantage of multi-bit architectures is that if implementation circuits require a different bit-width, some resources may be wasted.

The work in [29] deliberately avoids creating a heterogeneous architecture because the authors found that DSP applications contain both arithmetic and random logic, but that a suitable ratio between arithmetic and random logic is difficult to determine. Instead they develop two "mixed-grain" logic blocks that are suitable for implementing both arithmetic and random logic by looking at properties of arithmetic operations and properties

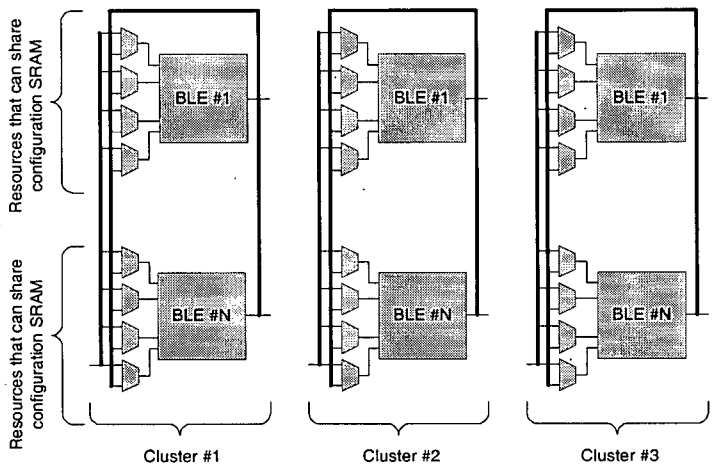

Figure 2.7: Multi-bit Logic Block (from [3])

of the 4-LUT. Their logic blocks are coarse-grained: each block can implement up to 4-bit addition/subtraction, 4 bits of an array multiplier, a 4-bit 2:1 multiplexer, or wide Boolean functions. Each logic block can, alternatively, implement single output random logic functions like a normal LUT. Their architecture reduces configuration memory requirements by a factor of four, which is good for embedded systems or those with dynamic reconfiguration. Compared to arithmetic-optimized FPGA architectures, it offers efficient support of a wider range of DSP applications that vary in the amount of random logic they contain. However, the cost of this flexibility is additional area overhead.

If applications with very little arithmetic are implemented, further area penalties are incurred in the case of all of these architectures with specialized arithmetic support, because most of the arithmetic support logic is left unused.

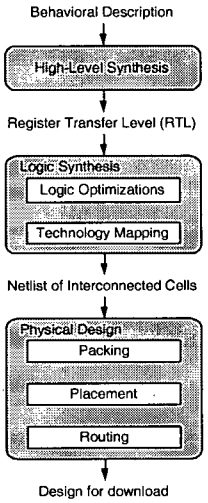

Figure 2.8: Steps in the FPGA CAD Flow

2.2 FPGA CAD Flow

In order to implement large designs on FPGAs, CAD tools are essential. They allow users to work at a high abstraction level, automating optimization and transformation to a low-level implementation. This section will give a brief overview of the steps in the FPGA CAD flow. Figure 2.8 illustrates the steps in the flow.

The input to the top level of the FPGA CAD flow is a behavioral description, typically in the form of Hardware Description Language (HDL) code or a schematic that describes what the circuit does, with little or no reference to its structure.

During high-level synthesis, an initial compilation of the design is done, some initial optimizations are performed, and functional units are scheduled and assigned to the operations required by the circuit [30]. The result of high-level synthesis is a Register Transfer Level (RTL) description of the design. An RTL description is typically written in HDL. RTL code is characterized by a straightforward flow of control, with subcircuits that are

connected together in a simple way [31].

During the optimization stage of logic synthesis, technology-independent optimizations are applied to the logic functions that the RTL describes; the result is a netlist of gates. During the technology mapping stage of logic synthesis, this netlist of gates is mapped to the set of cells that are available to build the functions. In the case of FPGAs, these cells are LUTs and flip-flops. The result is a structural netlist of interconnected LUTs and flip-flops.

Physical design has three stages: (1) packing, (2) placement, and (3) routing. During the packing stage, the LUTs and flip-flops are packed into coarser-grained cluster-based logic blocks, with the goals of improving routability and optimizing circuit speed. Packing together BLEs that share signals minimizes the number of connections between logic blocks, thus enhancing routability. Packing together BLEs likely to be on the critical path makes connections between them go via the fast local interconnect, thus optimizing circuit speed.

During the placement stage, the cluster-based blocks are assigned locations in the FPGA. Goals during placement are to minimize wiring by placing connected blocks close together, to enhance routability by balancing the wiring density across the FPGA, and to maximize circuit speed by putting blocks likely to be on the critical path close together.

Finally during the routing stage, paths are found in the channels for the wires that connect the logic blocks. The result is typically refered to as the placed and routed design. At that point it could be converted to a bitstream for programming an FPGA.

The widely used Versatile Place and Route (VPR) academic CAD tool for packing,

placement, and routing will be described in Chapter 3.

2.3 Power

The formula for the instantaneous power dissipated at time t, $p(t)$, is:

$$p(t) = i(t) \cdot V_{dd} \tag{2.1}$$

where V_{dd} is the supply voltage and $i(t)$ is the instantaneous current drawn from the supply. To obtain the average power over a time interval, we replace the instantaneous current by the average current drawn over that interval in Equation 2.1. Knowing the peak instantaneous power is useful when sizing supply lines, whereas knowing the average power helps in the calculation of battery life and cooling requirements. In this work we focus on average power.

Power dissipation can be broken down into dynamic and static components. Dynamic power is dissipated when a gate is switching; it is due to: (1) the charging and discharging of parasitic capacitances and (2) temporary short circuits between the high and low supply voltage lines. The average dynamic power of the gate is given by the equation

$$P = 0.5 \cdot \alpha C V_{dd}^2 f \tag{2.2}$$

where α is the activity of the gate, C is the parasitic capacitance of the gate, and f is the clock frequency. Static power is dissipated when the gate is not switching; it is due to leakage currents. Typically, in an FPGA, the majority of power dissipated is dynamic [22].

2.4 Power Estimation

It is important to distinguish between *power estimation* and *power measurement.* Power measurement involves obtaining voltage and current values from a real physical apparatus. Power estimation involves predicting what the power dissipation would be, based on a number of assumptions. One reason for performing power estimation is that a physical apparatus is not always available. Another reason is that a wider range of designs can be considered and evaluated more quickly when we are not constrained to using physical implementations. Performing power estimates earlier in the design flow is desirable to help guide design decisions or identify problems in the design.

Power estimation can take place at any stage in the FPGA CAD flow (Figure 2.8). The stages higher in the flow are at a higher abstraction level and do not involve implementation details. As we get lower in the flow, more physical details of the design have been determined. Performing power estimates at the lower stages in the flow will generally give more accurate estimates; however, it will take more computational resources to take into account these physical details. In our work, we will be performing estimates after placement and routing.

Power estimation can be done at different abstraction levels, as shown in Figure 2.9. At each stage, we need the following types of information so that we can perform the power analysis:

1. activity estimates, so we can compute the dynamic power dissipation,

2. a description of what the design looks like - this can be either an architectural

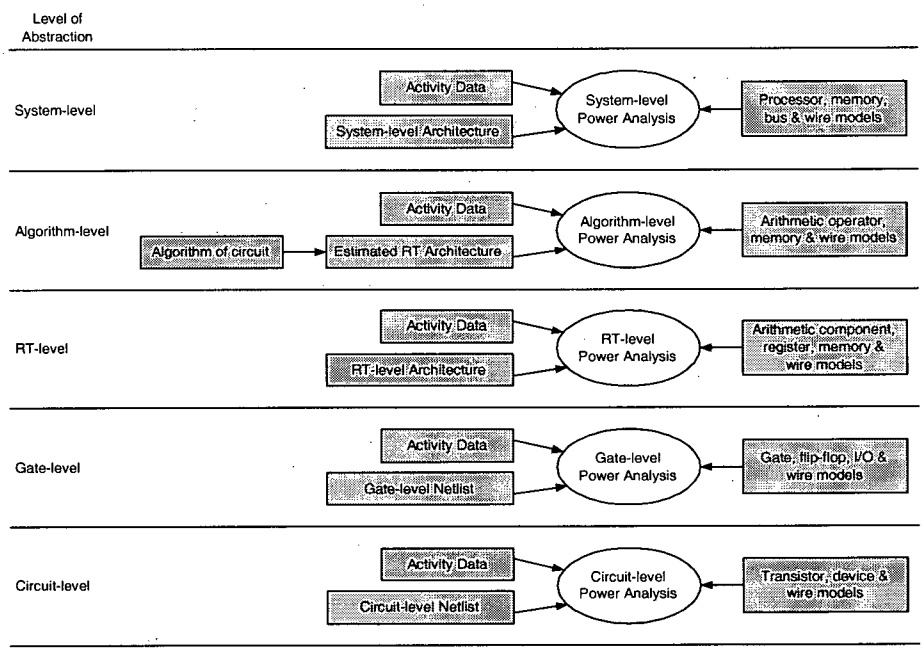

Figure 2.9: Abstraction Levels for Power Analysis (from [4])

description or a netlist - so that we know what components are used and how they are connected,

3. models of these components and connections.

In a platform-style FPGA, the placed and routed netlist contains representations of circuit components at multiple levels of abstraction: LUTs, flip-flops, and wires are essentially at the gate level, while DSP blocks and memories can be thought of as being RTL components, and hard processors can be thought of as being system-level components. For this work, we are interested in the lower three abstraction levels in Figure 2.9: circuit-level, gate-level, and RTL. In the following subsections, we will first describe two categories into which power estimation techniques can be classified. Then we will describe a number of existing techniques at the three lower abstraction levels. Finally we

will describe some FPGA-specific power estimation tools.

2.4.1 Simulation-based vs. Probabilistic

Power estimation techniques can be divided into two categories: (1) simulation-based and (2) probabilistic. Simulation-based techniques simulate the circuit to gather data about the switching of circuit nodes or even determine the waveform of the current being drawn. However, simulation-based techniques require complete and specific information about the input signals. The accuracy of the simulation results is dependent on how realistic the inputs are. Consequently, reference [23] calls simulation-based power estimation techniques *strongly pattern-dependent.*

To avoid the problem of determining complete and specific input signal characteristics, probabilistic techniques are based on typical input signal behaviour. They represent the average behaviour of the inputs using probabilities. Although the estimation is still dependent on the probabilities provided, it is sufficient to supply typical behaviour instead of specific behaviour. Thus [23] calls probabilistic power estimation techniques *weakly pattern-dependent.* Since calculations need only be performed once on the average data, instead of on a large number of simulation inputs, probabilistic techniques tend to require less computational resources than simulation-based techniques; however, some accuracy is sacrificed by of the use of averaging.

2.4.2 Techniques: Circuit-level

Typically, at the circuit-level users seek very precise estimates. As a result, circuit-level techniques tend to be simulation-based, while probabilistic techniques tend to be applied only at the gate-level and above [32].

SPICE

SPICE provides very detailed, low-level simulation data for a circuit. SPICE stands for Simulation Program with Integrated Circuits Emphasis and is a general purpose analog circuit simulator for nonlinear DC, nonlinear transient, and linear AC analyses. It uses mathematical models to represent the devices in the circuit, such as resistors, capacitors, and transistors [33]. This very detailed simulation can result in high accuracy estimates, but it requires substantial computational resources, making it unsuitable for large circuits.

SPICE was used in the creation of the Poon power model, which is discussed in Chapter 3. However, the work in this thesis will be done at higher levels of abstraction, due to runtime and complexity constraints.

2.4.3 Techniques: Gate-level

Simulation-based gate-level analysis is very mature. The most popular type of gate-level analysis uses event-driven logic simulation, where switching events at the inputs of a logic gate trigger events at the output after a pre-defined delay. Probabilistic gate-level techniques exist as well, to reduce the execution time of estimates. We used both simulation-based and probabilistic gate-level techniques in this work.

Synopsys PrimePower

Synopsys PrimePower is a simulation-based dynamic power analysis tool for gate-level power verification that can be used on multimillion-gate designs. It combines gate-level simulation results with delay and capacitance information from technology libraries to get detailed power information. In addition to average power numbers, PrimePower reports instantaneous power consumption in different parts of the design.

Transition Probability

The Transition Probability Technique relates the average dynamic power of nodes to the likelihood that they will switch. To use the Transition Probability Technique, we need the signal probability and the transition probability of each node. The *signal probability*, P_{signal}, of a node is the average fraction of clock cycles in which the steady state value of the node is logic high. The *transition probability*, P_t, of a node is the average fraction of clock cycles in which the steady state value of the node is different from its initial value [23].

The Transition Probability Technique makes some simplifying assumptions: zero-delay, spatial independence of inputs and internal nodes, and temporal independence of signal values. The assumption of *zero-delay* means there is, at most, a single transition of each signal per clock cycle; in reality, there are delays and they can cause the output of a gate to transition multiple times before settling at its final value for the clock cycle. The assumption of *spatial independence* means we assume that there is no correlation between nodes, although, in reality, the value of one signal may affect the value of another

signal, in the same cycle. The assumption is made because calculating the correlation between signals for a large circuit is prohibitively expensive. The assumption of *temporal independence* means that we assume, for a given signal, that values in consecutive clock cycles are independent of each other.

With those assumptions, the average power can be calculated using Equation 2.3:

$$P = 0.5 \cdot V_{dd}^2 f \sum_{all\ nodes} C_i P_{t\_i} \tag{2.3}$$

where V_{dd} is the supply voltage, f is the clock frequency of the circuit, C_i is the total capacitance at node i, and P_{t_i} is the transition probability at node i. Because of the zero-delay assumption, Equation 2.3 only gives a lower bound on the power - unmatched delays cause multiple transitions at gate outputs. With the temporal independence assumption, the transition probability can be calculated from the signal probability using Equation 2.4:

$$P_t = 2 \cdot P_{signal}(1 - P_{signal}) \tag{2.4}$$

Transition Density

The Transition Density Technique is more accurate than the Transition Probability Technique and more computationally efficient than event-driven logic simulation. The advantage of the Transition Density Technique over the Transition Probability Technique is that it distinguishes between multiple transitions of a node in a single cycle, making it more accurate. Switching activity can also be thought of as transition density, $D(x)$ (for

node x), which is the average number of transitions of node x per unit time. Formally, it is given by Equation 2.5,

$$D(x) \triangleq \lim_{T \to \infty} \frac{n_x(T)}{T} \tag{2.5}$$

where T is the length of the time interval and $n_x(T)$ is the number of transitions in the time interval of length T.

Given the transition density of all the nodes, the average power dissipation can be calculated using Equation 2.6:

$$P = 0.5 \cdot V_{dd}^2 \sum_{all\ nodes} C_i D(x_i) \tag{2.6}$$

where V_{dd} is the supply voltage, C_i is the capacitance at node i, and $D(x_i)$ is the transition density of node i.

There are two important quantities in the calculation of activities for all nodes in the circuit using the Transition Density model: *static probability* and *transition density.* Static probability is the probability that the signal is high. To calculate the activity of each node in the circuit, the transition density for each node is computed, gate-by-gate, going from the primary inputs to the primary outputs. If we assume that all inputs are uncorrelated, we can use the relationship

$$D(y) = \sum_{all\ input\ pins} P\left(\frac{\partial f(x)}{\partial x_i}\right) D(x_i) \tag{2.7}$$

where $f(x)$ is the logic function of the gate, $\frac{\partial f(x)}{\partial x_i} = f(x)|_{x_i=1} \oplus f(x)|_{x_i=0}$ is the boolean

difference at the output port with respect to each input x_i, $D(x_i)$ is the transition density at input x_i and $D(y)$ is the transition density at the output, y. $P\left(\frac{\partial f(x)}{\partial x_i}\right)$ can be calculated from the static probabilities of the inputs x_i using the relationships:

- $P(\overline{X}) = 1 - P(X)$
- $P(XY) = P(X) \cdot P(Y)$
- $P(X+Y) = P(X) + P(Y) - P(X) \cdot P(Y)$

where $P(X)$ is $P_1(X)$, the static probability of X.

Lag-one Model

The Transition Density model assumes that there is no temporal correlation. The purpose of using the lag-one model is to relax this assumption; the lag-one model assumes that the current value of a signal may depend on the value *immediately preceding* it. Using the lag-one model, the switching probability can be calculated using Equation 2.8:

$$P = \sum_{x_i \in X_1} \left[P(x_i) \cdot \sum_{x_j \in X_0} P(x_i, x_j) \right] + \sum_{x_i \in X_0} \left[P(x_i) \cdot \sum_{x_j \in X_1} P(x_i, x_j) \right] \tag{2.8}$$

For a boolean function, f, X_1 is the set of input states such that $f(x_i) = 1 \; \forall \; x_i \in X_1$ and X_0 is the set of input states such that $f(x_j) = 0 \; \forall \; x_j \in X_0$, $P(x_i)$ is the probability that the current input state is x_i, and $P(x_i, x_j)$ is the probability that the input state will be x_j at the end of a clock cycle if the state was x_i at the beginning of the clock cycle. This equation represents the summation of probabilities over all pairs of input states x_i, x_j such that $f(x_i) = \bar{f}(x_j)$, where an input state is a row of the truth table for f.

2.4.4 Techniques: RT-level

RT-level estimators are typically based on macro-modeling. Macro-modeling involves creating power macro-models for the basic functional components in the RTL libraries and characterizing them [4]. The user of an RTL estimator sees the macro-models as black boxes. However, creating a macro-model of a component involves characterizing its representation at a lower level of abstraction [32]. For example, to do power characterization for an adder, we might estimate its gate-level implementation and use information about the gates to derive overall values for its power characteristics.

Although power estimates at higher levels of abstraction are less accurate, they still provide valuable information. With the increase in the size and complexity of designs, it is desirable for designers to be able to estimate the power at a high level of abstraction so that the information can guide early architectural decisions. Another motivating factor is that the largest power reductions often come from architectural and algorithmic modifications [34], which are least costly to make early in the design flow. However, although RTL estimators are available in commercial tools, they have not yet gained widespread acceptance in design practice. Reference [4] attributes this to the difficulty of quantifying the accuracy gap between gate-level and RTL power estimation in an industrial setting. Another deterrent noted by reference [4] is the fact that a large amount of characterization must be done to make a library of macro-models; this process must be automated to be efficient.

Dual Bit Type Method

The Dual Bit Type (DBT) Method [34] is an architecture-level strategy for generating accurate black-box models of datapath power consumption. Its creators note that, while typical strategies quantify activity and physical capacitance for their estimates, the strategies do not account for the effect of signal statistics on the activity. In particular, the authors identify the correlation between sign bits of two's complement operands as being an important source of error when using the assumption of randomized inputs to the block being modeled. As an example, consider an FPGA with 8-bit adders and a user circuit where all the operands are 5 bits wide. The lower 5 bits could be adequately represented by uniform white noise (UWN) inputs, but the upper 3 bits would always be identical (correlated) sign-extension values.

The creators of the DBT method propose to account for two input bit types: (1) correlated sign bits, and (2) UWN operand bits. Recall Equation 2.2:

$$P = 0.5 \cdot \alpha C V_{dd}^2 f \tag{2.9}$$

To account for the two bit types, instead of using a single capacitative coefficient based on UWN inputs, they use multiple capacitative coefficients that account for transitions on each type of data on each input to the block. However, a two-input single-function module requires 73 capacitive coefficients; the number increases for semi-configurable multi-function DSP blocks that are found in FPGAs.

Entropy-based

In reference [35], the authors propose to characterize the average switching activity of a module by using the average switching activity of a *typical signal line* in the module. Their goal is to obtain an acceptable estimate with a limited number of design details and at a significantly lower computational cost. They derive simple closed form expressions to approximate the switching activity in the RTL blocks using the concepts of entropy and informational energy. However, to manage the complexity of their calculations, they make the following simpifying assumptions:

- Simplified, uniform network structure: Each level of the circuit has the same number of nodes and all the gates on each level are assumed to get their inputs from the previous level.
- Asymptotic network depth: The number of levels in the circuit is large enough to be considered infinity.

Unfortunately, DSP and arithmetic blocks in FPGAs do not have a uniform network structure and are not so large that we can approximate their network depth as infinite.

2.4.5 FPGA-specific Power Estimation Tools

Spreadsheets

The most accurate power estimation results for an FPGA design will be after the design has been implemented (i.e. placed, routed, and then simulated with accurate stimulus vectors). However, it is valuable to understand the impact of early high-level design

decisions on power dissipation. Power estimation spreadsheets can be used in the pre-implementation phase to obtain a rough idea of power dissipation for a design.

These spreadsheets contain detailed device data constants from FPGA manufacturer datasheets. The user enters environmental conditions, voltage and clock information, logic utilization, and toggle rates. Early spreadsheets only calculated total power dissipated for voltage sources and components [36][37]. The spreadsheets for the latest FPGA families from Altera and Xilinx are newer and calculate the static, dynamic, and total power consumption [38][39]. The Xilinx Virtex-4 spreadsheet also provides graphical representations of power, voltage, and temperature relationships and power used by each type of component.

It should be noted that these spreadsheets compute power in a device-specific manner, based on constants. The user is expected to provide toggle activity information for each block, but (s)he might not know what values to use at such an early stage.

CAD tools

Industrial CAD tools that offer more accuracy than spreadsheets are Xilinx XPower and Altera PowerPlay Power Analyzer. They are used in the implementation phase, when design details such as placement and routing have been established.

XPower requires either user supplied toggle rates, as with the spreadsheets, or post-implementation simulation data to estimate the power consumed [40]. PowerPlay is similar, but also includes (for some device families) vectorless activity estimation to statistically estimate the signal activity of a node using the activities of the signals feeding the

node and the logic function implemented by the node [41]. The limitation of these tools is that they apply only to some of the Altera and Xilinx devices.

The Poon power model is a freely available, detailed, flexible power model that has been integrated into the Versatile Place and Route (VPR) CAD tool. It estimates the dynamic, short-circuit, and leakage power consumed for a wide variety of user-specified FPGA architectures. It is described in detail in Chapter 3.

2.5 Focus and Contribution of Thesis

Section 2.1 describes the basic island-style architecture and the improvements that exist in commercial FPGAs to improve density and speed. Unfortunately, available academic power estimation tools only support basic island-style architecture components. The goal of this research project is to enable fast and accurate estimation of power dissipated in FPGA designs that include embedded multiplier and DSP blocks (for the remainder of the thesis, both embedded multipliers and DSP blocks will be referred to as DSP blocks). Our project uses both simulation-based and probabilistic information at the gate-level to create a *Power Estimation Tool Flow* that includes automated RT-level embedded DSP block macro-model characterization.

This work builds upon the Poon power model and the widely used VPR CAD Tool, which are described in Chapter 3.

The contributions of this thesis can be summarized as:

1. Identification of a fast and accurate technique to estimate the switching activity of an embedded DSP block

2. Identification of a fast and accurate technique to estimate power dissipated by DSP blocks

3. A tool flow for estimating embedded DSP block power in the context of FPGA designs.

The impact of our enhanced tool flow is threefold; the existence of a freely available, architecturally flexible FPGA CAD tool that includes power modeling for embedded DSP blocks enables:

1. the investigation of power-aware architectures containing embedded DSP blocks

2. the investigation of power-aware CAD algorithms for FPGA circuits containing embedded DSP blocks

3. the incorporation of power tradeoffs in the design of user circuits.

Chapter 3

Framework

This chapter introduces the existing experimental CAD tool suite that forms the basis of our work. Section 3.1 describes the flow of the framework. Section 3.2 describes the T-VPack tool for packing basic logic elements into cluster-based logic blocks and the original VPR CAD tool. Section 3.3 describes the Poon power model and the improved activity estimation tool, ACE-2.0. Section 3.4 describes how our work fits into the existing framework and the requirements for our work.

3.1 Overall Flow

Our work is based upon the VPR CAD tool suite, enhanced with the Poon power model (together PVPR). Frequently, "VPR" refers to the pair of tools T-VPack and VPR, since they are typically used together. In this thesis, we will do the same. Figure 3.1 illustrates the steps in the PVPR tool flow. The left side is the original VPR flow. For the Poon power model, activity estimation was added; this is shown to the right of the original VPR flow.

The first input to the flow is a netlist describing the user's circuit. This netlist must be pre-processed to generate the correct data and data format required by VPR. This pre-

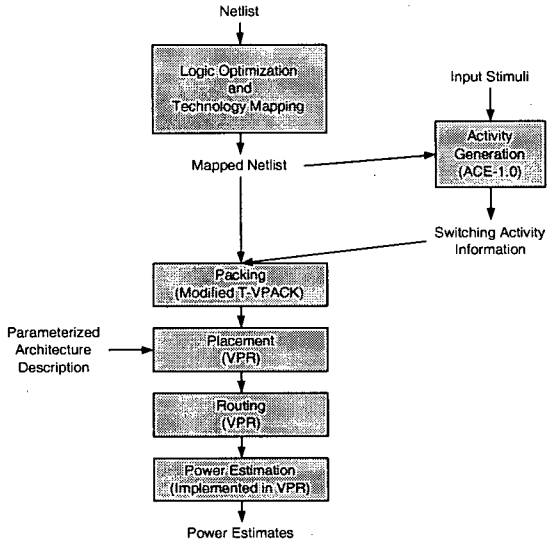

Figure 3.1: Overall CAD Flow (from [5])

processing involves logic optimization using SIS [42] and technology mapping to LUTs and flip-flops using FlowMap+FlowPack [43]. The result of technology mapping is a netlist mapped to the desired FPGA architecture. This mapped netlist and input stimuli are inputs to the activity estimation module, ACE-1.0, which is based on the Transition Density model. The output of ACE-1.0 is switching activity information for each node in the mapped netlist. The mapped netlist, switching activity information, and cluster architecture parameters are then input to T-VPack, which packs the LUTs and flip-flops into cluster-based logic blocks. The cluster-based blocks are placed using the VPR placement engine. The connections between the placed blocks are then routed using the VPR routing engine.

VPR generates reports of placement and routing statistics. Architectural investigations can then be performed by varying the parameters in the parameterized architecture

description and examining the resulting statistics. Algorithmic investigations can also be performed by making modifications to the packing, placement or routing engines and examining the statistics for a set of benchmark circuits. The Poon power model adds the generation of power statistics for the clock, logic, and interconnect to VPR. These power statistics can be used in both architectural and algorithmic studies for basic Island-style FPGA architectures.

3.2 Versatile Place and Route (VPR)

VPR is a freely available CAD tool that is widely used for performing FPGA architectural studies. It is composed of a packing tool, a placement and routing engine, and a detailed area and delay model.

3.2.1 Architectural Assumptions

There are a large number of architectural alternatives for FPGAs and not all are supported by VPR. VPR targets SRAM-based Island-style FPGAs with cluster-based logic blocks and perimeter I/O. Each SRAM cell is made of six minimum-sized transistors with gate voltage boosting to overcome the Body effect. Four types of switch block architectures are supported for the programmable connection of routing tracks: Disjoint [44], Universal [45], Wilton [46], and Imran [47].

3.2.2 T-VPack

T-VPack is a timing-driven CAD tool; it takes a circuit netlist that has been technology mapped to LUTs and flip-flops and packs these basic logic elements into larger cluster-based logic blocks. Before the placement stage of VPR, the circuit netlist is processed using the T-VPack tool. As described in Section 2.1.1, the use of coarse-grained logic blocks results in faster, denser circuits, and in faster place and route runtimes.

T-VPack has the optimization goals of:

- Minimizing the number of inter-cluster connections on the critical path of the circuit
- Reducing the number of connections required between clusters by minimizing the number of inputs to the clusters
- Minimizing the number of clusters needed

3.2.3 Placement and Routing Engine

The placement tool assigns the cluster-based logic blocks to locations in the FPGA. The FPGA is modeled as a set of legal locations where logic blocks or I/O pads can be placed. An initial random placement is constructed, then simulated annealing is used to improve the solution. Optimization goals involve minimizing wiring and maximizing circuit speed. As will be described in Section 6.1.2, we modified the placement tool to place DSP blocks as well.

Once placement is complete, the routing tool determines which programmable switches to turn on to make the required inter-logic block connections in the FPGA. VPR represents

the routing architecture of the FPGA as a directed graph called the *routing resource graph.* Two routing algorithms are available: a purely routability-driven algorithm and a timing- and routability-driven algorithm.

3.2.4 Architectural Flexibility

The reason for VPR's versatility is its flexible representation of architectures that the user specifies in an architecture file. The following features can be specified:

- Logic block architecture
- Detailed routing architecture
- Channel width
- Timing analysis parameters
- Process technology parameters and capacitances

3.3 Poon Power Model

3.3.1 Architectural Assumptions

As the Poon model is incorporated into VPR, it uses the architectural assumptions made by VPR. However, the original version of VPR assumes that the clock and other global signals are implemented using special dedicated resources. The version of VPR enhanced with the Poon model assumes an H-Tree clock distribution network and uses the total capacitance of the clock network for power estimation.

3.3.2 Activity Estimation

In the Poon model, the first step is to estimate the activities of the nodes in the FPGA. The activity estimation tool for the power model is called ACE. To distinguish between major revisions of the tool, the original will be referred to as ACE-1.0 and its sucessor will be referred to as ACE-2.0 [48]. This section will describe the techniques used for estimation in ACE-1.0 and ACE-2.0.

ACE-1.0

ACE-1.0 is the original activity estimation tool for the Poon model. It estimates the static probability (P_1), switching probability (P_s), and switching activity (A_s) for combinational and sequential gate-level circuits using the Transition Density signal model. The original Transition Density model only handles combinational circuits, but was enhanced to support sequential circuits. To support circuits with sequential feedback, an iterative technique is used to update the switching probabilities at the output of the flip-flops, using the expressions $P_1(Q) = P_1(D)$ and $P_s(Q) = 2 \cdot P_1(D) \cdot (1 - P_1(D))$. The original Transition Density model was also enhanced to account for logic gate inertial delays by adding an analytical low-pass filter to filter out very short glitches.

The authors of [48] found ACE-1.0 to be inaccurate for large and/or sequential circuits. They found that ACE-1.0 overestimates activities and suggest that the low-pass filter function is insufficient for reducing glitching. They also attribute the poor sequential circuit performance to the simple expressions used in the iterative technique for updating the switching probabilities at the outputs of flip-flops. The next subsection describes the

activity estimator from [48], ACE-2.0, which addresses the weaknesses of ACE-1.0.

ACE-2.0

ACE-2.0 is a faster and more accurate probabilistic activity estimator for the Poon power model. It has three stages that address the weaknesses of ACE-1.0:

1. Simulation of sequential feedback loops
2. Calculation of P_1 and P_s values for nodes not in sequential feedback loops using the Lag-one model
3. Calculation of A_s using a probabilistic technique that accounts for glitching

The first stage improves the accuracy of activity estimation in sequential circuits. Since simulation techniques were avoided because of runtime issues, ACE-2.0 only simulates the logic in sequential feedback loops.

In the second stage, ACE-2.0 obtains the P_1 and P_s values using the Lag-one model for the parts of the circuit not simulated, which produces exact switching probabilities if we assume that inputs are not correlated [48]. ACE-1.0 uses the Transition Density model, which assumes that there is no temporal correlation. The purpose of using the lag-one model is to relax this assumption; the lag-one model assumes that the current value of a signal may depend on the value *immediately preceding* it.

The most efficient known implementation of the Lag-one model uses a Binary Decision Diagram (BDD). However, there is an exponential relationship between BDD size and the number of inputs, making this implementation impractical for large circuits. ACE-2.0 combines BDD pruning with a partial collapsing technique to give smaller BDDs.

In the third stage, ACE-2.0 calculates the switching activities. It uses a generalization of the Lag-one model and accounts for glitching by incorporating the concept of a minimum pulse width for passing glitches.

3.3.3 Power Estimation

The Poon model uses estimated capacitances at the transistor level for each component inside the FPGA. Then, using the capacitance values and switching activity estimates, the average power dissipation is calculated. The model was compared against HSPICE simulations. The Poon model dynamic power estimates were found to be within 4.8% for routing and 8.4% for logic. For leakage, average difference between the estimates and the HSPICE results was 13.4%.

Dynamic power is the dominant component of the total power in an FPGA. The Poon model calculates capacitance values at the transistor level to determine the power dissipation of LUTs, multiplexers, and buffers inside logic blocks. It also uses the metal capacitance of each routing track and the parasitic capacitance of all switches attached to the track, specified using the process technology parameters in the architecture file, to calculate the power dissipated in the FPGA routing. The routing power is a large portion of the dynamic power dissipated. Since the SRAM programming bits in the FPGA do not change value after configuration, they are not included in the dynamic power calculations. The dynamic power is calculated using the equation:

$$P = \sum_{all\ nodes} C_y V_{supply} V_{swing} D(y) f_{clk}/2 \tag{3.1}$$

The short circuit power is modeled as 10% of the dynamic power, based on extensive HSPICE simulations in [5].

The leakage power has two components: reverse bias leakage and subthreshold leakage. As the Poon model was calibrated using a 0.18 μm process technology, it assumes that the reverse bias leakage is negligible. To calculate the subthreshold leakage the Poon model uses the equation:

$$P_{leak} = I_{drain\ (weak\ inversion)} \cdot V_{supply} \tag{3.2}$$

It uses a first order analytical estimation model to estimate the subthreshold current.

3.3.4 Architectural Flexibility

Enhancements for the Poon model add support to the architecture file for the flexible specification of:

- Supply, swing, and gate-source voltage levels
- Leakage and short circuit power parameters
- NMOS and PMOS transistor characteristics
- Clock network architecture parameters

3.4 DSP Block Power Model and Tool Flow

The DSP block power model that we propose is an extension for the PVPR flow. Section 3.4.1 discusses the requirements of the power model we have developed as part of this work. Section 3.4.2 explains where our work fits in to the PVPR flow.

3.4.1 Requirements

As stated earlier, our power model must fit into an existing FPGA CAD tool flow. In order to allow the investigation of future architectures, instead of simply existing commercial architectures, we prefer that this CAD tool be architecturally flexible; it should be possible to specify a wide range of logic block, routing, clock, and DSP block architectures.

In an architectural investigation, many iterations of PVPR are executed to gather data about the impact of varying certain architectural parameters. In order to not hinder the use of PVPR for an investigation requiring tens (or even hundreds) of iterations, our power estimation must be fast. Furthermore, in order for the power estimates from the investigation to be meaningful, they must be accurate.

The previous requirements pertain to the tool flow that is visible to the PVPR user. An important input to the tool flow in Figure 3.4.2 is the *DSP block characterization data*. As described in Section 2.4.4, a deterrent to the use of macro-modeling at the RT-level is the fact that a large amount of characterization must be done to make a library of macro-models; this process must be automated to be efficient. Therefore, to make our tool flow attractive, we must minimize the effort required when adding models for new DSP blocks and automate characterization.

3.4.2 Extending PVPR Flow

Figure 3.2 shows how our model fits into the PVPR flow. Pre-processing of the DSP blocks in the mapped netlist is required before the activities can be generated for the nodes in a user's circuit that contains DSP blocks. Additional characterization data

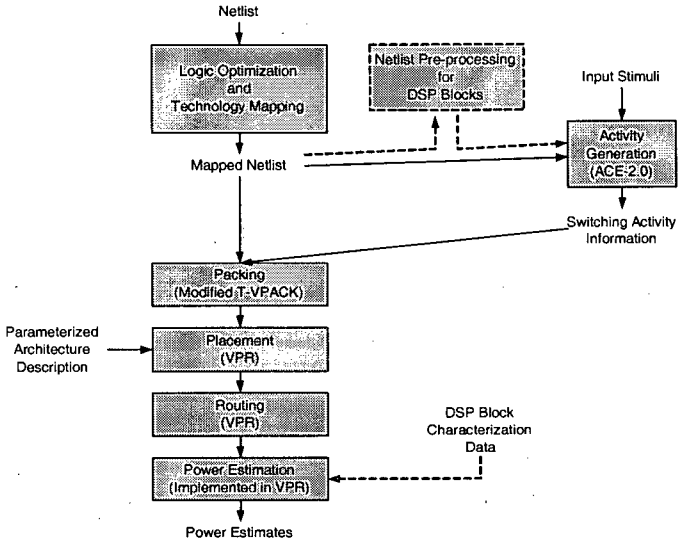

Figure 3.2: CAD Flow Enhanced to Support DSP Block Power Estimation

is also required in order to estimate the power of the DSP blocks in the final stage of processing. The addition of our work to the PVPR flow expands the support of PVPR to FPGA architectures that contain DSP blocks, thus enabling architectural and algorithmic investigations with circuits that contain these blocks.

3.5 Chapter Summary

Our work modifies the widely used PVPR tool flow. PVPR is composed of the VPR engines and the Poon power model. The VPR engines are the T-VPack clustering algorithm, the VPR placement engine, and the VPR routing engine. The Poon power model adds activity estimation and power estimation for logic blocks, interconnect, and the clock. Our work modifies the PVPR flow to add activity estimation for DSP blocks and power estimation of DSP blocks using characterization information. The requirements for our

work are:

- The DSP block power estimation must fit into an existing architecturally flexible FPGA CAD tool flow
- The power estimation must be fast
- The power estimation must be accurate
- The characterization effort must be low and should be automated

As mentioned in Section 1.2, the last three of these requirements are competing factors; fast estimation and low characterization effort will generally lead to less accurate results. Thus, we must find a suitable balance between speed, characterization effort, and accuracy. Details of how we determined accurate methods for performing activity estimation and power estimation for DSP blocks are discussed in Chapters 4 and 5, respectively. The complete automated tool flow, including DSP block characterization, is described in Chapter 6.

Chapter 4

Activity Estimation

4.1 Motivation for developing Activity Estimation Techniques

An important part of estimating the power dissipated in an FPGA is estimating the activity of each connection in the circuit. In the Poon power model, activities are calculated gate-by-gate, starting from the primary inputs. Since each gate (LUT) is small, the Transition Density or Lag-one model can be used to calculate the activity of the output of each LUT as a function of the activity of its inputs. It is not feasible, however, to propagate activities through a DSP block using the Transition Density or Lag-one model since the computation performed using these models (and other related models) is $O(2^k)$ where k is the number of inputs to the block. Thus, a new technique is required. In this chapter, two alternative techniques to estimate the activities of each DSP output pin are considered. The two techniques are compared to determine which is suitable for use within the experimental CAD flow that was developed in this work.

It is important to note that the techniques described below are only being used to estimate the activities of the output pins of each embedded block. Power estimation of

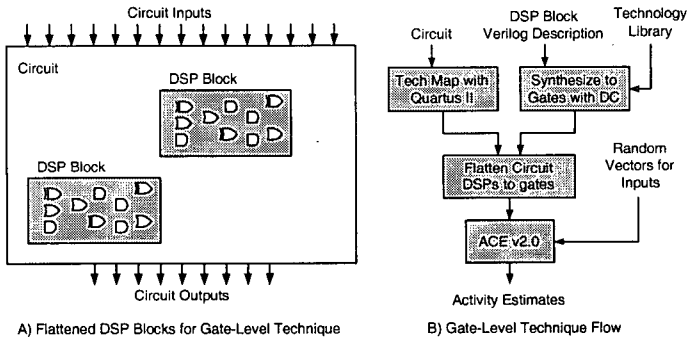

Figure 4.1: Gate-Level Technique

these blocks will be discussed in Chapter 5.

4.2 Techniques for Activity Estimation of an Embedded Block

This section describes two techniques for estimating the activity in circuits containing embedded DSP and multiplier blocks. Although Platform-style FPGAs also contain memories, processors, and other features, we limit our experiments to circuits containing only DSP and multiplier blocks, logic, and interconnect. This ensures that the results we obtain are not obscured by assumptions about the other Platform-style features.

4.2.1 Gate-Level Technique

The first technique is an extension of the Poon model activity estimation method. The embedded blocks are too large to for us to apply the Transition Density or Lag-one model to them directly. To be able to use the Transition Density or Lag-one models, this

technique involves representing the embedded blocks by their gate-level implementations. The flattened circuit would then consist of LUTs (for the parts of the circuits not in the embedded blocks) and the gates that make up each embedded block. The Transition Density or Lag-one model can then be applied directly to this flattened netlist. This technique will be referred to as the *Gate-Level Technique*. Figure 4.1(A) shows the circuit with the flattened DSP blocks in grey and the LUT-and-interconnect part of the circuit in white.

Figure 4.1(B) shows a flow that employs this technique. In this flow, the DSP block to be evaluated is described in Verilog. Synopsys Design Compiler is used to map the block to gates, using TSMC 0.18 μm technology library information. The parent circuit to be considered is technology mapped using Quartus II in order to determine what gets mapped to DSP blocks. The parent circuit is then flattened and the DSP blocks are replaced with their gate-level representations. Then, activity estimation is performed. Currently, the use of Quartus II for technology mapping restricts us to Altera-style DSP blocks; however, Quartus II could be replaced with another technology mapping tool to evaluate non-Altera-style DSP blocks.

It is important to emphasize that we do not modify the netlist that will be implemented in the FPGA. The flattened netlist is generated only during activity estimation.

The principal advantage of this technique is that it accounts for the correlation between the input activities and output activities of the DSP blocks. Another advantage of this technique is that it allows the use of ACE-2.0 to estimate the activity for all the nodes in the circuit, including the DSP block nodes.

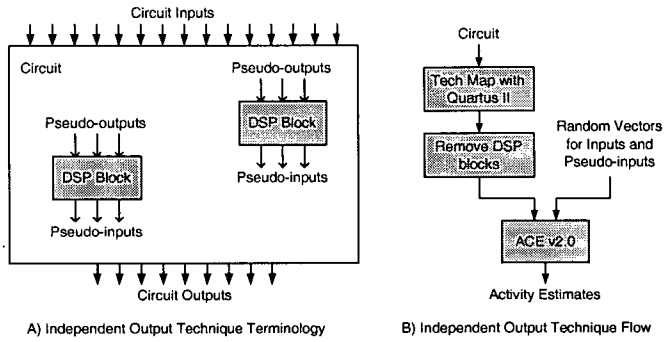

Figure 4.2: Independent Output Technique

The disadvantage of this technique is that it requires a gate-level implementation of the DSP block and proprietary technology library information. Since we expect that this technique would be used for evaluating a large number of architectures, a non-trivial amount of effort would be involved in generating gate-level implementations for each DSP block to be evaluated.

4.2.2 Independent Output Technique

The second technique that was evaluated addresses the disadvantage of the previous technique. If sufficiently accurate, we would prefer to use a technique that does not require proprietary technology or implementation details.

In this technique we propose to model the embedded blocks as if they are external to the circuit. The remainder of the circuit is composed of LUTs and interconnect, so the Poon model can be applied to that part.

To model the DSP blocks as external to the circuit, the inputs to the embedded block are treated as primary outputs of the circuit and the outputs of the embedded block are

treated as primary inputs to the circuit. We refer to the DSP block inputs and outputs as pseudo-outputs and pseudo-inputs of the circuit, respectively, as shown in Figure 4.2(A).

In this technique, the pseudo-inputs are assigned values in the same way that the primary inputs are assigned values by the Poon model. Random input vectors with a specified average activity are applied to the inputs and pseudo-inputs. The activities are then propagated through the circuit using ACE-2.0. When the inputs to a DSP block, pseudo-outputs, are encountered, the activities are not propagated through the DSP block. Instead, the pseudo-outputs are treated in the same way as the primary outputs. The activity calculations for the nodes downstream from the DSP block proceed using the pseudo-input values.

In effect, the estimated output activities of a DSP block are then independent of the input activities to the DSP block. This technique will be referred to as the *Independent Output Technique.*

Figure 4.2(B) shows a flow that employs this technique. The parent circuit is technology mapped using Quartus II. The DSP blocks are removed from the netlist and the nodes that were formerly outputs of the DSP block are represented as inputs to the circuit. Input vectors are then applied to the inputs and pseudo-inputs and activity estimation is performed using ACE-2.0.

The advantage of this technique is that it does not require gate-level implementation and technology information.

The disadvantage of this technique is that inaccuracies are being introduced because, in general, DSP block output transitions are not independent of their input transitions.

Reference [49] found that for word-parallel or bit-serial arithmetic, the average activity at the output of an adder can be closely approximated by the maximum of the average activities of the two inputs, implying that there is a dependence.

4.3 Methodology and Results for Activity Estimation

In comparing the accuracy of the two techniques, the *Gate Level Technique* will be used as the baseline. To determine how well the simpler *Independent Output Technique* correlates with the more accurate *Gate-Level Technique*, two quantities will be compared:

1. The activities at the outputs

2. The activities at the downstream nodes

In order to accurately estimate the power dissipated by the nets driven by the DSP block, accurate activities for these nets are needed. However, if the activities of all the DSP output pins are similar, then using a single average value could be sufficient.

When we refer to the downstream nodes of a DSP block, we mean the nodes between the DSP block outputs and the circuit outputs. Inaccurate activity estimates at the DSP block outputs may lead to inaccurate activity estimates for the downstream nodes due to the iterative nature of activity estimation algorithms (the output activity of each node is estimated based on the activities of the node inputs). Since there are typically many more downstream nodes than there are DSP output pins, inaccuracies in these downstream

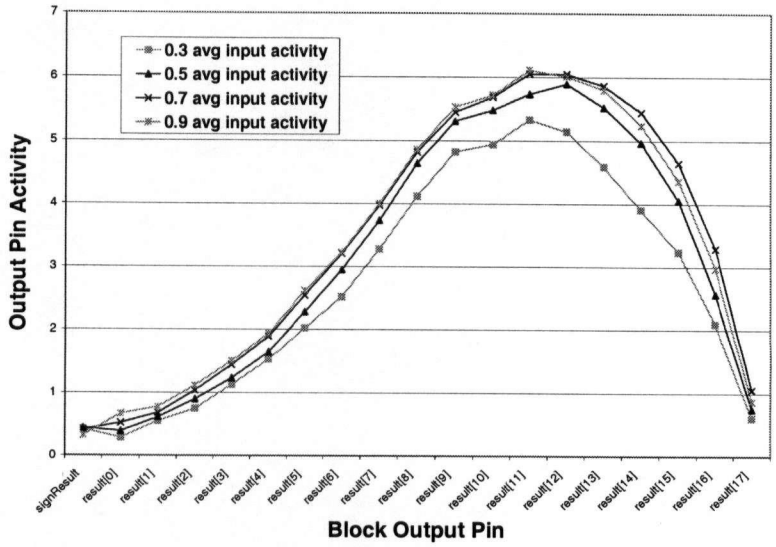

Figure 4.3: Output Pin Activities for Unregistered Multiplier

nodes may have a larger impact on the accuracy of the estimation than would inaccuracies in the DSP outputs.

In this section, both of these quantities were measured to determine whether the *Independent Output Technique* provides sufficient accuracy.

4.3.1 Output Nodes of DSP Block

In this subsection, the activities at the outputs are compared. To begin the investigation, the activity of each output pin of a 9-bit x 9-bit multiplier was examined. Random inputs (with a known average activity) were applied to the inputs of the multiplier, and ACE-2.0 was used to estimate the output activity of each pin. The results are plotted in Figure 4.3. The horizontal axis spans the set of 19 multiplier outputs (sign, LSB to MSB) and the vertical axis is the estimated activity of each of these outputs. Each line corresponds

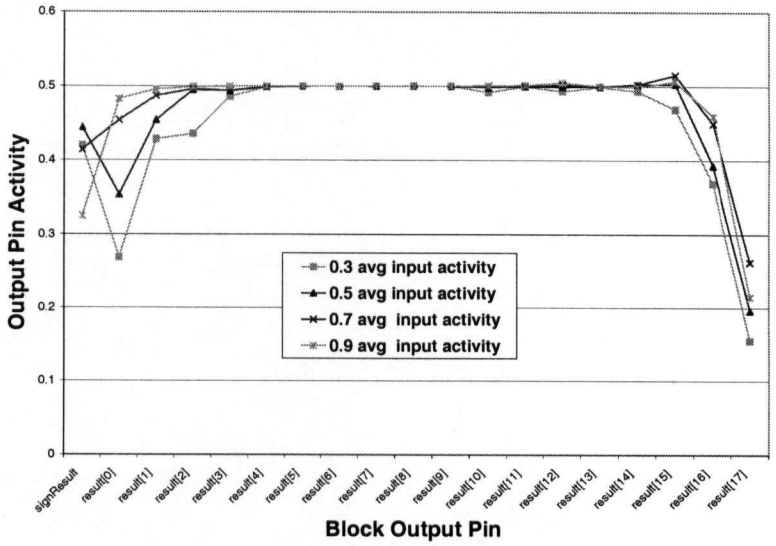

Figure 4.4: Output Pin Activities for Registered Multiplier

to a different average input activity. For all values of the average input activity, the conclusion is the same: the estimated activity differs across the output pins. The pins that are on the far left and right of the graph (the least and most-significant bits) have low activities, while the activities of the middle bits are large. This implies that there is a specific distribution for the activities of the output pins, and that choosing these activities randomly (as is done in the *Independent Output Technique*) will lead to inaccurate activity estimates for these nodes.

Note that the activities reported in Figure 4.3 are large, mostly greater than one. This is because multipliers tend to produce a large number of glitches on their outputs [50]. Most DSP blocks, however, contain registers on their output pins, which will remove these glitches. Figure 4.4 shows the results of the same experiment in which registers are added at the output of each multiplier. As the graph shows, the distribution is still there, especially for the extreme least and most significant bits.

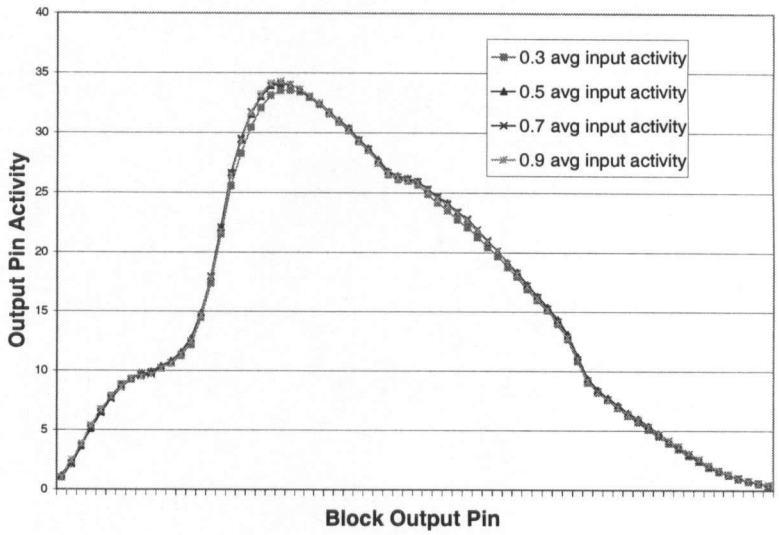

Figure 4.5: Output Pin Activities for Unregistered DSP Block

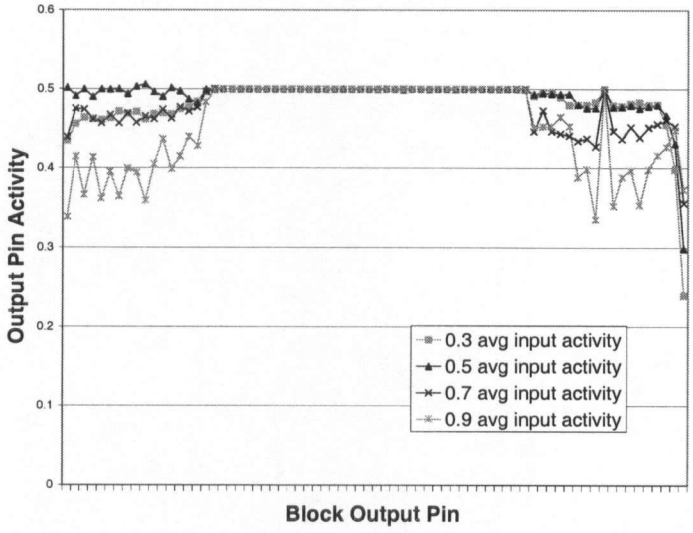

Figure 4.6: Output Pin Activities for Registered DSP Block

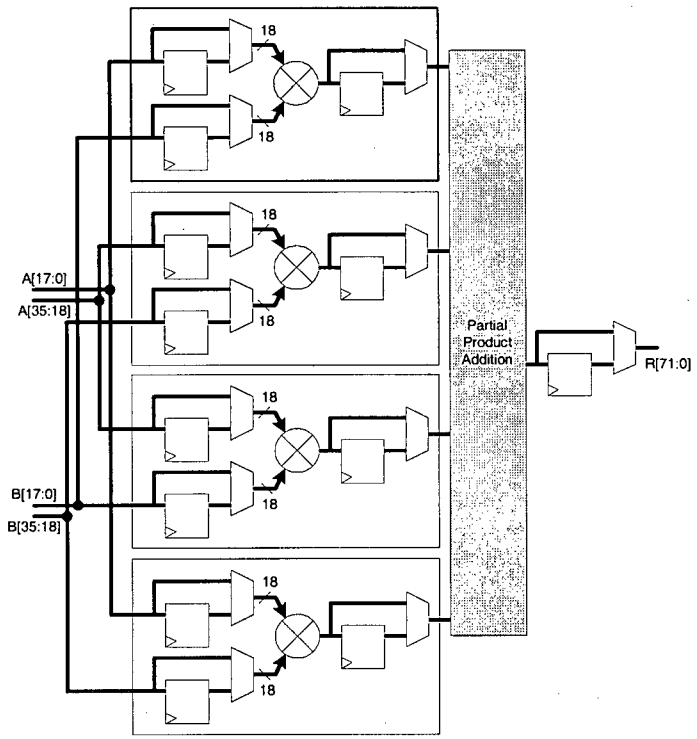

Figure 4.7: 18-bit x 18-bit Multipliers Combined for 36-bit x 36-bit Multiplier

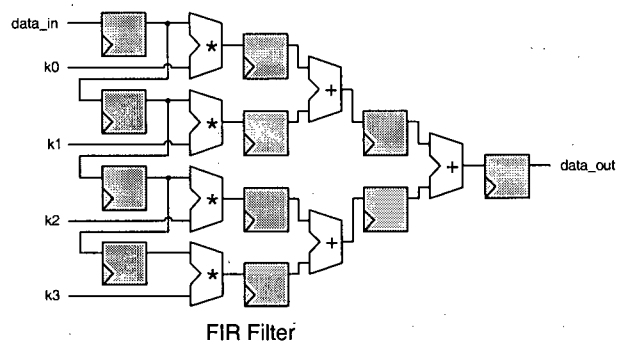

Figure 4.8: FIR Filter Used for Activity Experiments

Figures 4.5 and 4.6 show the results of the same experiment on a more complicated DSP block. The block is shown in Figure 4.7. It is similar to an Altera DSP block configuration: it combines four 18-bit x 18-bit multipliers and an adder to give a 36-bit x 36-bit multiplier. Again the conclusion is the same: a single average value to represent the activities will not capture the distribution of activities at the output pins.

4.3.2 Downstream Nodes of DSP Block

The results in Section 4.3.1 were for the DSP output pins only. In this section, we consider the nodes that lie downstream from the DSP blocks. Because ACE-2.0 propagates activities from inputs to outputs, inaccuracies in the DSP output activities will lead to inaccuracies in these downstream activities. The purpose of the remainder of this section is to understand how inaccurate these activities will be.

To perform these experiments, the FIR filter shown in Figure 4.8 was used. This circuit contains a bank of four multipliers followed by an adder tree; registers are included after the multipliers and within the adder tree to support pipelining, and to reduce glitch

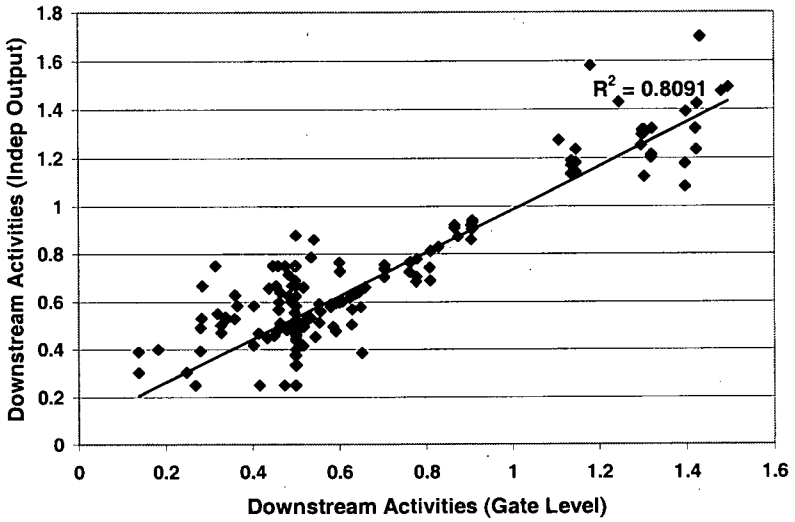

Figure 4.9: Activity Results for the FIR Filter

power. The activities of all nodes in the circuit were estimated using two methods: the *Independent Output Technique* flow shown in Figure 4.2(B) and the *Gate-Level Technique* flow shown in Figure 4.1(B). Figure 4.9 shows the results. In this graph, each dot corresponds to a node in the circuit; only nodes that are "downstream" (to the right of) the multipliers are included, starting with the outputs of the multiplier output registers. The x-coordinate of a dot is the activity predicted for the corresponding node by the *Gate-Level* (more accurate) *Technique*, while the y-coordinate of the dot is the activity predicted for the corresponding node by the *Independent Output* (less accurate) *Technique*.

In this plot, a straight line at y=x would indicate that there is perfect correlation between the two estimation techniques. As the graph shows, the correlation is good; the R^2 correlation metric is 0.8091. This is surprising, since the activities of the multiplier outputs are as shown in Figure 4.4 for the *Gate-Level Technique*, but random for the *Independent Output Technique*. The reason has to do with the nature of the downstream

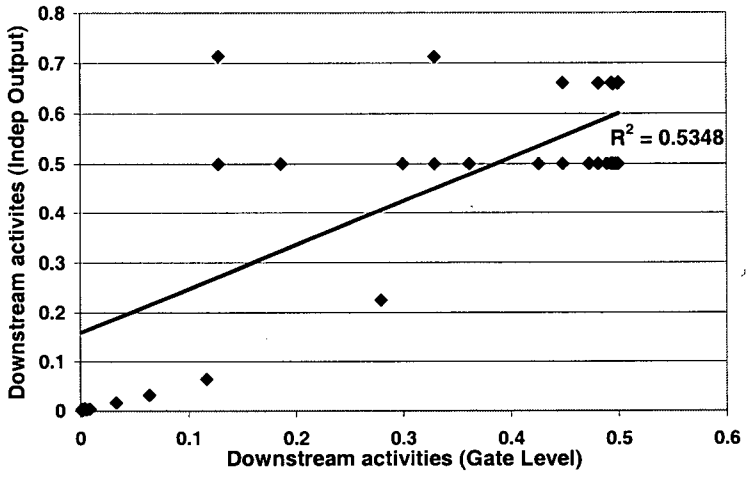

Figure 4.10: Activity Results for the Converter

circuit. As mentioned in Section 4.2.2, [49] found that for word-parallel or bit-serial arithmetic, the average activity at the output of an adder can be closely approximated by the maximum of the average activities of the two inputs. Other adder configurations were considered with similar results. The good correlation values do not hold for other downstream circuits, however. The adder tree in Figure 4.8 was replaced with a signed-magnitude to 2s complement converter, and found that R^2=0.542, as shown in Figure 4.10, which is not nearly as good.

4.4 Conclusions for Activity Estimation

Based on the results from the previous subsection, it was concluded that the activities obtained using the Independent Output technique do not correlate well with those obtained using the more accurate Gate-Level technique for all downstream circuits. Since

the *Power Estimation Tool Flow* must work for a wide variety of DSP architectures and downstream circuits, it was concluded that the Gate-Level technique is required for the flow.

Note that the results in Figures 4.4 and 4.6 suggest a third activity estimation technique (instead of the *Independent Output* and the *Gate-Level* techniques). Rather than generate the multiplier output activities randomly (as in the *Independent Output Technique*), it may be possible to construct a distribution function, and generate activities based on this distribution function. While this would be possible, it would require a significant amount of characterization effort each time a new embedded DSP block is to be evaluated, since the distribution function can be significantly different for different DSP block architectures. Given that ACE-2.0 is fast and accurate [48], and the *Gate-Level Technique* is easier, the extra characterization effort for this third method is not warranted.

4.5 Chapter Summary

The motivation for developing a new activity estimation technique is runtime. It is not feasible to propagate activities through a DSP block using the Transition Density or Lag-one model since the computation performed using these models (and other related models) is $O(2^k)$, where k is the number of inputs to the block. The Gate-Level Technique was introduced as an extension of the Poon model activity estimation method. The Independent Output Technique was introduced because we would prefer to use a technique that does not require proprietary technology or implementation details. Our experiments

showed that the more accurate Gate-Level Technique is required for our *Power Estimation Tool Flow.*

Chapter 5

Power Estimation

Once the activity estimates for each input and output pin of the DSP block have been obtained, the *Power Estimation Tool Flow* must estimate the power dissipated within each DSP block. This chapter describes and evaluates techniques for doing characterization-based power estimates.

5.1 Techniques for Power Estimation

5.1.1 Objectives

Since one of the objectives for this flow is to allow architectural exploration and experimentation, power estimation must be fast. This will facilitate many iterations of the flow in architectural parameter sweeps. Although it would be possible to create a gate-level model and use gate-level power simulation (such as with PrimePower), this would be far too slow to include in the inner loop of the *Power Estimation Tool Flow.* Therefore, a method is needed to quickly estimate the power of the embedded block, without resorting to modeling every internal node in the block.

Reduced estimation time typically comes at the cost of accuracy. In this chapter, we compare the accuracy of three fast and relatively simple techniques for estimating

the power of an embedded DSP block against simulation results. For each technique, offline characterization is used to obtain data that can be quickly referenced at runtime. A limited amount of data is found once for each DSP block architecture, offline, using PrimePower.

The following sections describe the three techniques we considered in increasing order of modeling effort.

5.1.2 The Constant Technique

The first technique is the simplest. For this technique, the power dissipated by a DSP block is assumed to be a constant, dependent on the DSP block type and independent of the activities of the input and output pins. This technique will be referred to as the *Constant Technique.*

The advantages of this technique are that it is simple and fast. The disadvantage of this technique is that it may lead to inaccurate estimates, since the power dissipated in an embedded block does depend on the input pin activities. However, this technique could be sufficient if the dependence is weak and the deviation from the average power is small.

5.1.3 The Lookup Technique

For the second technique, we approximate the power dissipated by the embedded block as a function of the average activity of all the DSP block inputs. The function need-not be linear, and may be implemented as a look-up table rather than as a closed-form function. This technique will be referred to as the *Lookup Technique.*

The advantage of this technique is that it is still relatively simple and fast, though it does involve more modeling effort than the Constant Technique. The disadvantage of this technique is that it assumes all input pins contribute equally to the power dissipated by the block. If we consider a multiplier block, for example, it will have pins corresponding to the multiplicand and multiplier operands. The fanout logic from the multiplicand operand pins may be very different from the fanout logic from the multiplier operand pins; thus, we expect that their contribution to the power dissipation will differ. However, if the difference does not cause substantial variation in the total power dissipation from an average over many trials, then this technique could be sufficient.

5.1.4 The PinCap Technique

The *Lookup Technique* may provide inaccurate power estimates, since only the average input activity is used to estimate the power. As mentioned in the previous section, in reality, not all inputs pins are equal; activity on some pins may have more impact on the power dissipation of a block than the same activity on other input pins. To take this into account, a third technique is considered, called the *PinCap Technique*. For this technique we estimate, for each input pin in isolation, how much of an impact that pin has on the overall power dissipation of the embedded block. This is quantified by calculating an effective capacitance, C_i, for each input pin i. Then, the total power can be calculated as:

$$P = \sum_{all\_input\_pins} \alpha_i C_i V_{dd}^2 f/2 \tag{5.1}$$

where f is the frequency of the circuit and α_i is the activity of pin i. Intuitively, this

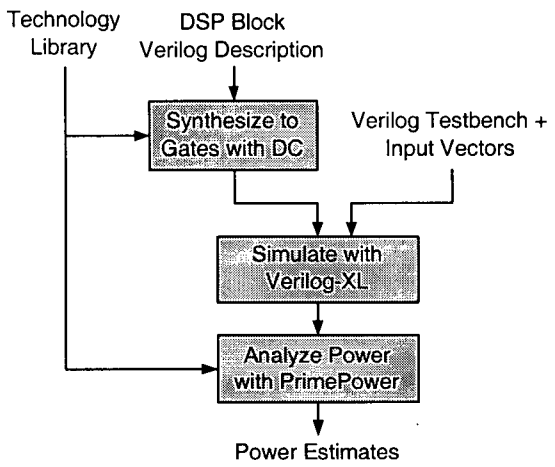

Figure 5.1: Methodology for Power Estimation Experiments

technique might provide accurate results, at the expense of more characterization effort.

5.2 Methodology for Power Characterization

Each of the three techniques requires some amount of offline characterization. This section describes the characterization methodology for each technique. Figure 5.1 shows the flow used to obtain this characterization data. A Verilog description of the DSP block was synthesized to gates using Synopsys Design Compiler. A Verilog testbench was used to simulate a set of input vectors applied to the gate-level description of the DSP block in Verilog-XL. The simulation data was then fed to the Synopsys PrimePower simulator to obtain characterization information. This is done for a training set of input vectors, according to the characterization technique used.

5.2.1 Constant Technique

For the *Constant Technique*, we repeated this characterization task nine times for the given DSP block for average input activity values in the set {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. For each of the 9 average activity values, a training set of 10 x 5000 input vectors was simulated to get 10 power estimates. We took the average of all 90 results to obtain a single value to use as the constant power value for the DSP block.

It should be noted that there is a one-to-many relationship between the average activity for a set of input vectors and power estimates for that average input activity. Different sets of input vectors may differ in the individual activities of each input pin and, as described in Section 5.1.4, different pins may contribute to the power of the block differently.

5.2.2 Lookup Technique

For the *Lookup Technique*, we reused the data from the Constant Technique characterization. We averaged the 10 estimates for each of the 9 activity values to obtain 9 data points. These 9 average input activities and their corresponding average power estimates became the activity-power estimate pairs for the *Lookup Technique* look-up table. This table was then included in the power model.

5.2.3 PinCap Technique

The input vector sets used for characterization for the *PinCap Technique* were different from those used for the *Constant* and *Lookup Technique* characterization. In those vector sets, all bits would toggle. For this technique, to assess the contribution of individual

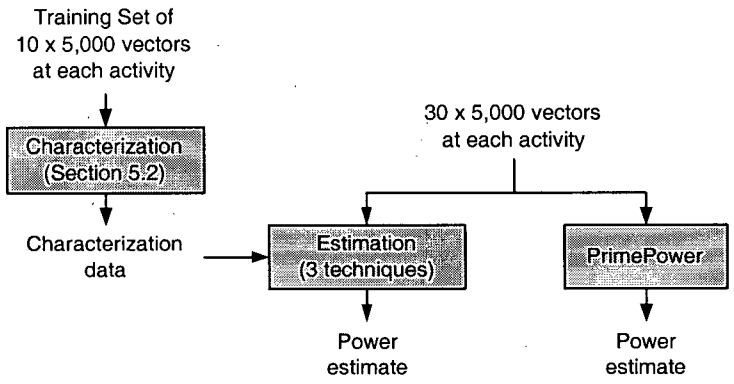

Figure 5.2: Power Estimation Experimental Methodology

pins, all other pins except the one in question were held constant; the pin in question was then toggled with the given activity. The resulting power from PrimePower and Equation 5.1 was then used to determine the effective capacitance for pin j, C_j, because $\alpha_i = 0$ for all $i \neq j$. This was repeated for each input pin to the DSP block. Once the effective capacitance for each input pin was determined, the values were included in the power model.

Input sets with average activities of 0.2 and 0.5 were used for the characterization. Conclusions about the PinCap technique could be drawn from the results for these two average input activities, so PinCap charaterization for the remaining activities between 0.1 and 0.9 was not necessary.

5.3 Evaluation

In this section, we evaluate the accuracy of each of the three power estimation techniques. The experimental methodology is shown in Figure 5.2. For each technique, we performed

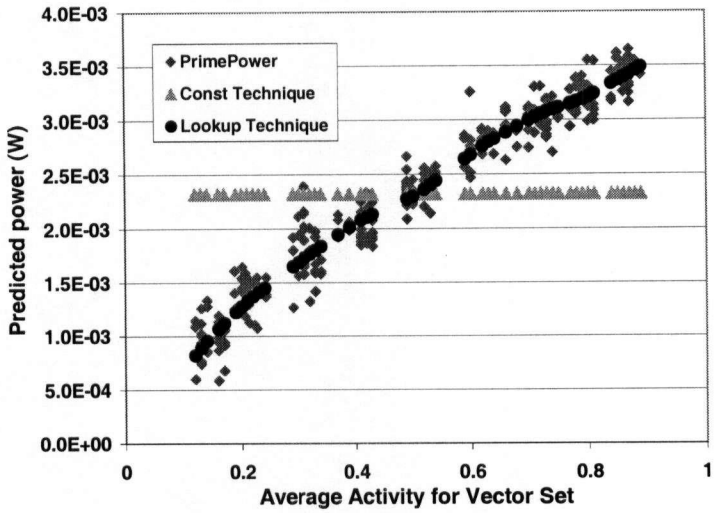

Figure 5.3: Power Estimation Experiment Results

the characterization tasks described in Section 5.2 for a 9x9-bit multiplier. We then compared the power estimated by each technique to that estimated by PrimePower simulation (which is presumably more accurate than any of our three techniques). For this experiment, we used 30 x 5000 randomly generated input vectors at each of the average input activity values in the set {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9} (this is more than was used during the offline characterization).

Figure 5.3 shows the results for the Constant Technique and the Lookup Technique. Each point on the graph represents the results for one set of 5000 vectors. The x-coordinate of the point is the average activity of all input vectors within the set, and the y-coordinate of the point is the estimated power. The triangular dots represent the estimates using the Constant Technique, while the circular dots represent the predictions from the Lookup Technique. For comparison, the PrimePower estimates are also plotted on the same graph; the PrimePower estimates are shown as diamonds. As the graph shows, the Constant

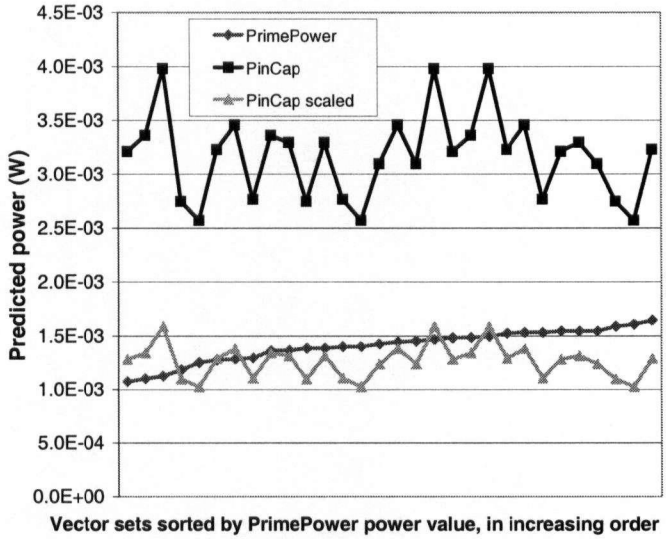

Figure 5.4: Power Estimation Experiment Results

Technique (in which a single power value is used, independent of the input activity) provides poor estimates, compared to the PrimePower results. The Lookup Technique, on the other hand, produces results that track well with the PrimePower results. The average difference between the Lookup Technique and PrimePower estimates was 8%. The experiment was repeated with a DSP block rather than a multiplier, and the conclusions were the same; the average difference between the Lookup Technique and the PrimePower results was 5%. This suggests that it is important to take the input activities into account when estimating power of the block, but that the average of the input activities is enough information to get reasonably accurate power estimates.

For the PinCap Technique, the results were not as good. Figure 5.4 shows the estimates obtained using the PinCap Technique, along with the estimates obtained using PrimePower. In this graph, only vector sets with an average input activity of 0.2 were considered. Each point in this graph corresponds to one such set. The points are dis-

tributed evenly across the x-axis, sorted by the values of the PrimePower estimate. As the graph shows, even for these sets, which all correspond to the same average activity, the power estimates predicted by the PinCap technique vary widely, and have no correlation to the PrimePower estimates.

The *PinCap Technique* always overestimates the actual power; this is because the technique assumes that the power contribution of each input pin is independent of the power contribution of each other input pin, when in fact, it is not. The total power is not simply the sum of the contributions from each pin. Even if the PinCap results are scaled by a constant value in an attempt to reflect this, the results do not track the simulation results well (this is also shown in Figure 5.4). Other input activities were also attempted and no simple way was found to scale the results to take into account overlap between the contributions of multiple input pins.

5.4 Conclusions for Power Estimation

Based on the results from the previous section, we concluded that the Lookup Technique is most appropriate for the *Power Estimation Tool Flow.* Not only does it provide reasonably accurate results, but the characterization effort is relatively simple, and the run-time of the power estimate is small.

5.5 Chapter Summary

The objective of the power estimation experiments was to determine a fast and accurate technique for estimating the power of DSP blocks, given input activities. We introduced three techniques, in increasing order of complexity: (1) the Constant Technique, (2) the Lookup Technique, and (3) the PinCap Technique. Power estimates using these techniques were performed and compared against PrimePower simulations for accuracy. We found that it is important to consider the input activities when estimating the power and that the Lookup Technique is most suitable for our *Power Estimation Tool Flow*.

Chapter 6

Power Estimation Tool Flow

This chapter describes how we combined our Gate-Level activity estimation technique from Chapter 4 and our Lookup power estimation technique from Chapter 5 with the PVPR framework to obtain a complete power estimation CAD tool flow. Section 6.1 describes the overall flow and the modifications we made to PVPR. Section 6.2 describes the processing of two benchmark circuits through the entire flow and compares our estimates against PrimePower simulations of the circuits.

6.1 Overall Flow

6.1.1 Functionality

Our CAD tool flow for estimating the power dissipated in FPGA circuits containing embedded DSP blocks is shown in Figure 6.1. The activity estimation and power estimation are performed as described in Chapters 4 and 5.

The steps in the box on the left correspond to activity estimation. For activity estimation, there are three inputs: (1) the user's circuit, (2) a Verilog description of the DSP block, and (3) input statistics (either as vectors for the circuit inputs or the transition density and static probability of each input). To facilitate architectural investigations,

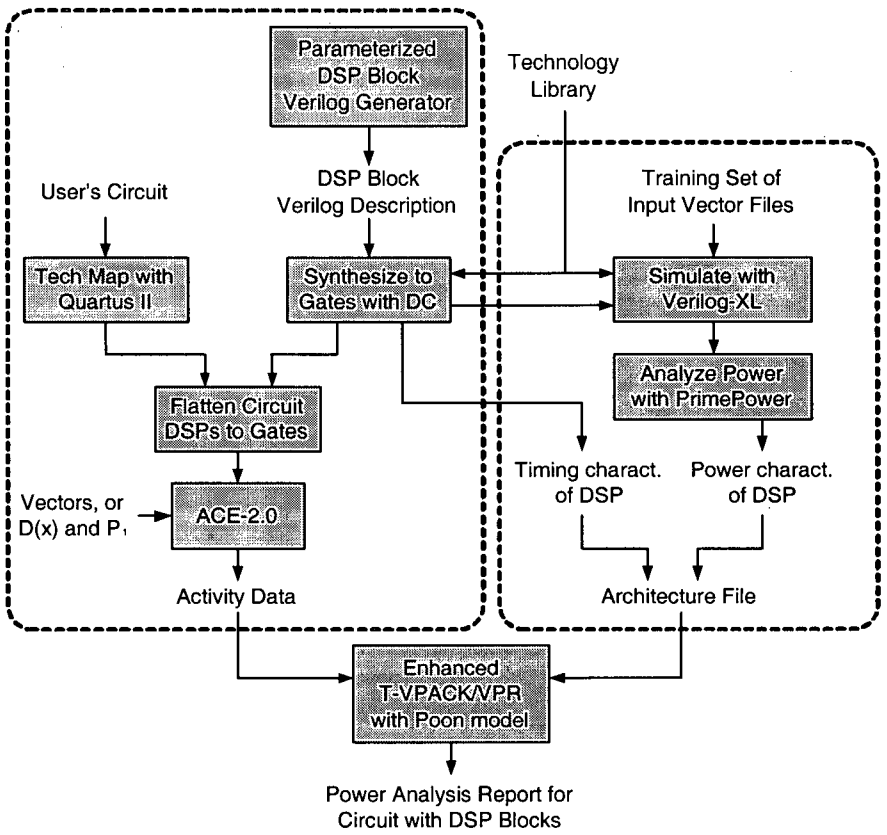

Figure 6.1: Power Estimation Tool Flow

we created a parameterized DSP block generator to automate the production of Verilog descriptions for a large number of DSP architectures. The DSP block description is synthesized to a gate-level netlist using Synopsys Design Compiler and a TSMC 0.18 μm technology library. The HDL description of the user's circuit is technology mapped to LUTs, flip-flops and DSP blocks using Quartus II to produce a mapped netlist of these elements and their connections. The mapped netlist is flattened by replacing the DSP block instances with the gate-level implementation obtained from Design Compiler. ACE-2.0 is then used to obtain the activity of each node in the flattened circuit, using the input

statistics that we provided.

The steps in the box on the right correspond to power characterization. Recall that, whereas the activity estimation steps must be repeated each time the user's circuit changes, the power characterization steps need only be done once when the DSP block is designed and can then be stored as library data. For power characterization, there are two inputs: (1) the gate-level implementation of the DSP block from Design Compiler, and (2) a training set of input vector files. The gate-level implementation is simulated using Verilog-XL and the resulting power determined by PrimePower for the training set of input vector files. The average activity and resulting power for each input vector file in the training set is saved as an activity-power pair in the Lookup Technique table in the PVPR architecture file. The DSP block timing characteristics are also stored in the PVPR architecture file.

The activity data for the circuit and the characterization data for the DSP block are then input to PVPR for packing, placement, routing, and power analysis. Currently, the use of Quartus II for technology mapping restricts us to Altera-style DSP blocks; however, Quartus II could be replaced with another technology mapping tool to evaluate different DSP block architectures.

6.1.2 Modifications to PVPR

Neither the original version of VPR nor PVPR support circuits with embedded DSP blocks. We enhanced the BLIF netlist format [42] to allow for the specification of DSP blocks. We modified the PVPR architecture file format [21] to include power and timing

numbers for these blocks (including the power look-up table proposed in Chapter 5); Table 6.1 describes the parameters we added. We assumed that DSP blocks are arranged in columns, as in Altera and Xilinx devices. We modified the placement algorithm to correctly position these blocks, modified the timing analysis algorithm to estimate the delay through these blocks, and use this information to calculate the critical path of an implementation [51]. We then modified the power model to use the *Lookup Technique* for DSP blocks.

Table 6.1: Parameters added to the PVPR Architecture File Format

Parameter	Meaning
Start	First column of DSP blocks
Repetition	Number of columns before next column of DSP blocks
Class	Nature of the DSP input and output pins
Location	Locations around the DSP block where the input and output pins can be programmably connected to the routing fabric
Leakage	Leakage power dissipated by the DSP block
Activity	Look-up value for activity in an activity-power pair
Energy	Energy dissipated for a given Activity in an activity-power pair (Energy is used to be independent of clock frequency)

6.2 Flow Demonstration and Comparison to Gate-Level Simulation

6.2.1 Motivation

The experiments in Chapter 5 consider the DSP blocks as stand-alone elements; random inputs are used and all bits have approximately the same average activity. When a DSP

block exists in a larger circuit, however, it may be located downstream from other logic; each bit in the input operands may have a different activity. This section demonstrates the functionality of our CAD tool flow using two benchmark circuits from [52] and compares our results with a more accurate method (using the same inputs) to see how much accuracy is sacrificed to obtain fast estimates.

6.2.2 Terminology

As this section discusses the simulation of multiple circuits, it is important to clarify the terminology we will use. The test circuits (the FIR filter and differential equation solver) will be refered to as the *parent circuit* or *circuit*. The circuit will contain multiple instances of a *DSP block*. The *DSP block* is described by the DSP Block Verilog Description in Figure 6.1. The instances will be referred to as *DSP block instances*. Figure 6.2 shows the *circuit* (in white) with *instances* (in grey) of a *DSP block*, which is shown to the right.

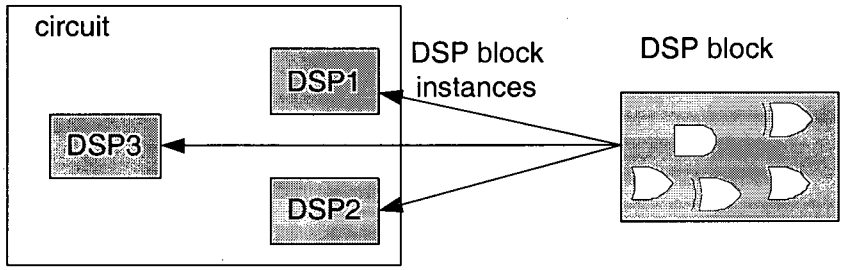

Figure 6.2: Terminology

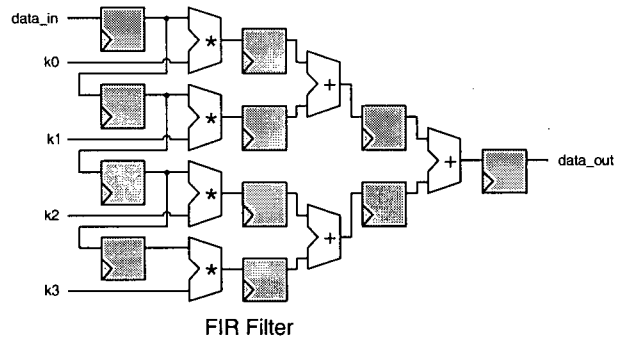

Figure 6.3: fir_3_8_8: FIR Filter Circuit

6.2.3 Methodology

Demonstration of Power Estimation Tool Flow

To demonstrate the functionality of our flow, we ran it on two benchmark circuits from [52]. The first circuit is fir_3_8_8, the FIR filter shown in Figure 6.3, which uses embedded multipliers; it consists of 272 LUTs, 148 flip-flops, and 4 multipliers. The second circuit is diffeq_paj_convert, a differential equation solver, shown in Figure 6.4; it consists of 850 LUTs, 193 flip-flops, and 3 DSP blocks. The differential equation solver uses a DSP block configuration similar to one found in Altera Stratix devices. This block is shown in Figure 6.5; it combines four 18-bit multipliers with a dedicated adder to make a 36-bit multiplier. For both example DSP blocks, a look-up table of activity-power pairs had been created offline, using Lookup Technique characterization as described in Chapter 5.

For each circuit we performed the characterization part of the flow only once and ran an iteration of the rest of the flow for each input vector file. For each activity in the set {0.25, 0.50, 0.75}, we created five input files of 5000 vectors having that average activity, for a total of 15 input files. Note that the width of these vectors is equal to the number

of inputs of the parent circuit. The purpose of providing vectors instead of activity and static probability values for each bit is that ACE-2.0 performs simulation for sequential feedback loops. If only activity and static probability values are given to ACE-2.0, then it randomly generates vectors for simulation. However, since we wanted to compare our results against a PrimePower simulation, we needed to provide the same set of vector files to both estimation tools.

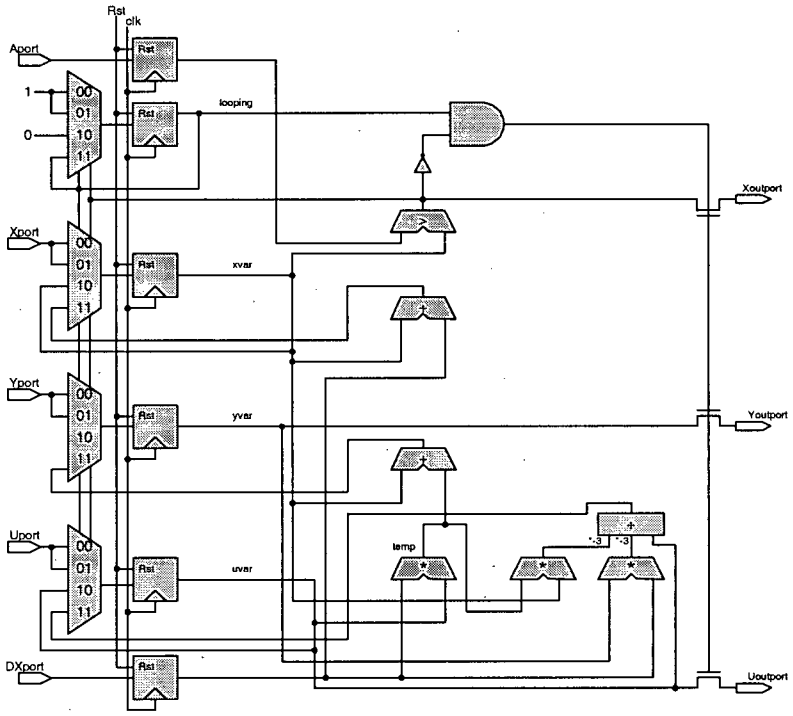

Figure 6.4: Differential Equation Solver Circuit

Comparison against PrimePower

For these experiments we compared the results from our *Power Estimation Tool Flow* against PrimePower. The pseudocode for the methodology is given in Figure 6.6 and

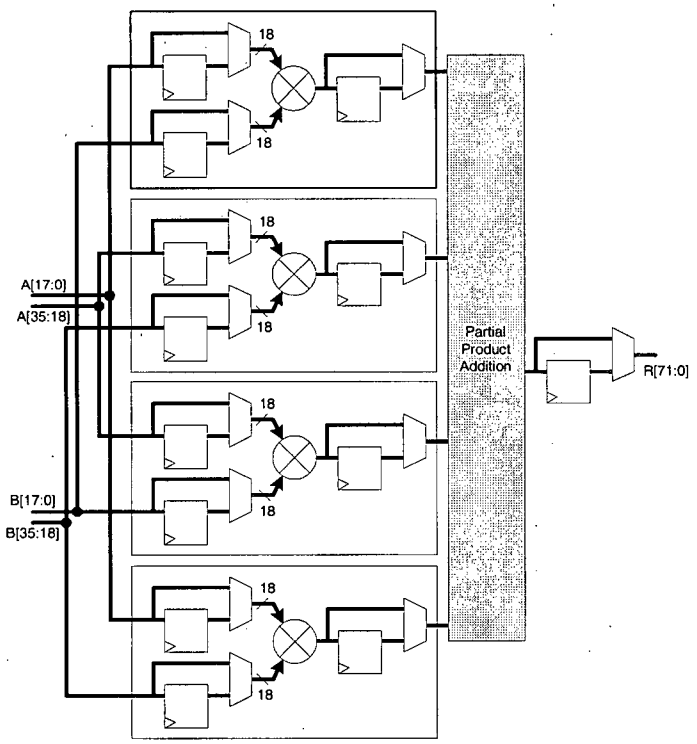

Figure 6.5: 18-bit x 18-bit multipliers combined for 36-bit x 36-bit multiplier

described below.

The same input vectors were used for both the *Power Estimation Tool Flow* and the simulation approaches. We assumed that the primary inputs to the parent circuit were free of glitching.

To determine the power dissipated by each *DSP block instance* using simulation, we used the flow shown in Figure 6.7. To determine the power of each DSP block instance, we had to determine the input waveforms to each DSP block instance separately. It is incorrect to assume that the DSP block instances will have identical input waveforms, because the logic upstream of (leading up to) each DSP block instance in the parent circuit may not be identical.

```
Synthesize Verilog description of DSP block to gates in Design Compiler
  (already done for Power Estimation Tool Flow);

Create simulation model of circuit in Quartus II;

For avg_act in (0.25, 0.50, 0.75) {

   For trial in (1 to 5) {
      Create input vector set V of 5000 vectors with activity avg_act
        and width equal to number of primary inputs in circuit;
      Apply input vector set to circuit sim model using testbench_cct;
      Use ModelSim to simulate and generate VCD;

      For each DSP block instance i in circuit {
         Parse VCD for input waveforms to this DSP instance;
         Apply waveforms at DSP instance inputs using testbench_dsp_i;
         Use Verilog-XL to simulate;
         Use PrimePower to calculate power of DSP instance i (when
           circuit is stimulated by input vector set V);
      }
   }
}
```

Figure 6.6: Simulation Flow Pseudocode

To obtain the input waveforms to each DSP block instance, we used Quartus II to generate a simulation model of the parent circuit. Since we used Quartus II to technology map the circuits in the *Power Estimation Tool Flow*, the mapped implementations for both methods match. We used ModelSim to simulate the circuit and generate a Value Change Dump (VCD) of the simulation, which is an ASCII file that describes the waveforms for the circuit internal nodes. We then used a VCD parser to extract the waveforms corresponding to the inputs of each DSP block instance. To obtain the simulation power estimates for each vector set, we used the input waveforms for each DSP block instance to simulate the

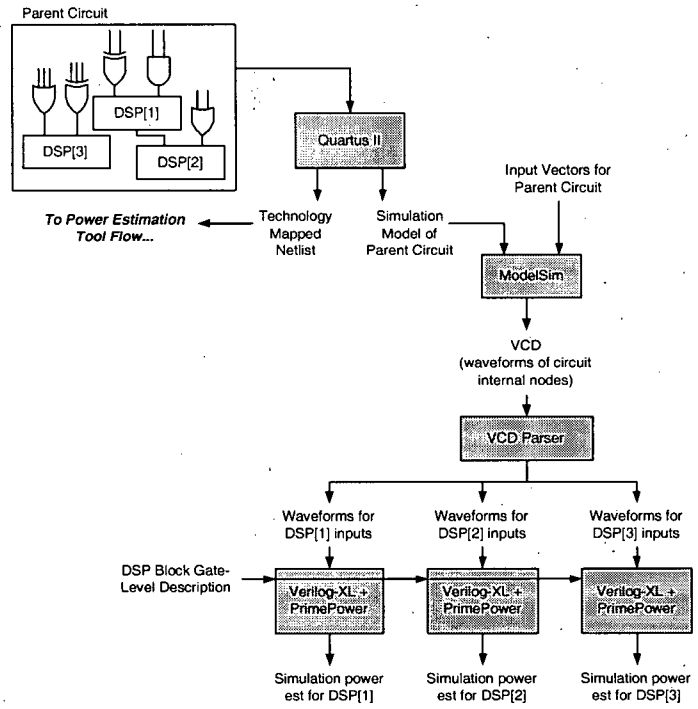

Figure 6.7: Flow for Determining Simulation Power Estimate of Each DSP Block Instance

gate-level implementation of the *DSP block* using Verilog-XL and PrimePower.

6.2.4 Results

Demonstration of Power Estimation Tool Flow

The results of the power analysis are shown in Table 6.2. In both circuits, the DSP blocks dominate the power dissipation. This may be surprising because routing power generally dominates the power dissipation in an FPGA. However, the circuits are DSP kernels and not complete systems. Thus, they contain only a small amount of non-DSP logic and substantial routing is internal to the DSP blocks.

Comparison against PrimePower

Tables 6.3 and B.1 to B.3 show the detailed results for the FIR filter circuit. Tables 6.4 and B.4 to B.6 show the detailed results for the differential equation solver circuit.

The average difference between the Power Estimation Tool Flow and PrimePower results for the FIR circuit was 20.4%. In the differential equation circuit, 2 of the 3 DSP blocks had an average difference of 22-26%, however the third had glitching on one input bus and was off by 77% on average. When creating the lookup table, the blocks had been characterized only for activities 0.1 to 0.9, as it was not clear how to properly imitate glitching. Although we use linear interpolation to determine the power corresponding to average input activities that are not in the Lookup Technique table, the power relationship is not necessarily linear with respect to DSP input activities. Consequently, it is not surprising that estimates for DSP block instances having an average input activity greater than 1 are not well represented by a linear interpolation using the points for activities 0.8 and 0.9. This indicates the importance of a lookup-table that includes data that considers glitching.

Appendix A describes preliminary unsuccessful attempts at including glitching during PrimePower characterizations.

Table 6.2: VPR Power Analysis Results

Power	fir_3_8_8	diffeq_paj_convert
Routing	5.46 mW, 18.4%	8.67 mW, 10.0%
Logic Blocks	4.04 mW, 13.6%	4.68 mW, 5.6%
Clock	1.61 mW, 5.4%	2.57 mW, 2.9%
DSP	18.59 mW, 62.6%	70.82 mW, 81.5%
Total	29.71 mW	86.74 mW

Table 6.3: Overall Percentage Error for FIR Filter Results

	Arithmetic Mean
average percentage error	20.4%
average for just multiplier instance #0	21.6%
average for just multiplier instance #1	19.7%
average for just multiplier instance #2	19.6%
average for just multiplier instance #3	20.5%

Table 6.4: Overall Percentage Error for Differential Equation Solver Results

	Arithmetic Mean
average percentage error	41.7%
average for just DSP block instance #0	76.8%
average for just DSP block instance #1	22.5%
average for just DSP block instance #2	25.8%

6.3 Chapter Summary

In this chapter we have demonstrated the functionality of our *Power Estimation Tool Flow* for estimating the power of FPGA circuits containing embedded DSP blocks, which addresses our requirements laid out in Chapter 3. We have compared our results against simulation using PrimePower. Our fast estimates are within 19% to 26% of the simulated results, except in the case where there is significant glitching at the inputs of the DSP blocks; the glitching case resulted in 77% error. We believe that adding characterization of glitches on the inputs of the DSP blocks is necessary to improve the accuracy of our method; however, it is not immediately clear how this can be done. This will be discussed

as future work in Section 7.2.2.

Chapter 7

Conclusion

7.1 Summary and Contributions

In this thesis we have described an experimental CAD flow that can be used to estimate the power dissipation of FPGA circuits containing embedded DSP blocks. We identified two technical challenges in creating such a flow: (1) estimating the activity of all nodes in a circuit containing one or more DSP blocks, and (2) estimating the power dissipated within a DSP block quickly and accurately.

The first challenge arises because standard activity estimation techniques cannot propagate activities through these DSP blocks. We address this by replacing each DSP block with a gate-level representation of the block, and using the standard activity techniques on the resulting circuit.

The second challenge arises because it is not possible to pre-characterize the DSP block for all possible input patterns and activities. We have shown that reasonable estimates can be obtained by creating a look-up table of power values. In the power model, the look-up table is indexed using the average activity of the block input nodes.

We then combined our findings to create a *Power Estimation Tool Flow* based on the PVPR framework. The impact of our enhanced tool flow is threefold; the existence of

a freely available, architecturally flexible FPGA CAD tool that includes power modeling for embedded DSP blocks enables:

1. the investigation of power-aware architectures containing embedded DSP blocks
2. the investigation of power-aware CAD algorithms for FPGA circuits containing embedded DSP blocks
3. the incorporation of power tradeoffs in the design of user circuits

This work is also one of a collection of projects at the University of British Columbia System-on-Chip Lab that each take a step towards the larger goal of enabling power estimation for platform-style FPGA architectures that contain embedded DSP blocks, embedded memories, embedded processors, and multiple clock domains.

A poster of our contributions will appear at the 2006 IEEE International Conference on Field Programmable Technology in Bangkok, Thailand.

7.2 Future Work

Given our tool flow, there are three enhancements that would be necessary before performing power-aware FPGA architecture studies that include embedded DSP blocks: (1) a suite of integer benchmarks representative of DSP and arithmetic-intensive user designs, (2) the incorporation of glitch characterization into the look-up data, and (3) board-level verification.

7.2.1 Benchmarks

A suite of integer benchmarks representative of DSP and arithmetic-intensive user designs is essential to be able to draw generalizable conclusions from an architectural study targeting embedded DSP blocks. The standard benchmark suite for FPGA studies is the collection of Microelectronics Center of North Carolina (MCNC) circuits; however, these circuits are not very representative of DSP applications.

Freely available circuits were obtained from [52] and [53]; however, most were not suitable for our experiments for the reasons listed below:

- Some circuits used floating point arithmetic. DSP blocks in commercial FPGAs target integer arithmetic, so our flow does the same.
- Some circuits used very simple and small DSP blocks, which would not exercise many of the features in DSP blocks embedded in commercial FPGAs.
- Some of the circuits were automatically generated using high-level synthesis. The RTL signal and module names were automatically generated alphanumeric character sequences. This hid the flow of control and data in the circuits and prevented analysis and debugging of the circuits.

A very useful future research project would be the creation of circuits for integer DSP and arithmetically intensive applications at the register transfer level.

7.2.2 Glitch Characterization

The comparison to gate-level simulation in Section 6.2 revealed that glitching may take place on the inputs to the DSP blocks and that it is important to include characterization data in the power estimation look-up table for cases where the activity at the inputs to the DSP blocks is greater than 1. When performing characterization, it was not clear how to properly imitate glitching in our testbenches. Appendix A describes our attempts.

Reference [30] describes word-level and bit-level glitch generation and propagation models for characterization of datapath circuits. It would be interesting to incorporate this into our flow and evaluate its effectiveness.

7.2.3 Board-Level Verification

When performing power estimation at higher levels of abstraction, as we do with the Lookup Technique, we are trading off accuracy for fast estimation. Consequently, at this level, fidelity is what we seek to provide (i.e. relative accuracy, instead of absolute). In order to verify that our tool flow will provide consistent estimates that provide usable trends, we must compare our results to physical measurements on an FPGA board.

The use of board-level measurements to verify our power model is not trivial. It is not simply a case of downloading our test circuits to boards with FPGAs containing the appropriate DSP blocks and comparing the measured power to our estimates. First, it is not feasible to create custom FPGAs for the set of DSP block architectures under evaluation; layout and fabrication costs are excessive. Second, the power of the DSP blocks alone cannot be measured; typically, we can only measure the total active and

quiescent power of the system. Consequently, we cannot simply compare our estimates against the results for a set of commercial FPGA boards. For example, Altera Cyclone II and Xilinx Virtex-II devices contain embedded 18x18-bit multipliers, and Altera Stratix and Xilinx Virtex-4 devices contain DSP blocks. One possibility is to compare our power estimates for a set of benchmark circuits against the power estimates for four boards containing each of these devices; however, their logic and routing architectures differ, making it impossible to distinguish between deficiencies in the power models for the DSP blocks, the logic blocks, and the interconnect.

A starting point for board-level verification could be to compare trends in measured values on a particular FPGA board against trends in the power estimates using the corresponding architecture description file, for a set of benchmark circuits. The desired result would be to see that the measurements and estimates both rank the power dissipation of the benchmark circuits in the same order. A prerequisite for this verification is a representative set of benchmark circuits, described in Section 7.2.1.

Bibliography

[1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs.* Springer, March 1999.

[2] S. J. E. Wilton, , S. Chin, and K. K. W. Poon, *Field-Programmable Gate Array Architectures.* Taylor and Francis, 2006.

[3] A. Ye and J. Rose, "Using Multi-Bit Logic Blocks and Automated Packing to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits," in *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology*, 2004, pp. 129–136.

[4] C. Piguet, Ed., *Low Power CMOS Circuits: Technology, Logic Design and CAD Tools.* New York: Taylor and Francis, 2006.

[5] K. K. Poon, "Power Estimation for Field Programmable Gate Arrays," Master's thesis, University of British Columbia, August 2002.

[6] Actel, *Actel IGLOO Flash Freeze Technology and Low Power Modes Application Note*, 2006.

[7] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, "Dynamic Voltage Scaling for Commercial FPGAs," in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology*, 2005, pp. 173–180.

[8] F. Li, Y. Lin, and L. He, "FPGA Power Reduction Using Configurable Dual-Vdd," in *Proceedings of the 41st Design Automation Conference*, 2004, pp. 735–740.

[9] V. George, H. Zhang, and J. Rabaey, "The Design of a Low Energy FPGA," in *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, 1999, pp. 188–193.

[10] J. H. Anderson and F. N. Najm, "Power-aware Technology Mapping for LUT-based FPGAs," in *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology*, 2002, pp. 211–218.

[11] J. H. Anderson and F. N. Najm, "Active Leakage Power Optimization for FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 423–437, 2006.

[12] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware RAM Mapping for FPGA Embedded Memory Blocks," in *FPGA'06: Proceedings of the International Symposium on Field Programmable Gate Arrays.* New York, NY, USA: ACM Press, 2006, pp. 189–198.

[13] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in *FPGA'06: Proceedings of the International Symposium on Field Programmable Gate Arrays.* New York, NY, USA: ACM Press, 2006, pp. 21–30.

[14] Altera, *Cyclone II Device Handbook*, 2005.

[15] Altera, *Stratix II Device Handbook*, 2005.

[16] Xilinx, *Virtex II Platform FPGAs: Complete Datasheet*, 2005.

[17] *UG073: XtremeDSP for Virtex-4 User Guide*, 1st ed., 2005.

[18] S. Hauck, M. M. Hosler, and T. W. Fry, "High-performance Carry Chains for FPGAs," in *FPGA '98: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays.* New York, NY, USA: ACM Press, 1998, pp. 223–233.

[19] S. D. Haynes, A. B. Ferrari, and P. Y. K. Cheung, "Flexible Reconfigurable Multiplier Blocks Suitable for Enhancing the Architecture of FPGAs," in *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference*, 1999, pp. 191–194.

[20] M. J. Beauchamp, S. Hauck, K. D. Underwood, and S. K. Hemmert, "Embedded Floating-Point Units in FPGAs," in *Proceedings of the 2006 ACM/SIGDA Fourteenth International Symposium on Field Programmable Gate Arrays.*, 2006, pp. 12–20.

[21] V. Betz, *VPR and T-VPack User's Manual, ver 4.30*, March 2000.

[22] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A Detailed Power Model for Field-Programmable Gate Arrays," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2004.

[23] F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 446–455, 1994.

[24] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and Single-driver Wires in FPGA Interconnect," in *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology*, 2004, pp. 41–48.

[25] A. Yang, "Design Techniques to Reduce Power Consumption," *XCell Journal*, 2005.

[26] Xilinx, *Virtex-5 LX Platform Overview*, 2006.

[27] Xilinx, *UG070: Virtex-4 User Guide*, 2006.

[28] A. Ye and J. Rose, "Using Bus-based Connections to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits," in *Proceedings of the 2005 ACM/SIGDA Thirteenth International Symposium on Field Programmable Gate Arrays*, 2005, pp. 3–13.

[29] K. Leijten-Nowak and J. L. van Meerbergen, "An FPGA Architecture with Enhanced Datapath Functionality," in *FPGA '03: Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM Press, 2003, pp. 195–204.

[30] A. Raghunathan, N. K. Jha, and S. Dey, *High-Level Power Analysis and Optimization*. Springer, November 1997.

[31] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*. McGraw-Hill Science/Engineering/Math, August 2002.

[32] G. K. Yeap, *Practical Low Power Digital VLSI Design*. Springer, August 1997.

[33] T. Quarles, D. Pederson, R. Newton, A. Sangiovanni-Vincentelli, and C. Wayne, "Interactive SPICE User Guide," website maintained by Jan Rabaey. [Online]. Available: http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/

[34] P. E. Landman and J. M. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 2, pp. 173–187, 1995.

[35] D. Marculescu, R. Marculescu, and M. Pedram, "Information Theoretic Measures for Power Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 6, pp. 599–610, 1996.

[36] Xilinx, "Xilinx Web Power Tools," April 2006. [Online]. Available: http://www.xilinx.com/products/design_resources/power_central/

[37] Altera, *PowerPlay Early Power Estimator User Guide for Stratix, Stratix GX, and Cyclone FPGAs*, October 2005.

[38] Xilinx, "Virtex-4 XPower - Early Power Estimator v8.1," April 2006. [Online]. Available: http://www.xilinx.com/ise/power_tools/license_virtex4.htm

[39] Altera, *PowerPlay Early Power Estimator User Guide for Stratix II, Stratix II GX and Hardcopy II*, December 2005.

[40] Xilinx, *Xilinx Development System Reference Guide 8.1i*, December 2005.

[41] Altera, *PowerPlay Power Analyzer*, October 2005.

[42] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and S. A. Vincentelli, "SIS: A System for Sequential Circuit Synthesis," UC Berkeley, Tech. Rep., 1992. [Online]. Available: http://citeseer.ist.psu.edu/sentovich92sis.html

[43] J. Cong and Y. Ding, "FlowMap: an Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-table Based FPGA Designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[44] G. G. F. Lemieux, S. D. Brown, and D. Vranesic, "On Two-step Routing for FPGAS," in *ISPD '97: Proceedings of the 1997 International Symposium on Physical Design.* New York, NY, USA: ACM Press, 1997, pp. 60–66.

[45] Y.-W. Chang, D. F. Wong, and C. K. Wong, "Universal Switch Modules for FPGA Design," *ACM Transansactions on Design Automation of Electronic Systems*, vol. 1, no. 1, pp. 80–101, January 1996.

[46] S. J. E. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory," Ph.D. dissertation, University of Toronto, 1997.

[47] I. M. Masud and S. J. E. Wilton, "A New Switch Block for Segmented FPGAs," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications.* London, UK: Springer-Verlag, 1999, pp. 274–281.

[48] J. Lamoureux and S. J. E. Wilton, "Activity Estimation For Field-Programmable Gate Arrays," in *International Conference on Field-Programmable Logic and Applications*, August 2006.

[49] A. Chatterjee and R. K. Roy, "Synthesis of Low Power Linear DSP Circuits Using Activity Metrics," in *Proceedings of the Seventh International Conference on VLSI Design*, 1994, pp. 265–270.

[50] S. J. E. Wilton, S. S. Ang, and W. Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," in *International Conference on Field-Programmable Logic and its Applications.* Springer-Verlag, August 2004, pp. 719–728.

[51] S. J. E. Wilton, "HHVPR Manual," Internal document, University of British Columbia, Tech. Rep., July 2005.

[52] P. Jamieson and J. Rose, "A Verilog RTL Synthesis Tool for Heterogeneous FPGAs," in *International Conference on Field Programmable Logic and Applications, 2005*, 2005, pp. 305–310.

[53] C. H. Ho, P. H. W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo, "Virtual Embedded Blocks: A Methodology for Evaluating Embedded Elements in FPGAs," in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

Appendix A

Glitching Characterization Attempt

This Appendix describes several unsuccessful attempts to account for glitches in the power characterization scheme described in Chapter 5.

Figure A.1 illustrates the flow we used for characterizing the power of the DSP blocks. A Verilog description of the DSP block was synthesized to gates using Synopsys Design Compiler. A Verilog testbench was used to simulate a set of input vectors applied to the gate-level description of the DSP block in Verilog-XL. The simulation data was then fed to the Synopsys PrimePower simulator to obtain characterization information.

For input activities less than one, the testbench applied exactly one vector to the inputs of the DSP block each clock cycle. The activity at each input could be controlled by keeping the value of the bit or changing the value.

To attempt to imitate glitching, we generated testbenches where more than one vector was applied during each clock cycle. For example, to attempt to characterize the power for input activities of 1.5, a vector file with activity 0.5 (assuming 1 vector per clock cycle) was read in and 3 vectors were applied each clock cycle.

We observed that applying the vectors for a particular clock cycle at equally spaced intervals in the clock cycle, as shown in the testbench pseudocode in Figure A.2, does not imitate glitching properly. $\$T\_fraction$ is the length of the equally spaced intervals and

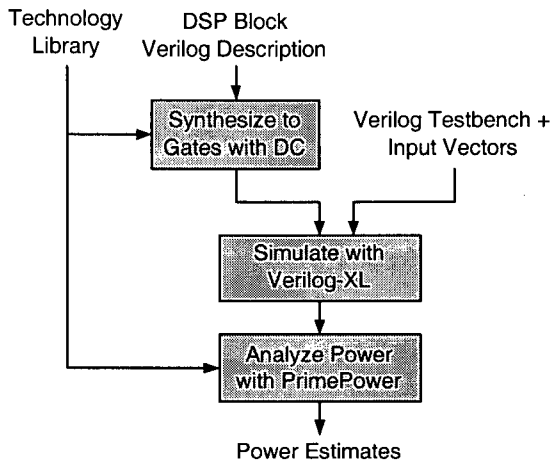

Figure A.1: Power Characterization Flow

```
reg [0:$inbits_ind] test_vector_input[0:N-1];
reg [0:$inbits_ind] inputs;
task test_top;
  integer i, j;
  begin
    @(posedge clock);
    @(negedge clock);
    reset = 0;
    @(posedge clock);
    for (i=0; (i+$integer_multiple) <= N; i=i+$integer_multiple) begin
      inputs = test_vector_input[i];
      #$T_fraction;
      inputs = test_vector_input[i+1];
      #$T_fraction;
     ...
      inputs = test_vector_input[i+($integer_multiple-1)];
      #$T_fraction;
      @(posedge clock);
    end
  end
endtask
```

Figure A.2: Testbench Pseudocode

$*integer_multiple* is the number of vectors to apply during each clock cycle.

For example, from Figure 5.3 we would expect that using a vector file with activity 0.5 (assuming 1 vector per clock cycle) and applying two vectors per cycle to achieve an effective input activity of 1.0 would give PrimePower estimates greater than the results for 0.9 input activity. For a DSP block where only input transitions during the high part of the clock cycle have an effect, the PrimePower estimates were approximately equal to the results for 0.5 input activity instead because only half the transitions had any effect.

To avoid this problem, a second method was attempted where $\$T\_fraction$ was set to half the period divided by $*integer_multiple*, so that all the vectors would be applied during the high part of the clock cycle. This led to overestimates of the power because, in reality, glitching at the inputs to the DSP block can take place in any part of the clock cycle.

To properly imitate glitching for characterization, a more sophisticated testbench generator would be required that incorporates some sort of statistical or probabilistic model that dictates when vectors should be applied to the DSP block inputs.

Appendix B

Detailed Results from Comparison

Table B.1: FIR Filter Results for 0.25 Average Input Activity

Vector Set	Multiplier	HHVPR Energy	PrimePower Energy	% Error
vec_025_0_a	mult_rtl_0	6.93E-11	6.31E-11	9.8%
	mult_rtl_1	6.92E-11	6.01E-11	15.1%
	mult_rtl_2	6.92E-11	5.87E-11	17.8%
	mult_rtl_3	6.93E-11	6.07E-11	14.2%
vec_025_0_b	mult_rtl_0	6.95E-11	6.14E-11	13.1%
	mult_rtl_1	6.92E-11	6.05E-11	14.3%
	mult_rtl_2	6.93E-11	5.82E-11	19.1%
	mult_rtl_3	6.97E-11	6.08E-11	14.5%
vec_025_0_c	mult_rtl_0	6.95E-11	6.32E-11	10.0%
	mult_rtl_1	6.93E-11	5.92E-11	17.1%
	mult_rtl_2	6.95E-11	5.95E-11	16.8%
	mult_rtl_3	6.95E-11	6.12E-11	13.5%
vec_025_0_d	mult_rtl_0	6.92E-11	6.24E-11	10.9%
	mult_rtl_1	6.91E-11	5.90E-11	17.2%
	mult_rtl_2	6.96E-11	5.93E-11	17.3%
	mult_rtl_3	6.96E-11	6.02E-11	15.5%
vec_025_0_e	mult_rtl_0	6.95E-11	6.25E-11	11.2%
	mult_rtl_1	6.95E-11	6.06E-11	14.7%
	mult_rtl_2	6.95E-11	5.92E-11	17.3%
	mult_rtl_3	6.97E-11	6.17E-11	12.9%
			average percentage error	14.6%
			average for just multiplier instance #0	11.0%
			average for just multiplier instance #1	15.7%
			average for just multiplier instance #2	17.7%
			average for just multiplier instance #3	14.1%

Table B.2: FIR Filter Results for 0.50 Average Input Activity

Vector Set	Multiplier	HHVPR Energy	PrimePower Energy	% Error
vec_050_0_a	mult_rtl_0	8.92E-11	1.09E-10	18.2%
	mult_rtl_1	8.90E-11	1.04E-10	14.4%
	mult_rtl_2	8.95E-11	1.04E-10	14.0%
	mult_rtl_3	8.93E-11	1.06E-10	15.5%
vec_050_0_b	mult_rtl_0	8.92E-11	1.10E-10	19.2%
	mult_rtl_1	8.92E-11	1.04E-10	14.1%
	mult_rtl_2	8.94E-11	1.03E-10	13.4%
	mult_rtl_3	8.92E-11	1.07E-10	16.2%
vec_050_0_c	mult_rtl_0	8.93E-11	1.11E-10	19.4%
	mult_rtl_1	8.92E-11	1.04E-10	14.2%
	mult_rtl_2	8.95E-11	1.04E-10	14.2%
	mult_rtl_3	8.95E-11	1.06E-10	15.4%
vec_050_0_d	mult_rtl_0	8.91E-11	1.07E-10	16.8%
	mult_rtl_1	8.94E-11	1.05E-10	15.0%
	mult_rtl_2	8.90E-11	1.01E-10	12.3%
	mult_rtl_3	8.94E-11	1.07E-10	16.2%
vec_050_0_e	mult_rtl_0	8.93E-11	1.12E-10	20.6%
	mult_rtl_1	8.92E-11	1.04E-10	14.5%
	mult_rtl_2	8.94E-11	1.02E-10	12.2%
	mult_rtl_3	8.92E-11	1.06E-10	15.7%
			average percentage error	15.6%
			average for just multiplier instance #0	18.9%
			average for just multiplier instance #1	14.4%
			average for just multiplier instance #2	13.2%
			average for just multiplier instance #3	15.8%

Table B.3: FIR Filter Results for 0.75 Average Input Activity

Vector Set	Multiplier	HHVPR Energy	PrimePower Energy	% Error
vec_075_0_a	mult_rtl_0	9.97E-11	1.52E-10	34.3%
	mult_rtl_1	9.96E-11	1.42E-10	29.7%
	mult_rtl_2	9.95E-11	1.38E-10	27.9%
	mult_rtl_3	9.96E-11	1.44E-10	30.7%
vec_075_0_b	mult_rtl_0	9.96E-11	1.52E-10	34.6%
	mult_rtl_1	9.96E-11	1.41E-10	29.2%
	mult_rtl_2	9.97E-11	1.38E-10	27.7%
	mult_rtl_3	9.96E-11	1.43E-10	30.5%
vec_075_0_c	mult_rtl_0	9.96E-11	1.49E-10	33.3%
	mult_rtl_1	9.95E-11	1.41E-10	29.4%
	mult_rtl_2	9.95E-11	1.38E-10	27.7%
	mult_rtl_3	9.96E-11	1.47E-10	32.4%
vec_075_0_d	mult_rtl_0	9.96E-11	1.55E-10	35.8%
	mult_rtl_1	9.97E-11	1.39E-10	28.3%
	mult_rtl_2	9.97E-11	1.41E-10	29.4%
	mult_rtl_3	9.97E-11	1.50E-10	33.6%
vec_075_0_e	mult_rtl_0	9.96E-11	1.55E-10	35.9%
	mult_rtl_1	9.96E-11	1.39E-10	28.4%
	mult_rtl_2	9.97E-11	1.37E-10	27.4%
	mult_rtl_3	9.96E-11	1.44E-10	30.9%
			average percentage error	30.9%
			average for just multiplier instance #0	34.8%
			average for just multiplier instance #1	29.0%
			average for just multiplier instance #2	28.0%
			average for just multiplier instance #3	31.6%

Table B.4: Differential Equation Solver Results for 0.25 Average Input Activity

Vector Set	DSP Block	PVPR Energy	PrimePower Energy	% Error
vec_025_0_a	dspblock0	9.51E-10	4.16E-09	77.1%
	dspblock1	1.01E-09	1.52E-09	33.3%
	dspblock2	1.12E-09	1.65E-09	32.2%
vec_025_0_b	dspblock0	9.04E-10	4.03E-09	77.6%
	dspblock1	1.04E-09	1.33E-09	21.8%
	dspblock2	1.23E-09	1.58E-09	22.2%
vec_025_0_c	dspblock0	8.95E-10	3.88E-09	76.9%
	dspblock1	1.01E-09	1.28E-09	21.2%
	dspblock2	1.23E-09	1.49E-09	17.4%
vec_025_0_d	dspblock0	1.05E-09	4.24E-09	75.4%
	dspblock1	1.04E-09	1.31E-09	21.0%
	dspblock2	1.17E-09	1.73E-09	32.1%
vec_025_0_e	dspblock0	8.02E-10	3.76E-09	78.7%
	dspblock1	1.00E-09	1.22E-09	17.7%
	dspblock2	1.12E-09	1.27E-09	12.0%
			average percentage error	41.1%
			average for just DSP block instance #0	77.1%
			average for just DSP block instance #1	23.0%
			average for just DSP block instance #2	23.2%

Table B.5: Differential Equation Solver Results for 0.50 Average Input Activity

Vector Set	DSP Block	PVPR Energy	PrimePower Energy	% Error
vec_050_0_a	dspblock0	8.88E-10	3.79E-09	76.6%
	dspblock1	1.06E-09	1.22E-09	12.9%
	dspblock2	1.26E-09	1.51E-09	16.9%
vec_050_0_b	dspblock0	9.24E-10	3.88E-09	76.2%
	dspblock1	1.06E-09	1.36E-09	21.9%
	dspblock2	1.29E-09	1.66E-09	22.5%
vec_050_0_c	dspblock0	8.78E-10	4.14E-09	78.8%
	dspblock1	1.05E-09	1.73E-09	39.2%
	dspblock2	1.20E-09	1.75E-09	31.6%
vec_050_0_d	dspblock0	9.59E-10	3.97E-09	75.9%
	dspblock1	1.04E-09	1.38E-09	24.8%
	dspblock2	1.28E-09	1.54E-09	16.9%
vec_050_0_e	dspblock0	9.19E-10	3.87E-09	76.2%
	dspblock1	1.06E-09	1.25E-09	15.1%
	dspblock2	1.31E-09	1.62E-09	19.0%
			average percentage error	40.3%
			average for just DSP block instance #0	76.7%
			average for just DSP block instance #1	22.8%
			average for just DSP block instance #2	21.4%

Table B.6: Differential Equation Solver Results for 0.75 Average Input Activity

Vector Set	DSP Block	PVPR Energy	PrimePower Energy	% Error
vec_075_0_a	dspblock0	8.95E-10	3.89E-09	77.0%
	dspblock1	1.03E-09	1.44E-09	28.4%
	dspblock2	1.23E-09	1.61E-09	23.5%
vec_075_0_b	dspblock0	8.70E-10	3.82E-09	77.2%
	dspblock1	1.01E-09	1.25E-09	19.3%
	dspblock2	1.27E-09	1.64E-09	22.7%
vec_075_0_c	dspblock0	8.83E-10	3.78E-09	76.6%
	dspblock1	1.01E-09	1.26E-09	19.7%
	dspblock2	1.27E-09	1.62E-09	21.6%
vec_075_0_d	dspblock0	9.13E-10	3.94E-09	76.8%
	dspblock1	1.03E-09	1.40E-09	26.2%
	dspblock2	1.24E-09	1.69E-09	26.5%
vec_075_0_e	dspblock0	9.37E-10	3.84E-09	75.6%
	dspblock1	1.01E-09	1.19E-09	14.9%
	dspblock2	1.18E-09	3.84E-09	69.3%
			average percentage error	43.7%
			average for just DSP block instance #0	76.7%
			average for just DSP block instance #1	21.7%
			average for just DSP block instance #2	32.7%