LOW-POWER ARCHITECTURAL DESIGN METHODOLOGIES

by

Paul Eric Landman

Memorandum No. UCB/ERL M94/62

30 August 1994

LOW-POWER ARCHITECTURAL DESIGN METHODOLOGIES

by

Paul Eric Landman

Memorandum No. UCB/ERL M94/62

30 August 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Low-Power Architectural Design Methodologies

by

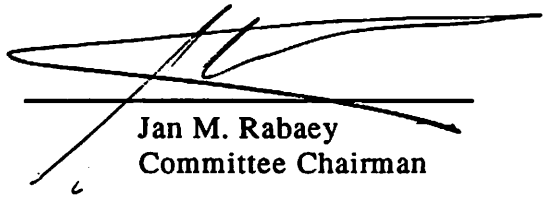
Paul Eric Landman

Doctor of Philosophy in Engineering -
Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Jan M. Rabaey, Chair

In recent years, power consumption has become a critical design concern for many VLSI systems. Nowhere is this more true than for portable, battery-operated applications, where power consumption has perhaps superceded speed and area as the overriding implementation constraint. This adds another degree of freedom - and complexity - to the design process and mandates the need for design techniques and CAD tools that address power, as well as area and speed. This thesis presents a methodology and a set of tools that support low-power system design. Low-power techniques at levels ranging from technology to architecture are presented and their relative merits are compared. Several case studies demonstrate that architecture and system-level optimizations offer the greatest opportunities for power reduction. A survey of existing power analysis tools, however, reveals a marked lack of power-conscious tools at these levels. Addressing this issue, a collection of techniques for modeling power at the register-transfer (RT) level of abstraction is described. These techniques model the impact of design complexity and signal activity on datapath, memory, control, and interconnect power consumption. Several VLSI design examples are used to verify the proposed tools, which exhibit near switch-level accuracy at RT-level speeds. Finally, an integrated design space exploration environment is described that spans several levels of abstraction and embodies many of the power optimization and analysis strategies presented in this thesis.



Jan M. Rabaey
Committee Chairman

To my wife, Karen

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisor Professor Jan Rabaey. His vision and foresight was instrumental in the definition of this project and in every stage of its development. He is a truly exceptional researcher whom I greatly admire. I would also like to acknowledge our research groups' other fearless leader, Professor Bob Brodersen. He was the driving force behind the InfoPad project and, thankfully, took the rest of us along for ride.

Professors John Wawrzynek, David Aldous, and Ed Lee all provided early guidance for my research as members of my Qualifying Exam Committee. Professors Wawrzynek and Aldous went above and beyond the call of duty by also serving on my dissertation committee and actually taking the time to read this monster.

I'd also like to thank a number of my coworkers and fellow slave laborers. Arthur Abnous took on the task of writing the parser and VHDL code generator for SPA. He finished them in record time and certainly did a better job than I ever could have. Ole Bentz (or King Ole, as we like to call him) was always available to track down those nasty hardware mapper bugs (except for when he took that vacation to Canada for which I'll never forgive him). Renu Mehra is responsible for all the algorithm-level power models in this thesis, and was my co-author on the paper from which Chapter 7 was derived. She also designed the infamous register file, which will certainly be remembered long after the rest of us are gone. Alfred Young, David Lidsky, and Steve Stoiber provided me with several excellent low-power case studies. I would also like to thank the other members of the HYPER team - Ingrid Verbauwhede, Lisa Guerra, and Sean Huang - for giving the HYPER project some personality and keeping it alive and well.

Several others students and staff also have my gratitude. Anantha Chandrakasan pioneered many of the low-power design strategies described in Chapter 2. My thanks also go out to Andy

Burstein for constructively criticizing (read "thrashing") my research papers, to Tom Burd for designing and supporting the low-power cell library, and to Sam Sheng, Kevin Zimmerman, Sue Mellers, Ken Lutz, and Brian Richards for supporting pretty much everything else. I would also like to mention some of the students who put me on the right track when I was first starting out. My sincerest thanks to Miodrag Potkonjak, Phu Hoang, Monte Mar, Mani Srivastava, Bill Barringer, Shankar Narayanaswamy, Kevin Kornegay, Lars Thon, and Jane Sun.

On the administrative side, I would like to acknowledge Tom Boot and Corey Schaffer for making things run so smoothly and for supplying me with an outrageous number of sheet protectors. I'm grateful to Carole Frank for making sure that I got paid each month, and I'm even more grateful to ARPA and the National Science Foundation for supplying her the money with which to pay me!

On a more personal note, there are several individuals who made the past years a little more bearable. First, thanks to Eric Boskin for proving to me that you can be an engineer and still have a life. Also, I would like to thank all of my friends for putting up with my moaning and groaning. David Yu, in particular, has been a good friend for more than eight years, and I owe him a lot.

Lastly, I'd like to recognize the members of my family. My mother-in-law Bonnie, her husband (Big) Paul, and my sister-in-law Katie have encouraged me every step of the way, and I appreciate their unwavering support. Also, to my brother-in-law Scott and sister Susan - you've always had more faith in me than I've had in myself. Thank you both. To my parents, Art and Fran, I owe more to you than I could ever repay. Thank you for your unconditional love and support.

And to my wife, Karen. What can I say. During the weekdays you did all the housework. During the weekends you sat with me in Cory Hall while I worked. You read, you edited, you listened, you typed... You worked as hard on this dissertation as I did. I know that the last five years were not quite what you had in mind. Thank you for putting my dreams ahead of your own.

CHAPTER 1

Introduction

Engineering is not an exact science. For a given engineering problem there is rarely (if ever) an "optimum" solution that is ideal in all respects. Instead, given a set of objectives and constraints, the engineer must constantly make trade-offs, balancing often conflicting requirements to arrive at a solution that best meets the project objectives while still satisfying the constraints. In the context of integrated circuit design, this often translates to maximizing the system performance while minimizing cost. The exact balance between these conflicting goals will depend on the particular system being designed. For low-end consumer electronics, low cost might be the key objective. In that case, the task of the designer would likely be to minimize the product cost, while still achieving the required performance. In contrast, the designer of a high-end component might be mainly concerned with obtaining the maximum possible performance within a less stringent cost requirement. In both cases the objective is to achieve maximum performance at minimum cost.

1.1 Historical Perspective

But by what metric should "performance" be measured, and what factors will influence "cost"? Historically, system performance has been synonymous with circuit speed or processing power. For example, in the microprocessor world, performance is often measured in Millions of Instructions Per Second (MIPS) or Millions of FLoating point Operations Per Second (MFLOPS). In other words, the highest "performance" system is the one that can perform the most computations in a given amount of time. Likewise, in the analog domain, bandwidth (a frequency-domain measure of circuit speed) is a common performance metric. The question of cost really depends on the implementation strategy being considered. For integrated circuits there is a fairly direct correspondence between silicon area and cost. Increasing the implementation area tends to result in higher packaging costs as well as reduced fabrication yield with both effects translating immediately to increased product cost. Moreover, improvements in system performance generally come at the expense of silicon real estate. So, historically, the task of the VLSI designer has been to explore the Area-Time (AT) implementation space, attempting to strike a reasonable balance between these often conflicting objectives.

But area and time are not the only metrics by which we can measure implementation quality. Power consumption is yet another criterion. Until recently, power considerations were often of only secondary concern - taking the back seat to both area and speed. Of course, there are exceptions to this rule; for example, designers of portable devices such as wrist watches have always placed considerable emphasis on minimizing power in order to maximize battery life. For the most part, however, designers of mainstream electronic systems have considered power consumption only as an afterthought - designing for maximum performance regardless of the effects on power.

1.2 Motivation for Low Power

In recent years, however, this has begun to change and, increasingly, power is being given equal weight to area and speed considerations. Several factors have contributed to this trend. Perhaps the most visible driving factor has been the remarkable success and growth of the portable consumer electronics market [Man91][May92][Mel93][Mob94][Wil92]. Lap-top computers, Personal Digital Assistants (PDA's), cellular phones, and pagers have enjoyed considerable success among consumers, and the market for these and other portable devices is only projected to increase in the future.

For these applications, power consumption has become a critical design concern and has perhaps superceded speed and area as the overriding implementation constraint [Cas94][Col93]. The reason for this is illustrated by Figure 1-1, which depicts the projected power budget for a future portable communications terminal were it implemented using off-the-shelf components not designed for low-power operation [Cha92a]. The total power consumption for such a system hovers around 40W. With modern Nickel-Cadmium battery technologies offering around 20 Watt-hours/pound (Figure 1-2) this terminal would require 20 pounds of batteries for 10 hours of operation between recharges [Eag92][Mal92]. Even more advanced battery technologies, such as Nickel-Metal-Hydride at 30-35 Watt-hours/pound, would only bring battery weights down to 7 pounds. In the absence of low-power design techniques, then, current and future portable devices will suffer from either very short battery life or unreasonably heavy battery packs. As a result, the portable electronics market provides a very strong motivation for low-power design.

Still, portability is not the sole driving force behind the push for low-power. There is also pressure on producers of high-end products to reduce power consumption. Figure 1-3 shows the current trend in microprocessor power consumption compiled from results reported at the International Solid-States Circuits Conference over the last 15 years [Rab94]. The figure demonstrates that contemporary high performance processors are dissipating as much as 30W

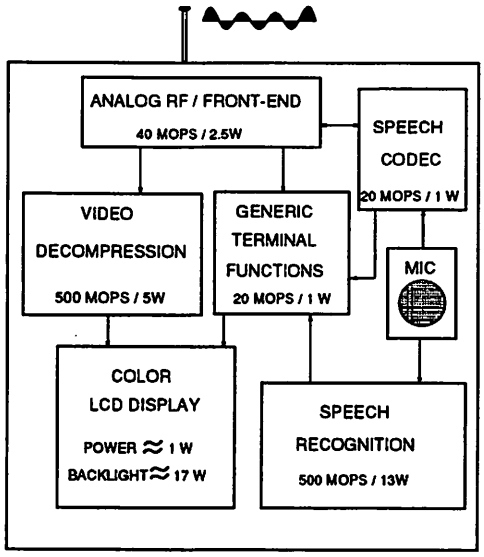


Figure 1-1 : Power budget for PCS terminal using off-the-shelf components [Cha92a]

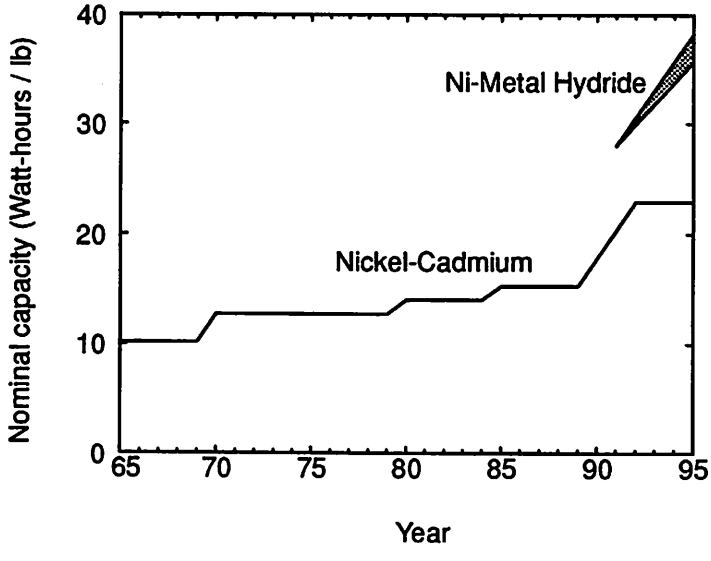


Figure 1-2 : Energy storage capacity trends for common battery technologies [Eag92]

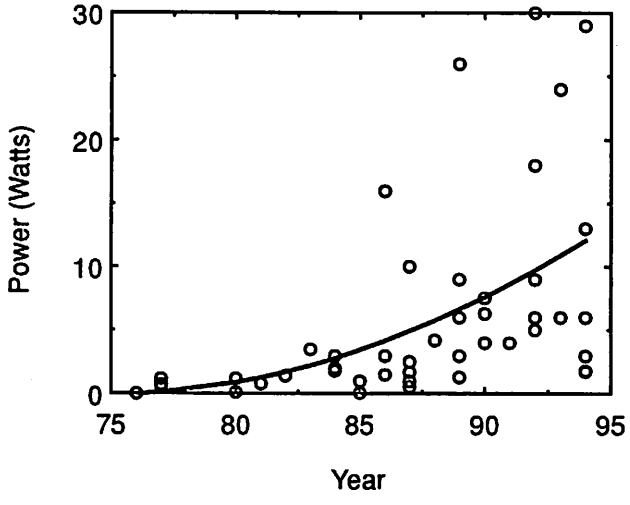


Figure 1-3 : Trends in microprocessor power consumption (after [Rab94])

[Dob92]. The cost associated with packaging and cooling such devices is becoming prohibitive. Since core power consumption must be dissipated through the packaging, increasingly expensive packaging and cooling strategies are required as chip power consumption increases [Piv94]. Consequently, there is a clear financial advantage to reducing the power consumed by high performance systems.

In addition to cost, there is the issue of reliability. High power systems tend to run hot, and high temperature tends to exacerbate several silicon failure mechanisms. Every 10°C increase in operating temperature roughly doubles a component's failure rate [Sma94]. Figure 1-4 illustrates this very definite relationship between temperature and the various failure mechanisms such as electromigration, junction fatigue, and gate dielectric breakdown.

1.3 Motivation for Power-Conscious Design Tools

So for a variety of reasons, designers are increasingly led to consider power as a major system design criterion. Now, instead of considering only area-time (AT) trade-offs, designers are

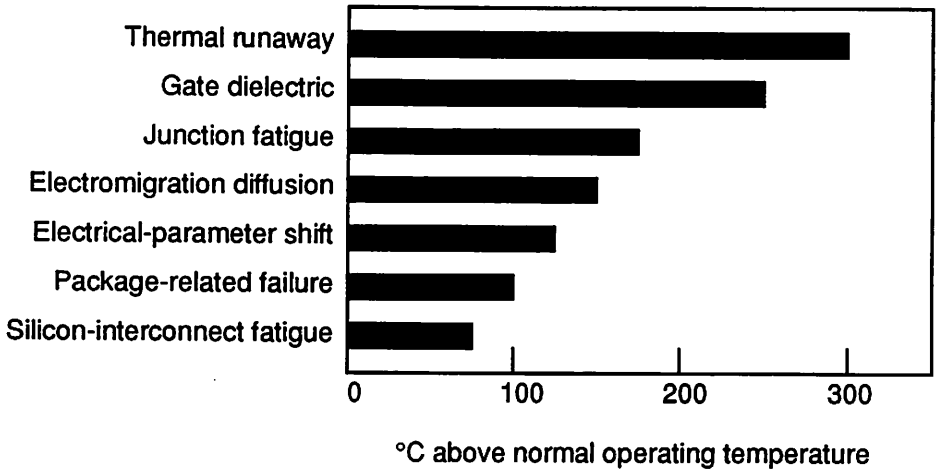


Figure 1-4 : Onset temperatures of various failure mechanisms (after [Sma94]).

encouraged to explore the full area-time-*power* (ATP) implementation space. This adds another degree of freedom - *and complexity* - to the design process, and with transistor counts of modern integrated circuits in the millions, designers must resort to computer-aided design (CAD) tools to manage the sheer volume of data involved. Since area and speed have for many years been the primary concern of system designers, many tools already exist to help the designer analyze and optimize his design in these dimensions. As power is a relatively new optimization criterion, however, few tools exist to facilitate low-power design. So if there is a need for low-power design tools, the next logical question is what kinds of tools are needed.

As in the case of area and speed, two of the most useful classes of design aids are analysis tools and optimization (or synthesis) tools. Actually, these two classes of tools are closely related. In fact, many optimization tools employ iterative improvement schemes and rely on embedded estimation or analysis routines to gauge progress at each step of the optimization process. Other optimization approaches may rely on heuristics, which have empirically demonstrated success at reducing system power consumption. Often the heuristics themselves will have been developed with the aid of analysis tools that indicate which low-power techniques might be the most

effective. Thus, whether directly or indirectly, optimization tools (for power or other criteria) tend to rely heavily on the existence of estimation and analysis tools. So a logical progression from analysis to optimization is evident in the evolution of CAD tools.

With low-power design still in its infancy, the focus has been primarily on tools for analyzing power. Even so, there are relatively few commercial tools available for this important task. Moreover, most contemporary power analysis tools operate at relatively low levels of abstraction such as the gate or circuit levels. Design decisions made at these lower levels tend to have a limited, local impact on power consumption. Typically, the decisions made at the highest levels (architecture and system) offer the most global and dramatic power reductions [Cha92b]. So, while offering good accuracy, gate- and circuit-level analysis tools provide feedback too late in the design process to be particularly useful in either the optimization of a specific design or in the development of generic low-power heuristics. Moreover, the large number of gate- and circuit-level components makes rapid analysis of large systems difficult, if not impossible, at these levels.

These facts suggest that high-level power exploration and analysis tools might prove extremely useful. Despite this, few such high-level tools are currently available. Those that do exist, while providing rapid results, suffer from significant inaccuracies - a weakness stemming from flaws in the underlying models [Lan94].

1.4 Research Objectives

The research presented in this thesis is an attempt to remedy this situation. The specific contributions are a methodology for low-power design space exploration, as well as a set of tools that support this methodology. The approach relies heavily on a novel technique for architectural power analysis, which allows the user to generate black-box power models for architecture-level components. These models preserve the accuracy associated with the gate or circuit levels, while retaining the estimation speeds more commonly associated with the higher levels of abstraction.

1.5 Overview

This thesis divides into nine chapters. This chapter has presented a motivation for and introduction to the low-power methodologies and tools that will be presented in subsequent chapters. Too often, methodologies and tools are developed in the abstract, with little regard for what is practical or useful from a design standpoint. In contrast, this research was completely motivated and driven by low-power design considerations. In accordance with this philosophy, Chapter 2 will describe techniques for low-power design at levels ranging from process technology to system and will motivate the need for high-level power exploration tools. Chapter 3 will then describe the state of the art in power estimation and in doing so will demonstrate a lack of power-conscious tools at the higher levels.

This will lead to a proposal for an architectural power analysis technique in Chapter 4. The chapter will discuss power estimation strategies for each of the key chip components: datapath, memory, control, and interconnect. It will do so in a general context, however, without referring to any particular tool that implements the strategy. The tools implementing this power analysis methodology will be described in Chapters 5 and 6. Chapter 5 will describe a tool targeted at DSP applications and architectures, while Chapter 6 discusses a more generic tool applicable to a wide range of general-purpose and application-specific architectures.

As stated above, the contribution of this research is not just a tool for power analysis, but rather a low-power methodology along with a set of supporting tools. Chapter 7 presents this methodology, using a case study to illustrate the key aspects of design space exploration for low power. Finally, Chapters 8 and 9 present topics for future research and conclusions regarding low-power tools and low-power design.

CHAPTER 2

Low-Power Design

The purpose of CAD tools is to aid and support the designer in the process of implementing complex systems. While this may seem like an obvious statement, it is surprising how often CAD development occurs in a vacuum with little influence from designers. Consequently, while undertaking this research a good deal of emphasis was placed on ensuring that the design techniques and methodologies drove the tool development - rather than the other way around.

In accordance with this philosophy, we present a comprehensive overview of low-power design before attempting a discussion of any tools or design environments. This chapter describes low-power design techniques at abstraction levels ranging from layout and technology to architecture and system. In reading this chapter, it should become clear that high-level design decisions - those made at the architecture or system level - have the most dramatic impact on power consumption.

Increasingly, this fact is forcing low-power designers to take a system-level perspective and place a substantial amount of effort into high-level design issues. This observation was a major

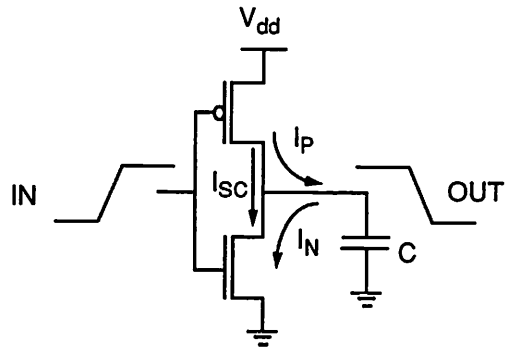


Figure 2-1 : Current flow in CMOS inverter

motivating factor in the development of the high-level power analysis and exploration techniques presented in this thesis.

This chapter, then, presents the state of the art in low-power digital design and divides into nine main sections. Sections 2.1-2.3 provide background, discussing the power consumption mechanism in CMOS, as well as recurring themes in low-power design. Next, Sections 2.4-2.9 cover specific low-power design techniques from the technology to the algorithm level of abstraction.

2.1 Power Consumption in CMOS Circuits

Before entering into a discussion of low-power design techniques and tools, we must first discuss the mechanics of power consumption in CMOS circuits. Consider the CMOS inverter of Figure 2-1. The power consumed when this inverter is in use can be decomposed into two basic classes: static and dynamic.

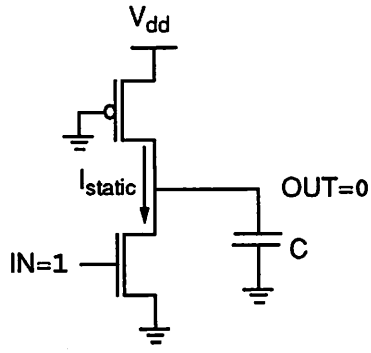


Figure 2-2 : Static power dissipation in pseudo-NMOS inverter

2.1.1 Static Power

Ideally, CMOS circuits dissipate no static (DC) power since in the steady state there is no direct path from V_{dd} to ground. This scenario can never be realized in practice, however, since in reality the MOS transistor is not a perfect switch. There will always be leakage currents, subthreshold currents, and substrate injection currents, which give rise to a static component of CMOS power dissipation. For a submicron NMOS device with an effective W/L = 10/0.5, the substrate current is on the order of 1-100μA for a V_{dd} of 5V [Wat89]. Since the substrate current reaches its maximum for gate voltages near $0.4V_{dd}$ [Wat89] and since gate voltages are only transiently in this range as devices switch, the actual power contribution of the substrate current is several orders of magnitude below other contributors. Likewise, the junction leakage currents associated with parasitic diodes in the CMOS device structure are on the order of nanoamps and will have little effect on overall power consumption [Wes88].

Another form of static power dissipation occurs for so-called ratioed logic. Pseudo-NMOS, as depicted in Figure 2-2 is an example of a ratioed CMOS logic family. In this example, the PMOS pull-up is always on and acts as a load device for the NMOS pull-down network. Therefore, when the gate output is in the low-state, there is a direct path from V_{dd} to ground and static currents flow.

In this state, the exact value of the output voltage depends on the strength ratio of the PMOS and NMOS networks. The static power consumed by these logic families can be considerable. For this reason, logic families that experience static power consumption should be avoided for low-power design. With that in mind, the static component of power consumption in *low-power* CMOS circuits should be negligible, and the focus shifts primarily to dynamic power consumption.

2.1.2 Dynamic Power

The dynamic component of power dissipation arises from the transient switching behavior of the CMOS device. At some point during the switching transient, both the NMOS and PMOS devices in Figure 2-1 will be turned on. This occurs for gates voltages between V_{tn} and V_{dd}-$|V_{tp}|$. During this time, a short-circuit exists between V_{dd} and ground and currents are allowed to flow. A detailed analysis of this phenomenon by Veendrick reveals that with careful design for balanced input and output rise times this component can be kept below 10-15% of the total power [Vee84]. Thus, although it cannot always be completely neglected, it is certainly not the dominant component of power dissipation in CMOS circuits.

Instead, dynamic dissipation due to capacitive switching consumes most of the power used by CMOS circuits. This component of dynamic power dissipation is the result of charging and discharging parasitic capacitances in the circuit. The situation is modeled in Figure 2-1 where the parasitic capacitances have been lumped at the output in load capacitance C. Consider the behavior of the circuit over one full cycle of operation with the input voltage going from V_{dd} to ground and back to V_{dd} again. As the input switches from high to low, the NMOS pull-down network is cut off and PMOS pull-up network is activated, charging load capacitance C up to V_{dd}. This charging process draws an energy equal to CV_{dd}^2 from the power supply. Half of this is dissipated immediately in the PMOS transistors, while the other half is stored on the load capacitance. Then, when the input returns to V_{dd} the process is reversed and the capacitance is discharged, its energy being dissipated in the NMOS network. In summary, every time a capacitive

node switches from ground to V_{dd}, an energy of CV_{dd}^2 is consumed.

This leads to the conclusion that CMOS power consumption depends on the switching *activity* of the signals involved. In this context, we can define activity, α, as the expected number of transitions per data cycle. If this is coupled with the average data-rate, f, which may be the clock frequency in a synchronous system, then the effective frequency of switching is given by the product of the activity and the data rate: αf. Half of these transitions charge the nodal capacitance, while the other half discharge it. This leads to the following formulation for average CMOS power consumption:

$$P_{dyn} = \left(\frac{1}{2}CV_{dd}^2\right)\alpha f \qquad \text{(EQ 1)}$$

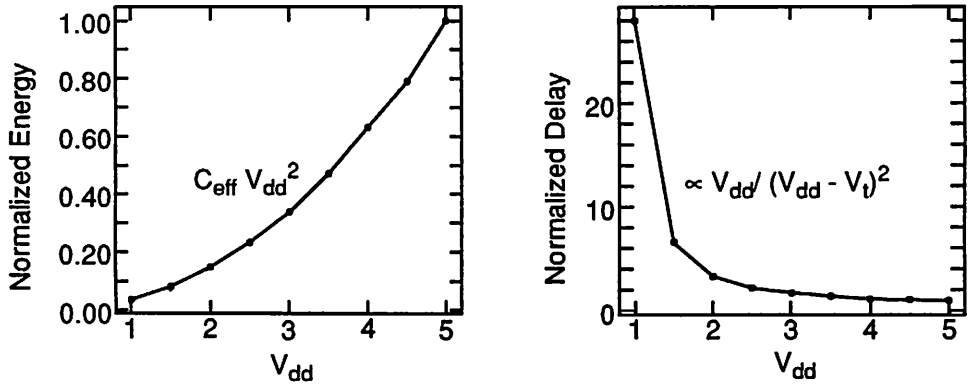
This result illustrates that the dynamic power is proportional to switching activity, capacitive loading, and the square of the supply voltage. In CMOS circuits, this component of power dissipation is by far the most important, typically accounting for at least 90% of the total power dissipation [Vee84].

2.2 Degrees of Freedom

The previous section revealed the three degrees of freedom inherent in the low-power design space: voltage, physical capacitance, and activity. Optimizing for power invariably involves an attempt to reduce one or more of these factors. Unfortunately, these parameters are not completely orthogonal and cannot be optimized independently. This section briefly discusses each of the factors, describing their relative importance, as well as the interactions that complicate the power optimization process.

2.2.1 Voltage

With its quadratic relationship to power, voltage reduction offers the most direct and dramatic means of minimizing energy consumption. Without requiring any special circuits or technologies,



(a) Energy vs. voltage

(b) Delay vs. voltage

Figure 2-3 : Energy and delay as a function of supply voltage

a factor of two reduction in supply voltage yields a factor of four decrease in energy (see Figure 2-3a). Furthermore, this power reduction is a global effect, experienced not only in one sub-circuit or block of the chip, but throughout the entire design. Because of this quadratic relationship, designers of low-power systems are often willing to sacrifice increased physical capacitance or circuit activity for reduced voltage. Unfortunately, supply voltage cannot be decreased without bound. In fact, several factors other than power influence selection of a system supply voltage. The primary factors are performance requirements and compatibility issues.

As supply voltage is lowered, circuit delays increase (see Figure 2-3b) leading to reduced system performance. To the first order, device currents are given by:

$$I_{dd} = \frac{\mu C_{ox} W}{2 L} (V_{dd} - V_t)^2 \qquad \text{(EQ 2)}$$

This leads to circuit delays of the order:

$$t = \frac{C V_{dd}}{I_{dd}} \propto \frac{V_{dd}}{(V_{dd} - V_t)^2} \qquad \text{(EQ 3)}$$

So, for $V_{dd} \gg V_t$ delays increase linearly with decreasing voltage. In order to meet system performance requirements, these delay increases cannot go unchecked. Some techniques must be applied, either technological or architectural, in order to compensate for this effect. As V_{dd} approaches the threshold voltage, however, delay penalties simply become unmanageable, limiting the advantages of going below a supply voltage of about $2V_t$.

Performance is not, however, the only limiting criterion. When going to non-standard voltage supplies, there is also the issue of compatibility and interoperability. Most off-the-shelf components operate from either a 5V supply or, more recently, a 3.3V supply [Bry93][Bur92b] [Swe93]. Unless the entire system is being designed completely from scratch it is likely that some amount of communications will be required with components operating at a standard voltage. This dilemma is lessened by the availability of highly efficient (>90%) DC-DC level converters, but still there is some cost involved in supporting several different supply voltages [Str94]. This suggests that it might be advantageous for designers to support only a small number of distinct intrasystem voltages. For example, custom chips in the system could be designed to operate off a single low voltage (e.g. $2V_t$) with level shifting only required for communication with the outside world. To account for parameter variations within and between chips, the supply would need to be set relative to the worst-case threshold, $V_{t,max}$.

To summarize, reducing supply voltage is paramount to lowering power consumption, and it often makes sense to increase physical capacitance and circuit activity in order to further reduce voltage. There are, however, limiting factors such as minimum performance and compatibility requirements that limit voltage scaling. These factors will likely lead designers to fix the voltage within a system. Once the supply has been fixed, it remains to address the issues of minimizing physical capacitance and activity at that operating voltage. The next two sections address these topics.

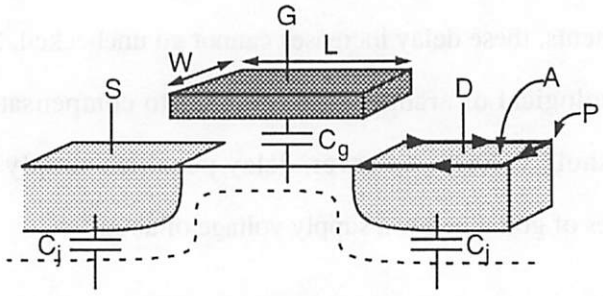


Figure 2-4 : Primary sources of device capacitance

2.2.2 Physical Capacitance

Dynamic power consumption depends linearly on the physical capacitance being switched. So, in addition to operating at low voltages, minimizing capacitance offers another technique for reducing power consumption. In order to properly evaluate this opportunity we must first understand what factors contribute to the physical capacitance of a circuit. Then we can consider how those factors can be manipulated to reduce power.

The physical capacitance in CMOS circuits stems from two primary sources: devices and interconnect. For devices, the most significant contributions come from the gate and junction capacitances as shown in Figure 2-4. The capacitance associated with the thin gate oxide of the transistor is usually the larger of the two. This term can be approximated as a parallel-plate (area) capacitance between the gate and the substrate or channel:

$$C_g = WLC_{ox} = WL\frac{\varepsilon_{ox}}{t_{ox}} \quad \text{(EQ 4)}$$

In addition, source/drain junction capacitances contribute to the overall device capacitance. These capacitances have both an area and a perimeter component and are non-linear with the voltage across the junction:

$$C_j(V) = AC_{j0}\left(1-\frac{V}{\phi_0}\right)^{-m} + PC_{jsw0}\left(1-\frac{V}{\phi_0}\right)^{-m} \quad \text{(EQ 5)}$$

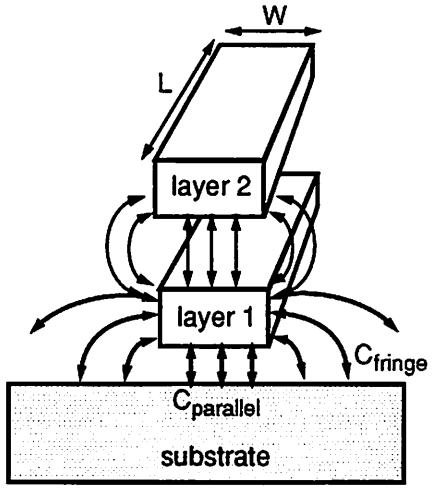


Figure 2-5 : Sources of interconnect capacitance

where A and P are the source/drain area and perimeter, C_{j0} and C_{jsw0} are equilibrium bottomwall and sidewall capacitances, ϕ_0 is the junction barrier potential, and m is the junction grading coefficient. Often, this non-linear capacitance is approximated by a large-signal equivalent linearized capacitance given by:

$$C_{jeq} = \frac{\int_{V_0}^{V_1} C_j(V)\,dV}{V_1 - V_0} \qquad \text{(EQ 6)}$$

where V_0 and V_1 describe the range of typical operating voltages for the junction. In the remainder of this thesis, we will often make the approximation that all circuit capacitances are fixed. In these cases, we assume that linearized, effective values are used to replace any non-linear capacitances.

In past technologies, device capacitances dominated over interconnect parasitics. As technologies continue to scale down, however, this no longer holds true and we must consider the contribution of interconnect to the overall physical capacitance. For the interconnect, there is the capacitance between each metallization layer and the substrate, as well as coupling capacitances between the layers themselves (see Figure 2-5). Each of these capacitances in turn has two

components: a parallel-plate component and a fringing component:

$$C_w = WLC_p + 2(W+L)C_f \quad \text{(EQ 7)}$$

Historically, the parallel-plate component, which increases linearly with both the width and the length of the wire, has been dominant. The fringing component starts to become significant, however, as the interconnect width becomes narrower and narrower relative to the wire thickness [Bak90].

With this understanding, we can now consider how to reduce physical capacitance. From the previous discussion, we recognize that capacitances can be kept at a minimum by using small devices and short wires. As with voltage, however, we are not free to optimize capacitance independently. For example, reducing device sizes will not only reduce physical capacitance, but will also reduce the current drive of the transistors, making the circuit operate more slowly. This loss in performance might prevent us from lowering V_{dd} as much as we might otherwise be able to do. In this scenario, we are giving up a possible quadratic reduction in power through voltage scaling for a linear reduction through capacitance scaling. So, if the designer is free to scale voltage it does not make sense to minimize physical capacitance without considering the side effects. Similar arguments can be applied to interconnect capacitance. If voltage and/or activity can be significantly reduced by allowing some increase in physical interconnect capacitance, then this may result in a net decrease in power. The key point to recognize is that low-power design is a joint optimization process in which the variables cannot be manipulated independently.

2.2.3 Activity

In addition to voltage and physical capacitance, switching activity also influences dynamic power consumption. A chip can contain a huge amount of physical capacitance, but if it does not switch then no dynamic power will be consumed. The activity determines how often this switching occurs. As mentioned above, there are two components to switching activity. The first is the data rate, f, which reflects how often on average new data arrives at each node. This data might

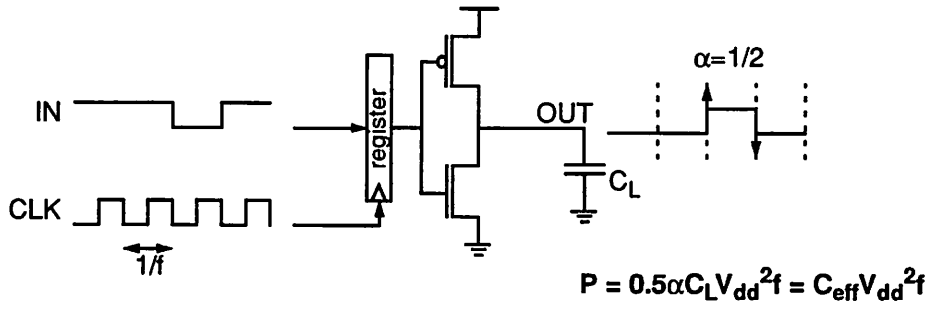


Figure 2-6 : Interpretation of switching activity in synchronous systems

or might not be different from the previous data value. In this sense, the data rate *f* describes how often on average switching *could* occur. For example, in synchronous systems *f* might correspond to the clock frequency (see Figure 2-6).

The second component of activity is the data activity, α. This factor corresponds to the expected number of transitions that will be triggered by the arrival of each new piece of data. So, while *f* determines the average periodicity of data arrivals, α determines how many transitions each arrival will spark. For circuits that don't experience *glitching*, α can be interpreted as the probability that a transition will occur during a single data period. For certain logic styles, however, glitching can be an important source of signal activity and, therefore, deserves some mention here [Ben94]. Glitching refers to spurious and unwanted transitions that occur before a node settles down to its final, steady-state value. Glitching often arises when paths with unbalanced propagation delays converge at the same point in the circuit. Since glitching can cause a node to make several power consuming transitions instead of one (i.e. $\alpha>1$) it should be avoided whenever possible.

The data activity α can be combined with the physical capacitance *C* to obtain an effective capacitance, $C_{eff}=\alpha C/2$, which describes the average capacitance *charged* during each 1/*f* data

period. This reflects the fact that neither the physical capacitance nor the activity alone determine dynamic power consumption. Instead, it is the effective capacitance, which combines the two, that truly determines the power consumed by a CMOS circuit:

$$P = \frac{1}{2}\alpha C V_{dd}^2 f = C_{eff} V_{dd}^2 f \qquad \text{(EQ 8)}$$

This discussion provides the first indication that data statistics can have a significant effect on power consumption. This is an important motivating force behind the power analysis techniques that will be presented in subsequent chapters. As we shall see, the weakness of many existing high-level estimation tools is that they ignore the effect of data statistics on power consumption. In contrast, the estimation techniques presented in this thesis accurately account for data activity. The precise manner in which this is achieved is the topic of Chapter 4.

As with voltage and physical capacitance, we can consider techniques for reducing switching activity as a means of saving power. For example, certain data representations such as sign-magnitude have an inherently lower activity than two's-complement [Cha94b]. Since sign-magnitude arithmetic is much more complex than two's-complement, however, there is a price to be paid for the reduced activity in terms of higher physical capacitance. This is yet another indication that low-power design is truly a joint optimization problem. In particular, optimization of activity cannot be undertaken independently without consideration for the impact on voltage and capacitance.

2.3 Recurring Themes in Low-Power Design

Sections 2.1 and 2.2 have provided a strong foundation from which to consider low-power CMOS design. Specifically, Section 2.1 derived the classical expression for dynamic power consumption in CMOS. This led to the realization that three primary parameters: voltage, physical capacitance, and activity determine the average power consumption of a digital CMOS circuit. Section 2.2 then went on to describe each of these factors individually, while emphasizing that

design for low-power must involve a joint rather than independent optimization of these three parameters. The upcoming sections present specific power reduction techniques applicable at various levels of abstraction. Many of these techniques follow a small number of common themes. The three principle themes are trading area/performance for power, avoiding waste, and exploiting locality.

Probably the most important theme is *trading area/performance for power.* As mentioned in Section 2.2.1, power can be reduced by decreasing the system supply voltage and allowing the performance of the system to degrade. This is an example of trading performance for power. If the system designer is not willing to give up the performance, he can consider applying techniques such as parallel processing to maintain performance at low voltage. Since many of these techniques incur an area penalty, we can think of this as trading area for power.

Another recurring low-power theme involves *avoiding waste.* For example, clocking modules when they are idle is a waste of power. Glitching is another example of wasted power and can be avoided by path balancing and choice of logic family. Other strategies for avoiding waste include using dedicated rather than programmable hardware and reducing control overhead by using regular algorithms and architectures. Avoiding waste can also take the form of designing systems to meet, rather than beat, performance requirements. If an application requires 25 MIPS of processing performance, there is no advantage gained by implementing a 50 MIPS processor at twice the power.

Exploiting locality is another important theme of low-power design. Global operations inherently consume a lot of power. Data must be transferred from one part of the chip to another at the expense of switching large bus capacitances. Furthermore, in poorly partitioned designs the same data might need to be stored in many parts of the chip, wasting still more power. In contrast, a design partitioned to exploit locality of reference can minimize the amount of expensive global communications employed in favor of much less costly local interconnect networks. Moreover,

especially for DSP applications, local data is more likely to be correlated and, therefore, to require fewer power consuming transitions. So, in its various forms, locality is an important concept in low-power design.

While not all low-power techniques can be classified as trading-off area/performance for power, avoiding waste, and exploiting locality these basic themes do describe many of the strategies that will be presented in the remainder of this chapter. The organization of these upcoming sections is by level of abstraction. Specifically, beginning with Section 2.4 and ending with Section 2.9, they cover low-power design methodologies for the technology, layout, circuit, gate, architecture, and algorithm levels, respectively.

2.4 Technology Level

At the lowest level of abstraction we can consider low-power design strategies in the context of both packaging and process technologies.

2.4.1 Packaging

Often a significant fraction of the total chip power consumption can be attributed not to core processing but to driving large off-chip capacitances. This is not surprising since off-chip capacitances are on the order of tens of picofarads while on-chip capacitances are in the tens of femtofarads. For conventional packaging technologies, Bakoglu suggests that pins contribute approximately 13-14 pF of capacitance each (10 pF for the pad and 3-4 pF for the printed circuit board traces) [Bak90]. Since dynamic power is proportional to capacitance, I/O power can be a significant portion of overall chip power consumption. The notion that I/O capacitance at the chip level can account for as much as 1/4 to 1/2 of the overall system power dissipation suggests that reduction of I/O power is a high priority in multi-chip systems. If the large capacitances associated

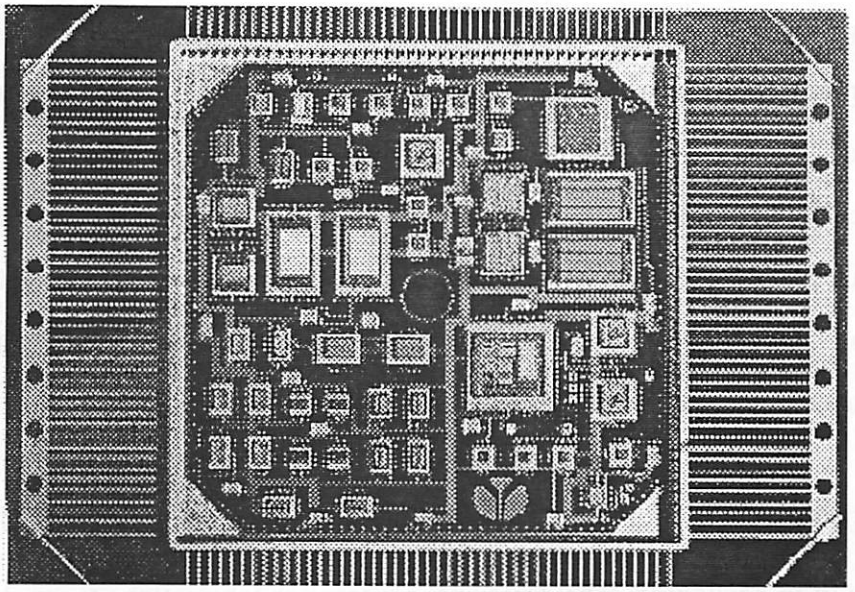


Figure 2-7 : A Rockwell International avionics processor using MCM packaging

with inter-chip I/O were drastically reduced, the I/O component of system power consumption would be reduced proportionally.

Packaging technology can have a dramatic impact on the physical capacitance involved in off-chip communications. For example, multi-chip modules or MCM's (an example of which is shown in Figure 2-7) offer a drastic reduction in the physical capacitance of off-chip wiring. In an MCM, all of the chips comprising the system are mounted on a single substrate, and the entire module is placed in a single package. Utilizing this technology, inter-chip I/O capacitances are reduced to the same order as on-chip capacitances [Ben90]. This is due not only to the elimination of the highly capacitive PCB traces, but also to the minimization of on-chip pad driver capacitances due to reduced off-chip load driving requirements. Thus, utilizing MCM technology, the I/O component of system power consumption can be kept at a minimum, shifting the focus of power optimization from I/O considerations to chip core considerations [Bur91].

Actually, low-power operation is only one of the advantages of MCM technology. In addition,

MCM's with their reduced chip-level interconnect lengths and capacitances can significantly reduce system delays resulting in higher performance, which can then be traded for lower power at the designer's discretion. Furthermore, this packaging technique raises the effective system integration level several orders of magnitude over existing packaging technologies. For projected submicron technologies, an 8"x10" MCM can be expected to house close to a billion transistors [Bak90]. This will relax current silicon area constraints and allow much needed flexibility in designing low-power architectures that trade area for power. So, both directly and indirectly selection of a packaging technology can have an important effect on system power consumption.

2.4.2 Process

In addition to packaging considerations, process (or fabrication) technology plays an important role in determining power consumption. This section presents two important process-based techniques for reducing power consumption: technology scaling and threshold voltage scaling.

Technology Scaling

Scaling of physical dimensions is a well-known technique for reducing circuit power consumption. Basically, scaling involves reducing all vertical and horizontal dimensions by a factor, S, greater than one. Thus, transistor widths and lengths are reduced, oxide thicknesses are reduced, depletion region widths are reduced, interconnect widths and thicknesses are reduced, etc. The first-order effects of scaling can be fairly easily derived [Bac84][Den74]. Device gate capacitances are of the form $C_g = WLC_{ox}$. If we scale down W, L, and t_{ox} by S, then this capacitance will scale down by S as well. Consequently, if system data rates and supply voltages remain unchanged, this factor of S reduction in capacitance is passed on directly to power:

$$\text{Fixed performance, fixed voltage: } P \propto \frac{1}{S} \qquad \text{(EQ 9)}$$

To give a concrete example, at the 1994 International Solid-State Circuits Conference, MIPS

Technologies attributed a 25% reduction in power consumption for their new 64b RISC processor solely to a migration from a 0.8μm to a 0.64μm technology [Yeu94].

The effect of scaling on delays is equally promising. Based on (EQ 2), the transistor current drive increases linearly with S. As a result, propagation delays, which are proportional to capacitance and inversely proportional to drive current, scale down by a factor of S^2. Assuming we are only trying to maintain system throughput rather than increase it, the improvement in circuit performance can be traded for lower power by reducing the supply voltage. In particular, neglecting V_t effects, the voltage can be reduced by a factor of S^2. This results in a S^4 reduction in device currents, and along with the capacitance scaling leads to an S^5 reduction in power:

$$\text{Fixed performance, variable voltage: } P \propto \frac{1}{S^5} \qquad \textbf{(EQ 10)}$$

This discussion, however, ignores many important second-order effects. For example, as scaling continues, interconnect parasitics eventually begin to dominate and change the picture substantially. The resistance of a wire is proportional to its length and inversely proportional to its thickness and width. Since in this discussion we are considering the impact of technology scaling on a *fixed design*, the local and global wire lengths should scale down by S along with the width and thickness of the wire. This means that the wire resistance should scale up by a factor of S overall. The wire capacitance is proportional to its width and length and inversely proportional to the oxide thickness. Consequently, the wire capacitance scales down by a factor of S. To summarize:

$$R_w \propto S \text{ and } C_w \propto \frac{1}{S} \qquad \textbf{(EQ 11)}$$

$$t_{wire} \propto R_w C_w \propto 1 \qquad \textbf{(EQ 12)}$$

This means that, unlike gate delays, the intrinsic interconnect delay does not scale down with physical dimensions. So at some point interconnect delays will start to dominate over gate delays and it will no longer be possible to scale down the supply voltage. This means that once again

power is reduced solely due to capacitance scaling:

$$\text{Parasitics dominated: } P \propto \frac{1}{S} \qquad \text{(EQ 13)}$$

Actually, the situation is even worse since the above analysis did not consider second-order effects such as the fringing component of wire capacitance, which may actually grow with reduced dimensions. As a result, realistically speaking, power may not scale down at all, but instead may stay approximately constant with technology scaling or even increase:

$$\text{Including 2nd-order effects: } P \propto 1 \text{ or more} \qquad \text{(EQ 14)}$$

The conclusion is that technology scaling offers significant benefits in terms of power only up to a point. Once parasitics begin to dominate, the power improvements slack off or disappear completely. So we cannot rely on technology scaling to reduce power indefinitely. We must turn to other techniques for lowering power consumption.

Threshold Voltage Reduction

Many process parameters, aside from lithographic dimensions, can have a large impact on circuit performance. For example, at low supply voltages the value of the threshold voltage (V_t) is extremely important. Section 2.2.1 revealed that threshold voltage places a limit on the minimum supply voltage that can be used without incurring unreasonable delay penalties. Based on this, it would seem reasonable to consider reducing threshold voltages in a low-power process.

Unfortunately, subthreshold conduction and noise margin considerations limit how low V_t can be set. Although devices are ideally "off" for gate voltages below V_t, in reality there is always some subthreshold conduction even for $V_{gs} < V_t$. The question is especially important for dynamic circuits for which subthreshold currents could causes erroneous charging or discharging of dynamic nodes. The relationship between gate voltage and subthreshold current is exponential. Each 0.1V reduction in V_{gs} below V_t reduces the subthreshold current by approximately one order of magnitude [Mul86]. Therefore, in order to prevent static currents from dominating chip power

and to ensure functionality of dynamic circuits, threshold voltages should be limited to a minimum of 0.3-0.5V.

Some researchers have considered operating circuits at threshold voltages that are much lower still [Bur93][Liu93]. For these circuits, implementation is confined to static circuit styles and a larger static power is accepted for the sake of reduced dynamic power. At these very low threshold voltages, the question of tolerance in the V_t specification becomes an issue. At best, this voltage is likely to vary 100mV in either direction, which forces the researchers to rely on a negative-feedback substrate biasing scheme to ensure chip functionality across process variations [Bur94].

Unfortunately, dimensional and threshold scaling are not always viable options. Aside from the drawbacks of interconnect non-scalability, submicron effects, and subthreshold conduction, chip designers often don't have complete freedom to arbitrarily scale their fabrication technology. Instead, economic factors as well as the capabilities of their fabrication facilities impose limits on minimum lithographic dimensions. For this reason, in order to achieve widespread acceptance, an ideal low-power methodology should not rely solely on technology scaling or specialized processing techniques. The methodology should be applicable not only to different technologies, but also to different circuit and logic styles. Whenever possible scaling and circuit techniques should be combined with the high-level methodology to further reduce power consumption; however, the general low-power strategy should not *require* these tricks. The advantages of scaling and low-level techniques cannot be overemphasized, but they should not be the sole arena from which the designer can extract power gains.

2.5 Layout Level

There are a number of layout-level techniques that can be applied to reduce power. The simplest of these techniques is to select upper level metals to route high activity signals. The higher level metals are physically separated from the substrate by a greater thickness of silicon

dioxide. Since the physical capacitance of these wires decreases linearly with increasing t_{ox}, there is some advantage to routing the highest activity signals in the higher level metals. For example, in a typical process metal three will have about a 30% lower capacitance per unit area than metal two [Rab95]. The DEC Alpha chip takes advantage of this fact by routing the high activity clock network primarily in third level metal [Dob92]. It should be noted, however, that the technique is most effective for global rather than local routing, since connecting to a higher level metal requires more vias, which add area and capacitance to the circuit. Still, the concept of associating high activity signals with low physical capacitance nodes is an important one and appears in many different contexts in low-power design.

For example, we can combine this notion with the locality theme of Section 2.3 to arrive at a general strategy for low-power placement and routing. The placement and routing problem crops up in many different guises in VLSI design. Place and route can be performed on pads, functional blocks, standard cells, gate arrays, etc. Traditional placement involves minimizing area and delay. Minimizing delay, in turn, translates to minimizing the physical capacitance (or length) of wires. In contrast, placement for power concentrates on minimizing the activity-capacitance product rather than the capacitance alone. In summary, high activity wires should be kept short or, in a manner of speaking, *local*. Tools have been developed that use this basic strategy to achieve about an 18% reduction in power [Cha94][Vai93].

Although intelligent placement and routing of standard cells and gate arrays can help to improve their power efficiency, the locality achieved by low-power place and route tools rarely approaches what can be achieved by a full-custom design. Design-time issues and other economic factors, however, may in many cases preclude the use of full-custom design. In these instances, the concepts presented here regarding low-power placement and routing of standard cells and gate arrays may prove useful. Moreover, even for custom designs, these low-power strategies can be applied to placement and routing at the block level.

2.6 Circuit Level

Many circuit techniques can lead to reduced power consumption. In this section, we go beyond the traditional synchronous fully-complementary static CMOS circuit style to consider the relative advantages and disadvantages of other design strategies. In doing so, it is important to remember that power is not the only measure of merit. In addition, ease of design and issues of robustness must be given equal weight. With this in mind, design styles that at first seem advantageous for low-power, often lose their appeal. This section will consider five topics relating to low-power circuit design: dynamic logic, pass-transistor logic, asynchronous logic, transistor sizing, and design style (e.g. full custom versus standard cell).

2.6.1 Dynamic Logic

In static logic, node voltages are always maintained by a conducting path from the node to one of the supply rails. In contrast, dynamic logic nodes go through periods during which there is no path to the rails, and voltages are maintained as charge dynamically stored on nodal capacitances. Figure 2-8 shows an implementation of a complex boolean expression in both static and dynamic logic. In the dynamic case, the clock period is divided into a precharge and an evaluation phase. During precharge, the output is charged to V_{dd}. Then, during the next clock phase, the NMOS tree evaluates the logic function and discharges the output node if necessary. Relative to static CMOS, dynamic logic has both advantages and disadvantages in terms of power.

Historically, dynamic design styles have been touted for their inherent low-power properties. For example, dynamic design styles often have significantly reduced device counts. Since the logic evaluation function is fulfilled by the NMOS tree alone, the PMOS tree can be replaced by a single precharge device. These reduced device counts result in a corresponding decrease in capacitive loading, which can lead to power savings. Moreover, by avoiding stacked PMOS transistors, dynamic logic is amenable to low voltage operation where the ability to stack devices is limited. In

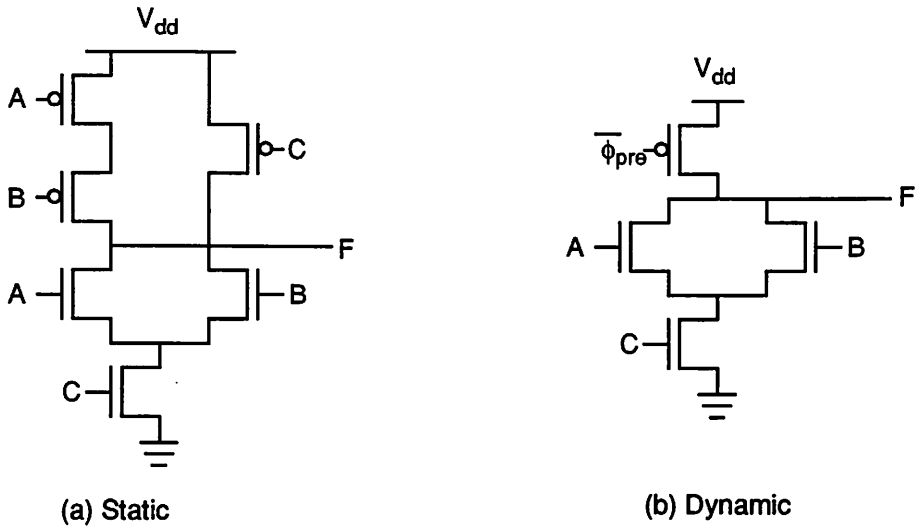


Figure 2-8 : Static and dynamic implementations of $F=\overline{(A+B)C}$.

addition, dynamic gates don't experience short-circuit power dissipation. Whenever static circuits switch, a brief pulse of transient current flows from V_{dd} to ground consuming power. Furthermore, dynamic logic nodes are guaranteed to have a maximum of one transition per clock cycle. Static gates do not follow this pattern and can experience a glitching phenomenon whereby output nodes undergo unwanted transitions before settling at their final value. This causes excess power dissipation in static gates. So in some sense, dynamic logic avoids some of the overhead and waste associated with fully-complementary static logic.

In practice, however, dynamic circuits have several disadvantages. For instance, each of the precharge transistors in the chip must be driven by a clock signal. This implies a dense clock distribution network and its associated capacitance and driving circuitry. These components can contribute significant power consumption to the chip. For example, the clock network of the DEC Alpha chip contributes 3.25 nF of capacitance to the design and is estimated to consume 7W or 23% of the total chip power at 3.3V [Dob92]. In addition, with each gate influenced by the clock, issues of skew become even more important and difficult to handle.

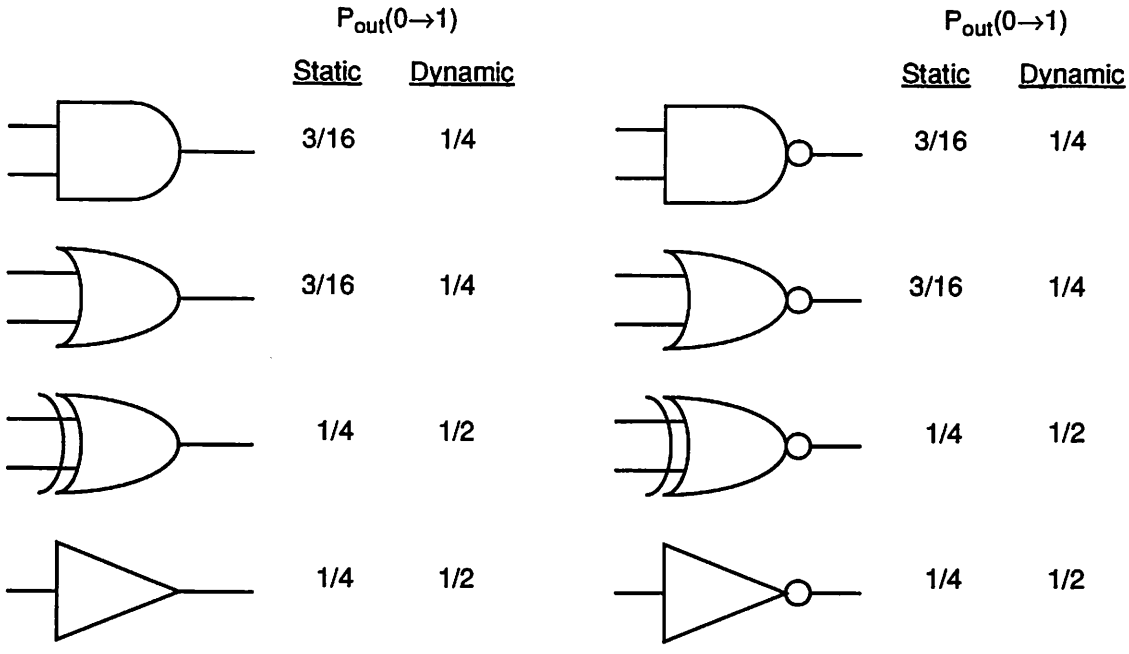


Figure 2-9 : Output activities for static and dynamic logic gates (with random inputs)

Also, the clock is a high (actually, maximum) activity signal, and having it connected to the PMOS pull-up network can introduce unnecessary activity into the circuit. For commonly used boolean logic gates, Figure 2-9 shows the probability that the outputs make an energy consuming (i.e. zero to one) transition for random gate inputs. In all cases, the activity of the dynamic gates is higher than that of the static gates. We can show that, in general, for any boolean signal X, the activity of a dynamically-precharged wire carrying X must always be *at least* as high as the activity of a statically-driven wire:

$$\text{dynamic case: } P_{wire}(0 \rightarrow 1) = P_X(0) \qquad \textbf{(EQ 15)}$$

$$\text{static case: } P_{wire}(0 \rightarrow 1) = P(X_t = 1 | X_{t-1} = 0)\, P_X(0) \leq P_X(0) \qquad \textbf{(EQ 16)}$$

In addition to clock power and difficulty of design, the issue of robustness should not be overlooked. A noise signal of an amplitude as small as V_t is all that would be required to turn on a nominally off transistor and erroneously charge or discharge a dynamic node. Even if the threshold

voltage has not been reduced by using a "low-power process" (Section 2.4.2) it still may only be 0.6-0.7V. A voltage drop of this magnitude on a dynamic node due to charge sharing, subthreshold conduction, or capacitive coupling would not be unexpected and could lead to faulty operation [Wes88].

In conclusion, dynamic logic has certain advantages and disadvantages for low-power operation. The key is to determine which of the conflicting factors is dominant. In certain cases, a dynamic implementation might actually achieve a lower overall power consumption. For example, a PLA implemented in fully-complementary logic would require huge PMOS trees to realize both the input and output planes. Dynamic PLA's eliminate this large source of capacitance and can operate at higher speeds and lower power. So in some specific cases, dynamic logic is, clearly, the superior alternative. In most cases, however, the transistor savings are not nearly as dramatic. Indeed, to achieve low voltage operation, designers of static logic are increasingly led to reduce stacking of PMOS transistors simply to achieve reasonable rise times. As a result, there are fewer gates with the complex PMOS trees for which dynamic logic is advantageous. So the power required to switch the high activity clock network in dynamic circuits typically outweighs any savings from reduced device counts. Furthermore, the savings in terms of glitching and short-circuit power, while possibly significant, can also be achieved in static logic through other means (discussed in Section 2.7). These arguments are supported by Lee in [Lee94] where he presents a conventional static CMOS adder that is 47% more energy efficient than an equivalent dynamic implementation. All of this, coupled with the robustness of static logic at low voltages gives the designer less incentive to select a dynamic implementation of a low-power system.

2.6.2 Pass-Transistor Logic

As with dynamic logic, pass-transistor logic offers the possibility of reduced transistor counts. Figure 2-10 illustrates this fact with an equivalent pass-transistor implementation of the static

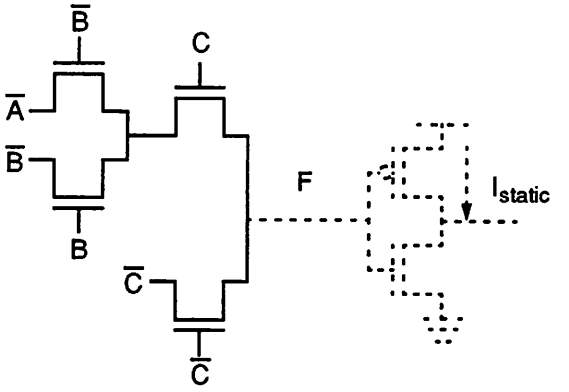


Figure 2-10 : Complementary pass-transistor implementation of $F=\overline{(A+B)C}$.

logic function of Figure 2-8. Once again, the reduction in transistors results in lower capacitive loading from devices. This might make pass-transistor logic attractive as a low-power circuit style.

Like dynamic logic, however, pass-transistor circuits suffer from several drawbacks. First, pass transistors have asymmetrical voltage driving capabilities. For example, NMOS transistors do not pass "high" voltages efficiently, and experience reduced current drive as well as a V_t drop at the output. If the output is used to drive a PMOS gate, static power dissipation can result. These flaws can be remedied by using additional hardware - for instance, complementary transmission gates consisting of an NMOS and PMOS pass transistors in parallel. Unfortunately, this forfeits the power savings offered by reduced device counts. Also, efficient layout of pass-transistor networks can be problematic. Sharing of source/drain diffusion regions is often not possible, resulting in increased parasitic junction capacitances.

In summary, at the voltages attractive for low-power design, the reduced current drive of pass-transistor logic networks becomes particularly troublesome. Low threshold processes can lessen this problem, but it is at the expense of robustness and static power dissipation. Full transmission gates can improve the situation, but require true and complement control signals, and also eliminate the benefit of reduced transistor count. As with dynamic logic, there may be situations in

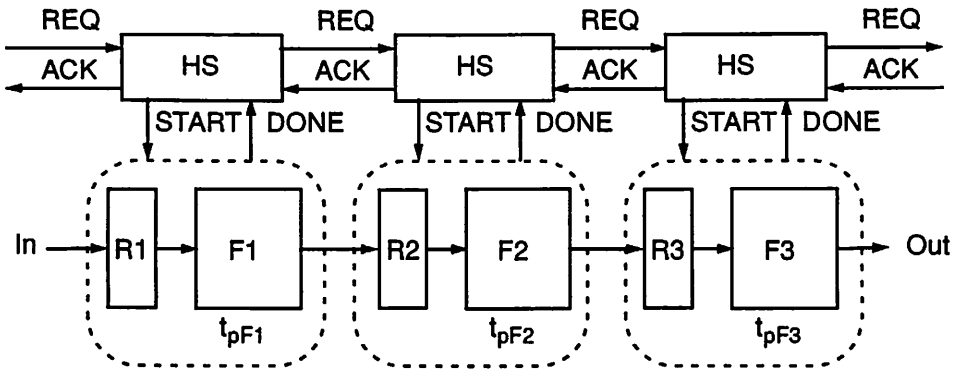


Figure 2-11 : Asynchronous circuit with handshaking (after [Rab95])

which pass-transistor logic is more power efficient than fully-complementary logic; however, the benefits are likely to be small relative to the orders of magnitude savings possible from higher level techniques. Indeed, in a comparison of a fully-complementary CMOS adder to a transmission gate adder, Lee reported that the pass-gate version was only 10% more energy efficient than the conventional adder [Lee94]. So, again, circuit-level power saving techniques should be used whenever appropriate, but should be subordinate to higher level considerations.

2.6.3 Asynchronous Logic

Asynchronous logic refers to a circuit style employing no global clock signal for synchronization. Instead, synchronization is provided by handshaking circuitry used as an interface between gates (see Figure 2-11). While more common at the system level, asynchronous logic has failed to gain acceptance at the circuit level. This has been based on area and performance criteria. It is worthwhile to reevaluate asynchronous circuits in the context of low power.

The primary power advantages of asynchronous logic can be classified as avoiding waste. The clock signal in synchronous logic contains no information; therefore, power associated with the clock driver and distribution network is in some sense wasted. Avoiding this power consumption

component might offer significant benefits. In addition, asynchronous logic uses completion signals, thereby avoiding glitching, another form of wasted power. Finally, with no clock signal and with computation triggered by the presence of new data, asynchronous logic contains a sort of built in power-down mechanism for idle periods.

While asynchronous sounds like the ideal low-power design style, several issues impede its acceptance in low-power arenas. Depending on the size of its associated logic structure, the overhead of the handshake interface and completion signal generation circuitry can be large in terms of both area and power. Since this circuitry does not contribute to the actual computations, transitions on handshake signals are wasted. This is similar to the waste due to clock power consumption, though it is not as severe since handshake signals have lower activity than clocks. Finally, fewer design tools support asynchronous than synchronous, making it more difficult to design.

At the small granularity with which it is commonly implemented, the overhead of the asynchronous interface circuitry dominates over the power saving attributes of the design style. It should be emphasized, however, that this is mainly a function of the granularity of the handshaking circuitry. It would certainly be worthwhile to consider using asynchronous techniques to eliminate the necessity of distributing a global clock between blocks of larger granularity. For example, large modules could operate synchronously off local clocks, but communicate globally using asynchronous interfaces. In this way, the interface circuitry would represent a very small overhead component, and the most power consuming aspects of synchronous circuitry (i.e. global clock distribution) would be avoided.

2.6.4 Transistor Sizing

Regardless of the circuit style employed, the issue of transistor sizing for low power arises. The primary trade-off involved is between performance and cost - where cost is measured by area and power. Transistors with larger gate widths provide more current drive than smaller transistors.

Unfortunately, they also contribute more device capacitance to the circuit and, consequently, result in higher power dissipation. Moreover, larger devices experience more severe short-circuit currents, which should be avoided whenever possible. In addition, if all devices in a circuit are sized up, then the loading capacitance increases in the same proportion as the current drive, resulting in little performance improvement beyond the point of overcoming fixed parasitic capacitance components. In this sense, large transistors become *self-loading* and the benefit of large devices must be reevaluated. A sensible low-power strategy is to use minimum size devices whenever possible. Along the critical path, however, devices should be sized up to overcome parasitics and meet performance requirements. Care should be taken in this sizing process to avoid the waste of self-loading [Bur94]. By following this approach, Nagendra et al. found that the average power dissipation of a signed-digit adder could be reduced by 36% with a delay penalty of only 0.3% [Nag94].

2.6.5 Design Style

Another decision which can have a large impact on the overall chip power consumption is selection of design style: e.g. full custom, gate array, standard cell, etc. Not surprisingly, full-custom design offers the best possibility of minimizing power consumption. In a custom design, all the principles of low-power including locality, regularity, and sizing can be applied optimally to individual circuits. Unfortunately, this is a costly alternative in terms of design time, and can rarely be employed exclusively as a design strategy. Other possible design styles include gate arrays and standard cells.

Gate arrays offer one alternative for reducing design cycles at the expense of area, power, and performance. While not offering the flexibility of full-custom design, gate-array CAD tools could nevertheless be altered to place increased emphasis on power. For example, gate arrays offer some control over transistor sizing through the use of parallel transistor connections. Employing the sizing strategy of Section 2.6.4 might offer some improvement in gate array power over existing

solutions. Furthermore, since most of the flexibility in gate array design lies in the placement and routing of gates, the techniques of Section 2.5 might prove useful - in particular, partitioning of the circuit to maximize locality coupled with minimizing the length of high activity wires.

Standard cell synthesis is another commonly employed strategy for reducing design time. Current standard cell libraries and tools, however, offer little hope of achieving low power operation. In many ways, standard cells represent the antithesis of a low-power methodology. First and foremost, standard cells are often severely oversized. Most standard cell libraries were designed for maximum performance and worst-case loading from inter-cell routing. As a result, they experience significant self-loading and waste correspondingly significant amounts of power. To overcome this difficulty, standard cell libraries must be expanded to include a selection of cells of identical functionality, but varying driving strengths. With this in place, synthesis tools could select the smallest (and lowest power cell) required to meet timing constraints, while avoiding the wasted power associated with oversized transistors [Bah94]. In addition, the standard cell layout style with its islands of devices and extensive routing channels tends to violate the principles of locality central to low-power design. Here, as in the gate array case, the place and route techniques of Section 2.5 could go a long way towards alleviating this difficulty.

2.6.6 Circuit-Level Conclusions

Clearly, numerous circuit-level techniques are available to the low-power designer. These techniques include careful selection of a circuit style: static vs. dynamic, synchronous vs. asynchronous, fully-complementary vs. pass-transistor, etc. Other techniques involve transistor sizing or selection of a design methodology such as full-custom or standard cell. Some of these techniques can be applied in conjunction with higher level power reduction techniques. When possible, designers should take advantage of this fact and exploit both low and high-level techniques in concert. Often, however, circuit-level techniques will conflict with the low-power strategies based on higher abstraction levels. In these cases, the designer must determine, which

techniques offer the largest power reductions. As evidenced by the previous discussion, circuit-level techniques typically offer reductions of a factor of two or less, while some higher level strategies with their more global impact can produce savings of an order of magnitude or more. In such situations, considerations imposed by the higher-level technique should dominate and the designer should employ those circuit-level methodologies most amenable to the selected high-level strategy.

2.7 Gate Level

This philosophy should also apply to the gate level. Still, as in the case of the circuit level, there are gate-level techniques that can be applied successfully to reduce power consumption. Once again these techniques reflect the themes of trading performance and area for power, avoiding waste, and exploiting locality. In this section we discuss a number of gate-level techniques and give some quantitative indication of their impact on power. In particular, this section presents techniques for technology mapping, glitching and activity reduction, as well as methods for exploiting concurrency and redundancy in the context of low-power design.

2.7.1 Technology Decomposition and Mapping

Technology decomposition and mapping refers to the process of transforming a gate-level boolean description of a logic network into a CMOS circuit. For a given gate-level network there may be many possible circuit-level implementations. For instance, a three-input NAND can be implemented as a single complex CMOS gate or as a cascade of simpler two-input gates. Each mapping may result in different signal activities, as well as physical capacitances. For example, complex gates tend to exhibit an overall lower physical capacitance since more signals are confined to internal nodes rather than to the more heavily loaded output nodes. The concept of technology mapping for low-power is to first decompose the boolean network such that switching activity is minimized, and then to hide any high activity nodes inside complex CMOS gates. In this

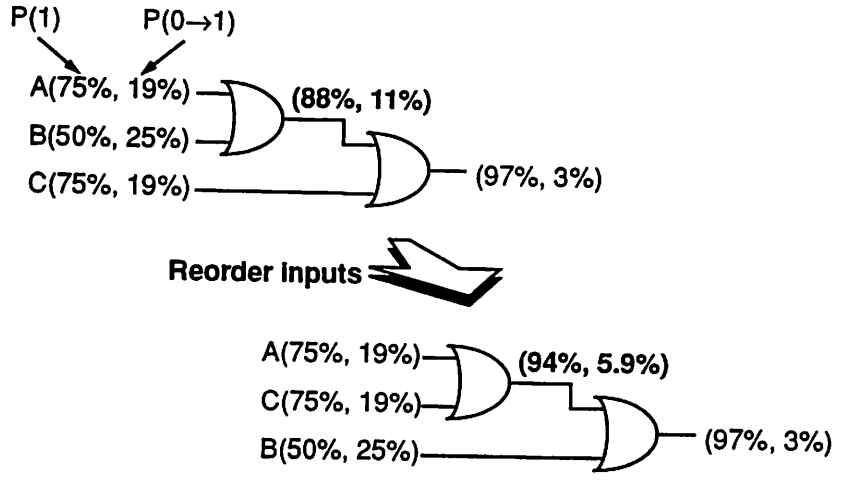


Figure 2-12 : Input reordering for activity reduction

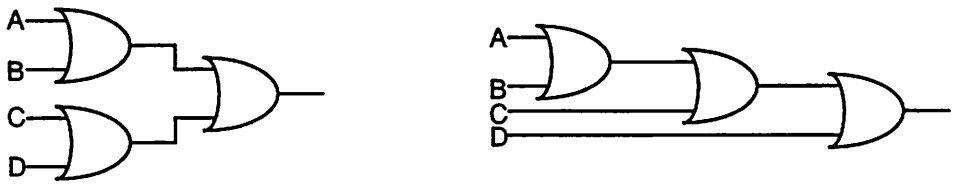
way, rapidly switching signals are mapped to the low capacitance internal nodes, thereby reducing power consumption. Making a gate too complex, however, can slow the circuit, resulting in a trade-off of performance for power. Several technology mapping algorithms for low power have been developed and offer an average power reduction of 12% [Tiw93] to 21% [Tsu94].

2.7.2 Activity Postponement

While technology mapping attempts to minimize the activity-capacitance product, other gate-level strategies focus on reducing activity alone. For example, an operation as simple as reordering the inputs to a boolean network can in some cases reduce the total network activity (see Figure 2-12) [Len93]. The basic concept is to postpone introduction of high activity signals as long as possible. In this way, the fewest gates are affected by the rapidly switching signals.

2.7.3 Glitch Reduction

Other gate-level activity reduction techniques focus on avoiding the wasted transitions associated with glitching. Figure 2-13 shows two implementations of the same logic function. One implementation employs a balanced tree structure, while the other uses a cascaded gate structure.



(a) Balanced tree structure (b) Cascaded structure

Figure 2-13 : Cascaded versus balanced tree gate structures

If we assume equal input arrival times and gate delays, we find that the cascaded structure undergoes many more transitions than the tree structure before settling at its steady-state value. In particular, the arrival of the inputs may trigger a transition at the output of each of the gates. These output transitions may in turn trigger additional transitions for the gates within their fan-out. This reasoning leads to an upper-bound on glitching that is $O(N^2)$, where N is the depth of the logic network. In contrast, the path delays in the tree structure are all balanced, and therefore, each node makes a single transition and no power is wasted. This concept can be extended to derive optimum tree structures for the case of unequal arrival times as well [Mur94]. Some studies have suggested that eliminating glitching in static circuits could reduce power consumption by as much as 15-20% [Ben94].

Aside from the use of tree structures, there are other techniques for balancing path delays and thereby reducing glitching. For example, gates can be sized down to increase the delay along fast paths. Using this approach, Bahar et al. were able to achieve an average power reduction of about 28% [Bah94]. In addition to gate sizing, buffer insertion can be used to balance delays along all reconverging paths as is done in wave pipelining [Won89]; however, the balancing requirements are not as strict in the low-power case since only paths with a high probability of transition activity need be balanced [Van92].These examples demonstrate two forms of the same glitch reduction strategy - path balancing.

There are other techniques that can be employed to achieve the same end. Recall that the amount of glitching in a circuit is related quadratically to the depth of the associated logic network. As a result, techniques which reduce logic depth can lead to reduced glitching activity. For the case of a 16x16 Booth array multiplier, Lemonds and Shetti were able to reduce the power consumption approximately 40% by inserting self-timed latches between the booth encoders and the full-adder array [Lem94]. In addition, Monteiro et al. proposed an algorithm that automatically retimes sequential logic, inserting flip-flops to minimize glitching activity [Mon93]. Using this technique, he demonstrated that retiming for power can result in up to 16% less power consumption than retiming for delay alone.

2.7.4 Concurrency and Redundancy

The final technique discussed in this section is the use of concurrency and redundancy at the gate level in low-power design. The principal concept is to use concurrency and redundancy to improve performance and then to trade this performance for lower power by reducing the voltage supply. In some sense, path balancing, which was found to be useful for reducing glitching activity, can be thought of as a form of gate-level concurrent processing. Referring to Figure 2-13, the path balanced tree structure is characterized by several logic gates computing in parallel, with the gates in their fan-out combining the results. In contrast, for the linear, cascaded structure computations must take place sequentially since the results from the initial stages are needed to compute the output of the later stages. So by using concurrency, the tree structure achieves a shorter critical path than the cascaded structure - quantitatively, logarithmic as opposed to linear. This reduced critical path can be used to improve performance, or this performance can be traded for power by reducing the operating voltage until the delay of the logarithmic structure matches that of the linear structure.

Another instance of the use of concurrency to reduce power is exemplified by a comparison of ripple-carry and carry-select adders. The carry-select adder is subdivided into several stages and

employs a dual carry chain in each stage: a zero carry chain and a one carry chain. In other words, each stage computes results for both possible carry inputs. Then, when the actual carry input arrives, the only delay is incurred by selecting the proper precomputed result. At fixed voltage, the typical performance achieved by the carry-select adder is $O(logN)$ or $O(\sqrt{N})$, whereas the ripple-carry delay is $O(N)$. As before, we can trade this increased performance for reduced power by operating at a lower voltage.

The carry-lookahead adder uses concurrency as well as redundancy to achieve gains in performance or, if desired, power. Rather than waiting for the carry to ripple from the lower to the higher bits, bits in the higher stages actually precompute the expected carry based on the primary inputs. Thus, there is some redundancy inherent in the computations of the carry-lookahead adder. However, this redundancy allows the structure to achieve a logarithmic performance with word length. As always, this performance can be used to implement a high-speed adder at fixed voltage or a low-power adder at reduced voltage.

The majority of the techniques employing concurrency or redundancy incur an inherent penalty in area, as well as in physical capacitance and switching activity. At first glance, a carry-select adder with 50% more physical capacitance and activity than a ripple-carry adder might not seem low power at all. The key concept is to identify the design paradigm under which you are working: fixed voltage or variable voltage. If the voltage is allowed the vary, then it is typically worthwhile to sacrifice increased physical capacitance and activity for the quadratic power improvement offered by reduced voltage. If, however, the system voltage has been fixed, then there is nothing gained by employing a carry-select adder in place of a ripple-carry adder, unless the slower adder does not meet the timing constraints. So in this situation its better to use the least complex adder that meets your performance requirements. This falls under the category of avoiding waste. Nagendra et al. performed some fairly extensive comparisons of the delay and power attributes of various adder types [Nag94]. Some of the results of their survey for a 5V supply and a 1.2 μm CMOS technology are summarized in Table 2-1.

Adder Type	16 bit			32 bit		
	Delay (ns)	Power (μW)	PDP (pJ)	Delay (ns)	Power (μW)	PDP (pJ)
Ripple carry	30.59	49.0	1.50	63.41	98.4	6.25
Carry skip	25.36	60.9	1.54	38.75	121.2	4.70
Carry select	16.63	138.6	2.30	27.75	296.6	8.23
Brent and Kung	22.31	141.1	3.15	36.90	274.9	10.14
Elm	21.64	108.3	2.34	33.10	239.5	7.93
Signed digit	12.60	160.1	2.02	12.60	326.7	4.12

Table 2-1 : Worst-case delay and power dissipation for various adders (after [Nag94])

2.7.5 Gate-Level Conclusions

In summary, this section has discussed several gate-level techniques for power optimization. The main concepts explored were reducing the waste associated with unnecessary activity and trading power for performance through low-voltage concurrent processing. The gains reported by low-power designers working at the gate level are typically on the order of a factor of two or less. So while they should be exploited whenever possible, this should not be done at the expense of the larger gains achievable at the architecture and algorithm levels, which will be discussed in the following sections.

2.8 Architecture and System Levels

This chapter has repeatedly suggested that decisions made at a high level (architecture or system) will have a much larger impact on power than those made at a lower level (e.g. gate or

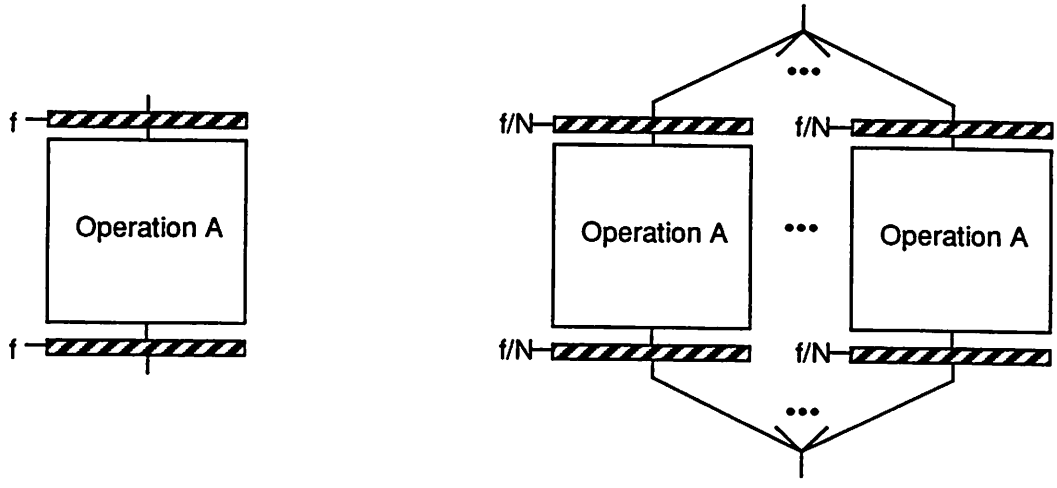
circuit). This section provides some evidence to support this claim. In the terminology of this thesis, *architecture* refers to the register-transfer (RT) level of abstraction, where the primitives are blocks such as multipliers, adders, memories, and controllers. Computer scientists often refer to this level of abstraction as the micro-architecture level. Having defined the terminology, this section discusses *architectural* or RT-level techniques for reducing power consumption. These techniques utilize the themes of avoiding waste, exploiting locality, and trading area/performance for power. Specifically, they include concurrent processing, power management, partitioning, selecting dedicated vs. programmable hardware, and selecting a data representation.

2.8.1 Concurrent Processing

Perhaps the most important strategy for reducing power consumption involves employing concurrent processing at the architecture level. This is a direct trade-off of area and performance for power. In other words, the designer applies some well-known technique for improving performance such as parallelism or pipelining, and then swaps this higher performance for lower power by reducing the supply voltage. There are, of course, limits to the benefits offered by concurrent processing. For instance, the interconnect overhead associated with massively parallel systems at some point outweighs the savings offered by reduced voltage. Still, the technique can be extremely useful as long as the proper balance is maintained.

Parallelism

As a quantitative example, consider the use of parallelism to perform some complex operation, A (see Figure 2-14a). The registers supplying operands and storing results for A are clocked at a frequency f. Further assume that algorithmic and data dependency constraints do not prevent concurrency in the calculations performed by A. When the computation of A is parallelized, Figure 2-14b results. The hardware comprising block A has been duplicated N times resulting in N identical processors. Since there are now N processors, a throughput equal to that of sequential processor, A, can be maintained with a clocking frequency N times lower than that of A. That is,



(a) Reference:
Capacitance = C
Frequency = f
Voltage = V
Power = $P \propto CV^2f$

(b) Parallel:
Capacitance ~ NC
Frequency = f/N
Voltage ~ V/N (neglecting V_t)
Power ~ P/N^2

Figure 2-14 : Voltage scaling and parallelism for low power

although each block will produce a result only 1/*N*th as frequently as processor *A*, there are *N* such processors producing results. Consequently, identical throughput is maintained.

The key to this architecture's utility as a power saving configuration lies in this factor of *N* reduction in clocking frequency. In particular, with a clocking frequency of *f*/*N*, each individual processor can run *N* times slower. Since to the first order, delays vary roughly linearly with voltage supply, this corresponds to a possible factor of *N* reduction in supply voltage. Examining the power consumption relative to the single processor configuration, we see that capacitances have increased by a factor of *N* (due to hardware duplication), while frequency and supply voltage have been reduced by the same factor. Thus, since $P \propto CV^2f$, power consumption is reduced by the square of the concurrency factor, *N*:

N-way concurrency: $P \propto 1/N^2$ **(EQ 17)**

The preceding analysis has considered only the first-order effects of concurrency on power consumption. There are second-order effects that detract from the power savings that can be achieved by using concurrent hardware. First, the discussion assumed that introducing N parallel processors would allow a factor of N speedup in performance. Often, the algorithm being executed on these processors does not contain this much concurrency. This is a problem that has been faced for many years by programmers writing compilers for multi-processor systems. They have developed several techniques for increasing the parallelism available in a given algorithm and some of these will be presented in the section on low-power algorithms. In any case, this issue tends to place an upper bound on the amount of parallelism that can be effectively exploited for power reduction.

Another important second-order consideration is the effect of finite threshold voltage on voltage scaling. As discussed in Section 2.2.1, performance degrades linearly with voltage reductions only if $V_{dd} \gg V_t$. As the supply voltage approaches the threshold voltage, the degradation is much more pronounced. As a result, a factor of N voltage reduction may actually require more than N-way parallelism. For various values of V_t, Figure 2-15 shows the *actual* power reduction that can be achieved by scaling the supply voltage down from 5V and introducing concurrency. The figure demonstrates that large values of V_t relative to V_{dd} can lessen the ability of voltage scaling to reduce power. This provides an excellent motivation for low-V_t processes.

This strategy of low-voltage parallel processing was applied to the design of a programmable color space converter as shown in Figure 2-16 [Lan90]. The processor can be used to convert a pixel stream arriving at 10Mhz from one color space (e.g. RGB) to another (e.g. YUV). The computation is basically a 3x3 matrix multiplication. In this implementation, the nine required multiplications were carried out on three parallel multipliers rather than on the single multiplier that traditional implementations employ. This parallelism allowed the chip to operate at 2.5V. The voltage reduction along with other low-power features incorporated into the design resulted in an implementation that consumed 5.5x less power than an equivalent commercial component (the

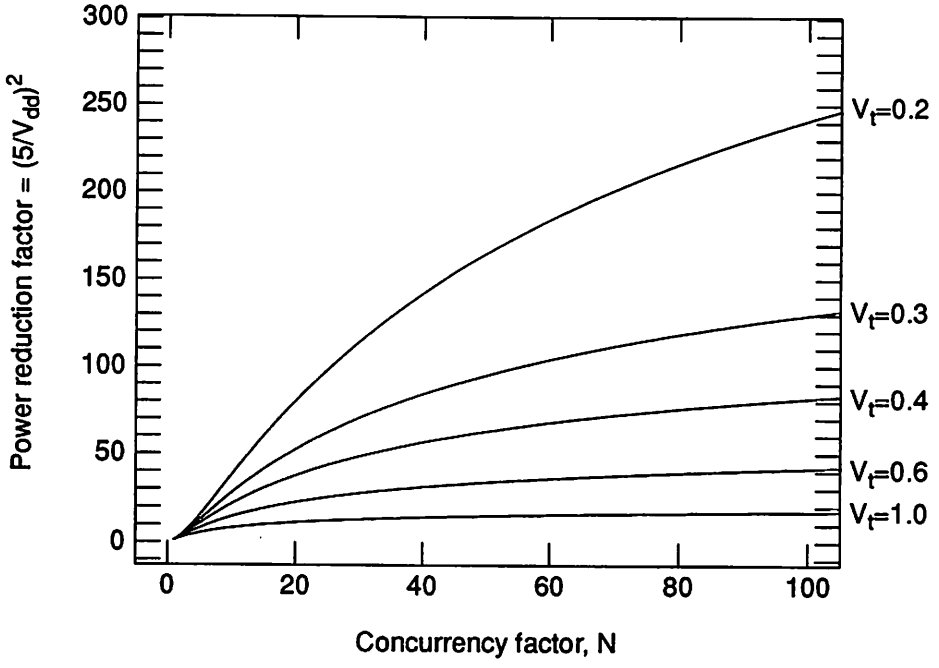


Figure 2-15 : Power reduction from concurrency and voltage scaling for various V_t's

Bt281).

Hardware parallelism also has its disadvantages. For instance, complete hardware duplication entails a severe area penalty. In addition, there is hardware and interconnect overhead related to signal distribution at the processor inputs and signal merging at the outputs. These contribute to increased power consumption and tend to limit the utility of excessive parallelism. Also, the area requirements of full parallelism can be a limiting factor; however, the advent of MCM's should help to minimize this concern. Still, other forms of concurrent processing offer some of the power savings of parallelism at reduced cost.

Pipelining

Pipelining is another form of concurrent computation that can be exploited for power reduction. An example of pipelining for low power is shown in Figure 2-17b. In this situation,

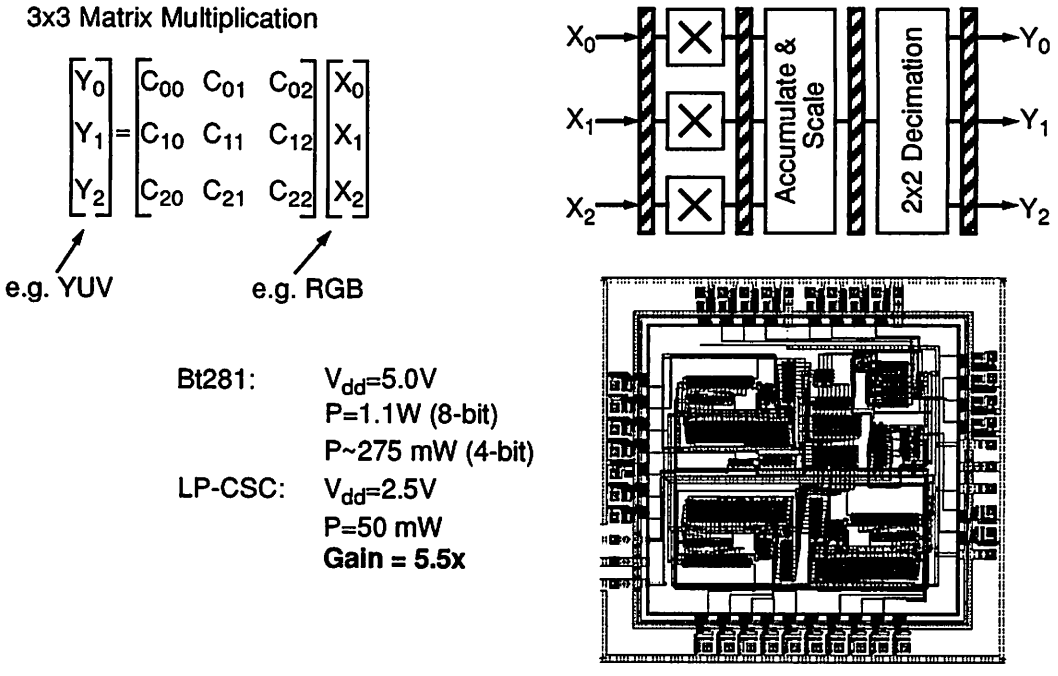


Figure 2-16 : Low-power programmable color space converter

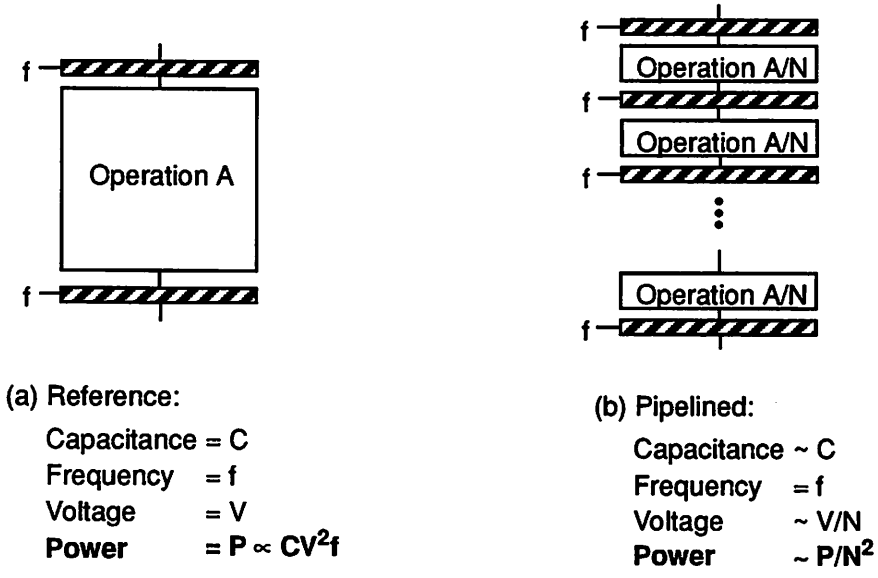


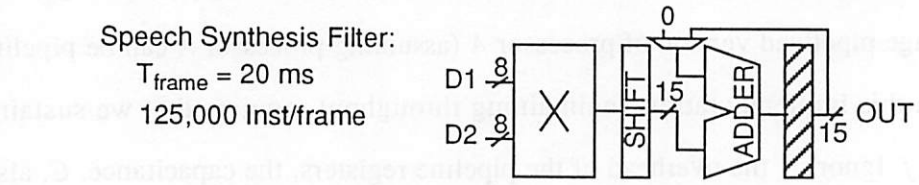
Figure 2-17 : Voltage scaling and pipelining for low power

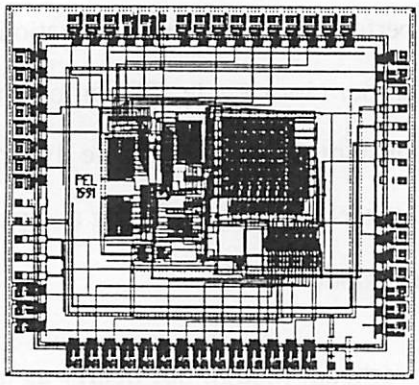
rather than duplicating hardware, concurrency is achieved by inserting pipeline registers, resulting in an N-stage pipelined version of processor A (assuming processor A can be pipelined to this extent). In this implementation, maintaining throughput requires that we sustain clocking frequency, f. Ignoring the overhead of the pipeline registers, the capacitance, C, also remains constant. The advantage of this configuration is derived from the greatly reduced computational requirements between pipeline registers. Rather than performing the entire computation, A, within one clock cycle, only $1/N$th of A need be calculated per clock cycle. This allows a factor N reduction in supply voltage and, considering the constant C and f terms, the dynamic power consumption is reduced by N^2. Thus, both concurrency techniques - pipelining and parallelism - result in a first-order quadratic reduction in power consumption.

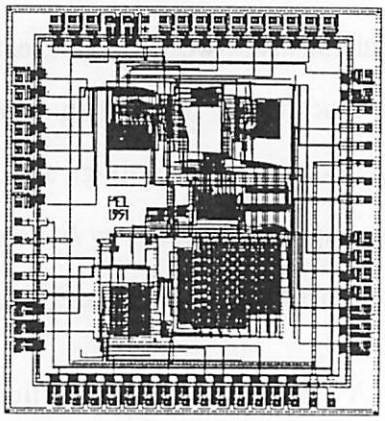
As with parallelism, pipelining incurs some overhead, though not nearly as much. In a pipelined processor, for example, the pipeline registers represent an overhead in both power and area. First, each register must be clocked, adding to the capacitive loading of the clock network. As the pipeline depth increases, the area and capacitance associated with the pipeline registers approaches that of the actual processor stages. At that point, further pipelining becomes unattractive. Still, the overhead associated with pipelining is typically much less than that of parallelism, making it a useful form of concurrent processing for power minimization.

As a demonstration of pipelining to reduce power consumption, two versions of a speech synthesis filter as might be found in a speech coder or synthesizer were implemented (see Figure 2-18). Both employ the same basic multiply-accumulate datapath as shown in the figure. The first was a reference version operating at 5V and containing no pipelining. The second version was pipelined, allowing it to operate at 2.3V. The pipelined version realized a power reduction of 4.2x. Analysis shows that a low-V_t process would allow further power reduction to an asymptotic limit of 12.3x for a zero-V_t process.

Still, pipelining has some of the same limitations as parallelism since not all algorithms are







Reference:
f_{clk} = 6.3 MHz
V_{dd} = 5.0 V
P = 16.7 mW

Pipelined:
f_{clk} = 6.3 MHz
V_{dd} = 2.3 V
P = 3.98 mW
Gain = 4.2x

V_t=0
V_{dd} = 1.4 V
P = 1.36 mW
Gain = 12.3x

Figure 2-18 : Sequential and pipelined implementations of a speech synthesis filter

amenable to pipelining. In particular, the efficiency of implementing certain algorithms on a pipelined processor suffers from so-called *pipeline hazards*. These hazards stem from the fact that in a deeply pipelined processor, the prefetching and execution of an instruction begins before the previous instruction(s) has completed.

The two main types of hazards are branching hazards and data hazards. The former occurs when, while prefetching the next instruction, the wrong assumption is made about whether a branch will be taken. If this occurs, the prefetched instruction(s) must be discarded, and prefetching must begin again at the correct instruction. A data hazard refers to an attempt to utilize the result of a previous computation before that calculation has completed. Both of these types of

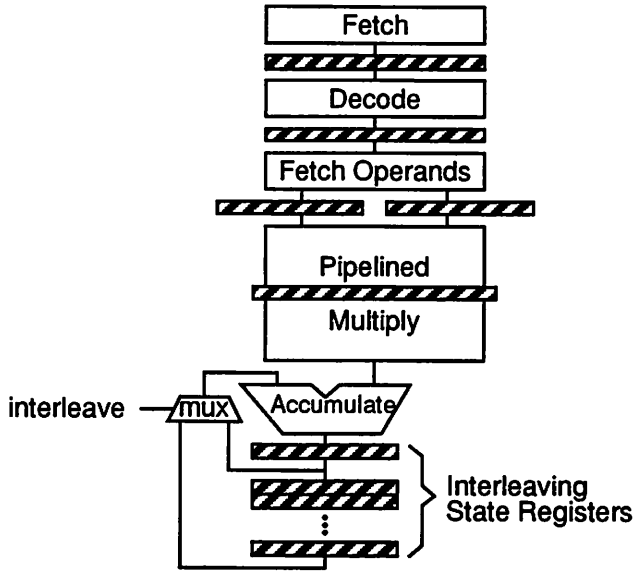


Figure 2-19 : Pipelined-interleaved processor architecture (after [Lee86])

hazards can impede attempts to efficiently program a pipelined processor.

A possible solution to this difficulty involves interleaving the execution of multiple programs within a single pipeline [Lee86]. As Lee demonstrates, under certain conditions, this pipelined-interleaved (PI or π) processor behaves as N parallel processors sharing a single memory without contention. Therefore, the π processor avoids the programming difficulties associated with pipelined processors while occupying only slightly more area (additional area is required to store the states of the virtual processors). Figure 2-19 shows a high-level diagram of Lee's π processor architecture. The architecture of this figure contains one refinement over Lee's implementation. That refinement is the ability to switch off the interleaving mechanism when desired, turning the processor into a deeply pipelined, but not interleaved, processor. This capability is manifested in the architecture by multiplexers, which allow the user to bypass the interleaving state registers associated with the accumulator, the indexing registers, and the program counter. In summary, pipelined-interleaved architectures combine desirable properties of both the parallel and pipelined styles of concurrent processing and might represent an attractive alternative for a low-power

programmable processor.

Concurrent vs. Time-Multiplexed Architectures

In concurrent architectures, several computations are performed simultaneously using distinct, parallel hardware units. In contrast, a time-multiplexed architecture performs required operations sequentially on a single hardware unit over a large number of clock cycles. Bit-serial processing is one example of a time-multiplexed design style. For instance, a bit-serial adder consists of a one-bit full adder cell that performs an N-bit addition one bit at a time over N clock cycles.

This form of processing has few advantages for low-power circuits. Consider, for example, a function, B, that operates on a word of N bits. In bit-serial form, this function would be implemented by a single-bit processor, which would possess $1/N$th the capacitance of the bit-parallel case. Unfortunately, in order to maintain throughput, the bit-serial processor must be clocked at a frequency N times higher than the bit-parallel case. As in the pipelined and parallel scenarios, then, the effects of the capacitive and frequency components in the dynamic power equation cancel out. Unlike, those cases, however, the bit-serial processor cannot operate at a reduced voltage supply. That is, due to the increased clocking frequency, the relative time allotted for the calculation of a single result bit has not increased over the bit-parallel case; therefore, no V_{dd} reductions and, subsequently, no power reductions are possible. Clearly then, it is time-*concurrent* rather than time-*multiplexed* architectures that are attractive for low-power systems.

2.8.2 Power Management

Any power consumed by a system that does not lead to useful results is wasted. For previous generations of chips where power was not a major concern this source of waste was typically ignored. Designers of modern, low-power systems, however, must give serious consideration to strategies for reducing the power wasted in their systems. Strategies for avoiding wasted power in systems are often called *power management* strategies. Some power management techniques

available to designers include selective power down, sleep mode, and adaptive clocking/voltage schemes.

Selective power down refers to deactivating processing modules that are not doing any useful work. A selective power down strategy requires additional logic to monitor the activity of the various modules within a system. Control lines signalling idle periods are then used to gate the clocks to the various system modules. As a result, idle modules are not clocked and no power is wasted. Some care must be taken in applying this strategy to dynamic circuits for which clocking is used to maintain information stored on dynamic nodes. In these cases it is more sensible to slow rather than stop the clock. In this way, power is reduced, but dynamic nodes are still periodically refreshed. As a side note, this power-down strategy applies only to synchronous systems. As mentioned in Section 2.6.3, asynchronous circuits provide this power-down functionality automatically.

Sleep mode is an extension of the selective power-down strategy. Here, the activity of the entire system is monitored rather than that of the individual modules. If the system has been idle for some predetermined time-out duration, then the entire system is shut down and enters what is known as *sleep mode*. During sleep mode the system inputs are monitored for activity, which will then trigger the system to wake up and resume processing. Since there is some overhead in time and power associated with entering and leaving sleep mode, there are some trade-offs to be made in setting the length of the desired time-out period.

Selective clocking and sleep mode are becoming quite popular in the commercial microprocessor arena. At ISSCC '94, Intel, MIPS Technologies, and IBM all reported microprocessors that include selective power down and sleep-mode capabilities [Sch94][Yeu94] [Bec94][Pha94]. IBM estimated that selective clocking saved 12-30% in the power consumption of their 80MHz superscalar PowerPC architecture [Pha94]. In addition, Intel estimated that the combination of both techniques resulted in an overall 2x reduction in average power when running

typical applications on their design [Sch94].

Another power management strategy involves adapting clocking frequency and/or supply voltage to meet, rather than beat, system performance requirements. Since the performance requirements of a system typically vary over time as the task it is performing changes, it is wasteful to run the system at maximum performance even when a minimum of compute power is required. Adapting the clocking frequency or supply voltage of the system to reduce performance (and power) during these periods can result in substantial power savings. Since it is difficult to directly measure how much performance is actually required at a given point in time, some indirect performance feedback scheme must be devised. This can be in the form of clock slow-down instruction issued by the software application. MIPS Technologies takes this approach in their 64b RISC processor, achieving a 4x reduction in power through reduced clock frequency [Yeu94]. Alternatively, adaptation can be performed automatically through some built-in feedback mechanism. Nielson and Sparso took this approach in designing a DSP processor with an adaptive voltage supply that reduced average power by 1.4-10x [Nie94]. The system contains FIFO's at the inputs and outputs and monitors dynamic changes in the length of the queues to determine how the system voltage should be scaled. As queues began to fill, the voltage is increased to boost performance. In contrast, nearly empty queues trigger a reduction in voltage (and power) accompanied by a corresponding slackening off of performance.

To summarize, without careful management, large amounts of system power can be wasted by continuing computations during idle periods. A power-conscious system will avoid this source of waste either by powering down inactive modules or processors or by adapting the processing power of the system to meet, rather than exceed, the current requirements. Strategies such as these have been applied to many existing low-power systems often leading to power savings up to 10x with no loss of functionality or perceived performance.

2.8.3 Partitioning

As mentioned previously, locality is a recurring theme in low-power design. Global data transfers, references to large global memories, and driving global signals across the chip are all power consuming tasks, and can often be avoided with architectural and system design techniques. Partitioning is one powerful tool for maximizing locality. This section considers partitioning of processors, memories, and controllers with the objective of reducing power consumption.

A distributed processing strategy can often operate with more power-efficiency than a single, centralized general-purpose processor. For instance, the power associated with global data transfers is substantially reduced. In addition, local processors can be optimized for a particular subsection of the computational graph. In contrast, a uni-processor must have the generality required to execute all parts of the algorithm. This generality leads to overhead and inefficiency in terms of performance and power. Use of a partitioned processing strategy relies on the assumption that the algorithm being executed can also be partitioned effectively. This issue is addressed in Section 2.9.2.

In considering the question of distributed versus centralized processing, one must also address the issue of memory partitioning. As with processing hardware, a distributed array of small local memories is often more power efficient than a shared, global memory subsystem. In particular, the energy consumed in accessing a memory is approximately proportional to the number of words stored in that memory. If this number can be reduced by partitioning the memory, then the total power associated with memory accesses will also be reduced. For example, assume 10 independent processors need to access one piece of data each. A single, shared memory will need to contain 10 words and each access will take 10 energy units for a total of 100 units of energy. On the other hand, if each processor utilizes a local memory containing the one piece of data it requires, then each of the 10 accesses will consume only one unit of energy for a total of 10 units - a factor of 10 savings in power. This example, though idealized, suggests the advantage of local or

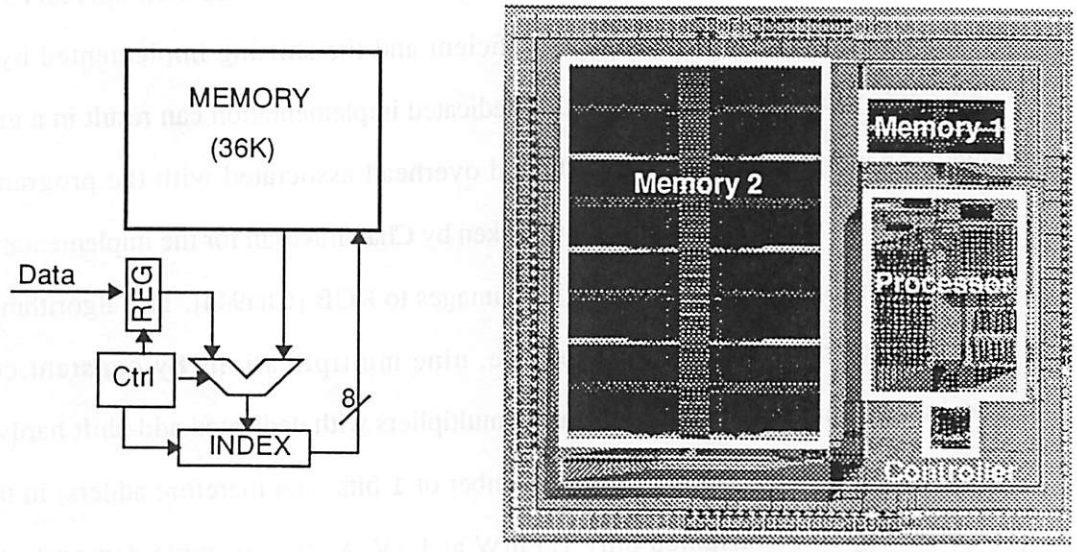
distributed memory structures over global or shared memories.

A similar result holds true for controller schemes. A centralized controller relies on a single finite state machine, which must generate all required control signals for the chip and distribute them globally. This global network of control signals and its associated interconnect capacitance can consume a large amount of power. Under a distributed or local control scheme, however, only a small number of global state bits must be distributed across the chip. The remainder of the control signals can be generated locally with a corresponding reduction in interconnect capacitance required to distribute these control lines.

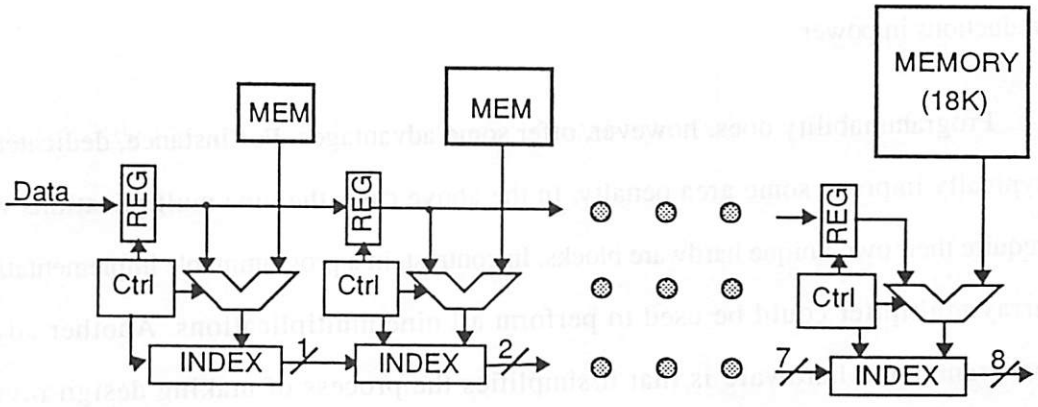
As a general rule, then, exploiting locality through distributed processors, memories, and controllers can lead to significant power savings. Several low-power designers have already recognized this fact. For example, as shown in Figure 2-20, Lidsky demonstrated the benefits of partitioning by implementing both a centralized and distributed version of a vector quantized image encoder [Lid94]. Through partitioning of memories, processing, and control Lidsky was able to achieve a 16.9x reduction in power at the expense of less than a 2x increase in area.

2.8.4 Programmability

The previous section suggested that distributed processors, which can be optimized for specific tasks, might consume less power than general-purpose, programmable processors that must execute many different tasks. This observation was made in the context of distributed versus centralized processing; however, it brings up the important issue of programmable versus dedicated hardware. As an example, consider the implementation of a linear, time-invariant filter. Such filters involve multiplication of variables by constants. All required constant multiplications could be implemented on a single array multiplier. This is a programmable scenario since the multiplier can be programmed to execute any of the different constant multiplications on the same hardware. Alternatively, we can consider implementing each of the constant multiplications on dedicated hardware. In this case, since the multipliers each need to perform multiplication by a



(a) Centralized implementation



(b) Distributed implementation

Centralized:	Distributed:
f_{clk} = 8.3 MHz	f_{clk} = 1.04 MHz
V_{dd} = 3.0 V	V_{dd} = 1.1 V
A = 53.3 mm^2	A = 92.9 mm^2
P = 6.96 mW	P = 0.412 mW
	Gain = 16.9x

(c) Comparison of implementations

Figure 2-20 : Centralized vs. distributed implementation of VQ image encoder ([Lid94])

single, fixed coefficient, the array multipliers can be reduced to add-shift operations with an adder required only for the 1 bits in the coefficient and the shifting implemented by routing. For coefficients with relatively few 1's, this dedicated implementation can result in a greatly reduced power consumption, since the waste and overhead associated with the programmable array multiplier is avoided. This approach was taken by Chandrakasan for the implementation of a video color space converter for translating YIQ images to RGB [Cha94a]. The algorithm consists of a 3x3 constant matrix multiplication (i.e. nine multiplications by constant coefficients). Chandrakasan not only replaced the array multipliers with dedicated add-shift hardware, but also scaled the coefficients to minimize the number of 1 bits (and therefore adders) in the algorithm. The resulting chip consumed only 1.5 mW at 1.1V. As this example demonstrates, avoiding excessive or unnecessary programmability in hardware implementations can lead to significant reductions in power.

Programmability does, however, offer some advantages. For instance, dedicated hardware typically imposes some area penalty. In the above case, the nine multiplications would each require their own unique hardware blocks. In contrast, in a programmable implementation a single array multiplier could be used to perform all nine multiplications. Another advantage of programmable hardware is that it simplifies the process of making design revisions and extensions. The behavior of a programmable component can be altered merely by changing the instructions issued by the software or firmware driving the device. A version of the chip relying primarily on dedicated hardware, however, might require extensive redesign efforts.

2.8.5 Data representation

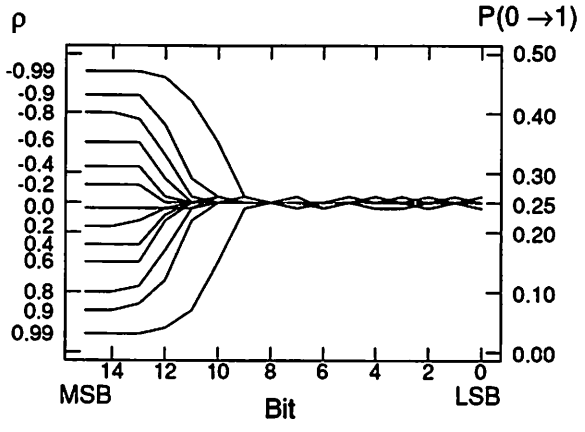
Another architectural decision that can impact power consumption is the choice of data representation. In making this decision, the designer typically has several different alternatives from which to choose: e.g. fixed-point vs. floating-point, sign-magnitude vs. two's-complement, and unencoded vs. encoded data. Each of these decisions involves a trade-off in accuracy, ease of

design, performance, and power. This section discusses some of the issues involved in selecting a data representation for low power.

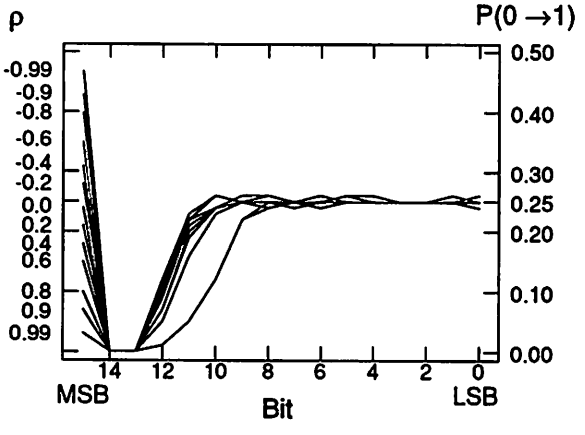
The most obvious trade-off involves deciding upon a fixed- or floating-point representation. Fixed-point offers the minimum hardware requirements and, therefore, exhibits the lowest power consumption of the two. Unfortunately, it also suffers the most from dynamic range difficulties. Software data scaling offers some relief from this problem; however, it must be incorporated into the processor micro-code and, thus, has some runtime overhead. Floating-point, in contrast, alleviates the dynamic range difficulties at the expense of extensive hardware additions. This increased hardware leads to correspondingly higher capacitances and power consumption. As a result, floating-point should be selected only when absolutely required by dynamic range considerations. Even in these cases, block floating-point may offer sufficient accuracy with much less hardware and power than a true floating-point implementation. At the expense of a small amount of additional normalization hardware (and power), it accomplishes the data-scaling process (for the most part) automatically. Thus, it achieves some compromise between dynamic range and hardware overhead.

A related decision involves selection of the required word length for the datapath. Often designers overestimate dynamic range requirements, building in quite a wide safety margin by using a larger word length than required by the application. This peace of mind comes, however, at the expense of performance and power. Therefore, a careful analysis of the required, rather than the desired, accuracy is essential for an efficient, low-power implementation.

Aside from issues of accuracy and word length, the designer must also select an arithmetic representation for the data. For example, two's-complement, sign-magnitude, and canonical signed-digit are all possible arithmetic representations for data. Two's-complement is the most amenable to arithmetic computation and, therefore, is the most widely used. In this representation, the least significant bits (LSB's) are data bits, while the most significant bits (MSB's) are sign bits.



(a) Two's-Complement



(b) Sign-Magnitude

Figure 2-21 : Bit transition activities for two different data representations

As a result, the MSB's contain redundant information, which can lead to wasted activity (and power) for data signals that experience a large number of sign transitions. In contrast, sign-magnitude data uses a single bit to describe the sign of the data and so sign transitions cause toggling in only one bit of the data [Cha94b]. Figure 2-21 shows the bit activities of both data representations for streams of gaussian data with varying temporal correlation, ρ. The figures demonstrate that sign-magnitude eliminates a good deal of the activity associated with the sign-extension region of the two's-complement representation. Unfortunately, the additional overhead

required to perform computations in sign-magnitude representation typically outweighs any benefits from reduced data activity. In some special cases, however, such as driving global data buses with large capacitive loads, it may be worthwhile to convert to sign-magnitude representation. For example, driving a 16-bit bus using the sign-magnitude data of Figure 2-21 requires anywhere from 4% (ρ=0.99) to 50% (ρ=-0.99) less power than would be needed for the equivalent two's-complement data.

A related issue is that of data encoding. In some sense, the preceding discussion has focused on avoiding waste in the chosen data representation. For example, the sign bits in two's-complement represent a waste of the available word length. This issue can be partially addressed through schemes for encoding data. For example, although it is typically not necessary to differentiate between large values as precisely as small values, representations like two's-complement that use linear quantization do not take advantage of this fact. Floating-point provides one solution to this problem, but other schemes such as logarithmic companding can be used to achieve similar results. Unfortunately, as with sign-magnitude, many computations (such as additions) don't have straightforward implementations in the logarithmic domain; however, some computations actually become simpler and less power consuming in this domain: e.g. multiplications translate to additions. Applications requiring a large number of multiplications can take advantage of this fact by using logarithmically encoded data. An example of one such implementation is a low-power hearing aid implemented in the logarithmic domain to maximize word length utilization and to reduce the power associated with required "multiply" operations [Eng89]. Speech recognition based on Hidden-Markov Models is another example of a multiplication-intensive algorithm that might benefit from this approach [Sto91]. Other techniques for low-power data encoding have been proposed as well and offer up to a 25% average reduction in I/O transitions [Sta94].

Clearly, there are many trade-offs involved in selecting a data representation for low-power systems. It is unlikely that any one choice would be ideally suited for all applications. Instead, the

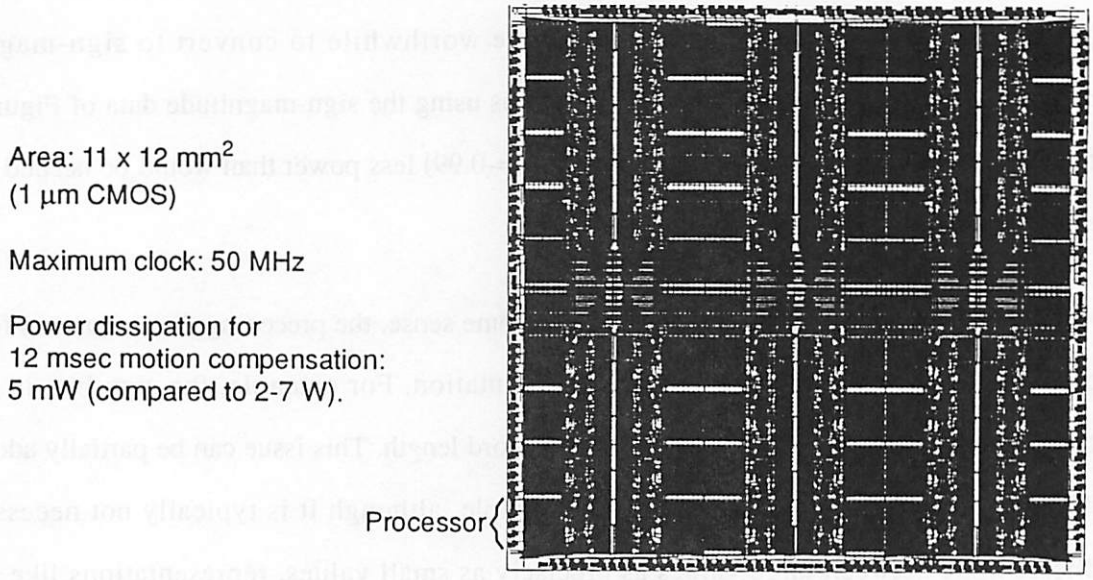


Figure 2-22 : Implementation of PADDI-2 signal processing architecture

designer must undertake a careful analysis of the application requirements in terms of performance and accuracy before selecting the appropriate representation. Moreover, it might be beneficial to use different data representations in different parts of the systems at the expense of some data conversion overhead.

2.8.6 Case Study: The PADDI-2 Architecture

A. Yeung has implemented an architecture that embraces many of the low-power themes presented in the preceding discussion [Yeu92]. The implementation is shown in Figure 2-22. The chip relies on a distributed processing scheme utilizing 48 nano-processors each containing a 16-bit datapath, register files, and a small controller and instruction cache. Processors are locally synchronous, but communicate with each other through handshaking. When operated at low voltage, the chip can implement a motion compensation algorithm consuming 100-1000 times less power than a programmable uni-processor. The architecture achieves this performance by

exploiting voltage scaling, parallelism, distributed processing, locality of reference, and automatic power-down.

2.8.7 Architecture-Level Conclusions

This section has demonstrated the importance of high-level optimization strategies in reducing the overall system power. Whereas, the technology-, layout-, circuit-, and gate-level techniques of Sections 2.4-2.7 demonstrated power savings of a factor of two or less, the architecture and system-level techniques of this section can often result in orders of magnitude power savings. The techniques discussed include power management, partitioning, using dedicated hardware, and selecting an appropriate data representation. Perhaps the most important strategy involved trading area/performance for power through concurrent processing at low voltages, since it offered a quadratic reduction in power. It is applicable at the architecture level through module or block pipelining and parallelism, as well as at the system level through multi-processor platforms.

Up to this point, we have discussed low-power techniques related purely to the hardware component of the system. Software and algorithmic considerations can also have an impact on power consumption. The next section discusses some of the issues involved in optimizing algorithms and software for low power.

2.9 Algorithm Level

Algorithms can have both a direct and an indirect influence on system power consumption. For example, algorithm complexity and operation count have a straightforward and direct impact on power. Other issues such as available computational concurrency have a more indirect influence on power by determining how effectively the algorithm can be mapped onto a "low-power" architecture (e.g. a low-voltage concurrent processor). This section evaluates both types of

Algorithm	Coder Complexity	Decoder Complexity	Total Complexity
VSELP	16.7	0.839	17.5
DoD CELP	24.7	1.04	25.7
LD-CELP	17.1	8.74	25.8

Table 2-2 : Speech coder complexities in Millions of Operations Per Second (MOPS)

considerations, first addressing low-power algorithms directly, then discussing how algorithms can be targeted towards low-power architectures.

2.9.1 Low-Power Algorithms

Three primary factors directly affect the power consumed in executing an algorithm regardless of the architecture chosen: complexity, regularity, and precision.

Complexity

The complexity of an algorithm can be measured in several ways. One simple measure of complexity is the instruction or operation count. Since each operation consumes power, this is a useful metric for gauging the quality of "low-power" algorithms. For minimum power, the number of operations in an algorithm should obviously be kept small. Unfortunately, higher performance algorithms often require more operations. So, often there is some trade-off of performance for power. This is not always the case, however. Table 2-2 shows three speech coding algorithms that achieve similar performance [Lan91]; however, the VSELP algorithm achieves this level of performance with about 32% fewer operations and, therefore, might be more suitable for a low-power implementation. Keep in mind, however, that not all types of operations consume the same

amount of power. For instance, multiplies typically consume much more power than additions. Thus, one must consider not only the total operation counts, but also the type of operations required by the algorithm.

The number of data accesses and the data storage requirements of an algorithm are another important measure of complexity. Accesses to memories and register files can be expensive in terms of power. Therefore, algorithms which minimize not only operation count, but also memory access and size requirements are often more amenable to low-power implementation.

Regularity

While computational complexity affects the power consumed in the datapath, the regularity of an algorithm can affect the complexity and power consumption of the control hardware and the interconnect network. For an ASIC component, a very regular algorithm will require fewer states to describe its behavior. This translates directly to a smaller finite state machine and reduced controller power. For a programmable processor, a regular algorithm leads to fewer power consuming branch instructions, and in the case of loops, a higher success-rate for branch prediction (where an incorrect branch prediction leads to wasted power). Moreover, regular algorithms tend to have regular communication patterns as well. This makes for more efficient use of interconnect networks, since the same networks can often be reused for many different parts of the algorithm. This is an important point since extensive global communications networks can contribute significantly to overall chip power consumption.

Precision

As stated in Section 2.8.5, architectures with larger word lengths consume more power. Yet an architecture must employ a large enough word length to satisfy the precision requirements of the algorithm. Even algorithms that implement the same function sometimes have widely varying word length requirements. For example, Table 2-3 shows five different filter structures that all implement the Avenhaus bandpass transfer function along with the word length required to

Filter Structures	Word Length
Cascade	13
Continued Fraction	23
Direct form II	19
Ladder	14
Parallel	13

Table 2-3 : Required word lengths for various Avenhaus filter structures

achieve adequate precision [Ave93]. The word length varies widely from a minimum of 13 to a maximum of 23 bits [Cro75]. Since word length can affect the performance and power of the final implementation, the precision required to implement a given algorithm should be taken into consideration when selecting a low-power algorithm. Indeed, variations in the complexity and word length among these functionally equivalent algorithms result in about a 3x difference between the lowest and highest power implementations (see Chapter 7).

2.9.2 Algorithms for Low-Power Architectures

From the previous section we recognize that issues such as the complexity, regularity, and precision of an algorithm all have a direct effect on the power consumed by the resulting implementation. Another more subtle factor to consider is how well the candidate algorithm maps onto the desired low-power architecture. Section 2.8 suggested several architectural styles (such as concurrent and distributed processing). These architectures will only be low power if the algorithms which they execute map efficiently onto the hardware. This imposes certain requirements on algorithms that are intended to execute on low-power architectures. The two most important requirements are concurrency and modularity.

Concurrency

Concurrent processing can be used to reduce power consumption by trading performance increases for lower voltage operation and, subsequently, power. This strategy is useless, however, unless the algorithm executing on the architecture has enough available concurrency. The bottleneck is set by the so-called *sequential* or *recursive* operations. That is, operations that depend on the results of previous computations to complete.

For example, Chapter 6 will describe a hardware divider based on the Newton-Raphson iteration. This unit recursively executes the following iteration:

$$x_{t+1} = x_t(2 - x_t b) \qquad \text{(EQ 18)}$$

Notice that each new sample depends on the results of the previous sample. Even if several parallel processors were available to perform several iterations simultaneously, they would be wasted due to the recursive nature of the algorithm.

Many techniques exist to break this recursive bottleneck. For example, here we can use *look unrolling* to expose additional parallelism in the algorithm:

$$x_{t+2} = [x_t(2 - x_t b)]\,\{2 - [x_t(2 - x_t b)]\,b\} \qquad \text{(EQ 19)}$$

Now two iterations can be performed simultaneously since the expressions for both x_{t+1} and x_{t+2} require only x_t as input. Additional applications of loop unrolling would allow the algorithm to execute efficiently on a processor architecture with an even higher degree of parallelism.

Other algorithmic transformations can also help to increase the concurrency available in a given algorithm. These include software pipelining, retiming, common subexpression replication and elimination, and algebraic transformations. The compiler literature contains many references to transformations for reducing critical paths and increasing concurrency, and some work has specifically addressed the application of transformations to the power optimization problem [Cha93].

Modularity or Locality

Another important low-power architectural technique is to exploit locality through distributed processors, memories, and control. Maximizing the modularity in an algorithm ensures that efficient use can be made of this distributed processing structure. In particular, computational graphs that are characterized by clusters of strongly connected computations with relatively few global interconnections can be expected to map efficiently onto a distributed architecture. Thus, exploiting architectural locality for low-power must be accompanied by a correspondingly modular algorithm.

2.9.3 Algorithm-Level Conclusions

Designing low-power algorithms involves considering not only issues such as complexity that directly affect power consumption, but also issues such as available concurrency that indirectly affect power by determining how effectively the algorithm can be implemented on a "low-power" architecture. The interaction between algorithm (software) and architecture (hardware) is a strong one, and optimization of the two levels cannot be undertaken independently. Finally, the combined effect of all high-level (architectural, algorithmic, and system) optimizations can have a dramatic (orders of magnitude) effect on power consumption.

2.10 Chapter Summary

This chapter has provided a comprehensive introduction to low-power digital CMOS design. This included an introduction to the power consumption mechanisms in CMOS circuits, as well as an overview of recurring themes in low-power design and a number of specific low-power techniques for levels of abstraction ranging from the technology to the architecture, system, and algorithm levels.

In this discussion it became clear that the dominant component of power consumption in

CMOS circuits arises from the dynamic charging and discharging of parasitic capacitances, with the power consumption given approximately by $P = \frac{1}{2}\alpha C V_{dd}^2 f$. Power reductions can be achieved by reducing the physical capacitance, C, the activity, αf, and the supply voltage, V_{dd}. Techniques were presented for reducing these various factors, and these techniques tended to follow certain recurring themes such as trading area and performance for power, avoiding waste, and exploiting locality. Trading performance for power turned out to be a particularly important strategy. Basically, it involved using (primarily) architecture- and system-level techniques such as concurrent processing to improve performance and then trading this performance for lower power by reducing the supply voltage. The quadratic dependence of power on voltage ensured that the savings were substantial.

It also became clear that the largest power reductions were due to decisions made at the higher levels of abstractions. While technology, layout, gate, and circuit optimizations may offer power reductions of a factor of two at best, optimization of architectures and systems was shown to result in orders of magnitude reductions in power consumption. Therefore, designers should concentrate their power optimization efforts on the higher levels.

It follows that CAD tools to aid low-power designers should focus primarily on these levels. Unfortunately, nearly all efforts in low-power tools for analysis and optimization have been and continue to be concentrated at the gate and circuit levels. These observations provide a strong motivation for the high-level design methodologies and tools that are the focus of this thesis. In that sense, this chapter has been useful not only in providing an overview of the state of the art in low-power digital design, but also in providing a solid explanation as to why the low-power design tools presented in this thesis represent a significant contribution to the field. Before going into a specific description of these tools, however, the next chapter will provide background about the currently available low-power design tools.

CHAPTER 3

Power Estimation: The State of the Art

Chapter 2 argued that increasingly significant power reductions are possible as designers focus optimizations on higher levels of abstraction. This provided a strong motivation for the architectural power analysis tools that will be described in this thesis. This does not imply that lower level tools are unimportant. On the contrary, each layer of tools provides a foundation upon which the next level can be built. The estimation techniques used at one level can often be abstracted to a higher level and applied again in a slightly modified form. This building process by definition must occur incrementally and, therefore, it makes sense to work up to architectural power analysis by beginning with the lower level techniques. One goal of this chapter, then, is to present a comprehensive overview of the state of the art in power analysis at the circuit, gate, architecture, and algorithm levels of abstraction.

Another aim is to further motivate the high-level power analysis techniques that will be presented in the following chapters. Chapter 2 demonstrated that high-level tools would be the most useful in terms of power optimization leverage, but it is not clear yet what tools are available in this domain. By providing an overview of all currently available power analysis tools, this

chapter will demonstrate the preponderance of low-level tools (circuit and gate level) and the lack of high-level tools (architecture and algorithm level). Moreover, the few architectural and algorithmic tools that do exist have several serious flaws and weaknesses. This chapter will discuss these weaknesses providing the reader with a clear insight into how the high-level tools presented in the following chapters improve upon the state of the art.

3.1 Circuit-Level Tools

The circuit level refers to that level of abstraction for which the transistor is the primitive element. This section will discuss the tools and techniques available for power analysis at the circuit level. Interestingly enough, the majority of these tools were actually designed for analog or digital circuit simulation with performance analysis being the main consideration and power coming into play only as an afterthought.

Within the confines of circuit-level power estimation there are trade-offs to be made in terms of estimation accuracy and speed. For example, transistor behavior can be modeled at the device level or at the less detailed switch level. This section will present tools of both types along with their associated speeds and accuracies.

3.1.1 Device-Level Modeling

The device level is the most accurate and detailed level of circuit modeling. Here, transistor behavior is modeled using the full non-linear current, voltage, and capacitance expressions. Very few approximations are made and typically all components of power consumption - both static and dynamic - are analyzed. In fact, the main limitation on accuracy is often due more to an incomplete knowledge and modeling of parasitics and interconnect than to inaccuracies in the device models. This problem can be minimized if a circuit layout is available from which wiring parasitics can be extracted.

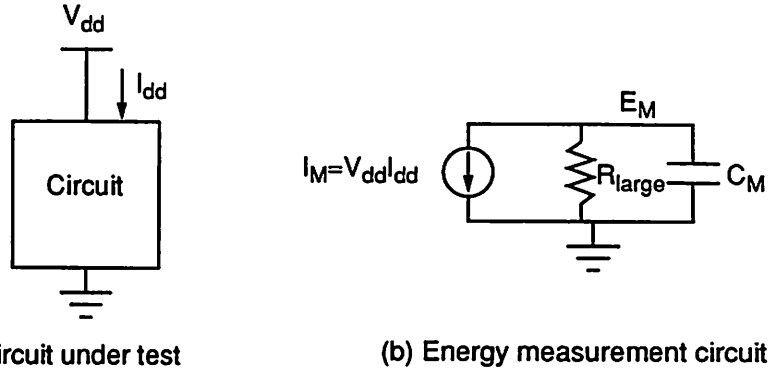
Within this category, some tools apply complex analytical device models. Others substitute lookup tables for these expressions in an attempt to improve the simulation speed by sacrificing accuracy. Still others combine the results from a hierarchy of simulators (from device level to register-transfer level) to reduce the simulation time further still. Examples of each of these strategies will be discussed in the following subsections.

Analytical Device Models

The most well known circuit simulator that uses fully analytical device models is SPICE [Nag75]. The acronym SPICE stands for Simulation Program with Integrated Circuits Emphasis. SPICE is a circuit simulator that models transistors at the device level. SPICE contains several levels of models with varying accuracies; however, even the least detailed SPICE models offer greater accuracy than most other tools that will be discussed in this chapter. Its creators designed SPICE more for timing and functionality verification than for power analysis; however, since SPICE keeps track of both node voltages and branch currents, it is possible to measure power consumption during circuit simulation. Monitoring the branch current drawn from the voltage supply gives an accurate picture of the instantaneous power consumption of the circuit, $P=I_{dd}V_{dd}$.

Dummy circuits can be constructed to monitor average power consumption automatically [Kan86][Yac89]. Figure 3-1 illustrates the concept of average power measurement using SPICE. Element I_M represents a current-controlled current source, which supplies a current proportional to that flowing out of voltage supply V_{dd}. This current charges capacitor C_M resulting in a voltage $E_M = \int_0^t I_{dd}(\tau)\, V_{dd} d\tau$, equal to the energy drawn from the supply up to time t. Dividing this energy by t gives the average power consumption.

With devices modeled at such a low level, SPICE allows measurements of phenomenon that cannot be captured with higher level modeling tools. For instance, SPICE accurately models non-linear capacitances. In particular, the gate capacitance of transistors, though often assumed constant, is actually a function of the operating region and terminal voltages of the transistor in



(a) Circuit under test (b) Energy measurement circuit

Figure 3-1 : Circuit for measuring average power in SPICE

question [Wes88]. Moreover, the source and drain regions of CMOS devices contain parasitic diodes that lead to a non-linear junction capacitance at these nodes as shown in Chapter 2. These effects, modeled by SPICE, are often ignored by higher level power analysis tools.

Furthermore, SPICE allows measurement of certain components of power consumption that cannot be captured with higher level modeling tools. SPICE models not only the dynamic power consumed in charging and discharging nodal capacitances, but also the dynamic power related to short-circuit currents. Recall from Section 2.1.2, that for a brief time during input transitions, both pull-up and pull-down devices are active causing a transient short-circuit current to flow from V_{dd} to ground. The magnitude of this power component really depends on the balance between input and output rise times - the more disparate the rise times are, the more important this component becomes. SPICE also models leakage currents, which arise due to reverse-biased parasitic diodes intrinsic to the CMOS device structure.

There is, of course, a price to pay for this level of modeling accuracy - namely, simulation time. Depending on the simulation platform, SPICE has a practical limit of several hundred to several thousand transistors. Moreover, even at these levels simulations may run for several hours, days, or even fail to converge altogether. Therefore, power simulation of large chips is simply not

feasible. In addition, the requirement of a transistor-level netlist makes SPICE more useful as a back-end power verification tool or as a tool for characterizing small modules than as a design exploration aid. For these reasons, tools developers have introduced several less accurate but faster simulators.

Table-Based Device Models

Some simulators use table lookup to speed the circuit simulation process. CAzM is an example of a tool that follows this approach, replacing the analytical device models used by SPICE with simplified piecewise-linear versions [Erd89]. These simplified models are stored as tables, which give device currents and charge distributions as a function of terminal voltages. The simulation process progresses through a series of table lookups rather than by solving a set of non-linear equations. Through this strategy, CAzM achieves a speed-up of about 10x relative to SPICE, while maintaining good accuracy. In fact, accuracies within 10% of SPICE are a reasonable expectation.

While an improvement over SPICE, the speed of CAzM is still not sufficient for chip-level design space exploration. One technique to speed up the simulation of large circuits is to employ an event-driven approach.

Event-Driven Table Lookup

Epic Design Technology's PowerMill is an example of a transistor-level simulator that uses an event-driven simulation algorithm to reduce analysis time [Den94]. Under this strategy, table lookup is still used to determine device currents based on terminal voltages, but in addition devices are only evaluated when they are active. An element is considered active when a "significant" voltage change occurs on its terminals. The impact of this nodal event is evaluated through a transient simulation of the node and its closest neighbors, where "closeness" is determined by the channel-connected component concept [Ter83]. A one-step-relaxation (OSR) approach is used to improve the simulation speed, instead of the simulate-to-convergence strategy

used by most circuit simulators [Sal83]. Rigorous monitoring and control of step-size ensures accuracy and convergence.

This event-driven table lookup approach makes PowerMill more than one hundred times faster than SPICE for results within 5%. Still, faster analysis is possible if we abandon modeling individual transistors and instead model typical patterns (or arrangements) of transistors.

Precharacterized Charging Patterns

Based on this concept, Deng developed a timing simulator that can be used to evaluate power consumption faster than tools which simulate individual devices [Den88]. TSIM uses precharacterized delay models based on SPICE simulations. More specifically, during a pre-characterization phase, SPICE is used to generate voltage waveforms for various *charging patterns* (arrangements of PMOS transistors) and discharging patterns of NMOS transistors. The waveforms are then parameterized by transistor sizes, input rise/fall times, and node capacitances. The resulting simplified waveform transition formulas are placed in a library for use by TSIM.

During simulation, TSIM decomposes more complex logic gates into combinations of the PMOS and NMOS charging patterns. Applying the SPICE-based formulas, TSIM produces an approximate voltage waveform for each of the circuit nodes. The current waveform is estimated as the superposition of a short-circuit and a capacitive component: $I = I_{sc} + I_{cap}$. The short-circuit component depends on the overlap time between the input and output voltage waveforms, and is calculated from precharacterized formulas in the aforementioned SPICE library. The capacitive component is derived from the well-known expression, $I_{cap} = CV/T$. In order to capture their non-linear dependence on voltage, the device capacitances are again taken from the SPICE-generated library data. Finally, the charge transferred during switching is distributed into triangular current pulses, which are then summed to arrive at the final current waveform. From this, peak and average current values can be derived.

To sum up, TSIM follows an event-driven strategy based on recognizing common patterns of transistors in order to achieve a 100-1000x speed-up over traditional device simulators such as SPICE. In order to achieve good accuracy it relies on SPICE-generated libraries of voltage and current waveforms for typical patterns of NMOS and PMOS transistors. Accuracy is consistently within 10% of SPICE; however, if we are willing to sacrifice slightly more accuracy we can achieve even faster simulation times.

Hierarchy of Simulators

One technique to speed up the simulation of large circuits is to partition them into smaller subcircuits, which can then be simulated independently. PRITI takes this approach [Van93]. At its core, PRITI still relies on device-level circuit simulation as performed by tools such as SPICE and CAzM. It does not, however, attempt to simulate large circuits as a single unit. Instead, PRITI achieves efficient chip-level simulation by employing a hierarchy of simulators with various levels of speed and accuracy. In particular, PRITI partitions large circuits into many subcircuits called combinational logic circuits (CLC's) whose inputs and outputs are delineated by registers. The entire chip, consisting of these CLC modules, is then simulated at the RT level. This functional simulation is quite fast and provides PRITI with the input data streams for the CLC's.

PRITI then spawns parallel circuit simulations for each of the CLC's using the RTL data patterns. Since the CLC's are much smaller than the full circuit, they can be simulated by a SPICE-like tool in a reasonable amount of time. Furthermore, if the user is only interested in average and peak power estimates, then the simulation need only continue until these estimates converge. Since CLC's are simulated separately, the required number of simulation cycles can be independently optimized for each. In contrast, if the entire chip were simulated as a whole, the slowest converging CLC would determine the required number of cycles for the entire chip.

In this way, PRITI provides an efficient simulation strategy for large circuits. Since at its foundation, the simulation is still performed at the circuit or device level, a high degree of

accuracy is maintained. While much faster than SPICE alone, the speed of this basically transistor-level approach is still insufficient for high-level design space exploration. Moreover, the reliance on a transistor-level netlist once again limits the applicability of PRITI to back-end verification rather than high-level optimization.

Device-Level Summary

The majority of designers today rely on device-level circuit simulation to analyze the power consumption of their circuits. This level of abstraction is attractive since it offers a high level of accuracy. SPICE-like simulators account not only for non-linear device currents and capacitances, but also for leakage and short-circuit currents that can be important for certain circuits. This accuracy comes at the expense of long simulation times even when intelligent speed-up and partitioning strategies are applied. This penalty can be unacceptable for chips with many thousands of transistors and has forced some designers to move from device- to switch-level models.

3.1.2 Switch-Level Modeling

Switch-level models treat transistors as devices with only two modes of operation: on and off. The "off" transistor is taken as a perfect open circuit, while an "on" transistor is typically modeled by an effective linear on-resistance. Wires are typically modeled as perfect conductors having only a capacitance attribute associated with them. As at the device level, wire capacitances can initially be estimated and later back-annotated with extracted values when the layout becomes available.

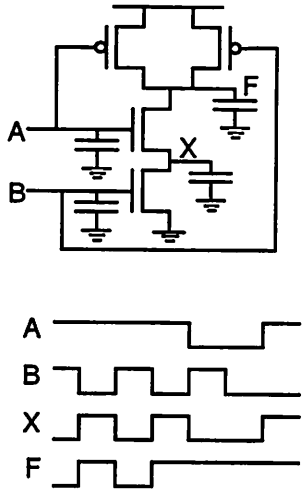
The simplified switch-level model is faster to evaluate than a non-linear device model at the cost of some loss in accuracy. This section presents several switch-level models which offer various performance-accuracy trade-offs relative to SPICE. Deterministic simulation techniques which give input *pattern-dependent* results will be presented, as will *pattern-independent* probabilistic tools.

Accumulating Nodal Transitions

Many switch-level simulators allow the user to estimate dynamic power consumption by accumulating transition counts for circuit nodes and multiplying by physical node capacitances. IRSIM (based on RSIM [Ter83]) is a switch-level simulator distributed by Stanford University that uses this technique [Sal89]. Like device-level simulators, it analyzes circuits described by transistor-level netlists, but unlike those tools it relies on completely linear device models to improve simulation speed at the expense of accuracy. The linear device models apply not only to the transistors, which IRSIM models as switches with effective on-resistances, but also to the gate and junction capacitors, which are modeled as fixed, effective capacitances rather than voltage-dependent capacitances. The IRSIM model is further simplified by assuming two-level voltage quantization. In other words, all nodes are assumed to be either at V_{dd} or ground rather than some intermediate voltage.

IRSIM was originally developed as a functionality and timing verification simulator; however, realizing the growing importance of power as a design constraint, the developers have added power measurements capabilities into version 9.0 of the simulator. During simulation, IRSIM-9.0 keeps track of the charging and discharging of all nodal capacitances. At the end of simulation, IRSIM generates a report describing the total capacitance switched at each node as demonstrated in Figure 3-2. This can be converted to an average power by multiplying by V_{dd}^2 and dividing by the simulation time. Thus, IRSIM can be used to measure the capacitive switching component of dynamic power consumption. Unlike SPICE and PowerMill, it does not measure the short-circuit nor the leakage components of power consumption.

The power measurement capabilities of IRSIM-9.0 are fairly rudimentary and, therefore, we have developed an extended version of IRSIM-9.0 called IRSIM-CAP. IRSIM-CAP has an improved measurement strategy for modeling glitching power, as well as additional commands to support incremental power measurement. IRSIM-CAP achieves improved accuracy by modeling



Signal	$C_{physical}$	N(0→1)	$C_{effective}$	% Total
A	20 fF	1	20 fF	6.9%
B	20 fF	2	40 fF	13.8%
X	10 fF	3	30 fF	10.3%
F	100 fF	2	200 fF	69.0%

Figure 3-2 : Power measurement in IRSIM

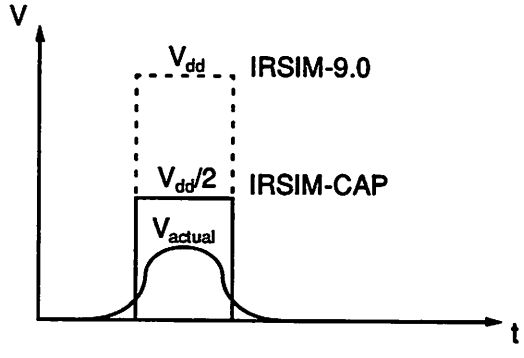


Figure 3-3 : Glitch modeling in IRSIM-9.0 and IRSIM-CAP

power consumption based on a three- rather than two-level voltage quantization scheme (see Figure 3-3). Rather than counting a glitch from low (L) to undefined (X) back to low (L) as a full rail-to-rail transition, IRSIM-CAP will model it as a swing from ground to $V_{dd}/2$ and back to ground. In most cases this three-level quantization scheme reflects actual circuit behavior much more closely than the two-level rail-to-rail model.

IRSIM-CAP includes several additional commands to aid the user in measuring power

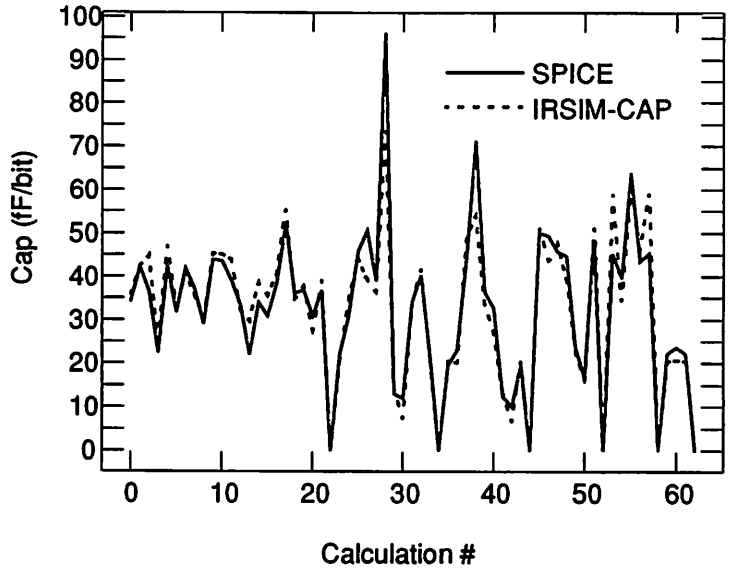


Figure 3-4 : IRSIM-CAP versus SPICE after parameter calibration

consumption. These additional commands include 'zeropower,' 'sumeffcap,' and 'capreport.' The support for incremental power measurements ('zeropower' and 'sumeffcap') has proven to be particularly useful. These commands allow measurement of the capacitance switched during a small subset of the simulation rather than over the entire simulation. This capability is useful for efficiently characterizing module power consumption in response to several different input transitions during a single simulation run. In addition, IRSIM-CAP contains support for power binning ('capreport'), which allows the user to attribute power consumption to the various modules in the design hierarchy (e.g. capacitance switched in control as opposed to the datapath).

The power estimates of IRSIM-CAP are reasonably close to SPICE as shown by Figure 3-4. The figure shows the capacitance switched by a 16-bit subtracter over 64 calculations. The root mean square (rms) error for this IRSIM-CAP simulation is 13.1%. SPICE simulation took about 30 minutes on a Sun SPARCstation 10 while IRSIM-CAP required only 3 seconds - a speed differential of about 580x. In this example, many of the inherent weaknesses of the linear RC model used by IRSIM-CAP were overcome by proper calibration of the IRSIM-CAP device

parameters to the SPICE device parameters. IRSIM-CAP commonly achieves average power measurements within 15% of SPICE results. Thus, it is slightly less accurate than TSIM or PowerMill, but is still a reasonable choice for many applications.

Extensions for Static and Short-Circuit Power Analysis

Many switch-level tools, such as IRSIM, ignore the static and short-circuit components of power consumption. Tjarnstrom's LDS is an attempt to remedy this situation [Tja89]. Like IRSIM, LDS is a switch-level simulator that can be used to estimate power consumption. It uses a linear RC delay model and, therefore, is subject to many of the same inaccuracies as IRSIM. It does, however, attempt to overcome one of the difficulties associated with switch-level power simulation. Namely, LDS accounts not only for CV^2f power, but also for short-circuit and static power.

LDS accounts for short-circuit currents by assuming that a conducting path exists between V_{dd} and ground for input voltages between V_{tn} and V_{dd}-$|V_{tp}|$. This, coupled with information regarding the input slope, allows an approximation of the energy consumed by short-circuit currents:

$$E = K_1 K_2 S_{in} \frac{V_{dd}^2}{R_p + R_n} \qquad \text{(EQ 20)}$$

where K_1 is a constant that determines the duration of the conducting path and K_2 determines the influence of the input slope, S_{in}. In particular, the larger the output slope relative to the input slope, the smaller the short-circuit current:

$$K_2 = \frac{1}{1 + K_3 \frac{S_{out}}{S_{in}}} \qquad \text{(EQ 21)}$$

where K_3 is approximately equal to nine, based on empirical observations of short-circuit current by Veendrick [Veen84].

LDS also attempts to account for static power dissipation. When circuit inputs result in a static

conducting path between V_{dd} and ground, LDS introduces a static power component:

$$P = \frac{V_{dd}^2}{R_p + R_n} \quad \text{(EQ 22)}$$

So in a limited way, LDS attempts to account for components of power beyond the traditional capacitance charging and discharging power. These extensions are still subject to the limitations of linear device models. Still, they can provide first-order approximations, which might be useful for some classes of circuits such as pseudo-NMOS or CMOS circuits with slow signal transitions.

Pattern-Independent Power Analysis - The Monte Carlo Approach

So far, all the techniques discussed have been simulation based. That is, the user provides some input vectors and the tool measures the response of the circuit to that data stream. In that sense, the techniques produce *pattern-dependent* estimates of power consumption. In some cases, typical input patterns are not readily available and it might be useful to have access to *pattern-independent* estimates of power - e.g. the average power over all possible sets of inputs.

Huizer suggested a technique that can be used in conjunction with any of the simulation approaches already mentioned in order to produce pattern-independent average powers within a desired confidence interval [Hui90]. Basically, Huizer advocates a Monte-Carlo approach, applying random input vectors to the circuit and monitoring the standard deviation of the resulting power estimates as they converge.

While this does, in some sense, produce pattern-independent results, it still requires simulation over a large number of cycles. In addition, the power consumption based on random inputs may be far from that caused by more realistic inputs. Some techniques have been developed (both at the circuit and gate levels) that avoid simulation altogether and compute the average power directly in a single pass.

Pattern-Independent Power Analysis - The Probabilistic Approach

Some tools, such as Cirit's LTIME, take a probabilistic approach to pattern independence, relying on the notion that given the switching rates at the circuit inputs, the switching rates of the internal nodes can be calculated [Cir87]. Thus, rather than monitoring actual transitions that occur during a simulation, LTIME makes one pass through the circuit and propagates transition *probabilities* to all nodes. The average power consumed by the circuit is then approximated by:

$$P = \frac{1}{2} V_{dd}^2 \sum_i C_{Li} T_i G_i \qquad \text{(EQ 23)}$$

where C_{Li} is the effective load capacitance of transistor i, T_i is the probability that transistor i will make a state transition from on to off or vice-versa, and G_i is the probability that the source of transistor i connects to a supply rail.

LTIME makes several important approximations. First, while simulators account for the temporal (sample to sample) and spatial (node to node) correlations of signals, LTIME assumes independence in both dimensions when computing probabilities. This approximation can lead to fairly large errors for circuits with a high degree of signal correlation such as those containing reconvergent fan-out or feedback. Reconvergent fan-out refers to cases in which a logic signal fans out to two or more separate logic trees which later recombine to influence a single node. For instance, in Figure 3-5, X influences Y through two separate paths. If we assume that X and $\overline{X}$ are uncorrelated, then we estimate that Y connects to ground with probability $P(X)P(\overline{X})=0.25$ (for a uniformly distributed X). In reality, the correlated nature of X and $\overline{X}$ precludes Y from ever connecting to ground. The difficulty of accounting for correlations turns out to be a recurring problem for probabilistic estimators. Indeed, the issue will arise again when we consider gate-level probabilistic estimation tools in Section 3.2. Another weakness of LTIME is that it assumes that all nodes make only a single transition each clock cycle and, thus, does not account for glitching in static circuits. These restrictions are fairly severe and limit the accuracy that can be achieved with a tool such as this.

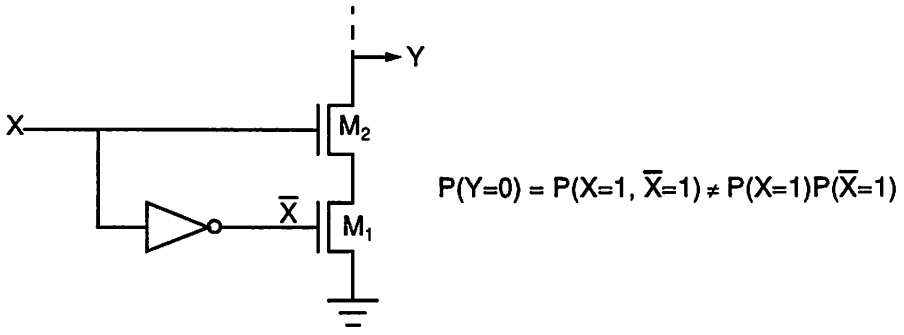


Figure 3-5 : Effect of reconvergent fan-out on probability calculations

Switch-Level Summary

Switch-level modeling sacrifices some of the accuracy associated with device simulators for a significant improvement in speed. The techniques at this level can be either deterministic (e.g. simulation-based) or probabilistic strategies. In either case, the improvement in speed may still not be enough to make feasible simulation of chips with many thousands (or millions) of transistors. In addition, system memory requirements for such large simulations add another practical limitation to the size chip that can be analyzed in practice.

3.1.3 Circuit-Level Summary

In summary, circuit-level power estimators can be divided into two levels of modeling accuracy. The most detailed are the device-level simulators. SPICE, the most well-known of these, is also the most accurate - modeling all important power consumption mechanisms including leakage, short-circuit, and charging currents, as well as non-linear device currents and capacitances. Unfortunately, this accuracy comes at the price of long simulation times (hours to days) for reasonable size circuits (hundreds to thousands of transistors). Simplifying the device models (e.g. CAzM, PowerMill, TSIM) or employing a hierarchical simulation strategy (PRITI) can improve the simulation times, as can moving from a device- to a switch-level transistor model.

At the switch-level, event-driven simulation, two-level voltage quantization, and linear device models can improve the speed by two to three orders of magnitude. Examples of switch-level tools for power simulation include IRSIM and LDS. The linear RC models used by most switch-level tools, however, do imply some sacrifice in accuracy, but with proper calibration to SPICE parameters, errors can be limited to about 10-15%. While fast compared to SPICE, even switch-level tools are too slow at chip-level complexities to be particularly useful for power optimization. It is possible to improve simulation times by moving from a simulation approach to a probabilistic approach. Tools following this strategy (e.g. LTIME) make a single pass through the circuit, propagating switching probabilities to all internal nodes. These switching probabilities can then be used to determine average power consumption independent of the input patterns.

Although probabilistic techniques may bring the speed of the power estimation process up to a reasonable level for some large circuits, they still rely on a transistor-level netlist as input. As a result, *the entire design must be completed* before the tools can be applied. This precludes the use of these tools for power optimization at the architecture or system levels, which were shown in Chapter 2 to be where the largest wins were possible. So in some sense, the issue of speed is beside the point. No matter how fast a circuit-level power estimation tool is, a designer will not use it to decide whether, for example, a RISC or CISC architecture is better for low-power. Nevertheless, transistor-level tools are currently the main source of power estimates for chip designers. This is primarily due to a lack of good higher level tools, although some progress has been made at the gate level as we shall see in the next section.

3.2 Gate-Level Tools

While traditional power analysis techniques focus on the circuit level, in the past few years several gate-level tools for power estimation have arisen. A gate or stage (as it is sometimes called) is usually defined as a channel-connected subnetwork of transistors. Typically, a stage consists of a complex CMOS gate with all the pass transistors following it. This section will

analyze three basic strategies for gate-level power estimation. The various estimation strategies are based on equivalent inverter reductions, precharacterized cell libraries, and boolean logic network evaluation. For each of these basic categories several tools will be presented. As before, some tools will be simulation-based, but the majority rely on probabilistic approaches. As with the probabilistic circuit-level tools, the issue of correlations due to reconvergent fan-out and feedback will present the primary difficulty for these techniques.

3.2.1 Equivalent Inverter Reduction

Some techniques estimate the power a gate consumes by reducing it to an equivalent inverter representation. Internal node capacitances are typically lumped at the output along with wiring capacitance, and the power of the resulting inverter is then analyzed. This section presents four techniques for power analysis that all rely in some sense on reducing gates to equivalent inverters for power analysis. The individual techniques differ primarily in how they account for the switching activity of the gate. Some ignore the activity issue by calculating the worst-case power regardless of the gate inputs. Others attempt to account for signal and transition activities through probability propagation techniques.

Worst-Case Power Consumption

Rather than average power consumption, some tools, like Tyagi's Hercules, focus on finding an upper bound for power [Tya87]. For a given input transition, Hercules estimates the worst-case average and the maximum current flow. Although Hercules takes transistor-level netlists as input, the tool immediately partitions the netlist into stages (or gates) to reduce the complexity of the current estimation process. Starting from the transistor netlist specified by the user, Hercules uses a depth-first search to identify all stages that can be affected by the given input signal. Reducing each stage to an equivalent inverter, Hercules calculates worst-case peak and average currents based on a SPICE-generated database indexed by rise/fall time ratios. Short-circuit currents are modeled with additional SPICE tables, also indexed by rise time ratio. Since each stage is treated

independently and computations are done without regard to actual signal values, Hercules can only give worst-case peak and average values. This leads to fairly inaccurate results with overestimates of peak and average power by as much as 100% and 25%, respectively.

Improving Upper Bounds

We can improve upon the upper bounds provided by the Hercules strategy by considering which input transitions specifically will lead to the largest currents. CEST is an attempt to achieve this end [Cho90]. Like Hercules, CEST divides the transistor netlist into stages. A branch-and-bound algorithm is used to determine the input transitions that lead to worst-case currents. The complexity of such an algorithm grows rapidly with the number of inputs, so heuristics are applied to solve the problem approximately but efficiently.

As with Hercules, the currents in each stage are estimated by reducing to an equivalent inverter and looking up data provided in SPICE-generated tables. The current estimation portion of CEST processes about 100 gates per minute on a VAX-750. Combining this with the branch-and-bound heuristics for determining worst-case inputs increases run times to about one gate per minute for a 100 gate circuit with 14 inputs. Thus, the more accurate results of CEST come at the expense of longer run times.

Probabilistic Simulation for Expected Current Waveforms

The previous two techniques find the worst-case power consumption that could be caused by a single input transition. In some cases it is desirable to produce an estimate of the entire current waveform that a circuit might exhibit over time. CREST produces just such an *expected current waveform* [Naj90][Naj91b]. At each instant in time, this waveform is the weighted average of all possible current values that could occur at that time. In other words, different possible input streams will lead to different current consumption waveforms. The expected current waveform is just the average of these individual waveforms, each corresponding to a distinct input waveform

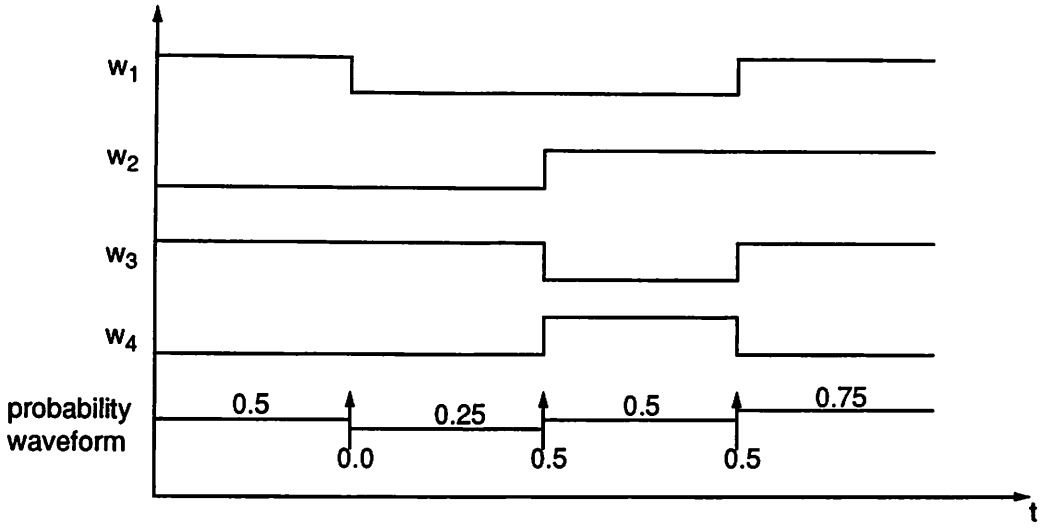


Figure 3-6 : Interpretation of probability waveforms (after [Naj90])

possibility. CREST also produces a *current variance waveform*, which gives a better indication of peak current than the expected current alone.

Rather than exhaustively simulating for all 2^{2n} input transitions of an n input circuit, CREST uses probabilistic simulation to directly derive the current waveforms in a single simulation run. CREST accomplishes this by simulating the circuit for input *probability waveforms* rather than explicit input waveforms. For example, the four explicit waveforms of Figure 3-6 can be represented by a single probability waveform, where the static waveform values represent signal probabilities and the waveform edge weights describe zero-to-one transition probabilities.

Given the input probability waveforms, event-driven simulation is used to derive the probability waveforms for all internal nodes. As with the previous two tools, the input netlist is transistor level, but is immediately partitioned into stages or gates. A probabilistic event appearing at a gate input triggers the evaluation of the mean and variance of the expected output current pulse (which is approximated as triangular). As with the previous tools, this evaluation is based on an equivalent inverter reduction of the gate. Unlike those tools, however, the size and duration of

the current pulse are derived from analytical expressions rather than from table lookup.

As mentioned above, handling temporal and spatial correlations can be a problem for probabilistic tools. CREST accounts for the temporal correlation between successive signal values through the notion of transition probabilities. Signal correlations separated by more than one time step are not accounted for. To some extent, CREST also handles spatial correlations between pairs of signals. For this it employs the notion of the *supergate* [Set85]. The idea is to encapsulate correlated signals within a tightly coupled group of gates classified as a supergate. The inputs to the supergate are assumed independent and exhaustive simulation is performed on its component gates to account for internal correlations explicitly. The exhaustive nature of this simulation sets a practical limit of about 10 on the number of gates within a supergate.

The developers of CREST report errors of 10-20% relative to SPICE with speedups of two to three orders of magnitude. The largest circuit reported was a 34-bit ALU with 1839 transistors for which CREST required 145 seconds on a VAX-11/780 (594x faster than SPICE). If a single current estimate rather than a waveform is sufficient, the following technique can be used to provide faster results.

Effective Frequency Propagation

Whereas the previous approach relies on waveform simulation to predict power consumption, Martinez presented a methodology for power estimation based on direct propagation of an activity measure he refers to as effective frequency [Mar89]. The paper focuses on estimating rms rather than average currents, although the concepts are applicable to both instances. The focus on rms currents stems from the fact that Martinez was primarily interested in evaluating electromigration effects in metallization networks, and rms current is a direct measure of energy flow in wiring.

The rms currents are estimated as a function of transistor width, output load, and frequency. The current waveform during the switching of a CMOS gate is assumed to be triangular. This

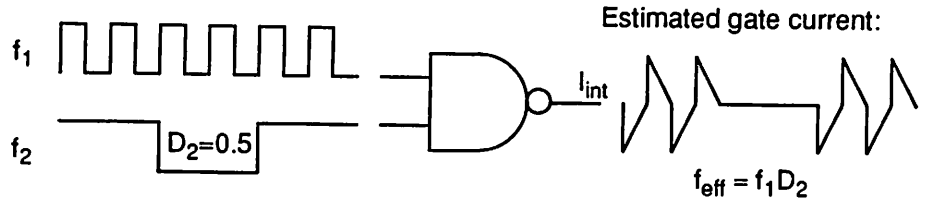


Figure 3-7 : Effective frequency propagation for a two-input NAND (after [Mar89])

leads to the following approximation for rms current flow during a full cycle (charging and discharging) of gate operation:

$$I_{RMS} = \sqrt{I_{NRMS}^2 + I_{PRMS}^2} \quad \text{(EQ 24)}$$

$$I_{NRMS} = R_n I_{np} \sqrt{(t_f f)/3} \quad \text{(EQ 25)}$$

$$I_{PRMS} = R_p I_{pp} \sqrt{(t_r f)/3} \quad \text{(EQ 26)}$$

where f is an effective switching frequency and R_n and R_p are empirical SPICE-based correction factors. The peak currents (I_{np} and I_{pp}) as well as the rise and fall times (t_r and t_f) can be derived from circuit parameters such as device dimensions and transconductance, $k'W/L$, and loading capacitance, C:

$$I_{np} = V_{dd}^2 k_n' (W/L)_n, \quad I_{pp} = V_{dd}^2 k_p' (W/L)_p \quad \text{(EQ 27)}$$

$$t_f = \frac{V_{dd} C}{I_{np}}, \quad t_r = \frac{V_{dd} C}{I_{pp}} \quad \text{(EQ 28)}$$

Complex gates with series and parallel transistors are handled by reducing to an equivalent inverter with an effective W/L values. For example, series transistors have an effective $W/L = [(L/W)_1 + (L/W)_2]^{-1}$ while parallel transistors reduce to $W/L = (W/L)_1 + (W/L)_2$.

The above equations determine current flow during switching events. It is also necessary to determine how often these events occur. For this, Martinez employs a technique he refers to as propagation of effective frequency. Figure 3-7 depicts the frequency propagation mechanism for a

NAND gate. The effective output frequency is generalized as $f_{eff} = f_1 D_2 \ldots D_n$, where f_1 is the highest input frequency and the D_i are duty cycles of the remaining inputs. Similar equations can be derived for other logic gates.

The main source of error in this technique relates to the independence assumption inherent in the frequency propagation strategy. In particular, correlations between input signals can lead to significantly lower output frequencies than predicted by the simplified formulas discussed above. For example, consider a NAND gate with inputs A and $\overline{A}$ both with frequency f_1 and 50% duty cycles. In this case, the equations will predict an output frequency of $0.5f_1$ when the actual frequency should be zero. So we see again that correlation can lead to erroneous activity estimates.

Equivalent Inverter Reduction Summary

Equivalent inverter techniques can be used to estimate power consumption at the gate level of abstraction. The strategy applies to the estimation of maximum or worst-case current, as well as average or typical currents. Deriving circuit activity is the main difficulty and can be accomplished through simulation or probability propagation.

Notice that although the above approaches use partitioning to produce a gate-level representation of the circuit, they still require a transistor-level netlist as input. Furthermore, reducing a gate to an equivalent inverter involves several approximations. For example, lumping the internal capacitances at the output assumes that internal nodes are always completely charged or discharged, which is rarely the case. A more accurate picture of gate power consumption can be formed using a precharacterized library based approach to power estimation.

3.2.2 Precharacterized Cell Libraries

The tools of the previous section took transistor-level netlists as input and then partitioned them into gates or stages prior to power analysis. If, instead, the designer selects gates directly from a library of pre-existing cells, the power analysis process can be simplified and made more

accurate. Under this paradigm gates can be characterized prior to power analysis. Then, the power analysis tool need only perform a number of table lookups during the actual estimation process. This section describes two variations of this basic approach.

Reference Waveform Extrapolation

One possible strategy is to simulate a design at the gate level and then extrapolate an expected current from reference waveforms stored in a hardware database. SIMCURRENT takes this approach, relying on two databases describing various properties of precharacterized library cells [Jag90]. The first database contains SPICE-derived current waveforms for two reference gates - one inverting and one non-inverting (e.g. NAND and AND). This database contains waveforms for various output loading conditions, $C_{L(ref)}$. The second database contains entries for each gate in the cell library. Specifically, for each gate the database contains an entry for the switching capacitance of the unloaded gate. This intrinsic gate capacitance, C_{sw}, can then be used to weight the reference-gate current waveforms from the first database to produce an estimated waveform for the gate under consideration. The calculation is decomposed into capacitive load and short-circuit current components as follows:

$$I_{load} = \left(I_{ref}|_{loaded} - I_{ref}|_{unloaded} \right) \frac{C_L}{C_{L(ref)}} \quad \text{(EQ 29)}$$

$$I_{sc} = I_{ref}|_{unloaded} \frac{C_{sw}}{C_{sw(ref)}} \quad \text{(EQ 30)}$$

where C_L includes loading due to other gates and to wiring capacitance.

Since the short-circuit current component is contained in the reference waveforms, it must be subtracted out in (EQ 29) when computing the load current. This approach gives about three orders of magnitude improvement in speed over SPICE with results within 10%.

Reference Energy Extrapolation

PowerPlay operates on principles similar to SIMCURRENT, but differs in the assumed shape of the estimated current waveforms [Kro91]. SIMCURRENT bases its waveform shapes on a database of reference waveforms from SPICE simulation. PowerPlay uses reference *energies* and assumes a rectangular shape for the current pulse. As a result, PowerPlay still gives good average current results, but does not produce as accurate an instantaneous current waveform as SIMCURRENT. PowerPlay is, however, about 10x faster than SIMCURRENT and four orders of magnitude faster than SPICE.

Precharacterized Cell Libraries Summary

SIMCURRENT and PowerPlay are both logic simulators with extensions to allow power analysis. Both tools assume that gates in the design have been taken from a library of available cells. For each cell, a database entry accurately describes the gate's power consumption and is usually derived from SPICE simulation of either the gate itself or a reference gate. The resulting tools are more accurate than equivalent inverter approaches, but have the limitation that arbitrary transistor networks are not allowed since gates must be selected from a library of possible cells. Of course, this library can be extended as new gates are designed.

SIMCURRENT and PowerPlay are both power simulators. Given specific input waveforms, they produce estimated current waveforms. This requires simulation over a long sequence of transitions. Execution time can be reduced by using a probabilistic approach to power estimation. In this scenario, the entire input sequence is characterized by a single set of input transition and signal probabilities. A probabilistic tool can then propagate these input probabilities throughout the circuit and produce an average power estimate in a single pass. SIMCURRENT and PowerPlay cannot employ this approach because they do not know the functionality of the component gates a priori (i.e. users are free to add gates of arbitrary function to the library). This means that the tools cannot know how probabilities should be propagated through the gates. If we

restrict the library elements to a small number of known gates - e.g. the common boolean logic functions: NOT, (N)AND, (N)OR, X(N)OR, etc. - then we open the possibility of applying the faster probabilistic estimation techniques.

3.2.3 Boolean Logic Networks

Whereas the previous gate-level tools depended on transistor-level representations either implicitly (library-based) or explicitly (equivalent inverters), power estimation tools based on boolean logic representations truly operate at the gate level of abstraction. The main advantage of restricting the input to boolean gates is that gate-level probability propagation formulas can be defined in a relatively compact way since the possible gate functions are very well defined. This also simplifies the task of dealing with reconvergent fan-out and feedback since the situations giving rise to signal correlations will be more limited as well. Also, the approximation of lumping internal capacitances at the output node is, in general, more accurate for basic gates than for complex gates.

Deterministic Boolean Simulation

One method of estimating average power is to simulate the boolean network while keeping track of switching on capacitive gate outputs. ENPOWER is an example of a tool that uses this deterministic approach to power analysis [Kal94]. Since ENPOWER is a simulator it does not suffer as probabilistic tools do from inaccuracies relating to temporal and spatial signal correlations. The downside is that simulation must proceed over enough cycles to ensure convergence of the power estimates. The exact length of this convergence time will vary from circuit to circuit, increasing with the number of reconvergent fan-out and feedback paths. The tool provides a mechanism for automatically terminating a simulation after a desired convergence criterion has been reached.

Power consumption is estimated based purely on capacitive charging considerations:

$$P = \frac{1}{2}\left(\sum_i C_i T_i\right) V_{dd}^2 \quad \text{(EQ 31)}$$

where C_i is the total parasitic capacitance of gate i (including estimated wiring capacitance) lumped at its output and T_i is the output transition activity for gate i. The results from ENPOWER are typically within 10% of the switch-level tool IRSIM.

Transition Density Propagation

We can also estimate transition counts without simulating by directly propagating *transition density* through the network. The DENSIM power estimator adheres to this strategy [Naj91a]. Najm defines transition density of a circuit node as its "average switching rate" - that is, the average number of nodal transitions per unit time. If a module has outputs, y_i, and inputs, x_i, then the formulas for propagating transition density through the module are as follows:

$$D(y_j) = \sum_i P\left(\frac{\partial y_j}{\partial x_i}\right) D(x_i) \quad \text{(EQ 32)}$$

$$\text{where } \frac{\partial y}{\partial x} = y|_{x=1} \oplus y|_{x=0} \quad \text{(EQ 33)}$$

In this formula $\frac{\partial y}{\partial x}$ is one if a transition at x will cause a simultaneous transition at y. The signal probability of this propagation indicator is then used to weight the density of input transitions.

This model ignores correlations among the module inputs, which are assumed independent. Correlations within a module are modeled, so making modules larger can increase accuracy. The boolean differences of (EQ 33) are calculated using binary decision diagrams (BDD's) [Bry86], which tends to limit the practical size of modules. Also, gates within a module are taken to be zero-delay and this can introduce an underestimate of transition density since glitching will not be taken into account. Furthermore, input transition times are assumed independent and, thus, the output transition density tends to be overestimated since in real circuits inputs often transition almost simultaneously. Overall, DENSIM gives results within 20-30% of logic simulators such as ENPOWER, but at a 10-1000x improvement in simulation speed.

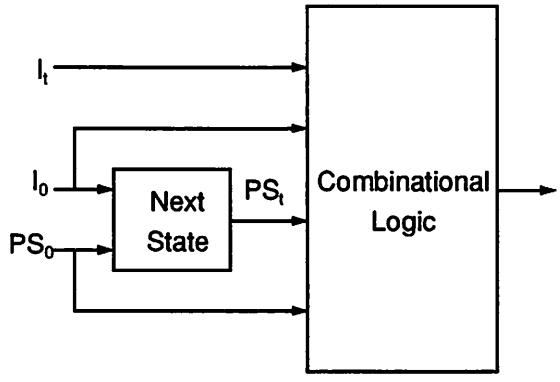


Figure 3-8 : Gate-level power estimation for sequential circuits

Symbolic Simulation under a General Delay Model

Ghosh attempts to improve upon DENSIM by introducing a general delay model that allows him to account for glitching within a probabilistic framework [Gho92]. In addition, his tool correctly treats correlations introduced by reconvergent fan-out and feedback paths in sequential circuits. The tool operates by constructing symbolic boolean equations for transitions at each node in the network based on the previous and current values of the primary inputs: $<I_0, I_t>$. Glitches are accounted for by performing a symbolic simulation of the boolean equations under a general delay model. Correlations between next-state, *NS*, and present-state, *PS*, lines in sequential circuits are handled by prepending the next-state logic in front of the symbolic simulation equations as in Figure 3-8. All primary inputs are assumed uncorrelated, however, amounting to the assumption that arbitrarily long after start-up all states are equally likely.

Though not as fast as DENSIM, Ghosh's tool is still 2-100x faster than gate-level power simulators and gives results well within 10% of those tools. The use of BDD's tends to limit the size of circuits that can be analyzed. This is a common limitation of many gate-level probabilistic estimators.

Event-Driven Probabilistic Simulation under a General Delay Model

The symbolic simulation approach requires building of symbolic equations and BDD's for each module in the network in terms of the primary inputs. This leads to excessive memory requirements. To avoid this we can consider using an event-driven probabilistic simulation approach as proposed by Tsui [Tsu93]. In this approach, Tsui borrows the notion of probability waveforms from CREST (see above). The simulator then propagates probabilistic transition events and signal probabilities through the logic network. Tsui improves on CREST by keeping track of correlations between reconvergent signals. Like Ghosh, Tsui also addresses the issue of glitching by adopting a general delay model. The result is a tool that is about 10x faster than Ghosh's technique while introducing an additional error typically well below 5%.

Boolean Logic Network Summary

The tools presented in this section take a boolean logic description of a network as input. No transistor-level implementation is implied, so different technology mappings can be represented simply by altering the size of the physical capacitances lumped at the gate outputs. The user must recognize, however, that this lumped capacitance is an approximation that may be fairly inaccurate for circuits where the internal nodes don't fully charge or discharge with all output transitions.

3.2.4 Gate-Level Summary

Several tools for gate-level power analysis have recently become available. Some of these tools require a transistor-level circuit description as input. This network is then partitioned into a gate-level representation. Power analysis proceeds by reducing the gates to equivalent inverters for which power consumption can be easily estimated. The requirement of a transistor netlist as input to a large extent defeats the purpose of gate-level tools since the design must be complete down to the circuit level before the tools can be applied.

Other techniques allow the user to specify the network at the gate level using elements from a

precharacterized cell library. Since the utilized gates are characterized individually this approach can be fairly accurate. One drawback is that if we don't restrict the types of cells that can be included in the library then the functionality of the library elements cannot be assumed a *priori* and, consequently, probability propagation techniques are difficult to apply. Instead, tools are forced into the slower simulation strategy.

By restricting the gate selection to the common boolean logic functions, we facilitate probability propagation techniques. In addition, separating the gate-level description from any assumed transistor-level implementation allows the tools to be applied to several technology mappings by merely changing the lumped physical capacitances at the gate outputs.

Just like circuit-level power estimators, the gate-level tools are useful for certain purposes. Most notably, for parts of a chip that will be designed using logic synthesis techniques, the gate-level is probably the appropriate level of abstraction. The gate level might also be appropriate for comparing various adder structures that can be described by a gate-level netlist (e.g. ripple-carry vs. carry-lookahead) as was done in Section 2.7. In short, any gate-level power optimization strategy is most easily analyzed with a gate-level power estimation tool.

Recall, however, that Chapter 2 made it clear that optimization at the circuit and gate levels of abstraction offered limited improvements (<2x) in power. Therefore, while gate- and circuit-level power estimation tools are useful for certain purposes, they will not allow designers to achieve the large, global power reductions that high-level tools can facilitate. So, ideally, CAD tools should support architectural and algorithmic design space exploration. Gate- and circuit-level tools, which require a completed design as input, do not satisfy this criterion.

3.3 Architecture-Level Tools

In the terminology of this thesis, *architecture* refers to the register-transfer (RT) level of abstraction, where the primitives are blocks such as multipliers, adders, memories, and controllers.

Power estimation at this level offers two primary advantages over lower level tools. First, the input is a register transfer level (RTL) description of the design. This implies that the designer can begin to obtain power estimations and make optimizations and trade-offs very early in the design flow. Second, the functional blocks in an RTL description are of a large enough granularity that the complexity of an architectural description (even for a large chip) is relatively low compared to the corresponding gate- or circuit-level descriptions. This allows architecture-level tools to operate much faster than lower level tools.

This section presents two primary techniques for architectural power analysis. The first technique is based on the concept of gate equivalents and two tools using this approach will be discussed and compared. The next technique is a precharacterized cell library approach similar to that discussed at the gate level, but here the library contains adders and multipliers, for example, rather than OR's and AND's. While an important step toward high-level power analysis, these approaches have several weaknesses which limit their usefulness and accuracy. The following sections will discuss the advantages and disadvantages of each technique.

3.3.1 Gate Equivalents

The complexity of a chip architecture can be described roughly in terms of gate equivalents (GE's). The gate-equivalent count specifies the average number of reference gates (e.g. 2-input NAND's) that are required to implement a particular function (e.g. 16-bit counter). The power required for that function can then be estimated by multiplying the approximate number of gate-equivalents required by the average power consumed by each gate. Several architectural power analysis strategies follow this basic approach. Two specific examples will be described here.

Class-Independent Power Modeling

The first approach is to estimate chip area, speed, and power dissipation based on information about the complexity of the design in terms of gate equivalents making no distinction about the

functionality of different blocks (i.e. class independent). This is the technique used by the Chip Estimation System (CES) [Mül91]. The chip is first broken down into blocks such as counters, decoders, multipliers, memories, etc. Each of these functional blocks is then assigned a complexity in terms of Gate Equivalents (GE's). The number of GE's for each unit type can be specified in a library or provided by the user. Each gate equivalent is assumed to dissipate an average energy, E_{typ}, when active. The activity factor, A_{int}, describes the average percentage of gates switching per clock cycle and is allowed to vary from function to function. The result is the following estimate for chip power consumption:

$$P = \sum_{i \in \{fn's\}} GE_i (E_{typ} + C_L^i V_{dd}^2) f A_{int}^i \quad \text{(EQ 34)}$$

The capacitive load, C_L, contains components due to fan-out loading as well as wiring. The wiring capacitance can be obtained from an estimate of the average wire length. This is provided by the user and cross-checked by using a derivative of Rent's Rule [Lan71].

This technique for power estimation relies on several approximations. First, all power estimates are based on the energy consumption of a single reference gate. This does not take into account different circuit styles, clocking strategies, or layout techniques. Moreover, activity factors are assumed to be fixed regardless of the input patterns. The typical gate switching energy is characterized based on the assumption of a completely random uniform white noise (UWN) distribution of the input data. This UWN model ignores how different input distributions affect the power consumption of gates and modules. The result is the same estimate of average power regardless of whether the circuit is idle or at maximum activity. Clearly, this is a very rough approximation.

Class-Dependent Power Modeling

Svensson and Liu developed another technique for architectural power analysis that employs a similar modeling strategy [Sve94]. Their approach attempts to improve the modeling accuracy by

applying customized estimation techniques to the different types of functional blocks: logic, memory, interconnect, and clock.

The logic component of power is estimated in a manner quite similar to CES. The basic switching energy is based on a three-input AND gate and is calculated from technology parameters (e.g. gate width, t_{ox}, and metal width) provided by the user. The total chip logic power is estimated (as before) by multiplying the estimated gate equivalent count by the basic gate energy and the activity factor. The activity factor is provided by the user and assumed fixed across the entire chip.

The memory power computation is divided into a separate calculation for the cell array, the row decoders, the column decoders, and the sense amps. The power consumption of each of these components is then related to the number of columns (N_{col}) and rows (N_{row}) in the memory array. Under most conditions, the precharging of the bit lines is found to dominate the power consumption and is given by:

$$P_{bitlines} = \frac{N_{col}}{2}(L_{col}C_{wire} + N_{row}C_{cell})\,V_{dd}V_{swing} \qquad \text{(EQ 35)}$$

where C_{wire} is the bit line wiring capacitance per unit length and C_{cell} is the loading due to a single cell hanging off the bit line.

As in CES, the interconnect length and capacitance is modeled by Rent's Rule. The clock capacitance is based on the assumption of an H-tree distribution network. Activity is modeled using a UWN model.

Like CES, this tool requires very little information in order to produce an estimate of chip power. Basically, just a few technology parameters, the chip area, and the count of gate equivalents are required. The disadvantage is that the estimate does not model circuit activities accurately and is, therefore, fairly rough. An overall activity factor is assumed for the entire chip and, in fact, must be provided by the user. In reality, activity factors will vary throughout the chip. So even if

the user provides an activity factor that results in a good estimate of the total chip power, the predicted breakdown of power between modules is likely to be incorrect. This is evidenced by Svensson and Liu's analysis of the DEC Alpha chip [Dob92]. The chosen activity factor gives the correct total power, but the breakdown of power into logic, clock, memory, etc. is less accurate. Thus, this tool has many of the same weaknesses that limit CES.

3.3.2 Precharacterized Cell Libraries

Svensson and Liu use a separate power model for logic, memory, and interconnect. Powell and Chau took this decomposition one step further suggesting a Power Factor Approximation (PFA) method for individually characterizing an entire library of functional blocks [Pow90]. Instead of a single gate-equivalent model for "logic" blocks, the PFA method provides for different types of "logic" blocks such as multipliers, adders, etc.

The power over the entire chip is approximated by the expression:

$$P = \sum_{i \in \{all\ blocks\}} \kappa_i G_i f_i \qquad \text{(EQ 36)}$$

where each functional block i is characterized by a PFA proportionality constant κ_i, a measure of hardware complexity G_i, and an activation frequency f_i. The paper considers three specific examples: multipliers, I/O drivers, and memories.

The hardware complexity of the multiplier is related to the square of the input word length, N^2. The activation frequency is simply the frequency with which multiplies are performed by the algorithm, f_{mult}. Finally, the PFA constant, κ_{mult}, is extracted empirically from past multiplier designs (taken from ISSCC proceedings) and shown to be about 15 fW/bit^2-Hz for a 1.2 μm technology at 5V. The resulting power model for the multiplier is:

$$P_{mult} = \kappa_{mult} N^2 f_{mult} \qquad \text{(EQ 37)}$$

Similar formulas can be derived for other functional blocks.

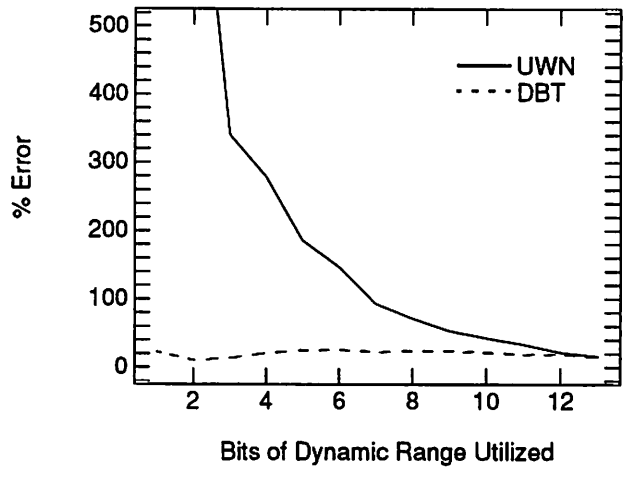


Figure 3-9 : Errors in modeling power of 16x16 array multiplier

The PFA method improves upon the previous techniques in that it provides a general strategy for characterizing different functional blocks. The power models can be parameterized in terms of whatever complexity parameters are appropriate for that block. For example, in the case of the multiplier the square of the word length was appropriate. For the memory, the storage capacity in bits is used and for the I/O drivers the word length alone is adequate.

The weakness of the model again relates to the way it models activity. The PFA constant κ_{mult} is intended to capture the intrinsic internal activity associated with a multiply operation. Since this is taken to be a constant, there is the implicit assumption that the inputs do not affect the multiplier activity. As mentioned in the previous section this is rarely the case. It being impossible, however, to characterize the module for all possible input streams, purely random inputs - that is to say, independent uniform white noise (UWN) inputs - are typically applied when deriving the PFA constant κ.

This is the chief source of error in existing architectural power modeling techniques such as the PFA method. Figure 3-9 displays the estimation error (relative to switch-level simulation) for a

16x16 multiplier. Clearly, when the dynamic range of the inputs doesn't fully occupy the word length of the multiplier, the UWN model becomes extremely inaccurate. Granted, good designers attempt to maximize word length utilization. Still, errors in the range of 50-100% are not uncommon [Lan93][Lan94]. In any case, the figure clearly suggests a flaw in the UWN model. The Dual Bit Type (DBT) model, presented in Chapter 4, addresses this deficiency and, as a preview of later results, its modeling error for the multiplier has also been included in Figure 3-9.

3.3.3 Architecture-Level Summary

This section has presented existing techniques for architecture-level power analysis. The first class of techniques operates based on a complexity measure called gate equivalents. Each function is characterized by a gate-equivalent count, which describes the approximate number of reference gates (e.g. 2-input NAND's) required to implement the function. The average power for the chip is then estimated by multiplying the gate-equivalent count by the average power consumed per gate. This technique gives fairly rough estimates as it does not take into account anything about the internal construction of the unit. In addition it assumes a white noise activity model, which may not reflect the actual signal activity. The second class of techniques is based on precharacterized libraries of RTL modules. Since each module is characterized individually, the power estimates can be much more accurate. This requires, however, that all functional blocks be taken from a pre-existing library. In addition, the blocks are still characterized using uniform white noise inputs, so activity factors will still be inaccurate. The main weakness, then, of existing architectural power analysis strategies is that they do not accurately model the activity stemming from actual data signals. Instead, they assume purely random activity factors that can give inaccurate results. If this is the case, we might expect it to be even more difficult to get good power estimates at the algorithm level where even less information is available to the tool. This problem and its current solutions will be discussed in the next section.

3.4 Algorithm-Level Tools

The task of algorithmic power estimation is to predict how much power (on average or as a lower bound) will be consumed by executing a given algorithm. The challenge is that this estimate must be produced with very little knowledge about what architecture or hardware platform the algorithm will be executed upon. This makes the task nearly impossible since the same operation can take vastly different amounts of power when performed on different pieces of hardware. All techniques proposed thus far for algorithmic power estimation operate by making some assumptions about the class of architectures with which the algorithm will be implemented. By narrowing down the class of applications considered (e.g. DSP algorithms), the developers hope to improve the validity of these assumptions. With some basic architecture assumed, algorithmic power estimation reduces to architectural power analysis using the same gate-equivalent and library-based techniques.

3.4.1 Gate Equivalents

In [War84], Ward et al. took on the problem of estimating the power consumed by DSP algorithms using a gate-equivalent power model. The basic approach is to consider only those operations that are likely to dominate the power for DSP applications. Specifically, Ward considers ALU operations, multiplications, background/foreground memory accesses, and multiplexing.

For each of these operations, Ward assumes an approximate gate-equivalent count for the hardware implementation:

$$GE_{ALU} = 16N \quad \text{(EQ 38)}$$

$$GE_{mult} = 10N^2 + 30N + 80 \quad \text{(EQ 39)}$$

$$GE_{RAM} = 0.3WN, \quad GE_{ROM} = 0.03WN \quad \text{(EQ 40)}$$

$$GE_{latch} = 10N \quad \text{(EQ 41)}$$

$$GE_{mux} = 4N + 3 \quad \text{(EQ 42)}$$

where N is the word length of the appropriate operation and W is the number of words stored in the background memory element. After converting algorithmic operations into gate-equivalent counts, power consumption is estimated by multiplying with a technology dependent power per gate conversion factor. For example, Ward quotes 1.5 mW/gate for a 1984 bipolar technology.

This approach allows the designer to make some trade-offs in algorithm selection, but suffers from several drawbacks. First, there is the implicit assumption about exactly how the operations will be implemented in hardware. Moreover, since the estimates are based on the gate equivalent model of power consumption, the technique inherits all the weaknesses of that approach as discussed in the previous section. The model is useful in that it requires very little information in order to get some basic classification of algorithm power consumption; however, it is vital that the user understand its limitations.

3.4.2 Precharacterized Cell Libraries

Just as it is possible to build an algorithmic power estimation strategy on top of the gate equivalent model, it is also possible to envision an approach based on the precharacterized library models discussed previously. This approach has been taken by Powell and Chau [Pow91] (developers of the PFA method). Like Ward, their approach focuses on DSP filtering applications and assumes an underlying architectural model. The algorithm is assumed to be implemented on an array of similar or identical multiply-add (MA) based processing elements (PE's) whose power is dominated by an array multiplier. The other components of power consumption considered are memory and I/O power. The PFA models for these components are coupled with the following algorithmic parameters to yield an overall power estimate for the algorithm:

f_{samp}	= overall I/O sample rate
A	= number of PE's that can fit on a single chip
L	= number of PE's available to implement algorithm
N_s	= number of PE's required to implement algorithm
N_w	= number of words transmitted between PE's
Q	= number of bits in internal word length

C = number of coefficients (multipliers) per PE
S = number of state registers per PE (EQ 43)

Directly applying the PFA method to the three dominant power components using these parameters yields:

$$P_{mult} = \kappa_{mult} C Q^2 N_s f_{mult} \quad \text{(EQ 44)}$$

$$P_{mem} = \kappa_{mem} (C^2 + S^2) \frac{Q N_s^2}{L} f_{samp} \quad \text{(EQ 45)}$$

$$P_{I/O} = \kappa_{I/O} [Q N_s N_w (A^{-1} + L^{-1}) + Q] \quad \text{(EQ 46)}$$

Using these formulas, high-level algorithmic and architectural parameters can be used to analyze trade-offs that will affect power consumption.

As with the Ward technique, the advantage is the ability to get early power estimates while requiring very little detailed information from the user; however, trying to make intelligent estimates of power given so little information is quite difficult. Narrowing the application domain to DSP as both techniques do, simplifies the problem to some extent by allowing some assumptions about the implementation architecture. Still, even within the range of DSP applications there is quite a bit of flexibility in implementation style (e.g. bit-serial, bit-parallel, pipelined, parallel, dataflow, Von Neumann, etc.), so giving a single power number for each algorithm is a little misleading. Designers using these estimation techniques must be aware that the results should only be used to generate a relative (rather than absolute) ranking of algorithms in terms of intrinsic power consumption.

3.4.3 Algorithm-Level Summary

From this discussion we see that current strategies for estimating power at the algorithm level consist of assuming some architecture and applying architectural power analysis techniques. For a very restricted application domain this strategy may produce at least some classification of algorithms in terms of power but, in general, the results can be misleading. For example, even DSP

filtering algorithms can be implemented on many different architectures ranging from off-the-shelf DSP's to dedicated custom hardware, which might all result in very different power consumptions.

A possible solution to this problem might be to have algorithmic estimators for several different target architectures. The results of each could be combined to map out a region rather than a point in the implementation space. For instance, the PE-based tool described here could be used to plot a point in the design space corresponding to a processing-element implementation of the proposed algorithm. Likewise, some other tool, targeted say at bit-serial DSP architectures, could be used to plot an additional point - and so on.

The remaining difficulty is that even within a specific class of architectures there may still be many different ways to implement the same algorithm. For example, it may be possible to implement an algorithm on a single time-shared PE or on several parallel PE's. Both implementations would perform the same task on the same class of architecture, but would very likely consume different amounts of power. The power associated with interconnect would certainly be different. Also, from the discussions of Chapter 2, we know that the parallel architecture might be able to run at a much lower voltage and, thus, at lower power than the uni-processor. Consequently, even when a class of target architectures has been specified it is still difficult to give a single figure of merit for algorithm power consumption.

Once again the solution is to extend the goal of algorithmic power estimation from one of producing a single figure of merit to one of producing a range of area, performance, and power (ATP) figures corresponding to different possible implementations of the algorithm. So in the case of the PE-based architectures, a useful algorithmic ATP exploration tool would produce a continuum of power estimates for particular PE architectures while varying the supply voltage and degree of parallelism. R. Mehra has developed just such a design space exploration tool [Meh94], which will be described further in Chapter 7.

In addition, rather than focusing solely on a single level of abstraction, a truly useful low-

power design environment would contain tools appropriate to each of the various levels of design. Chapter 7 will describe how the architectural power analysis tools described in this thesis can be coupled with the algorithmic power estimation and exploration tools from other researchers to produce just such an environment and methodology for low-power design.

3.5 Chapter Summary

This chapter has presented an in-depth overview of the current state of the art in power estimation. The most accurate power analysis tools discussed were at the circuit level. The majority of these were simulators (e.g. SPICE), which designers have been using for many years as performance analysis tools. Therefore, designers are comfortable with these tools and rely primarily on this level of modeling for evaluating circuit power consumption. Unfortunately, even with switch- rather than device-level modeling, these tools are either too slow or require too much memory to handle large chips.

For this reason, gate-level power estimation tools have begun to gain some acceptance. At this level, faster, probabilistic techniques have begun to gain a foothold; however, this speedup is achieved at the expense of accuracy, especially in the presence of correlated signals. Unfortunately, many custom datapath or memory modules containing complex CMOS networks and pass transistors don't have a straightforward representation in terms of gates. The issue of generality coupled with that of accuracy has prevented gate-level power estimation tools from gaining wide acceptance.

In some sense, this is all a moot point since Chapter 2 tells us that the biggest wins in low-power design do not come from circuit- and gate-level optimizations. Instead, architecture, system, and algorithm optimizations tend to have the largest impact on power consumption. Therefore, tool developers should be focusing on high-level analysis and optimization tools for power. We have seen in this chapter that while there are many tools for circuit- and gate-level

power estimation, there are very few at the architecture level and above. Furthermore, the tools that do exist are subject to weaknesses that limit their accuracy. In particular, the assumption of uniform white noise signals leads to unrealistic estimates of circuit activity. So while these high-level tools have the advantage of being very fast and being useful at a very early stage in the design process, they also have the disadvantage that the results they provide may be somewhat inaccurate. This inaccuracy prevents designers from relying on the information provided to them by high-level tools. As a result, it has been difficult for these tools to gain acceptance.

We have developed a new model for architectural power analysis that addresses these issues. The solution combines the speed of RT-level power estimation with the accuracy more commonly associated with circuit- or gate-level tools. The concept behind this new model will be presented in Chapter 4. This will be followed by Chapters 5 and 6, which describe two distinct implementations of the model - one targeted for DSP applications and one for more general-purpose architectures. Finally, Chapter 7 will describe an integrated CAD environment including tools for analysis and optimization at several levels of abstraction and supporting a top-down methodology for low-power design. This discussion will include a case study that demonstrates how the tools and methodology can be applied to realistic low-power design problems.

CHAPTER 4

Architectural Power Analysis

The previous chapters have demonstrated two important facts. First, low-power design strategies should address high-level issues before dealing with lower level concerns. Chapter 2 made this point clear by demonstrating that circuit- and gate-level techniques typically have less than a 2x impact on power, while architecture- and algorithm-level strategies offer savings of 10-100x or more. The point was also made that power-conscious design space exploration tools are the key to efficient and successful low-power design. Chapter 3 made it clear, however, that power analysis tools are available primarily at the gate and circuit levels, not at the architecture and algorithm levels where they could really make an impact. The architecture-level tools that do exist were shown to have serious limitations, mainly related to inaccurate handling of activities. This chapter addresses these issues, presenting novel techniques for fast and *accurate* architectural power analysis.

Figure 4-1 gives an overview of the power analysis strategy that we propose in this chapter. The inputs from the user are a description of a candidate architecture at the register-transfer level and a set of data and instruction inputs for which a power analysis is desired. Rather than

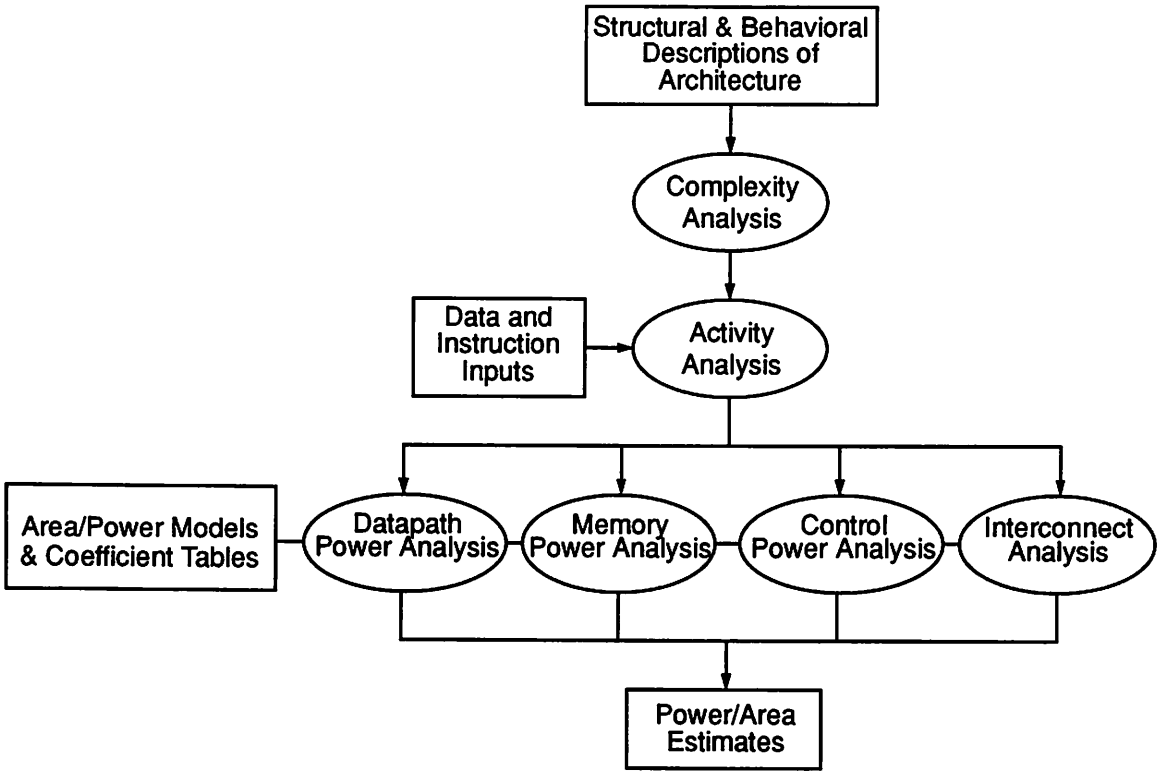


Figure 4-1 : Overview of architectural power analysis strategy

attempting to find a single model for the entire chip, we take the approach of identifying four basic classes of components: datapath, memory, control, and interconnect. The modeling of power consumption for each class is addressed in a separate section of this chapter. For datapath analysis, Section 4.1 introduces a new word-level data model known as the Dual Bit Type (or DBT) model. The model allows a more accurate accounting of activity than the traditional UWN models described in the previous chapter. Indeed, error rates for the DBT method are on the order of 10-15% relative to switch-level simulation, while UWN errors may be 50-100% or more. Section 4.2 extends this model to handle memory elements (e.g. registers, register files, and SRAM's). An architectural model for control path power consumption is presented in Section 4.3. This model handles activity more accurately than the gate-equivalent models of Chapter 3 and is, therefore, referred to as the Activity-Based Control, or ABC, model. Section 4.4 describes techniques for

estimating the physical capacitance of interconnect (while producing chip area estimates as a side effect) and for combining this with the appropriate activity measures to estimate control and data bus power consumption. The chapter concludes with Section 4.5 which brings together the four classes of models, describing how they can be integrated to achieve architectural power analysis for entire chips. This section also describes how the complexity and activity parameters required by the DBT and ABC models can be derived.

4.1 Datapath

The datapath is the part of a chip that performs the numerical computations required by an algorithm. Adders, multipliers, shifters, and comparators are examples of typical datapath modules. This section will describe a technique for analyzing the power consumption of datapath modules at the architecture level. This is done by producing a black-box model of the capacitance switched in each module for various types of inputs. If desired, these capacitance estimates can be converted to an equivalent energy, $E=CV^2$, or power, $P=CV^2f$.

The black-box capacitance models are easily parameterized. For example, the user can specify precisely how the physical capacitance of each module should scale with its "size" or complexity. This allows the model to reflect the fact that a 16-bit adder will contain roughly twice the physical capacitance of an 8-bit adder.

Unlike previous attempts at architectural power analysis, the model accurately accounts for activity as well as physical capacitance. Instead of having a single capacitive coefficient based on a uniform white noise assumption, the model employs several coefficients, each corresponding to a different input type. As a result, the effect of the input statistics on module power consumption is reflected in the estimates. More specifically, the technique accounts for two classes of input bits and, therefore, is referred to as the Dual Bit Type or DBT model.

The following sections describe the DBT model in more detail. Section 4.1.1 tells how

capacitance models can be parameterized to scale with module complexity. Section 4.1.2 describes the capacitive coefficients that allow the model to accurately handle a wide range of input activities. Next, Section 4.1.3 outlines a library characterization strategy for deriving the capacitive coefficients. Section 4.1.4 shows how the model can be applied in practice to analyze datapath power consumption. Finally, Section 4.1.5 presents results demonstrating the flexibility and accuracy of the black-box modeling technique.

4.1.1 Complexity Model

Intuitively, the total power consumed by a module should be a function of its complexity (i.e. "size"). This reflects the fact that larger modules contain more circuitry and, therefore, more physical capacitance. For instance, at a fixed computation rate one would expect a 16x16 multiplication to consume more power than an 8x8 multiplication. Under the DBT model, the user can specify arbitrary capacitance models for each module in the datapath library.

For example, consider modeling the capacitance of a ripple-carry subtracter. The amount of circuitry and physical capacitance an instance of this subtracter will contain is determined by its word length, N. In particular, an N-bit subtracter will be realized by N one-bit full-subtracter cells as shown in Figure 4-2. The total module capacitance should, therefore, be proportional to the word length as shown here:

$$C_T = C_{eff} N \qquad \text{(EQ 47)}$$

In this equation, C_{eff} is a capacitive coefficient, which describes the effective capacitance switched for each bit of the subtracter. A detailed discussion of the capacitive coefficients will be postponed until the section on modeling activity.

Many modules besides the subtracter also follow a simple linear model. For example, ripple-carry adders, comparators, buffers, multiplexers, and boolean logic elements all obey (EQ 47). The DBT method is not, however, restricted to linear capacitance models.

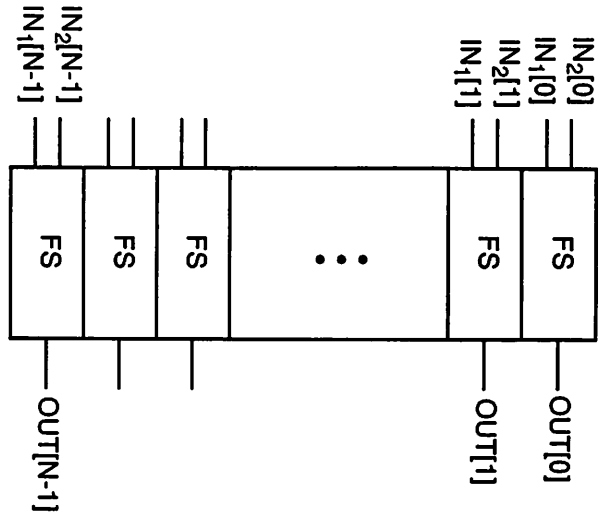


Figure 4-2 : Decomposition of *N*-bit subtracter into bit-slices

As an example of a non-linear model, consider the array multiplier depicted in Figure 4-3. The size of this module grows quadratically with the input word length. Specifically, if the two inputs have word lengths N_1 and N_2, respectively, then the multiplier will require N_1N_2 full-adder cells. We refer to modules in which the inputs are laid out in a grid with a cell at each intersection as a *bit-meshed*, as opposed to *bit-sliced*, module. The appropriate capacitance model for a bit-meshed module is given by:

$$C_T = C_{eff}N_1N_2 \qquad \text{(EQ 48)}$$

The flexibility of the DBT modeling strategy allows the user to specify even more complex capacitance models when necessary. Take the case of the logarithmic shifter depicted in Figure 4-4. This module consists of several stages, each capable of performing successive shifts by powers of two, conditional on the binary value of the SHIFT input. If the unit is capable of performing a maximum shift by M bits, then the shifter will require $L = \lceil \log_2(M+1) \rceil$ stages. The capacitance model for the log shifter is a function of three parameters: the word length, N, the maximum shift value, M, and the number of shift stages, L. The exact capacitance model is given by the following

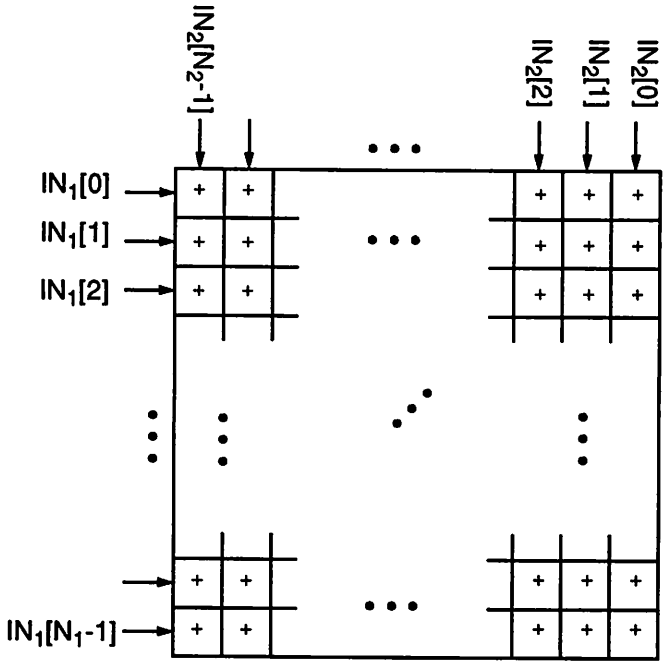


Figure 4-3 : Decomposition of array multiplier into a bit mesh

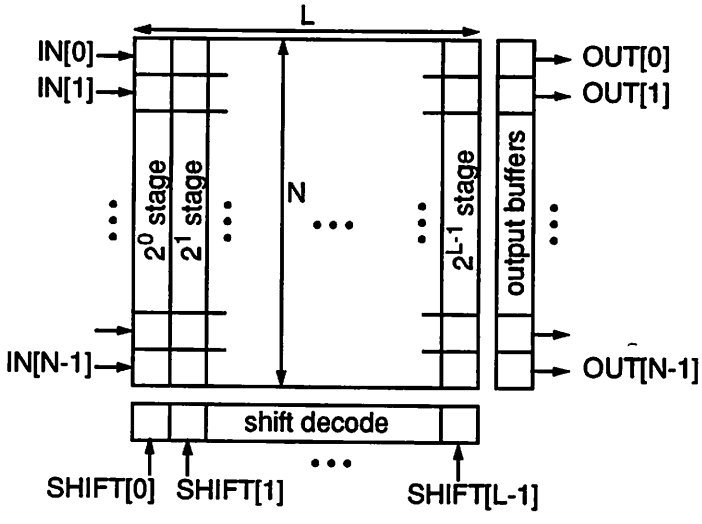


Figure 4-4 : Decomposition of logarithmic shifter

equation:

$$C_T = C_0N + C_1L + C_2NL + C_3N^2L + C_4MNL \quad \text{(EQ 49)}$$

Referring to the figure, the C_0N term reflects the capacitance of the output buffers. Likewise, the C_1L term represents the overhead capacitance of the decoders and drivers for the L shift control signals. Since there are NL cells in the shift array, the capacitance model also contains a C_2NL term. The last two terms correspond to the amount of routing in the array. Since all cells in the array contain inter-bit routing, these two terms are proportional to the total number of cells, NL. This must be multiplied by the average number of wires passing through each of the cells. For shifters where the word length, N, is significantly larger than the maximum shift value, M, the number of wires intersecting most of the cells will be proportional to M. For shifters where N is comparable to M, the number of wires intersecting most of the cells will be N. In summary, since the amount of routing crossing each cell can range between N and M and since there are NL cells, the total capacitance expression includes a term for each case: C_3N^2L and C_4MNL.

Since this complex capacitance model has several terms, it also requires several capacitive coefficients. This does not pose a problem for the DBT model. The solution is to use vector rather than scalar arithmetic. In other words, instead of a scalar capacitive coefficient, C_{eff}, we refer to a capacitive coefficient vector:

$$\mathbf{C}_{eff} = \left[C_0\ C_1\ C_2\ C_3\ C_4\right]^T \quad \text{(EQ 50)}$$

Similarly, rather than a single complexity parameter (e.g. N), we use a complexity vector:

$$\mathbf{N} = \left[N\ L\ NL\ N^2L\ MNL\right]^T \quad \text{(EQ 51)}$$

The result is the following vector capacitance model:

$$C_T = \mathbf{C}_{eff} \cdot \mathbf{N} \quad \text{(EQ 52)}$$

In summary, the capacitance models used by the DBT method are extremely flexible. The user or library developer can specify precisely how various complexity parameters affect the total

capacitance of each module. Typical models range from simple linear expressions involving a single parameter and capacitive coefficient to complex functions of multi-term complexity vectors.

4.1.2 Activity Model

The previous section described how to model the effect of module "size" on capacitance through a complexity parameter, N. In that discussion, N was weighted by an effective capacitance coefficient C_{eff}. This section describes how C_{eff} can be used to model activity, as well as physical capacitance.

The details of any activity model depend on several factors. First, the data representation influences the activity. For example, fixed-point and floating-point data exhibit different activity patterns. Likewise, two's-complement and sign-magnitude data each require slightly different modeling techniques. Signal statistics will also influence model selection. Stationary signals, for instance, must be treated differently than signals whose statistics vary over time. Finally, the activity model also depends on the types of logic families to be accommodated. For example, the activity of dynamic logic families can be described by signal probabilities alone. This is because in dynamic logic all signals are precharged to a known state each clock cycle. This more or less eliminates the influence of the previous signal value on the current power consumption of the dynamic circuit. Since static logic does not return to a known state each cycle, its activity depends on transition probabilities as well.

We will begin by making some initial assumptions about data representation, signal statistics, and logic styles. This will allow us to develop a basic activity model, which can then be extended to other more general forms. Since fixed-point two's-complement is one of the most common data representations we will use this as our foundation. Initially, we will also assume that signal statistics are stationary over time. Finally, since the transition probabilities needed for analyzing static logic contain all the information required to derive signal probabilities, we will develop an activity model capable of handling the more general and difficult static logic case. Dynamic logic

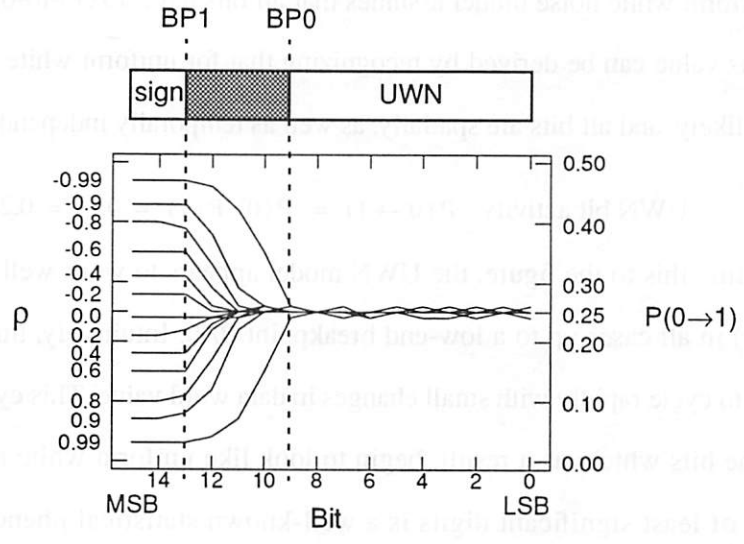


Figure 4-5 : Bit transition activity for data streams with varying temporal correlation

will be included as a subset of this more general model.

The Dual Bit Type Data Model

The key to modeling activity, as well as physical capacitance, is the use of several effective capacitance coefficients, C_{eff}. Traditional architectural modeling techniques use a single coefficient for each module. The coefficient is usually derived for uniform white noise inputs. Therefore, activity due to anything but random input transitions is poorly modeled. The solution is to use many effective capacitance coefficients - one for each type of input transition being modeled.

It remains to be seen what types of data (aside from uniform white noise) should be modeled. Figure 4-5 shows the bit transition activity for several different two's-complement data streams. In particular, each curve corresponds to a gaussian process with a different temporal correlation,

$$\rho = \frac{\mathrm{cov}(X_{t-1}, X_t)}{\sigma^2} \qquad \text{(EQ 53)}$$

where X_{t-1} and X_t are successive samples of the process and σ^2 is the variance.

The uniform white noise model assumes that all bits have a zero-to-one transition probability of 0.25. This value can be derived by recognizing that for uniform white noise, zero and one bits are equally likely, and all bits are spatially, as well as temporally independent. Mathematically,

$$\text{UWN bit activity: } P(0 \rightarrow 1) = P(0)\,P(1) = 0.5^2 = 0.25 \qquad \text{(EQ 54)}$$

Comparing this to the figure, the UWN model appears to work well for the least significant bits (LSB's) in all cases up to a low-end breakpoint *BP*0. Intuitively, this is not surprising. The LSB's tend to cycle rapidly with small changes in data word value. This cycling has a randomizing effect on the bits which, as a result, begin to look like uniform white noise bits. Actually, the uniformity of least significant digits is a well-known statistical phenomenon, and a detailed discussion of the topic is given by Preece in [Pre81].

In contrast, for two's-complement the MSB's down to the high-end breakpoint *BP*1 are sign bits and, in general, their behavior differs markedly from the data bits. Among the sign bits, $P(0 \rightarrow 1)$ represents the probability that the data will transition from a positive to a negative value. As a result, positively correlated signals ($\rho>0$) experience lower activity in the sign region, while negative correlation ($\rho<0$) leads to increased sign activity as shown in Figure 4-6.

As a minor point, the UWN and sign bit regions do not quite cover the entire range of bits. A small intermediate region separates the two constant activity regions. It is not necessary to create a new bit type to model this region. Instead, linear interpolation of the sign and UWN activities models the intermediate region quite well.

The positions of the model breakpoints depend on how many bits are required to represent the numbers in the data stream; the unused bits will be devoted to sign. A discussion of how to derive the breakpoint positions will be presented in Section 4.1.4. For now, suffice it to say that analytical expressions can be found that express the breakpoints of a stationary data stream as a function of word-level statistics such as mean (μ), variance (σ^2), and correlation (ρ):

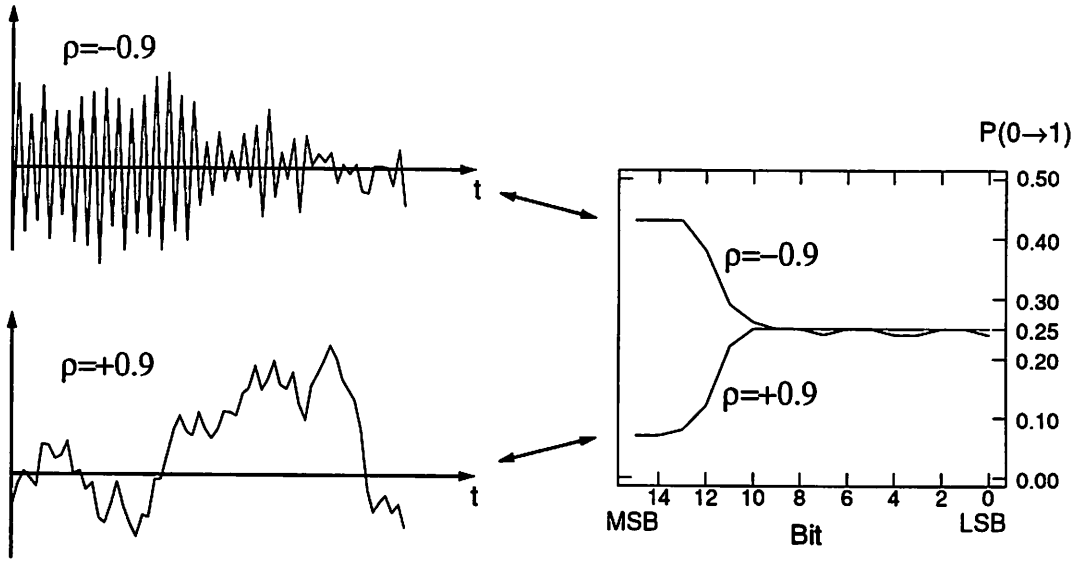


Figure 4-6 : Activity for positively and negatively correlated waveforms

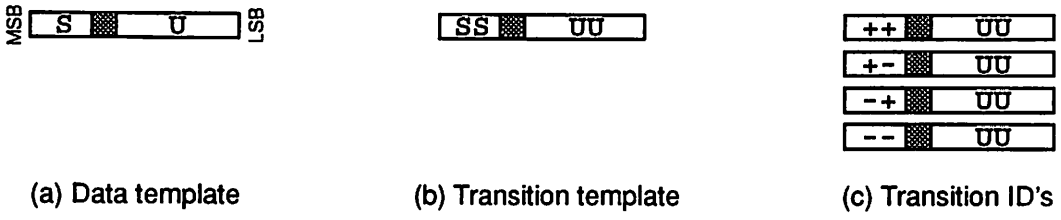
$$BP1 = \log_2(|\mu| + 3\sigma) \quad \text{(EQ 55)}$$

$$BP0 = \log_2\sigma + \Delta BP0 \quad \text{(EQ 56)}$$

$$\Delta BP0 = \log_2[\sqrt{1-\rho^2} + |\rho|/8] \quad \text{(EQ 57)}$$

Since the sign bit activity is so different from the LSB, or UWN, bit activity, the capacitance model should account for both types of bits. In other words, rather than having a single capacitive coefficient for UWN bits, there should also be capacitive coefficients for different sign bit transitions. The first step in determining these coefficients is to specify what types of bit transitions need to be characterized.

Figure 4-5 demonstrated that the activity of any two's-complement data stream can be characterized by two types of bits: UWN and sign. This implies that a *data template* such as that shown in Figure 4-7a can be associated with any data stream. The data template classifies bits in the data stream as either U or S for UWN and sign bits, respectively. The qualifier "template" refers to the fact that no specific value for the sign bits (i.e. + or –) is indicated.



(a) Data template (b) Transition template (c) Transition ID's

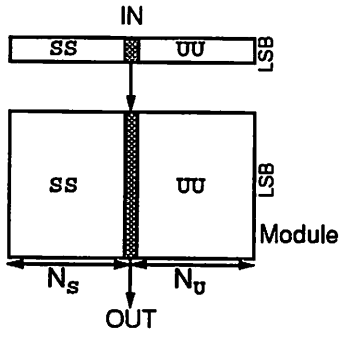
Figure 4-7 : Templates for identifying data bit types

Since the ultimate intent is to classify data *transitions* - which cause capacitive switching - rather than static values, the concept of a *transition template* illustrated in Figure 4-7b will also prove useful. The SS indicates a transition from one sign value to another (possibly equal) value. Similarly, UU suggests that the LSB bits all transition from their current random values to some new random values. As before, a transition "template" does not imply any particular sign values; however, a *transition ID* can be used to denote a specific sign transition. For instance, the transition template of Figure 4-7b encompasses the four possible transition ID's of Figure 4-7c.

To summarize, the activity of datapath signals cannot be adequately described by using a pure UWN model. This section has proposed a Dual Bit Type (DBT) model that reflects the distinct behaviors of both the LSB's and the MSB's. The LSB's are still modeled as UWN bits; however, the MSB's are recognized as sign bits. Terminology was presented for identifying these bit types, as well as the possible transitions they can make. The next section will describe how the capacitance switched within a module for these various data transitions can be captured by using several effective capacitance coefficients, rather than just one.

Capacitive Data Coefficients

The DBT method accurately accounts for different types of input statistics by having a separate coefficient for each possible transition of both the UWN and the sign bits. The



(a) Module transition templates

Transition Templates	Capacitive Coefficients			
UU	C_{UU}			
SS	C_{++}	C_{+-}	C_{-+}	C_{--}

(b) Capacitive coefficients

Figure 4-8 : Transition templates and capacitive coefficients for one-input modules

terminology of the previous section, which was used to distinguish possible input transitions can also be used to categorize the required capacitive coefficients. For example, consider a bit-sliced module with a single data input as shown in Figure 4-8a. The module can be split into two regions based on the input bit types. The LSB region experiences random (UWN) input transitions and the effective capacitance of this region can, therefore, be characterized by a single capacitive coefficient, C_{UU}. The MSB region, in contrast, experiences sign transitions and will require several capacitive sign coefficients, C_{SS}, to reflect its effective capacitance. Since there are four sign transitions that fit this template, there must also be four distinct coefficients.

The reason for taking this exhaustive approach to enumerating coefficients is to ensure the generality of the modeling technique over a wide range of module types and implementation styles. For many modules, some of the coefficients are redundant; however, since the redundant coefficients vary from case to case, trying to eliminate them could lead to a large number of special cases, complicating the model and limiting its generality. For instance, consider static and dynamic implementations of an arithmetic negation unit (Out = -In). For the static version, C_{++} and C_{--} are redundant since only sign *transitions* would lead to power consumption in the sign region. For the dynamic version, however, power consumption would depend on the current sign

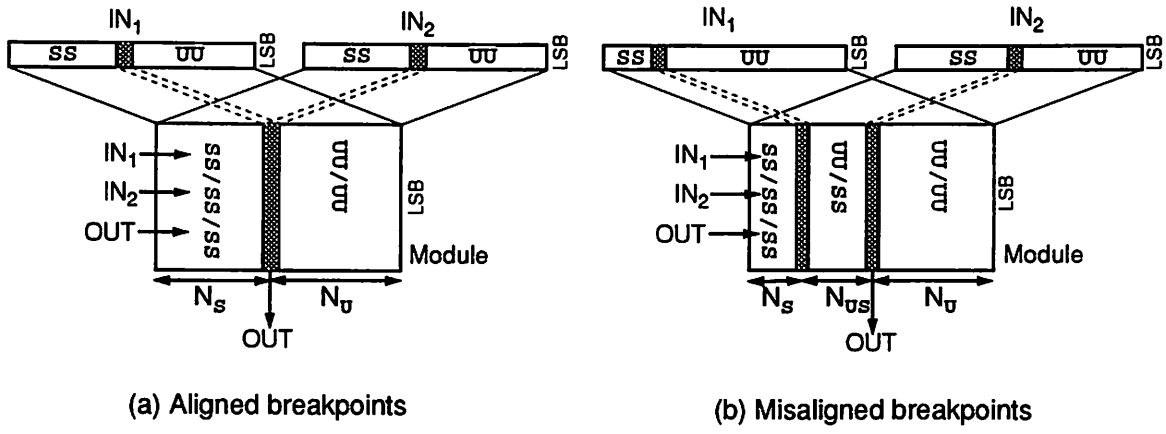


Figure 4-9 : Transition templates for two-input modules

of the output and not on whether an input transition occurred. For a positive result, the output would precharge to 1 and then evaluate to 0. So even if no input transition occurred, power would still be consumed, and in this case C_{++} and C_{--}would not be redundant. So, coefficients that are redundant in one instance may not be so in another. Therefore, using an exhaustive set of capacitive coefficients keeps the model as general as possible. Returning to the one-input module, we conclude that its effective capacitance is completely characterized by a table of five coefficients (see Figure 4-8b).

The number of coefficients that are required will depend on how many inputs the module has. The one-input module discussed above requires five coefficients since only five types of input bit transitions are possible. A two-input module, on the other hand, must be characterized for transitions on more than one data stream. This situation is illustrated by Figure 4-9a. For this case, the LSB region sees random (UWN) transitions on both inputs. The transition ID for this region of the module is written UU/UU, where the "/" separates the transition ID for IN_1 from IN_2. The effective capacitance of the module in this region will be described by the coefficient $C_{UU/UU}$.

In the sign region, the module transition template has three components (SS/SS/SS) rather

Transition Templates	Capacitive Coefficients				
UU/UU	$C_{UU/UU}$				1
SS/SS/SS	$C_{++/++/++}$	$C_{++/++/+-}$	$C_{++/++/-+}$	$C_{++/++/--}$	64
	$C_{++/+-/++}$	$C_{++/+-/+-}$	$C_{++/+-/-+}$	$C_{++/+-/--}$	
	...	...	...	...	
	$C_{--/-+/++}$	$C_{--/-+/+-}$	$C_{--/-+/-+}$	$C_{--/-+/--}$	
	$C_{--/--/++}$	$C_{--/--/+-}$	$C_{--/--/-+}$	$C_{--/--/--}$	
Misaligned breakpoints only: UU/SS	$C_{UU/++}$	$C_{UU/+-}$	$C_{UU/-+}$	$C_{UU/--}$	8
Misaligned breakpoints only: SS/UU	$C_{++/UU}$	$C_{+-/UU}$	$C_{-+/UU}$	$C_{--/UU}$	

Table 4-1 : Capacitive coefficients for two-input modules

than the two that might be expected. This is because the output sign can affect the capacitance switched in the module, and for some modules the sign of the inputs does not completely determine the output sign. For example, consider characterizing the effective capacitance of a subtracter. Subtracting two positive numbers could produce either a positive or a negative result. If the transition template for the module only includes inputs, then a particular transition might be classified as ++/++. This makes it appear as though the module undergoes no activity even though it is possible that the output does make a sign transition (e.g. positive to negative). Specifying the output sign (e.g. ++/++/+-) avoids this ambiguity. For this reason, the transition template for the sign region of a two-input module should contain an entry for the output, as well as the inputs.

Expanding the module transition templates yields a maximum of 65 different transition ID's - each of which must have a characteristic capacitive coefficient (see the UU/UU and SS/SS/SS entries of Table 4-1). Not all of these transition ID's, however, are feasible for every module

function. For example, given two positive inputs, an adder cannot produce a negative output. Therefore, capacitive coefficients corresponding to transition ID's of the form +□/+□/-□ and □+/□+/□- are not allowed. For an adder, it so happens that only 36 of the maximum 65 capacitive coefficients are feasible; the remaining table entries can be set to zero. Other modules require a different subset of coefficients - the specific subset being determined by the function the module implements.

In Figure 4-9a on page 124 the situation was simplified by assuming that the breakpoints of the two inputs aligned perfectly. In reality, the inputs might have different statistics that cause the number of UWN bits and sign bits required by each of the inputs to be different. The general case of misaligned breakpoints is shown in Figure 4-9b. Relaxing breakpoint alignment creates the possibility of a third module region with one UWN input and one sign input. This increases the number of capacitive coefficients and transition templates required to fully characterize the module to 73 as shown in the bottom rows of Table 4-1.

Recall from the previous section that capacitive coefficients can be either scalars or vectors depending on the number of complexity terms in the capacitance model. For models with more than one term, the capacitive coefficients described here would also be vectors rather than scalars. For instance, the logarithmic shifter described above would require five coefficient vectors to fully characterize the effective capacitance (one for each table entry in Figure 4-8b on page 123). Each of these vectors in turn would consist of five scalar coefficient elements. For example, $\mathbf{C}_{UU} = \left[C_{0,UU}\ C_{1,UU}\ C_{2,UU}\ C_{3,UU}\ C_{4,UU}\right]^T$ for the uniform white noise coefficient.

The number of capacitive coefficients required to characterize a module grows rapidly with the number of inputs. Consequently, approximations must be used for datapath blocks with more than two inputs. As an example, consider a multiplexer with three or more inputs. We can derive an approximate model for such a unit by recognizing that the selected input alone fully determines the output data and, therefore, has a larger impact on module power consumption than the inputs

that are not selected. As a result, treating a multiplexer as a module with a single, effective input (i.e. the selected input) is a reasonable approximation. This is analogous to the common gate-level practice of lumping all capacitance at the output and considering only output transitions in determining power consumption. Using similar approximations, the DBT model can be extended to handle modules with more than two inputs. This is typically not a major concern since most common datapath elements have two inputs or less (e.g. multipliers, shifters, adders, subtracters, comparators, etc.). Also note that this section has focused on bit-sliced modules. Bit-meshed modules require a slightly different approach. The exact differences will be described in Section 4.1.5 when the modeling of an array multiplier is discussed.

In summary, there are two types of bits (sign and UWN) that both have quite different activities. As a result, the effective capacitance of a module will be different depending on the exact input transitions that the module experiences. To accurately account for this effect, the DBT model uses distinct capacitive coefficients for each type of data transition that can occur.

Multi-Function Units and Capacitive Control Coefficients

If a unit performs only a single function (e.g. an adder), then its power can be described by a single set of capacitive coefficients. Some units, however, are able to perform multiple functions, which may each consume different amounts of power - e.g. an ALU which can perform add, subtract, and shift operations. In this case, a single set of coefficients is insufficient. Instead, we must have a unique set of capacitive coefficients for each distinct function that the unit may be called on to perform. This minor extension allows the DBT model to handle input data transitions for multi-function modules in much the same manner as has been described above. This strategy works well for most cases; however, sometimes the number of functions a unit can perform becomes unwieldy. For example, one could view a 32-bit shifter as having 64 different functions: shift left by one, two, three, etc. and shift right by these same amounts. Techniques for handling such cases will be described in Section 4.1.5.

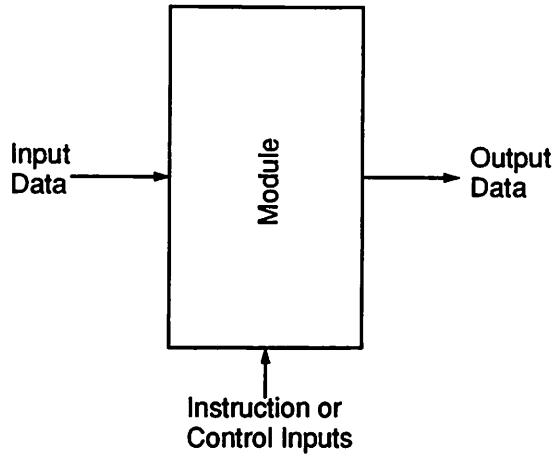


Figure 4-10 : Datapath module with control as well as data inputs

Having a table of capacitance coefficients for each function is not in itself sufficient to fully model the power consumption of a multi-function module. We also have to consider power consumed during transitions from one function to another. Using a table of coefficients for each function would only handle the case of data input transitions for a given function. A multi-function module, however, will have control as well as data inputs as illustrated by Figure 4-10. The control inputs together form an *instruction* to the module which tell it what function to perform. For example, an ALU may have several encoded control bits whose values determine whether the module will implement an add, subtract, shift, or logic operation. As long as these control lines remain fixed, the module will execute the same function, and data transitions can be handled in the manner already described using the set of capacitive coefficients corresponding to the current instruction. When a control input makes a transition, however, the state of the module can be affected just as if a data transition had occurred. This change in state is most often accompanied by some energy consumption as the internal nodes of the module transition to their new values. Therefore, in order to analyze the total power consumption of a module we must consider capacitance switching initiated by control transitions, as well as data transitions.

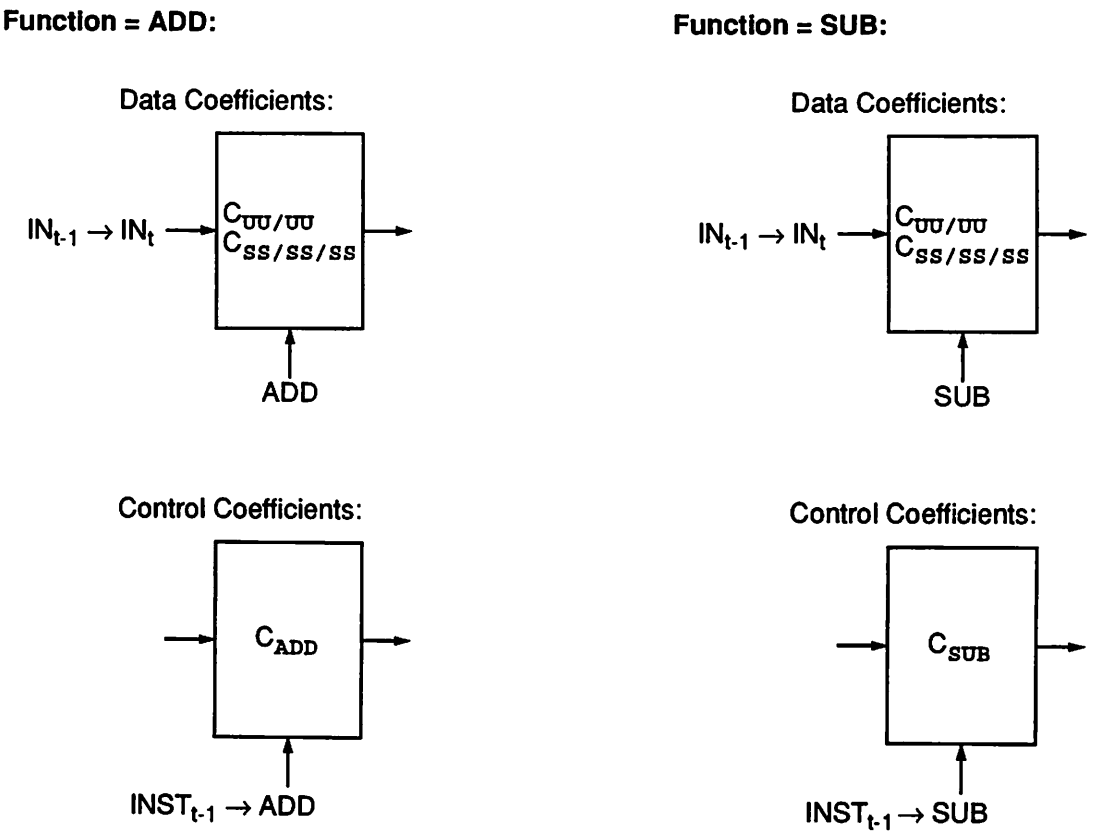


Figure 4-11 : Full capacitive coefficient set for an ADD/SUB unit

This capacitance can be modeled by adding *capacitive control coefficients* that describe the average amount of capacitance switched in the module when the control inputs transition to a new instruction. Since a *transition* involves an initial and final value, it may seem that a module with N functions would require N^2 coefficients to characterize all possible instruction transitions. Technically, this is true, however, from a practical standpoint, it is primarily the final instruction that determines the power consumption since the module outputs are being recomputed for this new instruction, not the previous one. So, a sufficient set of capacitive coefficients for a multi-function module would consist of a capacitive control coefficient for each instruction, as well as a table of capacitive data coefficients for each instruction. As an example, Figure 4-11 shows the coefficients required for an ADD/SUB unit.

To summarize, the capacitance switched under various types of input activity can be accurately modeled by using a table of effective capacitance coefficients rather than the traditional single (UWN) coefficient. Separate coefficients can be used to characterize transitions on data inputs as well as control inputs (for multi-function modules). These coefficients can be plugged into the capacitance models described in Section 4.1.1 in order to accurately model the effect of *complexity* and *activity* on module power consumption.

4.1.3 Library Characterization Method

The preceding discussion assumes the existence of capacitive coefficient tables, but does not describe how to produce them. *Library characterization* refers to the process of generating effective capacitance coefficients for each module in the datapath library. This is a one-time process, not required during power analysis, but instead performed whenever a new cell is added to the library.

The procedure consists of several steps (see Figure 4-12) but can be automated so as to require very little intervention on the part of the user. *Pattern generation* is the first step in the three-stage process. During this phase, input patterns for various UWN, sign, and control transitions are generated for the module being characterized. Next, *simulation* is used to measure the capacitance switched for these input activity patterns. In order to characterize the influence of complexity, as well as activity, the module may be characterized for several complexity parameter values (e.g. word length, number of shift stages, etc.). Then, during *coefficient extraction*, the capacitance models are fit to the simulated capacitance data to produce a set of "best fit" capacitive coefficients.

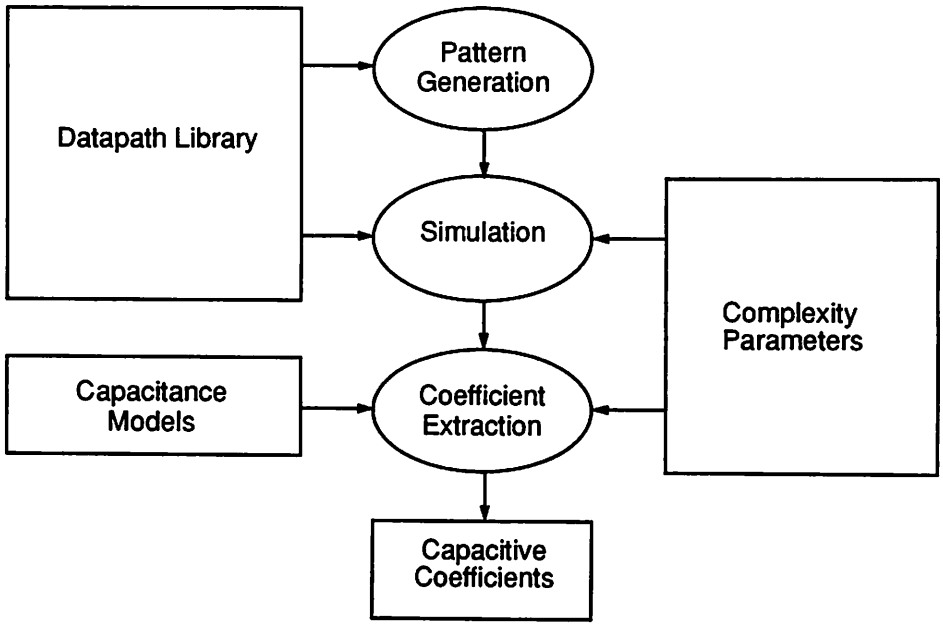


Figure 4-12 : Process flow for library characterization

Pattern Generation

The input patterns applied during characterization must embody all important types of input transitions. This includes both data (UWN/sign) and control (instruction) input transitions. The pattern generation problem can be simplified by handling data and control separately.

Data pattern generation is performed assuming fixed control inputs. In order to fully characterize a module, the data stream must include patterns for each transition ID discussed in Section 4.1.2. So for a one-input module, the UU transition ID requires a random data stream. Similarly, the SS transition template requires inputs corresponding to the following sign transitions: positive to positive (++), positive to negative (+-), negative to positive (-+), and negative to negative (--). Figure 4-13 shows what a data stream used to characterize a module (with a word length of 16 bits) might look like. Notice that input patterns containing a UWN component must be simulated for several cycles to allow the average capacitance switched to

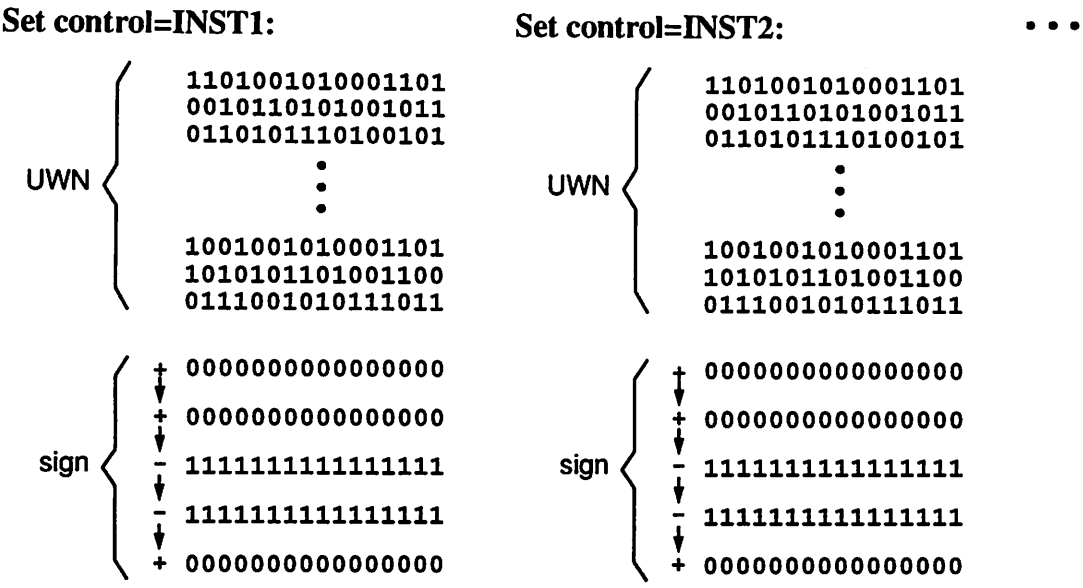


Figure 4-13 : Sample data input patterns for 16-bit one-input module

converge during simulation. Also, the data stream must be repeated for all possible instructions of multi-function units.

Similar input patterns can be generated for two-input modules. This can be done in a manner that minimizes the overall simulation length by, whenever possible, using the previous data values as the initial data types for the next transition ID. For example, if the current data values have signs given by +/+/+, then the next transition ID characterized should be of the form +□/+□/+□ to avoid having to insert an initialization step into the simulation. The full characterization sequence for the SS/SS/SS transition template of an adder is given in Figure 4-14.

Notice that as mentioned previously, the transition ID's for two-input modules include an output sign specification. In order to control the output sign, the "sign" inputs cannot always be limited to all 0's (for positive) or all 1's (for negative) as was done for the one-input module. For example, a multiplier must be characterized for (+) x (-) = (-), but using all 0 or all 1 inputs would produce:

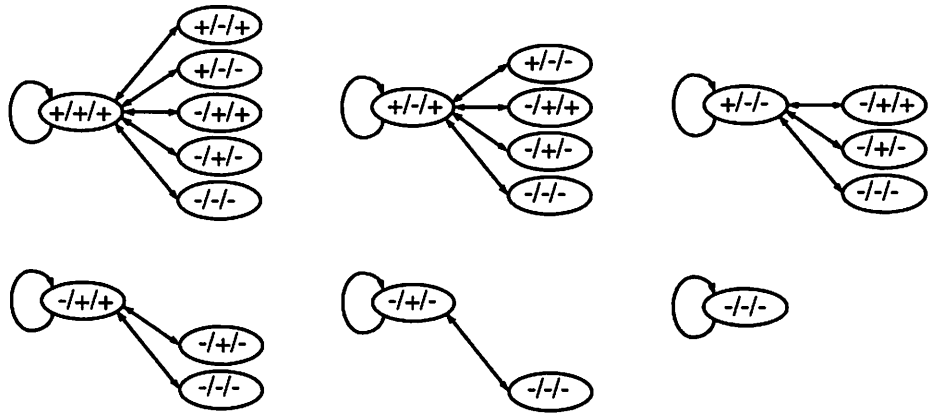


Figure 4-14 : Sign characterization sequence for an adder

$$(000...00) \times (111...11) = (000...00) \quad \text{(EQ 58)}$$

The solution is not to require sign inputs to have all 0's or all 1's, but instead to allow a small number of non-sign bits that can be used to manipulate the output sign. For instance, the correct multiplier signs can be generated by:

$$(000...01) \times (111...11) = (111...11) \quad \text{(EQ 59)}$$

In general, the specific data patterns required to exercise a module depend only on the function that the module realizes. For instance, the feasible input and output sign combinations for any multiplier, regardless of its internal construction, can be characterized by the data patterns of Table 4-2.

For multi-function units, the input patterns must include control, as well as data transitions. In particular, input patterns exercising the various possible instruction transitions must be generated. Recall, however, that characterization of instruction transition energies depends primarily on the final instruction value. This simplifies the pattern generation process since it must only generate a transition *to* each instruction regardless of the initial instruction. This means that all control coefficients can be characterized by a single pass through the instruction set. A sample input sequence for an ALU with add, subtract, and shift functionality is shown in Figure 4-15.

Sign Pattern	IN_1	IN_2	OUT
(+) x (+) = (+)	0	0	0
(+) x (-) = (-)	1	-1	-1
(-) x (+) = (-)	-1	1	-1
(-) x (-) = (+)	-1	-1	1

Table 4-2 : Sign patterns and corresponding data values for multiply function

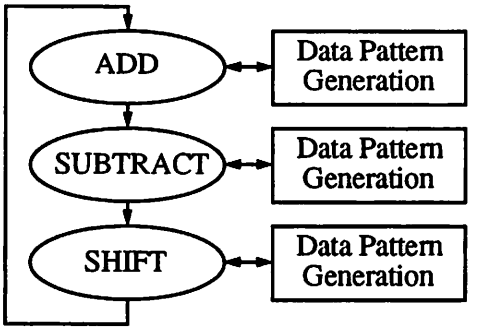


Figure 4-15 : Pattern generation for multi-function units

To summarize, the inputs patterns should exercise all important combinations of sign, UWN, and instruction transitions. This will allow all required effective capacitance coefficients to be represented during simulation. The algorithm for generating minimal length input pattern sequences is relatively straightforward and an automatic pattern generation tool (APG) that performs this task for many common datapath functions and modules has been implemented.

Simulation

Once generated, the input patterns are fed to a simulator from which module switching capacitances are extracted. Any simulator may be chosen, depending on the accuracy desired and

the time allotted to characterization. Chapter 3 gave a detailed discussion of the speed/accuracy trade-offs involved in this decision. Switch-level simulators offer, perhaps, the best combination of speed and accuracy. All results presented here were derived using IRSIM-CAP, which is a version of IRSIM-9.0 [Sal89] with improved capacitance measurement capabilities. As stated in Chapter 3, IRSIM-CAP is three to four orders of magnitude faster than SPICE with its average power estimates usually differing by only 10-15%.

Ideally, the simulation would be based on a netlist extracted from an actual implementation of the function being characterized. In this way, design parameters such as device sizes, as well as technology parameters including gate capacitances and device transconductances are automatically incorporated into the resulting model.

Unfortunately, transistor-level netlists of the hardware module are not always readily available. In these cases, logic synthesis tools can be used to produce a gate-level mapping of the function. The UWN and sign-bit input patterns, discussed above, can then be applied to this boolean network using one of the many existing gate-level simulators with capacitance measurement capabilities described in Chapter 3.

Characterization does not have to be based on simulation. If physical implementations of library modules are available, characterization can be performed by applying the input patterns to the actual chip while *measuring* incremental power consumption. This would, of course, be the most accurate technique; however, since library cells are not commonly fabricated individually on chips, simulation must in most cases suffice.

In any case, it is interesting to note that since each module is characterized as a multi-bit unit (e.g. a 16 or 32-bit adder) glitching within the module can actually be accounted for during characterization. This is an improvement over many gate-level tools that employ a zero-delay model and, therefore, ignore glitching. Of course, the DBT model will only account for glitching if the simulator used during characterization models this phenomenon.

Coefficient Extraction

The simulation step produces an output file containing effective switching capacitances for the entire series of applied input transitions. It only remains to extract the capacitive coefficient values that make the capacitance model best fit the measured data. This can be achieved using a variety of model fitting techniques such as linear least-squares regression, which minimizes the mean-squared error of the model. This amounts to solving the following matrix equation for the capacitive coefficient vector, $\mathbf{C}_{eff}$, that minimizes the modeling error, $\mathbf{e}$:

$$\mathbf{C}_{sim} = \mathbf{P}\mathbf{C}_{eff} + \mathbf{e} \qquad \text{(EQ 60)}$$

where $\mathbf{C}_{sim}$ is a vector of simulated capacitance observations, $\mathbf{P}$ is a matrix of complexity parameter values corresponding to each observation. Since for each module, there are several effective capacitance coefficients (e.g. $\mathbf{C}_{00}$, $\mathbf{C}_{++}$, $\mathbf{C}_{+-}$, etc.) the equation must be solved several times - once for each capacitive coefficient vector.

The first step in solving these equations is to form the parameter matrix, $\mathbf{P}$. Each row of the matrix corresponds to a different value of the complexity vector, $\mathbf{N}$, described in Section 4.1.1. For example, assume the logarithmic shifter described in that section was characterized for word lengths $N \in \{8, 16, 32\}$ and number of stages $L \in \{1, 3, 5\}$, implying $M \in \{1, 7, 31\}$. Plugging these values into the complexity vector, $\mathbf{N} = [N\ L\ NL\ N^2L\ MNL]$, the parameter matrix would be given by:

$$\mathbf{P} = \begin{array}{c} \begin{matrix} N & L & NL & N^2L & MNL \end{matrix} \\ \begin{bmatrix} 8 & 1 & 8 & 64 & 8 \\ 8 & 3 & 24 & 192 & 168 \\ 8 & 5 & 40 & 320 & 1240 \\ 16 & 1 & 16 & 256 & 16 \\ 16 & 3 & 48 & 768 & 336 \\ 16 & 5 & 80 & 1280 & 2480 \\ 32 & 1 & 32 & 1024 & 32 \\ 32 & 3 & 96 & 3072 & 672 \\ 32 & 5 & 160 & 5120 & 4960 \end{bmatrix} \end{array} \qquad \text{(EQ 61)}$$

This matrix and the vector, C_{sim}, of corresponding capacitance observations then feed directly into a least-squares regression algorithm to produce the best-fit coefficient vector, C_{eff}.

To review, during coefficient extraction the mass of raw capacitance data produced by simulation is used to derive values for any coefficients in the capacitance model. The coefficient values should be chosen to make the model best fit the simulated data. This can be accomplished using a number of model fitting algorithms - for example, least squares regression would be a suitable choice. As with pattern generation and simulation, the coefficient extraction process has been automated and is performed by a tool called ACE (Automatic Capacitance Extractor).

Library Characterization Summary

Characterization is the process used to generate tables of capacitive coefficients for library modules, which can then be used to evaluate effective capacitances during power analysis. Characterization consists of three phases: pattern generation, simulation, and coefficient extraction.

The accuracy obtained by characterization depends on the suitability of the chosen capacitance model, as well as the number of observations used in fitting the model; however, as an approximate figure, the logarithmic shifter described here can be modeled with an rms error of only 9.9%. Of course, other sources of error, aside from model characterization, such as inaccuracies in the capacitance simulations, contribute to the overall error of the DBT technique, which is more on the order of 10-15% (relative to switch-level simulation).

4.1.4 Power Analysis Method

The capacitive coefficient tables produced by library characterization can be used to analyze the power consumed by datapath modules. This section describes the specifics of the DBT power analysis method. The first step in the process is to split the module under analysis into UWN and sign bit regions. This is followed by a region-by-region analysis of the effective capacitance

switched within each region. These two topics are discussed in the subsections entitled, "Regional Decomposition" and "Effective Capacitance Computation," respectively.

The DBT analysis method requires several inputs. First, a pointer to the library element being analyzed must be provided. This gives the analysis tool access to the parameterized capacitance model and the capacitive coefficient tables that characterize the module's power consumption. Next, to accurately account for activity, the power analysis tool will require several module input statistics. These include the mean, variance, and correlation of the data inputs, which affect the placement of breakpoints during regional decomposition. In addition, sign and instruction transition probabilities are required to weight the appropriate capacitive data and control coefficients during the effective capacitance computation. Techniques for calculating these parameters will be discussed in Section 4.5.

Regional Decomposition

Once its activity parameters are known, a module can be decomposed into sign and UWN regions. This allows the power analysis tool to account for differences in the behavior and activity levels of these regions. The decomposition strategy utilizes the breakpoint formulas, i.e. (EQ 55)-(EQ 57), presented in Section 4.1.2.

Consider the case of a one-input module as depicted in Figure 4-8 on page 123. The task of regional decomposition reduces to figuring out the number of sign bits, N_S, and the number of UWN bits, N_U. Linear interpolation of power consumption for the small intermediate region can be achieved by attributing half of the N_I intermediate bits to the sign region and half to the UWN region:

$$N_I = BP1 - BP0 - 1 \qquad \text{(EQ 62)}$$

$$N_S = (N - BP1) + \frac{N_I}{2} \qquad \text{(EQ 63)}$$

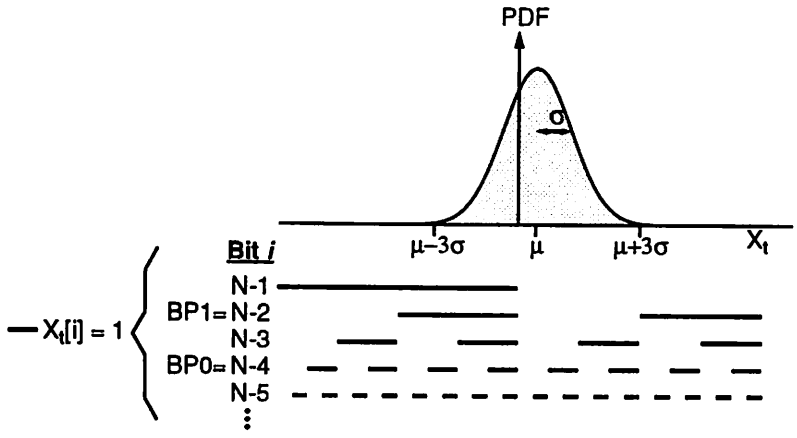


Figure 4-16 : Relationship between bit and word values

$$N_U = (BP0 + 1) + \frac{N_X}{2} \qquad \text{(EQ 64)}$$

Similar decomposition formulas can be derived for the two-input case of Figure 4-9 on page 124. These expressions show that the size of the module regions depends on the positions of the input breakpoints. Section 4.1.2 presented the breakpoint formulas but did not explain how they were derived. This section undertakes that task.

First, consider the high-end breakpoint, $BP1$. The bits between $BP1$ and the MSB are all sign bits. Therefore, the high-end breakpoint corresponds to the maximum number of non-sign bits required to represent all likely values of the data word. The range of likely values is illustrated by Figure 4-16, which shows the probability density function (PDF) for a gaussian variable X_t. The shaded curve denotes the probability that a particular value of X_t will occur. This probability is highest at the mean (μ) and decays to almost zero within three standard deviations (3σ). So the most common values of X_t fall within the range $R_x = [\mu\text{-}3\sigma, \mu\text{+}3\sigma]$. Since the largest likely magnitude of the signal, then, is $|\mu|+3\sigma$, the maximum number of non-sign (data) bits will be:

$$BP1 = \log_2(|\mu| + 3\sigma) \qquad \text{(EQ 65)}$$

The line segments at the bottom of Figure 4-16 provide a graphical interpretation of this result.

Each set of line segments corresponds to a bit of X_t. For a given bit i, solid segments coincide with ranges of X_t where bit i is one ($X_t[i]=1$). In contrast, unshaded segments correspond to a bit value of zero ($X_t[i]=0$). The length of these segments is 2^i, corresponding to the place value of the bit. With this in mind, the high-end breakpoint occurs when the length of the line segments is comparable to $|\mu|+3\sigma$, the magnitude of the maximum likely value of X_t (e.g. bit N-2 in this figure). At this bit, $X_t[BP1]=1$ for all negative values in R_x and $X_t[BP1]=0$ for all positive values in R_x. In other words, $BP1$ corresponds to the least significant "sign" bit. This graphical criterion of comparing the length of the line segments (2^i) to $|\mu|+3\sigma$ is another way of arriving at (EQ 65).

When X_{t-1} and X_t are uncorrelated ($\rho=0$), a similar approach can be used to determine the lower breakpoint ($BP0$). Instead of comparing segment lengths (representing place value) to $|\mu|+3\sigma$, however, we now compare to σ alone. A UWN bit should be basically uncorrelated with other bits of X_t, approximately obeying P(0)=P(1)=0.5. For line segments of length σ or smaller, many $X_t[i]=1$ and $X_t[i]=0$ line segments occur within the range R_x. Summing the areas under the PDF curve for each of these regions reveals nearly equal likelihood for zero and one bit values. For line segments of length σ or longer, however, this no longer holds. In that case, very few line segments will span R_x and bit probabilities will start to depend on μ. In summary, $BP0$ occurs when the line segment length (2^i) approaches the standard deviation (σ):

$$BP0 = \log_2\sigma \qquad \text{(EQ 66)}$$

By assuming uncorrelated signals, we were able to arrive at (EQ 66) using a one-dimensional analysis. A precise analysis, however, requires a bivariate approach since the breakpoints of P(0→1) depend on both $X_{t-1}[i]$ and $X_t[i]$. Figure 4-17 illustrates the two-dimensional situation. The figure shows the joint PDF of X_{t-1} and X_t suspended above a bit grid. The bit grid contains rectangular strips corresponding to the values of X_{t-1} and X_t for which $X_{t-1}[i]=0$ and $X_t[i]=1$, respectively. These strips intersect at shaded squares representing 0→1 transitions of bit i.

In the one-dimensional case of Figure 4-16, we determined $BP0$ by comparing 2^i, the length of

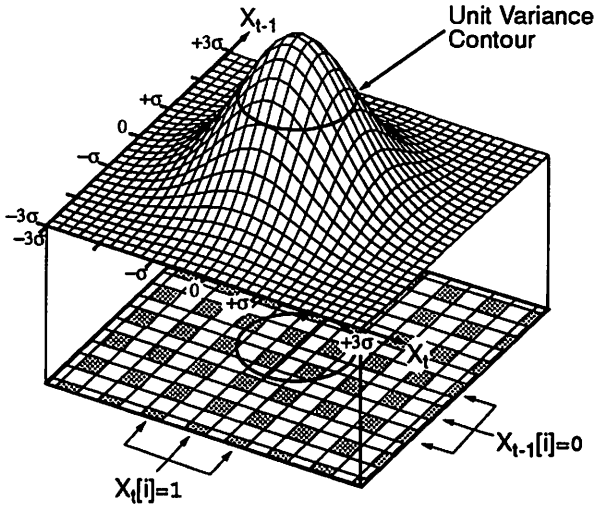


Figure 4-17 : Joint PDF with bit grid appropriate to the LSB region (ρ=0)

$X_t[i]$=1 line segments, to σ, the standard deviation of X_t. In the two-dimensional case, instead of line segments corresponding to 1 bits, we have shaded squares corresponding to 0→1 bit transitions. In addition, the one-dimensional measure of distribution "spread" σ, is replaced by the notion of a unit variance contour (i.e. a circle of radius σ for ρ=0 and an ellipse for ρ≠0).

So, by analogy, to determine *BP*0 for the two-dimensional case, we must compare the "size" of the grid squares (2^ix2^i) with the "size" of the ellipse. For uncorrelated samples, the two axes can be treated separately. So, comparing 2^i to radius σ we arrive once again at (EQ 66). Relating this to Figure 4-17, since squares of the bit grid are much smaller than the σ-ellipse, bit *i* must fall within the LSB region. Similar reasoning can be used to re-derive (EQ 65) for *BP*1. Graphically, a bit *i* in the MSB region would produce a graph similar to Figure 4-18, with a bit grid of size comparable to a 3σ-ellipse.

The importance of a two-dimensional analysis becomes clear when X_{t-1} and X_t are correlated as in Figure 4-19. For this example, the significant correlation (ρ=0.8) compresses the σ-ellipse,

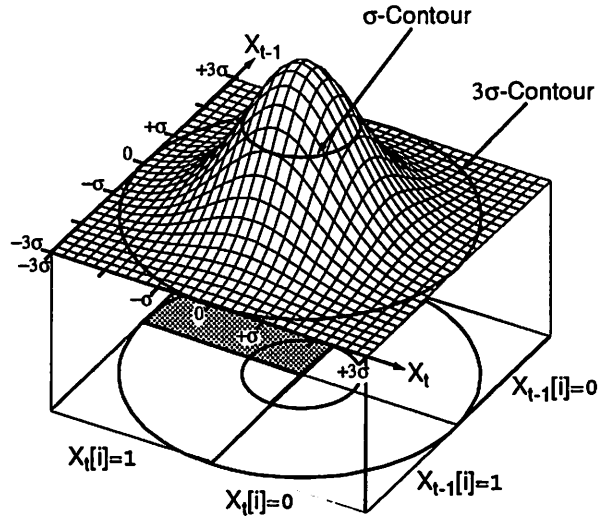


Figure 4-18 : Joint PDF with bit grid appropriate to the MSB region (ρ=0)

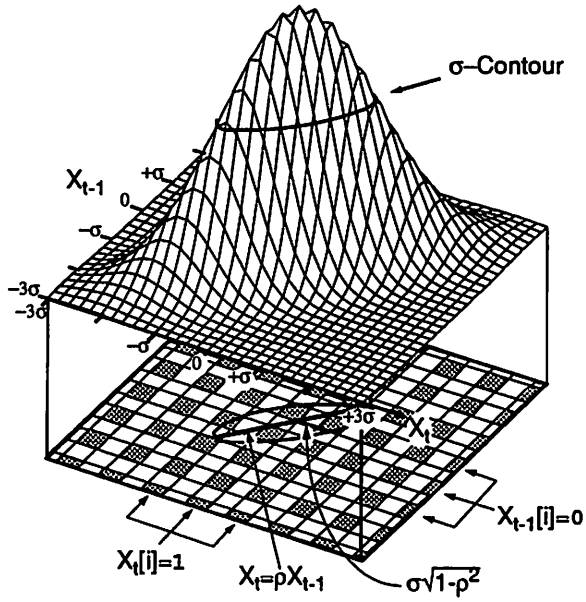


Figure 4-19 : Joint PDF with bit grid appropriate to the LSB region (ρ=0.8)

and skews it from the (X_{t-1}, X_t) axes. The effect is to reduce the apparent "size" of the ellipse, causing breakpoint $BP0$ to occur at lower bits. One measure of the ellipse "size" is the conditional standard deviation of X_t given X_{t-1}: $\sigma(X_t|X_{t-1}) = \sigma\sqrt{1-\rho^2}$ as shown in the figure. But as $|\rho|\rightarrow 1$, this expression approaches zero, and the ellipse collapses into its horizontal centroid: $X_t=\rho X_{t-1}$. Even though the ellipse no longer has width in the X_t dimension, its axis is skewed and can still traverse a wide range of X_t values - possibly intersecting many squares of the bit grid.

Consequently, the "size" measure of the ellipse must have a component based on the length of the centroid measured along the X_t axis. Since the centroid follows $X_t=\rho X_{t-1}$ and extends from $X_{t-1}=-\sigma$ to $X_{t-1}=\sigma$, it spans a range of X_t values of length $2|\rho|\sigma$. Experimentation has shown that this range of values becomes significant when the centroid intersects eight or more $X_t[i]=1$ strips. This occurs for $2^i=|\rho|\sigma/8$.

Combining these two measures of ellipse size yields $\sigma' = \sigma\,[\sqrt{1-\rho^2}+|\rho|/8]$ - where the first term dominates for small correlations (wide ellipses) and the second term dominates for large correlations (narrow ellipses). Comparing to (EQ 66), the effect of correlation on the low-end breakpoint can be handled as an offset given by:

$$\Delta BP0 = \log_2[\sqrt{1-\rho^2}+|\rho|/8] \qquad \textbf{(EQ 67)}$$

At this point, we have derived all necessary equations for determining the breakpoints of the DBT data model. Although the derivation of these equations utilized gaussian random variables for the purposes of illustration, the resulting breakpoint formulas are fairly independent of the underlying distribution. Altering the distribution merely introduces a small multiplicative constant to the σ terms in the above equations, but the effect of this is minimized by the logarithmic form of the breakpoint equations.

In summary, this section derived breakpoint formulas for the input streams of datapath modules. These important expressions determine how the module will be decomposed into regions of various bit types during power analysis and are summarized below:

$$BP1 = \log_2 3\sigma \quad \text{(EQ 68)}$$

$$BP0 = \log_2 \sigma + \Delta BP0 \quad \text{(EQ 69)}$$

$$\Delta BP0 = \log_2 [\sqrt{1-\rho^2} + |\rho|/8] \quad \text{(EQ 70)}$$

The first step in the DBT power analysis process is to apply these formulas to derive the input breakpoints and to decompose the module under analysis into UWN and sign bit regions. Figure 4-8 on page 123 and Figure 4-9 on page 124 show how the positions of the input breakpoints relate to the size of the various module regions, and as an example expressions for the one-input case were given in (EQ 62)-(EQ 64).

Effective Capacitance Computation

The output of the module decomposition process is a list of regions contained in the module along with their relative sizes (e.g. N_U, N_S, etc.). This information can be used to calculate the effective capacitance switched during an average access of the module by analyzing the effective capacitance for each region independently and then summing the results. The effective capacitance of an individual region, r, of N_r bits can be computed by first calculating what the capacitance would be if the region occupied all N bits of the module. Then this number can be scaled by the fraction of the module which the region actually occupies: N_r/N.

In computing the effective capacitance of region r we must remember that a region is described by a transition *template* and, therefore, actually embodies several different transition ID's. For example, the capacitance switched in an SS region will have contributions from ++, +−, −+, and −− transitions. The effective capacitance associated with any one transition ID, call it i, would be the probability that transition i occurs multiplied by the capacitance switched when it does occur. From the previous sections we know that this capacitance is equal to the product of the associated capacitive coefficient vector $\mathbf{C}_i$ and the complexity vector $\mathbf{N}$. To sum up, the effective capacitance switched by a module during an average access is given by:

$$C_T = \sum_{r \in \left(\begin{smallmatrix}\text{regions in}\\ \text{module}\end{smallmatrix}\right)} \frac{N_r}{N} \left[\sum_{i \in \left(\begin{smallmatrix}\text{transition IDs}\\ \text{for region r}\end{smallmatrix}\right)} P(i)\, C_i \cdot N \right] \qquad \text{(EQ 71)}$$

This general formula applies to the cases of both one- and two-input bit-sliced modules. For example, when applied to the case of one-input modules, this general formula leads to the more specific expression:

$$C_T = \frac{N_U}{N}[C_{UU} \cdot N] + \frac{N_S}{N}\left[\sum_{SS \in \left(\begin{smallmatrix}++,\ +-,\\ -+,\ --\end{smallmatrix}\right)} P(SS)\, C_{SS} \cdot N \right] \qquad \text{(EQ 72)}$$

Similarly, the expression for a two-input bit-sliced module is given by:

$$\begin{aligned} C_T = {} & \frac{N_U}{N}[C_{UU/UU} \cdot N] + \frac{N_{US}}{N}\left[\sum_{UU/SS} P(UU/SS)\, C_{UU/SS} \cdot N \right] + \\ & \frac{N_{SU}}{N}\left[\sum_{SS/UU} P(SS/UU)\, C_{SS/UU} \cdot N \right] + \frac{N_S}{N}\left[\sum_{SS/SS/SS} P(SS/SS/SS)\, C_{SS/SS/SS} \cdot N \right] \end{aligned} \qquad \text{(EQ 73)}$$

As a final note, all of these formulas reflect the average capacitance switched during a single access or computation of a datapath module. If the total capacitance switched over a number of computations is desired, then the above formulas must be weighted by the total number of input data transitions, N_{DT}, that occur during that period:

$$C_T\big|_{\text{multi-cycle}} = N_{DT} \cdot C_T\big|_{\text{single-cycle}} \qquad \text{(EQ 74)}$$

To summarize, given the required complexity and activity parameters, analysis of module power consumption using the DBT model occurs in two stages. In the first step, the module is decomposed into UWN and sign regions of various sizes. Then, the effective capacitance of each region in the module is computed using the appropriate capacitance models and capacitive coefficients, which are stored in a hardware database.

Transition Templates	Capacitive Coefficients (fF/bit)															
UU/UU	264															
UU/SS	203	351	342	115												
SS/UU	118	316	315	199												
SS/SS	0	190	273	21	302	0	124	0	363	171	0	0	4	0	0	0
	0	236	0	347	107	254	242	107	0	375	0	0	348	184	0	0
	0	0	461	405	0	0	305	0	51	218	318	71	307	0	129	0
	0	0	0	16	0	0	366	486	0	230	0	319	5	169	152	19

Table 4-3 : Capacitive coefficients for 1.2 µm subtracter

4.1.5 Results

The DBT model applies to a wide variety of hardware units. The following case studies present results achieved for a representative subset of those modules.

Subtracter

The ripple-carry subtracter (and adder) are staples of any hardware library. Therefore, it is critical that any proposed modeling techniques perform well on these units. Before the power consumed by the subtracter can be analyzed, we must specify a capacitance model and derive a table of capacitive coefficients. For a linear unit, such as this, the appropriate capacitance model was derived in Section 4.1.1:

$$C_T = C_{eff}N \qquad \text{(EQ 75)}$$

The table of coefficients (in fF/bit) resulting from the characterization of the subtracter in a 1.2 µm CMOS technology is given in Table 4-3.

To test the overall accuracy of the DBT model, the time-multiplexed input stream for a 16-bit subtracter in a sub-band speech coding filter was captured, applied to a switch-level simulator

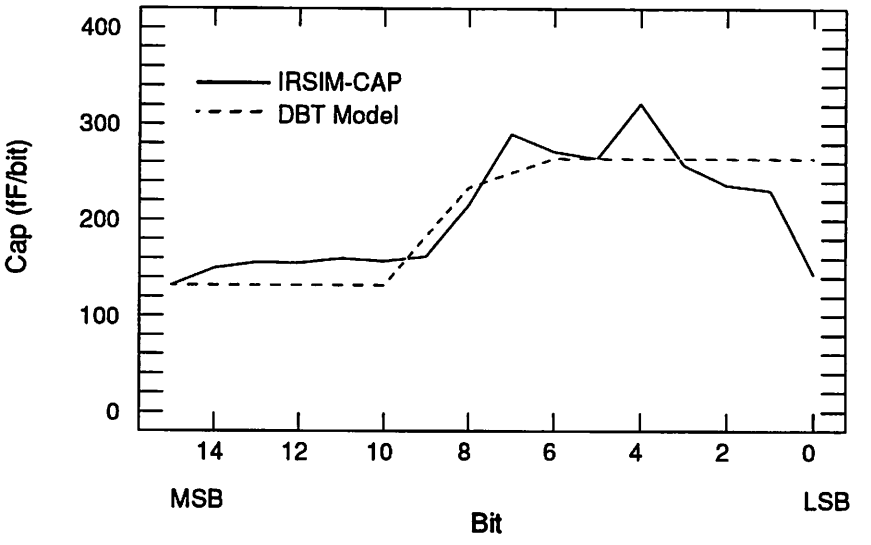


Figure 4-20 : Subtracter: IRSIM-CAP vs. DBT model

(IRSIM-CAP), and also analyzed using the DBT method (see Figure 4-20). The results are quite good, with the DBT model achieving an overall error of only 0.14%. We might suspect that some error cancellation is occurring here and, indeed, when we split the error into its UWN and sign region components we get +5.1% and -12.9%, respectively. This degree of accuracy is achieved based purely on an architecture-level model - that is, without any low-level run-time simulation nor any direct knowledge of the internal construction of the unit.

Logarithmic Shifter

The subtracter obeys a very simple linear capacitance model. The case of a logarithmic shifter illustrates what can be achieved for more complex models. The complexity parameters for this example were presented in Section 4.1.1 and are: the word length, N; the maximum shift value, M; and the number of shift stages, $L=\lceil \log_2(M+1) \rceil$. The value, S, of the SHIFT input also affects the power consumption. This influence can be handled in two ways. Since, technically, SHIFT is a control input, the technique suggested in the preceding text would be to consider each SHIFT

Transition Templates	Capacitive Coefficient Vectors (fF per unit complexity)
UU	[29.05 -113.7 19.883 -0.006 0.241 -0.200]
SS	[5.87 64.1 -0.610 -0.045 0.166 0.204]
	[62.52 -133.3 62.229 0.142 1.046 -0.458]
	[63.55 -65.0 5.783 -0.118 0.166 0.050]
	[4.82 170.2 -4.412 -0.013 0.139 0.171]

Table 4-4 : Capacitive coefficients for 1.2 μm logarithmic shifter

value as a different instruction and have a separate coefficient table for each case. This is not practical, however, since there may be 16, 32, or more possible values of *S*. Another solution would be to add *S* as an additional capacitance model parameter along with *N*, *M*, and *L*. This is a preferable solution since only one coefficient table would be required. With this addition, the new capacitance model for the shifter is given by:

$$C_T = C_0N + C_1L + C_2NL + C_3N^2L + C_4MNL + C_5SNL \qquad \text{(EQ 76)}$$

or equivalently:

$$C_T = \mathbf{C}_{eff} \cdot \mathbf{N} \qquad \text{(EQ 77)}$$

where the new coefficient and parameter vectors for the shifter are given by:

$$\mathbf{C}_{eff} = \left[C_0\ C_1\ C_2\ C_3\ C_4\ C_5\right]^T \qquad \text{(EQ 78)}$$

$$\text{and } \mathbf{N} = \left[N\ L\ NL\ N^2L\ MNL\ SNL\right]^T \qquad \text{(EQ 79)}$$

Table 4-4 presents the capacitive coefficient vectors corresponding to this model for a 1.2 μm implementation of the shifter. In addition, Figure 4-21 shows the fit of the extracted DBT model to simulation results for a left shift over a wide range of parameter values. The rms error over all cases is only 9.9%.

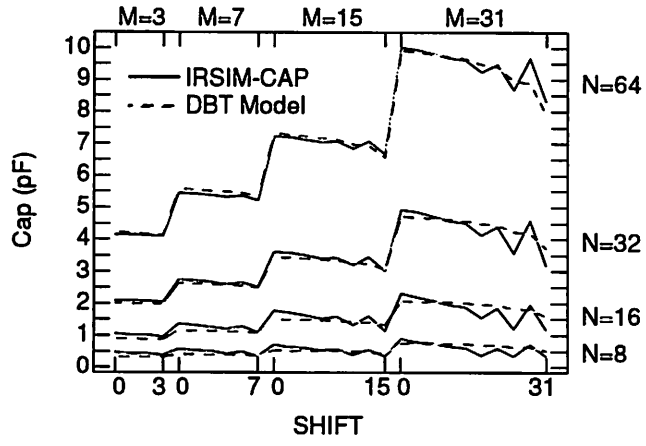


Figure 4-21 : Log shifter: IRSIM-CAP vs. DBT model

Array Multiplier

The array multiplier is a two-dimensional bit-meshed block. As discussed in Section 4.1.1, the capacitance model for the block is quadratic:

$$C_T = C_{eff} N_1 N_2 \qquad \text{(EQ 80)}$$

Since the cells of a bit-meshed module bring together all combinations of input bits, $IN_1[i]$ and $IN_2[j]$, all input region pairs are represented in the regional decomposition of the unit. So if each of the two inputs contains a UWN region and a sign region, then the bit-meshed module will decompose into a total of four regions as illustrated by Figure 4-22. The figure demonstrates that the capacitive coefficients of Table 4-1 on page 125 still apply to the bit-meshed case even though their interpretation differs.

The coefficients for a 1.2 μm array multiplier are given in Table 4-5. Given these effective capacitance coefficients along with the requisite activity and complexity parameters, the following generalization of (EQ 71) can be used to calculate the total effective capacitance for a single multiplication operation:

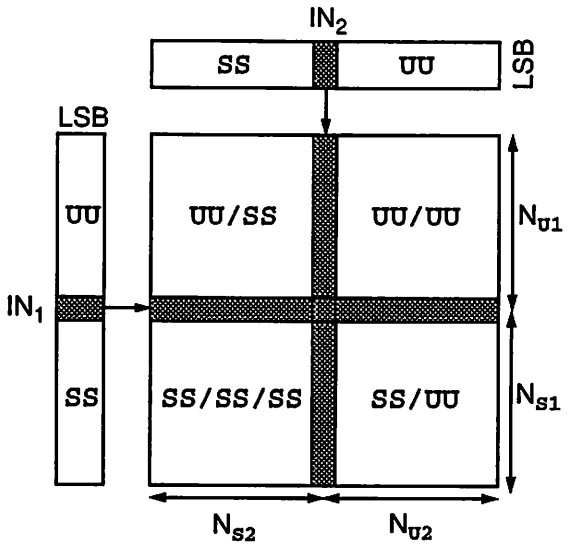


Figure 4-22 : Regional decomposition for bit-meshed array multiplier

Transition Templates	Capacitive Coefficients (fF/bit^2)															
UU/UU	253															
UU/SS	10	299	254	272												
SS/UU	5	317	254	262												
SS/SS	0	0	0	0	0	98	0	0	0	0	60	0	0	0	0	0
	0	141	0	0	397	0	0	0	0	0	0	158	0	0	355	0
	0	0	82	0	0	0	0	126	311	0	0	0	0	311	0	0
	0	0	0	0	0	0	333	0	0	317	0	0	0	0	0	0

Table 4-5 : Table of capacitive coefficients for 1.2 μm array multiplier

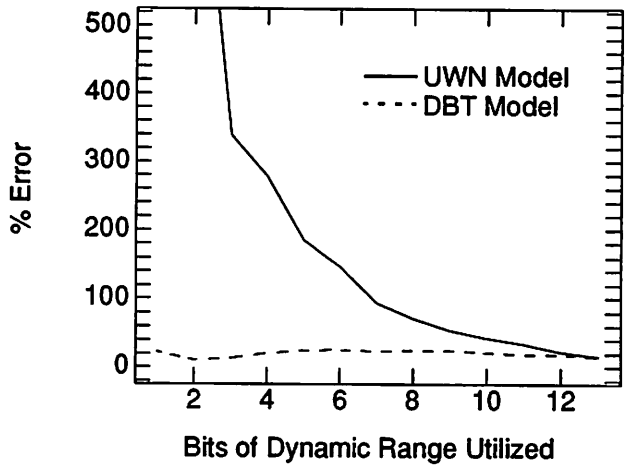


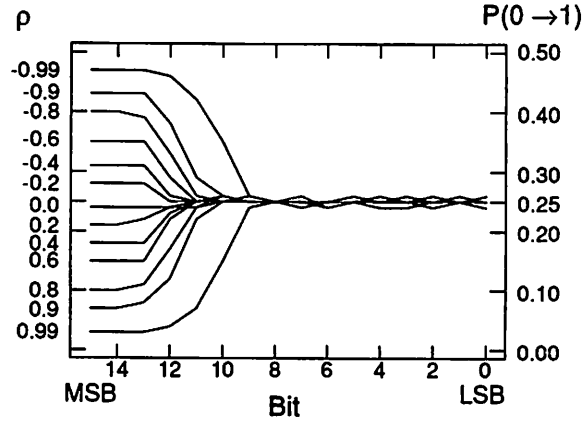
Figure 4-23 : DBT and UWN modeling errors for 16x16 multiplier

$$C_T = \frac{N_{U1}N_{U2}}{N^2}[C_{UU/UU} \cdot N] + \frac{N_{U1}N_{S2}}{N^2}\left[\sum_{UU/SS} P(UU/SS)\, C_{UU/SS} \cdot N\right] + \frac{N_{S1}N_{U2}}{N^2}\left[\sum_{SS/UU} P(SS/UU)\, C_{SS/UU} \cdot N\right] + \frac{N_{S1}N_{S2}}{N^2}\left[\sum_{SS/SS/SS} P(SS/SS/SS)\, C_{SS/SS/SS} \cdot N\right] \quad \text{(EQ 81)}$$

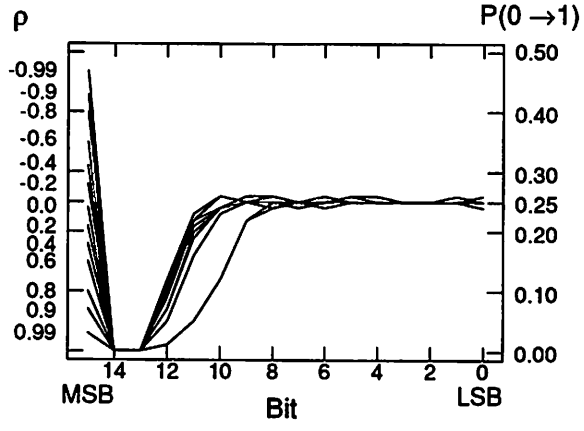
As a demonstration of model accuracy, the power of a multiplier within an LMS noise cancellation filter was analyzed using IRSIM-CAP, as well as the DBT and UWN models. IRSIM-CAP placed the average capacitance switched per cycle at 40.2pF, while the DBT model predicted 41.9pF - an error of only 4.2%. In contrast, the UWN model put the multiplier capacitance at 64.8pF - a substantial overestimate of 61.2%. Figure 3-9 from Chapter 3 is repeated here as Figure 4-23 to provide the reader with additional multiplier results.

4.1.6 Limitations and Extensions

As stated from the outset, some of the details of the DBT model depend on assumptions about data representation, stationarity of statistics, and logic style. Still, the model does provide important insights into the key factors that must be considered when deriving any activity model. Moreover, many of the assumptions can be relaxed through relatively simple extensions to the



(a) Two's-Complement



(b) Sign-Magnitude

Figure 4-24 : Comparison of bit activities for two's-complement and sign-magnitude

model.

For instance, although the DBT model was originally developed for two's-complement representation, we could certainly consider extending it to other representations such as sign-magnitude. As shown in Figure 4-24, this could be done by considering three, instead of two, activity regions: a UWN region, a one-bit sign region, and a zero-activity intermediate region. In a similar fashion, one could consider extending the DBT model to handle other data representation

such as floating point.

Stationarity of signal statistics was another assumption, implicit particularly in the breakpoint equations. If, in fact, data statistics such as mean, variance, and correlation adapt over time, the validity of the model could be affected; however, often even non-stationary signals exhibit stationarity during short segments of time. Therefore, one technique for handling changing statistics is to use a windowing approach. In other words, the analysis period could be segmented into several time windows, during which the signal statistics are approximately stationary. The DBT analysis techniques could then be applied to the successive windows and the results over all windows could be averaged.

It is also reasonable to expect different logic families to require modifications in the modeling strategy. For example, here we have assumed a logic style for which dynamic power consumption dominates over static. This is a good assumption for most CMOS logic families, but there are instances in which modeling static power might prove useful. A possible solution to this dilemma would be to characterize modules not only for capacitance switched, but also for static current flow. This static power component could be parameterized as a function of module complexity and added as an additional term to the overall power estimation equations. This strategy, of course, would require a characterization tool capable of modeling static power consumption (e.g. SPICE, PowerMill, etc.).

So, while there are limitations to the DBT model, many of them can be overcome by fairly simple extensions. The current DBT model is not necessarily an end in itself, but a foundation upon which more general datapath power analysis strategies can be built.

4.1.7 Datapath Summary

To review, previous attempts at architectural datapath power models have left room for improvement in terms of accuracy. These inaccuracies can be attributed to the fact that standard

UWN-based models ignore the influence of sign bits and signal correlations on power consumption. The Dual Bit Type (DBT) model presented here addresses this concern by accounting for both LSB and MSB (sign) behavior. Under this model, datapath library cells can be characterized for both types of bits. This process results in a table of effective capacitance coefficients, which allow the power analysis process to be reduced to little more than a series of table lookups. In addition, the DBT power analysis method uses parameterizable capacitance models that accurately reflect the effect of complexity, as well as activity on module power consumption. Aside from the coefficient table, the inputs to the analysis process are a small number of word-level statistics describing the data being fed to the module - in particular, mean, variance, correlation, and sign transition probabilities.

4.2 Memory

VLSI chips, of course, consist of more than just computational elements - data must also be stored and retrieved in memory units. Memory can either be foreground or background. Foreground memory usually consists of registers and register files, which are used to store small amounts of frequently accessed local data for rapid retrieval. Background memory typically refers to larger, denser, and slower storage units such as RAM's and ROM's, which may contain more global data or instructions. Memories form an important part of any realistic system, sometimes consuming 50% or more of the chip area and power [Dem90][Lip93][Wuy94]. Consequently, there is a strong motivation to accurately account for memory power consumption, as well as datapath (execution unit) power.

In this section, we develop a model for estimating the power consumed by registers, register files, and memories. As with the datapath elements, this model should account for the statistics of the data being read or stored. Also, the model should be parameterizable in terms of the key complexity parameters applicable to storage units - specifically, word length and storage capacity (in words). We should also be able to account for the control or function inputs to the module such

as Address, Read, Write, etc.

The DBT model, which has already been applied to computational modules, satisfies all these criteria. With relatively few modifications, we can extend the model to account for memory units as well. The primary difference between datapath and memory modules is that memories have *state*, whereas datapath modules typically do not. Circuits with state are sometimes referred to as sequential, while those without are called combinational. State in sequential circuits will require an extension of the DBT model to account not only for the previous and current value at the module inputs, but also for values that may have been stored many cycles in the past.

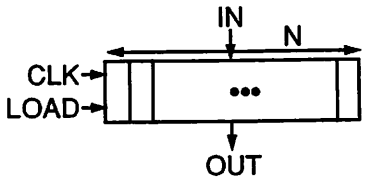
Section 4.2.1 and 4.2.2 contain the details required to apply the DBT model to memory modules. In particular, Section 4.2.1 describes how the model can be parameterized to account for the complexity of the memory. Section 4.2.2 then tells how the DBT activity model, with its capacitive data and control coefficients, can be modified to account for state. Finally, Section 4.2.3 presents results showing the accuracy of memory power modeling using the DBT method.

In each of these upcoming sections we shall consider the problem of modeling three types of storage units: registers, register files, and memories. The form of these units is shown in Figure 4-25. In this figure, W is the storage capacity of the units in words, while N is the length of the words stored in the memories.

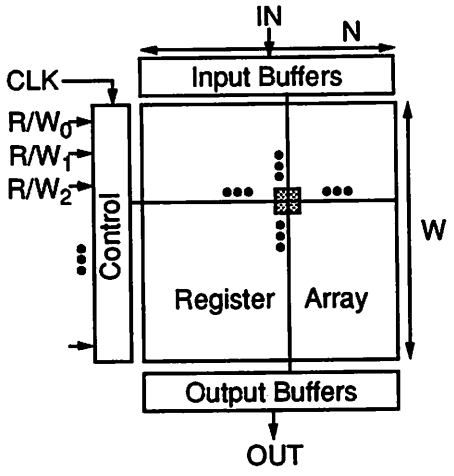
4.2.1 Complexity Model

The capacitance switched during an access to a memory module will be affected by the size, or complexity, of the module. Therefore, the capacitance formula must be parameterized in terms of the appropriate complexity measures. For the single register (Figure 4-25a), the only parameter required is the word length, N. As with many of the datapath elements already discussed, we expect the total module capacitance to scale linearly with word length. Thus, for the lone register:

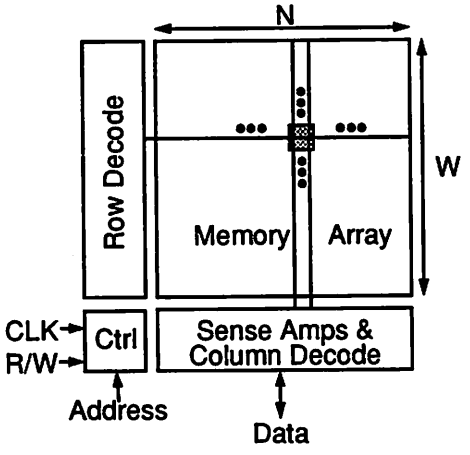
$$C_T = C_{eff} N \qquad \text{(EQ 82)}$$



(a) Register



(b) Register file



(c) Memory

Figure 4-25 : Structure of three common types of storage units

A register file offers a slightly more complex example. Such a unit contains a total of W registers with N bits each. In determining the appropriate capacitance model for the register file we refer to Figure 4-25b. When we read from or write to a register, we activate the input buffers, the output buffers, and a single register in the file. Each of these contains N bits. We therefore expect a capacitance term proportional to N. We also must broadcast the data across the entire register file traversing W words for each bit. This gives us a WN term in the overall capacitance expression. In addition, there might be some control overhead for each of the W registers in the file giving rise to a W term in the formula and, perhaps, a constant overhead term for the entire file. Weighting each of these by a capacitive coefficient, gives us an aggregate capacitance model of:

$$C_T = C_0 + C_1 W + C_2 N + C_3 WN \qquad \text{(EQ 83)}$$

or in vector notation:

$$C_T = \mathbf{C}_{eff} \cdot \mathbf{N} \qquad \text{(EQ 84)}$$

where $\mathbf{C}_{eff} = \left[C_0\ C_1\ C_2\ C_3\right]^T$ and $\mathbf{N} = \left[1\ W\ N\ WN\right]^T$.

Background memories such as SRAM's, DRAM's, and ROM's have a capacitance model identical to the register file with some slight difference in interpretation. The dominant term comes from charging and discharging the bit lines. The number of bit lines is proportional to the word length of the memory, N, and the length of each bit line is proportional to the number of words contained in the memory, W. Therefore, the capacitance switched during the charging of the bit lines is proportional to WN. Another component of power consumption stems from the word line driver, column decode, sense amplifier, and output driver circuitry. Each of these terms is proportional to the number of columns in the memory and, therefore, the capacitance model should contain a term proportional to N. In addition, there are the row decoders to consider. While typically implemented in a logarithmic tree structure, the final driver stage should dominate the power and contains W elements. Again, we include a constant overhead term. The result is a capacitance model identical to that of the register file:

$$C_T = C_0 + C_1 W + C_2 N + C_3 WN \quad \text{(EQ 85)}$$

This model assumes a memory partitioned into a single block of cells. Often, memories will be partitioned into several blocks, with only a subset powered-up for any one access. In this case, the form of the model remains the same, but the interpretation of the W and N parameters must be modified. Specifically, W should be interpreted as the number of rows in the active memory block and N should be interpreted as the number of columns in the same block. Aside from this slight modification, the overall capacitance model remains unchanged.

These parameterized capacitance models account for how the total memory power scales with the size of the unit. Alone, however, they do not account for the effect of the input activity seen by the module.

4.2.2 Activity Model

As with the datapath model, we must consider how to model the activity associated with two types of memory inputs: data and control.

Data Inputs

Since memories contain state, power consumption can be affected not only by inputs presented to the module at the current and previous clock cycles, but also by data that was stored at some point further in the past. As the capacitive coefficients used by the DBT model are intended to reflect transitions in the data associated with a module, state information should be represented in these coefficients. The precise interpretation of state is determined by the function currently being performed: e.g. Read or Write. Therefore, a memory will have two sets of capacitive coefficients - one for the Read function and one for the Write function. In this section we consider both cases.

Consider first the simplest unit: the register. It is a single input module and will have capacitive coefficients of the form $\mathbf{C}_{T_{t-1}T_t}$, where T_{t-1} is the bit type (U, +, or -) of interest before a

transition and T_t is the bit type after the transition. But when we say "the bit type of interest" it remains to be seen to what signal we are referring: the input, the value stored in the register, or the output. This depends on whether the register is engaged in a Read or a Write operation. During a Read, capacitance is switched when the value previously at the module output is replaced with the current value stored in the register. Consequently, the T_{t-1} entry in the capacitive coefficient should be interpreted as the bit type of the previous output value. Similarly, T_t would represent the bit type of the new value being driven to the output - that is, the current value stored in the register. For a Write operation, the interpretation is different. Here capacitance is switched when the value in the register at the previous cycle is overwritten by whatever value is now sitting at the input. Here, T_{t-1} should be interpreted as the current register value (i.e. the state), while T_t reflects the input bit that will overwrite it. If transition ID's and capacitive coefficients are interpreted in this manner, the DBT model can be applied to a register just as successfully as any of the previously discussed datapath modules.

The register file can be handled in a similar fashion. The only difference is that the module now contains many stored values rather than a single value so we must make a distinction as to precisely which register is being accessed. For example, in the context of a Read operation the bit type, T_{t-1}, should refer to the value at the output before the Read occurs, while T_t should reflect the value stored in whichever register is being read. Likewise, for a Write operation, T_{t-1} would correspond to the previous value stored in the register of interest and T_t would describe the input value which is about to be written into that register. So basically, the only difference between the register and register file is that for the file we must always explicitly refer to the register being accessed, while for a single register this is implicit. Care does have to be taken, however, when gathering activity statistics to be used in power estimation. For example, the activity for a Write to a register cannot be ascertained merely by looking at the previous and current inputs to the file since the previous Write may have been to another location entirely. Instead, the activity analysis must follow the above strategy, taking into account memory location, when determining what the

"previous" and "current" data values really are.

Memories can be handled in a manner identical to register files. In this case, however, the index to the register being accessed is replaced by a row address. Aside from this, the interpretation of the capacitive coefficients is the same.

From this discussion, we see that the Dual Bit Type model applies equally well to both datapath modules and data storage units. The capacitance models for the storage units can be parameterized to reflect the impact of word length and storage capacity on the total module capacitance. Also, the capacitive coefficients can be tailored to handle the state of these sequential components and properly account for temporal correlation and signal activity.

Control Inputs

While we now know how to handle data transitions, we have yet to discuss the effect of the control inputs. The control inputs of a memory module determine the function that the module will perform (e.g. Read or Write) and the location or address that the function will access. As with the datapath modules already discussed, the value of these control signals modify the operation performed by the module and, therefore, can have an effect on power consumption. Likewise, transitions on the control lines of storage modules can initiate internal state transitions just as transitions on the data inputs can. We already discussed how the effect of control inputs can be modeled for multi-function datapath modules. This section now reviews that discussion in the context of memory modules.

For a simple register, the primary control signal of interest is the LOAD signal. Based on this flag, the module can perform two functions: Write and Retain. The Write function stores a new data value in the register and, typically, propagates the new value to the output (performing an implicit Read operation). Therefore, a register should have one set of capacitive coefficients which characterizes the Write (and implicit Read) function. The Retain function would also have an

associated set of capacitive coefficients that would be characterized with the LOAD line disasserted. It may at first glance appear that no capacitance would be switched in this mode, since no input data transitions can propagate through the register; however, data input and clock transitions will switch the input capacitance of the module and, therefore, some power can be consumed in the Retain mode. In summary, for static values of the control inputs, the register is handled like any other multi-function datapath module.

Transitions on the LOAD signal, however, are handled differently than control transitions for multi-function datapath modules. The majority of energy consumed by a register is due to overwriting previously stored values or driving new values onto the outputs. These energies are characterized by the Write and Retain data coefficient sets. Therefore, it is not worthwhile to have a separate control coefficient table for characterizing the energy due to transitions on the LOAD signal. Instead, we model the capacitance by using the data coefficients alone.

The R/W control inputs for the register file and memory can be handled exactly as the LOAD signal for the register. The only additional control signal to consider is the address input, which describes what register or memory location is being accessed. When the address lines are fixed, their value typically does not affect the energy consumed by a Read or Write operation. In other words, the power consumption would be the same whether we write a value into address 5 or 10. When the address lines make a transition, the capacitance switched is in most cases dominated by reading or writing the new memory location in the array and not by the address decoding. Thus, as in the case of the register, we need not be concerned with modeling control input transitions. Capacitive data coefficients, one set for each module function, are usually sufficient to accurately model memory power consumption. The state issue is handled by properly interpreting the "previous" and "current" data values when gathering memory activity statistics and determining transition ID's as described in the previous section. If this turns out to be insufficient, the techniques described for modeling control input transitions for multi-function datapath modules can be applied equally well to the memory modules.

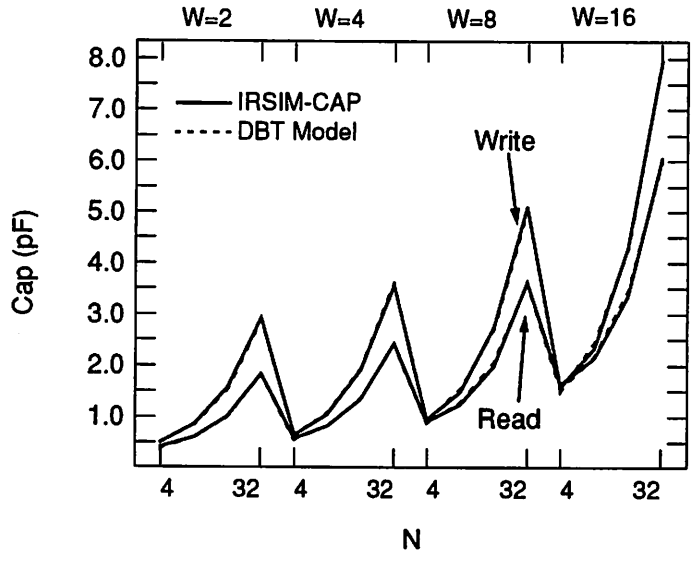


Figure 4-26 : Register file: IRSIM-CAP vs. DBT model

4.2.3 Results

This section validates the preceding discussion by presenting results for both a foreground memory module - namely, a register file - and a background memory module - an SRAM. In particular, Figure 4-26 shows the results of a comparison between the DBT model and switch-level simulations for register files of varying sizes. For the Read operation, the maximum error is only 5.7%, and it is even lower for the Write operation at about 4.2%. Similarly, for the SRAM Read and Write operations, the memory model provides estimated capacitances within 7.7% and 0.63% of IRSIM-CAP, for memories ranging in size from 16x4 to 128x32 (see Figure 4-27).

4.2.4 Memory Summary

In conclusion, memory accesses can consume a substantial portion of the total power budget for many applications. This section demonstrated that the DBT method is capable of modeling memory power consumption as well as it handles datapath modules. The state associated with

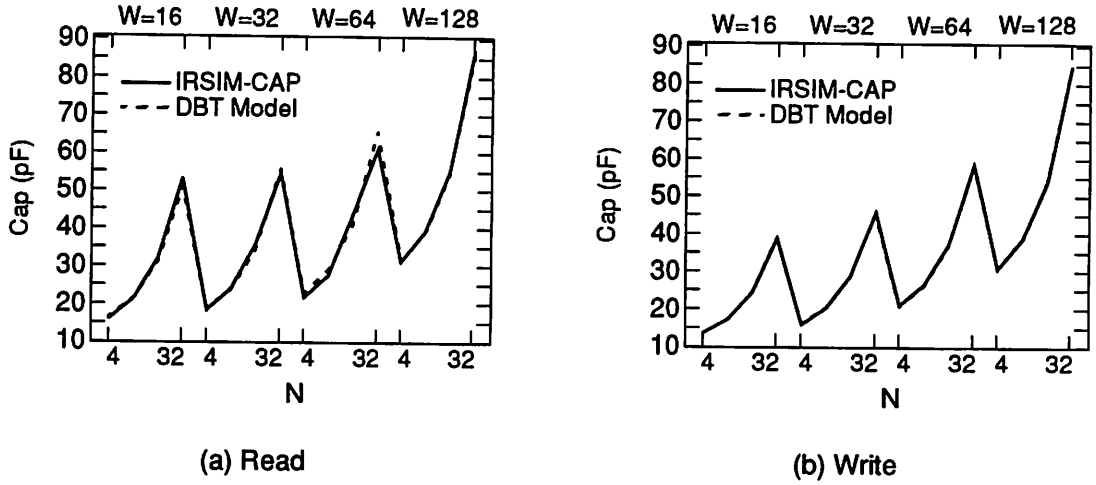
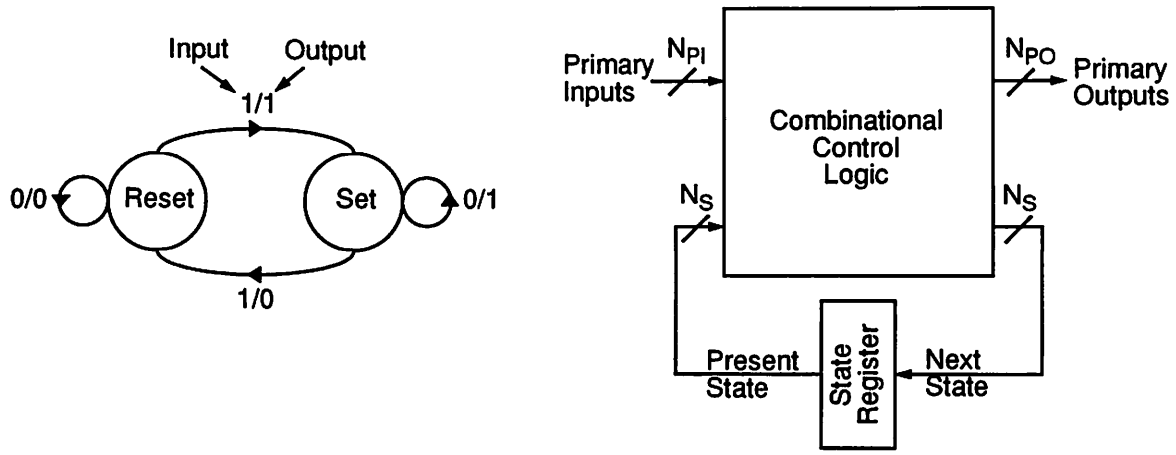


Figure 4-27 : SRAM: IRSIM-CAP vs. DBT model

memories can be handled by a change in how the bit types of the transition ID's and capacitive coefficients are interpreted. The resulting models are quite accurate and reflect the effect of both storage capacity and data activity on memory power consumption.

4.3 Control Path

The control path refers to those components in a system that determine or *control* the macro-behavior of the rest of the system. Controllers direct the sequence of operations to be executed by the datapath, they initiate memory accesses, and they coordinate data transfers over interconnect. In order to accomplish this, the typical controller steps through a series of *states*, where the exact sequence of states and output control signals is determined by the present state and the primary inputs to the controller, which often correspond to status flags from the datapath. Figure 4-28 shows a sample state transition graph (STG) and the associated finite state machine (FSM) controller that implements it. The nodes in the STG correspond to states, and the edges correspond



(a) State transition graph for T flip-flop

(b) Finite state machine controller structure

Figure 4-28 : Illustration of typical STG and FSM structure

to transitions between states. The edges are labeled with the input condition that initiates the particular transition. The output control signals can be a function of the state, the inputs, or both. Equivalently, this STG can be represented by a *control* (or *state*) *table*. This representation contains the same information, but lists the possible input and present state values and their associated output and next state values in tabular form. In either case, if the outputs are a function only of the state, the controller is referred to as a Moore machine; otherwise, it is called a Mealy machine.

A typical partitioning of a controller, as shown in Figure 4-28b, consists of some combinational logic to generate the next state and the outputs, as well as a state register to store the present state and feed it back to the input of the combinational logic block. The implementation of the combinational logic can take many forms. For instance, the functionality can be implemented in random logic using standard cells. Alternatively, the block can be implemented as array logic in the form of a ROM or a PLA.

The task of architectural controller power analysis is to produce an estimate of the final implementation power given only the target implementation style and a description of the state machine to be realized, say, in the form of an STG or control table. It is also possible to envision algorithm-level power prediction where estimates are based solely on the STG without any information about how the controller will be implemented. This is a difficult problem since the power that the controller consumes depends strongly on the target implementation style. For example, a ROM-based implementation will most likely consume a different amount of power than a standard cell implementation. Therefore, at the algorithm level it is probably most useful to provide the user with a range of possible power consumptions. This can be accomplished by invoking target-specific architectural power analysis techniques for several available implementation styles. So, in any case, the problem reduces to one of estimating controller power consumption assuming the implementation style has already been specified.

Still, this does not solve the difficulties entirely. Even if the implementation style is known, the exact circuit-level or gate-level implementation of the controller depends on the STG being realized. This makes the task of controller power estimation more difficult than datapath estimation. Since the circuitry of the datapath modules was known a priori, the physical capacitance of the modules was in effect fixed. The only variable was activity, which was handled by characterizing the modules for various types of input transitions. In contrast, the physical capacitance of a controller depends on the STG being realized, as well as the state and control signal assignments being used. This is not only the case for random logic controllers, but is also true for ROM- and PLA-based controllers. For example, the binary encoding of the state and control signals will determine the number of ones and zeros in the input and output planes of a ROM or PLA. This, in turn, determines the presence or absence of transistors in the array and alters the physical capacitance of the controller.

One obvious solution would be to synthesize the controller from the finite state machine description at "run-time" (i.e. during power analysis) and use a gate-level power estimation tool on

the resulting implementation. This may be reasonable for small controllers, but at some point the time required to synthesize, extract, and simulate a large controller would become prohibitive. Moreover, this merely sidesteps the issue of how to perform true *architecture-level* power analysis for controllers. Instead, we would like some technique for characterizing classes of controllers purely at the architecture level.

Since it is impractical to characterize each class of controller for all possible control table contents, we need to employ an approximation that will allow us to assume a "data-independent" physical capacitance. A sensible first-order approximation is to characterize each class of controller for random control tables. This will result in a fixed, average physical capacitance for the controller that is somewhere between the extremes of an "empty" control table (all zeros) and a "full" control table (all ones). Under this approximation, the physical capacitance of the controller is now independent of the "data" stored in the control table.

Then control modeling can proceed in a manner quite similar to datapath modeling. In particular, for each implementation class (e.g. ROM, PLA, random logic) prototype controllers of different complexities can be synthesized a priori and characterized for various activity levels. This will result in target-specific models that, like the datapath models, account for both physical capacitance and activity. The exact form of the models will be target-specific; however, the complexity and activity parameters used by the models are fairly independent of the target platform. Therefore, the discussion of controller power modeling will be divided into two parts. Section 4.3.1 will describe target-independent parameters that influence controller power consumption, while Section 4.3.2 will present target-specific power models. This will be followed by Section 4.3.3 which will discuss techniques for characterizing the power models and Section 4.3.4 which will review the controller power analysis method being proposed here. Since the method accounts explicitly for the effect of activity on power consumption we will refer to it as the Activity-Based Control model, or ABC model.

4.3.1 Target-Independent Parameters

The first step in developing a model for architectural controller power analysis is to determine what high-level factors influence the power regardless of the target implementation style. For the datapath, two influential classes of parameters were recognized: complexity parameters and activity parameters. The same distinction can be made for controllers. These two classes of parameters and their influence on controller power are described in the following two subsections.

Complexity Parameters

From previous discussions, we know that the size or complexity of a module directly influences its physical capacitance and, therefore, its power consumption. The complexity of a controller can be measured in several ways depending on how it is implemented. For a ROM implementation, the complexity might be measured in terms of the height and width of the memory array. For a PLA implementation, the complexity metric might include the number of product and sum terms in the AND (input) and OR (output) planes. The same measure might be used for random logic implementations.

In order to perform architecture-level power analysis for controllers it is important to identify which high-level parameters will affect complexity. Ideally, these parameters should be fairly general and apply in some form or another to all target implementation styles. Also, since we are discussing complexity and not activity, the parameters should be independent of the particular input, output, and state values of a given FSM and relate more to the overall complexity of the STG. Three basic parameters fit these criteria: the number of states in the FSM, the number of inputs, and the number outputs.

Intuitively, the *number of states, S*, or equivalently, the *number of state bits,* $N_S = \lceil \log_2 S \rceil$, in an FSM should have some relationship to the complexity of the final implementation. If there are many states, a large number of outputs and next state possibilities must be stored (ROM) or generated (PLA, random logic). Fewer states will require correspondingly less storage and/or

logic. There are, of course, degenerate cases where a large number of states can be implemented with a relatively simple controller. For example, even a controller with 1000 states could be implemented with only a 10-bit counter if the states progressed sequentially; whereas, a random progression of states might require a large amount of logic to implement. Notice, however, that exploiting the degeneracy of the STG requires a particular implementation style (in this case a counter). If the controller had been implemented as a ROM, any sequence of states would have resulted in the same complexity. So in a discussion of target-independent complexity parameters, it is best to neglect degenerate cases and use the state count as an initial measure of controller complexity.

Similarly, the *number of primary controller inputs*, N_{PI}, can be used as a generic measure of complexity. Referring back to Figure 4-28b, we see that the controller inputs feed the combinational logic block in the same manner as the present state lines. So in terms of controller complexity, the distinction between the number of controller inputs and the number of state bits is a somewhat artificial one. Consequently, an aggregate input complexity measure, $N_I=N_S+N_{PI}$, can be used to remove the unnecessary distinction. Both sets of lines together form the STG "address" that must be decoded in some manner or another by the combinational logic block. Therefore, the total number of inputs, N_I, is the appropriate input complexity measure for the controller.

As might be expected, the *number of primary controller outputs*, N_{PO}, is equally important in determining the complexity of the controller. Given a decoded state/input "address", there must be some logic or memory cells that provide the required outputs. The more output signals there are, the more computations there are to be performed. As a result, we would expect the controller complexity to grow with the number of output control signals. Since next state values must also be produced, the aggregate output complexity, $N_O=N_S+N_{PO}$, provides a more complete measure of the output-plane complexity.

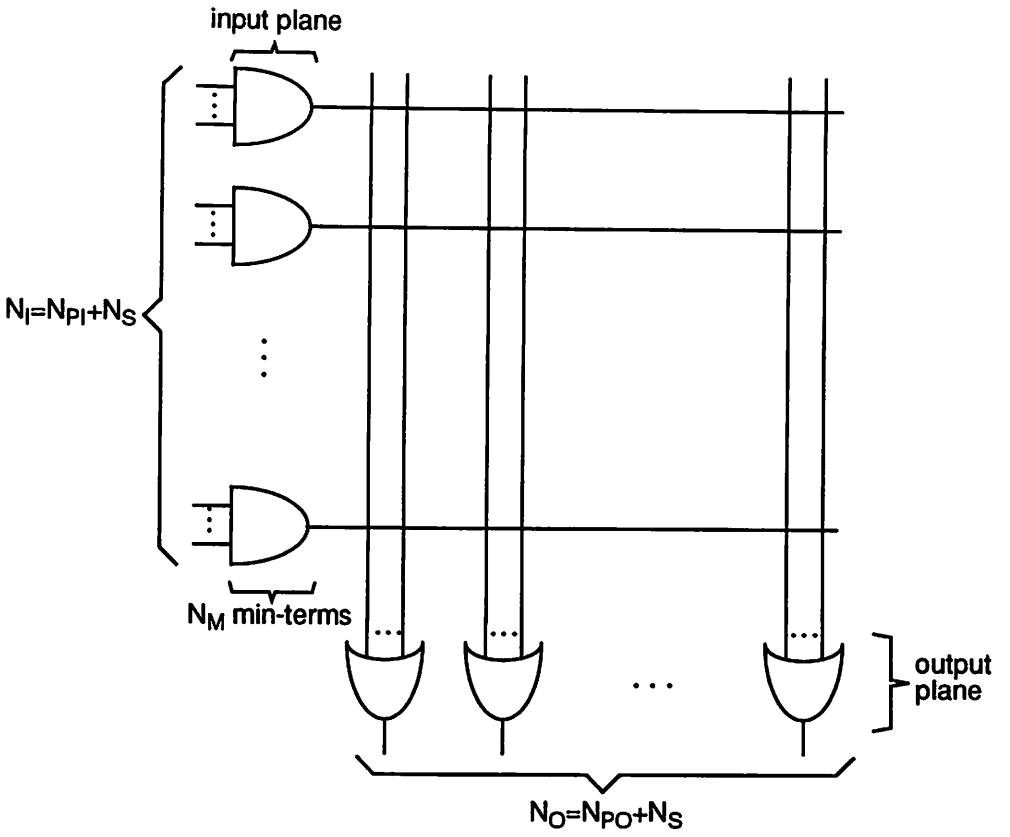


Figure 4-29 : Two-level logic template and its associated complexity parameters

The situation is summarized by Figure 4-29, which is useful for visualizing how the various parameters discussed here can affect implementation complexity. The figure contains a template for the canonical sum-of-products implementation of some arbitrary combinational control logic. The input plane decodes the primary inputs and present state bits to produce a number of product terms, or *min-terms*, which are then summed by the output plane to generate the primary outputs and next state bits. The complexity of the entire structure depends on the number of product terms, as well as the number of sum terms. The number of sum terms is simply the total number of primary and next state outputs, N_O. The number of product terms, however, cannot be determined from N_I alone unless exhaustive "address" decoding is used (as in a ROM). In other cases, an additional parameter, N_M, describing the number of min-terms in the controller is required.

Techniques for estimating N_M will be discussed in future sections. For now, suffice it to say that since the STG is assumed given, it is always possible to perform a trivial state assignment and logic minimization step (i.e. *espresso* [Bra84]) to provide a fairly accurate min-term estimate. Together, N_I, N_M, and N_O provide a reasonable estimate of the input- and output-plane complexities.

Activity Parameters

As has been mentioned in previous chapters, complexity (or equivalently, physical capacitance) is not the only factor determining power consumption. If the capacitance is not switched, no power is consumed. Activity refers to the frequency with which switching occurs. At the architecture level we would like to treat the combinational logic and state register blocks of Figure 4-28b as black boxes. With this in mind, the activity of the controller can best be described by three external activity measures: the state activity, the input activity, and the output activity.

The *state and input activity*, tell us something about how much switching will occur in the input plane, or "address" decoding, portion of the combinational logic. In a ROM-based implementation this would correspond to the row decoder activity, while for a PLA-based controller this refers to the AND-plane activity. For static logic, the transition activity, α_I, is an intuitive measure of circuit activity in the input plane and is equal to the fraction of input or state bits that switch each cycle. For dynamic logic, the signal probabilities of the inputs - that is, the probability that an input bit is one (P_I) or zero ($1\text{-}P_I$) - tend to have more of an impact on circuit activity. The reason for this was discussed in Section 4.1.2 and boils down to the fact that precharging negates the influence of the previous signal value on current power consumption. In summary, the input-plane activity can be adequately described by two activity parameters: the fraction of input bits that are one, P_I, and the fraction of bits that switch, α_I, during an average clock cycle. The input-plane activity tells only half of the story, however - we also need to have some estimate of the output-plane activity.

The *output activity* provides this measure. Note that the "outputs" of the combinational logic block include both the controller outputs, as well as the next state bits. At first glance it might appear that the output activity can be derived directly from the state and input activities; however, if we measure the input and state bit activities merely by the expected number of transitions on those bits each cycle, then we have made an implicit independence assumption. In reality the input signals and state lines may be correlated, which could have an impact on the activity in the combinational logic block. Only if we explicitly account for all input correlations can we actually propagate the correct activity through the combinational logic to the output plane. As we saw in Chapter 3, this exact propagation of probabilities to account for reconvergent fan-out and feedback is a difficult and expensive task. Therefore, it is more economical to measure the output activity in terms of signal probabilities (P_O) and transition activities (α_O) directly. Section 4.5.2 will describe techniques for acquiring the necessary activity factors through functional simulation.

So far this section has presented several parameters that can affect the power consumption of a controller implementation through either its physical capacitance or its switching activity. The physical capacitance was related to complexity parameters such as the numbers of states in the FSM, the number of inputs, and the number of output signals. In addition, the activity of the state, input, and output bits were parameters that described the switching activity of the controller. These parameters will influence the power of the controller regardless of the implementation style. The exact manner of influence, however, will vary from implementation to implementation. Precise target-specific models for estimating controller power is the topic of the next section.

4.3.2 Target-Specific Capacitance Models

The purpose of the capacitance model under the ABC method is to predict how the effective capacitance of a particular controller class will scale with changes in complexity and I/O activity. This relationship differs for each target implementation style much like the DBT model of a shifter differs from that of an adder. This section illustrates how to construct an ABC capacitance model

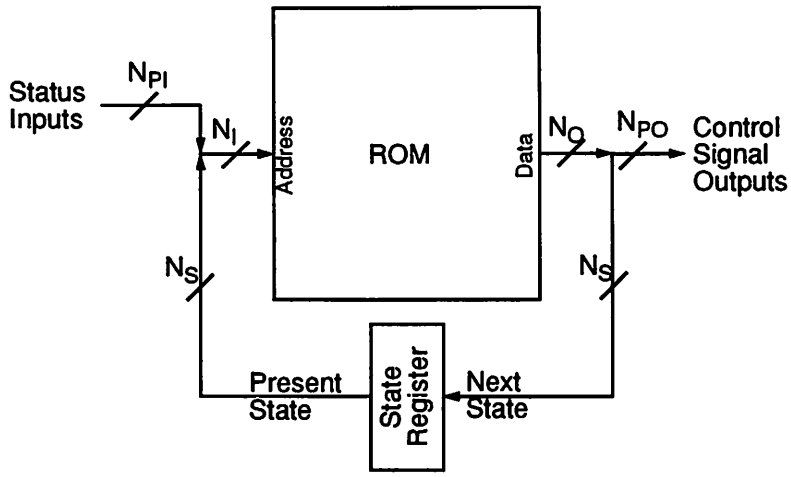


Figure 4-30 : ROM-based controller

through three case studies: a ROM-based controller, a PLA-based controller, and a random logic controller. While the case studies presented here provide examples of how to compose ABC capacitance models for some specific controller implementations, the same concepts are readily applicable to other design styles.

ROM-Based Controllers

One technique for implementing the combinational logic portion of an FSM is with a ROM as illustrated by Figure 4-30. The power consumed by this ROM-based controller will be affected by the complexity or size of the ROM, as well as the activity associated with the ROM input and output planes. The task of estimating the power consumed by a ROM-based controller, then, reduces to finding a model that relates the appropriate complexity and activity parameters to the average capacitance switched within the ROM each clock cycle. As with the datapath modules, the exact form of the model can vary from implementation to implementation. In other words, just as a barrel shifter and a log shifter might have different capacitance models, a ROM employing a static row decoder might have a different capacitance model than one using dynamic decoding. To address this issue, we again take a library-based approach, allowing the user, if he desires, to

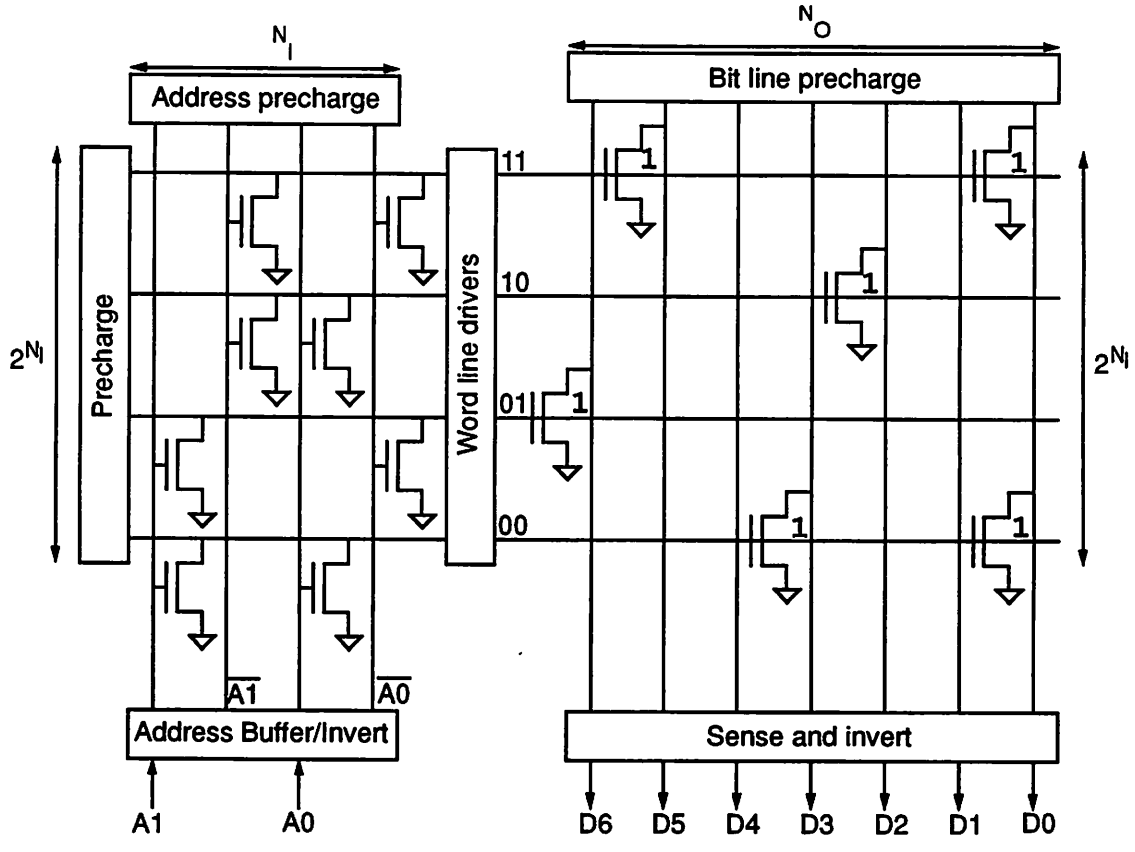


Figure 4-31 : Basic structure of prototype ROM (4x7 in this example)

define a unique capacitance model for each ROM implementation in the library. If no ROM's are available the model could either be based on some assumed typical structure or on measurements taken from data books or previous implementations.

For the purposes of illustration, we will demonstrate the modeling procedure using the prototype ROM structure of Figure 4-31. The capacitance model appropriate to this structure is:

$$C_T = C_0 + C_1 N_I 2^{N_I} + C_2 P_O N_O 2^{N_I} + C_3 P_O N_O + C_4 N_O \qquad \text{(EQ 86)}$$

The terms in this expression relate to power consumed in the input plane (address decoding) and the output plane (bit lines). Each term will be explained in the following discussion.

Starting with the input plane, we must determine the appropriate measures of complexity

(physical capacitance) and activity. The width of the input plane is proportional to the number of address inputs, N_I. For a ROM-based controller this is the sum of the number of primary input bits and state bits for the FSM. The height of the input plane is proportional to the number of addresses that can be formed from N_I bits - specifically, 2^{N_I}. The complexity of the input plane is the product of the width and height: $N_I 2^{N_I}$. The physical capacitance in the input plane is composed of the gate and drain capacitances of the decoding transistors and the wiring in the block. Since ROM's employ full address decoding, the number of transistors in the input array is not related to the control table contents in any way. Instead, the input-plane capacitance relates only to the complexity measure $N_I 2^{N_I}$.

After complexity, the next concern is activity. For the prototype ROM under consideration, the address decoding is implemented dynamically with all address lines precharged high. In addition, since true and complement address lines are present, half of the lines will always remain high and the other half will discharge. So for this particular case, the activity statistics of the inputs will not affect the ROM power consumption, and the capacitance switched in the input plane during an access is given by $C_1 N_I 2^{N_I}$.

In the output plane, the power consumption is dominated by charging and discharging the bit lines. The number of bit lines is proportional to the number of output bits, N_O. The physical capacitance of each bit line is contributed by wiring and by transistors placed in the memory array for each 1 stored in the output plane. Under the aforementioned approximation that the output plane is filled with random data - half zeros and half ones - the physical capacitance of the bit lines is simply proportional to the height of the array, 2^{N_I}. The combined complexity term, $N_O 2^{N_I}$, describes the total physical capacitance in the output plane. Next, consider the activity on the bit lines. The bit lines in this particular memory design are precharged high and then discharge conditionally for 1 output bits (Note: for this ROM, the bit lines carry complemented data). Therefore, the capacitance should be proportional to P_O, the average fraction of 1 output bits. So the capacitance switched in the output plane will be $C_2 P_O N_O 2^{N_I}$.

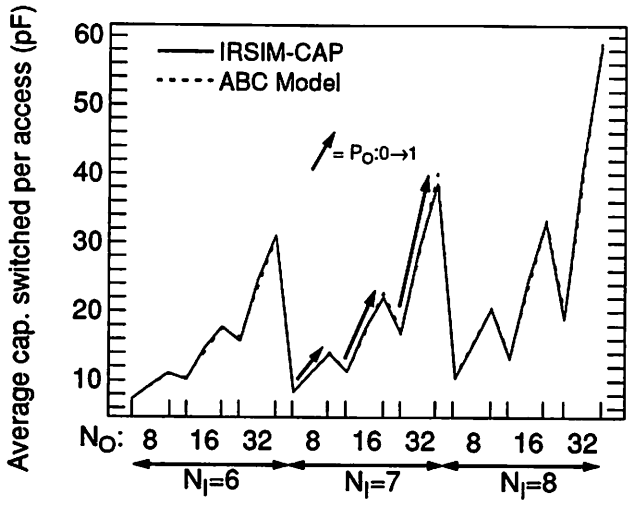


Figure 4-32 : ROM-based controller: IRSIM-CAP vs. ABC model

The values on the bit lines must be sensed and driven to the outputs. The complexity of this circuitry is proportional to the output width, N_O. As in the output plane, activity considerations introduce a $C_3 P_O N_O$ term; however, since the sense circuitry in this memory produces both true and complemented signals, there must also be an activity-independent term, $C_4 N_O$.

Combining all these terms yields the ROM capacitance model of (EQ 86), where C_0, C_1, C_2, C_3, and C_4 are capacitive coefficients dependent on the exact circuitry and technology used by the ROM. Just as for the datapath and memory models, these coefficients are extracted through a library characterization process that will be described in Section 4.3.3.

The precise form of the capacitance model can be specified by the user or library designer and may differ from one ROM implementation to another. For the ROM example discussed here, Figure 4-32 shows the results of a comparison between switch-level simulations and the above model after characterization in a 1.2 μm technology using random control table contents. The rms error over all cases is about 2.5% and the maximum error is 4.5%. The arrows in the figure denote results for controllers of fixed complexity for which the output signal probability, P_O, varies from zero to one. The fact that power consumption varies significantly with P_O is a strong argument in

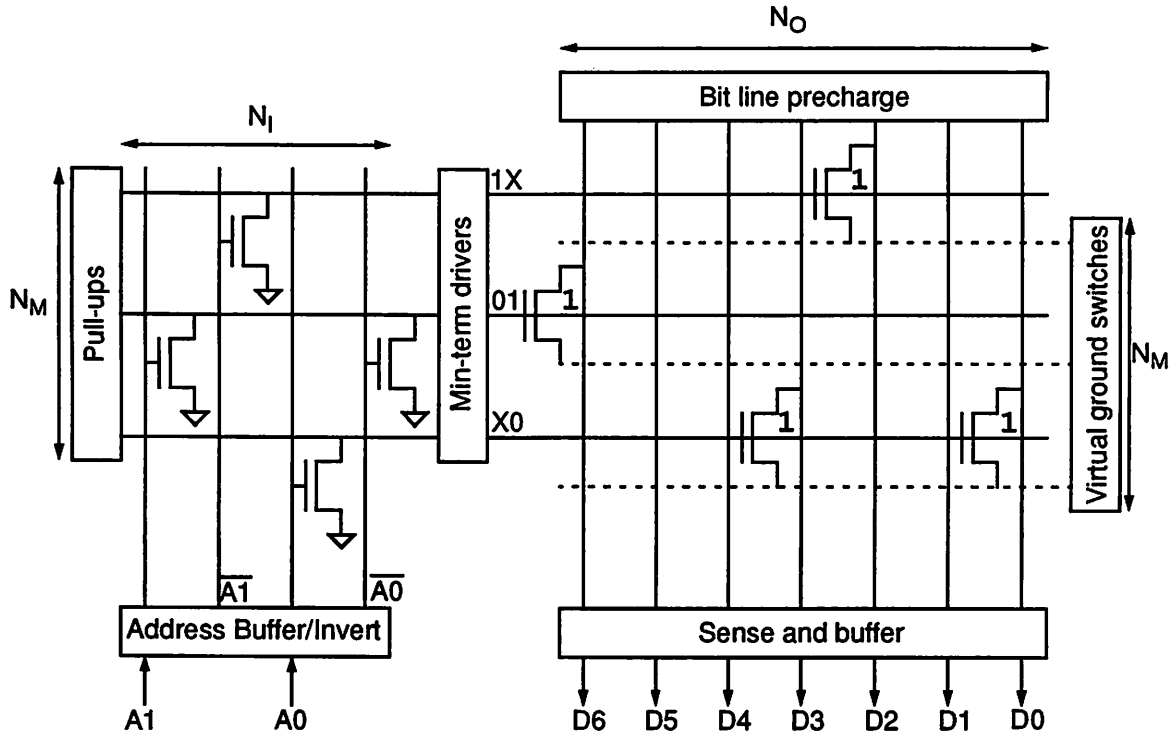


Figure 4-33 : Basic structure of prototype PLA

favor of a modeling strategy (such as the ABC technique) which accounts for activity.

PLA-Based Controllers

Conceptually, the structure of a PLA is quite similar to that of a ROM. Both have input and output planes that implement the product and sum functions, respectively, of a two-level sum-of-products logic expression. The principle difference between a PLA and a ROM is that the input plane of the ROM performs a full decoding of all possible addresses, while a PLA uses logic minimization to reduce the amount of decoding required. As a result, the height of the input and output planes in a PLA will actually be less than 2^{N_I}. Instead, the height will be given by the number of unique min-terms, N_M, in the minimized sum-of-products output expressions.

Consider the PLA of Figure 4-33. As done for the ROM, this section will first present the

appropriate capacitance model and then describe how each term arises. Note that different PLA designs may require slightly modified capacitance models. This particular PLA has a static input plane and a dynamic output plane and obeys the following model:

$$C_T = C_0\alpha_I N_I N_M + C_1 P_O N_O N_M + C_2 P_O N_O + C_3 N_O N_M + C_4 N_M \quad \text{(EQ 87)}$$

Beginning with the input plane, the physical capacitance is proportional to the number of rows and the number of columns. The number of columns is just the total number of input bits, N_I. As stated above, however, the number of rows is, N_M, since only a partial address decoding is performed in a PLA. Since this PLA uses static rather than dynamic decoding, input *transitions* lead to power consumption and the appropriate activity parameter is α_I. Combining the activity and complexity measures for the input plane gives rise to the $C_0\alpha_I N_I N_M$ term.

The output plane complexity is the product of the number of rows, N_M, and the number of output columns, N_O. For this PLA, the bit lines are dynamically precharged. During evaluation, any 1's in the output table cause the corresponding bit line to discharge. Then, during the following precharge phase, these lines are returned to V_{dd}. Therefore, the output signal probability, P_O, is the appropriate activity parameter. Combining factors, the effective output plane capacitance term is $C_1 P_O N_O N_M$. The output drivers contribute an additional term: $C_2 P_O N_O$.

Notice that the output plane transistors connect to virtual rather than true ground. Virtual ground is separated from true ground by clocked ground switches. Opening these ground switches during precharge allows the bit lines to rise to V_{dd} even though the min-term outputs may still be asserted. A side-effect of this configuration is that the virtual ground plane charges to V_{dd} each cycle, regardless of the output signal values. Charging the physical capacitance of the ground plane and ground switches give rise to the activity-independent terms: $C_3 N_O N_M$ and $C_4 N_M$, respectively.

As in the ROM case, the precise values of the capacitive coefficients C_0-C_4 are derived during library characterization. For completeness, this characterization ranges over different values of

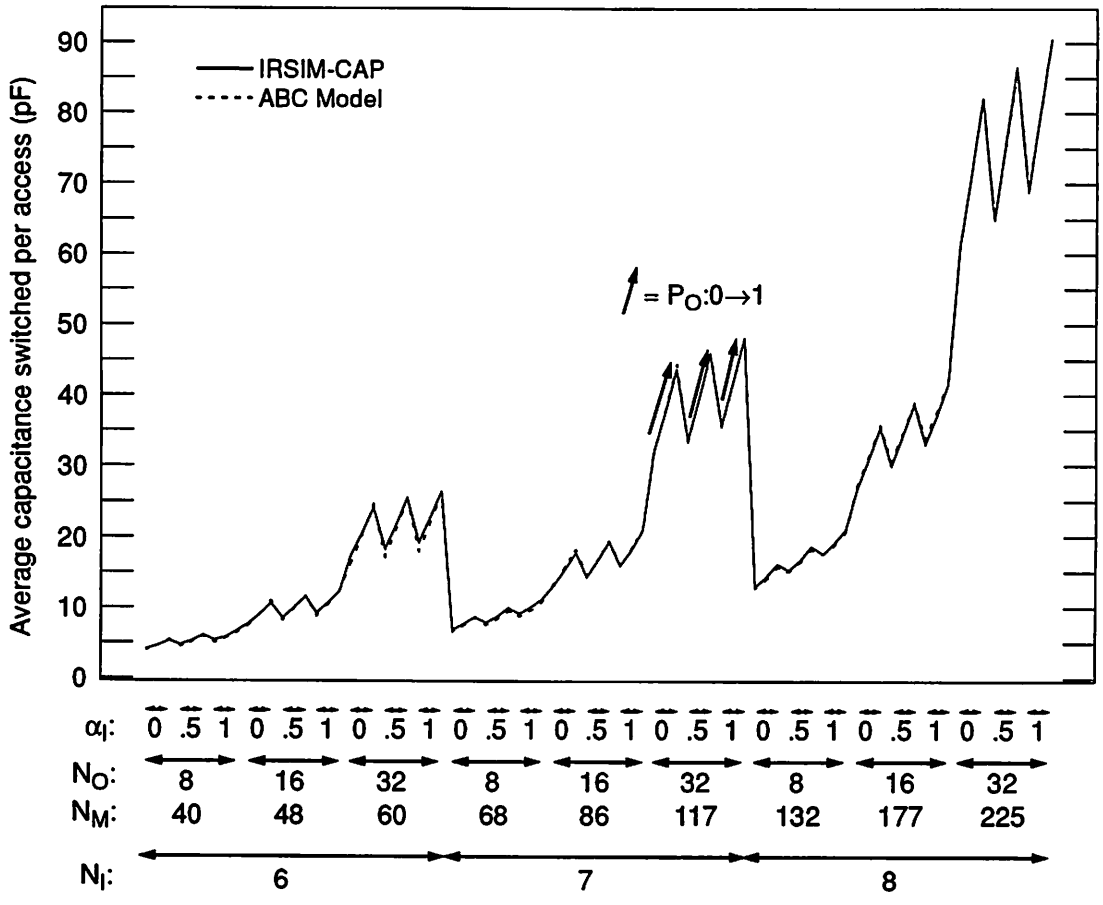


Figure 4-34 : PLA-based controller: IRSIM-CAP vs. ABC model

both the complexity and activity parameters. Notice that while the ROM had a single activity parameter, P_O, the PLA discussed here requires an input activity factor, α_I, as well. Other PLA implementations might require still different activity parameters. The important thing is that the user or library designer has the freedom to choose the model appropriate to each implementation.

The model resulting from characterization of the PLA discussed here (in a 1.2 μm technology) is compared to switch-level simulations in Figure 4-34, again for random controller contents. The model has an rms and maximum error of 2.5% and 6.3%, respectively. As in the ROM case, the arrow notation is used to denote regions where the power for a fixed-complexity controller varies solely due to changes in the input activity, α_I, and the output signal probability, P_O.

Random Logic Controllers

A controller implemented in random logic consists of two or more levels of boolean gates, which realize the next state and output tables of the FSM. Such a controller would typically be implemented in standard cells. Since this form of implementation is much less regular than a PLA or ROM structure, it is more difficult to come up with a precise capacitance model. Still, the activity and complexity parameters that influence the model remain the same. For example, a standard cell controller implemented in static logic could be modeled by:

$$C_T = C_0 \alpha_I N_I N_M + C_1 \alpha_O N_O N_M \qquad \text{(EQ 88)}$$

This expression contains two components - one relating to the input plane capacitance and the other relating to the output plane. Since this example is based on static logic, the appropriate activity measures are α_I and α_O, respectively. For the input plane, the complexity is given by the product of the number of inputs to that plane, N_I, and the number of outputs that plane produces, N_M. The same is true for the output plane, except in this case there are N_M inputs to the plane and N_O outputs.

Thus, for both cases the complexity is given by the product of the number of inputs and outputs for the given logic plane. The explanation for this is two-fold. First, since a certain amount of logic is associated with each output, the total power consumption should be proportional to the number of output bits. The number of inputs also has an influence on the power consumption, albeit a more subtle one. Multi-input logic functions are often implemented by several two-input gates. This is illustrated by Figure 4-35 which shows the decomposition of an N-input AND into N-1 two-input AND's. In this way, the capacitance of the computation is proportional to the number of inputs. Even if the function is implemented by a single multi-input gate, however, the number of transistors, and hence the physical capacitance, in that gate will be proportional to the fan-in. This reasoning culminates in the simple expression of (EQ 88).

The two capacitive coefficients are derived during a characterization phase and will be a

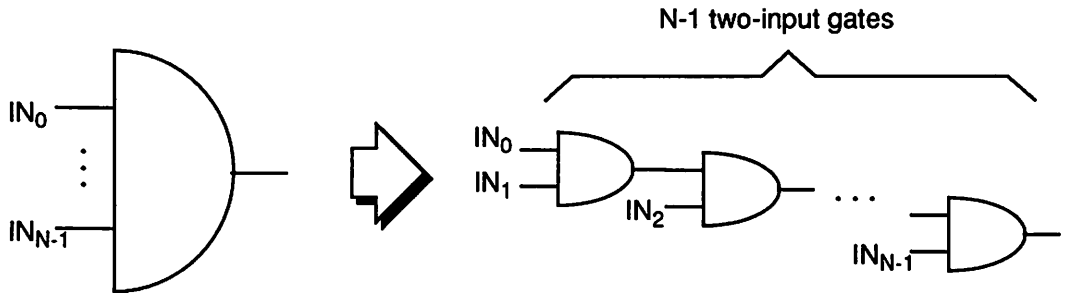


Figure 4-35 : Decomposition of *N*-input AND into 2-input AND's

function of the standard cell library being used and, to some extent, the logic minimization, placement, and routing tools being applied. A comparison of the characterized model to switch-level simulations for a 1.2 μm cell library is shown in Figure 4-36. The results are for control tables with random contents that have been minimized using espresso and synthesized using MIS [Lav90] and the Lager IV silicon assembly system [Bro92]. While the agreement is not as good as the ROM and PLA models, the rms error is still a respectable 15.1%.

4.3.3 Characterization Method

Before the controller capacitance models can be used, circuit- and technology-dependent values of the capacitive coefficients must be measured. The measurement process that produces these coefficients is known as characterization. Characterization of the controller models is quite similar to the datapath and memory characterization procedures that have already been discussed. For a given class of controller, the idea is to actually measure the capacitance switched within implementations of various complexities for different input and output activities. The resulting capacitance observations, which should span a wide range of the complexity-activity parameter space, are then used to find capacitive coefficients that give the capacitance models the best fit to the measured data. Like the previously discussed characterization methods, the process occurs in three distinct phases: pattern generation, simulation, and coefficient extraction.

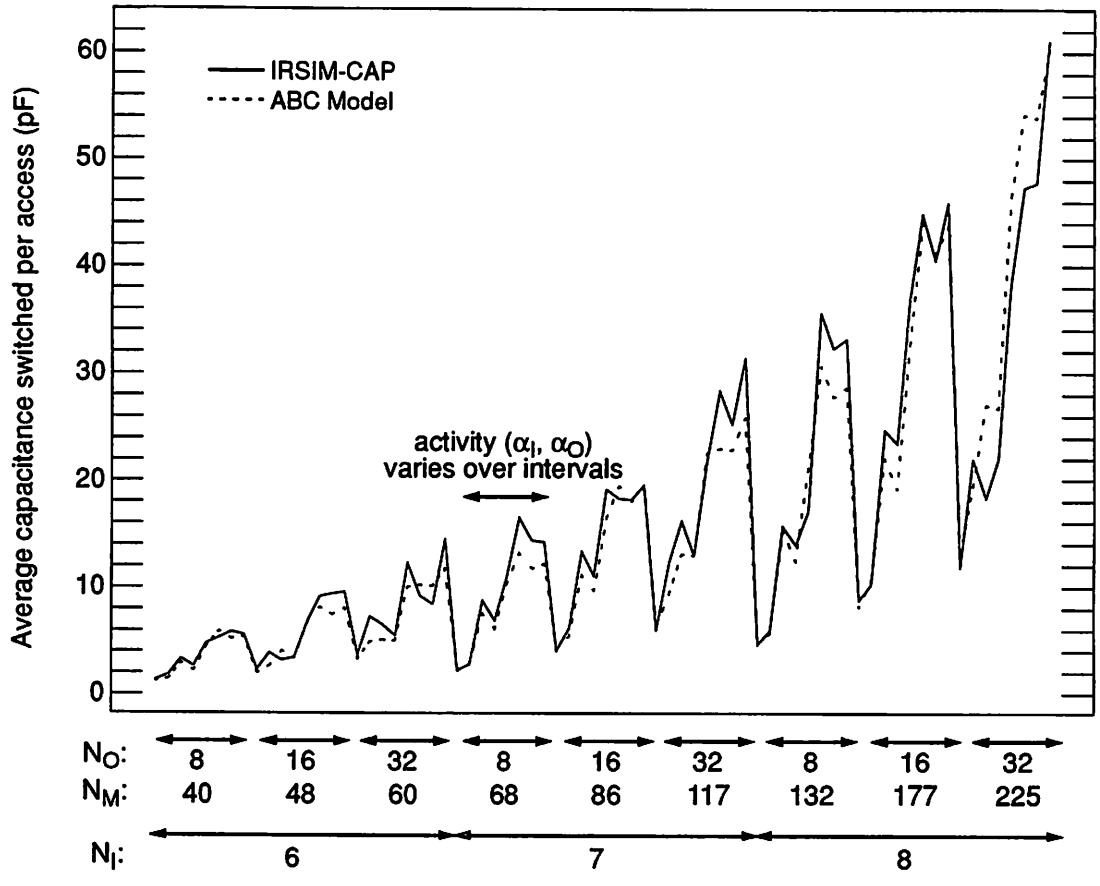
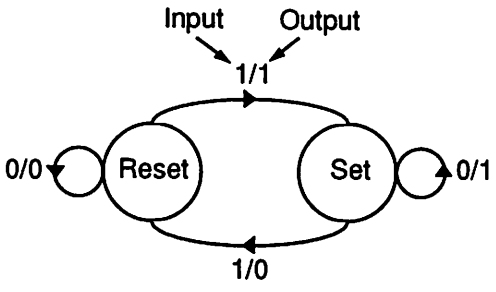


Figure 4-36 : Random logic controller: IRSIM-CAP vs. ABC model

Pattern generation refers to producing the data that will be used when taking the capacitance measurements. As in the datapath case, this includes generating input streams that will span a wide range of activity parameters. Unlike datapath modules, however, the physical capacitance and output behavior of controllers also depends on the data (i.e. control table) stored in the module. This data must be generated in a way that results in a representative "average" physical capacitance for the controller. In addition, the control table must be constructed in a way that facilitates precise control over the output activity. This allows the module to be characterized over a wide range of output activities, as well as input activities.



(a) State transition graph for T flip-flop

Input Address:		Output:	
A_1	A_0	O_1	O_0
PS	IN	NS	OUT
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

(b) Control table for T flip-flop

Figure 4-37 : STG and control table for T flip-flop

During simulation and coefficient extraction, the capacitance switched for the generated data patterns is measured and fit to capacitance models such as those derived in the previous section. The process is identical to that used for datapath and memory cells with the distinction that the parameter matrix now includes activity, as well as complexity factors. The result of the process is a capacitive coefficient vector specific to a controller class (e.g. ROM, PLA, standard cell), a circuit style, and a technology.

Pattern Generation

Controller characterization requires two distinct sets of data patterns. The first is the set of data patterns stored in the control table (or STG) that the FSM implements. The second is the set of input patterns that are applied to the controller during simulation. This section discusses how each of these data sets can be generated. In both cases, the pattern generation process lends itself well to automation.

The behavior of any FSM can be represented by an STG, or equivalently, a *control table*. The control table specifies for each present state and input pattern what the next state and output pattern should be. For example, Figure 4-37 shows the STG and control table for a T flip-flop,

which remains in the same state for a 0 input and toggles state for a 1 input. In this table, the input bits form an "address" input, which references a data value specified in the corresponding row of the output plane.

The physical capacitance of the controller can be affected by the data patterns stored in the input and output planes since for some implementation styles (e.g. PLA and standard cell), the presence or absence of a 1 can result in an increase or reduction in the number of transistors or gates in the design. Since it would be inefficient to characterize for all possible STG's, the goal of pattern generation is to produce a control table that results in some sort of average physical capacitance. A good approximation is to use a random output table. The values should be chosen from three possibilities: zero (0), one (1), and don't-care (X). The don't-care value arises since in real designs individual controller outputs may not affect the system behavior for all states. One technique for generating a "random" output table would be to first uniformly set half of the output bits to X. The remaining bits can then be set half to zero and half to one, again using a uniform distribution.

Since the ROM stores the entire input and output plane, regardless of contents, the don't-care values do not have an effect - they should simply be stored as half zero and half one just like the rest of the bits. The don't-care values do, however, have an effect on the PLA and standard cell implementations. Specifically, the control table can be fed into a logic minimization program (such as espresso) to produce a PLA table or set of boolean equations with input and output planes that are somewhere between completely full and completely empty. The resulting implementations should, therefore, have physical capacitances near some "average" value between the two extremes. The effect of this random control table selection process is to remove most of the data-dependency from the physical capacitance of a controller. Instead, the physical capacitance will depend primarily on the complexity of the controller (in terms of number of states, inputs, and outputs) and not on the particular STG being implemented. This greatly simplifies the characterization process, reducing it to a difficulty closer to that of datapath or memory modeling.

The second phase of pattern generation entails producing input streams that exercise the controller over a full spectrum of activities. Unfortunately, if the output plane is completely random, there is no way to deterministically control the output activity from the inputs. Achieving the desired level of control over activity will require some modification of the output plane. In order to provide the best model fit over the widest range of parameters, the patterns should allow activities all the way from 0% to 100%. This can be provided by characterizing for three evenly spaced activity levels: 0, $\frac{1}{2}$, and 1. This in turn can be accomplished by correct sequencing of three basic patterns: `00...00`, `01...01`, and `11...11`.

In different situations, activity can be interpreted either as a signal probability, $P_{I/O}$ (i.e. the fraction of 1's in the data) or as a transition probability, $\alpha_{I/O}$ (i.e. the fraction of bits that switch between successive data sample). The three data patterns suggested here can be combined to produce activity levels of 0, $\frac{1}{2}$, and 1 regardless of the interpretation of "activity."

The input and output activity of an FSM can both be controlled by employing these three basic bit patterns. First, the input activity can be controlled by using the patterns as input "addresses" to the combinational logic block of the controller. Likewise, the output activity can be controlled independently by storing the same three data patterns at each of these three "addresses." Since, three data values cannot be stored at a single address, however, a couple of the address bits must deviate from the specified patterns and can be used to map the three data patterns to unique (but very similar) addresses.

This scenario is depicted in Table 4-6 for a controller with eight input "address" bits and eight output "data" bits. The left-most column provides a convenient label for each (address, data) activity pair. For example, $A(0, 0)$ corresponds to an all-zero address and an all-zero data field. Also, notice that the two least significant address bits are used to map the three different data values to unique locations while still approximately maintaining the desired address activity level. For example, this allows data values `00000000`, `01010101`, and `11111111` to all be mapped

Label	Address A[7:2]	Map A[1:0]	Data
$A(0,0)$	000000	00	00000000
$A(0,\frac{1}{2})$		01	01010101
$A(0,1)$		11	11111111
$A(\frac{1}{2},0)$	010101	00	00000000
$A(\frac{1}{2},\frac{1}{2})$		01	01010101
$A(\frac{1}{2},1)$		11	11111111
$A(1,0)$	111111	00	00000000
$A(1,\frac{1}{2})$		01	01010101
$A(1,1)$		11	11111111

Table 4-6 : "Address/data" patterns for generating desired controller activities

to addresses with mostly zero bits: $A(0, 0)$=00000000; $A(0, \frac{1}{2})$=00000001; and $A(0, 1)$=00000011, respectively. Likewise, these same data values can be mapped to addresses with activity $\frac{1}{2}$ using $A(\frac{1}{2}, \square)$ and activity 1 using $A(1, \square)$. Thus, while the output table is filled primarily with random data, it also contains nine deterministic values that allow precise control of input and output activity during characterization. These few non-random values should not adversely affect the physical capacitance of the controller, especially since the ratio of zeros to ones in the values is still 1:1.

Using this strategy, input and output activities can be controlled fairly independently. For instance, desired input and output *signal probabilities* can be chosen from nine possibilities, $(P_I, P_O) \in \{0, \frac{1}{2}, 1\} \times \{0, \frac{1}{2}, 1\}$, simply by accessing address $A(P_I, P_O)$. It is also possible to generate any of nine desired *transition* activities of the form $(\alpha_I, \alpha_O) \in \{0, \frac{1}{2}, 1\} \times \{0, \frac{1}{2}, 1\}$ by accessing the correct *sequence* of addresses. In particular, if the address sequence $A(\alpha_I^0, \alpha_O^0) \rightarrow$

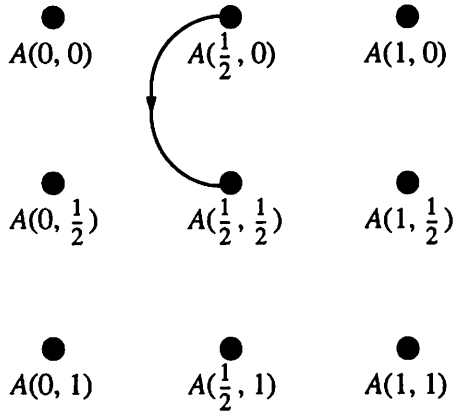


Figure 4-38 : Graphical representation of controller I/O activity patterns

$A(\alpha_I^1, \alpha_O^1)$ satisfies $\alpha_I = |\alpha_I^0 - \alpha_I^1|$ and $\alpha_O = |\alpha_O^0 - \alpha_O^1|$, then the approximate desired input and output activities (α_I, α_O) will be realized. For example, $A(\frac{1}{2}, 0) \rightarrow A(\frac{1}{2}, \frac{1}{2})$ yields a transition activity of *approximately* $(\alpha_I, \alpha_O) = (0, \frac{1}{2})$. Using the data of Table 4-6, the input address would go from `01010100` to `01010101` (close to zero switching activity) and the output data would go from `00000000` to `01010101` ($\frac{1}{2}$ activity).

Desired address sequences can be represented graphically by associating the ordered activity pairs with a coordinate system as shown in Figure 4-38 for the transition $A(\frac{1}{2}, 0) \rightarrow A(\frac{1}{2}, \frac{1}{2})$. This graphical representation will be used to describe pattern generation for the three examples of controllers modeled in the previous section: ROM, PLA, and standard cell.

Pattern Generation: ROM-Based Controller

The only activity parameter present in the ROM capacitance model of (EQ 86) on page 173 was the output signal probability P_O. Therefore, the ROM must be characterized for only three activity parameter values: $P_O \in \{0, \frac{1}{2}, 1\}$. Any input transition that terminates at address A(□, P_O) will satisfy this criterion. Figure 4-39a-c show the input transitions that characterize $P_O = 0$,

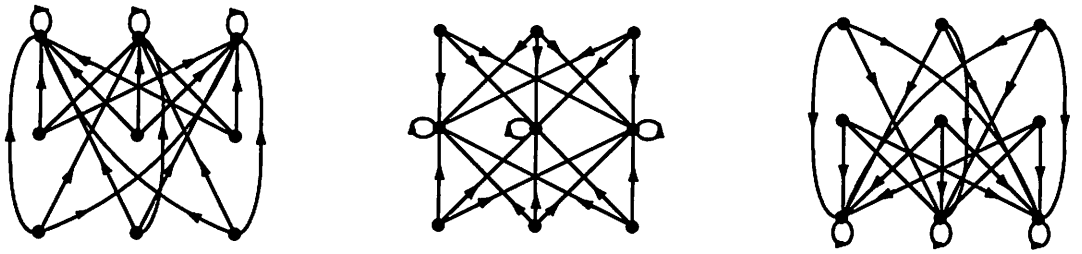


Figure 4-39 : Address sequences for generating desired ROM activities

$\frac{1}{2}$, and 1, respectively. In each figure, the simulated capacitance observations for each transition shown would be averaged together to produce the overall switching capacitance for that activity value. In this case, 27 initial observations must be averaged together to produce each of the three final observations.

Pattern Generation: PLA-Based Controller

The PLA modeled by (EQ 87) on page 177 has both an input activity and an output activity parameter. The input activity is a transition activity, α_I, while the output activity is a signal probability, P_O. Mixing of signal and transition probabilities is not a problem. Figure 4-40a-c show the input patterns for $(\alpha_I, P_O) \in \{ (0, 0); \left(0, \frac{1}{2}\right); (0, 1) \}$. As the figure demonstrates, these cases are characterized by inputs that remain fixed at addresses having the desired output probabilities. In this case, the aggregate capacitance observation for each activity pair is taken as the average of three individual observations.

Figure 4-40d-f show the pattern sequences for an input transition activity of $\frac{1}{2}$: i.e. $(\alpha_I, P_O) \in \{\left(\frac{1}{2}, 0\right); \left(\frac{1}{2}, \frac{1}{2}\right); \left(\frac{1}{2}, 1\right)\}$. Transitions with an input activity of $\frac{1}{2}$ are any that traverse between adjacent columns. As before, all transitions that terminate on nodes corresponding to a particular output probability are averaged together. For example, the four transitions in Figure 4-

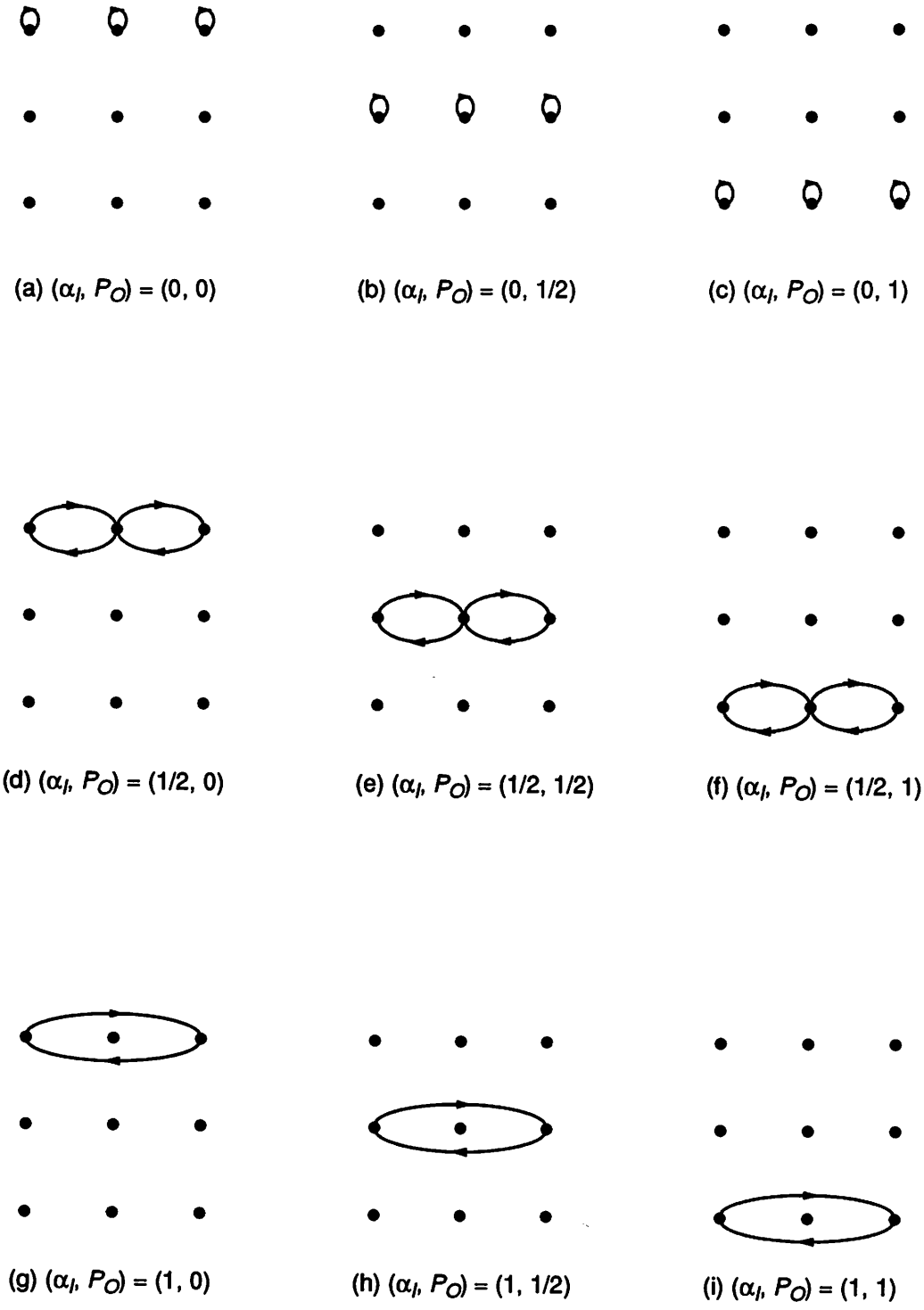


Figure 4-40 : Address sequences for generating desired PLA activities

40d are averaged together to produce the $(\alpha_I, P_O) = (\frac{1}{2}, 0)$ capacitance observation. Similarly, the transitions of Figure 4-40e-f contribute to the $(\frac{1}{2}, \frac{1}{2})$ and $(\frac{1}{2}, 1)$ observations, respectively.

Finally, Figure 4-40g-i depict the six input address transitions required to produce the $(\alpha_I, P_O) \in \{(1, 0); \left(1, \frac{1}{2}\right); (1, 1)\}$ capacitance observations. These transitions all skip two columns instead of one since an input activity of 1 and not $\frac{1}{2}$ is desired.

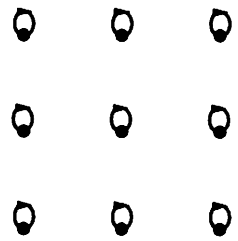
Taken all together, 27 input transitions are required to produce the 9 capacitance observations that fully characterize the PLA-based controller for different activity levels.

Pattern Generation: Random Logic Controllers

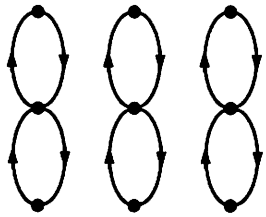
The capacitance model for the random logic, standard cell controller also has two activity parameters. For this controller, however, both parameters are transition activities: (α_I, α_O). The input sequences that characterize the standard cell controller for all activity permutations are shown in Figure 4-41.

Simulation

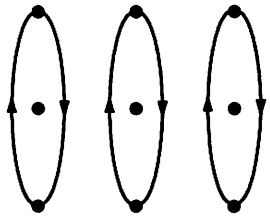
The simulation phase is identical to that used for characterizing datapath and memory modules. A module corresponding to a given set of complexity parameters (e.g. number of inputs, min-terms, and outputs) is synthesized and simulated for the data patterns generated by the aforementioned procedure. The simulation can be performed with either a circuit- or gate-level tool depending on the time the designer wishes to allow for characterization and the accuracy desired. The output of the simulation process is a number of capacitance observations corresponding to different input and output activity levels. A set of these observations is produced for each controller class and complexity being characterized. From these observations capacitive coefficients are extracted.



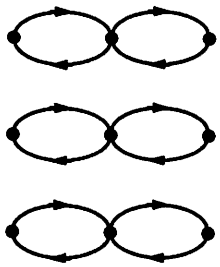
(a) $(\alpha_I, \alpha_O) = (0, 0)$



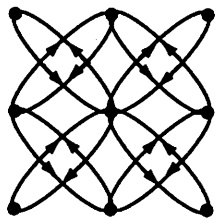
(b) $(\alpha_I, \alpha_O) = (0, 1/2)$



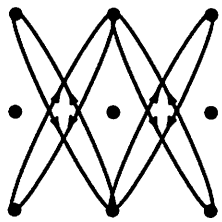
(c) $(\alpha_I, \alpha_O) = (0, 1)$



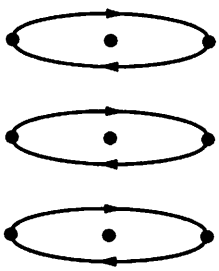
(d) $(\alpha_I, \alpha_O) = (1/2, 0)$



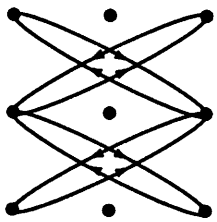
(e) $(\alpha_I, \alpha_O) = (1/2, 1/2)$



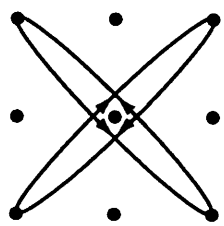
(f) $(\alpha_I, \alpha_O) = (1/2, 1)$



(g) $(\alpha_I, \alpha_O) = (1, 0)$



(h) $(\alpha_I, \alpha_O) = (1, 1/2)$



(i) $(\alpha_I, \alpha_O) = (1, 1)$

Figure 4-41 : Address sequences for generating desired random logic activities

Coefficient Extraction

Simulation produces effective switching capacitances for a sequence of characteristic input and output transitions. Coefficient extraction refers to the process of deriving best-fit model coefficients from the raw simulation data. This can be achieved using techniques such as least-squares regression, which minimizes the mean-squared error of the model. As for the datapath modules, this amounts to solving the following matrix equation for the capacitive coefficient vector, $\mathbf{C}_{eff}$:

$$\mathbf{C}_{sim} = \mathbf{P}\mathbf{C}_{eff} + \mathbf{e} \qquad \text{(EQ 89)}$$

where $\mathbf{C}_{sim}$ is a vector of simulated capacitance observations, $\mathbf{P}$ is a matrix of parameter values corresponding to the observations, and $\mathbf{e}$ is the modeling error.

The primary difference is that the controller parameter matrix $\mathbf{P}$ contains activity, as well as complexity parameter values. For example, assume the ROM-based controller model of (EQ 86) was characterized for complexity parameters, $N_I \in \{6,7,8\}$ and $N_O \in \{8,16,32\}$, and activity parameter, $P_O \in \{0, \frac{1}{2}, 1\}$. Then the parameter matrix would be given by:

$$\mathbf{P} = \begin{bmatrix} 1 & N_I 2^{N_I} & P_O N_O 2^{N_I} & P_O N_O & N_O \\ & & \cdot & & \\ & & \cdot & & \\ & & \cdot & & \end{bmatrix} \qquad \text{(EQ 90)}$$

with one row for each $(N_I, N_O, P_O) \in \{6,7,8\} \times \{8,16,32\} \times \{0, \frac{1}{2}, 1\}$. Similar constructions would be performed for PLA, standard cell, or other controller classes being characterized.

The result of solving the regressions would be capacitive coefficient vectors for each of the controller classes being characterized. These coefficients can then be used to evaluate effective controller capacitances during power analysis.

4.3.4 Power Analysis Method

The capacitive coefficients produced by library characterization can now be used to analyze the power consumed by controller modules. This analysis requires several inputs. The first input needed is a pointer to the hardware database entry corresponding to the target controller implementation (e.g. ROM-based, PLA-based, etc.). This allows the analysis tool to access the appropriate capacitance model and coefficients. Other necessary inputs include the complexity and activity parameters used in the capacitance models. These can be derived from the structural and behavioral descriptions of the design being analyzed (as will be discussed in Section 4.5).

Given these inputs, the appropriate target-specific capacitance model can be evaluated:

$$C_T^{CL} = \mathbf{C}_{eff}^{CL} \cdot \mathbf{N} \qquad \text{(EQ 91)}$$

This will produce an estimate of the average capacitance switched during a single evaluation of the combinational logic portion of the controller.

Aside from the combinational logic, the state register also contributes to the overall power consumption of the controller. Since N_S state bits must be stored, the capacitance model for the state register will have the following basic form:

$$C_T^{reg} = \alpha_S C_0 N_S \qquad \text{(EQ 92)}$$

where α_S is the activity of the state bits.

The total capacitance switched in the controller module is the sum of the combinational logic capacitance and the state register capacitance:

$$C_T = C_T^{CL} + C_T^{reg} \qquad \text{(EQ 93)}$$

If many accesses are involved, the total capacitance switched over a number of input control transitions, N_{CT}, can be computed using the following expression:

$$C_T|_{\text{multi-cycle}} = N_{CT} \cdot C_T|_{\text{single-cycle}} \quad \text{(EQ 94)}$$

4.3.5 Results

The preceding discussion presented a general strategy for estimating controller power consumption at the architecture level. Three implementation styles were used as examples, and results were presented showing that the corresponding models correctly track the influence of complexity and activity on power consumption. In order to automatically synthesize and test a large number of controllers, random control table contents were used in gathering the results. In this section we present data from a more realistic example to demonstrate that the models still behave well for real controllers.

The example we will use is a global controller for a low-power speech recognition front-end currently under development by S. Stoiber of U. C. Berkeley. The state machine contains over 100 states, 10 inputs, and 25 outputs. The majority of the control table entries are redundant, making the FSM a good test of how well the estimation models can handle non-random control table contents. We will return to discuss this controller example (as well as others) more fully in Chapter 6. For now we merely quote results from that discussion.

ROM, PLA, and standard cell implementations of the controller were synthesized in a 1.2 μm technology down to the layout level. The layouts were extracted and switch-level simulations of power consumption were compared to RT-level ABC estimates as shown in Table 4-7. All predictions were within 30% of the actual power consumption. Moreover, the models properly ranked the power efficiency of the different implementation styles. This suggests that the ABC model behaves reasonably for realistic controllers and that architecture-level estimates can be used to make ATP trade-offs very early in the controller design process.

Implementation	Average power (μW)		
	Actual	Estimated	Error
ROM	702	677	-3.6%
PLA	249	236	-5.2%
Standard cell	79	102	+29%

Table 4-7 : Summary of results for speech recognition controller

4.3.6 Control Path Summary

Previous attempts at architectural controller power modeling have had several weaknesses. Those based on statistical modeling of benchmark examples are fine for a narrow range of applications and controller architectures that fit in with the synthesized examples, but they do not provide a vision as to how the techniques can be extended to a more general setting. Other models, such as those based on gate equivalents (see Section 3.3.1 in Chapter 3), are more generally applicable but less accurate in that they take into account complexity but not activity. The Activity-Based Control (ABC) model presented here explicitly accounts for activity and provides a general framework for modeling different controller structures. Three specific examples - based on ROM's, PLA's, and random logic - were presented in this section, but the same general techniques could be used to model other implementation styles. Analyzing controller power consumption under the ABC model is very simple. It amounts to plugging the appropriate activity and complexity parameters into an equation that weights these parameters by technology-dependent capacitive coefficients derived during a one-time precharacterization process. The result is an accurate architecture-level power estimate that reflects both the physical capacitance and the circuit activity of the controller being analyzed.

4.4 Interconnect

The final class of component in a typical chip is interconnect. In this context, interconnect refers to any wiring used to distribute signals (including clocks) from one part of the design to another. This wiring can be either local or global. An example of local interconnect would be the interconnect from an adder to a shifter within a datapath block. Global interconnect includes clock distribution networks, global control signals, and data buses. This section addresses the problem of estimating the power consumed in charging and discharging the capacitance associated with interconnect wiring.

In general, the task of power estimation for all classes discussed thus far has decomposed into estimating a physical capacitance and combining it in some manner with an estimated measure of switching activity. The same approach applies to the interconnect class.

4.4.1 Interconnect Activity

First, consider the activity component of interconnect power consumption. The activity of a wire depends on the type of signal that wire carries: data or control. The activity of a data signal can be described by the DBT model of that data stream. Using this model, the total capacitance switched during a series of N_{DT} data transitions (i.e. bus accesses) on a bus driven by static logic is given by:

$$\text{static: } C_{DBT} = N_{DT}\left[\frac{1}{4}C_w N_U + \mathrm{P}(+-)\, C_w N_S\right] \qquad \textbf{(EQ 95)}$$

where C_w is the physical capacitance of the wires, N_U is the number of UWN bits in the data model, N_S is the number of sign bits, and P(+-) is the probability of a positive to negative transition between two successive samples in the data stream.

For a control wire driven by static logic, the appropriate activity is given by the ABC parameter α, the fraction of bits in the control signal that transition. The total control bus

capacitance switched over a series of N_{CT} transitions in the value of the control word is:

$$\text{static: } C_{ABC} = N_{CT}\left[\frac{1}{2}\alpha C_w N\right] \quad \text{(EQ 96)}$$

where N is the number of bits required to represent all values that the control word can assume.

Equations (EQ 95) and (EQ 96) apply to static buses. Some buses, however, use precharged logic. In other words, each clock cycle they are precharged to V_{dd} and then the 0 bits discharge during an evaluation phase. For these precharged buses, the appropriate DBT effective capacitance equation is:

$$\text{dynamic: } C_{DBT} = N_{clk}\left[\frac{1}{2}C_w N_{\mathrm{U}} + \mathrm{P}(+)\, C_w N_{\mathrm{S}}\right] \quad \text{(EQ 97)}$$

where N_{clk} is the number of clock cycles being evaluated and P(+) is the probability that a data sample is positive. Similarly, the ABC capacitance model becomes:

$$\text{dynamic: } C_{ABC} = N_{clk}\left[(1-P)\, C_w N\right] \quad \text{(EQ 98)}$$

where P is the ABC signal probability parameter.

All of these interconnect capacitance models depend on the availability of DBT and ABC activity parameters. Section 4.5 will describe how these parameters can be derived for a given chip. The capacitance models also rely on an estimate of the average physical capacitance of each wire in the design. Techniques for estimating this capacitance will be discussed in the following sections.

4.4.2 Physical Capacitance

Ideally, the physical capacitance of the interconnect network is known. If this is the case, it can be back-annotated onto the architectural description and used directly to compute the interconnect power. In most cases, however, the goal is to estimate the power of an architecture before the design has been mapped to layout; therefore, it is unlikely that the physical capacitance of the

wires would be know a priori. Some way of estimating the capacitance based on known architectural parameters is required.

From Chapter 2, we know that the main factors determining the physical capacitance associated with a wire are its width, its length, and the thickness of oxide separating it from the substrate. The oxide thickness, in turn, depends on the layer of metal used to route the interconnect. For certain signals, it is known before layout what layers will be used for routing. For example, in a three-level metal process, the designer might desire to route the clock primarily in level three metal to reduce clock capacitance. In such cases, the oxide thickness used for estimation can be chosen in accordance with these constraints. Typically, however, it is not known before layout which layers will be used to route which signals. In these cases, it makes sense to use some sort of average oxide thickness estimate. For instance, in a two-level metal process it is probably reasonable to assume that signals will be routed half in metal one and half in metal two. Once the average oxide thickness is decided upon, the area and perimeter capacitance parameters for an average wire can be determined as described in Chapter 2.

Given these parameters, it remains to determine the expected width and length of the interconnect wires. For width, estimates can again be made based on the design rules of the target technology and any agreed upon sizing techniques. For example, wires in the clock network, which can experience quite large transient currents, will often be larger than minimum width to avoid resistive drops and susceptibility to electromigration failure. In most cases, however, wires will be sized to the minimum allowed by the design rules in order to reduce wiring capacitance. Therefore, a minimum width approximation can often be used for signals that have no default width specification.

The last factor determining the capacitance associated with routing a signal is the length of the wire used. Wire length is an even more important factor than width since length contributes not only to parallel-plate capacitance, but also to fringe capacitance. Unfortunately, the wire length is

more difficult to estimate accurately at the architecture level than the width or oxide thickness. Whereas, reasonable estimates can be made for width and oxide thickness based purely on design rules and accepted practices, wire length depends more intimately on design-specific layout considerations. The next section describes previous work in wire length estimation and the two sections which follow describe our strategy for estimating local and global wire lengths.

4.4.3 Previous Work

Average wire length estimation has received a good deal of attention in the past. The primary works are those of Feuer [Feu82] and Donath [Don79], although Sorkin [Sor87] also provides some interesting insight into the problem. Feuer and Donath both base their derivations on Rent's Rule [Lan71], which relates block count in a region to the number of external connections to the region. In both cases, the derivations result in expressions relating average interconnect length for a square array placement to some power of the block count within a region. Converting from normalized length units to physical units gives rise to expressions for average lengths which are proportional to a power ($p<1$) of the region area. Sorkin confirms this general rule noting that empirical data support average lengths proportional to the square root of the chip area (i.e. $p=1/2$). Donath also introduces the notion of hierarchical design partitioning and placement. Our estimation techniques rely heavily on this notion, as well as the empirical observations of Sorkin resulting in an intuitive, yet fairly accurate, strategy for interconnect analysis.

4.4.4 Hierarchical Interconnect Analysis

In the following sections we describe our approach to wire length estimation. The input to this process is the structural RTL description of the chip being analyzed, which is provided by the designer. The description can be hierarchical as shown in Figure 4-42. The current level of hierarchy being analyzed is referred to as the *complex* and consists of a number of *blocks* joined by an interconnect network. These component blocks can be of four basic types. A *composite* block is

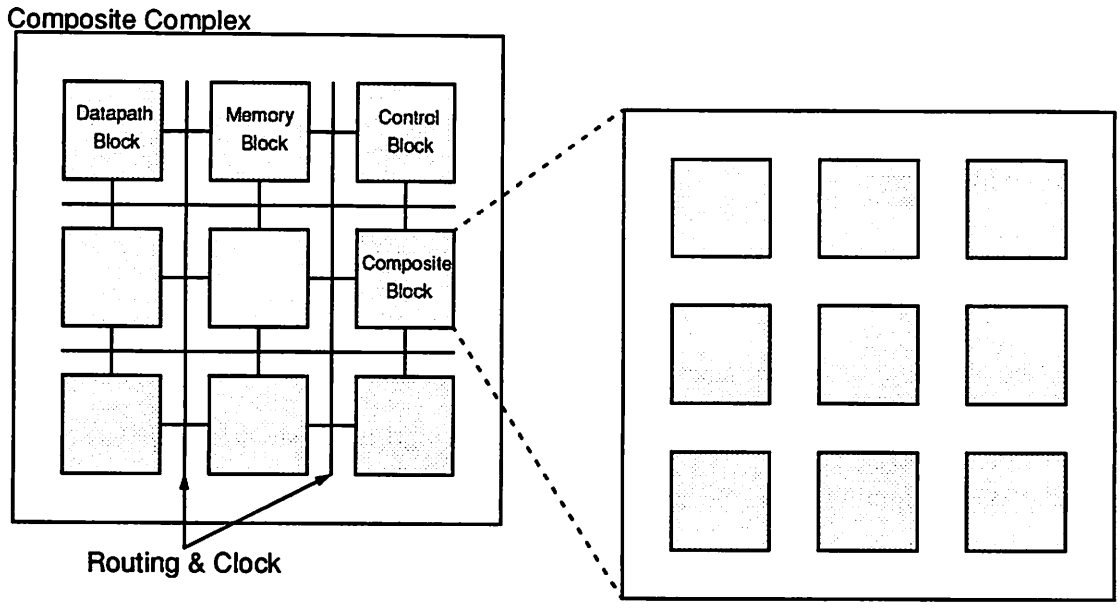


Figure 4-42 : Hierarchical structure of chip

used to introduce further hierarchy into the design and consists of a collection of sub-blocks with a structure similar to that of the complex. Although each composite block in the hierarchy may have the same structural composition as the complex, the term *complex* is used only when referring to the level of hierarchy currently being analyzed. The other three block types are named for the type of structures within the block: datapath, memory, and control.

The wire length estimation process begins with the composite complex at the top level of hierarchy. In order to analyze the interconnect at this level, however, each of the blocks in the complex must first be analyzed. For composite blocks, this is achieved by recursively invoking the estimation process with the composite block as the new complex. Datapath, memory, and control blocks are handled by dedicated analysis strategies. An advantage of this hierarchical approach is that it preserves valuable partitioning clues supplied by the designer. Analyzing a flat description results in a single average wire length that will be used for all nets in the design. In an actual implementation wire lengths vary, with shorter, local wires confined to blocks in the lower levels of hierarchy and longer, more global wires more frequent at the higher levels. The hierarchical

strategy employed here, therefore, allows more accurate estimates of interconnect length. Sections 4.4.5-4.4.8 describe the interconnect analysis strategies for each of the four possible block types: composite, datapath, memory, and control. Finally, Section 4.4.9 discusses how to analyze a clock distribution network.

4.4.5 Composite Blocks

The objective of interconnect analysis for a composite complex is to estimate the average length of global data and control wires. The actual length of these wires depends on the exact placement and routing of the blocks in the complex. Since we are interested in early estimates of wire length (prior to place and route), this information is not available. Instead, the task of architecture-level interconnect analysis is to predict what the average wire length is likely to be for a "good" placement of the complex. One way to achieve this end would be to perform a simplified floorplanning and placement step and to use the results of this process to approximate the wire length; however, if this process is deemed too expensive there are other reasonable alternatives.

The approach advocated here relies on the widely accepted empirical observation that the "quality" of a "good" placement often differs from a random placement by a constant factor, k [Don79][Sor87]. In this context "quality" is measured by average wire length. So a "good" placement is one in which blocks joined by many wires are kept close together. If the average wire length for random and "good" placements really differ by a constant factor, the task of wire length estimation is greatly simplified. Specifically, if we can ascertain the average wire length for a random placement, we can extrapolate the results for a "good" placement by applying a conversion factor. It is a well-known fact that the average wire length for a random placement on a square array (such as Figure 4-42) is 1/3 the side-length of the complex [Pre88]. This suggests that the average wire length for a "good" placement is:

$$L = k\frac{\sqrt{A}}{3} \qquad \text{(EQ 99)}$$

The exact value of the k factor will depend on the characteristics of the placement and routing tools being used; however, an oft-quoted conversion factor is approximately 3/5. In other words, a "good" placement is typically a little less than a factor of two better than a random placement. Since this value results in the formula $L = \sqrt{A}/5$, it has been called the "1/5 rule" in the literature [Sor87].

While (EQ 99) is straightforward and intuitive, it does require that the area of the complex be known. The area of a composite complex is the sum the areas of the component blocks and the area occupied by routing:

$$A = A_w + A_B \qquad \text{(EQ 100)}$$

$$A_B = \sum_{i \in \{\text{Blocks}\}} A_i \qquad \text{(EQ 101)}$$

The topic of area estimation for datapath, memory, and control blocks will be covered in Sections 4.4.6-4.4.8.

Composite blocks are handled recursively using formulas that will now be derived. The component of area related to routing depends on the total number of wires in the complex (N_w), the average pitch at which they are routed (W_p), and their average length (L):

$$A_w = N_w W_p L \qquad \text{(EQ 102)}$$

Taken simultaneously, (EQ 99)-(EQ 102) result in a quadratic expression that can be solved to yield:

$$L = \frac{k^2 N_w W_p + \sqrt{(k^2 N_w W_p)^2 + 36 k^2 A_B}}{18} \qquad \text{(EQ 103)}$$

This result can then be substituted back into (EQ 102) and (EQ 100), if desired, to arrive at the total area for the complex.

In summary, the above derivation led to expressions which can be used to estimate the area and average interconnect length of a composite complex using a hierarchical analysis strategy.

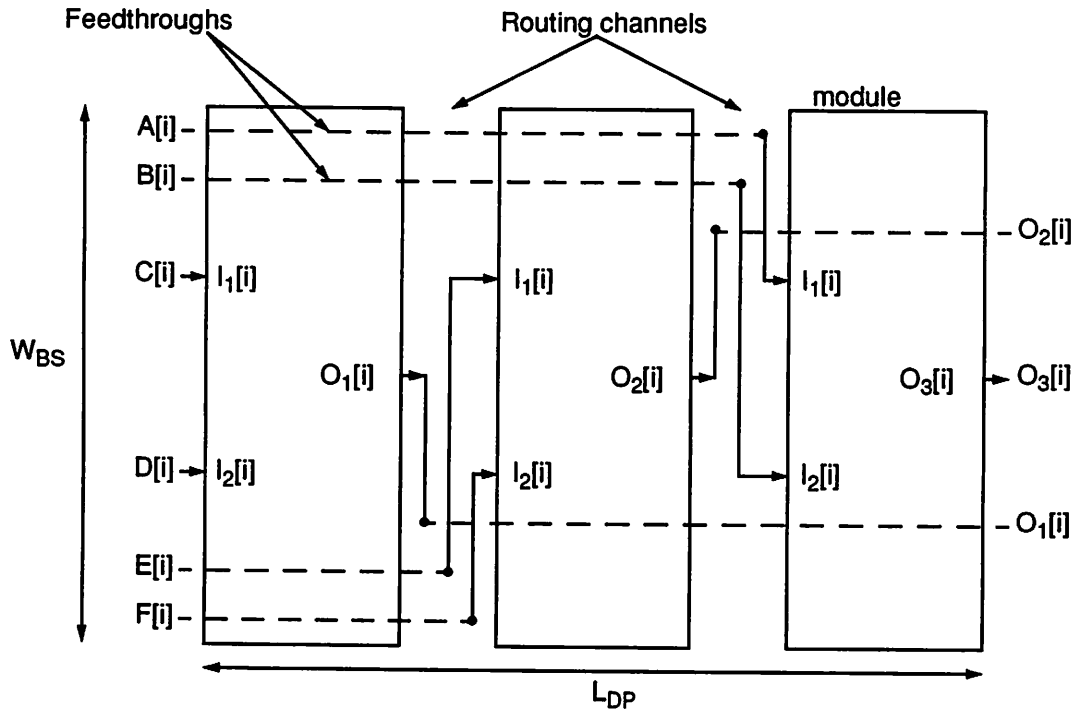


Figure 4-43 : Placement strategy for a representative bit-slice of a datapath complex

4.4.6 Datapath Blocks

The previous section considered area and interconnect estimation for composite blocks. In this section, we discuss how these tasks can be performed for datapath blocks. Datapath blocks contain computational modules such as adders, comparators, and shifters. As in a composite block, a datapath block may contain several of these elements joined together by some interconnect network. The primary difference between the two types of blocks is the placement and routing strategy. Since data often flows through a datapath in a fairly linear fashion, datapath blocks often employ a one-dimensional placement. Figure 4-43 shows a representative placement for a single bit-slice of a datapath block. In the figure, L_{DP} denotes the length of the datapath block and W_{BS} represents the width of the bit-slice. For an N-bit datapath, N of these bit-slices would be tiled together vertically to yield a total datapath width of $W_{DP}=NW_{BS}$.

From the figure we see that the length of a local interconnect within the datapath is composed

of a horizontal component and a vertical component:

$$L = L_x + L_y \quad \text{(EQ 104)}$$

Consider first the length of the horizontal component. As with the composite complex, the average length of a local interconnect within a datapath complex depends on the precise placement of the two modules the wire connects. Once again, we can use a k factor (specific to the datapath tiler) to allow us to convert from an average length for a random placement to one for a "good" placement. Considering first the horizontal component of the interconnect length, we have $L_x = L_{DP}/3$ for a random placement and:

$$L_x = k\frac{L_{DP}}{3} \quad \text{(EQ 105)}$$

for a "good" placement.

The length of the datapath, L_{DP}, in turn is given by the sum of the length of the modules within the complex and the length of the routing channels separating the modules:

$$L_{DP} = L_R + \sum_{i \in \{\text{Blocks}\}} L_i \quad \text{(EQ 106)}$$

In this equation the length of the modules within the datapath complex is known and can be read from a hardware database. The length of the routing channels is not deterministic and must be estimated.

The total length of the routing channels will depend on the number of vertical tracks required, as well as the pitch, W_p, at which the wiring will be placed. Since both the inputs and the outputs of the modules may have to be routed in the channels, the number of tracks required will be close to the total number of I/O terminals on all the modules. This may also be read from the hardware database:

$$L_R = W_p \sum_{i \in \{\text{Modules}\}} N_{IO_i} \quad \text{(EQ 107)}$$

Combining (EQ 105)-(EQ 107) gives us the average horizontal component of interconnect length in the datapath complex.

There is also a vertical component to local interconnect in the datapath. In order to connect the terminals of two modules in the datapath, the wire must first be routed vertically from the terminals to the selected feedthrough. The length of the vertical wiring component is given by:

$$L_y = 2k\frac{W_{BS}}{3} \qquad \text{(EQ 108)}$$

where the factor of 2 reflects the fact that vertical routing segments are required at each end of the wire, and W_{BS} is the width of a datapath bit-slice, which can be ascertained from the hardware database. Combining L_x and L_y gives us our average interconnect length, L.

Aside from average interconnect length, the other piece of information desired from datapath analysis is the area, A, of the complex. This information will be used when estimating the area and average interconnect length of the composite complex at the next level up in the hierarchy. The area of the datapath can be approximated as:

$$A = W_{DP}L_{DP} = NW_{BS}L_{DP} \qquad \text{(EQ 109)}$$

To review, this section has presented an interconnect analysis strategy for a linear placement of a datapath complex. The primary results of the analysis are an estimate of the average length of a local interconnect within the datapath and an estimate of the area of the datapath block. The calculations were based on raw area data for the primitive modules (e.g. adders, shifters, etc.), which is stored in a hardware database. A few examples of these primitive datapath area models will now be described. Note that all dimensions are based on hardware units taken from a 1.2 μm CMOS library. The same characteristic parameters, however, would apply to any library. Only the parameter values would differ.

Subtracter

Unlike power, the area of primitive datapath modules is a deterministic function of their complexity parameters. Thus, the area data can be entered into the hardware database from direct measurement of layout dimensions. If no layout exists, estimates from past designs or based on logic synthesis results can be used instead.

The width and length of a single bit-slice of the 1.2 μm subtracter used in this chapter are:

$$W_{BS} = 38.4\mu m \quad \text{(EQ 110)}$$

$$L = 124.2\mu m \quad \text{(EQ 111)}$$

Therefore, the full N-bit subtracter would have dimensions:

$$W = NW_{BS} \quad \text{(EQ 112)}$$

$$L = 124.2\mu m \quad \text{(EQ 113)}$$

Logarithmic Shifter

The logarithmic shifter has dimensions based on two parameters: word length, N, and number of stages, $\lceil \log_2(M+1) \rceil$, where M is the maximum shift value possible. The dimensions of the 1.2 μm shifter are:

$$W_{BS} = 38.4\mu m \quad \text{(EQ 114)}$$

$$W = NW_{BS} \quad \text{(EQ 115)}$$

$$L = (66\lceil \log_2(M+1) \rceil)\mu m \quad \text{(EQ 116)}$$

Register File

Often, foreground memory such as registers and register files are designed as bit-sliced modules and, therefore, may be included with other modules within a datapath block. For this type of module, storage capacity, N_W, and word length, N, are the complexity parameters that determine its dimensions:

$$W_{BS} = 38.4\mu m \quad \text{(EQ 117)}$$

$$W = NW_{BS} \quad \text{(EQ 118)}$$

$$L = (37.2\mu m)N_W \quad \text{(EQ 119)}$$

Array Multiplier

Unlike the previous modules, the array multiplier is typically not designed as a bit-sliced unit, but rather as an array block (as the name implies). When this is the case, the multiplier will not be tiled with other datapath modules in a datapath block, but instead will be placed individually as an independent block in some composite complex. Therefore, the required area parameters are just the total width and length of the block as a function of the input word lengths, N_1 and N_2:

$$W = (77.4\mu m)N_1 \quad \text{(EQ 120)}$$

$$L = (81.0\mu m)N_2 \quad \text{(EQ 121)}$$

4.4.7 Memory Blocks

Background memories such as RAM's are similar to array multipliers in that they are stand-alone blocks that are placed within a higher composite complex. Since the power models for memories already account for internal wiring, interconnect analysis is not required, per se, for these modules. All that is required is to provide area estimates that can be used by the parent composite complex when making its own interconnect and area estimates. The complexity parameters for the RAM are the same as for the register file - N and N_W. The area model for the 1.2 μm SRAM used in Section 4.2.3 is:

$$W = (165 + 30.6N)\,\mu m \quad \text{(EQ 122)}$$

$$L = (217.2 + 10.2N_W)\,\mu m \quad \text{(EQ 123)}$$

4.4.8 Control Blocks

The final type of block that may occur in a composite complex is a control block. The power model for these blocks was described in Section 4.3 where it was recognized that the target implementation type of the controller (e.g. ROM, PLA, random logic) had a large impact on the

power consumption. The same is true for the area of the controller. As was the case for the memory blocks, it is not necessary to perform an interconnect analysis on a controller block since internal wiring is already included in the power models when these blocks are characterized. So, once again, all that is required are area models for the three basic classes of controllers.

As with the power models, these area formulas will all depend on the same basic complexity parameters: the number of inputs to the combinational logic block of the controller (N_I), the number of outputs (N_O), and the number of min-terms (N_M). Since activity does not come into play, the area models for controller blocks will, in general, be more straightforward than the power models.

ROM-Based Controllers

Referring back to Figure 4-31 on page 173, the width of the ROM (measured in the horizontal direction) is proportional to the number of input bits as well as the number of output bits. The length of the ROM (measured vertically) is proportional to the number of words stored in the unit, 2^{N_I}. Constant overhead terms representing control logic and timing circuitry are also included in both the width and length terms:

$$W = W_0 + W_1 N_I + W_2 N_O \quad \text{(EQ 124)}$$

$$L = L_0 + L_1 2^{N_I} \quad \text{(EQ 125)}$$

For the 1.2 µm ROM described in Section 4.3.2, the coefficients have the following values: (W_0, W_1, W_2) = (96.6, 9.6, 21.6) µm and (L_0, L_1) = (150, 1.2) µm.

PLA-Based Controllers

The form of the PLA expressions is identical to that of the ROM with one minor exception. Recall that PLA's, unlike ROM's, do not employ full address decoding. This means that the total number of rows in the PLA will be equal not to 2^{N_I}, but to the number of min-terms, N_M, in the minimized two-level logic expressions:

$$W = W_0 + W_1 N_I + W_2 N_O \quad \text{(EQ 126)}$$

$$L = L_0 + L_1 N_M \quad \text{(EQ 127)}$$

For the 1.2 μm PLA of Section 4.3.2, the coefficients have the following values: (W_0, W_1, W_2) = (69.9, 9.6, 5.8) μm and (L_0, L_1) = (94.3, 6.2) μm.

Random Logic Controllers

As with the power models, the random logic (or standard cell) implementations present the most difficult area estimation challenges. This is because neither the user-provided complexity information, nor the behavioral information (i.e. symbolic control table contents) allow us to determine what logic decomposition and technology mapping will be used to implement the FSM in random logic. Moreover, many standard cell place and route routines are non-deterministic, adding another degree of difficulty to the problem. The solution proposed here takes the same statistical modeling approach that was used to build up the ABC capacitance models for the controller. In other words, the user or library designer proposes an area model that describes how the standard cell implementation area should vary with changes in the complexity parameters. Then, during characterization, actual implementation areas are used to extract best-fit coefficients for the area model, just as is done for the capacitance model.

For example, a reasonable standard cell area model might be:

$$A = C_0 N_M + C_1 N_O + C_2 N_I N_M + C_3 N_M N_O + C_4 N_I N_M N_O \quad \text{(EQ 128)}$$

In this expression, there are terms for the number of min-terms, $C_0 N_M$, and the number of outputs, $C_1 N_O$, which give us a partial measure of wiring and logic complexity in the input and output planes. There are also terms that track growth in the total number of decomposed (e.g. 2-input) gates required in the two planes: $C_2 N_I N_M$ and $C_3 N_M N_O$. Finally, a third order term $C_4 N_I N_M N_O$ reflects the total wiring complexity of the standard cell block. The fit of the model for controllers synthesized from a 1.2 μm standard cell library is shown in Figure 4-44. The rms error for the data points in the figure is 8.1% and the maximum error is 16.2%. The results were gathered for random

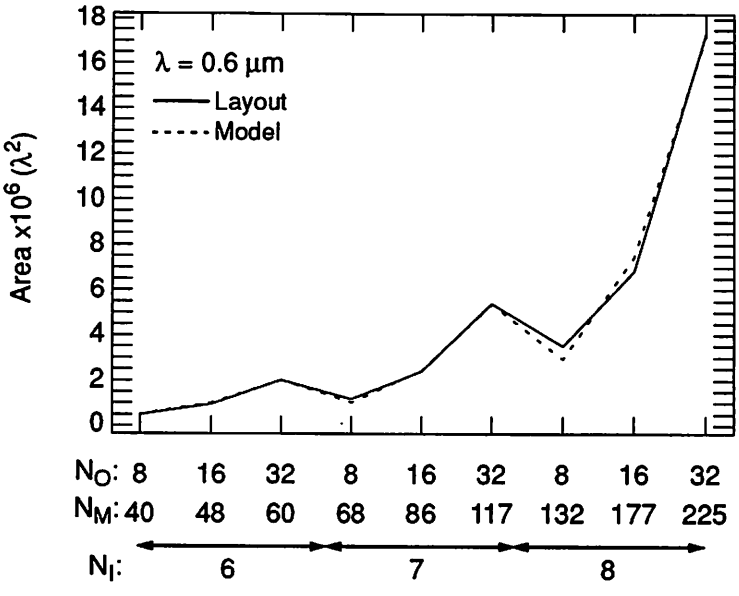


Figure 4-44 : Standard cell implementation area vs. model for various complexities

control table contents which were minimized with espresso and synthesized with MIS and Lager IV. Other authors have developed more detailed models for standard cell area estimation, but they generally require more information than is available at the architecture level (i.e. prior to state assignment, logic minimization/decomposition, and technology mapping) [Kur89].

4.4.9 Clock Distribution Network

Aside from data and control signals, interconnect is also used to distribute the clock signal to synchronous blocks in the chip. Switching the capacitance associated with the clock distribution network consumes power and, therefore, interconnect analysis must include an estimate of the total length of wire used to distribute the clock within each composite complex in the hierarchy. For this purpose, let us assume the clock distribution network shown in Figure 4-45.

In the figure, A is the total area of the composite complex and A_B is the area occupied by the N_B blocks in the complex. The formulas needed to estimate these areas for a composite complex were given in Section 4.4.5. Given these values, the figure suggests that the width of the routing

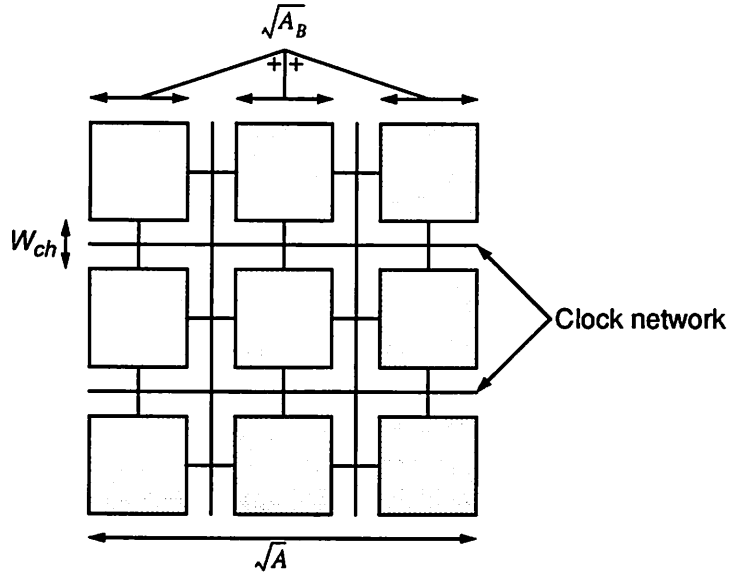


Figure 4-45 : Sample clock distribution network in composite complex

channels in the complex is given by:

$$W_{ch} = \frac{\sqrt{A} - \sqrt{A_B}}{\sqrt{N_B} - 1} \qquad \text{(EQ 129)}$$

This can be used to estimate the total length of wiring used in the clock distribution network:

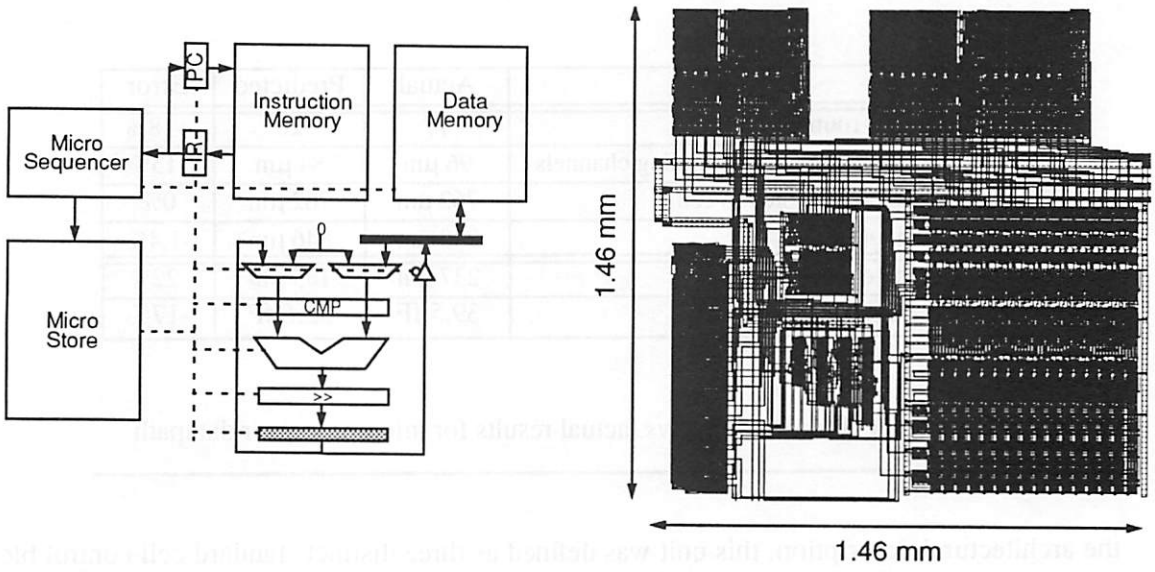
$$L_{clk} = 2k_{clk}(\sqrt{N_B} - 1)(\sqrt{A} + W_{ch}\sqrt{N_B}) \qquad \text{(EQ 130)}$$

where k_{clk} is an empirical scaling factor used to relate the distribution strategy assumed in the model to the actual strategy used by a particular clock routing tool.

4.4.10 Results

The preceding sections have presented various strategies for modeling area and interconnect at the architecture level. This section presents results verifying that these models provide reasonable results.

As a first example, we will consider the simple microprocessor depicted in Figure 4-46. The



(a) Architecture used for estimates (b) Implementation used for verifying results

Figure 4-46 : Microcoded instruction set processor

Block	Actual (mm^2)	Predicted (mm^2)	Error (%)
Instruction memory	0.16	0.16	0%
Instruction regs	0.05	0.04	-20%
Micro sequencer	0.08	0.14	+75%
Micro store	0.13	0.11	-15%
Data memory	0.25	0.25	0%
Datapath	0.57	0.52	-9%
Wiring/overhead	0.90	0.74	-18%
Total	2.14	1.96	-8%

Table 4-8 : Predicted vs. actual area for simple microprocessor chip

example will be discussed in more detail in Chapter 6; for now, we consider only the accuracy of the area and interconnect models used to analyze this example. Table 4-8 gives the area analysis for the design, showing that the estimated area is within 8% of the actual area. The estimated areas of the various blocks typically err by less than 20%. The micro sequencer is a notable exception. In

Item	Actual	Predicted	Error
Vertical routing tracks	17	20	+18%
DP length devoted to routing channels	96 μm	84 μm	-13%
DP length devoted to cells	762 μm	762 μm	0%
Total DP length	858 μm	846 μm	-1.4%
Average wire length	237 μm	185 μm	-22%
Average wire capacitance	39.5 fF	32.6 fF	-17%

Table 4-9 : Predicted vs. actual results for microprocessor datapath

the architectural description, this unit was defined as three distinct standard cell control blocks, while in the final implementation they were merged into a single block. This explains the overestimate and demonstrates the important point that architecture-level estimates are always subject to limitations imposed by an imperfect knowledge of the final implementation details.

Based on the area estimates, the average interconnect and clock length are estimated at 554 μm and 2174 μm. The actual lengths are 595 μm and 2796 μm, implying estimation errors of -7% and -22%. To demonstrate that the models are valid for other design examples, a hardware divider (also to be discussed in Chapter 6) was analyzed as well. The predicted area is 4.68 mm^2 versus an actual implementation area of 4.40 mm^2 - an error of 6%. The measured interconnect and clock wire lengths are 858 μm and 2124 μm, while the predicted lengths are 856 μm and 2861 μm for errors of -0.2% and 35%, respectively.

Since datapath blocks use a linear (rather than array) placement strategy, their interconnect models must be verified separately. To this end, Table 4-9 gives the modeling results for the microprocessor datapath. In this case, the average interconnect capacitance for the datapath is predicted to within 17% of the actual value.

In general, since the placement and routing strategies for bit-sliced datapaths are fairly well-

defined and regular, we expect the datapath interconnect models to be more consistently accurate than the composite models, which may be expected to work well in some cases but not as well in others. The important point demonstrated by these examples is not that the interconnect models are perfect, but that they behave reasonably for practical design examples.

4.4.11 Interconnect Summary

To review, this section has described the tasks involved in analyzing the interconnect and clock distribution networks of a chip given its RTL description. The primary objective of interconnect power analysis reduces to estimating the average length of interconnections in the design. The approach taken here was hierarchical in order to take advantage of partitioning clues provided by the designer. This makes wire length estimates more accurate, allowing a distinction to be made between local and global interconnects. At each level of hierarchy, the length of the wires depended largely on the area of the block being analyzed. Therefore, a large portion of this section was devoted to developing RT-level area estimation techniques for the different classes of components in a typical chip.

Once ascertained, the length of the wires in the chip can be used to estimate the power consumed in driving the interconnect and clock networks. This power depends not only on the physical capacitance (i.e. length) of the wires, but also on the activity of the signals. The activity appropriate to each wire can be determined by looking at the driving module. If it is a datapath or memory block, the DBT activity model can be applied, and if it is a control block, the ABC activity model can be used. While the interpretation of the DBT and ABC activity parameters has been discussed in previous sections, no means of deriving these parameters for an actual design has yet been presented. This topic is the primary focus of final section in this chapter.

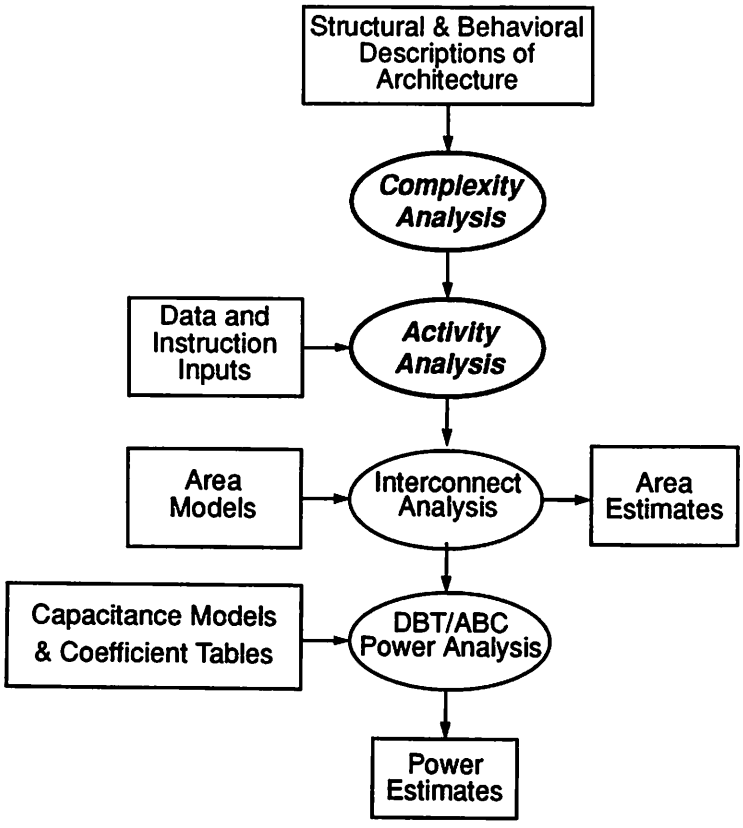


Figure 4-47 : Process flow for architectural power analysis

4.5 Chip-Level Power Analysis Method

The previous sections have presented models and methods for analyzing the power consumption of the four basic classes of components within a chip: datapath, memory, control, and interconnect. This section describes how these module-level techniques can be combined to achieve power analysis at the chip level. Figure 4-47 shows the sequence of steps involved in the power analysis process. First, the user must provide a register-transfer level descriptions of the chip architecture in terms of both its structure and its behavior. The structural description is simply a hierarchical RTL netlist that describes the entities used in the design and how they are interconnected (see Figure 4-48). The behavioral description tells how the individual modules and

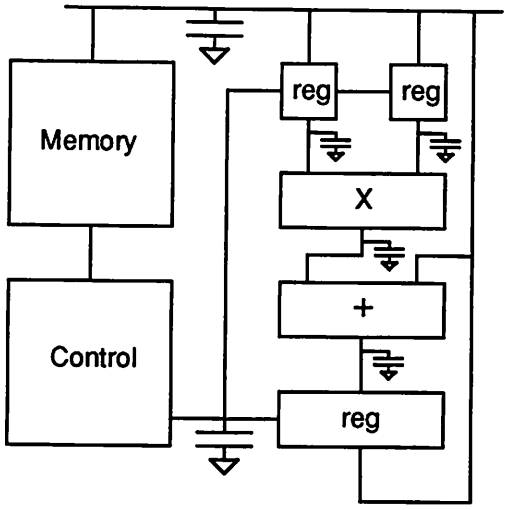


Figure 4-48 : RT-level structural description of chip

buses work together to achieve the desired functionality.

Given these architectural descriptions, the next step is to derive the complexity and activity parameters that are required by the DBT and ABC power analysis models. Once this is accomplished, interconnect analysis is performed using area models stored in the hardware database to produce hierarchical estimates of area and average wire length. Finally, the power of individual modules and buses is analyzed by combining the complexity and activity information with the appropriate capacitance models and coefficient tables (also stored in the hardware database).

The area and power analysis phases of this process have already been described in previous sections of this chapter. It only remains to describe how the complexity and activity parameters, which are the inputs to these analyses, can be derived. Section 4.5.1 will describe how to calculate the necessary complexity parameters, while Section 4.5.2 will focus on activity parameters.

4.5.1 Complexity Analysis

Intuitively, the complexity of an entity corresponds to its "size". Since the exact complexity parameters required and the method for calculating them differ depending on whether the entity falls under the DBT or ABC models, each type will be dealt with individually.

DBT Complexity Parameters

The DBT model recognizes two types of entities: modules and buses. The most important complexity parameter, shared by both of these entities, is their word length, N. For example, a data bus may consist of 32 bits (N=32) or an adder may contain 16 bit-slices (N=16). Individual modules may have additional complexity parameters. For instance, the complexity of a register file is described not only by the length of the words it contains, but also by the number of words stored. Similarly, a logarithmic shifter has a complexity parameter describing the number of stages it contains.

For each module, its associated set of complexity parameter values must be specified by the user in the structural description of the architecture. In other words, whenever the user instantiates a library element, not only must the nets to which it connects be enumerated, but also the "size" of the unit must be specified by assigning values to the complexity parameters associated with the module. The word length parameters for buses can then be ascertained by looking at the modules to which they connect.

So the analysis of DBT complexity parameters consists of stepping through the various DBT entities in the structural description of the architecture and reading off the values specified by the user in the argument list of each module.

ABC Complexity Parameters

Aside from the DBT parameters, we must also consider how to derive the necessary ABC parameters for the control buses and modules. Since determining the complexity of control buses is the most straightforward of the two, we will deal with it first.

As in the DBT case, the key complexity parameter for a control bus is the number of bits that it contains since this will, in part, determine how much physical capacitance gets switched during a control word transition. Recall that in order to maintain a high level of abstraction, all control signals are specified as enumerated types that assume symbolic, rather than binary, values. For example, if a control bus carries the function input to an ALU, the type of the bus might be defined as:

$$\text{ALU\_TYPE} \equiv \{\text{ADD, SUB, SHIFT, CMP}\} \quad \textbf{(EQ 131)}$$

The number of bits required to carry an enumerated data type such as this one is determined by the "size" of the enumerated types involved. For example, if an enumerated type, T, can take on $|T|$ different values then, the word length, N, (in bits) required to represent this signal in a binary encoding is:

$$N = \lceil \log_2 |T| \rceil \quad \textbf{(EQ 132)}$$

For the ALU example, the number of bits required for a control bus of ALU_TYPE is:

$$N = \lceil \log_2 |\text{ALU\_TYPE}| \rceil = \lceil \log_2 4 \rceil = 2 \quad \textbf{(EQ 133)}$$

This formula assumes a binary encoding of enumerated types. Not all control signals, however, are encoded in this fashion. For a "one-hot" encoding strategy, the appropriate complexity formula would simply be $N = |T|$. Other encoding schemes can also be envisioned. One way to handle these variations would be to have the user associate an *encoding style attribute* with each ABC data type or entity. Alternatively, the complexity analysis routines could simply assume a default encoding.

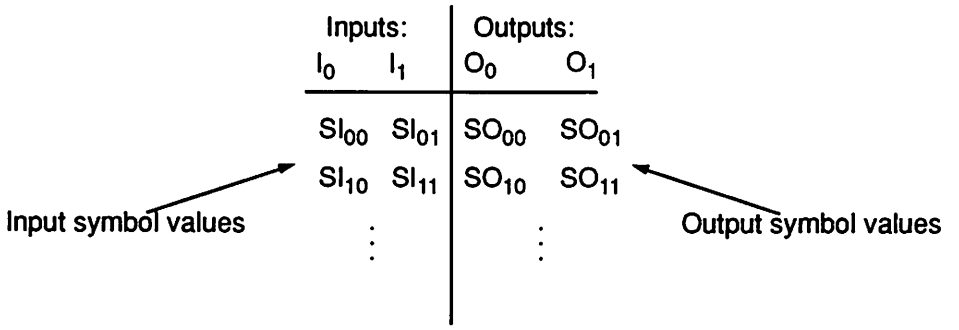


Figure 4-49 : Symbolic controller truth table

The complexity parameters for control modules can be calculated by applying a similar strategy. Recall that two important complexity parameters for these modules are the number of input bits (including state), N_I, and the number of output bits, N_O, associated with the combinational logic block of each controller. The total input and output widths can be determined by computing the width of each of the input and output fields individually and then summing them. In other words:

$$N_I = \sum_{i \in \{\text{Inputs}\}} N_{I_i} = \sum_{i \in \{\text{Inputs}\}} \lceil \log_2 |\text{INPUT\_TYPE}_i| \rceil \qquad \text{(EQ 134)}$$

$$N_O = \sum_{i \in \{\text{Outputs}\}} N_{O_i} = \sum_{i \in \{\text{Outputs}\}} \lceil \log_2 |\text{OUTPUT\_TYPE}_i| \rceil \qquad \text{(EQ 135)}$$

The final complexity parameter that is required for ABC combinational logic blocks is the number of min-terms, N_M, in the minimized control table. This parameter is more difficult to estimate since the amount of minimization that can be performed depends on the binary encoding of the control table, which is probably not available at this stage of design. It is more likely that the control table is specified in terms of symbolic values as shown in Figure 4-49. We can approximate N_M, however, by assuming some binary mapping (e.g. random), performing logic minimization, and using the number of min-terms in the minimized table. Tools for truth table minimization such as U. C. Berkeley's espresso are fast, efficient, and readily available [Bra84].

In summary, the complexity parameters for the DBT and ABC models can be determined from the structural and behavioral descriptions of the architecture in question. The most important parameter for both models is word length, and it applies to both bus and module entities. The ABC model also requires a min-terms parameter which describes the "size" of the control table after redundancies have been eliminated. Finally, the user is free to add complexity parameters to individual modules when appropriate as long as their values are specified when instantiating the module.

4.5.2 Activity Analysis

After determining the complexity of an architecture, the next step is to analyze the switching activity it undergoes. This activity depends on the behavior of the architecture, as well as its structure, so these two *views*, which the user must provide, are among the inputs to the activity analysis process. In general, the activity will also depend on the input vectors (data and instruction) being applied to the design, so a typical input stream must also be supplied by the designer.

We can imagine two strategies for deriving the necessary activity statistics depending on how closely tied the behavioral and structural descriptions are. In some cases, the behavioral and structural views of a design may be fairly independent. For instance, when dealing with DSP applications it is quite common to represent the behavior of an algorithm by a signal flowgraph as shown in Figure 4-50, which specifies the desired functionality independent of the structural implementation. In such a case, it might be beneficial to derive the signal activity from this data structure. This algorithmic activity could then be translated to an architectural activity by a set of *bindings* - i.e. a mapping between the behavioral and structural representations. For a one-to-one mapping of operations to hardware units the two activity measures would be identical; however, in the case of time-multiplexed resources, the activity seen at the inputs to shared modules would be some combination of the flowgraph signal activities. The use of dual representations in the context

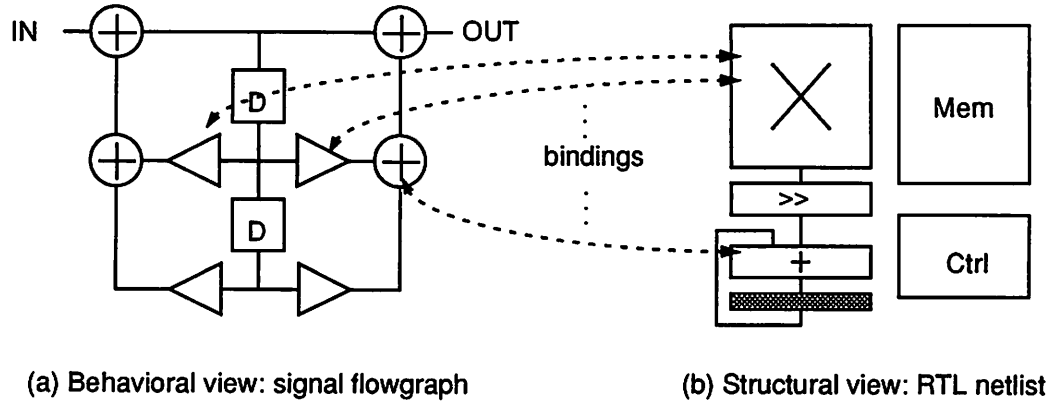


Figure 4-50 : Independent structural and behavioral representations

of DSP applications will be discussed further in Chapter 5.

In other contexts, it might be more useful to employ a more tightly coupled and unified representation for the architecture. In this case it is not necessary to derive the activity statistics from the behavioral description first, and then map them to the structure. Instead, the required activity parameters can be derived from a direct simulation of the architecture as a whole. For instance, the chip might be represented by a simulatable VHDL description with both a structural and a behavioral component. This strategy will be developed more fully in Chapter 6.

Regardless of the representation, two sets of parameters must be derived. These parameters are the DBT activity parameters used to analyze datapath power consumption and the ABC parameters used to predict control path power consumption. Strategies for generating both sets of parameters will be discussed in the following two sections.

DBT Activity Parameters

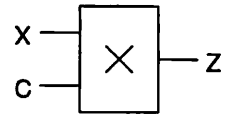
The activity parameters for the Dual Bit Type model are word-level statistics that help to determine the relative activities of the UWN and sign regions of datapath modules and buses. In particular, these statistics include: mean (μ), variance (σ^2), correlation (ρ), and sign transition

probabilities.We can consider several possible techniques for deriving these parameters including statistics propagation, transfer function evaluation, and functional simulation. We will see that the first two techniques are useful for a specific class of applications, but are of limited general applicability. In contrast, functional simulation will be shown to be a general and efficient technique for deriving activity statistics.

Reminiscent of gate-level methodologies, we first consider a propagation approach. Specifically, given the statistics of the inputs to an operation, we calculate the statistics of its outputs. At the gate level, the appropriate statistics were bit-level signal and transition probabilities. At the architecture level, however, these are replaced with the word-level statistics enumerated above. As with the gate level, propagation is only practical if the allowed set of operations is restricted. For example, in DSP applications two common operations are additions and constant multiplications.

Figure 4-51a-b present the appropriate propagation equations for these two key operations, where ρ_1^X is the correlation between successive samples of data stream X. As at the gate level, the issue of reconvergent fan-out tends to complicate matters by introducing correlations between module inputs, which are not handled by these equations. We can, however, overcome this difficulty by abstracting techniques similar to those applied at the gate level. One such technique is the Correlation Coefficient Method in which pair-wise signal correlations are stored and used to improve the accuracy of probability calculations, while higher order correlations are ignored [Erc89]. Applied to the RT level, the revised propagation equations for the adder are given by Figure 4-51c. The issue of feedback is more difficult but can be addressed by approximating the portion of the flowgraph containing feedback by an equivalent non-recursive structure [Lan93].

Still, handling reconvergent fan-out and feedback by these propagation techniques introduces several approximations into the statistics calculation. These approximations can be avoided if the exact transfer function, $H(z)$, from the primary inputs to each internal node in the flowgraph is

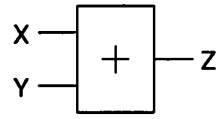


$\mu_z = C\mu_x$

$\sigma_z^2 = C^2\sigma_x^2$

$\rho_1^z = \rho_1^x$

(a) Constant multiplication



$\mu_z = \mu_x + \mu_y$

$\sigma_z^2 = \sigma_x^2 + \sigma_y^2$

$\rho_1^z = (\rho_1^x\sigma_x^2 + \rho_1^y\sigma_y^2)/\sigma_z^2$ } if X⊥Y

(b) Addition (uncorrelated inputs)

$\mu_z = \mu_x + \mu_y$

$\sigma_z^2 = \sigma_x^2 + 2\rho^{xy}\sigma_x\sigma_y + \sigma_y^2$

$\rho_1^z = (\rho_1^x\sigma_x^2 + [\rho_1^{xy} + \rho_1^{yx}]\sigma_x\sigma_y + \rho_1^y\sigma_y^2)/\sigma_z^2$

(c) Addition (correlated inputs)

Figure 4-51 : Activity propagation through typical DSP operations

known. Since these transfer functions account for all reconvergent fan-out and feedback structures, no approximations are involved. Given these transfer functions, evaluation of the desired statistics reduces to a series of numerical integrations:

$$\mu = H(0)\,\mu_{in} \qquad \text{(EQ 136)}$$

$$\sigma^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(j\omega)|^2 S_{in}(j\omega)\,d\omega \qquad \text{(EQ 137)}$$

$$\rho = \frac{\frac{1}{2\pi}\int_{-\pi}^{\pi} |H(j\omega)|^2 S_{in}(j\omega)\cos(\omega)\,d\omega}{\sigma^2} \qquad \text{(EQ 138)}$$

where $S_{in}(j\omega)$ is the power spectrum of the primary input to the flowgraph.

The problem then reduces to one of generating the necessary symbolic transfer functions, $H(z)$, from the primary input to each node in the flowgraph. This can be accomplished by representing the flowgraph as a set of network equations in matrix form:

$$[\mathbf{A} + z^{-1}\mathbf{B}]\,\mathbf{h} = \mathbf{U} \qquad \text{(EQ 139)}$$

where $\mathbf{h}$ is a vector of nodal transfer functions, $\mathbf{U}$ is a vector of input sources, and $\mathbf{A}$ and $\mathbf{B}$ are matrices of constant coefficients describing the topology of the flowgraph in question. The transfer functions can be found by solving for the symbolic matrix inverse, $[\mathbf{A} + z^{-1}\mathbf{B}]^{-1}$. Techniques to solve this set of symbolic equations are readily available in the literature [Bel75][Kad81].

Both parameter propagation and transfer function evaluation have several limitations. First, both techniques produce mean, variance, and correlation statistics, but do not directly provide the sign transition probabilities which are also required for DBT power analysis. These probabilities can be derived indirectly, however, by assuming a distribution for the signals involved. Noting that many typical DSP signals are closely approximated by Gaussian processes, the multivariate normal distribution is perhaps the best choice for a distribution. Unfortunately, calculating the sign transition probabilities for a two-input module requires the evaluation of a quadrivariate (i.e. fourth-order) normal distribution function. While approximate techniques for evaluating such distribution functions have been developed, they are often relatively expensive [Gup63]. Another disadvantage of the propagation and transfer function approaches is that they both apply to a very restricted set of applications. Specifically, DSP applications - and even more specifically, linear time-invariant (LTI) DSP applications.

Functional simulation provides a more widely applicable strategy for deriving DBT activity parameters. Given a set of typical input vectors (data and instruction) for the chip, the design is simulated at the functional (or RT level). During this simulation, the required statistics, including sign transition probabilities, are accumulated directly. Actually, we can achieve even faster activity analysis by using behavioral, rather than functional, simulation and subsequently mapping the behavioral activities to the physical structure. This approach will be described more fully in Chapter 5, but has the disadvantage that we lose certain timing information that may affect power consumption. Therefore, functional or RT-level simulation is, in general, the more desirable

approach.

As at the gate level, the drawback of simulation is that it must be carried out over a long enough sequence of data samples to allow the statistics to converge to their true values. This allows reconvergent fan-out and feedback to exert their influence on the activity of the design. The exact number of simulation cycles required depends on the particular design being analyzed; however, it is possible to monitor the activity statistics for convergence and then to terminate the simulation once the desired confidence interval is reached. This approach was developed in detail by Van Oostende in [Van92]. He states that most designs require much less than 100 cycles to adequately converge. Since functional simulation is *much* faster than gate-level simulation, run time is not an important concern and is typically on the order of seconds or minutes. The relative speed stems from the significant reduction in the number of components at the architecture level, as well as the move from bit-level to word-level computations.

The output of the simulation process is a set of activity parameters that can be fed into the DBT power analysis models. The mean, variance, correlation, and sign transition probabilities allow the tool to compute the average capacitance switched during a single invocation of each module. Another output of the simulation process is a count of the number of times each module is accessed. In general, this amounts to keeping track of the number of data and control transitions, N_{DT} and N_{CT}, at the inputs of each module. For a synchronous system, this access count may be less than or equal to the number of clock cycles simulated; or, in the presence of block-level glitching, it may actually be higher.

All of these activities are specific to the data and instruction input vectors that were supplied by the user. In this way, the designer can characterize how much power his architecture consumes for various data and instruction streams. Such a characterization might prove useful at the algorithm level in designing "low-power" software that consumes the least amount of power while achieving a desired functionality on a given processor.

To summarize, activity propagation, symbolic transfer function evaluation, and functional simulation are all techniques that can be used to derive the activity statistics required for DBT power analysis. The propagation and transfer function strategies are most applicable to the domain of DSP applications, while functional simulation is a general scheme that can be used on any target architecture. The widespread applicability and efficiency of functional simulation make it the most useful technique for deriving DBT activity parameters in the vast majority of cases.

ABC Activity Parameters

We must also consider techniques for deriving the activity of the control signals and the logic which generates them. The activity of a control word (which may take on symbolic values) is described by three parameters: the number of transitions it makes during the simulation (N_{CT}) and the average signal and transition probabilities (α, P) for the control bits during a single symbol transition. While only these parameters are required when analyzing interconnect power consumption, the ABC model for controller blocks requires two sets of activity parameters: one for inputs and one for outputs. In particular, we need to know the number of number of transitions the input word to the control module makes (again, N_{CT}), as well as the input activities (α_I, P_I) and output activities (α_O, P_O) associated with each controller module. Similar to the datapath case, this activity information can be derived by either activity propagation or simulation.

First, consider activity propagation as a means of deriving ABC activity statistics. Since the control tables can be viewed as a sum-of-products boolean logic network, activity propagation through a controller is identical to the gate-level probability propagation problem dealt with in Chapter 3. The solutions suggested in that chapter, therefore, can be applied equally well to this problem. This propagation requires a binary mapping of all control signals and experiences the same difficulties in handling correlated signals due to reconvergent fan-out and feedback. So, as in the DBT case, it is in most cases best to rely on functional simulation as a means of deriving activity statistics. The following discussion will address simulation-based activity derivation for

control buses and then extend the concepts to cover controller modules.

Power consumption on control wires is triggered by transitions of the control signals. Therefore, during functional simulation we must monitor the control transition count, N_{CT}, for each control word. We must also monitor two other ABC activity parameters. The first is the signal probability, P. This is the probability that a bit in the binary representation of the control word is one. If a binary encoding for the control signals is provided by the user, then this probability can be computed exactly; however, if symbolic values are used, as is likely the case, we must estimate the signal probability. If we assume a random encoding, the probability should be approximately one-half, $P = 1/2$. The next activity parameter is the transition activity, α. This is the average fraction of bits in the binary representation that toggle when the control word transitions from one value to another. Again, if binary values are not provided, we can assume that during an average transition half of the bits will toggle, $\alpha = 1/2$.

These strategies can be extended to handle activity derivation for controller modules. For this case, there are two important sets of (α, P) activity parameters: one for inputs and one for outputs. The input and output words may consist of several individual signals bundled together. In order to come up with an overall activity for the controller, the individual activities of these signals must be combined in some manner. For example, a transition of the controller input word is defined to occur when *any* of the component input signals transition, since this will initiate a re-evaluation of the combinational logic block. Using this rule, N_{CT} for the module can be accumulated during functional simulation. Each input transition should also trigger an update of the two other pairs of activity parameters: (α_I, P_I) and (α_O, P_O). If we assume a random encoding, we again expect half of the input and output bits to be one, so $P_I = P_O = 1/2$. As for the transition activities, the following expressions apply:

$$\alpha_I = \frac{\sum_{i \in \{\text{Inputs}\}} \frac{1}{2} N_{I_i} N_{CT_i}}{N_I N_{CT}} \qquad \text{(EQ 140)}$$

$$\alpha_O = \frac{\sum_{i \in \{\text{Outputs}\}} \frac{1}{2} N_{O_i} N_{CT_i}}{N_O N_{CT}} \qquad \text{(EQ 141)}$$

where i refers to the individual enumerated-type signals that make up the input and output words. The input activity equation, (EQ 140), can be explained as follows: $\frac{1}{2}N_{I_i}$ is the number of bits that switch on average when input i makes a transition. This is then weighted by the total number of transitions that input i makes, N_{CT_i}, to yield the number of bits within input i that toggle over the entire simulation. This is then summed across all of the individual inputs to yield the total number of bits that switch in the full input word during the simulation. Dividing this by the number of controller input transitions, N_{CT}, and the total number of input bits, N_I, results in the fraction of input bits that toggle during a typical input transition - that is, α_I. (EQ 141) is subject to a similar interpretation.

To review, the ABC activity parameters can be derived using functional simulation of chip behavior at the same time that the DBT parameters are being accumulated. The ABC activity parameters depend on the encoding of the control words as binary values. If the user specifies the actual mapping, then simulated activities will be quite accurate; however, more likely the user will specify abstract symbolic types for the control signals and state machine control tables. In this case, a random assignment can be assumed in order to allow estimates of the control signal activities to be formed. The ABC activity parameters include a count of the control word transitions that occur during the simulation period, as well as the average signal and transition probabilities for the bits forming the control words.

4.5.3 Chip-Level Summary

This section has focused on strategies for deriving the parameters required by the DBT and ABC power analysis models. These include complexity parameters, as well as activity parameters. A structural description of the chip in RTL format is required to ascertain the complexity (e.g. word length, storage capacity, etc.) of the DBT components. A behavioral description of the

design is also required and provides information necessary to extract the complexity of the control path, as well as the activity of the entire chip - datapath, memory, control, and interconnect.

4.6 Chapter Summary

This chapter introduced a comprehensive collection of techniques for analyzing the power consumption of chips at the architecture or RT level of abstraction. Specific strategies for analyzing the power of the four main classes of chip components: datapath, memory, control, and interconnect were described. For the first two classes, a new model called the Dual Bit Type (or DBT) model was developed and was demonstrated to provide excellent results, with power estimates typically within 10-15% of results from switch-level simulations. As shown in Chapter 3, previous attempts at architectural power analysis have been less accurate with error rates approaching 100% in some cases. This can be attributed primarily to weaknesses in the way these tools model circuit activity. The DBT model achieves its high accuracy by carefully modeling both physical capacitance and circuit activity. The key concept behind the technique is to model the activity of the most significant (sign) bits and least significant (UWN) bits separately.

The DBT model applies only to parts of the chip that manipulate data. A separate model was introduced to handle power estimation for control logic and signals. This model is called the Activity-Based Control (ABC) model, and as the name implies, it too improves upon contemporary techniques by accurately reflecting the effect of activity on control path power consumption. The method relies on the observation that although the implementation style of the controller (e.g. ROM, PLA, random logic, etc.) can heavily impact the power consumption, it is still possible to identify a number of fundamental parameters that influence the power consumption regardless of the implementation method. These parameters can be classified as either complexity or activity parameters. Using these target-independent parameters, the library designer is free to choose a capacitance model specific to the target implementation style. The flexibility and accuracy of the modeling technique was demonstrated for ROM, PLA, and random

logic based controllers. The random logic controller was the most difficult to model deterministically, but the ABC model was still able to provide results with an rms error of only 15% relative to switch-level estimates.

In a chip, datapath, memory, and control blocks are joined together by an interconnect network. The wires comprising the network have capacitance associated with them and, therefore, driving data and control signals across this network consumes power. The precise amount of power consumed depends on the activity of the signals being transferred, as well as the physical capacitance of the wires. The DBT and ABC models provide the activity information for control and data buses, but the physical capacitance depends on the average length of the wires in each part of the design hierarchy. This chapter presented a strategy for average wire length estimation that relied on obtaining good estimates of chip area. The area estimation problem was tackled for four types of blocks: datapath, memory, control, and composite. The estimation process was defined hierarchically, allowing the analysis to make use of partitioning hints provided by the designer in his structural chip specification.

To a large extent, the accuracy of all the component models presented in this chapter depends heavily on how accurately the complexity and activity parameters of the chip can be estimated. The final section of this chapter dealt with this issue, proposing the use of RT-level functional simulation to derive required activity parameters. Other techniques such as parameter propagation and transfer function evaluation were also discussed, but were shown to have several drawbacks including difficulty handling correlations due to reconvergent fan-out and feedback. Moreover, they were applicable primarily to a small subset of algorithms and architectures (e.g. DSP), whereas the application domain of the functional simulation strategy is basically unlimited.

This chapter has introduced many concepts that we feel represent significant advances in the state of the art in power analysis. As yet, however, these concepts have only been presented in a fairly abstract context. No unified power analysis tool that implements these various models has

been described. Until the accuracy and utility of such a tool is proven, the ideas presented here are of more theoretical than practical interest. The next several chapters will remedy this situation. In particular, Chapter 5 will present a first implementation of an architectural power analysis tool called SPADE, which uses several of the strategies developed in this chapter. The intended application domain for SPADE is DSP systems. A more general power analysis tool called SPA will be described in Chapter 6. SPA provides a full implementation of all the power analysis models presented in this chapter. Finally, Chapter 7 will describe how architectural and higher-level power analysis tools can be used to facilitate a coherent methodology for low-power design. The chapter will introduce the notion of algorithmic power estimation and will provide a case study that demonstrates the large power savings that are made possible by CAD-assisted design space exploration.

CHAPTER 5

SPADE: A Stochastic Power Analyzer with DSP Emphasis

The first implementation of an architectural power analysis tool to be discussed is SPADE - a Stochastic Power Analyzer with DSP Emphasis. As the name implies, SPADE is targeted towards DSP applications. Within this domain, SPADE implements several of the power models described in the previous chapter. The purpose of this chapter is to discuss implementation details of SPADE that differ from what has been discussed so far, as well as to demonstrate how architectural power analysis can be applied to real-world design problems.

The organization of the chapter is as follows. Section 5.1 will describe the HYPER high-level synthesis environment in which SPADE resides. With this background, Section 5.2 will go on to describe how SPADE performs architectural power analysis. This section is divided into subsections on datapath, memory, control, and interconnect, and for each class of component it will describe how the analysis models used by SPADE differ from the models of Chapter 4. Finally, Section 5.3 will present the results of applying SPADE to several practical designs of varying complexity. This section will demonstrate the accuracy SPADE provides by comparing its results to switch-level simulations of laid-out and extracted chips. It will also show how SPADE

can be used to evaluate the efficacy of power optimization strategies that cannot be analyzed with existing architectural power analysis tools.

5.1 The HYPER Environment

SPADE resides in the HYPER environment - a high-level synthesis framework targeted at datapath-intensive DSP applications. This includes applications ranging from filtering and compression for speech and video to equalization and coding for telecommunications. Such algorithms are typically implemented on clusters of heavily-pipelined datapaths with execution directed by a relatively simple controller.

As an example, Figure 5-1 shows the computational structure for a Viterbi processor, which might be used in speech recognition applications [Rab91]. This algorithm embodies the aforementioned characteristics, requiring a large amount of computation that is applied in a very regular way. Therefore, implementation of the Viterbi typically consists of a complex datapath, but a very simple controller. The designer of this processor might consider several different implementation platforms, ranging from a highly time-multiplexed, Von Neumann architecture to a concurrent, pipelined processor. In the end, the complexity of the algorithm coupled with the real-time requirements result in a fairly high throughput constraint, and this precludes a significant degree of resource sharing. The opposite extreme, consisting of fully pipelined and dedicated hardware, while meeting the real-time requirements, might not represent the optimum implementation in terms of silicon real-estate (cost). Unfortunately, the vastness of the global design space and time-to-market considerations often prohibit the designer from seeking out the optimum middle-ground solution.

This is precisely the motivation for the HYPER high-level synthesis system. HYPER automates the task of design space exploration, allowing the user to rapidly synthesize several distinct algorithmic and architectural alternatives, and then to select the optimal solution. In order

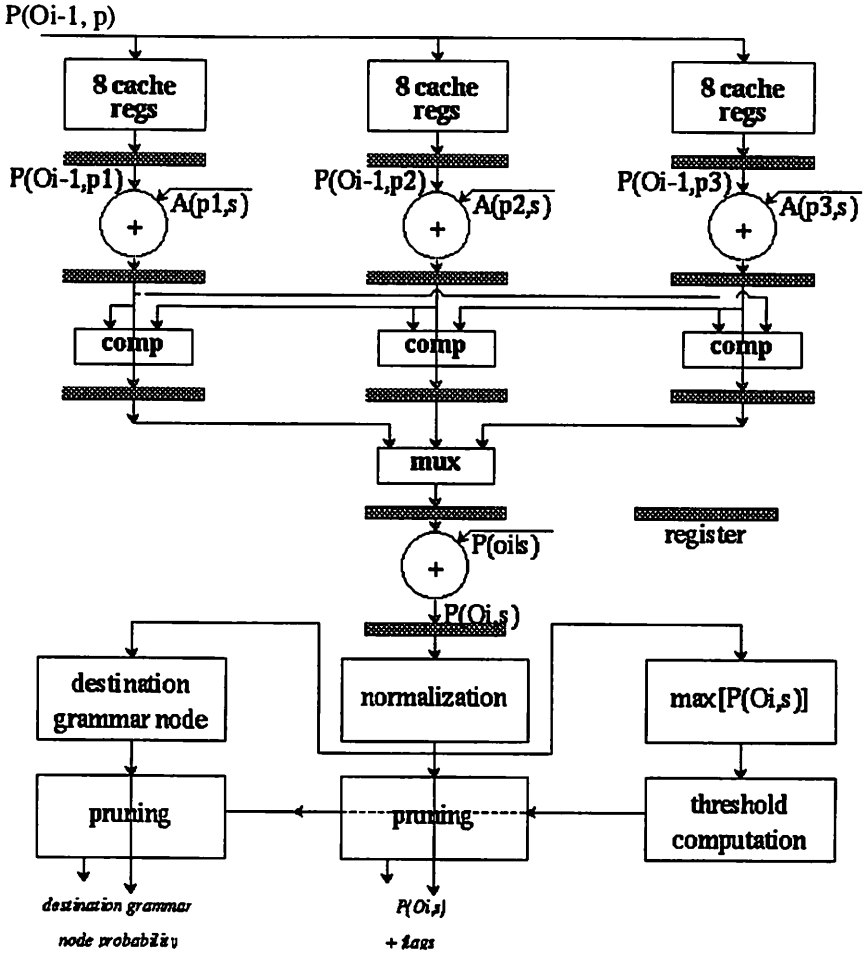


Figure 5-1 : Computational structure for Viterbi processor (from [Rab91])

to achieve this objective, HYPER relies on several layers of tools.

Figure 5-2 contains an overview of the HYPER design environment. Initially, the user specifies their algorithm in the Silage high-level, applicative language [Hil85]. At this point, the user has access to several classes of tools:

- Parsing: Takes in Silage in textual format and produces an internal data structure known as the Control-Data Flow Graph (CDFG).
- Simulation: Verifies functionality of CDFG through behavioral simulation. May be

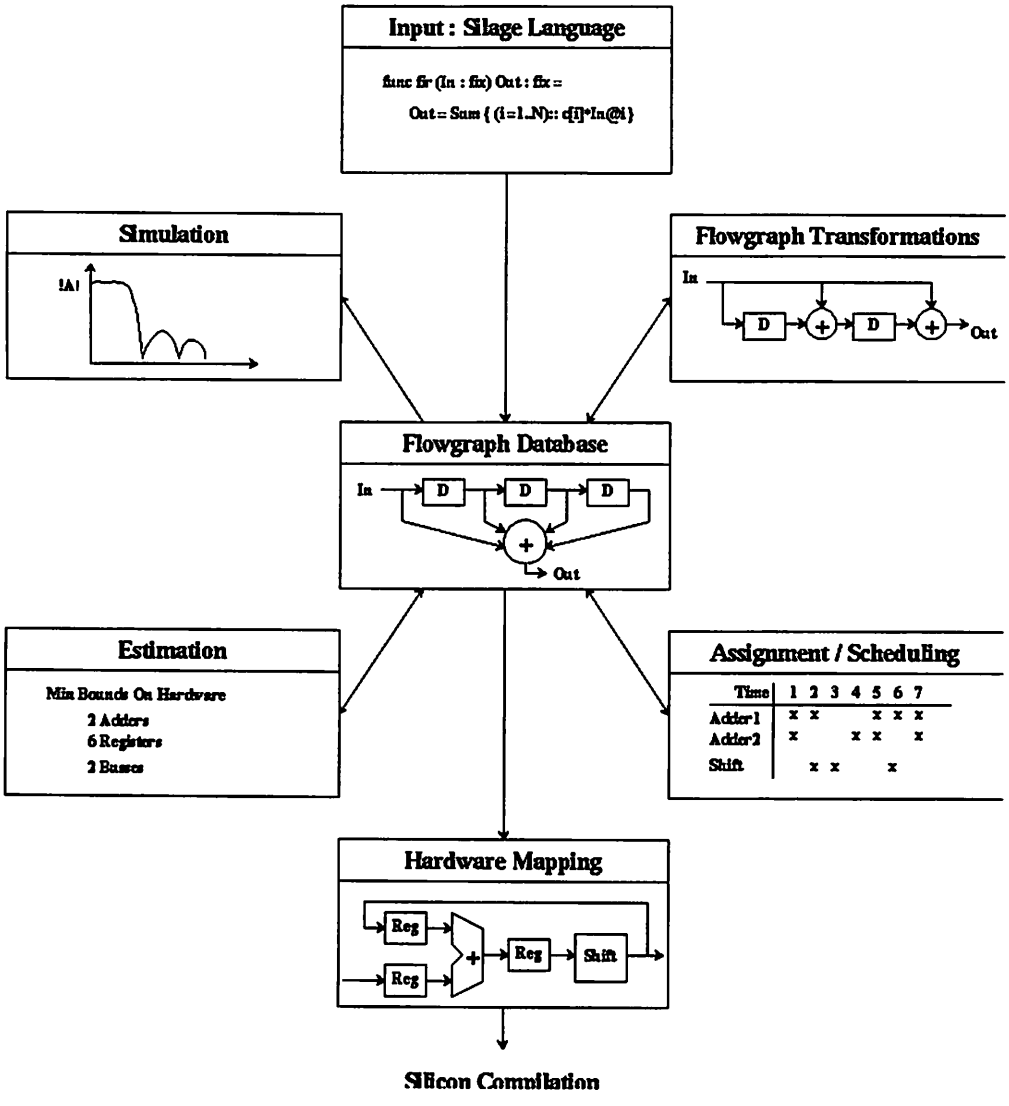


Figure 5-2 : Overview of HYPER environment (from [Rab91])

applied at any stage in the design flow.

- Module Selection: Selects an appropriate hardware library element to implement each operation based on area, timing, and power constraints. This is the first step in defining the processor architecture.
- Transformations: Provides a library of behavioral and structural manipulation routines targeted at improving various cost factors such as area, speed, and power. This library

includes retiming, pipelining, and loop unrolling, as well as many algebraic transformations such as commutativity, distributivity, and associativity.

- Estimation: Derives minimum and maximum hardware resource bounds and provides approximate area, delay, and power values for the current flowgraph. These estimates are based on algorithmic rather than architectural considerations.
- Allocation, Assignment, and Scheduling: Given a minimal resource allocation as a starting point, iteratively converges to a fully-specified architecture in which each operation is assigned to a particular hardware unit and time slot.
- Hardware Mapping: Generates a finite state machine to control the datapath and emits a description of the architecture in a format suitable for silicon compilation or for architectural power analysis using SPADE.
- Architectural Power Analysis: Using SPADE, estimates power that the given algorithm will consume when executed by the candidate architecture.

The precise ordering of these tasks will affect the final implementation. The HYPER paradigm calls for an exploratory approach to the design problem. At each stage, estimations are used to determine the placement of the current implementation in the ATP design space. These estimations provide clues to the designer regarding bottlenecks of the current solution. Using these hints, the designer can select from a number of basic moves or transformations in order to incrementally improve the solution. So while the designer is free to manually guide the selection of a final architecture, the time-consuming transformation and estimation tasks are automated. For more information about the HYPER system, the reader is referred to the following references: [Rab91][Rab93][Pot93].

Figure 5-3 shows the graphical user interface for HYPER in the midst of performing power analysis using SPADE. In the figure, the main text window lists power estimates produced by SPADE. Bar charts are also produced summarizing the information graphically and giving a breakdown of average capacitance switched per sample by hardware class (e.g. Exu, Control,

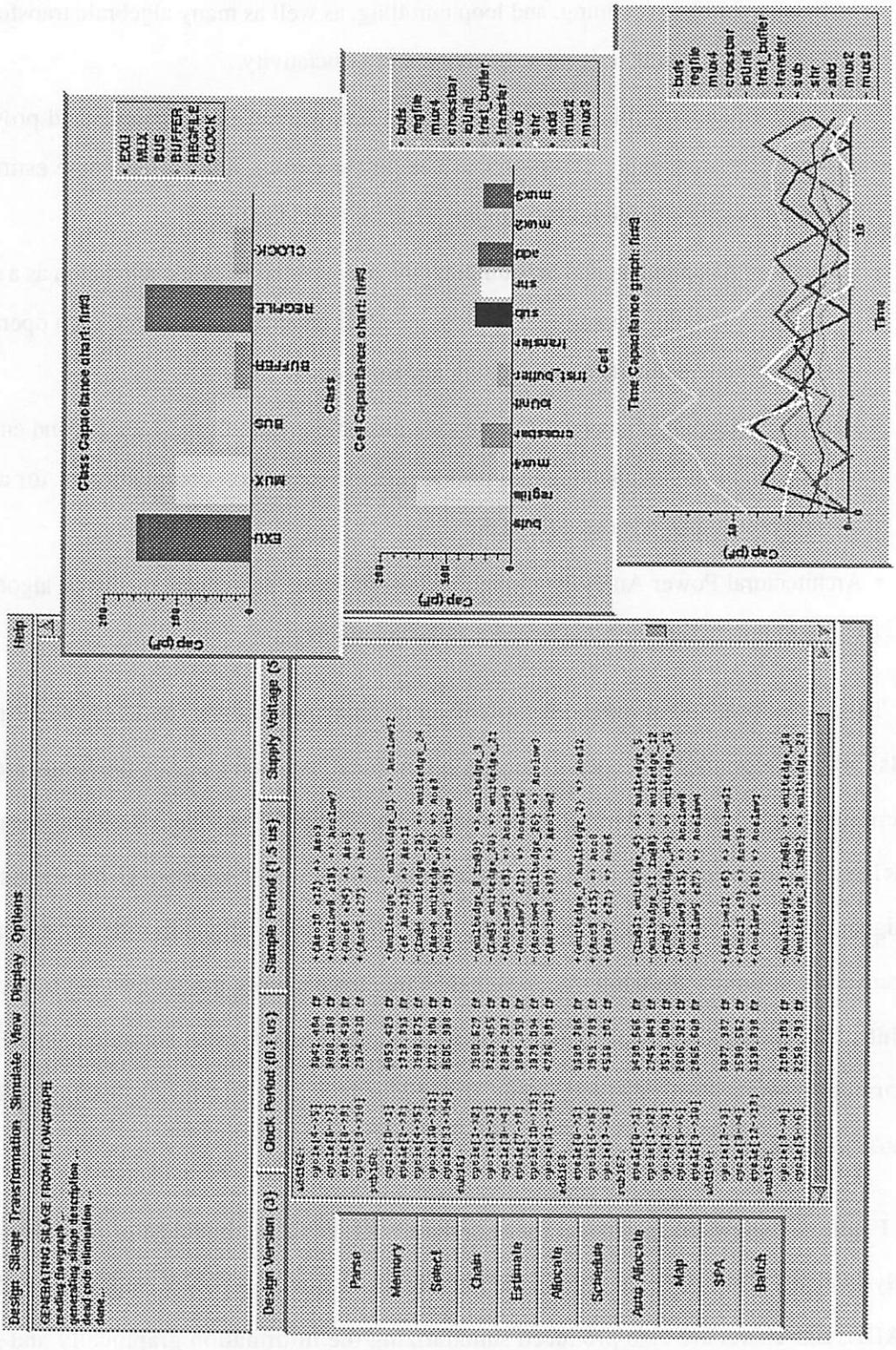


Figure 5-3 : Graphical user interface for HYPER and SPADE

Memory, etc.) and by cell type (e.g. adder, shifter, SRAM, etc.). In addition, SPADE generates a profile of power consumption versus time for each cell in the design. The remainder of this chapter discusses the inner workings of SPADE and the results that it can achieve.

5.2 Architectural Power Analysis using SPADE

After completing the hardware mapping step, the user will have two complementary descriptions of his target application: behavioral and structural. At this stage, the designer is able to invoke SPADE in order to ascertain the power that will be consumed when the candidate architecture executes the algorithmic behavior. SPADE is an RT-level power analysis tool based on the techniques of Chapter 4. Recall that the models introduced there require two types of input parameters: complexity and activity. The complexity parameters are available in a fairly straightforward manner from the structural description, however, the activity parameters must be derived from an analysis of the combined behavior and structure making up the architecture.

There are two ways of accomplishing this analysis. The first is to simulate the architecture itself (functional simulation) while gathering the necessary bus and module activities directly. The advantage of this is that it captures timing information such as time-multiplexing of signals and block-level glitching, which might affect power consumption. The second alternative is to simulate the algorithm alone (behavioral simulation) and then to derive the physical bus activities from the algorithmic signal activities. The advantage of this approach is that behavioral simulation is even faster than functional simulation. The disadvantage is that we lose valuable timing information that, in general, is difficult to recover.

So for most cases, functional simulation of the architecture is the more desirable approach. Indeed, this was the strategy chosen for the generic architectural power analysis tool SPA, which will be described in the next chapter. The class of architectures and algorithms covered by HYPER, however, is restricted and well-defined enough to allow some of the timing information

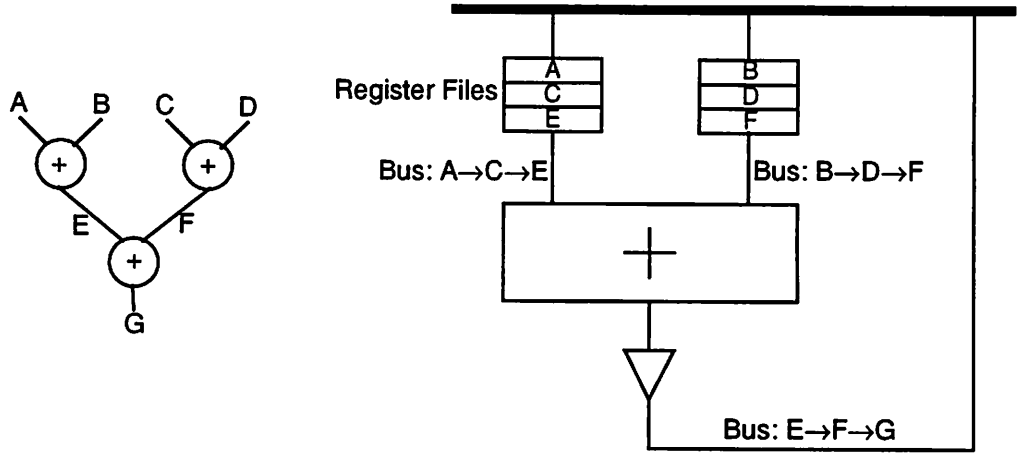
to be reconstructed given only the results of a behavioral simulation and a set of behavioral/ structural bindings. Therefore, SPADE opts for this faster, though less general technique as will be detailed in the following sections.

5.2.1 Activity Analysis

There are two important categories of activity. The first is the activity associated with the hardware modules, while the second is the interconnect or bus activity. These two components are actually closely related since the modules take their inputs from buses (global or local). Therefore, by examining the statistics of the bus data, SPADE can determine both the module and interconnect activity.

Since the edges of the behavioral flowgraph (CDFG) correspond to signals, it might appear that the behavioral view provides all necessary data statistics. This is not the case, however, since we are interested in *bus* statistics and the CDFG edges do not correspond to physical buses. In reality, we must consider both the behavioral and the structural views simultaneously in order to determine bus statistics. This process requires two steps. In the first step, simulation of the CDFG provides flowgraph signal statistics for a given set of input vectors. Next, scheduling and assignment information can be used to map the flowgraph signals onto the physical buses. This two-step process results in the desired bus statistics.

The distinction between flowgraph signals and bus signals is illustrated by Figure 5-4. Since more than one CDFG operation may be bound to the same hardware module, the inputs seen by the module will be a time-multiplexed version of the flowgraph signals. In other words, at each clock cycle during a sample period, a different signal from the CDFG might be active on a physical bus. The behavioral simulation of the CDFG provides the statistics of the CDFG signals, which give us the statistics of the buses, registers, and memories, at each clock cycle. So instead of having a single set of bus statistics averaged over the entire simulation period, we have a whole sequence of statistics for each bus. The impact of this on the DBT model will be discussed in the



(a) Behavior: operations and signals (b) Structure: modules and buses

Figure 5-4 : Mapping of flowgraph signals to physical buses

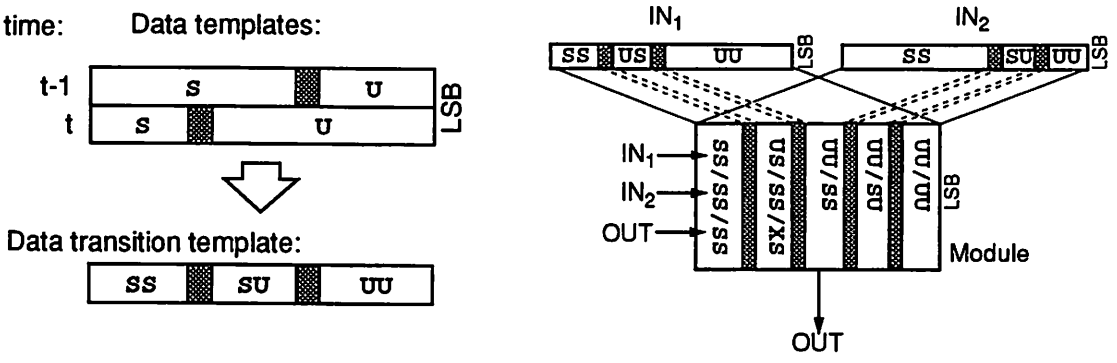
next section.

5.2.2 Datapath and Memory

After deriving the activity parameters required by the DBT model, SPADE then proceeds to evaluate the power consumed by the datapath and memory elements. It does this by stepping through each of the architectural entities and applying the DBT power analysis method.

The method is similar to that described in Chapter 4. That discussion, however, assumed that the activity of a bus or module is described by one set of aggregate statistics - mean, variance, correlation, and sign transition probabilities - that applies over the entire simulation lifetime. Since these statistics determine the location of the model breakpoints, a single set of "average" breakpoint locations were used to determine regional decompositions at all points in time.

In SPADE, however, the (possibly varying) statistics of a data bus are known on a clock cycle by clock cycle basis. This means that a single data bus undergoing a transition may have different



(a) Data transition templates

(b) Regional decomposition of two-input module

Figure 5-5 : The effect of non-stationary statistics on the DBT model

Transition Templates	Capacitive Coefficients			
UU	C_{UU}			
SS	C_{++}	C_{+-}	C_{-+}	C_{--}

(a) Original coefficients

Transition Templates	Capacitive Coefficients	
US	C_{U+}	C_{U-}
SU	C_{+U}	C_{-U}

(b) Additional non-stationary coefficients

Table 5-1 : Capacitive coefficients for one-input modules with non-stationary statistics

breakpoints at clock cycle t than it did at t-1 as illustrated by Figure 5-5a. The result is a transition template with possibly three regions: a sign region, a UWN region, and a hybrid region in which bits may transition from a UWN value to a sign value or vice-versa. This, in turn, means more transition ID's and, consequently, more capacitive coefficients to cover the new transition possibilities. Table 5-1 shows the full set of capacitive coefficients used by SPADE to characterize buses or one-input modules.

Transition Templates	Capacitive Coefficient Counts
UU/UU	1
UU/SS	4
SS/UU	4
SS/SS/SS	64

(a) Original coefficients

Transition Templates	Capacitive Coefficient Counts
UU/US	2
UU/SU	2
US/UU	2
SU/UU	2
US/SU	4
SU/US	4
US/US/XS	8
SU/SU/SX	8
US/SS/XS	16
SU/SS/SX	16
SS/US/XS	16
SS/SU/SX	16

(b) Additional non-stationary coefficients

Table 5-2 : Capacitive coefficient for two-input modules with non-stationary statistics

Consider the effect of this extension on the regional decomposition of a two-input module. As Figure 5-5b demonstrates, the new input breakpoints can add as many as two additional regions to the module. So instead of a maximum of three regions, a module with misaligned breakpoints may now have up to five regions. Again, this introduces new transition ID possibilities, and the number of capacitive coefficients required for two-input modules expands as shown in Table 5-2.

This discussion describes how SPADE handles the case of two signals multiplexed onto the same bus over two successive clock cycles. Additional multiplexing during subsequent clock cycles can be handled without any additional coefficients by applying this strategy on a cycle-by-

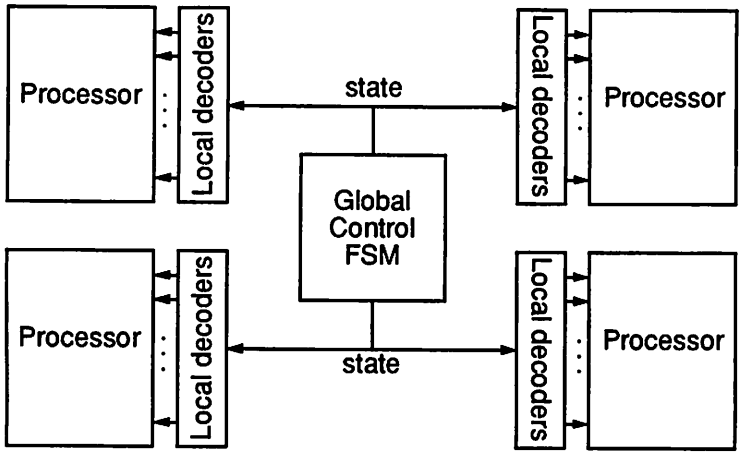


Figure 5-6 : Distributed control scheme in HYPER

cycle basis for each clock cycle in an average sample period.

Aside from these distinctions, datapath power analysis under SPADE follows the methodology described in Chapter 4.

5.2.3 Control Path

HYPER relies on a fixed and well-defined control model. The sequence of states for an implementation is generated by a central global controller which broadcasts the current state to all processors in the system. Control signals for each processor cluster are then generated locally based on these global state lines. This distributed configuration is illustrated by Figure 5-6. The fact that HYPER is targeted towards a fixed application domain and employs a fixed hardware model for the control path simplifies the power analysis task. In particular, the full-blown ABC model is unnecessary; instead, a statistical modeling approached based on a priori characterization of a number of representative benchmarks works well in this situation.

The model (developed by Mehra) relates the power consumed by several previously designed controllers to parameters such as the number of states in the finite state machine (FSM) and its

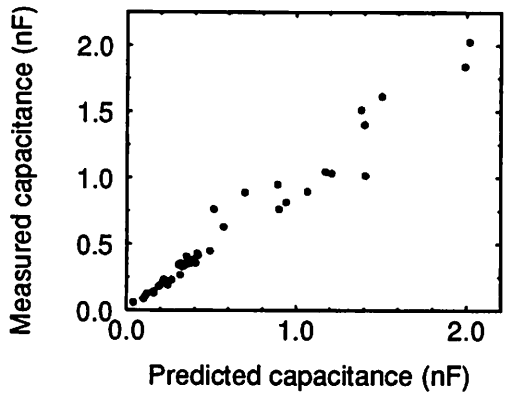


Figure 5-7 : Predicted controller capacitance vs. switch-level simulation (after [Meh94])

output control signal activity [Meh94]. For the global FSM, Mehra found that the effective capacitance follows the model:

$$C_{FSM} = \alpha N_{states} + \beta \quad \text{(EQ 142)}$$

Likewise, the capacitance switched by the local controllers was found to be expressed by:

$$C_{LC} = \beta_0 + \beta_1 N_{trans} + \beta_2 N_{states} + \beta_3 B_f \quad \text{(EQ 143)}$$

where N_{trans} is the number of transitions on the control signals. The bus factor, B_f, is the ratio of the total number of point-to-point connections in the flowgraph to the number of physical buses - in other words, B_f is an estimate of the number of accesses per bus.

The coefficients for both models were extracted from a set of 46 benchmarks representing a wide cross-section of DSP applications. The resulting aggregate model is depicted in Figure 5-7. The average and maximum modeling errors for the benchmark set are 12% and 41%, respectively.

5.2.4 Interconnect

The physical capacitance of interconnect can be determined from the structural information produced by the hardware mapper. Since SPADE operates before placement and routing of the

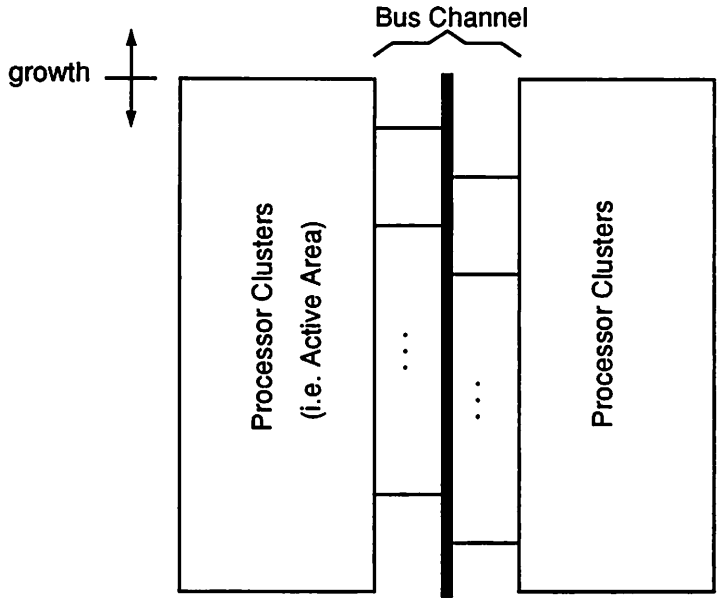


Figure 5-8 : Floorplanning model employed by HYPER

chip, however, estimates of interconnect capacitance must be used in place of actual values. Since interconnect length is related to chip area as discussed in Chapter 4, area estimates can be used to approximate wire lengths:

$$L = k\frac{\sqrt{A}}{3} \qquad \text{(EQ 144)}$$

In Chapter 4, the proportionality constant, k, was said to be approximately equal to 3/5. If desired, k can be customized for HYPER, by fitting it to the results of several benchmark examples.

The fairly strict floorplanning model (see Figure 5-8) employed by HYPER, makes the task of estimating area relatively straightforward. Mehra proposed the following empirical model [Meh94]:

$$A = \alpha_1 + \alpha_2 A_{act} + \alpha_3 N_{bits} N_{bus} A_{act} \qquad \text{(EQ 145)}$$

where A_{act} is the total active area of the chip (read from the hardware database), N_{bits} is the average word length of the buses, and N_{bus} is the number of buses. The second term in the

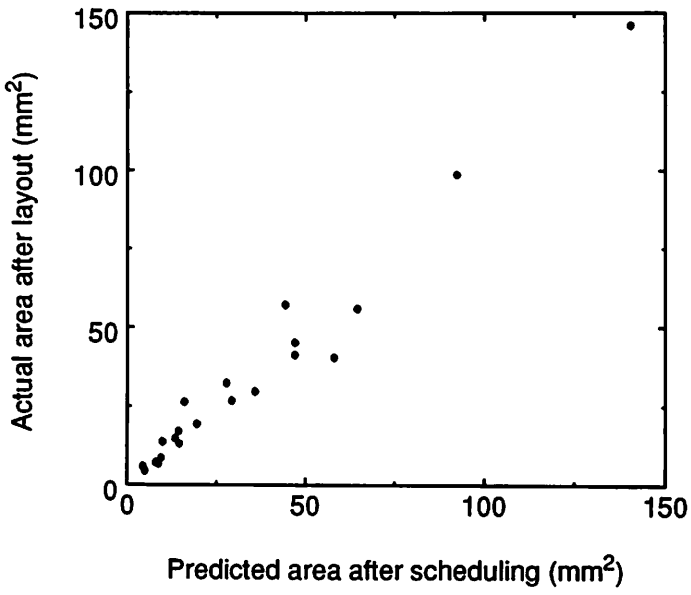


Figure 5-9 : Predicted vs. actual area results for HYPER (after [Meh94])

equation accounts for the area of the processing elements, while the third term account for the area of the bus channel. In this term, $N_{bits}N_{bus}$ is proportional to the width of the routing channel, and A_{act} is proportional to the length of the channel. The length is proportional to A_{act} because HYPER uses a linear floorplanning strategy in which additional processing clusters add directly to the length of the design. The correlation of the actual implementation area to the predicted area is shown in Figure 5-9. The average and worst-case errors are 17% and 44%, respectively. Of course, if the routed chip is available, back-annotated capacitances could be used in place of estimates.

5.3 Results

The previous sections have described the environment and operation of SPADE. The purpose of this section is to demonstrate the usefulness and accuracy of the proposed approach on some realistic examples. In particular, the first goal will be to verify that SPADE and the power analysis models of Chapter 4 provide accurate results. This will be accomplished by comparing the power

figures from SPADE to switch-level simulations of extracted chip layouts. In addition, this section will show how SPADE can be used not merely to provide back-end verification of power consumption, but also to optimize architectures for low power. Finally, the examples will demonstrate how SPADE can facilitate the development of new techniques for low-power design.

The remainder of the section is divided into three parts. Each part demonstrates a distinct application of SPADE. For instance, the first example shows how SPADE can be used as a pure *analysis* tool and will demonstrate accuracy close to that of switch-level tools. The second example focuses on the use of SPADE as an architectural exploration and *optimization* tool, taking a design through several revisions to reach the desired low-power solution. Finally, the third example indicates how SPADE can aid researchers and designers alike in the *development* of new low-power design techniques such as those described in Chapter 2.

5.3.1 Analysis Example

In this example, we will analyze the power consumed by a noise cancellation chip. The chip uses an adaptive least mean squares (LMS) algorithm. Such a chip would be used in voice communications for situations where interference due to background noise greatly hinders speech intelligibility. For example, the voice of a pilot communicating with the control tower can often be overpowered by the background noise from the aircraft engines. In this situation, noise cancellation techniques are critical to reduce interference from background noise and increase overall speech intelligibility.

A diagram depicting the LMS noise cancellation scheme is given in Figure 5-10. The chip requires two signals as input. The primary input is simply the noisy speech signal, which might be acquired by a microphone suspended near the pilot's mouth. The second is known as the reference noise input and is a sample of the ambient noise source (i.e. the background engine noise). The noisy speech signal is the superposition of clean speech with a version of the reference noise that has been filtered by some channel representing the effects of signal absorption, reflection, phase

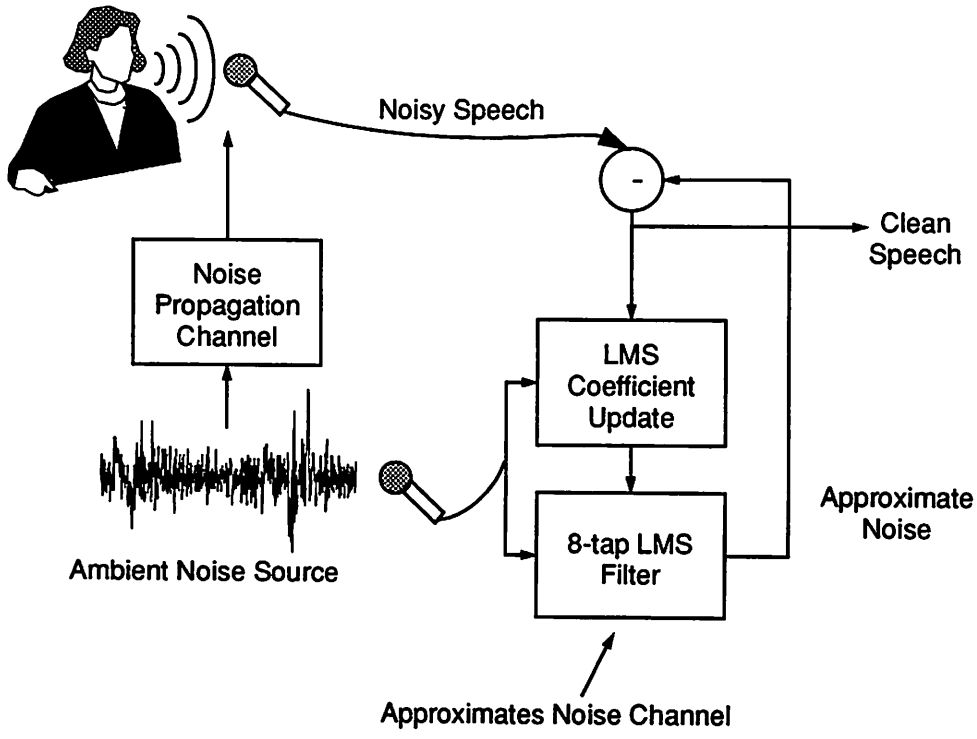


Figure 5-10 : Adaptive LMS noise cancellation scheme

reversal, etc. as the noise signal travels from the reference source to the primary input location. The noise canceller operates by adapting an internal channel filter to match as closely as possible the physical noise channel. The reference noise can then be filtered through this channel and subtracted out from the corrupted speech to produce clean speech. For further information on noise cancellation schemes the reader is referred to an excellent overview provided by Widrow [Wid75].

The noise canceller provides a good test of SPADE since it requires operations that are representative of a wide range of DSP applications. First, the chip contains a channel filter and, therefore, must employ adders, multipliers, and memory elements. Also, the chip must adapt the coefficients of this filter to match the noise channel. This adaptation loop requires control logic and also makes the algorithm non-linear. As a result, this example fully exercises the SPADE power

analysis models and provides a meaningful verification of their accuracy.

In order to apply SPADE, we must first synthesize a candidate architecture using HYPER. A Silage description of the algorithm is the input to the HYPER system. The application also requires audio inputs, which must be provided by the user to SPADE so that it can estimate the impact of input statistics on power consumption. For this example, speech and noise inputs sampled at 8 KHz were supplied to SPADE. This sampling rate also represents the real-time throughput requirements of the chip.

The estimation features of HYPER indicate that at a standard 5V supply, the critical path of the algorithm is only 468 ns while the sample period is 125 μs. This suggests that the performance of an implementation at 5V would be wasted. This performance can instead be traded for lower power by reducing the supply voltage. Assuming we are designing this chip as part of a portable system, it might be wise to select a standard battery voltage (such as 1.5 V) as the new V_{dd}. At this voltage, the critical path grows to 2.83 μs. This is still much less than the available time, but remember that the critical path is the *minimum* time in which the algorithm can possibly be executed. To actually achieve this performance often requires the use of extensive hardware resources (i.e. parallelism). Since we only need to meet the 125 μs throughput constraint, however, we can save area by time multiplexing operations onto a minimal set of units - in this case, a single multiplier, adder, and subtracter as illustrated in Figure 5-11. When this is done the algorithm requires 41 clock cycles to sequentially execute all operations. This implies a clock period of around 3 μs.

The results of invoking SPADE to analyze the power consumption for this architecture (given speech and reference noise inputs) are shown in Table 5-12. The average capacitance switched per sample is 3372 pF. This can be converted to an average power of 60.7 μW by multiplying with the square of the supply voltage and dividing by the sample period. As might be expected, most of the power (43.7%) is consumed by the execution units - primarily, the multiplier. The control logic,

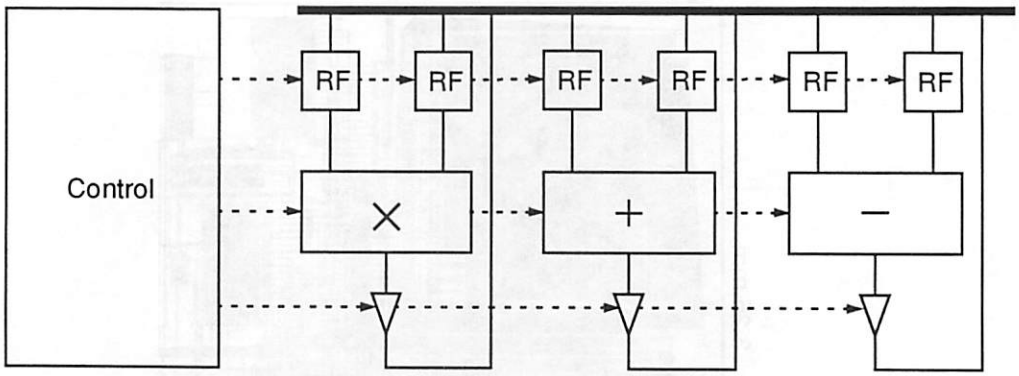


Figure 5-11 : HYPER-synthesized architecture for noise canceller

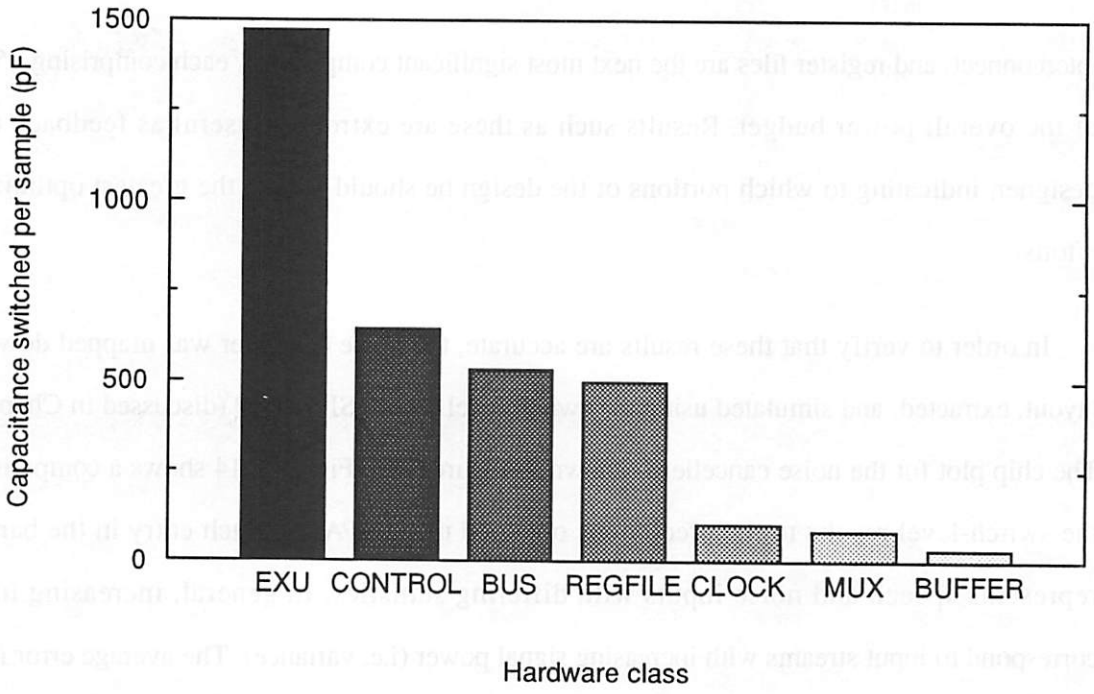


Figure 5-12 : SPADE results for noise canceller

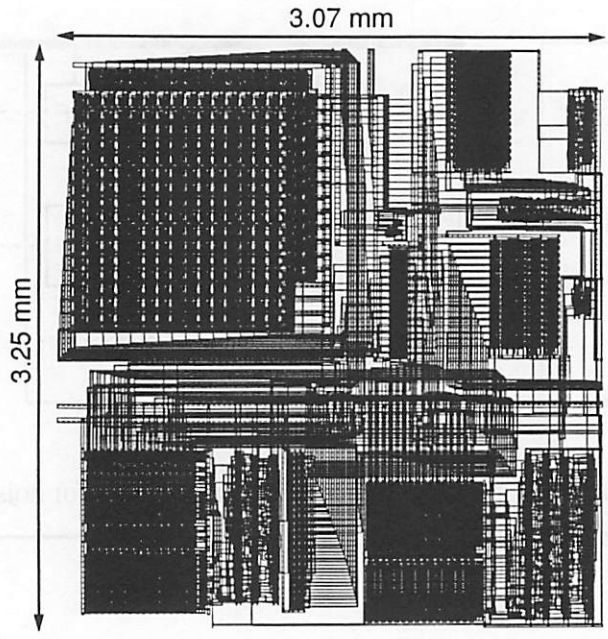


Figure 5-13 : Implementation of LMS noise canceller in 1.2μm technology

interconnect, and register files are the next most significant components, each comprising 15-20% of the overall power budget. Results such as these are extremely useful as feedback to the designer, indicating to which portions of the design he should devote the greatest optimization efforts.

In order to verify that these results are accurate, the noise canceller was mapped down to a layout, extracted, and simulated using the switch-level tool IRSIM-CAP (discussed in Chapter 2). The chip plot for the noise canceller is shown in Figure 5-13. Figure 5-14 shows a comparison of the switch-level results to the predictions obtained using SPADE. Each entry in the bar chart represents speech and noise inputs with differing statistics. In general, increasing indices correspond to input streams with increasing signal power (i.e. variance). The average error is only 4.5% and all SPADE estimates are within 16% of switch-level measurements. For reference, the power predicted by the uniform white noise model is also included in the figure. Since this model ignores signal statistics, it predicts the same power consumption for all data streams and errs by

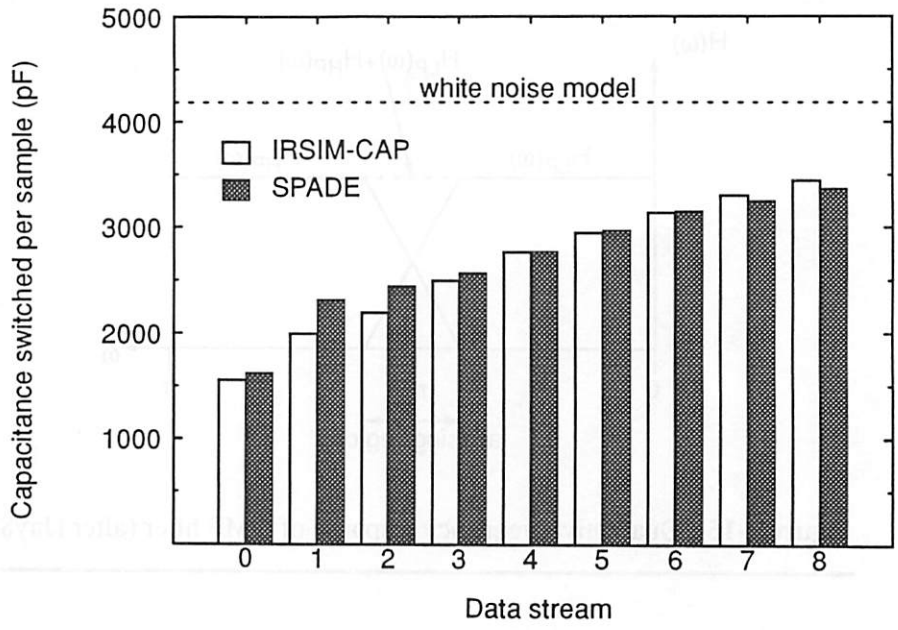


Figure 5-14 : Comparison of SPADE to switch-level simulation for different data streams

21-169% in its estimates.

SPADE's accuracy is even more impressive when we consider the relative speed of RT- versus switch-level power analysis. The HYPER front-end required one minute on a Sun SPARCstation 10 to synthesize the target architecture. From this point, SPADE took 40 seconds to predict the power consumption. Generating a layout and extracting a circuit simulation file, however, required 24 minutes and switch-level simulation for 64 samples another 67 minutes. This means that by using an architecture-level tool such as SPADE, designers can obtain power estimates 56x faster while sacrificing only 16% in accuracy. This facilitates thorough design space exploration in a reasonable amount of time. An example of how SPADE can be applied to this exploration and optimization process will now be described.

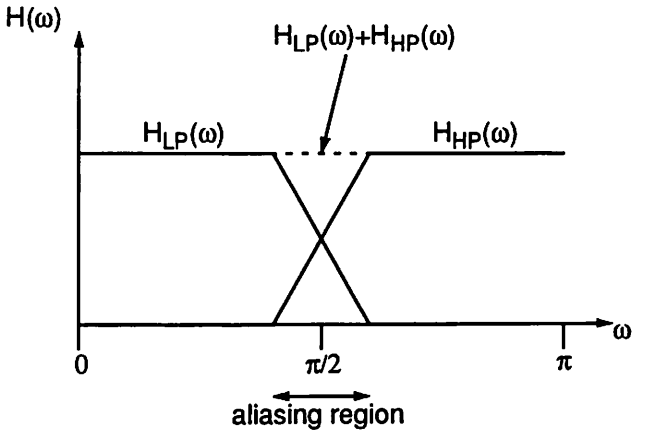


Figure 5-15 : Qualitative frequency response of QMF filter (after [Jay84])

5.3.2 Optimization Example

SPADE allows the designer to efficiently explore the design space, searching for low-power solutions. This example will demonstrate a design flow that employs SPADE to minimize the power consumed by a Quadrature Mirror Filter (QMF) [Jay84]. A quadrature mirror filter takes an input signal and splits it into two bands: a low-pass band, $H_{LP}(\omega)$, and a high-pass band, $H_{HP}(\omega)$, as shown in Figure 5-15. QMF filters are often used in sub-band coding algorithms. Sub-band coders split the frequency band into several sub-bands. Each of these sub-bands is then coded independently taking advantage of the individual properties of each band to achieve the highest possible level of compression. At the receiver, the sub-bands are decoded and summed to produce the output signal. QMF filters are often used in sub-band coders because they have nice symmetry properties that circumvent the aliasing distortion most filters with overlapping pass-bands would experience when their outputs were summed (see Figure 5-15).

In this example, four implementations of the QMF filter will be analyzed and the ATP characteristics will be compared. The sample rate chosen for the designs would, in general, depend on the application. For the purposes of this example, we select a sample period of 0.3 μs or about

Hardware Class	Initial (mW) @ 5V	Shift-Add (mW) @ 5V	Retimed (mW) @ 1.5V	Pipelined (mW) @ 1.25V
Bus	208.9	46.3	5.60	4.48
Exu	51.4	10.8	1.28	0.57
Register File	27.4	30.2	5.57	1.17
Control	25.2	27.8	2.90	0.34
Multiplexer	16.7	23.3	3.70	0.00
Clock Wiring	15.3	6.9	1.39	0.25
Buffer	3.9	4.2	0.57	0.23
Total Power	348.8	149.5	21.0	7.0
Area	95.9 mm^2	17.2 mm^2	24.4 mm^2	115.4 mm^2

Table 5-3 : SPADE power/area predictions for QMF example

3.33 MHz.

The first version is a direct implementation of the algorithm that does not employ any low-power design techniques. The power and area predictions provided by SPADE are shown in the "Initial" column of Table 5-3. Four costly array multipliers are required to meet the throughput requirements of the algorithm at 5V and this leads to a large die size of 95.9 mm^2. The sheer size of the chip necessitates long, highly capacitive global buses. The power consumed in driving these buses represents 73% of the total power budget for the chip.

In the second version of the design, the expensive array multipliers are replaced by shift-add operations, reducing the chip area to a more reasonable 17.2 mm^2. The "Shift-Add" column of Table 5-3 shows that the bus power has been drastically reduced by a factor of 4.5x. In addition, the SPADE estimates show that replacing the array multipliers with shifters and adders has reduced the execution unit power by 4.8x. In general, whether or not array multipliers or shift-adds are more efficient depends on the values of the constant coefficients being used. Overall, the shift-add version of the chip consumes 57% less power than the initial version, while at the same time occupying 82% less area.

For both of these designs, the critical path is approximately 0.3 μs at 5V, permitting no reduction in voltage supply. If we can modify the algorithm, however, to reduce the critical path, then we can consider trading some of this increased performance for lower power by scaling down the voltage. *Retiming* is one technique for reducing the critical path of an algorithm. Basically, retiming minimizes the critical path by shuffling flowgraph delays in a manner that exposes more concurrency within the algorithm. The retiming operation is available to the HYPER user through the transformation library. A third implementation of the QMF example was generated by applying retiming, which reduces the critical path to 60 ns at 5V. This allows us to lower the supply voltage of the implementation to about 1.5V while still meeting the sampling rate constraint. The "Retimed" column of the table shows results from SPADE revealing that trading performance for power in this case yields a 7.1x reduction in power. Notice that, based on SPADE's area estimates, the additional hardware required to exploit the concurrency increases the implementation area from 17.2 mm^2 to 24.4 mm^2.

While going from 349 mW to 21 mW is already a significant improvement in power, it is worthwhile to explore other architectural possibilities that might yield even lower power consumptions. For instance, a fourth version of the filter can be generated by applying the algorithmic pipelining operation (also available in the transformation library). By inserting four pipeline stages, the algorithm can be made completely parallel. If each operation is given its own dedicated hardware unit, then all required operations can be performed concurrently in a single clock cycle. This represents the minimum possible critical path for the algorithm and allows the voltage supply to be further reduced to 1.25V. The "Pipelined" column of Table 5-3 shows that SPADE confirms an additional power reduction of 3x for an overall reduction of 50x. A graphical summary of the SPADE power breakdown is given in Figure 5-16. The power of the pipelined version (shown in black in the figure) is so low compared to the original implementation that the power bars for most of its components are barely visible in the scale of the figure.

It is worthwhile to analyze the SPADE results in more detail to determine how much of an

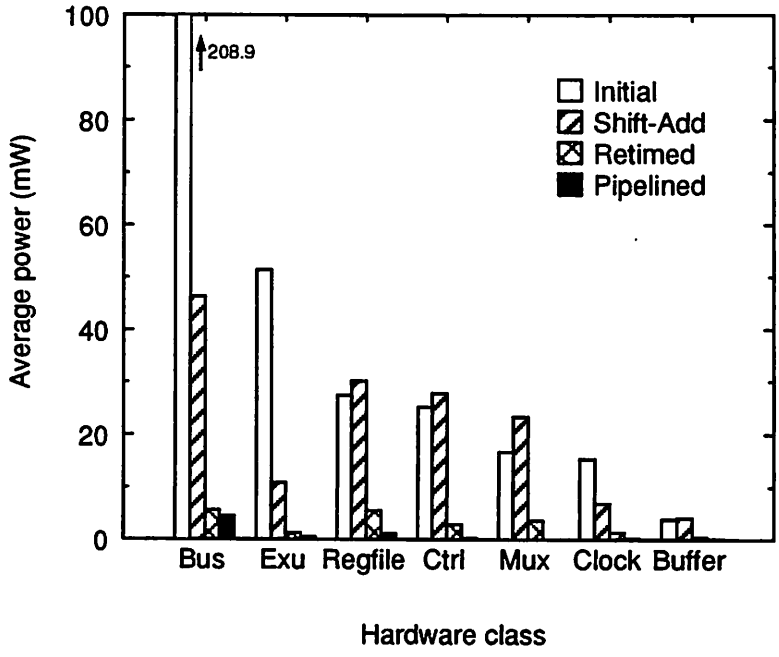


Figure 5-16 : Graphical breakdown of power consumption as predicted by SPADE

impact the various optimizations have on power. Table 5-4 indicates that the overhead involved in exploiting parallelism can involve a significant penalty not only in area, but also in power. In our example, overhead causes the retimed design to achieve a 7.1x reduction in power over the shift-add design, instead of the 11.1x that would be expected from pure voltage scaling considerations.

Interestingly enough, just the opposite occurs when going from the 21 mW retimed filter to the 7 mW pipelined design. Here, the power savings is more than would be expected from the voltage reduction alone. This can be explained by recalling some of the other low-power concepts presented in Chapter 2. The pipelined design can be thought of as a distributed processing architecture employing a dedicated hardware unit for each required operation. This means that the entire algorithm can be executed in a single clock cycle, eliminating the need for complex control. In addition, since operations aren't multiplexed onto the same hardware units, signal correlations are preserved, reducing the switching activity. Fully 54% of the power saved by pipelining can be

Design Version	Influencing Factors	Resulting Power (mW)	Impact on Power (Δ%)
Initial	Array multipliers	348.8	-
Shift-Add	Area/interconnect reduction	149.5	-57%
Retimed	Voltage reduction (5V → 1.5V)	13.5	-91%
	Overhead of parallel hardware	21.0	+56%
Pipelined	Voltage reduction (1.5V → 1.25V)	14.6	-30%
	Architectural factors: Distributed architecture Partitioned memories No control/muxes necessary	9.4	-36%
	Increased correlations (locality)	7.0	-26%

Table 5-4 : Impact of various optimizations on power

attributed to a distributed architecture and improved signal correlations - the other 46% is due to voltage reduction. Note that while SPADE is able to model these effects, traditional UWN-based estimators are not.

Unfortunately, while the pipelined design consumes the least power, it is also the largest design, occupying 115.4 mm^2 of silicon real-estate. Selection of which design version to actually implement would depend on how much area the designer is willing to trade for lower power. The SPADE power and area estimates can aid the designer in this selection process. While the pipelined example at 7 mW consumes less power than the 21 mW retimed design, it is at the cost of a 4.7x area increase. As a result, the retimed example, which is still 17x lower power than the initial solution and requires only 24.4 mm^2 is probably a more desirable solution.

To verify that SPADE provided accurate area and power estimates, this version of the filter has been synthesized (down to layout), extracted, and simulated. The chip plot is shown in Figure 5-17. The predicted area of 24.4 mm^2 is within 9% of the actual 26.8 mm^2 area. A comparison of the

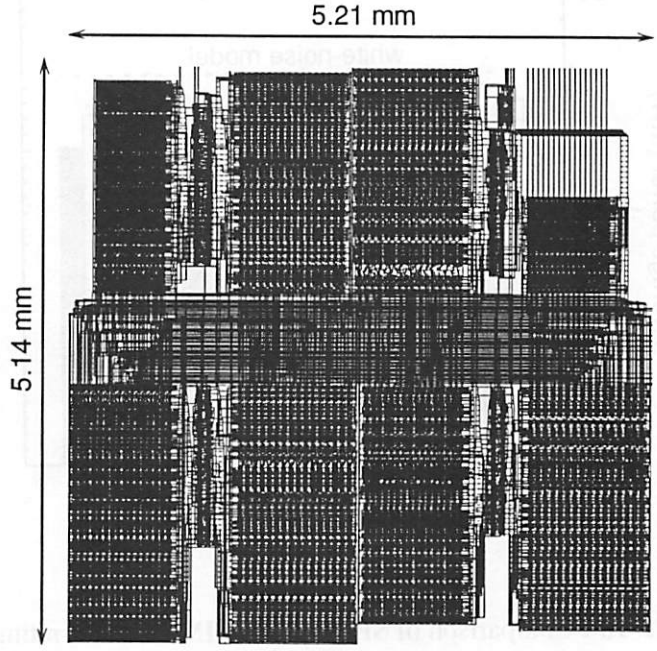


Figure 5-17 : Layout of retimed QMF filter design

SPADE power predictions to switch-level simulation using IRSIM-CAP is given in Figure 5-18. The figure shows the average power consumed by the chip for data streams corresponding to increasing input signal powers. SPADE's estimates are within 5% to 14% of IRSIM-CAP for all data streams. Estimates based on the white-noise model are also included. For some cases, the white-noise estimates are accurate (within 11% of switch-level); however, unlike the SPADE results, the white-noise estimates do not track signal statistics and, therefore, err by as much as 71% for some of the data streams.

Since SPADE can provide such accurate results, it is reasonable to use architecture-level predictions to make high-level design decisions, rather than mapping candidate architectures down to the transistor level to get power estimates. In the case of the QMF filter the savings in design time would be significant. The three filter versions described here were synthesized and analyzed with SPADE in 5 minutes on a Sun SPARCstation 10. In contrast, generating and simulating the

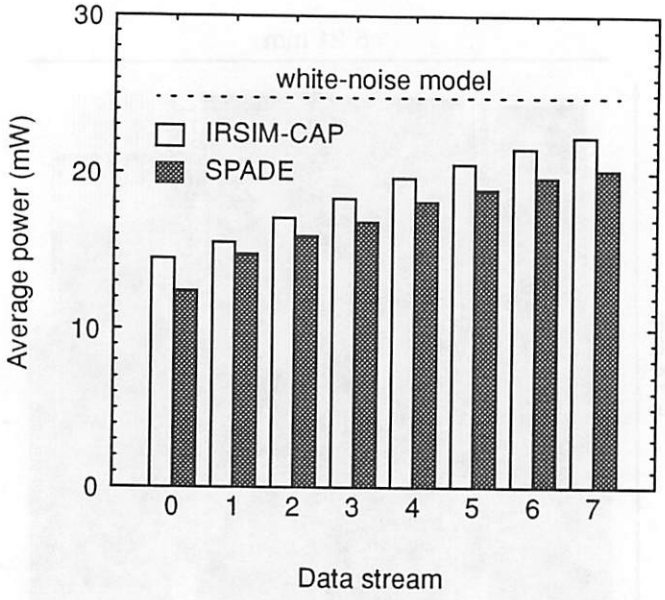


Figure 5-18 : Comparison of SPADE to IRSIM-CAP for retimed QMF

retimed version at the layout level took 3.2 hours. Laying out and analyzing all four designs using low-level power analysis tools would have required 13 hours or more.

In summary, SPADE can be used to allow a thorough exploration of the ATP design space in a reasonable amount of time. It exhibits good absolute accuracy, but more importantly it correctly tracks the relative impact of complexity *and activity* on power consumption. This sets SPADE apart from previous attempts at architectural power analysis.

5.3.3 Development Example

SPADE is useful not only for analysis and optimization of power consumption, but also for the development of new power reduction techniques. Specifically, if a designer comes up with some novel low-power design strategy, he can use SPADE to rapidly and thoroughly test the concept on many examples. In this way, he can discover how effective the technique actually is, as well as what its limitations are.

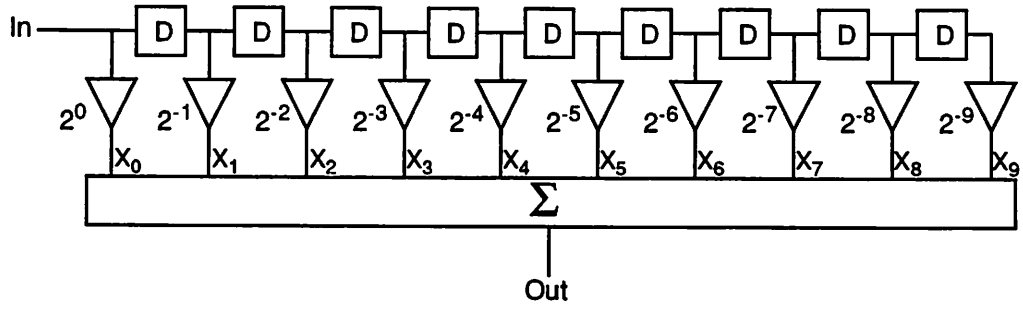


Figure 5-19 : FIR filter for demonstrating activity minimization and postponement

To illustrate this point, this section will walk through the evaluation of two low-power design techniques. In Chapter 2, these techniques were referred to as activity minimization and activity postponement. The idea behind *activity minimization* is to minimize the sign transition activity on buses and modules by maximizing the correlation of the signals multiplexed onto these entities. Since samples of a signal that are close together in time often display the highest correlation, assignment of signals based on temporal locality is one specific method of activity minimization.

Activity postponement refers to the notion that when signals are being combined in a computational tree, the highest activity signals should be introduced as late as possible. This helps the activity for the majority of the tree to remain low. One useful measure of activity in data streams is the variance (actually, $\mu+3\sigma$) of the samples. As the breakpoint formulas of Chapter 4 indicate, the larger the mean or variance, the more high activity UWN bits are present. Lower variance signals, on the other hand, have more sign bits, which tend to have lower activity (for positively correlated signals). Therefore, one strategy for activity postponement would be to introduce the data with the lowest mean and variance into the computational tree first and to introduce the higher variance (and activity) signals later.

SPADE can be used to study the efficacy of these two low-power techniques. For the purposes of this study, consider the FIR filter structure of Figure 5-19. The filter consists of a delay line with

10 taps. The taps weight the delayed input signals by coefficients ranging from 2^0 to 2^{-9}. The outputs of the taps are then summed by an adder tree to produce the filter output. For easy reference, the tap outputs will be indexed by exponent. In other words, the output of the 2^{-i} tap will be referred to as X_i. Notice that for a zero-mean input with a positive temporal correlation (a segment of classical music was used for this example), the most highly correlated signals are those with the closest indices (e.g. X_{i-1} and X_i). Also notice that the variance of the taps is largest for tap 0 and gets smaller as the tap index increases. This filter provides a good example for testing our low-power techniques since we can alter the correlation and variance of our adder tree simply by changing the order in which we combine the various taps.

With this background, we begin the study by setting a "worst-case" reference point. In particular, we try to combine the taps in a manner that minimizes the correlation of the signals being fed to the adders. One way to destroy correlation is to time-multiplex signals onto the same hardware. So for our first architecture we choose a single adder and register and perform accumulation-based summing. Now we must assign signals to the adder inputs. First, consider input one of the adder. We would like the signal stream applied to this input to have as little correlation as possible. This can be achieved by keeping the tap indices of successive signals far apart: for example, X_0, X_4, X_8, X_2, X_6, etc. A similar arrangement can be used for input two; however, in choosing input two we must also consider how to keep the variance of additions high throughout the sequence. This can be achieved by pairing a low-variance signal on input one with a high variance signals on input two. This overall assignment strategy is summarized in Table 5-5. The table also gives the SPADE estimates for the average capacitance switched per sample while performing each addition. These numbers will be used as a worst-case reference in order to quantify how much power can be saved by the proposed activity minimization and postponement techniques.

We now consider improving the correlation of the data by assigning signals so as to maximize temporal locality (i.e. correlation). The maximum correlation can be achieved by scheduling

Time	Input 1	Input 2	Output	Capacitance (pF)
0	X_0	X_9	X_{09}	3.57
1	X_4	X_5	X_{45}	3.23
2	X_8	X_1	X_{18}	3.10
3	X_2	X_7	X_{27}	2.15
4	X_6	X_3	X_{36}	1.90
5	X_{09}	X_{18}	X_{0189}	3.40
6	X_{45}	X_{27}	X_{2457}	2.98
7	X_{0189}	X_{36}	X_{013689}	2.81
8	X_{2457}	X_{013689}	$X_{0123456789}$	2.23
Total:				25.4

Table 5-5 : Version 1: low correlation, high variance (reference)

signals separated by the fewest number of taps to successive time slots - for instance, input one could be assigned X_0, X_1, X_2, X_3, X_4, etc. We must also schedule signals for adder input two. In order to test the effect of activity minimization independent of activity postponement, we do not attempt to minimize variances for this design version. This can be achieved, while still maximizing correlations, by scheduling input two with the data stream: X_9, X_8, X_7, X_6, X_5, etc. Table 5-6 contains the complete addition sequence along with the capacitance estimates from SPADE. The estimates reveal that this first application of activity minimization has yielded a significant 21% reduction in capacitance and, therefore, power.

The correlation can be improved still further by going from using a single time-multiplexed adder to having a dedicated adder for each addition. This ensures that the inputs to each adder at each successive time slot are always maximally correlated. Once again, to test the effects of activity minimization alone, input two is assigned so as to maximize adder variances. Table 5-7

Time	Input 1	Input 2	Output	Capacitance (pF)
0	X_0	X_9	X_{09}	3.40
1	X_1	X_8	X_{18}	1.84
2	X_2	X_7	X_{27}	1.75
3	X_3	X_6	X_{36}	1.76
4	X_4	X_5	X_{45}	1.78
5	X_{45}	X_{36}	X_{3456}	1.65
6	X_{27}	X_{18}	X_{1278}	2.65
7	X_{1278}	X_{09}	X_{012789}	2.45
8	X_{3456}	X_{012789}	$X_{0123456789}$	2.85
Total:				20.1

Table 5-6 : Version 2: higher correlation, high variance

gives the assignment, as well as the SPADE results. Remember, each addition is now performed on its own dedicated adder. SPADE indicates that the power of this version is 10% lower than the previous version, with an overall reduction due to activity minimization of 29%. The use of dedicated hardware can be thought of as trading area for power - not through voltage reduction, but rather through activity reduction.

A final version of the adder tree can be synthesized to test the impact of activity postponement. For this version, we preserve the maximum correlation by still using nine dedicated adders; however, we order the adder tree such that the lowest variance (and activity) signals are merged first, while the highest variance signals are postponed as long as possible. This results in the adder tree of Figure 5-20 and the SPADE estimates of Table 5-8. Thus, activity postponement offers an additional power reduction of 8.3%.

In summary, SPADE has allowed us to analyze the impact of two low-power techniques.

Adder	Input 1	Input 2	Output	Capacitance (pF)
0	X_0	X_9	X_{09}	1.86
1	X_1	X_8	X_{18}	1.75
2	X_2	X_7	X_{27}	1.74
3	X_3	X_6	X_{36}	1.67
4	X_4	X_5	X_{45}	1.56
5	X_{09}	X_{18}	X_{0189}	2.26
6	X_{27}	X_{45}	X_{2457}	2.21
7	X_{0189}	X_{36}	X_{013689}	2.43
8	X_{013689}	X_{2457}	$X_{0123456789}$	2.48
Total:				18.0

Table 5-7 : Version 3: highest correlation, high variance

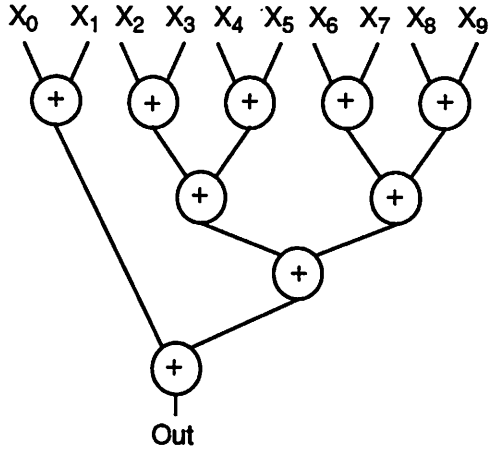


Figure 5-20 : Minimum activity adder tree (version 4)

Adder	Input 1	Input 2	Output	Capacitance (pF)
0	X_8	X_9	X_{89}	1.17
1	X_6	X_7	X_{67}	1.28
2	X_{67}	X_{89}	X_{6789}	1.53
3	X_4	X_5	X_{45}	1.58
4	X_2	X_3	X_{23}	1.89
5	X_{23}	X_{45}	X_{2345}	2.26
6	X_{2345}	X_{6789}	$X_{23456789}$	1.90
7	X_0	X_1	X_{01}	2.26
8	X_{01}	$X_{23456789}$	$X_{0123456789}$	2.64
Total:				16.5

Table 5-8 : Version 4: highest correlation, lowest variance

Version	Cap (pF)	Incremental Δ%	Overall Δ%
1	25.4	-	-
2	20.1	-21%	-21%
3	18.0	-10%	-29%
4	16.5	-8.3%	-35%

Table 5-9 : Summary of results for activity minimization and postponement

Specifically, through some simple tests we showed that for this example, activity minimization and postponement together can yield an overall 35% reduction in power consumption (see results summary of Table 5-9). SPADE allowed us to rapidly quantify the types of improvements that can

Version	DBT (pF)	UWN (pF)	UWN Error
1	25.4	39.4	55%
2	20.1	39.4	96%
3	18.0	39.4	119%
4	16.5	39.4	139%

Table 5-10 : Comparison of DBT and UWN results

be gained by these two low-power design techniques. It is interesting to note that architecture-level power analysis tools that employ a UWN model of activity could not have been used to evaluate these strategies. Table 5-10 compares the results of the UWN model to those of the DBT model (SPADE) for each design version discussed above. The UWN model is oblivious to the power trade-offs offered by the different versions, predicting the same power consumption for each. The error exhibited by the UWN technique varies from a low of 55%, for this example, to a high of 139%. This provides yet another motivation for the activity-sensitive power models of Chapter 4 and for tools that implement these models such as SPADE.

5.4 Chapter Summary

This chapter has described SPADE, a tool for architectural power analysis. The tool has been integrated into the HYPER high-level synthesis system which is targeted towards datapath-intensive DSP applications. SPADE fully implements the DBT model described in Chapter 4, providing extensions to allow for a more accurate handling of time-multiplexed hardware resources. SPADE also takes advantage of the constrained application domain and architectural style of HYPER to simplify estimation of controller-related power consumption. Specifically, rather than the ABC model, SPADE employs a benchmarking approach where controller power

consumption is modeled statistically based on results from previous designs. In addition, SPADE takes advantage of HYPER's fixed floorplanning strategy to simplify and improve the accuracy of its interconnect estimates.

The results achieved by SPADE for several examples were presented in this chapter. These results show that it is possible for architecture-level power estimates to come well within 20% of switch-level estimates from extracted layouts. They also gave an indication of how high-level power estimation can be effectively applied to the task of power optimization. Some of the optimizations discussed, such as those dealing with activity postponement and minimization, could not be analyzed by any other existing architectural power analysis tool.

The main limitation of SPADE is that it resides in an environment which is best suited for DSP applications and architectures. While this is an important class of design problems, it prevents SPADE from having truly widespread applicability. Furthermore, SPADE uses behavioral rather than functional simulation to gather activity statistics and, therefore, cannot model certain timing effects that sometimes influence power consumption. The next chapter will present a general Stochastic Power Analyzer (SPA) that does not suffer from these constraints. SPA fully implements all of the power analysis models of Chapter 4 and can, therefore, be used to provide a more complete verification of those techniques. The topic of SPADE and HYPER, however, will be revisited in Chapter 7, which describes a complete high-level methodology for CAD-assisted low-power design.

CHAPTER 6

SPA: A Stochastic Power Analyzer

The last chapter discussed SPADE, an implementation of the power analysis techniques presented in Chapter 4 targeted towards datapath-intensive DSP applications. While SPADE is useful for certain situations, it lacks the generality that would allow it to be useful in mainstream designs. This is not a weakness of the power analysis models discussed in Chapter 4; rather it is an artifact of the specific implementation and environment of SPADE. This chapter addresses these concerns, presenting a more generic architectural power analysis tool called SPA. This tool has several key advantages over SPADE. Most importantly, the user is not constrained to any fixed architectural model, but instead is free to specify an arbitrary architecture in the form of an RTL netlist. This allows SPA to handle designs ranging from DSP ASIC's to general-purpose microprocessors.

The description of SPA will be broken into several sections. First, Section 6.1 describes the SPA environment, including the required inputs, the sequence of steps performed during power analysis, and the outputs produced. Next, Section 6.2 compares and contrasts the power analysis models used by SPA to those described in Chapter 4. Finally, Section 6.3 presents results

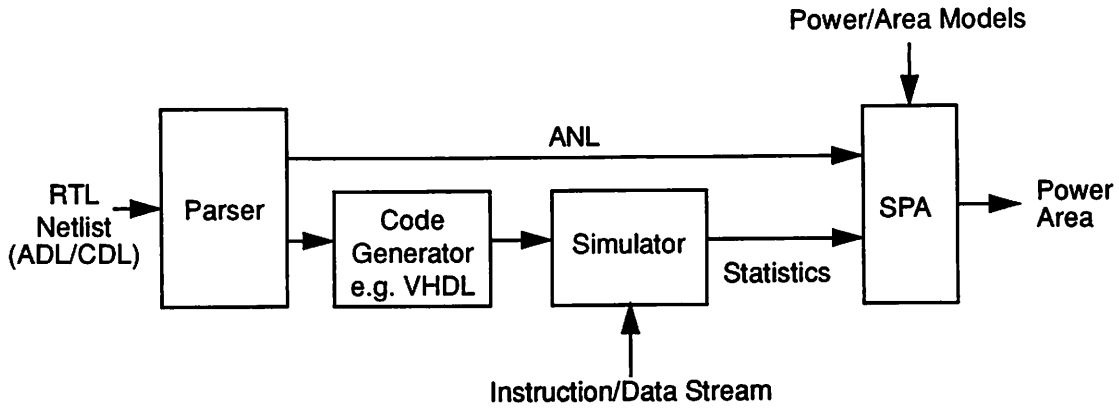


Figure 6-1 : SPA power analysis environment

demonstrating SPA's speed, accuracy, and flexibility.

6.1 Environment

Power analysis under SPA consists of several phases. Figure 6-1 depicts the entire process flow including required inputs and resulting outputs. This section will briefly describe each step in the process, highlighting key points for later, more detailed discussions.

The primary input to SPA is an RT-level description of the architecture under consideration. Conceptually, this description consists of the information displayed in Figure 6-2. In particular, it contains a (possibly hierarchical) description of blocks used in the chip, as well as the interconnect network joining these blocks. Currently, SPA uses a textual architectural description language (ADL) for this purpose; however, the same information could be provided by a graphical schematic capture interface. We chose not to use VHDL as the primary input because of the overhead associated with writing a VHDL description of an architecture.

SPA also must be told how the blocks interact in order to produce activity estimates. This information is embodied in the control path of the chip. Using a control description language

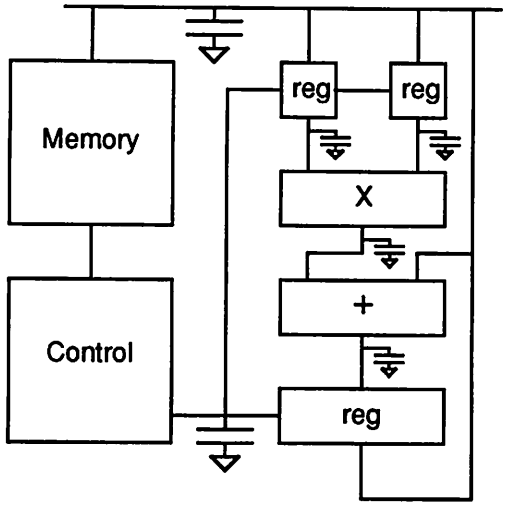


Figure 6-2 : Graphical representation of RTL netlist

(CDL), the control flow of the design can be described as a set of control tables. For each control module, these tables specify how the next state and outputs of the module relate to the present state and inputs. In order to maintain a relatively high level of abstraction, CDL allows the user to specify control signals and states as enumerated (symbolic) types rather than bit vectors. For example, instructions to an ALU can take the form of "ADD", "SHIFT", or "CMP" rather than some obscure binary codes.

With the structure and behavior of the architecture fully defined, the first task is to translate these description into a set of internal data structures. Two parsers, one for ADL and one for CDL, perform this task, generating a graph-based representation of the architectural netlist called ANL. Both parsers were written by A. Abnous, a fellow U. C. Berkeley researcher.

The next step in the process is to derive the activity statistics for the design. This is accomplished by a functional simulation of the architecture. Many RT-level simulators would be suitable for this task; currently, the activity statistics are based on VHDL simulation results since this is a widely accepted and available standard. A code generator translates ANL to VHDL, and the VHDL System Simulator™ (provided by Synopsys) is invoked. If the chip requires data or

instruction inputs, these must be supplied by the user. In this way, the user is able to characterize the power consumption of an architecture under various instructions and operating conditions. The result of the functional simulation is a set of activity parameters for each module and bus in the architecture.

This information is then fed to the core power analysis tool, SPA, along with the parsed ANL description. SPA then steps through each datapath, memory, control, and interconnect entity in the design, applying the appropriate power analysis models using the simulated activity parameters. Finally, SPA dumps its results, categorizing area and power consumption in terms of the four classes of components (see Figure 6-3). This points the designer to the most power-intensive portions of the implementation and provides useful guidance for further optimizations.

In summary, the SPA environment consists of several tools working in unison to realize the process flow of Figure 6-1. The user specifies the architectural structure and macro-behavior in ADL and CDL, and also specifies any data and instruction inputs to the chip. This information is then used to spawn a VHDL simulation that captures the activity statistics of the design. Combining this activity information with the ANL description, the power models of Chapter 4 are applied to produce estimates of chip power consumption.

6.2 Architectural Power Analysis using SPA

This section describes the details of how SPA performs architectural power and area analysis. First, Section 6.2.1 describes how the required DBT and ABC activity statistics are derived. Then, Sections 6.2.2-6.2.4 give those details regarding datapath, memory, control, and interconnect analysis in SPA that have not already been presented in Chapter 4.

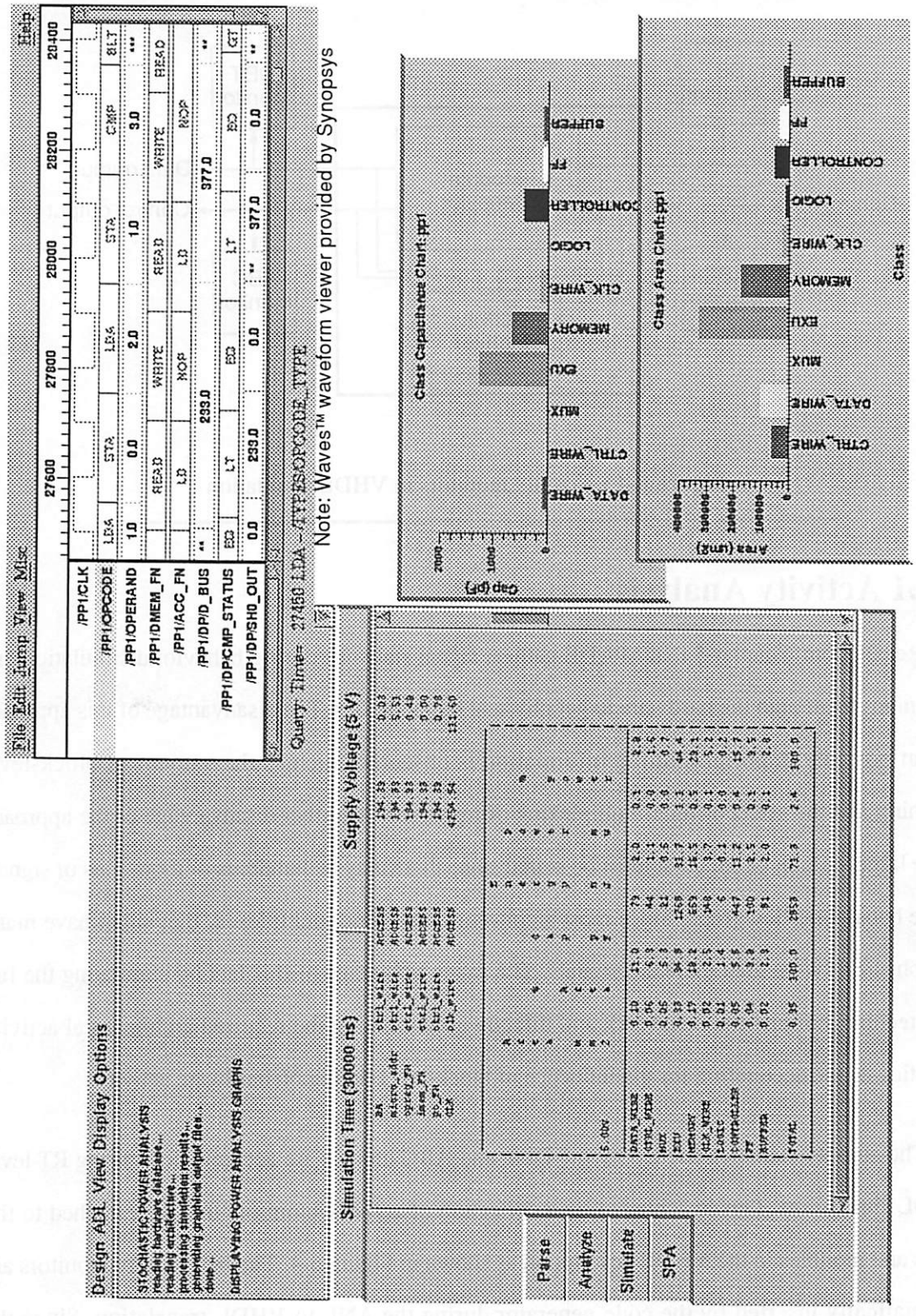


Figure 6-3 : Graphical user interface for SPA

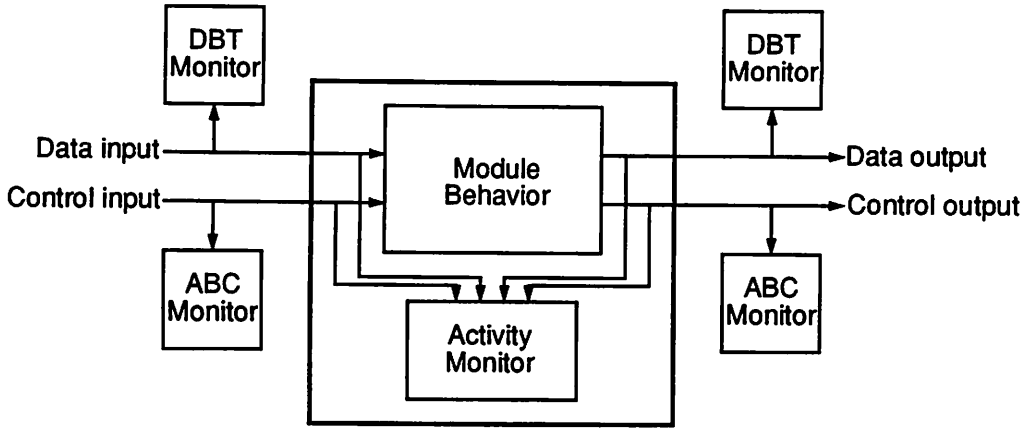


Figure 6-4 : Activity monitors in VHDL simulation

6.2.1 Activity Analysis

Recall from Chapter 5 that SPADE gathers signal statistics during behavioral simulation and then maps them onto the hardware to get physical bus activities. The disadvantage of this approach is that possibly important timing information is obscured, such as the amount of block-level glitching and the effect of time-multiplexing of resources. A further disadvantage of the approach is the large amount of data that must be stored since there may be hundreds or thousands of signals in the behavioral flowgraph. Since most architectures are time-multiplexed, they often have many less physical buses than there are signals. SPA takes advantage of this fact by simulating the full architecture (not just the behavior), and directly accumulating the required architectural activity statistics, while accounting for the influence of block-level timing on hardware activity.

The activity statistics needed by the DBT and ABC models are accumulated during RT-level VHDL simulation. This is achieved by a collection of activity monitors that are attached to the buses and modules in the VHDL description as shown in Figure 6-4. The bus activity monitors are automatically inserted by the code generator during the ANL to VHDL translation. Since the module instances are just pointers to masters in some hardware library, the module activity

monitors can be placed inside the VHDL models of each cell when it is added to the hardware library.

The DBT variables accumulated for buses and one-input modules with input signal X are as follows:

DTC	= Count of transitions on data inputs	
ITC	= Count of transitions on instruction (control) inputs	
EX	= Average or expected value of X	
EXX	= Expected value of X^2	
EX_0X_t	= Expected value of product of successive samples of X	
$STC[0:3]$	= Counts for sign transition occurrences: ++, +−, −+, −−	(EQ 146)

A sample of the VHDL code that might be used to accumulate these variables during simulation is shown in Figure 6-5. From these variables all desired activity parameters including mean, variance, correlation, and sign transition probability can be determined. For example,

$$\mu = EX \tag{EQ 147}$$

$$\sigma^2 = EXX - (EX)^2 \tag{EQ 148}$$

$$\rho = \frac{EX_0X_t - (EX)^2}{\sigma^2} \tag{EQ 149}$$

Similar statistics are gathered for two-input modules with the addition of statistics such as joint input/output correlations and sign transition probabilities. DBT statistics accumulation also includes monitoring the control (or instruction) inputs to datapath and memory entities. The appropriate activity parameters are the number of instruction transitions that occur (ITC) and the destination instruction of each transition.

In addition to DBT activity parameters, several activity parameters must be accumulated for ABC buses and modules. The monitoring strategy is identical to the DBT case (Figure 6-4). Since the encoding of control signals and states is not necessarily known at this level of abstraction, these monitors simply count symbol transitions on control buses and estimate the following parameters assuming a random encoding of control bits:

```
procedure Update_DBT_Stats1(INST_TRAN: boolean;
                            Xt: real; stats: inout SPA_Stats1) is
    variable T: real;
    variable STC_INDEX: integer;
begin
    -- Bail out if data is tri-stated
    if Xt=TRI_STATED then return; end if;

    -- Update current I/O values
    stats.Xt := Xt;

    -- Accumulate previous/current-sample statistics
    T := stats.DTC;
    if (not INST_TRAN and T>=1.0) then
        stats.EX0Xt := stats.EX0Xt * (T-1.0)/T + stats.X0*stats.Xt/T;
    end if;

    -- Accumulate current-sample statistics
    T := stats.DTC + stats.ITC;
    stats.EX  := stats.EX  * T/(T+1.0) +          stats.Xt/(T+1.0);
    stats.EXX := stats.EXX * T/(T+1.0) + stats.Xt*stats.Xt/(T+1.0);

    -- Ascertain current sign values
    if stats.Xt>=0.0 then stats.XSt:=0; else stats.XSt:=1; end if;

    -- Accumulate instruction and sign transition counts
    if INST_TRAN then
        -- Increment instruction transition count
        stats.ITC := stats.ITC + 1.0;
    else
        -- Compute index into sign transition count array
        -- and increment sign transition count
        STC_INDEX := stats.XS0*2 + stats.XSt*1;
        stats.STC(STC_INDEX) := stats.STC(STC_INDEX) + 1;

        -- Increment data transition count
        stats.DTC := stats.DTC + 1.0;
    end if;

    -- Update previous sample values
    stats.X0 := stats.Xt; stats.XS0 := stats.XSt;
end;
```

Figure 6-5 : VHDL code for gathering the DBT activity statistics of one signal

```
procedure Update_ABC_Stats(Xt: integer; stats: inout ABC_Stats) is
begin
    -- Update current control value
    stats.Xt := Xt;

    -- Update the bit transition count.  NOTE: For a random encoding,
    -- each transition will cause half of the bits in the control
    -- word to transition.
    if (stats.Xt /= stats.X0) then stats.BTC := stats.BTC + 0.5; end if;

    -- Update the bit signal count - that is, the number of 1 bits.
    -- NOTE: For a random encoding, half of the bits in the control
    -- word will be 1.
    stats.BSC := stats.BSC + 0.5;

    -- Increment control transition count
    stats.CTC := stats.CTC + 1.0;

    -- Update previous control values
    stats.X0 := stats.Xt;
end;
```

Figure 6-6 : VHDL code for gathering the ABC activity statistics of one signal

CTC	= Count of transitions on control word
BTC	= Count of transitions on control bits (as fraction of total word length) Used to calculate bit transition probability, α
BSC	= Count of 1 control bits (as fraction of total word length) Used to calculate bit signal probability, *P* (EQ 150)

A sample of the VHDL code that might be used to gather these statistics during simulation is shown in Figure 6-6. From these variables all desired activity parameters (α, *P*) can be determined:

$$\alpha = \frac{BTC}{CTC} \quad \textbf{(EQ 151)}$$

$$P = \frac{BSC}{CTC} \quad \textbf{(EQ 152)}$$

At the end of the functional VHDL simulation, the DBT and ABC parameters for each bus and module in the architecture are dumped into a statistics file. This file can then be accessed by SPA during power analysis of the individual design entities. This analysis process is the subject of the next several sections.

```
(CAP          (* (/ N N_TOT) (+ C0 (* C1 W) (* C2 N_TOT) (* C3 W N_TOT))))
(CAP-COEFFS (LOOKUP FUNCTION
               ((READ (LOOKUP TRAN_ID
                         ((UU ( (87 51 35 8) ))
                          (SS ( (88 43 17 3)   (88 43 51 3)
                                (89 43 58 23)  (88 43 17 3) )))))
                (WRITE (LOOKUP TRAN_ID
                         ((UU ( (99 30 65 10) ))
                          (SS ( (104 43 38 3)  (104 43 92 32)
                                (104 43 92 3)  (104 43 38 3) ))))))))
(BITSLICE-WIDTH  64)
(BITSLICE-LENGTH (* 62 W))
(AREA            (* 64 62 W))
.
.
.
```

Figure 6-7 : DBT capacitance/area models for register file in hardware database

6.2.2 Datapath and Memory

A side-effect of the behavioral simulation and structural mapping approach taken by SPADE is that the variation of statistics from clock cycle to clock cycle due to time-multiplexing of resources is known. In contrast, SPA simulates the architecture directly and, therefore, the statistics it accumulates are aggregate or average bus statistics over the entire simulation lifetime. This means that, unlike SPADE, SPA does not have to worry about DBT breakpoints changing over time. Instead, it can apply the DBT models directly as described in Chapter 4.

The power analysis of each module requires three inputs. The first is a set of complexity parameters for the module, which are available from the structural description. The second is the set of activity parameters for the module, which can be read from the statistics file produced during VHDL simulation. The third is the module capacitance model and its associated table of coefficients, which can be read from a hardware database indexed by the type of module. For instance, sample entries for a register file are shown in Figure 6-7. The output of this stage of power analysis is a list of power consumptions for each memory, datapath module, and bus.

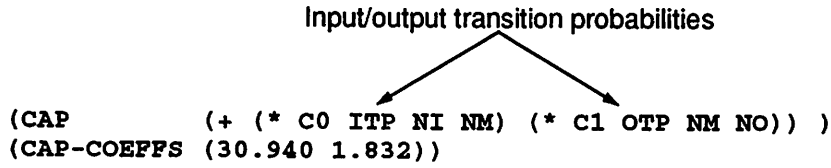


Figure 6-8 : ABC capacitance model for standard cell controller

6.2.3 Control Path

Since SPADE operates under the assumption of a fixed application domain and architectural model, it was possible to obtain reasonable estimates of controller power consumption based on a statistical model fit to a number of benchmark examples. SPA, however, must cover a wider range of implementation possibilities; therefore, it is less suited to a benchmarking approach. Instead, SPA employs the full ABC model of Chapter 4 for the task of estimating the power consumed by control logic and signals.

Each type of controller module (e.g. ROM, PLA, standard cell) has an entry in the hardware database. These entries include capacitive coefficient tables and capacitance models similar to the datapath modules. As an example, the capacitance entries for a standard cell control module are shown in Figure 6-8. The power of each control module is estimated by plugging in the appropriate complexity and activity parameters. The activity parameters are available in the statistics file produced by the VHDL simulation, and the complexity parameters are derived using the strategy presented in Section 4.5.1. Given the activity and complexity parameters, controller power analysis in SPA proceeds precisely as described in Chapter 4.

6.2.4 Interconnect

Interconnect analysis under SPA proceeds as in Chapter 4 with few variations. The structural ADL provides the hierarchical description used to estimate the length of an average interconnect

wire within each subsection of the design. This provides the physical capacitance of the wires forming the interconnect network. The DBT and ABC statistics derived using the simulation strategy of Section 6.2.1 then provide the activities associated with these wires. The combination of these factors results in an effective capacitance for the control and data buses throughout the design, which can be used to estimate the average power consumption associated with driving the interconnect and clock distribution networks.

6.3 Results

In this section we present results gathered using the SPA power analysis tool. These results demonstrate the accuracy offered by SPA, as well as its benefits over existing power analysis tools. To some extent, however, the effectiveness of the power analysis techniques developed in this thesis was already demonstrated by the results presented for SPADE in Chapter 5. Therefore, the primary focus of this section will be to reveal the specific benefits that SPA offers over SPADE - namely, flexibility and generality.

This section is divided into three examples. The examples demonstrate how SPA can be used to analyze the power of widely differing applications and architectural styles. The first example is a Newton-Raphson iterative divider, which will show SPA's applicability to datapath-intensive applications. The second example undertakes the analysis of a control-intensive design - specifically, a finite-state machine that implements the global control function for a speech recognition front-end. The final case study is a programmable instruction set processor that demonstrates SPA's ability to handle a real-world example containing significant datapath and control components. As the microprocessor employs on-chip instruction and data memories, this example will also serve as a verification of SPA's efficacy for memory-intensive designs and will show how SPA can be used to evaluate the impact of different instruction and data streams on chip power consumption.

6.3.1 Datapath Intensive: Newton-Raphson Divider

The first example we consider is that of a hardware divider based on the Newton-Raphson iteration. Such a divider might be used to implement floating-point division in a microprocessor. For instance, Texas Instruments employed a similar iterative division algorithm in their TI 8847 processor.

Using Newton's technique we can find the zero of a function $f(x)$ by performing the following iteration [Pat90]:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \text{(EQ 153)}$$

where x_i asymptotically approaches the zero of the function as i increases. In order to use this iteration to perform a division we must reinterpret the division as finding the zero of a function. If we let $f(x) = 1/x - b$ then the zero of this function occurs at $1/b$. If we multiply this zero by a, then we have effectively computed a divided by b. Substituting this function for $f(x)$ in (EQ 153) we arrive at the following iteration:

$$x_{i+1} = x_i - \frac{1/x_i - b}{-1/x_i^2} = x_i(2 - x_i b) \quad \text{(EQ 154)}$$

In summary, the Newton algorithm for implementing iterative division is as follows [Pat90]:

- Set $x_0 \approx 1/b$ using table lookup.
- Iteratively compute $x_{i+1} = x_i(2 - x_i b)$ until convergence (n steps).
- Compute $a/b = ax_n$.

The number of iterations, n, required depends on the number of bits of precision desired. Since each iteration step approximately doubles the number of correct bits, and since the inverse table we will employ is correct to about one bit, four iterations are enough to compute a quotient to 16 bits of accuracy.

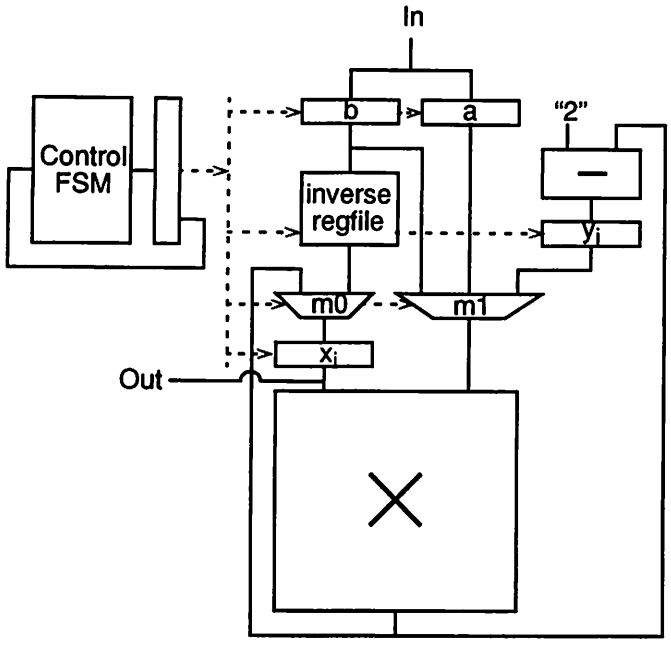


Figure 6-9 : Architecture for Newton-Raphson divider

Figure 6-9 shows an architecture which can implement the above algorithm. It contains an eight-word register file that holds the 1/*b* lookup table, as well as several other registers that store the dividend, the divisor, and intermediate results. The dominant feature of the datapath is a 16x16 carry-save array multiplier.

Excerpts from the ADL and CDL descriptions for this design are shown in Figure 6-10a-b. The high-level symbolic specification style employed by CDL greatly simplifies the task of specifying the divider behavior. Feeding these descriptions into SPA along with a stream of desired division operations results in the power/area predictions of Figure 6-11a-b. The power estimates are given for a voltage supply of 1.5V and a clock rate of 5 MHz. As expected, the datapath (Exu) dominates over the other components.

To verify the accuracy of these predictions, the divider has been implemented down to layout as shown in Figure 6-12. The total area of the final implementation is 4.40 mm^2. SPA's estimate of

```
(cell datapath)
(class DATAPATH)
(library low_power.hwlib.all)

(terminal din  (direction in) (class data)    (type data))
...
(terminal a_fn (direction in) (class control) (type Counter_Fn_Type))
...
                                  Note: word length
(instance areg counter      (parameters 16) (nets din a a_fn CLK))
(instance breg counter      (parameters 16) (nets din b b_fn CLK))
(instance inv_table regfile (parameters 8 16)
   (nets b binv binv_rd binv_wr CLK))
(instance m0mux mux2        (parameters 16) (nets binv p xi m0_sel))
(instance xireg counter     (parameters 16) (nets xi m0 xi_fn CLK))
(instance s sub             (parameters 16) (nets two p yi1))
(instance yireg counter     (parameters 16) (nets yi1 yi yi_fn CLK))
(instance m1mux mux3        (parameters 16) (nets b a yi m1 m1_sel))
...
```

(a) Excerpt from ADL description

```
(inputs  (State_Typestate))
(outputs (Counter_Fn_Typea_fn) ... (State_Type next_state))

(output-table
;  State   | a   b   m0   xi  yi  m1   next
;  -----   | --- --- ---- --- --- ---- ----
  (LDB    ) (NOP LD  SEL0 NOP NOP SEL1 INV)
  (INV    ) (LD  NOP SEL0 LD  NOP SEL1 DIV0a)
  (DIV0a  ) (NOP NOP SEL1 NOP LD  SEL0 DIV0b)
  (DIV0b  ) (NOP NOP SEL1 LD  NOP SEL2 DIV1a)
  (DIV1a  ) (NOP NOP SEL1 NOP LD  SEL0 DIV1b)
  (DIV1b  ) (NOP NOP SEL1 LD  NOP SEL2 DIV2a)
  (DIV2a  ) (NOP NOP SEL1 NOP LD  SEL0 DIV2b)
  (DIV2b  ) (NOP NOP SEL1 LD  NOP SEL2 DIV3a)
  (DIV3a  ) (NOP NOP SEL1 NOP LD  SEL0 DIV3b)
  (DIV3b  ) (NOP NOP SEL1 LD  NOP SEL2 MULT)
  (MULT   ) (NOP NOP SEL1 LD  NOP SEL1 LDB)
)
```

(b) Excerpt from CDL description

Figure 6-10 : Excerpts from description of divider architecture

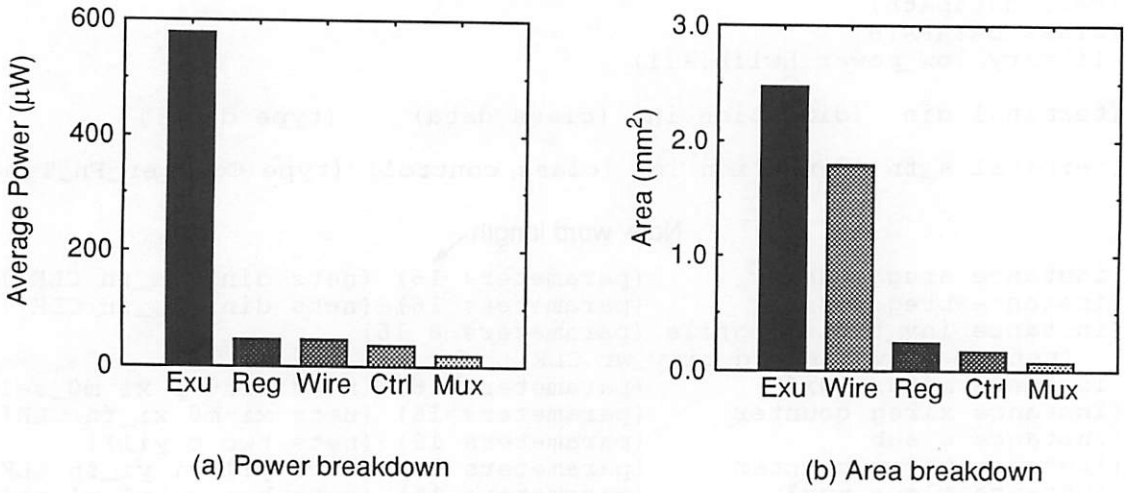


Figure 6-11 : SPA power and area breakdown for divider

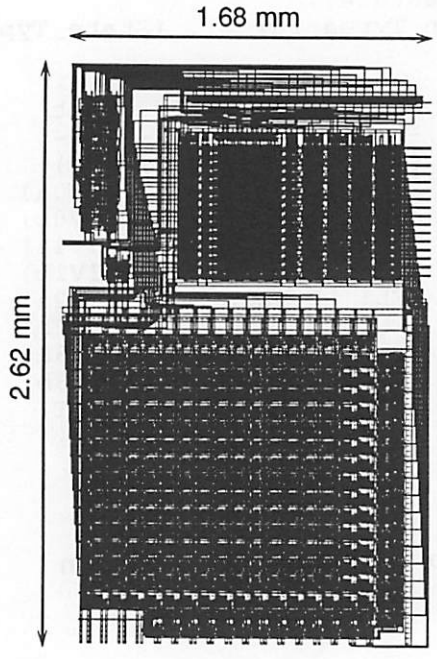


Figure 6-12 : Layout of Newton-Raphson divider in 1.2 μm CMOS technology

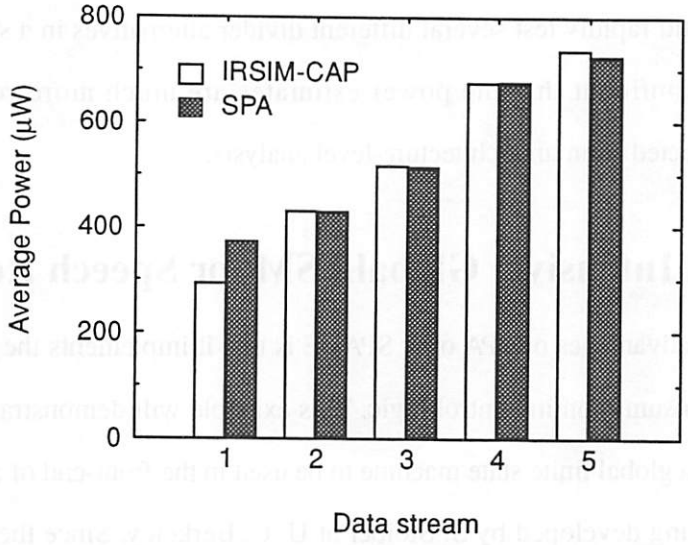


Figure 6-13 : Comparison of SPA to switch-level simulation for various data streams

4.68 mm^2 is within 6% of the actual area. The layout has also been extracted and simulated at the switch level using IRSIM-CAP. Figure 6-13 shows a comparison of the IRSIM-CAP and SPA results for several different data streams each of which leads to a different level of hardware activity. The error in the SPA power estimates varies from as little as 0.2% to at most 27% with an average error of 6%. The variation in power for the different data streams is another strong reminder that data statistics can influence circuit activity (and, therefore, power consumption) to a degree that cannot be ignored. Currently, SPA (and SPADE) are the only architecture-level tools available that can resolve these differences.

Another advantage of SPA can be seen by considering the time required to obtain architecture-level versus switch-level estimates of power. Specifying the design in ADL and CDL required approximately 15 minutes, while the time required to execute SPA including simulation and estimation was only about 20 seconds on a Sun SPARCstation 10. In contrast, even with extensive use of existing hardware libraries, the layout-level implementation of the divider required about 6 hours. Moreover, switch-level simulation took another 10 minutes. Therefore, for this example

SPA offers results within about 25% of switch-level simulation at a time savings of 24x. Using SPA, a designer could rapidly test several different divider alternatives in a short amount of time while remaining confident that the power estimates are much more reliable than would traditionally be expected from an architecture-level analysis.

6.3.2 Control Intensive: Global FSM for Speech Recognizer

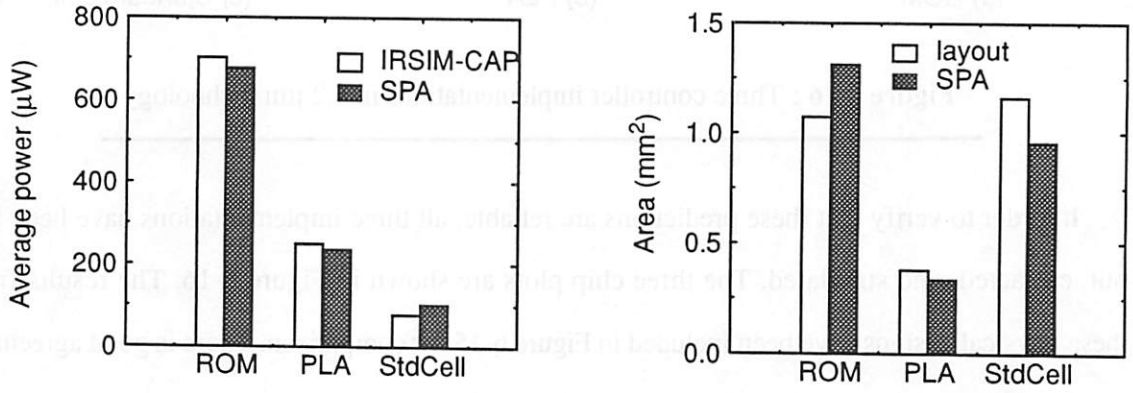
One of the key advantages of SPA over SPADE is that it implements the full ABC model for analyzing power consumption in control logic. This example will demonstrate the use of SPA to aid in the design of a global finite state machine to be used in the front-end of a speech recognition system currently being developed by S. Stoiber at U. C. Berkeley. Since the speech recognition chip-set is targeted at mobile applications, minimizing power consumption is a significant consideration.

One of the most important tasks of a speech recognizer is the extraction of cepstral coefficients. This task, in turn, consists of several sub-tasks such as queuing and windowing the input speech, performing the Levinson-Durbin recursion, and inverting covariance matrices. The complex interaction of these operations and the need for proper sequencing leads to a significant control component within the chip. In fact, the global control module for the design has over 100 states, 10 input signals, and 25 output signals. Therefore, it will serve as a good verification of SPA's ability to handle control-intensive designs.

Prior to invoking SPA, the user must first describe the desired controller behavior in CDL. An excerpt of this description is shown in Figure 6-14. Since CDL allows the use of symbolic, or enumerated, types to specify the value of control fields, generating the description can be performed at a fairly intuitive level. Also, since each field can represent a collection of many bits (e.g. seven state bits become a single state field) the complexity of the overall description is reduced. For example the 10 state and status inputs can be expressed in three fields and the 25 outputs in 9 fields.

```
(output-table
;   rst   pstate    done   |  nstate    gen_ctr  pad     stg_ctr  fft  sym_ctr  sym  bin_ctr  cb
;  (  ---   -------   ----  ) (  ------   -------  -----   -------  ---  -------  ---  -------  ----  )
   (  RST   -         -     ) (  INIT     NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
      ...
   (  RUN   GO_LP     -     ) (  ADD1_LP  NOP      NOP     NOP      GO   NOP      NOP  NOP      NOP   )
   (  RUN   ADD1_LP   -     ) (  ADD2_LP  NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
   (  RUN   ADD2_LP   -     ) (  OE_A_LP  NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
   (  RUN   OE_A_LP   -     ) (  OE_C_LP  NOP      WRITE   NOP      OEA  NOP      NOP  NOP      NOP   )
   (  RUN   OE_C_LP   NONE  ) (  LD_A_LP  NOP      WRITE   NOP      OEC  NOP      NOP  NOP      NOP   )
   (  RUN   OE_C_LP   SYMB  ) (  LD_A_LP  NOP      WRITE   NOP      OEC  NOP      NOP  NOP      NOP   )
   (  RUN   OE_C_LP   BIN   ) (  LD_A_LP  NOP      WRITE   NOP      OEC  NOP      NOP  NOP      NOP   )
   (  RUN   OE_C_LP   STG   ) (  PP5_END  NOP      WRITE   NOP      OEC  NOP      NOP  NOP      NOP   )
   (  RUN   LD_A_LP   -     ) (  LD_C_LP  CNT      READ    NOP      LDA  NOP      NOP  NOP      NOP   )
   (  RUN   LD_C_LP   -     ) (  LD_W_LP  CNT      READ    NOP      LDC  NOP      NOP  NOP      NOP   )
   (  RUN   LD_W_LP   NONE  ) (  PP8_LP   NOP      NOP     CNT      LDW  NOP      NOP  NOP      NOP   )
   (  RUN   LD_W_LP   SYMB  ) (  PP8_LP   NOP      NOP     CNT      LDW  NOP      NOP  NOP      NOP   )
   (  RUN   LD_W_LP   BIN   ) (  PP8_LP   NOP      NOP     CNT      LDW  NOP      NOP  NOP      NOP   )
   (  RUN   LD_W_LP   STG   ) (  PP8_LP   NOP      NOP     NOP      LDW  NOP      NOP  NOP      NOP   )
   (  RUN   PP8_LP    -     ) (  PP9_LP   NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
   (  RUN   PP9_LP    -     ) (  GO_LP    NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
   (  RUN   PP5_END   -     ) (  PP6_END  NOP      NOP     NOP      NOP  NOP      NOP  NOP      NOP   )
      ...
)
```

Figure 6-14 : Excerpt from CDL description of FSM behavior

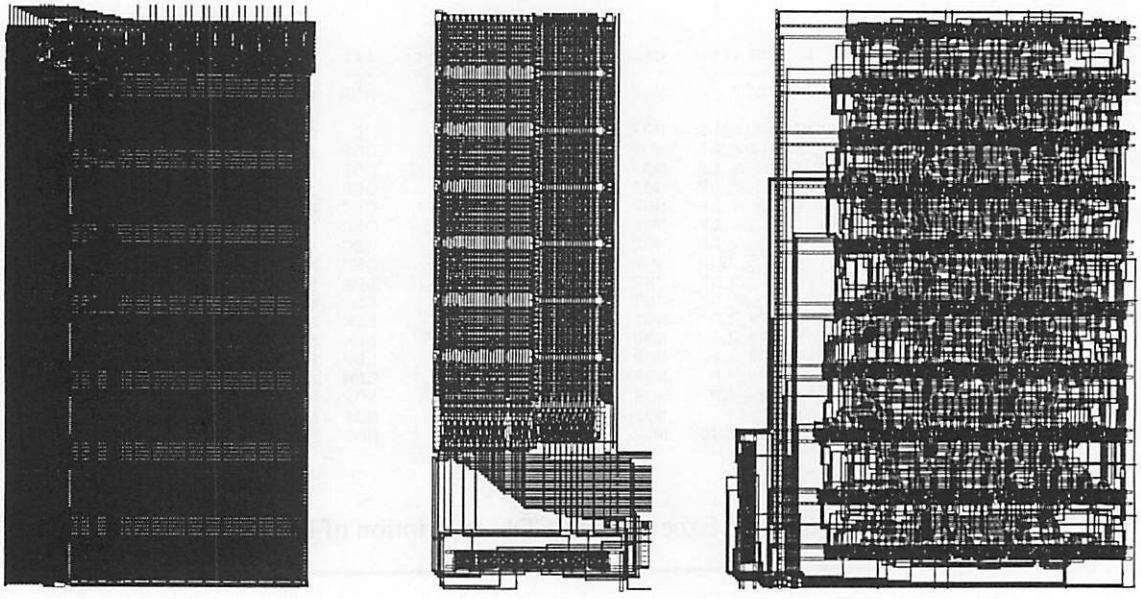


(a) Predicted vs. actual power

(b) Predicted vs. actual area

Figure 6-15 : Power and area results for three possible controller implementations

Given the CDL description, the designer can then use SPA to determine the optimum implementation in terms of area and power. For example, Figure 6-15 shows the estimates provided by SPA for three possible implementation styles: ROM, PLA, and standard cell. The estimates are for a system clock of 3.3 MHz and a supply voltage of 1.5V. The results show the area and average power for each candidate implementation.



(a) ROM (b) PLA (c) Standard cell

Figure 6-16 : Three controller implementations in 1.2 µm technology

In order to verify that these predictions are reliable, all three implementations have been laid out, extracted, and simulated. The three chip plots are shown in Figure 6-16. The results from these physical designs have been included in Figure 6-15 for comparison. There is good agreement between SPA and the actual results as summarized in Table 6-1.

As might be expected, SPA performs best on the more regular structures. For these, the precise binary contents of the control table have less of an effect on power and, therefore, it is possible to make very good predictions at the architecture level. The standard cell case is more difficult since the binary contents of the control table can significantly influence the logic minimization step, as well as the activity of the controller during operation. SPA still provides reasonable results, however, and more importantly it correctly tracks the influence of implementation style on area and power as demonstrated by Figure 6-15. Thus, it can be confidently used to make area-power trade-offs when implementing control-intensive designs.

Implementation	Average power (μW)			Area (mm^2)		
	Actual	SPA	Error	Actual	SPA	Error
ROM	702	677	-3.6%	1.07	1.31	+22%
PLA	249	236	-5.2%	0.378	0.334	-12%
Standard cell	79	102	+29%	1.16	0.956	-18%

Table 6-1 : Summary of results for speech recognition controller

Based on the SPA results the designer could immediately eliminate the ROM-based solution. It consumes by far the most power and is also very large. This is because the ROM does not take advantage of redundancies in the control table (note the large number of repeated entries in Figure 6-14); instead, it performs a full decoding of all input "addresses." The trade-off between the PLA and the standard cell implementation is more subtle. Both implementations use logic minimization to reduce the amount of input decoding required. The higher regularity of the PLA allows it to be significantly smaller than the standard cell design, but the precharged nature of the unit leads to a higher activity (and power consumption) than the static logic standard cell implementation. Based on this information and a set of implementation constraints the designer could select the appropriate candidate.

The time that can be saved by exploring these kinds of trade-offs at the architecture level is significant. For this example, on a Sun SPARCstation 10, SPA analyzed the area and power of the three controller implementations in about 2 minutes. In contrast, generating and extracting the layout for the three controllers took about 5 hours. Simulation required another 30 minutes. Without high-level tools such as SPA, it would not be practical to thoroughly explore all implementation possibilities.

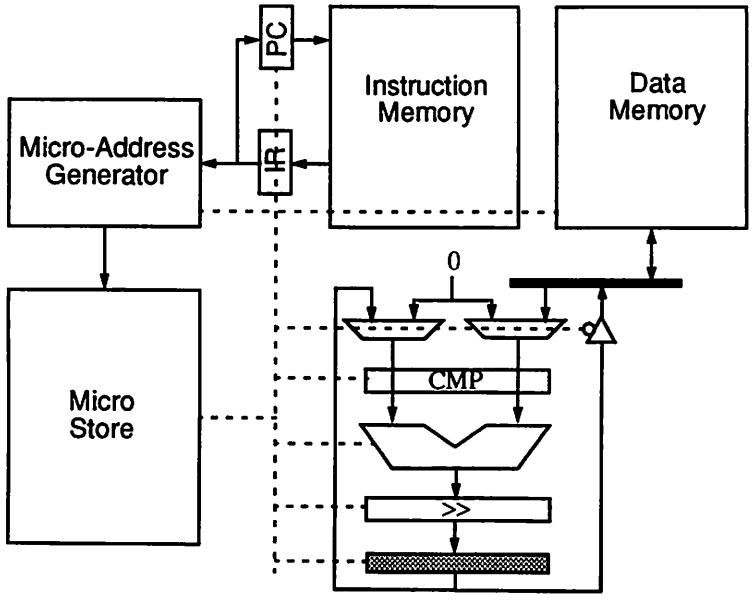


Figure 6-17 : Architecture of microcoded instruction set processor

6.3.3 Memory Intensive: Programmable Microprocessor

The previous two examples have demonstrated SPA's accuracy and flexibility by showing how it can be used to analyze both datapath- and control-intensive applications. Realistic designs, however, often contain both significant datapath and control components simultaneously. In this example, we tackle such a real-world design problem in the form of a programmable instruction set processor. In order to make the example even more realistic we include on-chip storage in the form of instruction and data memories. The results show that these memories represent a significant portion of the total power budget for the design. Therefore, this case study demonstrates SPA's usefulness not only for analyzing datapath and control power consumption, but also for analyzing the often significant memory component of power.

Figure 6-17 depicts the high-level architecture of the simple microcoded processor under consideration. The processor contains two on-chip memories: an instruction memory which holds the program being executed and a data memory which contains program inputs and results. The

data memory feeds a small 16-bit datapath capable of performing add, shift, and compare operations. These basic operations can also be combined to realize more complex operations such as multiplication. The processor control path consists of a program counter that is used to index the instruction memory, the output of which is an eight-bit instruction containing four-bit opcode and operand components. The operand portion of the instruction drives the microsequencer which selects the next section of microcode to be executed by the architecture. This selection process is also a function of the next address (NA) and branch address (BA) fields of the current microword, as well as the datapath status field generated by any compare operations. Finally, the microstore takes this microaddress and generates the next microword to be executed by the architecture. An excerpt from the contents of the microstore is given in Figure 6-18a. Likewise, an excerpt from the ADL description of the architecture is shown in Figure 6-18b. From these descriptions we see that the processor can execute the following nine basic instructions: NOP, LDA, STA, ADD, SHR, CMP, JMP, BLT, and BGE.

Given the ADL and CDL descriptions, SPA is able to make predictions regarding the area and power consumed by the proposed design while executing a given program on a given data stream. SPA breaks down the area and power predictions in terms of the class of hardware unit with which it is associated. Figure 6-19 contains the breakdowns as predicted by SPA for a multiplication program running on the processor with a 1.5V supply and a 10 MHz clock rate. A graph such as this provides the designer with useful information. For instance, in this case SPA reveals that the instruction and data memory accesses account for 47% of the total power consumption. This information could be used by the designer to select an appropriate focus for his optimization efforts.

SPA can also be used to analyze the influence of instruction stream on average power consumption. For example, Figure 6-20 shows the average power while running three different programs: a multiplication program (MULT), a fibonacci sequence generator (FIB), and a circular queue (QUEUE). The power differs between these programs by as much as 38% making a

```
(output-table
;  State | pc    imem  opr  dmem  tbuf0 mux0 mux1 mux2 acc  tbuf1 NA_SEL  NA    BA
;  ----- | ---   ----  ---  ----  ----- ---- ---- ---- ---  ----- ------  ----  ----
   (NOP1)  (INC  READ  LD   NOP   OE    SEL0 SEL0 SEL0 NOP  TRI   SEL_OP  -     -)

   (LDA1)  (NOP  NOP   NOP  READ  OE    SEL0 SEL1 SEL0 NOP  TRI   SEL_NA  LDA2  -)
   (LDA2)  (INC  READ  LD   NOP   OE    SEL0 SEL1 SEL0 LD   TRI   SEL_OP  -     -)

   (STA1)  (INC  READ  LD   WRITE TRI   SEL0 SEL0 SEL0 NOP  OE    SEL_OP  -     -)

   (ADD1)  (NOP  NOP   NOP  READ  OE    SEL1 SEL1 SEL0 NOP  TRI   SEL_NA  ADD2  -)
   (ADD2)  (INC  READ  LD   NOP   OE    SEL1 SEL1 SEL0 LD   TRI   SEL_OP  -     -)

   (SHR1)  (NOP  NOP   NOP  READ  OE    SEL1 SEL0 SEL1 NOP  TRI   SEL_NA  SHR2  -)
   (SHR2)  (INC  READ  LD   NOP   OE    SEL1 SEL0 SEL1 LD   TRI   SEL_OP  -     -)

   (CMP1)  (NOP  NOP   NOP  READ  OE    SEL1 SEL1 SEL0 NOP  TRI   SEL_NA  CMP2  -)
   (CMP2)  (INC  READ  LD   NOP   OE    SEL1 SEL1 SEL0 NOP  TRI   SEL_OP  -     -)

   (JMP1)  (LD   NOP   NOP  NOP   OE    SEL0 SEL0 SEL0 NOP  TRI   SEL_NA  JMP2  -)
   (JMP2)  (NOP  READ  NOP  NOP   OE    SEL0 SEL0 SEL0 NOP  TRI   SEL_NA  NOP1  -)

   (BLT1)  (NOP  NOP   NOP  NOP   OE    SEL0 SEL0 SEL0 NOP  TRI   SEL_LT  NOP1  JMP1)

   (BGT1)  (NOP  NOP   NOP  NOP   OE    SEL0 SEL0 SEL0 NOP  TRI   SEL_GT  NOP1  JMP1)
)
```

(a) CDL description

```
(cell dp1)
(class DATAPATH)
(library work.types.all)
(library low_power.hwlib.all)

(terminal  dmem_OUT    (direction in)    (class data) (type data))
(terminal  D_BUS       (direction inout) (class data) (type data))
(terminal  SHIFT       (direction in)    (class data) (type data))

(terminal  tbuf0_FN    (direction in)  (class control)  (type Trist_Buffer_Fn_Type))
(terminal  mux0_SEL    (direction in)  (class control)  (type Mux2_Sel_Type))
(terminal  mux1_SEL    (direction in)  (class control)  (type Mux2_Sel_Type))
(terminal  cmp_STATUS  (direction out) (class control)  (type LogComp_Status_Type))
(terminal  acc_FN      (direction in)  (class control)  (type Counter_Fn_Type))
(terminal  tbuf1_FN    (direction in)  (class control)  (type Trist_Buffer_Fn_Type))

(terminal  CLK         (direction in)  (class clk)      (type Bit))

...
                Note: word length
(instance dpmux0 mux2 (parameters 16)
    (nets       ZERO acc_OUT mux0_OUT mux0_SEL))

(instance dpmux1 mux2 (parameters 16)
    (nets       ZERO D_BUS mux1_OUT mux1_SEL))

(instance cmp0 logcomp (parameters 16)
    (nets       mux0_OUT mux1_OUT cmp_STATUS))

...
```

(b) Excerpt from ADL description

Figure 6-18 : Excerpts from architectural description of microprocessor

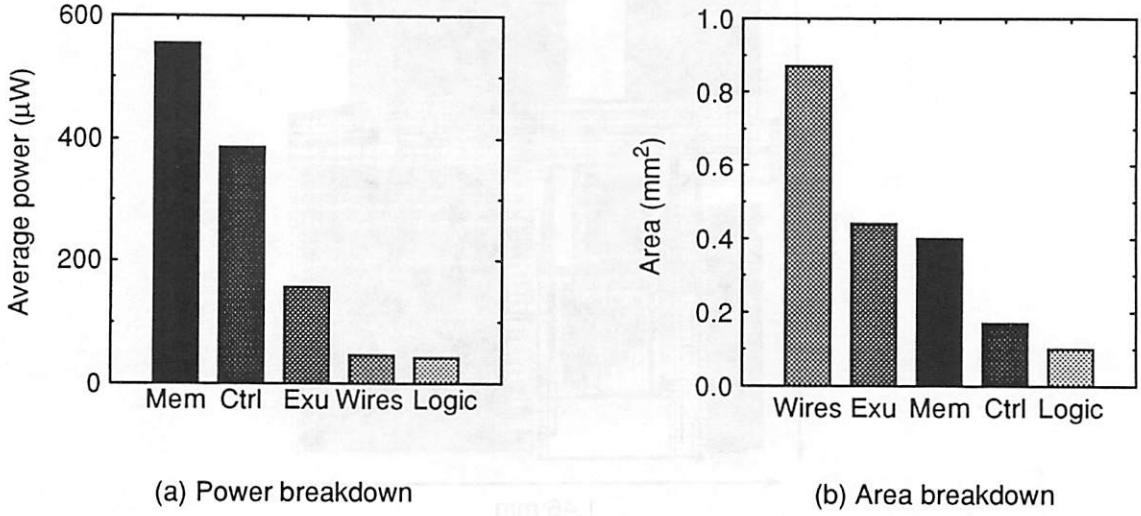


Figure 6-19 : Power and area breakdowns for microprocessor

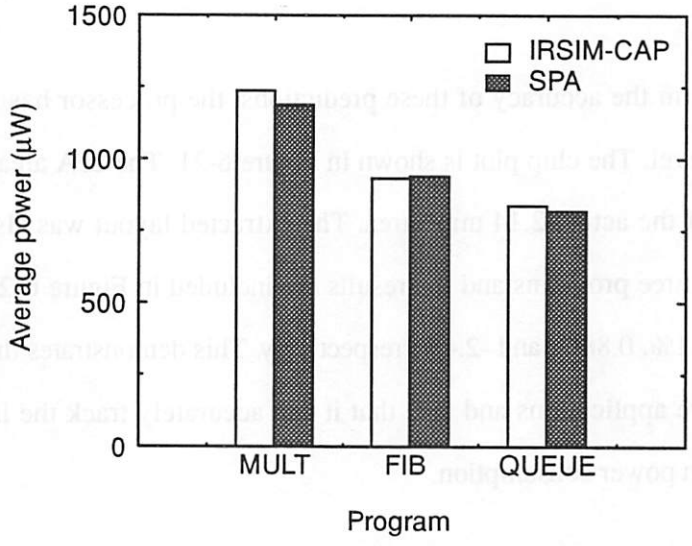


Figure 6-20 : Influence of instruction stream on power consumption

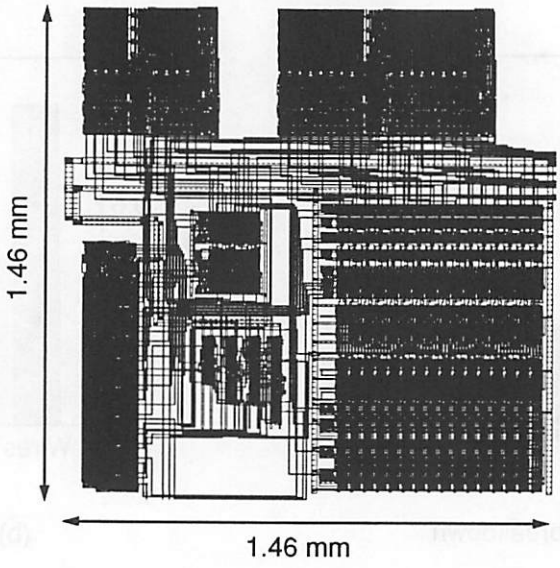


Figure 6-21 : Implementation of programmable microprocessor in 1.2 μm CMOS

prediction tool such as SPA, which can account for the influence of instruction statistics on power, quite valuable.

In order to confirm the accuracy of these predictions, the processor has been implemented down to the layout level. The chip plot is shown in Figure 6-21. The SPA area prediction of 1.96 mm^2 is within 8% of the actual 2.14 mm^2 area. The extracted layout was also simulated at the switch level for the three programs and the results are included in Figure 6-20. The error in the SPA estimates are -4.1%, 0.88%, and -2.4%, respectively. This demonstrates that SPA works well for memory-intensive applications and also that it can accurately track the impact of differing instruction streams on power consumption.

SPA is also able to analyze the influence of data stream on power consumption. Figure 6-22 compares predicted and actual powers for a single program (MULT) operating on several different data streams. For all data SPA is accurate to within 4% of the switch-level results. Even more importantly, it properly tracks the trend of the power fluctuations as data statistics vary.

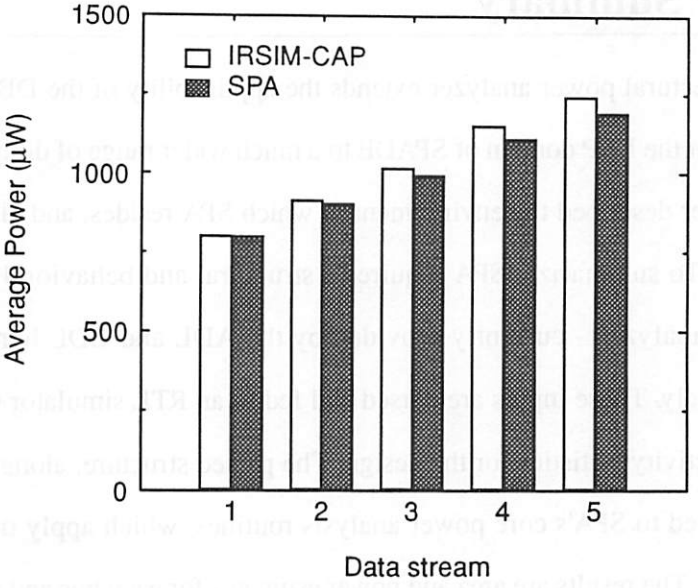


Figure 6-22 : Influence of data stream on power consumption

These results suggest that one possible application of SPA would be to characterize processor power consumption as a function of instruction and data statistics. The resulting models could then be used to develop a software compiler targeted at the generation of "low-power" code.

SPA's main advantage is that it requires only a high-level description of an architecture. For this example, the description took only about two hours to produce, whereas the layout-level implementation and debugging required more than 25 hours. Furthermore, SPA was able to analyze the chip power consumption in about one minute (on a Sun SPARCstation 10), including parsing, simulation, and estimation times. In contrast, extraction and simulation of the layout consumed about 45 minutes. These numbers are a clear indication that a high-level analysis tool like SPA is the only practical alternative for a designer wishing to consider the area and power efficiency of several architectural alternatives.

6.4 Chapter Summary

The SPA architectural power analyzer extends the applicability of the DBT and ABC power analysis models from the DSP domain of SPADE to a much wider range of digital applications and systems. This chapter described the environment in which SPA resides, and also provided details as to its operation. To summarize, SPA requires a structural and behavioral description of the architecture being analyzed - currently provided by the ADL and CDL hardware description languages, respectively. These inputs are parsed and fed to an RTL simulator (currently VHDL), which derives the activity statistics for the design. The parsed structure, along with the extracted activities, are then fed to SPA's core power analysis routines, which apply the analysis models derived in Chapter 4. The results are area and power estimates for each bus and module throughout the design hierarchy. The results can be categorized in terms of hardware classes such as Exu, register, multiplexer, buffer, control, etc.

SPA was applied to several examples. The range of designs chosen demonstrates the flexibility of the tool, as well as its accuracy. As with the SPADE examples, the results presented here also demonstrate specifically how the power analysis techniques developed in this thesis improve upon the state of the art. Specifically, the results showed several cases for which SPA correctly predicted the power savings, but for which existing architecture-level tools could not. The examples also showed the time that can be saved by optimizing power using high-level analysis tools such as SPA.

The next chapter will further address the issue of CAD-assisted power optimization. In particular it will present a methodology for low-power design, which relies heavily on high-level power estimation. This will provide a final motivation for tools such as SPADE and SPA and will also provide some initial indications as to which directions will be most fruitful for future research into CAD for low power.

CHAPTER 7

A High-Level Methodology for Low-Power Design

From the first, this thesis has advocated a practical approach to CAD - suggesting that design methodologies should drive tool development. To this end, early on Chapter 2 presented a collection of design techniques, which demonstrated the advantages of a high-level approach to low-power design. This motivated the development in subsequent chapters of tools for architectural power analysis. Through several examples these chapters demonstrated that it is possible for high-level tools to provide good early estimates of power consumption. Most of these results and examples focused on verifying the viability and accuracy of architectural power analysis. Relatively few, however, considered where architecture-level optimizations fit in the overall design optimization hierarchy, which spans from the algorithm level down to the layout level. Moreover, as of yet no coherent design methodology has been explicitly proposed. This chapter, then, returns to the driving issue of design, describing how power-conscious CAD tools can facilitate a high-level low-power design methodology.

Section 7.1 will describe the proposed methodology in detail. The methodology echoes many of the low-power themes described in Chapter 2 and advocates the same high-level approach

focusing on algorithmic and architectural optimizations. The proposed design space exploration strategy would be difficult to accomplish without the aid of power-conscious CAD tools. Section 7.2 will describe a possible CAD framework for low-power design that encompasses the necessary analysis and optimization tools. Finally, Section 7.3 will show how the proposed methodology and tools can be applied to a real-word design - in particular, the implementation of a low-power digital filter. High-level optimization of this filter is shown to lead to more than an order of magnitude savings in power. This result is achieved by a thorough search of the area-time-power design space, which would not have been practical without the assistance of high-level design tools.

7.1 Low-Power Design Methodology

Chapter 2 showed that high-level optimizations often lead to orders of magnitude reductions in power consumption, while gate, circuit, and layout optimizations usually impact power by less than a factor of two. We, therefore, propose a design methodology (and associated set of tools) that focuses on the algorithm and architecture levels of abstraction.

Since the key concept is to make decisions and optimizations starting with the highest level of abstraction and then to work down from there, the designer should first explore the algorithmic alternatives. There is rarely just one way to accomplish a task and, often, several different algorithms can be used to achieve the same functionality. Moreover, these different algorithms may have quite different complexities (i.e. required operation count, word length, etc.) and, therefore, different power needs. For example, Chapter 2 showed a speech coding example in which different algorithms of approximately equal quality had complexities that differed by close to 50%. Selecting the minimum power algorithm that satisfies the design constraints is consistent with the low-power design theme of *avoiding waste*. So, the first step in a top-down design methodology is to use a behavioral power estimation tool to evaluate different algorithms in terms of their intrinsic power requirements.

Complexity is not, however, the sole predictor of power consumption. For example, Chapter 2 also emphasized that algorithmic concurrency was an important consideration, since it determines how well the algorithm will map to the low-voltage concurrent processing architectures that were advocated in that chapter. In some cases, an algorithm with slightly higher complexity may be more amenable to, say, a pipelined implementation. Since this implementation may be able to run at a lower voltage than a non-pipelined processor, the more complex algorithm may actually result in a lower power consumption than a less complex algorithm with lower available concurrency. This suggests that the designer should not base his decisions solely on an algorithmic analysis. Instead, the algorithmic power estimates should only be used to narrow the search space. A tool is also required that explores the possibility of *trading area/performance for power* through parallel, low-voltage architectures.

Using this ATP exploration tool, the designer can further narrow the search space. The results from this tool, however, still contain a good deal of uncertainty. In order to be fast enough to allow a thorough exploration of many algorithmic and architectural alternatives, the exploration tool is forced to rely on a uniform white noise model for power analysis. As this thesis has repeatedly demonstrated, results based on UWN techniques don't model activity well and can give quite inaccurate results. Therefore, the algorithmic power estimation and exploration tools should be used only for a relative classification of algorithms and architectures in terms of power. They should not be relied upon to give accurate absolute estimates of power consumption. Furthermore, the designer should recognize that these tools will provide only a *coarse* classification of alternatives.

So the appropriate strategy is to use the algorithm-level estimation and ATP exploration tools to get a rough ranking of implementation possibilities. At this point, the designer should narrow the search space down to a few promising candidates that seem to offer reasonably close performance (i.e. within the confidence interval of the estimation tools). These candidates can then be analyzed more carefully at the architecture level.

At this level, the analysis tools can employ the more accurate DBT and ABC power models. These afford the designer the higher resolution required to make a final selection of the optimum solution. Moreover, these models allow the designer to analyze phenomenon that are transparent to the higher level estimation tools. For example, once the basic architecture has been selected, the designer must still decide how to assign operations to physical hardware units. This assignment affects the signal statistics and activity of the architecture and, therefore, its power consumption. By assigning operations from a temporally and spatially local portion of the computational graph onto the same processing elements (i.e. *exploiting locality*), maximum correlation and, therefore, minimum power can be attained. The UWN model used at the algorithm level cannot analyze these effects. Architectural power analysis must be used to make these kinds of refinements.

In summary, the methodology proposed in this thesis is a top-down approach to design optimization. Beginning at the algorithm level, behavioral power estimators can begin to classify alternative algorithms in terms of intrinsic power requirements as dictated by computational complexity. ATP exploration tools can then be used to evaluate the suitability of algorithms for implementation on low-power (e.g. low-voltage, concurrent) architectures. Finally, architectural power analysis can be used to verify design decisions made at the algorithm level and to explore additional opportunities for power optimization that are transparent to UWN-based estimation tools.

After an algorithmic and architectural selection have been made, the designer can proceed to gate-, circuit-, and layout-level implementation of the system. Further power optimizations are possible at these levels (see Chapter 2), but they should usually be considered subordinate to higher level optimizations. As a final verification step, switch- or device-level analysis tools (such as SPICE, PowerMill, or IRSIM-CAP) can be used to validate the results of the higher level tools.

7.2 A CAD Environment for Low-Power Design

The above discussion suggests that low-power solutions can be found by exploring different architectural and algorithmic implementation alternatives. It would be very difficult for a designer to explore these alternatives in a reasonable amount of time without the help of high-level, power-conscious CAD tools. The HYPER environment (first described in Chapter 5) contains all the necessary tools to support this exploration process. Using these tools, HYPER automates the task of design space exploration, allowing the user to rapidly synthesize several distinct algorithmic and architectural alternatives and select the optimal solution. As described in Chapter 5, the HYPER paradigm supports an exploratory approach to the optimization problem. At each stage, estimations are used to determine the location of the current implementation in the area-time-power (ATP) design space. These estimations provide clues to the designer regarding bottlenecks of the current solution. Using these hints, the designer can select from a number of basic moves or transformations in order to incrementally improve the solution. So, while the designer is free to manually guide the selection of a final architecture, the time-consuming transformation and estimation tasks are automated.

As shown in Figure 7-1, HYPER employs three main tools to facilitate low-power design. The first tool performs power estimation at the behavioral or algorithm level and will be briefly described in Section 7.2.1. The second, a power exploration tool, allows the user to map out area-power trade-offs in the ATP design space and will be discussed in Section 7.2.2. Finally, the architectural power analysis tool SPADE (described in Chapter 5) offers the higher degree of accuracy required to select and refine the final, optimum low-power architecture. SPADE relies on user-defined test vectors to improve power analysis accuracy.

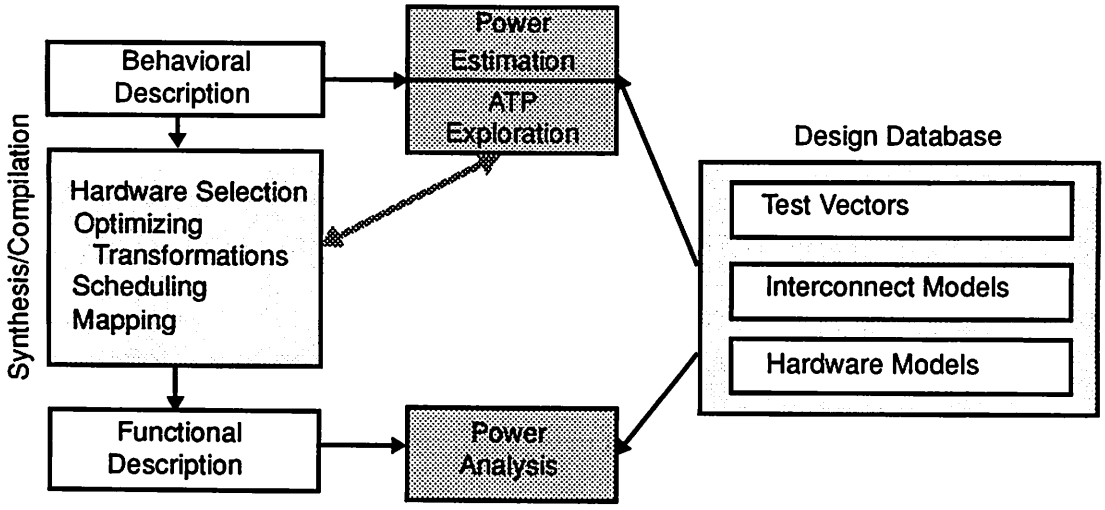


Figure 7-1 : HYPER low-power design environment

7.2.1 Algorithmic Power Estimation

The task of algorithmic, or behavioral, power estimation is to predict the power consumption of a chip given only the algorithm to be executed. This is not an easy task since the same operations can consume varying amounts of power when performed on different pieces of hardware. Still, it is possible to produce some sort of power estimates based solely on the computational requirements of the algorithm. Mehra and Chandrakasan have developed algorithmic power estimation techniques within the HYPER framework [Meh94][Cha93]. This section describes the power estimation strategies for the four main classes of chip components: datapath, memories, control, and interconnect.

The datapath power consumption can be estimated by looking at the type and quantity of the various operations required by the algorithm. For example, a given algorithm may require N_m multiplications, N_a additions, and N_c comparisons. In order to estimate the total energy required to execute the algorithm we need an estimate of the energy required per operation. To some extent,

these characteristic energies will depend upon the final hardware implementation; however, some rough estimates can be made based perhaps on results from previous designs or from existing hardware libraries. We can then estimate the total datapath energy by weighting the operation counts by the appropriate energies: e.g. $E_T=E_mN_m+E_aN_a+E_cN_c$.

In the case of HYPER, execution units are selected from a custom, low-power hardware library [Bur94]. Therefore, the tool can base its estimates on the measured power consumption of the appropriate library modules. Since the library is predefined, the power consumption of each cell can be precharacterized as a function of its complexity, reducing the datapath power estimation task to a series of table lookups.

The complexity parameters help determine the physical capacitance of the modules, but they do not describe the effect of data and control input activity on power consumption. Unfortunately, without knowledge of the final architecture, we cannot determine the statistics of the inputs to the hardware modules. Consequently, the modules are characterized for uniform white noise (UWN) inputs. For some examples, the UWN models are sufficient; however, as we know from the previous chapters, when data streams are correlated UWN estimation errors grow and results can be off by 50-100% or more. These inaccuracies can be remedied at the architecture level of analysis.

Memory power consumption can be estimated in a fashion similar to datapath modules. The number of memory accesses (Reads and Writes) can be determined directly from the algorithm specification. As with the execution units, the power consumed by each access can be approximated based on published data, measurements from past designs, or information from a precharacterized hardware database. The power estimator then weights the access counts by the energy per access for both Read and Write operations. Memory power estimation differs from datapath only in the parameters used to describe the complexity of the module, which include storage capacity (in words), as well as word length.

While operation and memory access counts can help us to estimate datapath and memory power consumption based solely on the algorithm description, it is difficult to estimate controller power without more detailed implementation-specific information. Here, one has to rely on experience from past designs to get reasonable power estimates. This information might take the form of a database relating the power consumed by several previously designed controllers to parameters such as the number of states in the finite state machine (FSM) and its output control signal activity. This is precisely the approach HYPER has taken. A statistical model by Mehra based on these high-level parameters was presented in Chapter 5 in the context of architecture-level power analysis [Meh94]. The reader is referred to that discussion for details. The same model can be applied at the algorithm level. The only difference is that the model parameters such as the number of states, N_{states}, the number of transitions, N_{trans}, and the number of accesses per bus, B_f, are not known at the algorithm-level and must be estimated from the behavioral flowgraph alone. The controller modeling error for the benchmark set ranges from an average of 12% to a maximum of 41%.

The final component of power consumption relates to interconnect. The physical capacitance of global buses can be determined from an estimate of the average wire length. This in turn can be derived from the area of the chip. An architecture-level model for area and interconnect (developed by Mehra) was also presented in Chapter 5. Once again, the parameters required by the model were known at the architecture level; however, at the algorithm level they must be estimated. It is possible to estimate these values based on an analysis of the data dependencies and timing constraints of the algorithm [Meh94][Rab91]. When these estimates are employed instead of actual parameter values, the average and worst-case modeling errors are 43% and 96%, respectively.

The power estimate for the entire chip is obtained merely by summing the datapath, memory, control, and interconnect components. If we are to rely on this estimator when making high-level design decisions we must establish some level of confidence in the results it produces. Note that

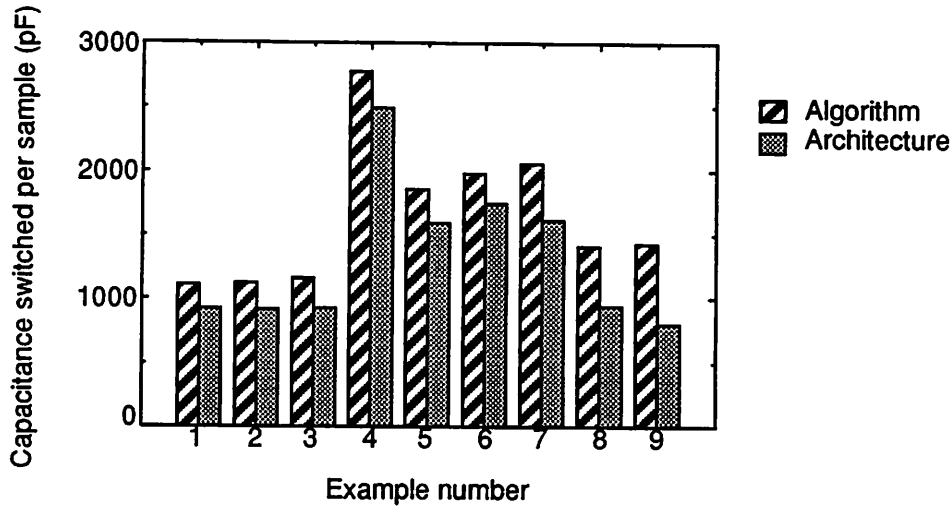


Figure 7-2 : Correlation between the algorithm- and architecture-level estimates

relative accuracy is more important than absolute accuracy at this stage of design. Figure 7-2 shows the algorithm- and architecture-level energy estimates for nine versions of the Avenhaus filter that will be presented in the upcoming case study. In general, there is good correlation between the algorithmic and architectural estimates in spite of the lower absolute accuracy of the algorithm-level tool. Therefore, estimates at the algorithm level can be used for relative evaluation of different designs. One must bear in mind, however, the limitations of the UWN model and make only rough classifications that are based on significant differences in algorithmic estimates. For the examples of Figure 7-2, if the designer were to rely solely on the algorithmic estimates he might assume that versions 1, 2, and 3 were lower power than versions 8 and 9. The architectural estimates show that just the opposite is true. Therefore, when algorithmic estimates are closely spaced, architectural analysis should be used to produce a more fine-grain, reliable classification.

While behavioral estimation allows the designer to plot a single point in the ATP space, the next section describes a tool which allows him to explore a whole range of trade-offs between area, time, and power.

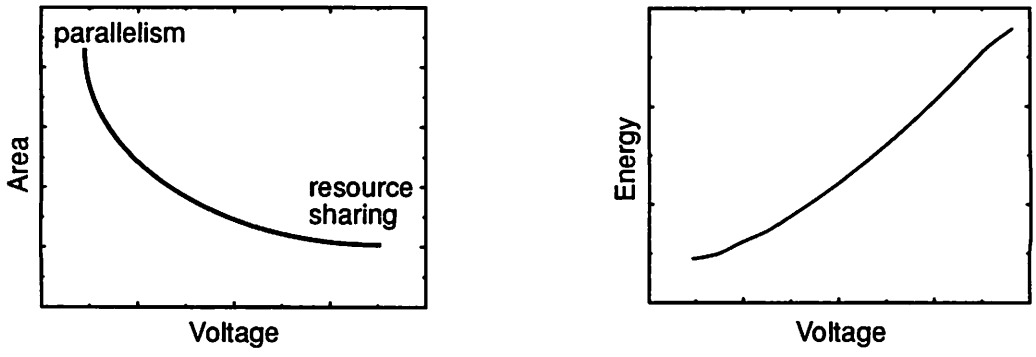


Figure 7-3 : Area-power exploration graphs

7.2.2 Design Space Exploration

Section 7.1 mentioned trading area-performance for power as a key low-power design strategy. The basic concept was to scale down voltage and make up for the loss in performance by employing techniques such as parallel processing. The HYPER environment provides an exploration tool (developed by Mehra) that allows the designer to see this trade-off between area and power graphically [Meh94].

The output of the tool is a set of exploration curves that plot the area and power of an algorithm as a function of supply voltage (see Figure 7-3). As shown in the figure, the area increases for lower voltages since the longer circuit delays must be compensated for by additional parallel processors. This parallel processing allows the design to meet the performance requirements while operating at reduced voltage. The exploration curves are produced point-by-point by iteratively invoking the algorithmic power estimator over a range of operating voltages. The curves can prove extremely useful to the designer in selecting the algorithm and architecture that offer the best compromise between area, performance, and power.

As the exploration curves are based directly on the behavioral power estimates discussed above, they are subject to the same inaccuracies. For a finer grain classification of

implementations, we must again rely on architectural power analysis.

7.3 Case Study: The Avenhaus Filter

Sections 7.1 and 7.2 presented an overall methodology for low-power design, as well as a set of tools forming a CAD framework that supports this methodology. In this section we present a case study that demonstrates the viability of the methodology, the tools, and the design environment. This case study was carried out in cooperation with R. Mehra, a fellow U. C. Berkeley researcher. The general approach taken here is suitable for a wide variety of applications. For the purposes of illustration, however, we will consider the specific task of producing a low-power implementation of the Avenhaus filter [Ave93].

Using this example, we will proceed through a sample design flow, at each stage highlighting issues of particular importance. The process begins with a preliminary evaluation of the Avenhaus filter, comparing the various structures that can be used to implement its transfer function. Next, we explore the design space at the algorithm level, applying several of the low-power strategies of Chapter 2. After narrowing down the design space, we proceed to architecture-level analysis in order to verify and refine our design decisions. We conclude with a review of the power savings achieved at each stage of optimization.

7.3.1 Preliminary Evaluation

The Avenhaus, an eighth-order bandpass filter, can be implemented using several different filter structures. The different structures considered here are the cascade, continued fraction, direct form II, ladder, and parallel forms proposed by Crochiere [Cro75]. Each of these forms has a very different computational form and, thus, might be expected to map to distinct points in the ATP design space. Other studies have considered the algorithm-level area and performance trade-offs

	Critical path (ns)	Max. sample freq. (MHz)	Word Length	Mult	Add	Energy (nJ)
Cascade	340	2.94	13	13	16	27.7
Continued Fraction	950	1.05	23	18	16	89.5
Direct form II	440	2.27	19	16	16	59.5
Ladder	1224	0.82	14	17	32	52.1
Parallel	306	3.27	13	18	16	36.1

Table 7-1 : Complexity of the initial structures operated at 5V

for these structures [Pot93][Rab93], but to the best of our knowledge this is the first attempt to study the power aspects.

In the following, we shall assume that the designer is free to select the algorithm (i.e. filter structure), apply any number of transformations to it, and choose an appropriate supply voltage, as long as he meets a throughput requirement of 2.75 MHz, imposed by the surrounding system.

As a preliminary step in the algorithm selection process, we can profile each filter structure in terms of several key parameters that are expected to influence power consumption (see Table 7-1). The table shows the *maximum* throughput of each algorithm (in terms of critical path and sample frequency) and the complexity (in terms of required word length and operation count). It also gives an estimate of the energy required per sample assuming a straightforward implementation of each structure. To allow a direct comparison, maximum frequency and energy results are quoted for standard 5V implementations with clocking frequencies set so that all operations finish within a single cycle. Note that all estimates were produced using the behavioral estimator described in Section 7.2.1.

As discussed in that section, the complexity of an algorithm has a major impact on the power

consumed. Extra bits of word length contribute to larger physical capacitance and increased activity. Likewise, higher operation counts contribute directly to increased activity and can also necessitate increased hardware and routing, resulting in larger physical capacitance. This is reflected in the table, which shows that the cascade and parallel implementations have among the lowest operation counts and the smallest word lengths and, consequently, the lowest energies. This relates directly to the previously mentioned low-power theme of avoiding waste.

Complexity is not, however, the only factor determining power consumption. The minimum operating voltage also has an important effect. Notice that, in their current forms, only the cascade and parallel structures can meet the 2.75 MHz throughput constraint at 5V. The critical paths for the other algorithms are much longer and, thus, they cannot realize the required sample rate even at 5V.

As yet, however, we have only considered the direct implementation of each filter type. It is quite possible that transformations such as constant multiplication expansion or pipelining could reduce the critical path of one of the other algorithms, allowing it to operate at a much lower voltage and power. Likewise, other transformations might drastically reduce the complexity of one of the structures, making it the minimum power solution. In summary, at this stage the cascade and parallel forms look promising, but more exploration is needed before making a final selection.

7.3.2 Programmable vs. Dedicated Hardware

One transformation that can be very useful in power reduction is expansion of constant multiplications into adds and shifts. Under this transformation, only additions and shifts corresponding to `1`'s in the coefficients are performed. In contrast, an array multiplier implementation performs an addition for each bit of the coefficient even if it is a `0`. Therefore, if the coefficients have a relatively large number of `0`'s, it may be reasonable to use shift-adds since the array multiplier would perform unnecessary operations.

	Critical path (ns)	Max sample freq. (MHz)	Add	Sub	Shifts	Energy (nJ)
Cascade	361	2.77	38	23	51	43.5
Continued Fraction	1104	0.91	68	50	116	98.7
Direct form II	440	2.27	54	40	91	95.6
Ladder	1406	0.71	36	31	46	75.4
Parallel	437	2.29	40	30	63	61.3

Table 7-2 : Complexity of the structures after constant multiplication expansion

On the other hand, the dedicated array multiplier has certain advantages. First, it performs shifts by hard-wired routing, while the shift-add version uses programmable shifters, latching intermediate results between stages. These shifters and latches contribute additional overhead and can offset the gains from the reduced number of additions. Whether this overhead is dominant or not depends on the particular value of the coefficients.

For the case of the Avenhaus filters, we see an increase in energy after expanding constant multiplications (see Table 7-2). Notice that while no multiplications remain, the number of additions has increased substantially and a number of subtract and programmable shift operations have been added. This increased complexity has a significant impact on the power consumed. Moreover, the programmable hardware adds to power consumption through additional control logic and signals.

Table 7-2 also reveals a considerable increase in some of the critical paths. This is a substantial penalty since, as we have noted in Chapter 2, a high speed design is almost always desirable due to its potential for allowing voltage reduction. For this example we, therefore, choose dedicated multipliers over programmable shift-adds. Again, this may not always be the optimum decision; the trade-offs must be evaluated on a case-by-case basis.

	original	1 stage	2 stage	3 stage	4 stage	5 stage
Cascade	340	170	136	102	-	-
Continued Fraction	950	850	-	-	-	-
Direct form II	440	132	-	-	-	-
Ladder	1224	612	432	324	252	216
Parallel	306	170	102	-	-	-

Table 7-3 : Critical paths (in ns) after pipelining to different extents

7.3.3 Critical Path Reduction and Voltage Scaling

Supply voltage reduction is an excellent way to reduce power consumption. As is apparent from Table 7-1, however, not all the algorithms can meet the throughput constraint even at 5V. Therefore, in their current forms we cannot consider reducing the voltage for any of these structures below 5V. If we can reduce the critical paths, however, we open the possibility of voltage reductions.

Flowgraph transformations provide a powerful tool for design optimization at the algorithm level and can be used for critical path reduction. Transformations alter graph structures, while preserving input/output relationships. The HYPER high-level design environment automates the task of applying transformations. This allows us to explore the effect of many different transformations on each of the candidate algorithms - a task that would not be practical for a designer using a manual approach.

One important transformation for critical path reduction is pipelining. By allowing more operations to occur in parallel, pipelining reduces the critical path of the design, enabling voltage reduction while still maintaining the required throughput. As a result of pipelining, some of the filter structures that could not meet the throughput constraint initially become feasible, while those already feasible are able to operate at lower voltages. Table 7-3 shows the reduction in the critical

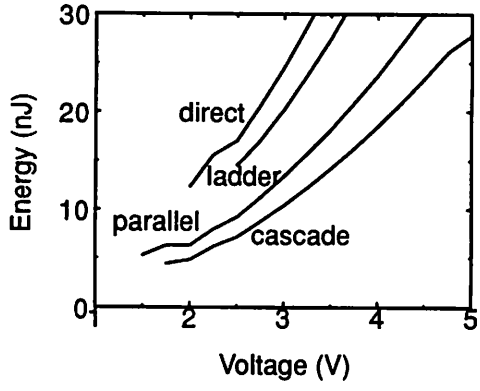


Figure 7-4 : Voltage reduction (and its effect on energy) after "optimal" pipelining

paths after pipelining.

But critical path reduction is only an indirect measure of improvement. We are actually interested in the minimum voltage and energy achieved after pipelining (as shown in Figure 7-4). The curves in this figure were produced using the exploration tool discussed in Section 7.2.2. These curves graphically illustrate that voltages (and energies) can be appreciably reduced for all examples (except the continued fraction) by applying pipelining.

The results from the exploration curves are summarized in Table 7-4. Notice that the optimum level of pipelining is not always equal to the maximum amount of pipelining. This is due to the overhead associated with pipelining. For example, plotting the exploration curves for the maximally pipelined cascade (three stages) would reveal that it consumes a minimum energy of 5.1 nJ which is higher than the two-stage version which consumes 4.4 nJ.

It is apparent that the cascade and parallel versions still yield the lowest power solutions. Therefore, based on the results obtained from our high-level exploration tools, we can at this point eliminate the continued fraction, direct form, and ladder implementations from consideration. We do not eliminate the parallel form at this stage since it gives results close to the cascade, and the error inherent in the high-level estimation tools does not allow us to resolve differences that are so

	# stages	Critical path	Min. voltage	Energy (nJ)	Area (mm^2)
Cascade	2	136	1.75	4.4	157
Continued Fraction	1	850	-	-	-
Direct form II	1	132	2.00	12.3	374
Ladder	5	216	2.50	14.5	126
Parallel	2	102	1.50	5.3	411

Table 7-4 : Effect of "optimal" pipelining on the energy and area of the designs

small.

Power is not, however, the only consideration. As we reduce voltage we require more parallel hardware to meet the throughput constraints. So, while energy can be reduced by this technique, a price has to be paid in terms of area. The area exploration curves of Figure 7-5 illustrate this point. Since the cost of a component is directly related to the amount of silicon real-estate it requires, designers have a limited amount of area available to them. For our example, in order to achieve the minimum energies, the designer would have to accept significant area penalties as shown in the figure. But at a slightly higher operating voltage of about 2V, the area penalties are much less severe. Moreover, the resulting energies at 2V are not significantly higher (see Figure 7-4). For the purposes of this example, we choose to avoid the large area penalties by operating at 2V.

There are, of course, other power reduction techniques which the designer could explore within the HYPER environment. A number of transformations for low power are described in [Cha93]. But the purpose of this case study is to present a general methodology and show how high-level tools can be used to facilitate low-power design - not to enumerate all possible implementations of the Avenhaus filter.

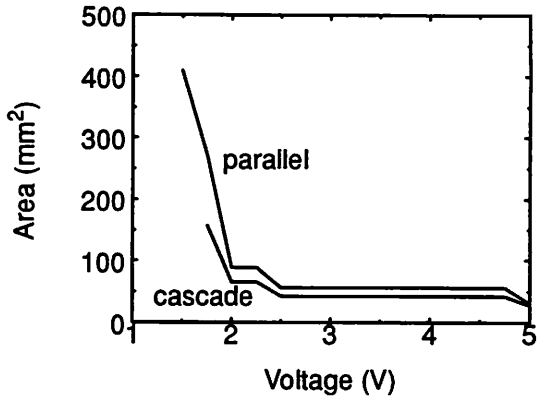


Figure 7-5 : Area - energy trade-off for the cascade and parallel versions

So far, the optimization process has been limited to algorithmic exploration and analysis. There are additional techniques at the architecture level for reducing power. Moreover, architectural power analysis will allow us to verify and refine the decisions already made at the algorithm level.

7.3.4 Architectural Exploration

Using algorithmic power estimation and exploration tools, we have been able to narrow down the design space to two filter structures: a cascade and a parallel form, both with two stages of pipelining operating at 2.75 MHz and 2V. A detailed breakdown of the energy estimates at the algorithm level is given in Table 7-5. As described in Section 7.2.1, at this level we are unable to account for the effect of signal statistics on power. So if we assume that we are filtering speech data (which is highly correlated) we might expect some inaccuracy in the algorithmic power estimates. In order to verify and refine these results, we now synthesize the selected structures using HYPER and turn to architectural power analysis for more accurate energy estimates. The results of this process are also contained in Table 7-5 (the meaning of the "Local Assignment" column will be explained shortly).

	Cascade			Parallel	
	Algorithm	Architecture	Local Assignment	Algorithm	Architecture
Exu	2.42	1.65	1.44	3.27	2.05
Registers	0.59	0.50	0.50	0.64	0.52
Control	0.62	0.62	0.62	0.73	0.73
Buffers	0.08	0.06	0.07	0.09	0.06
Mux	-	0.18	0.18	-	0.21
Bus	1.01	1.04	0.29	1.31	1.31
Clock	0.31	0.25	0.25	0.38	0.33
Total	5.03	4.30	3.35	6.42	5.21

Table 7-5 : Energy breakdown (in nJ) for cascade and parallel filters

For a speech input, we see that the algorithmic power estimator did, indeed, overestimate the execution unit power by 47% for the cascade and 60% for the parallel; however, the total chip power estimates were more accurate with 17% and 23% errors, respectively. The important point, however, is that the errors in the algorithmic power estimates were systematic overestimates rather than random errors. This suggests that the relative classifications made during algorithmic design space exploration were meaningful.

Notice that the cascade structure continues to be the lowest power solution. Therefore, based on these accurate architecture-level estimates, we are able to select the cascade filter as our final low-power Avenhaus implementation. Some final optimizations are still possible, however, at the architecture level.

For instance, assignment of operations to hardware units can have a significant impact on power. One possible assignment strategy involves assigning operations for maximum temporal

and spatial locality so as to minimize activity as described in Section 5.3.3 of Chapter 5. This is beneficial because signals that are local to a small portion of the data stream or filter tend to be more highly correlated than widely separated signals. If we assign the operations on highly correlated signals to the same hardware units, there is likely to be less activity in the sign bits. This in turn reduces the power consumption of the implementation. Note that the analysis of this effect can only be performed using an activity-sensitive technique such as the DBT model. The UWN-based algorithmic estimator is not able to make this distinction. The results of assignment for locality on the cascade are also shown in Table 7-5. The overall energy is reduced an additional 22% with no penalty in area or performance.

7.3.5 Gains from Design Space Exploration

The preceding case study demonstrates that design decisions at the algorithm and architecture levels may have a tremendous impact on the power. Table 7-6 summarizes the impact of the various optimizations that were performed. Selecting the correct algorithm (cascade) saved a factor of three in power, relative to the worst case (direct form). Moreover, the direct form could not even achieve the required 2.75 MHz sampling rate at 5V. If the algorithms were compared for the same throughputs, the cascade would actually be even more than 3x better. The case study also demonstrated that, perhaps counterintuitively, expanding multiplications into shifts and adds is not always beneficial and can actually increase the power consumption in some instances. Other transformations on the flowgraph structure, such as pipelining, were found to offer power reductions of a factor of more than 6x. Finally, assignment for maximum locality enabled a further power reduction of 22%. As Table 7-6 illustrates, coupling the algorithmic improvements with architectural optimizations can allow us to achieve more than an order of magnitude power reduction (27x for this example).

	Inputs	Voltage (V)	Energy (nJ)	Energy reduction
Worst algorithm (direct form)	UWN	5.0	> 89.5	1
Best algorithm (cascade)	UWN	5.0	27.7	3x
After constant mult. expansion on cascade	UWN	5.0	43.5	2x
Pipelining (no area constraint)	UWN	1.5	4.4	20x
Pipelining (with area 100 mm^2)	UWN	2.0	5.0	18x
Architecture-level estimate	Speech	2.0	4.3	21x
Assignment for locality	Speech	2.0	3.3	27x

Table 7-6 : Energy reductions from design space exploration

7.4 Chapter Summary

This chapter has presented strategies for design space exploration at the algorithm and architecture levels of abstraction. The HYPER high-level synthesis system and low-power design environment was used as a vehicle to describe the proposed tools and methodologies. The power optimization methodology emphasized high-level techniques based on the recurring low-power themes of trading area/performance for power, avoiding waste, and exploiting locality. The efficacy of the methodology and supporting tools was demonstrated through a case study using the Avenhaus filter structures. With algorithmic and architectural optimization strategies, power savings of almost 30x were demonstrated.

In conclusion, the high-level design space is vast and presents the designer with numerous trade-offs and implementation options. The multiple degrees of freedom preclude a manual exploration of all avenues for power optimization. An exploration environment (such as HYPER) provides the user with a means for rapidly and efficiently searching the design space for solutions

that best meet area, performance, and power criteria.

The current HYPER environment is interactive and primarily user-driven. In the future, one can envision a fully-automated environment, which intelligently searches the design space attempting to optimize some cost function based on area, performance, and power. In the next chapter, we explore this and other directions for future research into CAD for low power.

CHAPTER 8

Directions for Future Work

As with most research efforts, the concepts presented in this thesis raised perhaps as many questions as they solved. This chapter will discuss some of these issues, presenting directions for future work in the area of low-power design methodologies and tools. This discussion will begin by pointing out limitations of the power analysis models presented in the previous chapters and suggesting improvements that could be made to these techniques. Next, the discussion will turn from models to tools, tackling the question of what types of tools and CAD frameworks would be useful for low-power designers today and in the future.

We begin by discussing possible improvements to the power analysis models. One drawback of the DBT model is the large number of capacitive coefficients required to characterize a module with two or more inputs. Currently a separate coefficient is required for every possible sign transition of the inputs and outputs. This grows exponentially with the number of I/O pins on the module. It would be worthwhile to explore ways of reducing the amount of data required to characterize each module. One possibility would be to use a less detailed, less exhaustive measure of sign activity - perhaps something similar to the activity parameters used by the ABC model. In

addition, it would be useful to extend the DBT model to account for more than just two's-complement data types. Although this is the most common data representation, it would not be difficult to add generality by supporting sign-magnitude or even floating-point formats.

There is also room for improvement in the ABC control model. The biggest approximation made by the ABC model stems from characterizing controllers for random control tables only. This is equivalent to assuming that the physical capacitance of a controller is independent of the specifics of the STG being implemented. Such approximations are difficult to avoid when attempting to predict a single, average power consumption for an implementation. One solution to the dilemma might be to attempt to bound the power consumption, estimating minimum and maximum likely power consumptions for a given STG. Providing a range rather than a single estimate might give the user a more realistic picture of the variability inherent in high-level estimates of controller power consumption.

Estimation of average interconnect length is subject to the same limitations. It might be worthwhile in the future to look into ways of reducing this uncertainty. For example, an early floorplanning and preliminary routing step might result in more reasonable estimates of interconnect and clock network capacitances.

It might also be noted, that both the DBT and ABC models rely heavily on a library-based approach to chip design. The need to precharacterize all modules in terms of power consumption is, in fact, one of the major pitfalls of this approach. Even if the cell is available in a library, it is not always trivial to determine the capacitance model appropriate to the cell and the precise control and data input patterns that should be applied to the cell during characterization. Techniques for encapsulating and automating this characterization process need further exploration. Moreover, since not all designers and organizations employ extensive hardware libraries, it would be quite useful to explore methods of power analysis that are less library-dependent. It might even be possible, given a process technology and logic family, to come up

with theoretical lower bounds for power consumption, perhaps relying on results from information theory.

We can also consider power analysis when even less information is available - for example, at the algorithm level. Ideas for power analysis at this level of abstraction have been developed by other researchers and some were presented in this thesis; however, they lack the ability to account for activity statistics in any meaningful way. Since activity can play an important role in determining power consumption, it might be worthwhile to look into developing activity-sensitive algorithmic power estimators.

In addition to the possibility of future improvements in the power analysis models, it is worthwhile to consider the future of CAD in low-power design. Specifically, what types of tools and frameworks will the designer find most helpful in implementing high quality solutions in the least amount of time. Ultimately, it may be possible for the designer to specify a desired functionality along with a set of area, performance, and power constraints and have the tools automatically synthesize a nearly optimal solution. There are, however, substantial obstacles barring us from these goals. Good designers have a base of experience that may always allow them to achieve results far superior to synthesized solutions. So the role of CAD for today and the foreseeable future is not to force the designer towards a particular solution, but to allow him to guide the design process, while automating tedious tasks and providing feedback on his progress. This implies a focus on tools for analysis and exploration rather than optimization.

SPADE and SPA were designed to adhere to this philosophy; however, a comprehensive design space exploration environment is still some distance in the future. It will take some effort to seamlessly integrate area, performance, and power estimation into a single environment, but this is an important step since real designs must satisfy constraints in all three dimensions. The environment should also take a system-level perspective to design. This will involve support for exploration of design alternatives ranging from full-custom to ASIC's to off-the-shelf

programmable processors. This will also require seamless handling of analog, as well as digital components since almost all systems contain components of both types. An equally important, but often overlooked, aspect of this environment will be the user interface. If the user cannot easily and intuitively specify the system structure and behavior, it unlikely the tool will gain wide acceptance. Likewise, some thought needs to go into how design information can best be communicated back to the user. For instance, what types of graphs or tables would be most useful to the designer at each phase of the design process. The user interface is often not considered an "interesting" research problem, but it is certainly of enormous practical importance.

Beyond a CAD framework for user-directed design space exploration, there is at least the hope of a synthesis tool for automated and global design optimization. While relatively little progress has been made thus far towards this goal, it is nonetheless worthwhile to continue research in this area. The first step in the process will be to develop strategies and heuristics for low-power design. Tools such as SPA and SPADE can make an important contribution to these efforts by allowing the user to identify which techniques are most successful in reducing system power consumption.

This chapter has described several directions for future research in the area of low-power design methodologies and tools. Some of these were simply areas in which the research presented in this thesis can be improved, while others addressed higher level issues such as the place of CAD in the future of low-power design. Regardless of what direction CAD for low-power takes, however, the research presented here should provide a strong foundation upon which to build.

CHAPTER 9

Conclusions

As its primary contribution, this dissertation has presented a novel technique for architectural power analysis. Rather than attempting to find one model that fits all components of a chip, the technique uses specialized power analysis strategies for different classes of hardware. For example, datapath and memory elements are analyzed with the Dual Bit Type, or DBT, capacitance model. This model improves on the state of the art by modeling not only the noise-like activity of the data bits, but also the activity of the sign bits. It was demonstrated that while white noise techniques often err by more than 50%, the DBT model can be used to obtain results within 15% of switch-level simulations.

A high-level model for analyzing the control path power consumption was also presented. Like the DBT model, its results accurately reflect the activity of the control signals and logic - therefore, it is referred to as the Activity-Based Control, or ABC, model. Control power estimation is a more difficult problem than datapath estimation since the contents of the controller are not known during module precharacterization. Still, the ABC model achieves good results, often within 20% of switch-level simulations.

Interconnect/area analysis was another difficult task undertaken in this research. An approach for average interconnect length estimation was described that combines ideas taken from existing research with a hierarchical decomposition strategy to provide reasonable estimates of average bus capacitance that reflect the relative locality of a bus within the design hierarchy.

Using these concepts, two chip-level power analyzers were implemented. The first was targeted towards datapath-intensive DSP applications and was dubbed SPADE, which stands for the Stochastic Power Analyzer with DSP Emphasis. SPADE resides within the HYPER high-level synthesis environment. This environment allows the designer to rapidly synthesize several different target architectures and efficiently map out the area-time-power implementation space. The HYPER environment does, however, limit the applicability of SPADE to DSP ASIC's. This constraint was addressed by a second power analyzer, SPA, which allows the user to specify fairly arbitrary architectures and algorithms using high-level structural and behavioral description languages. Both tools improve upon existing architectural power analysis methodologies by accounting for the dependence of circuit activity on the statistics of the data and instructions being processed.

As a result, derivation of activity parameters proved to be an important task for both power analyzers. Both tools rely on simulation: SPADE translates the behavioral flowgraph to C for simulation and then maps activity statistics to the architectural structure, while SPA performs RT-level functional simulation of the architecture as a whole using VHDL. The accuracy of these tools was verified against switch-level simulation with several examples. Typically, the tools exhibited aggregate error rates of less than 20% while operating orders of magnitude faster than lower level analysis tools - requiring seconds rather than hours to produce results.

A high-level low-power design methodology and CAD environment that relies on these tools was also presented. The proposed methodology reflects a top-down approach to low-power design. Beginning with the desired functionality of the chip, algorithm-level power estimators can be used

to compare the intrinsic power efficiency of different algorithms that all achieve the same functionality. A case study demonstrated that variations in power consumption can often reach 3x for algorithms with identical functionality. The designer can also use high-level exploration tools to map out trade-offs between area and power consumption for architectures employing various degrees of parallelism and operating at differing voltage supplies. Once an approximate voltage supply and degree of parallelism have been selected, the designer can perform a more detailed comparison on the most promising architectures using an RTL power analyzer such as SPADE or SPA. These analyzers can also be used to guide architecture-level optimizations such as assignment of operations to hardware units for maximum locality and minimum power. This methodology was demonstrated to result in more than an order of magnitude reduction in power consumption for the case study presented.

These high-level power optimization strategies and analysis tools are consistent with the notion that high-level design decisions have the most dramatic influence on power consumption. This was demonstrated through a comprehensive summary of power optimization techniques at all levels of abstraction from technology to system. In this discussion, low-level techniques such as technology scaling and circuit- and gate-level optimization were shown to typically offer a factor of two or less improvement in power consumption. In contrast, architecture-, algorithm-, and system-level strategies including selective power down, partitioning for locality, and low-voltage concurrent processing were shown to offer one to three orders of magnitude reduction in power consumption. This fact provides a motivation for the high-level power analysis tools and design methodologies discussed in this work.

This motivation is strengthened by the fact that currently, power-conscious high-level CAD tools are almost non-existent. Instead, tool developers have focused primarily on circuit-level, and more recently, gate-level power analysis tools. A comprehensive overview of the state of the art in power estimation made evident this preponderance of low-level tools. The few architecture- and algorithm-level estimation techniques that do exist were shown to be fairly inaccurate, especially

in modeling the effect of activity on power consumption.

We, therefore, feel that the tools and techniques developed as part of this research represent a significant contribution to the field of high-level design for low power. First, they operate at the architecture level and, therefore, take advantage of the fact that high-level design decisions have the largest impact on power consumption. They are also much faster than circuit- or gate-level tools and will allow designers to rapidly compare the power consumption of several alternative architectures in a thorough search of the area-power-performance design space. This efficiency stems from two sources. First, the relatively small number of components at the architecture level of abstraction allows rapid analysis of designs. Second, since only an RT-level description of the design is required, analysis can be performed very early in the design flow. In contrast, circuit- and gate-level tools are more useful as a back-end verification of power consumption than as exploration aids since the design must be more or less complete before these tools can be applied.

A truly useful design space exploration environment must allow the user to analyze performance, as well as area and power. Therefore, the analysis strategies presented in this thesis should really be combined with performance estimation techniques; however, this thesis has focused primarily on power (although area was also considered) since to a large extent tools for performance and area analysis already exist. This is not surprising since area and performance have been the traditional measures of design "quality" for many years. Only recently has power entered the picture as an important design criterion. For high-performance systems, packing increased functionality in the same die area has led to power-related reliability concerns, as well as extremely expensive packaging requirements. Another motivation for lowering power consumption has been the phenomenal growth of the portable consumer electronics market. Devices such as cellular phones, lap-top computers, and personal digital assistants operate off batteries and, therefore, must be extremely frugal with their power consumption. From all indications, these trends will continue and even accelerate, making low-power design methodologies and tools of ever increasing interest to the designers of today and of tomorrow.

CHAPTER 10

Bibliography

[Ave93] E. Avenhaus, "On the design of digital filters with coefficients of limited word length," *IEEE Trans. on Audio and Electroacoustics*, vol. 20, pp. 206-212, 1972.

[Bac84] G. Baccarani, M. Wordeman, and R. Dennard, "Generalized Scaling Theory and its Application to 1/4 Micrometer MOSFET Design," *IEEE Transactions on Electron Devices*, ED-31(4), p. 452, 1984.

[Bah94] R. Bahar, H. Cho, G. Hachtel, E. Macii, and F. Somenzi, "An Application of ADD-Based Timing Analysis to Combinational Low Power Re-Synthesis," *1994 International Workshop on Low Power Design*, Napa Valley, CA, April 1994.

[Bak90] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, Menlo Park, CA, 1990.

[Bec94] R. Bechade et al., "A 32b 66MHz 1.8W Microprocessor," *Proceedings of the International Solid-State Circuits Conference '94*, San Francisco, CA, pp. 208-209, February 1994.

[Bel75] S. Belter and S. Bass, "Computer-Aided Design of Digital Filters with Arbitrary Topology," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, no. 10, pp. 810-819, October 1975.

[Ben94] L. Benini, M. Favalli, and B. Ricco, "Analysis of Hazard Contributions to Power Dissipation in CMOS IC's," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 27-32, April 1994.

[Ben90] D. Benson, Y. Bobra, B. McWilliams et al., "Silicon Multichip Modules," *Hot Chips Symposium III*, Santa Clara, CA, Aug. 1990.

[Bra84] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[Bro91] R. W. Brodersen et al., "Technologies for Personal Communications," *Proceedings of the VLSI Symposium '91 Conference*, Japan, pp. 5-9, 1991.

[Bro92] R. W. Brodersen ed., *Anatomy of a Silicon Compiler*, Kluwer Academic Publishers, 1992.

[Bry93] J. Bryan, "Low-Power Technology: The 3V Revolution," *Computer Shopper*, vol. 13, no. 1, pp. 801-804, Jan. 1993.

[Bry86] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions*

on Computing, vol. C-35, pp. 677-691, August 1986.

[Bur92a] R. Burch, F. Najm, P. Yang, and T. Trick, "McPower: A Monte Carlo Approach to Power Estimation," *Proceedings of the International Conference on Computer-Aided Design*, pp. 90-97, November 1992.

[Bur94] T. Burd, *Low-Power CMOS Library Design Methodology*, Masters Thesis, ERL U.C. Berkeley, June 1994.

[Bur91] J. Burr and A. Peterson, "Energy Considerations in Multichip Module Based Multiprocessors," *International Conference on Circuit Design '91*, pp. 593-600, 1991.

[Bur93] J. Burr, "Stanford Ultra Low Power CMOS," *Hot Chip Symposium III*, Santa Clara, CA, Aug. 1990.

[Bur94] J. Burr and J. Shott, "A 200mV Self-Testing Encoder/Decoder using Stanford Ultra-Low-Power CMOS," *Proceedings of the International Solid State Circuits Conference '94*, San Francisco, CA, pp. 84-85, February 1994.

[Bur92b] D. Bursky, "A Tidal Wave of 3-V IC's Opens Up Many Options; Logic Chips that Operate at 3V or Less Offer a Range of Low-Power Choices for Portable Systems," *Electronic Design*, vol. 40, no. 17, pp. 37-45, August 20, 1992.

[Cas94] B. Case, "Low-Power Design, Pentium Dominate ISSCC," *Microprocessor Report*, vol. 8, no. 4, pp. 26-28, March 28, 1994.

[Cha94a] A. Chandrakasan, A. Burstein and R. Brodersen, "A Low Power Chipset for Portable Multimedia Applications," *Proceedings of the International Solid-State Circuits Conference '94*, San Francisco, CA, pp. 82-83, February 1994.

[Cha94b] A. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen, "Design of Portable Systems," *Proceedings of CICC '94*, San Diego, May 1994.

[Cha92a] A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-Power Techniques for Portable Real-Time DSP Applications," *VLSI Design '92*, India, pp. 203-208, 1992.

[Cha92b] A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-Power CMOS Design," *IEEE Journal of Solid-State Circuits*, pp. 472-484, April 1992.

[Cha93] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Brodersen, "Optimizing Power Using Transformations," submitted to *IEEE Transactions on Computer-Aided Design*, May 1993.

[Cha94] K. Chao and D. Wong, "Low Power Considerations in Floorplan Design," *1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 45-50, April 1994.

[Cho90] S. Chowdhury and J. Barkatullah, "Estimation of Maximum Currents in MOS IC Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 6, pp. 642-654, June 1990.

[Cir87] M. A. Cirit, "Estimating Dynamic Power Consumption of CMOS Circuits," *Proceedings IEEE International Conference on Computer Aided Design*, pp. 534-537, November 1987.

[Col93] B. Cole, "At CICC: Designers Face Low-Power Future," *Electronic Engineering Times*, no. 745, pp. 1-2, May 10, 1993.

[Cro75] R. Crochiere, A. Oppenheim, "Analysis of Linear Networks," *Proceedings of the IEEE*, Vol. 63, No. 4, pp. 581-595, 1975.

[Dah91] D. Dahle, "Designing High Performance Systems to Run from 3.3V or Lower Sources," *Silicon Valley Personal Computer Conference*, pp. 685-691, 1991.

[Dem90] H. De Man, F. Catthoor, G. Goossens, J. Vanhoof, J. Van Meerbergen, S. Note, and J. Huisken, "Architecture-Driven Synthesis Techniques for Mapping Digital Signal Processing Algorithms into Silicon," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 319-335, Feb. 1990.

[Den88] A. Deng, Y. Shiau, and K. Loh, "Time Domain Current Waveform Simulation of CMOS Circuits," *International Conference on Computer-Aided Design '88*, pp. 208-211, 1988.

[Den94] A. Deng, "Power Analysis for CMOS/BiCMOS Circuits," *1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 3-8, April 1994.

[Den74] R. Dennard et al., "Design of Ion-Implanted MOSFETS with Very Small Physical Dimensions," *IEEE Journal of Solid State Circuits*, SC-9, pp. 256-258, 1974.

[Dev92] S. Devadas, K. Keutzer, and J. White, "Estimation of Power Dissipation in CMOS Combina-

tional Circuits Using Boolean Function Manipulation," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 3, March 1992.

[Dob92] D. Dobberpuhl et al., "A 200MHz, 64b, Dual Issue CMOS Microprocessor," *Digest of Technical Papers, ISSC '92*, pp. 106-107, 1992.

[Don79] W. Donath, "Placement and Average Interconnection Lengths of Computer Logic," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 4, pp. 272-277, April 1979.

[Dun93] P. Duncan, S. Swamy, and R. Jain, "Low-Power DSP Circuit Design Using Retimed Maximally Parallel Architectures," *Proceedings of the 1st Symposium on Integrated Systems*, pp. 266-275, March 1993.

[Eag92] J. Eager, "Advances in Rechargeable Batteries Spark Product Innovation," *Proceedings of the 1992 Silicon Valley Computer Conference*, Santa Clara, CA, pp. 243-253, Aug. 1992.

[Eng89] G. Engel, R. Morely, S. Kwa, and R. Fretz, "Integrated Circuit Logarithmic Digital Quantizers with Applications to Low-Power Data Interfaces for Speech Processing," *VLSI Signal Processing IV*, pp. 230-238, 1989.

[Erc89] S. Ercolani et al., "Estimate of Signal Probability in Combinational Logic Networks," *IEEE European Test Conference*, pp. 132-138, 1989.

[Erd89] D. Erdman, G. Nifong, R. Subrahmanyan, and S. Kenkey, "CAzM: A Circuit Analyzer with Macromodeling," *Technical Report, Microelectronics Center of North Carolina*, 1989.

[Feu82] M. Feuer, "Connectivity of Random Logic," *IEEE Transactions on Computing*, vol. C-31, pp. 29-33, Jan. 1982.

[Gal93] J. Gallant, "Low-Power Microprocessors Simplify Design of Portable Computers," *EDN*, vol. 38, no. 9, pp. 39-44, April 29, 1993.

[Gho92] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits," *Proceedings of the 29th Design Automation Conference*, pp. 253-259, June 1992.

[Gup63] S. Gupta, "Bibliography on the Multivariate Normal Integrals and Related Topics," *Annals of Mathematical Statistics*, pp. 829-838, 1963.

[Hil85] P. Hilfinger, "A High-Level Language and Silicon Compiler for Digital Signal Processing," *Proceedings of Custom Integrated Circuits Conference*, Los Alamitos, CA, pp. 213-216, 1985.

[Hop90] B. Hoppe, "Optimization of High Speed CMOS Logic Circuits, with Analytical Model for Signal Delay, Chip Area, and Dynamic Power Dissipation," *IEEE Transactions on Computer Design*, pp. 236-247, March 1990.

[Hui90] C. Huizer, "Power Dissipation Analysis of CMOS VLSI Circuits by Means of Switch-Level Simulation," *Proceedings of the European Solid-State Circuits Conference '90*, pp. 61-64, 1990.

[Jag90] U. Jagau, "SIMCURRENT - An Efficient Program for the Estimation of the Current Flow of Complex CMOS Circuits," *Proceedings of International Conference on Computer-Aided Design '90*, pp. 396-399, 1990.

[Jay84] N. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall Signal Processing Series, 1984.

[Kad81] A. Kader and A. Talbot, "Inversion of Polynomial Network Matrices with Particular Reference to Sensitivty Analysis," *IEE Proceedings*, vol. 128, Part G, no. 4, pp. 170-172, August 1981.

[Kak90] M. Kakumu and M. Kinugawa, "Power-Supply Voltage Impact on Circuit Performance for Half and Lower Submicrometer CMOS LSI," *IEEE Transactions on Electron Devices*, vol. 37, no. 8, pp. 1902-1908, Aug. 1990.

[Kal94] M. Kallu, *ENPOWER: A Vector-Driven Power Estimation Tool for CMOS Logic Circuits*, Masters Thesis, ERL UC Berkeley, 1994.

[Kan86] S. M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits," *IEEE Journal of Solid-State Circuits*, pp. 889-891, October 1986.

[Kro91] T. Krodel, "PowerPlay - Fast Dynamic Power Estimation Based on Logic Simulation," *Proceedings of International Conference on Circuit Design '91: VLSI in Computers*, pp. 96-100,

1991.

[Kur89] F. Kurdahi and A. Parker, "Techniques for Area Estimation of VLSI Layouts," *IEEE Transactions on Computer-Aided Design*," vol. 8, no. 1, pp. 81-92, January 1989.

[Lan71] B. Landman and R. Russo, "On a Pin Versus Block Relationship for Partitions of Logic Graphs," *IEEE Transactions on Computing*, vol. C-20, pp. 1469-1479, Dec. 1971.

[Lan90] P. Landman, A. Chandrakasan, and A. Kalavade, "A Low-Power VLSI Color Space Translator," Final Project, CS 250, U. C. Berkeley, December 1990.

[Lan91] P. Landman, *High Quality, Low Bit-Rate Speech Coding for Low-Power VLSI Implementation*, Masters Thesis, ERL UC Berkeley, May 1991.

[Lan93] P. Landman and J. Rabaey, "Power Estimation for High Level Synthesis," *Proceedings of EDAC-EUROASIC '93*, Paris, France, pp. 361-366, February 1993.

[Lan94] P. Landman and J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis," *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 165-170, April 1994.

[Lav90] L. Lavagno, S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli, "MIS-MV: Optimization of Multi-Level Logic with Multiple-Valued Inputs," *Proceedings of 1990 IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, pp. 560-563, November 1990.

[Lee86] E. Lee, *A Coupled Hardware and Software Architecture for Programmable Digital Signal Processors*, Ph.D. Dissertation, University of California, Berkeley, 1986.

[Lee94] W. Lee, U. Ko, and P. Balsara, "A Comparative Study on CMOS Digital Circuit Families for Low-Power Applications," *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 129-132, April 1994.

[Lem94] C. Lemonds and S. Mahant Shetti, "A Low Power 16 by 16 Multiplier Using Transition Reduction Circuitry," *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 139-142, April 1994.

[Len93] C. Lennard, *Personal Communication*, Berkeley, CA, March 1993.

[Lid94] D. Lidsky and J. Rabaey, "Low-Power Design of Memory Intensive Functions Case Study: Vector Quantization," *VLSI Signal Processing Workshop*, San Diego, CA, 1994.

[Lip93] P. Lippens, J. Van Meerbergen, W. Verhaegh, and A. Van Der Werf, "Allocations of Multiport Memories for Hierarchical Data Streams," *Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, pp. 728-735, Nov. 1993.

[Liu93] D. Liu and C. Svensson, "Trading Speed for Low Power by Choice of Supply and Threshold Voltages," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 1, pp. 10-17, 1993.

[Mal92] D. Maliniak, "Better Batteries for Low-Power Jobs," *Electronic Design*, vol. 40, no. 15, pp. 18, July 23, 1992.

[Man91] D. Manners, "Portables Prompt Low-Power Chips," *Electronics Weekly*, no. 1574, pp. 22, Nov. 13, 1991.

[Man92] D. Manners, "Portability the Key to Power Struggle," *Electronics Weekly*, no. 1618, pp. 21, Nov. 18, 1992.

[Mar89] A. Martinez, "Quick Estimation of Transient Currents in CMOS Integrated Circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 520-531, April 1989.

[May92] J. Mayer, "Designers Heed the Portable Mandate," *EDN*, vol. 37, no. 20A, pp.65-68, November 5, 1992.

[Mel93] J. Mello and P. Wayner, "Wireless Mobile Communications," *Byte*, vol. 18, no. 2, pp. 146-153, Feb. 1993.

[Meh94] R. Mehra, "High-Level Power Estimation and Exploration," submitted to *1994 International Workshop on Low Power Design*, April 1994.

[Mob94] -, "Mobile Madness: CCD Becomes PC," *LAN Magazine*, vol. 9, no. 1, pp. 46-48, Jan. 1994.

[Mon93] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming Sequential Circuits for Low Power," *Proceedings of the International Conference on Computer-Aided Design*, pp. 398-402, November 1993.

[Mul86] Muller and Kamins, *Device Electronics for Integrated Circuits*, John Wiley & Sons, 1986.

[Mül91] K. Müller-Glaser, K. Kirsch, and K. Neusinger, "Estimating Essential Design Characteristics to Support Project Planning for ASIC Design Management," *IEEE International Conference on Computer-Aided Design '91*, Los Alamitos, CA, pp. 148-151, November 1991.

[Mur94] R. Murgai, R. Brayton, A. Sangiovanni-Vincentelli, "Decomposition of Logic Functions for Minimum Transition Activity," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 33-38, April 1994.

[Nag75] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Technical Report ERL-M520*, University of California, Berkeley, 1975.

[Nag94] C. Nagendra, U. Mehta, R. Owens, and M. Irwin, "A Comparison of the Power-Delay Characteristics of CMOS Adders," *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 231-236, April 1994.

[Naj90] F. Najm, I. Hajj, and P. Yang, "Probabilistic Simulation for Reliability Analysis of CMOS VLSI Circuits," *IEEE Transactions on Computer-Aided Design*, pp. 439-450, April 1990.

[Naj91a] F. Najm, "Transition Density, a Stochastic Measure of Activity in Digital Circuits," *Proceedings of the 28th Design Automation Conference*, pp. 644-649, June 1991.

[Naj91b] F. Najm, I. Hajj, and P. Yang, "An Extension of Probabilistic Simulation for Reliability Analysis of CMOS VLSI Circuits," *IEEE Transactions on Computer-Aided Design*, pp. 1372-1381, November 1991.

[Nas93] R. Nass, "Energy Star Program Gets a Boost from Low-Power CPUs," *Electronic Design*, vol. 41, no. 13, pp. 38-40, June 24, 1993.

[Nie94] L. Nielsen and J. Sparso, "Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage," *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp. 99-104, April 1994.

[Pat90] D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[Pha94] D. Pham et al., "A 3.0W 75SPECint92 85SPECfp92 Superscalar RISC Microprocessor," *Proceedings of the International Solid-State Circuits Conference '94*, San Francisco, CA, pp. 212-213, February 1994.

[Piv94] D. Pivin, "Pick the Right Package for Your Next ASIC Design," *EDN*, vol. 39, no. 3, pp. 91-108, Feb. 3, 1994.

[Pot93] M. Potkonjak and J. Rabaey, "Exploring the Algorithmic Design Space using High Level Synthesis," *VLSI Signal Processing VI*, pp. 123-131, 1993.

[Pou93] D. Pountain, "Computing without Clocks: Asynchronous Operation Could Signal a New Era in Portable, Low-Power Computing," *Byte*, vol. 18, no. 1, pp. 145-149, Jan. 1993.

[Pow90] S. R. Powell and P. M. Chau, "Estimating Power Dissipation of VLSI Signal Processing Chips: The PFA Technique," *VLSI Signal Processing IV*, pp. 250-259, 1990.

[Pow91] S. Powell and P. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips," *IEEE Transactions on Circuits and Systems*, pp. 646-650, June 1991.

[Pre88] B. Preas and M. Lorenzetti, eds., *Physical Design Automation of VLSI Systems*, Menlo Park, CA, 1988.

[Pre81] D. A. Preece, "Distributions of the Final Digits in Data," *The Statistician*, vol. 30, no. 1, pp. 31-60, March 1981.

[Rab91] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures," *IEEE Design & Test of Computers*, pp. 40-51, June 1991.

[Rab94] J. Rabaey, *DAC Tutorial*, San Diego, CA, May 1994.

[Rab95] J. Rabaey, "Digital Integrated Circuits: A Design Perspective", Prentice Hall, Englewood Cliffs, N.J., to be published Spring 1995, currently available as EE141 Course Reader, U.C. Berkeley.

[Rab93] J. Rabaey and L. Guerra, "Exploring the Architecture and Algorithmic Space for Signal Processing Applications," *Technical Digest of International Conference on VLSI and CAD*, Taejon,

Korea, pp. 315-319, Nov. 1993.

[Rab91] J. M. Rabaey and M. Potkonjak, "Complexity Estimation for Real Time Application Specific Circuits", *Proceedings of IEEE ESSCIRC Conference*, Sept. 1991.

[Sal83] R. Saleh, J. Kleckner, and A. Newton, "Iterated Timing Analysis and SPLICE1," *Proceedings of ICCAD*, 1983.

[Sal89] A. Salz and M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," *Proceedings of the 26th Design Automation Conference*, pp. 173-178, 1989.

[Sch94] J. Schutz, "A 3.3V 0.6μm BiCMOS Superscalar Microprocessor," *Proceedings of the International Solid-State Circuits Conference '94*, San Francisco, CA, pp. 202-203, February 1994.

[Set85] S. Seth, L. Pan, and V. Agrawal, "PREDICT - Probabilistic Estimation of Digital Circuit Testability," *Proceedings of IEEE 15th Annual International Symposium on Fault-Tolerant Computing*, Ann Arbor, MI, pp. 220-225, June 1985.

[She92a] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," *Proceedings of the International Conference on Computer-Aided Design*, pp. 402-407, November 1992.

[She92b] S. Sheng, A. Chandrakasan, and R. Brodersen, "A Portable Multimedia Terminal," *IEEE Communications Magazine*, pp. 64-75, December 1992.

[She92c] P. Sherer and V. McCarthy, "Microsoft, Intel Tout Spec for Low-Power PCs," *PC Week*, vol. 9, no. 2, pp. 1-2, Jan 13, 1992.

[Sma94] C. Small, "Shrinking Devices Put the Squeeze on System Packaging," *EDN*, vol. 39, no. 4, pp. 41-46, Feb. 17, 1994.

[Sor87] G. Sorkin, "Asymptotically Perfect Trivial Global Routing: A Stochastic Analysis," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 820, 1987.

[Sta94] M. Stan and W. Burleson, "Limited-Weight Codes for Low-Power I/O," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 209-214, April 1994.

[Sto91] A. Stolzle, *A Real-Time Large Vocabulary Connected Speech Recognition System*, Ph.D. Dissertation, U. C. Berkeley, 1991.

[Str94] A. Stratakos, R. W. Brodersen, and S. R.Sanders, "High-Efficiency Low-Voltage DC-DC Conversion for Portable Applications," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 105-110, April 1994.

[Swe93] J. Sweeney and K. Ulery, "3.3V DSPs for Multimedia and Communications Products; Designers Harness Low-Power, Application-Specific DSPs," *EDN*, vol. 38, no. 21A, pp. 29-30, Oct. 18, 1993.

[Sve94] C. Svensson and D. Liu, "A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 171-176, April 1994.

[Ter83] C. Terman, *Simulation Tools for Digital LSI Design*, Ph.D. Dissertation, MIT, September 1983.

[Tiw93] V. Tiwari, P. Ashar, and S. Mailk, "Technology Mapping for Low Power," *Proceedings of the 30th Design Automation Conference*, Anaheim, CA, pp. 74-79, June 1993.

[Tja89] R. Tjarnstrom, "Power Dissipation Estimate by Switch Level Simulation," *Proceedings IEEE International Symposium on Circuits and Systems*, pp. 881-884, May 1989.

[Tse93] E. Tsern, T. Meng, and A. Hung, "Video Compression for Portable Communications Using Pyramid Vector Quantization of Subband Coefficients," *Proceedings of IEEE Workshop on VLSI Signal Processing*, Veldhoven, pp. 444-452, October 1993.

[Tsu93] C.-Y. Tsui, M. Pedram, and A. Despain, "Efficient Estimation of Dynamic Power Consumption Under a Real Delay Model," *Proceedings of the International Conference on Computer-Aided Design '93*, pp. 224-228, 1993.

[Tsu94] C.-Y. Tsui, M. Pedram, and A. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation," *Proceedings of the 30th Design Automation Conference*, Anaheim, CA, pp. 68-73, June 1993.

[Tya87] A. Tyagi, "Hercules: A Power Analyzer for MOS VLSI Circuits," *Proceedings of the Interna-*

tional Conference on Computer-Aided Design '87, pp. 530-533, Nov. 1987.

[Vai93] H. Vaishnav and M. Pedram, "PCUBE: A Performance Driven Placement Algorithm for Lower Power Designs," *Proceedings of the Euro-DAC '93*, pp. 72-77, September 1993.

[Van93] P. Van Oostende, P. Six, J. Vandewalle, and H. De Man, "Estimation of Typical Power of Synchronous CMOS Circuits Using a Hierarchy of Simulators," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 1, pp. 26-39, Jan. 1993.

[Van92] P. Van Oostende, *Techniques and Tools for Power and Interconnect Delay Analysis of CMOS VLSI Circuits*, Ph.D. Dissertation, K. U. Leuven, Belgium, December 1992.

[Vee84] H. J. M. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits," *IEEE Journal of Solid-State Circuits*, pp. 468-473, August 1984.

[War84] J. Ward et al., "Figures of Merit for VLSI Implementations of Digital Signal Processing Algorithms," *Proceedings of the Institute of Electrical Engineers*, vol. 131, Part F, pp. 64-70, February 1984.

[Wat89] R. K. Watts, ed., *Submicron Integrated Circuits*, John Wiley & Sons, New York, 1989.

[Web94] S. Weber, "The Low-Power Brokers," *Electronic Engineering Times*, no. 782, pp. 18-20, Jan. 31, 1994.

[Wes88] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 1988.

[Whi92] J. White, S. Devadas, and K. Keutzer, "Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation," *IEEE Transactions on Computer-Aided Design*, pp. 377-383, March 1992.

[Wid75] Widrow, et. al., "Adaptive Noise Cancelling: Principles and Applications," *Proceedings of the IEEE*, Vol. 63, No. 12, December 1975.

[Wil92] R. Wilson, "Phones on the Move; Pocket Phone Sales are on Line for a Boom," *Electronics Weekly*, no. 1606, pp. 25, Aug. 12, 1992.

[Won89] D. Wong, G. De Micheli, and M. Flynn, "Inserting Active Delay Elements to Achieve Wave Pipelining," *1989 IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, pp. 270-273, 1989.

[Wuy94] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, and H. De Man, "Global Communication and Memory Optimizing Transformations for Low Power Systems," *1994 International Workshop on Low-Power Design*, Napa Valley, CA, pp. 203-208, April 1994.

[Yac89] G. Y. Yacoub and W. H. Ku, "An Accurate Simulation Technique for Short-Circuit Power Dissipation Based on Current Component Isolation," *Proceedings IEEE International Symposium on Circuits and Systems*, pp. 1157-1161, May 1989.

[Yeu92] A. Yeung and J. Rabaey, "A Reconfigurable Data-Driven Multi-Processor Architecture for Rapid Prototyping of High Throughput DSP Algorithms," *Proceedings of the 26th Hawaii International Conference on System Sciences*, vol. 1, pp. 169-178, January 1993.

[Yeu94] N. Yeung et al., "The Design of a 55SPECint92 RISC Processor under 2W," *Proceedings of the International Solid-State Circuits Conference '94*, San Francisco, CA, pp. 206-207, February 1994.