HIGH-LEVEL ESTIMATION AND SYNTHESIS TECHNIQUES FOR LOW-POWER DESIGN

by

Renu Mehra

Memorandum No. UCB/ERL M97/37

28 May 1997

HIGH-LEVEL ESTIMATION AND SYNTHESIS TECHNIQUES FOR LOW-POWER DESIGN

by

Renu Mehra

Memorandum No. UCB/ERL M97/37

28 May 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

High-Level Estimation and Synthesis Techniques for Low-Power Design

by

Renu Mehra

Doctor of Philosophy in

Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Jan M. Rabaey, Chair

The explosive growth in the computational requirements imposed on current day digital systems and the rapid proliferation of portable devices have made low power a critical design issue. Low-power studies published in the literature indicate that large power savings are possible by addressing this problem at the higher — algorithm and architecture — levels of abstraction. In fact, high-level design tools and methodologies are becoming increasingly important due to the integration of tens of millions of transistors on single chips and narrowing time-to-market windows.

This dissertation presents automated techniques and methodologies for power reduction at the algorithm and architecture levels of abstraction. The core contributions include mechanisms for power estimation from a behavioral description, and architecture synthesis techniques for low-power design.

The key contributions in the first part of this work are techniques for *algorithm-level power estimation*. The estimates are based on information from an architecture model and a specified hardware library, and are hence *technology-targeted*. Each of the different components of power dissipation on a chip is considered separately, and a combination of analytic and stochastic schemes is proposed. The estimation methods are encapsulated in an exploration framework that allows the user to quickly

quickly evaluate several points in the algorithmic design space without synthesizing each one.

The primary contributions in the second part of this work are *architecture-synthesis schemes* targeting interconnect power reduction. We propose two techniques that are based on exploiting algorithm properties for reducing power. The properties considered are the algorithm's *spatial locality,* which refers to the existence of tightly connected substructures in it, and its *regularity,* which refers to the repetition of computational patterns in it. The synthesis approaches suggested exploit these properties to derive a simpler interconnect infrastructure with shorter buses and lower multiplexor and buffer overhead.

The concepts and ideas developed in this thesis have been embodied in a synthesis system called *Synergy* which allows the user to explore the algorithmic design space and synthesize the design to a low-power architecture in an integrated way.

[illegible]

Jan M. Rabaey
Committee Chairman

To my loving parents, Subhashini and Surinder Mehra

and

my dearest husband, Rajeev.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The realization of this thesis has been possible due to the encouragement, guidance, influence, and support from several people. Without them, this work could not have achieved the state it has today.

First, I would like to express my gratitude to my research advisor, Prof. Jan Rabaey, for his guidance and vision. He spent a lot of time with me in defining the project during the early stages, and giving many comments and criticisms as my ideas matured. He always encouraged me to step back and look at the bigger picture. I would also like to thank Prof. Robert Brodersen for heading the infopad project which paid my salary for most of my stay here. Both he and Jan have provided us with incredible research resources, support staff, and other fun things like our weekly lunch meetings and semi-annual retreats at very nice locations.

Prof. John Wawrzynek, Prof. Richard Newton, and Prof. Dorit Hochbaum served on my qualifying examination committee and gave me feedback on my work. Prof. Wawrzynek and Prof. Hochbaum also kindly agreed to read this 200 page thesis for me. I would like to express my sincere thanks to them. I am extremely grateful to Prof. Van Duzer who was co-advisor for my master's thesis. I would like to thank ARPA and the Regents scholarship for providing financial support for my stay here.

I would like to express my thanks to my research collaborators: the power estimation work was done with Anantha Chandrakasan, the design space exploration case study was done with Paul Landman and the work on exploiting spatial locality was done with Lisa Guerra. They spent several hours brainstorming with me on various ideas. I would also like to thank the entire Hyper team for their support.

The research and administrative staff in Cory hall and in the BJgroup have been a great help in making my stay at Berkeley smooth and pleasant, and I would like to acknowledge all of them — Tom Boot, Heather Brown, Fred Burghardt, Ruth Gjerde, Dierdre McAulife, Sue Mellers, Elise Mills, Brian Richards, Corey Schaffer, Carol Sitea, Genevieve Theibaut, and Kevin Zimmerman.

My stay in Berkeley has been a really enjoyable one due to several friends who made this a fun place, helped me through several stressful situations, gave me company during long hours of work at Cory, and several times dropped me home in the wee hours of the morning! They have prevented this entire process from becoming an ordeal and kept my life interesting at all times. I would like to thank all of them — my cubicle mates Ole Bentz, Stephen Chan, Lisa Guerra, and Sean Huang, for providing a friendly atmosphere in the cubicle, Arthur Abnous who has recently shaved his hair and has been persuading everyone (including my husband) to follow, Randy Allmon who brought us plastic toys from DEC, Tom Burd the only Cowboy fan in the Bay Area and the in-house bookie for the NCAA tournaments, Dominique Borrione who became a very good friend during her short stay in USA, Premal Buch who drove his Toyota Corolla like a Lexus — hope he gets his dream car soon!, Andy Burstein who gave me great cooking tips, Anantha Chandrakasan who used to steal potato chips from my desk, Edoardo Charbon who threw great pizza and pasta parties, Vijay and Kamal Chaudhary — we could always drop in at their place for food without any notice, Miguel Corazao and Mario Khalaf the inseparable duo, Stephen Edwards who had mastered the art of irritating anyone he wanted to, Varghese George who treats his motorcycle like a Ferarri and explains his wrinkles as "laughter lines", Jeff Gilbert the local coffee supplier who never missed a chance to take a dig at anyone, Rick Han who gave me lots of interviewing advice, Harry Hsieh who introduced me to ballroom dancing, Desmond Kirkpatrick who gave me inside info about Intel (just kidding), Sriram Krishnan whose frank opinions often got him into trouble, Chris Lennard who threw a hot buffalo wing party every year, David Lidsky who consulted along with me for NTU, Paul Landman who is one of the most organized people I have known, Kathy Lu who is one of the first people I met at Berkeley, Rajeev Murgai who used to be in Cory at all hours of the night until he finally got married, Shankar Narayanaswamy who had good advice for all occasions (from how to buy a car to how to look for jobs), Satyajit Ranganathan who had an incredible memory — with at least 1GByte of cache, Jaijeet Roychowdhury who has not been able to keep away from Berkeley even after graduating years ago, Jagesh and Alpa Sanghavi who were always having parties with fantastic food, Marco Sgroi who took great pains to teach us Italian, Sam Sheng the computer guru, Tom Shiple whose well-organized collection of CAD journals and conference proceedings allowed us to find reference material at all odd hours, Sangwan Son and Adam Eldredge who TAed for my 141 class, Mani Srivastava who gave me great career advice, Gitanjali Swamy who has refused to leave me alone for the last 10 years, Steven Stoiber who drove a Fiat Spader with no rearview, Marlene Wan who had the most amusing philosophy of "beating-up" George to relieve stress!, Scarlett Wu who introduced me to salted prunes, and all the people in my group and in the neighbouring groups in 550 Cory who have helped me at various times and brought character to 550 — Heather Bowers, Dev Chen, Jennie Chen, Nelson Chow, Cormac Conroy, Ian O'Donnell, Richard Edell, Eva Endress, Eric Kusse, Eric Ng, Peggy Laramie, Sovarong Leang, Lapoe Lynn, Monte Mar, Keith Onodera, Trevor Pering, David Pini, Vason Srini, Kevin Stone, Jane Sun, Roy Sutton, Tom Truman, Tony Stratakos, Ingrid Verbauwhede, Dennis Yee, Engling Yeo, and Alfred Yeung.

All the members of the Berkeley's famous CAD group deserve special mention since they always made me feel welcome to their group parties, picnics and get-togethers and were especially nice to me — Prof. Robert Brayton and his wife Ruth, Adnan Aziz, Zeina Daoud, Dan Engels, Naji Ghazal, Paulo Guisto, Adrian Isles, Sunil Khatri, Yuji Kukimoto, Enrico Malavasi, Gurmeet Singh Manku, Amit Mehrotra, Amit Narayan, Andre Neiuwland, Mukul Ranjan Prasad, Flora Oviedo, Shaz Qadeer, Sriram Rajamani, Vigyan Singhal, and Serdar Tesiran.

My family — my parents and my sisters, Vineeta and Radhika, — has been instrumental in making me what I am today, and nothing I can say will express my appreciation to them completely. They have shaped my ideas and my values, they always encouraged me to do my very best, and they have provided the secure foundations and support from which I have derived my stability and confidence in life.

Finally, this thesis has been possible due to the encouragement and support of my beloved husband, Rajeev, whom I can never thank enough. He has proof-read my papers and this thesis, brainstormed with me on several of my ideas, helped me prepare for my presentations and job search, and has had a huge impact on my work. His patience, hard work, and determination in all situations have been my inspiration, and his love, understanding, and support have been my source of strength.

1 Introduction

For many years, area and performance were the only criteria used in the design of commercial integrated circuits. Analysis of power consumption was performed only as an afterthought with the results being used to determine packaging requirements rather than to drive any design optimizations. Recently, however, power consumption has begun to play an increasingly important role in digital systems. The principle driver for this has been the explosive increase in the demand for portable electronics such as personal digital assistants (PDAs), laptops, and personal communicators. There has also been an increased desire for low power in the high-performance computing market, motivated by reliability and cost issues associated with packaging and cooling high-power devices.

Existing computer-aided design tools for low-power design focus primarily on low levels of abstraction. For example, power analysis tools at the gate and circuit levels are widely available and logic synthesis tools targeted at low power are beginning to appear as well. Unfortunately, the higher levels of abstraction, where the most significant optimizations are possible, have not been explored as much.

In this dissertation, we consider automated computer-aided design techniques for power reduction in digital electronic systems. Since it is generally accepted that the largest gains are obtained by working at the highest levels of abstraction, the approach adopted in this thesis proposes optimizations and design aids at the algorithm and architecture levels, identifying and addressing some critical issues. Both the power estimation and synthesis problems are studied and proposed techniques are implemented in an automated design space exploration and synthesis system, *Synergy*.

1.1. Low-power design

The last ten years have seen an explosive growth in the computational requirements imposed on digital electronic devices. Current consumer applications demand complexities and speeds originally seen only in the realm of large powerful mainframes. Personal computers are now equipped with sophisticated graphics, imaging, and video capabilities and run at over 100 MHz speeds. The problem is that the increasing complexity and speed are accompanied by growing power dissipation. Table 1.1, which shows the speed, complexity (number of transistors), and power consumption of some recent microprocessors, indicates the enormity of this problem. Increased power dissipation results in reliability problems due to over-heating, and requires expensive packaging and sophisticated heat removal techniques. This additional cost is not as easily absorbed in the personal computer market as in the high-priced supercomputer business. In fact, in order to avoid high-cost packaging and cooling requirements, some processors now have advanced heat detection schemes that slow down the processor's clock if the chip gets too hot [107].

Another important development in recent years has been the rapid increase in the deployment of portable versions of previously tethered consumer products. While original portable applications were limited to low-speed devices with restricted functionality like wrist-watches and pocket calculators, today's portable devices provide infinitely more functionality at much higher speeds. Typical examples are laptop computers, which now form the fastest growing segment of the computer market, personal digital assistants (PDAs), cellular phones, and pagers. The consumer-oriented nature of these devices solicits user-friendly environments with graphics, video, and audio capabilities and speech and handwriting based user-interfaces, all of which put addi-

Table 1.1. Power consumptions of a few recent microprocessors.

Microprocessor	Clock speed (MHz)	Voltage (V)	Number of transistors (millions)	Year	Power (W)
Pentium Pro[1] (Intel)	150	3.1	5.5	1996	29.4
Pentium with MMX[2] (Intel)	300	2.8	7.5	1997	7.8
PA-RISC 8000[3] (HP)	180	-	3.9	1996	>40
UltraSparc I[2] (Sun Microsystems)	167	3.3	5.2	1995	<30
PowerPC 604[1] (IBM, Motorola)	133	3.3	3.6	1995	17.5
PowerPC[2] (Exponential Technology)	533	3.6, 2.1	2.7	1997	<85
Alpha 21064[2] (DEC)	300	3.3	9.3	1995	30
Alpha 21264[2] (DEC)	600	2.0	15.2	1997	72

1. Numbers from vendor information.
2. Numbers from the *International Solid State Circuits Conference.*
3. Numbers from *Compcon.*

tional computational and speed requirements on the systems. While the computational requirements have grown considerably, the growth in battery technology has been relatively slow. This has resulted in a critical need for power reduction techniques to enable batteries to support these applications for reasonable periods of time.

Historically, power issues have been a limiting factor in determining the capabilities of digital electronic systems and have only been solved through key technology advances. The computing power of vacuum tube computers was limited by reliability and heat dissipation problems. The advent of the transistor [6, 112] and of wafer integration techniques [89] introduced devices with orders of magnitude lower power, and techniques to fabricate a large number of them on a single die. The bipolar based TTL and ECL logic families ruled the digital semiconductor market through the 70's. This technology was again limited by power dissipation problems and was replaced by CMOS/NMOS devices in the 80's. Today however, no radically new low-power device structures are in the horizon to address the power problems facing the digital electronics community. Therefore, it is critical for designers to limit the power requirements *through careful design techniques and methodologies*. Over the last five years this topic has gathered increased momentum leading to a large body of research work in low-power design.

1.2. High-level design

The surge of interest in low power has spawned numerous research efforts into design techniques for reducing power consumption. Several papers have surveyed the work in this field [27, 85, 99, 115]. In particular, designers have reported low-power strategies at the algorithm and architecture levels that promise orders of magnitude savings in power [18, 56, 80, 115], while published results based on gate and circuit-level optimizations typically offer only a factor of two or less improvement [27, 56]. The reason is that, at the later stages of a design, many of the decisions are already made which limit the opportunities for optimization at the lower levels of abstraction. The greater degree of freedom at the higher levels allows for much larger power savings. This suggests that a top-down approach should be adopted for low-power design. Specifically, optimization efforts should begin at the algorithm level, proceeding then to the architecture level, and finally to the gate, circuit, and layout levels of abstraction.

Another motivation for high-level design tools comes from the widening gap between the design complexities and designer productivity. While the number of transistors per chip has gone up by about a 1000x in the last 15 years (from 10,000 in 1981 to about 10 million in 1996), the number of transistors designed per staff-month has gone up by only about 20x in the same period (from about 100 to 2000) [110]. By the year 2010, the number of transistors per chip is expected to rise to 1 billion! This indicates an explosion in the number of designers required in a design team for a single chip. Clearly there is a need for automated approaches to address this problem. In particular, more effective tools are required at the high levels of abstraction, especially the algorithm and architecture levels. At these levels, most of the work has focussed on speed and area optimization, and the power optimization arena has remained largely unexplored.

1.3. Contributions of this work

The key contributions of this work include techniques for estimating power at the algorithm level and an associated exploration facility, as well as a set of synthesis techniques aimed at reducing interconnect power with minimal area penalty.

Several power optimizations techniques have been proposed at the algorithm level (an overview is presented in Chapter 2). Though these techniques have a large potential for power reduction, it is seen that the power savings obtained are design dependent, the techniques have different effects on the different design components, and in several cases they work better in combination with other techniques than alone. A critical design aid at this level is a power estimation mechanism that allows the user to apply various techniques and evaluate their effects on the overall power without synthesizing the entire design. The challenge lies in providing reasonably accurate predictions in the face of extremely limited information. One of the key contributions of this work is a facility to provide such predictions. A set of *heterogeneous estimation techniques* are presented to address the different characteristics of the various power-consuming components on a chip. While for some components, such as functional units and memory, the power can be computed analytically through algorithm analysis, other components like registers, interconnect, and control require stochastic techniques for power prediction.

To further aid in evaluating the different degrees of freedom and making design decisions at the algorithm level, the estimation algorithms are encapsulated into a *design space exploration tool* called Explore.

The second key contribution of this thesis includes *architecture-synthesis techniques* for power reduction. While some synthesis approaches to power optimization have been proposed, most of them are aimed at reducing signal activity at the inputs of the various modules in the final architecture. Comparing manual and synthesized implementations, we identified that the interconnect component is a significant power bottleneck in automated approaches. Also, this component constitutes overhead power (not necessary for the basic computation) and is significantly lower in manual designs. Our architecture synthesis approaches capture some of the techniques used by human designers and uses them in *automated schemes to reduce the interconnect power component.*

The core contribution in this part is the identification and exploitation of two structural properties of the algorithm for power reduction. First, we identify tightly connected sub-structures in the algorithm and use them to generate a localized implementation. The existence of tightly connected sub-structures in the algorithm is referred to as its *spatial locality*. Exploiting spatial locality in this way restricts the interconnect elements to portions of the chip, and helps in reducing the physical capacitance and power dissipation in the buses and buffers. Additionally, the localized hardware sharing reduces multiplexor overheads. Next, we identify repeated patterns of computation in the algorithm, which constitute its *regularity*, and use them to synthesize a simpler interconnect structure. In this case, the interconnect elements, once instantiated, can be reused several times without extra multiplexing overhead. The technique results in power reductions in the buses and multiplexors.

The concepts and ideas introduced in this thesis are encapsulated in a synthesis system called *Synergy* which is targeted at exploiting the large power savings available at the higher levels of abstraction. At the highest level, the algorithm level, power estimation techniques encapsulated in a design space exploration tool provide the user with enhanced capabilities to evaluate designs and explore the algorithmic design space. After an algorithm with appropriate parameters is chosen, architecture synthesis techniques allow the designer to realize a low-power implementation. In the end, architectural power analysis can be used to verify and fine-tune the decisions.

1.4. Scope

For implementation and demonstration purposes, the synthesis system presented in this thesis targets a specific application domain and architecture model. However, most of the ideas presented here are general and may be applied to other domains. This section outlines the application domain and architecture model targeted in the synthesis system. The input specification and internal representation of the algorithm to be synthesized are also explained.

1.4.1. Application domain

The Synergy synthesis system is targeted for digital signal processing (DSP) applications. The main characteristic of these applications is that they are data-flow intensive with relatively few control constructs. These algorithms process or transform an infinite stream of input data in an endlessly repetitive loop in real-time. The rate at which the input data is received and processed is constant, independent of the values of the input signals, and is called the required *throughput or sampling frequency*. This rate is fixed by the DSP function and the surrounding system requirements. The inverse of the throughput is the *sample period*, which is the time between the arrival of successive input samples. Usually, the timing constraints are stringent and require parallel architectures for feasible implementations.

Some typical applications include filters (IIR, FIR), transforms (FFT, DCT, Hilbert), speech and audio processing (synthesis, coding, echo cancellation), and video processing (compression, decompression).

1.4.2. Input specification

The input algorithm is specified in a applicative language called *Silage* [48]. This language is specially suited for the DSP algorithms since it is simply a textual representation of a signal flowgraph where each variable represents a signal. The language is based on the single-assignment principle — each signal is defined in a unique way by an equation rather than a statement. The language supports control-flow constructs such as loops and if-then-else statements, and hierarchical constructs such as subroutines.

1.4.3. Internal representation

The algorithm in internally stored in a *control-data flowgraph* (CDFG) representation with nodes, data edges, and control edges. The nodes represent operations in the algorithm, data edges represent variables or data dependencies, and the control edges represent extra precedence rules between operations. The CDFG serves as the central data-structure on which synthesis and optimization operations such as complexity estimations, flowgraph transformations, hardware allocation, and scheduling tasks are performed and to which results are annotated. It also provides a convenient versioning mechanism to record the history of a design.

Nodes in a CDFG may represent arithmetic operations (addition, multiplication, etc.), control operations (if-then-else), sample delays, or hierarchical constructs. Hierarchical constructs are represented by nodes that are themselves CDFGs instead of primitive operations.

1.4.4. Targeted architecture

The targeted ASIC architecture is shown in Figure 1.1. It consists of a set of functional units (the exact number is derived during synthesis) that are connected together through an interconnect network. Though the network may be complete if needed, only those connections required for a given design are actually made.

Temporary storage (other than background memory) is provided through a distributed register file architecture. A register file is attached to each input of the functional units to store the requisite operands. The register files are assumed to have one write and one read port though the synthesis scheme can be modified to handle more complex register architectures. A variable is written into a particular register file when

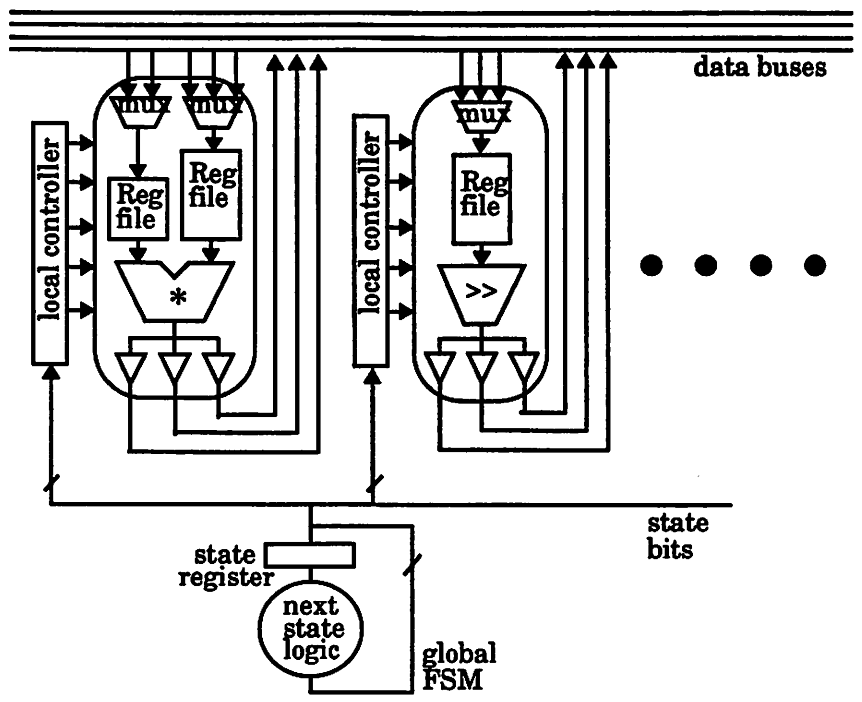


Figure 1.1. The targeted architecture model.

its producer operation is executed and is read when the consumer (the functional unit to which it is attached) executes.

Since a functional unit may receive data from more than one source, multiplexors are used whenever needed to select the data from the correct source. Also, buffers are used at the output of the functional units to drive the global buses. These buffers may be tristated, if needed, to control which unit drives each bus in a particular clock cycle. Physically, each functional unit and its associated registers, multiplexors, and buffers are clustered into a datapath block.

This interconnect structure is the most general and other simplified interconnect structures may be derived from it, if required. Two simplified schemes that are supported, the multiplexor-based and the tristate-buffer based schemes, are discussed below.

In the multiplexor based interconnect scheme, each function unit has a dedicated output bus that may be driven by a buffer if required. However the buffer is not tristated, and the unit owns the bus at all times. The receiving units select the data from the correct bus using multiplexors. In the tristate-buffer based interconnect scheme, each unit has a dedicated source bus for each input. The producer units decide which bus the data must be transmitted on, based on where they want to send the data. Bus selection by the producer units is done using tristate buffers.

In this thesis, we adhere to the generalized interconnect model as much as possible. However, one of the simpler cases may be used sometimes in order to facilitate the demonstration of results from using a particular optimization technique.

The controller is executed as single Moore machine (which implies a single thread of control) with a distributed structure as shown in Figure 1.1. A central finite state machine (FSM), generates the state information which is distributed to a set of local controllers. Each local controller generates the control signals for a particular datapath and is placed close to it in the final layout. The local control signals include read/write signals for registers, select signals for multiplexors, shift amount for shifters, tristate signals for buffers, etc. This distributed control scheme reduces power dissipation since only the state bits are globally distributed to the entire chip and the control signals are locally generated in the glue-logic blocks.

1.4.5. Hardware libraries

Underlying the synthesis system is a hardware library parameterized delay, area, and power information of the different library cells. Parameters include input and output wordlengths, supply voltage, maximum shift value for shifters, number of registers for register files, etc. The library information is used to drive decisions throughout the synthesis and implementation phases. The library does not need to be fully designed, but the accuracy of the numbers provided will affect the overall results. Library information may be obtained from either post-fabrication test results, prefabrication cell characterizations, preliminary estimations based on a paper design, measurements obtained from previous designs, or numbers reported in the literature.

This library-based approach has several advantages: (i) design tools can use information from of library components to make high-level decisions and (ii) library cells are designed only once, reducing design cycles.

1.5. Thesis organization

This thesis is organized into 9 chapters. This chapter has presented the motivation for the work, its key contributions and scope.

Previous and related works and background material are presented in Chapters 2 and 3. The various power related issues — the sources of power dissipation, power models, and the principal power reduction themes — are introduced in Chapter 2. This chapter also extensively describes previous work in power optimization and estimation. Chapter 3 gives an overview of architecture synthesis and the associated tasks. The Hyper high-level synthesis system which has been used as a research platform for experimentation of ideas in this thesis is also described.

Techniques for algorithm-level power estimation, their limitations, and results are presented in Chapter 4. The associated exploration environment is presented in Chapter 5, along with several examples illustrating the various degrees of freedom at this level and an extensive case study.

Architecture synthesis techniques for low power are treated in Chapters 6-8. In Chapter 6, the various power consuming components are studied to identify bottlenecks and qualify the impact of architecture-level techniques on them. Chapter 6 also discusses an extensive set of models for estimating interconnect power at the architecture level. Chapter 7 introduces a partitioning-based scheme for exploiting algorithm locality. A synthesis approach for identifying and exploiting algorithm regularity for interconnect power reduction is detailed in Chapter 8. Chapter 9 presents an overview of the entire synthesis system along with the key concepts, and also indicates possible future directions.

2 Power Optimization and Estimation: Overview

This chapter presents background material in power optimization and estimation. Section 2.1 discusses some of the basic concepts related to power dissipation in integrated circuits — the main sources of power, how it is modeled, and the common themes for power reduction. The next two sections overview related work in power reduction at the algorithm (Section 2.2) and architecture (Section 2.3) levels. Section 2.4 surveys previous work in power estimation at various levels of abstraction.

2.1. Power — sources, models, and themes

Power dissipation in an integrated circuit stems from four main sources — (i) *switching or dynamic power* dissipated while charging and discharging the capacitances in the circuit, (ii) *short-circuit power* consumed due to currents through direct paths from the supply to ground during switching, (iii) *static power* due to current flows in paths from the supply to ground when the circuit is stable (not switching), and (iv) *leakage power* due to sub-threshold and reverse-biased diode currents in the circuit.

At the algorithm and architecture levels, the short-circuit and leakage power are neglected since they (i) can be reduced to less than 15% of the total chip power by power conscious circuit design techniques [127], and (ii) are not influenced by the algorithm or the architecture style used. The static power component is also ignored since the most popular circuit styles exhibit rail-to-rail swings and do not consume static power. Therefore, only dynamic power is targeted for optimization at the algorithm and architecture level. The power dissipation is described by the following equation:

$$Power = C_{eff}(V_{sw} \cdot V_{DD})f \qquad \text{(Eq. 2.1)}$$

where f is the frequency of operation, V_{sw} is the switched voltage, V_{DD} is the supply voltage, and C_{eff} is the effective capacitance switched. The effective capacitance, C_{eff}, depends on C, the physical capacitance being charged/discharged, and α, the activity factor:

$$C_{eff} = \frac{1}{2}\alpha C \qquad \text{(Eq. 2.2)}$$

The activity factor α, associated with a node with capacitance C, is the average number of transitions per clock period at that node. Also, due to the assumption of rail-to-rail swings, $V_{sw} = V_{DD}$. Thus the overall power equation further reduces to:

$$Power = \frac{1}{2}\alpha C V_{dd}^2 f \qquad \text{(Eq. 2.3)}$$

Based on this power equation, four main degrees of freedom for power optimization can be easily identified. These correspond to reduction of the various power components: the supply voltage, activity, physical capacitance, and frequency. Each of these are briefly discussed below:

1. The recurring theme in low power design is *voltage reduction*. Since the power is quadratically dependent on the supply voltage, changes in supply voltage can potentially have a large impact on power. This power reduction comes at reduced speeds, however, since lowering the power supply increases the delay of the hardware blocks (Figure 2.1). Speed-up techniques at the algorithm and architecture levels can then be used to recover this loss in performance.

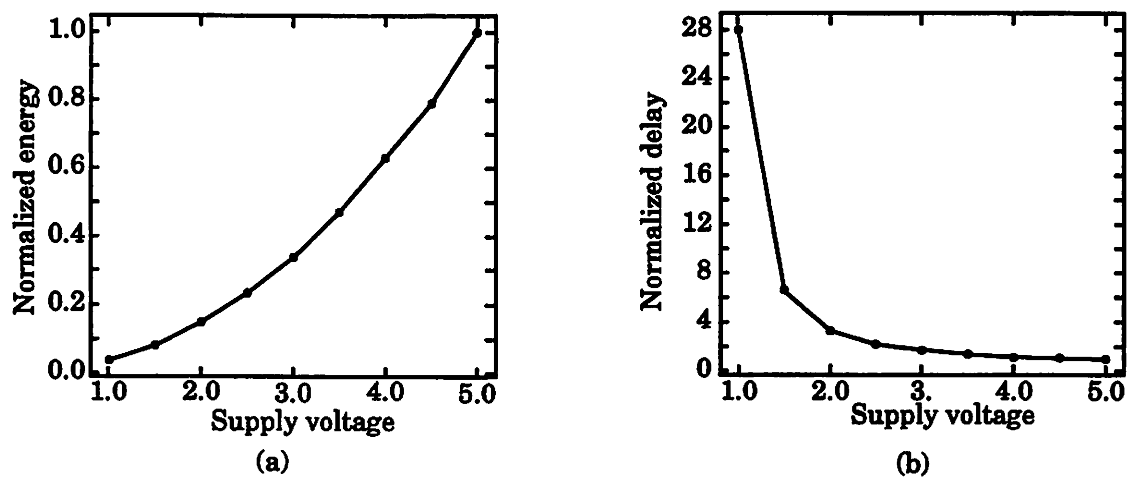


Figure 2.1. Effect of supply voltage: (a) on energy, (b) on delay.

2. Another important theme is *avoiding wasteful activity*. A computation should only be performed when absolutely necessary to avoid useless switching of capacitive nodes. This may be achieved either by optimizing the algorithm so that minimum number of operations are performed, by power-down techniques at the architecture level, or by reducing glitching activity at the gate and circuit levels.

3. *Reduction of the physical capacitance* is relatively less addressed at the algorithm level but forms the basis of a large number of architecture-, gate-, and circuit-level techniques.

4. While *frequency reduction* is a popular technique for lowering the power and hence the heat dissipation in a device, it has no effect on the energy used per operation and does not affect the number of operations per battery life. Thus it is not an attractive alternative for portable applications. It also has the associated problem of reducing the application speed and is not used for real-time (DSP) or high-performance (microprocessor) applications. However, it is widely used for reducing heat dissipation during idle periods of a device.

The rest of this chapter surveys various power reduction and estimation techniques that have been proposed in the literature.

2.2. Algorithm-level power reduction techniques

The algorithm-level presents a large design space with several degrees of freedom for power reduction. Algorithm-level techniques can be classified along the overall themes for power reduction presented in the last section. Below we revisit the main themes and mention algorithm-level optimization approaches for each.

1. *Voltage reduction*: Several algorithm-level techniques can be used to increase the speed of the algorithm, enabling voltage reductions. Among the most important are algorithm selection (Section 2.2.2) and speed-up transformations (Section 2.2.3). While it is desirable to operate at the lowest possible voltages for power reduction, the delay penalty involved introduces other effects which are analyzed in Section 2.2.1.

2. *Avoiding wasteful activity*: The *size* and *complexity* of a given algorithm (e.g. operation counts, wordlengths) determine its activity. Activity reduction can be achieved through algorithm selection (Section 2.2.2) by using a less computationally intensive algorithm; through dataflow transformations (Section 2.2.4) by reducing operation counts and substituting compute-intensive operations by simpler ones; and through memory transformations (Section 2.2.6) by reducing memory accesses.

3. *Reducing physical capacitance:* Behavioral decisions may impact the physical capacitance of several components of a design by affecting the area of the design (which changes interconnect capacitances); affecting the resources selected (which changes functional unit capacitances); or by affecting the memory sizes (which changes memory capacitance). For example, clock selection (Section 2.2.1) affects the overall area of the design and the selection of hardware resources, transformations for increasing resource utilization (Section 2.2.5) reduce the design area, and memory transformation (Section 2.2.6) can be used to reduce memory size.

The following sections elucidate the various techniques mentioned above — voltage, clock, and algorithm selection, as well as the different transformations — explaining the main concepts and presenting their advantages and disadvantages.

2.2.1. Parameter selection

Some of the parameters that can be selected at the algorithm level are the supply voltage and the clock period. Below we consider the various effects of the choice of these parameters on the overall power dissipation.

Voltage selection

For real-time applications with fixed throughput constraints (fixed sample period), maximum power savings can be obtained by operating at the lowest possible voltage at which the throughput constraint can be met. It is useful to analyze some of the other effects of voltage reduction (besides a quadratic reduction in power) on such applications. Firstly, as the voltage is lowered, it may be necessary to use faster modules (usually with higher area and power requirements) to meet the timing constraints, affecting both the area and the power of the design. Secondly, it may be necessary to perform more operations concurrently to meet the throughput constraints at lower voltages. This leads to area increases that may or may not be acceptable depending on the cost constraints, and if acceptable, would impact the interconnect lengths and power. These effects of voltage reduction are studied in detail in Chapter 5.

Clock selection

Another way to reduce power is to operate at reduced speeds since the power dissipation is directly proportional to the frequency. This imposes a delay penalty which may be acceptable for low-speed applications but is undesirable for high-performance applications. In real-time applications that are targeted in this thesis, the overall algorithm throughput is fixed by system requirements, but the clock rate used in the implementation is variable and is selected by the user. In this scenario, the choice of the clock period can impact the implementation in several conflicting ways.

Since the overall sample period is fixed, a longer clock period results in fewer clock cycles available for completing the algorithm. This reduces the total number of distinct states in the design, decreasing the power consumed in the controller. Other components affected are the interconnect and clock whose power depends not only on the number of states (which affects the switching activity) but also the overall area of the design (which determines the wirelengths). The choice of the clock period can have several conflicting effects on the overall area of the design. Since a longer clock period results in fewer clock cycles per sample period, more operations may need to be performed concurrently, increasing the number of hardware resources required. At the same time, a longer period reduces the number of clock cycles required by a particular operation (in case of multiple-cycle operations), allowing more operations to be performed sequentially and reducing the area requirements. Also, a longer clock period gives each operation more time to complete, allowing slower (and smaller) hardware units to be used.

The impact of the clock period is further explored through examples in Section 5.2.1.

2.2.2. Algorithm selection

For a given task, many algorithms may be available that trade-off complexity versus quality, area, or some other cost function. For example, three different speech coding algorithms with up to 50% variation in quality and complexity are presented in [63]. Given several algorithms for a particular task, the one with least number of operations is generally preferable for power purposes. Selecting the correct algorithm may result in large power savings and is a very important step in high-level design space exploration. Algorithm selection is demonstrated though a case study in Section 5.3.

2.2.3. Speed-up transformations

Speed-up transformations (first presented in [18]) reduce power by enabling supply-voltage reduction. However, they often translate into larger area requirements. Some of these transformations are discussed below.

Retiming and pipelining:

The most important speed-up transformations are retiming and pipelining. In the example of Figure 2.2, retiming is used to reduce the critical path of a third-order FIR filter from 3 clock cycles to 2 (assuming each operation takes one clock cycle). Since the throughput of the application is fixed, this speed-up can be used to scale the voltage from 5 V to 3.2 V, thereby reducing the power from 136.3 to 53.2 mW, a 61% reduction.

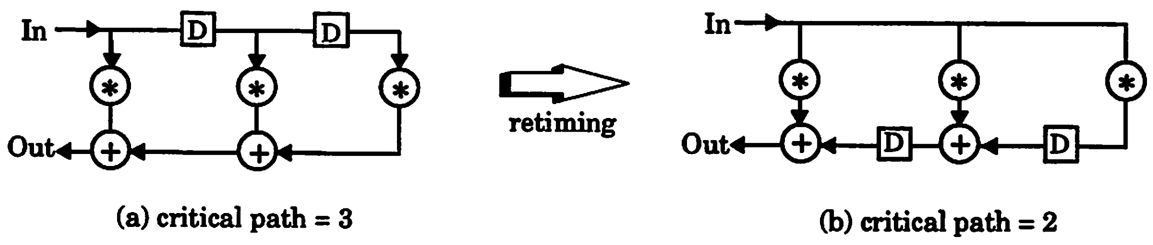


Figure 2.2. Using retiming to reduce power: (a) original structure, (b) after retiming.

Often other transformations such as algebraic and loop transformations are performed to enable retiming and pipelining. As an illustration, considér a first order IIR filter with a critical path of 2 (assuming each operation takes one clock cycle) shown in Figure 2.3a. The critical path cannot be reduced by retiming or pipelining and the simple structure does not provide opportunities for algebraic transformations. However, applying loop unrolling (Figure 2.3b) enables these transformations (distributivity, constant propagation, and pipelining) which result in a significant power reduction. The final transformed block (Figure 2.3d) has twice the time available for processing each sample (since it processes 2 samples in parallel), and the same critical path (2 clock cycles). Based on the Figure 2.3, this 2x speed-up can be used to reduce the voltage from 5 V to 2.9 V without changing the overall throughput of the filter. However, the effective capacitance switched has increased since the transformed graph requires 3 multiply and 3 add operations for 2 samples while the initial graph requires only 1 multiply and 1 add per sample — a 50% increase in complexity and hence in switched capacitance. Fortunately, the reduced supply voltage more than compensates for this increase resulting in an overall reduction of the power by a factor of 2.

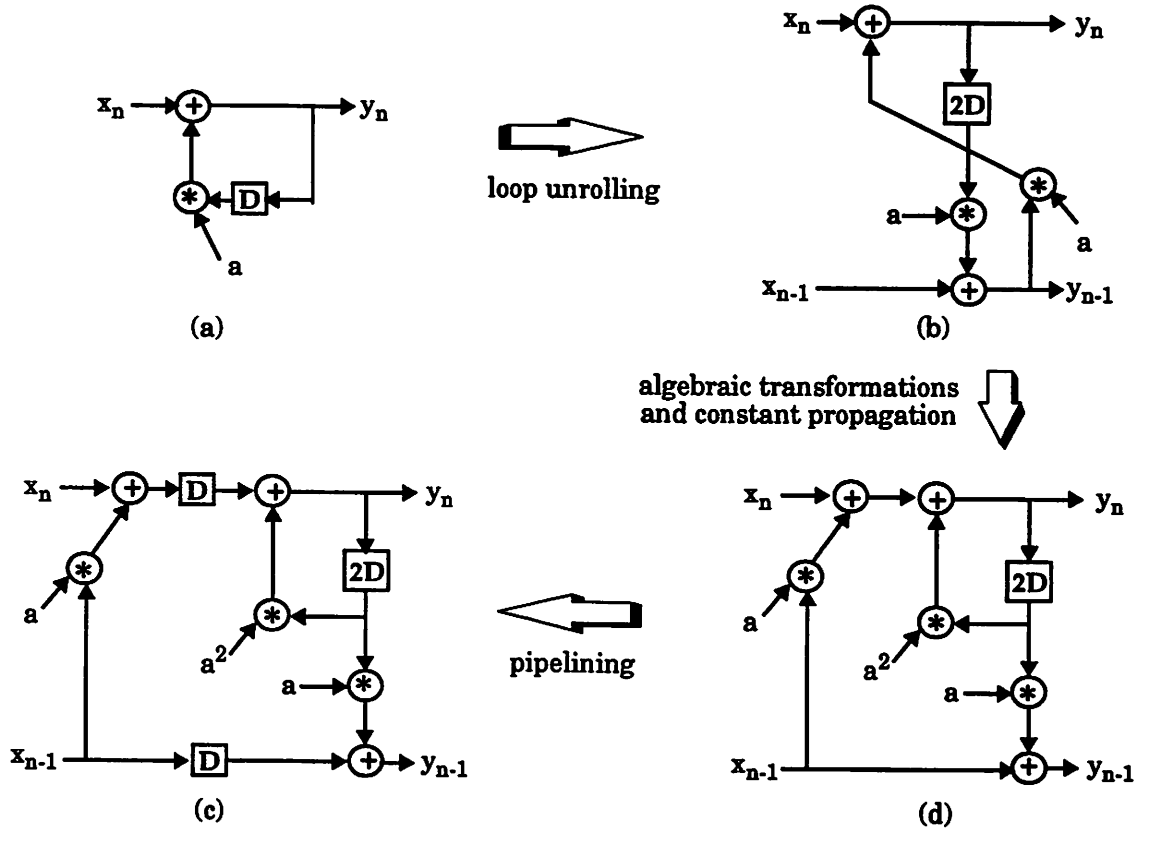


Figure 2.3. Illustration of speedup transformations on a first-order IIR filter.

The above example shows that a particular transformation can have conflicting effects on the different factors in the power equation — it may enable voltage reduction and simultaneously increase capacitance. In the above filter, arbitrary speedup can be achieved by further unrolling. This speedup can be used to drop the supply voltage for power reduction at a fixed throughput rate. But the capacitance grows linearly with unrolling factor and soon limits the power gains from reducing the supply voltage. Thus, the *fastest* solution is often *not* the lowest power solution.

Wordlength Reduction:

Certain transformations change the numerical stability of the algorithm, thus affecting the wordlengths needed to attain the required accuracy. While retiming, pipelining, and commutativity do not affect wordlength, applying associative and distributive identities may have a dramatic influence [39].

The wordlengths used in a design strongly affect its key parameters — speed, area and power. Smaller wordlengths are desirable for power optimization since they:

- increase operation speed and enable voltage scaling
- result in fewer switching events and lower capacitance
- reduce the overall area, resulting in lower interconnect lengths and capacitances

For example, the parallel-form IIR filter structure has higher numerical stability than the direct-form structure. Implementing an eighth-order Avenhaus bandpass filter [4] in the parallel-form requires only 11 bits while the direct-form implementation required 23 bits to get the same accuracy [25]. As a result of this, even though the parallel-form has more operations in its critical path (23) compared to the direct-form (20), the direct-form has a 50% longer critical path and thus a factor of four higher power consumption. In some cases, however, wordlength reduction comes at the expense of increased operations.

2.2.4. Activity-reducing transformations

Several transformations do not reduce the critical path of the algorithm but reduce its complexity and hence associated switching activity. These transformations were first presented in [18] and some of them are reviewed below.

<u>**Operation Reduction:**</u>

The most obvious approach is to reduce the number of switching events by reducing the number of operations in the algorithm. Transformations which directly reduce the number of operations include common sub-expression elimination, manifest expression elimination, and distributivity. These transformations may, however, increase the critical path.

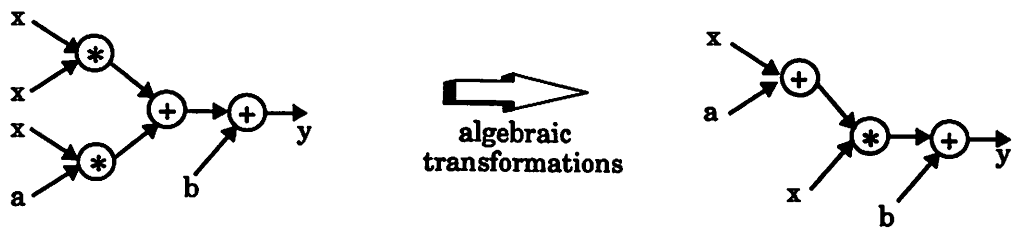


Figure 2.4. Operation reduction without increase in critical path: x^2+ax+b.

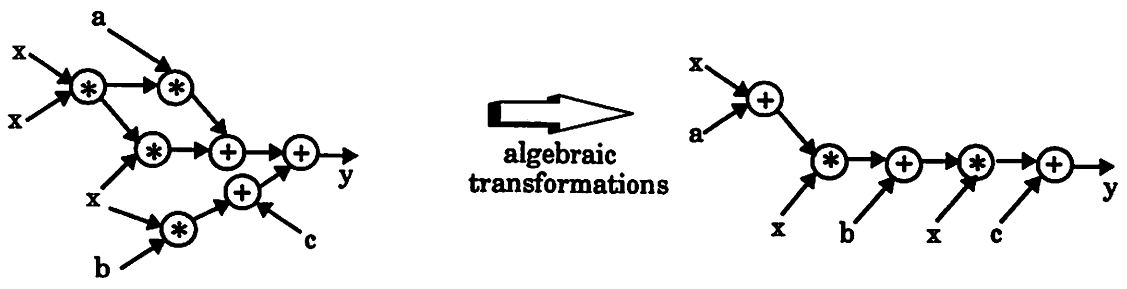


Figure 2.5. Operation reduction with critical path penalty: x^3+ax^2+bx+c.

Figures 2.4 and 2.5 show examples of operation reduction. In Figure 2.4, using the associative identity reduces the number of multiplications without changing the critical path and therefore results in a power reduction. In Figure 2.5, applying algebraic transformations reduces the number of multiplications by two, resulting in a lower effective capacitance but increases the critical path 4 to 5 clock cycles requiring a higher supply voltage to achieve the same computational throughput. In this case the transformation has two conflicting effects on power dissipation that need to be carefully evaluated to assess the overall effect on power.

Operation Substitution:

Operation substitution is used to replace power hungry (more complex) operations in the algorithm by low-power (less complex) ones. A powerful transformation in this category is conversion of multiplications with constants into shift-add operations. Since multiplications with fixed coefficients are quite common in signal processing applications such as transforms (DCT, FFT) and filters (IIR, FIR), the scope of application of this transformation is large.

Other transformations can be used in combination to achieve operation substitution. An example is shown in Figure 2.6, where redundancy manipulation, distributivity, and common sub-expression elimination are used to replace a multiplication by an addition. Unfortunately, the power reduction may come at the expense of an increase in the critical path.

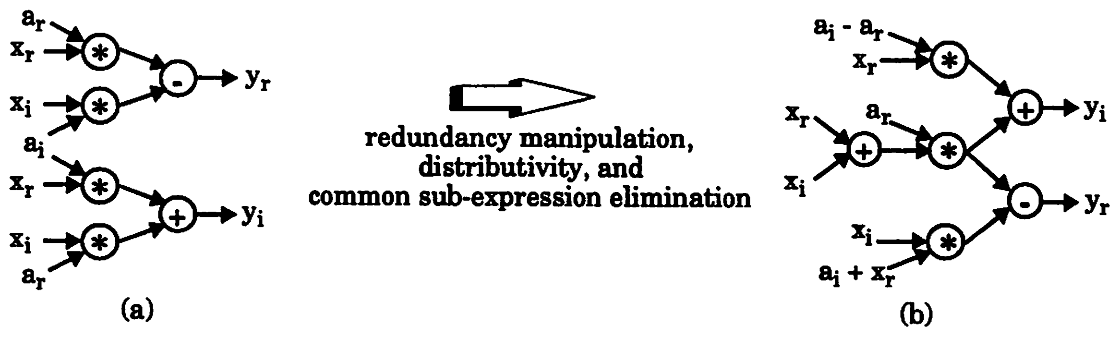


Figure 2.6. Replacing power-hungry operations by low-power ones.

In [20] Chatterjee and Roy studied the effect of operand activity on the power consumption of additions and constant multiplications and used it to guide architectural transformations.

2.2.5. Capacitance-reducing transformations

Transformations that reduce the implementation area save power by reducing the physical capacitance of global resources like buses [18].

<u>**Resource Utilization:**</u>

One way to reduce implementation area is to distribute the operations more uniformly permitting higher resource utilizations. Here power savings come from reduced overall area and hence lower wiring capacitance. However the increased hardware sharing may result in higher power in multiplexors and control circuitry. Therefore the optimization strategy must consider power consumed in interconnect and control.

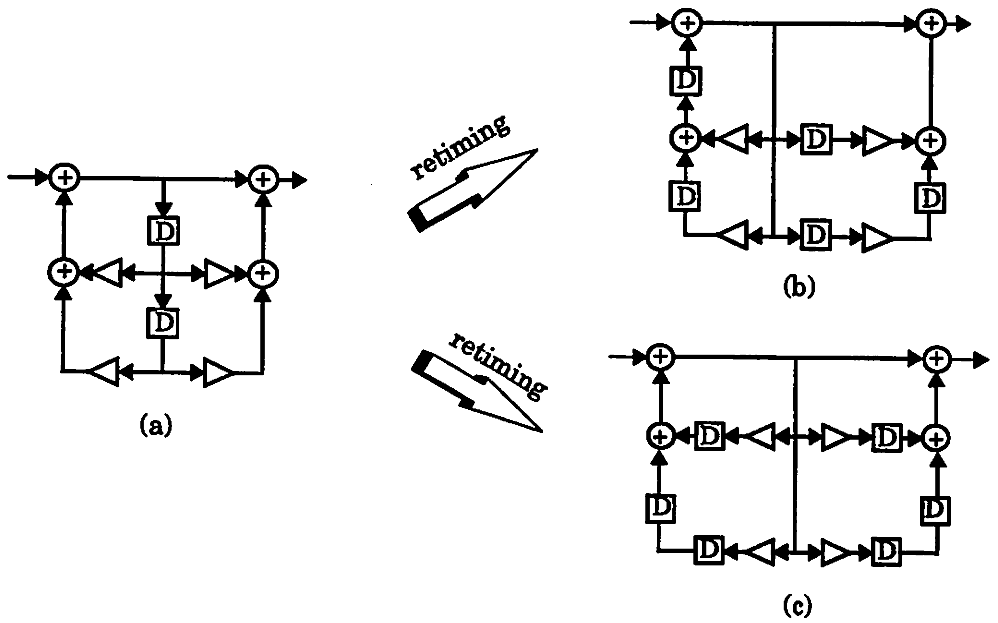


Figure 2.7. Different retiming schemes with different hardware requirements.

Transformations for increasing resource utilization include retiming, associativity, distributivity and commutativity. Figure 2.7 shows two different retimed versions of a second-order IIR filter. Both the transformed graphs are obtained from retiming and have a critical path of 3. While the graph of Figure 2.7b can be scheduled in three clock cycles using only 2 multipliers, the graph of Figure 2.7c needs 4 multipliers. Therefore, Figure 2.7b will result in a smaller implementation with shorter, lower capacitance buses than Figure 2.7c.

2.2.6. Memory transformations

Several DSP algorithms, especially those for video and graphics applications, are memory intensive and their power consumption is often dominated by the memory component. Transformations for memory play an important role in power reduction for these applications. These transformations differ substantially in their goals and features from those discussed in the previous two sections and warrant a separate discussion.

A significant amount of work has been done in memory-related transformations for area or performance improvement in the compiler [135], computer architecture [47], and high-level synthesis domains [5, 120], but relatively less work has studied their effects on power. Catthoor et al. were the first to specifically study the effect of memory transformations on power in the high-level synthesis context [15] and some of their main techniques are discussed below. First, we consider the factors influencing power dissipation in memories.

The power consumed in memories depends on the size of the memory, the number of accesses to it, and the activity on its input signals. While memory size and accesses can be influenced by both algorithm- and architecture-level changes, signal activity predominantly depends on architecture-level decisions. We discuss the first two aspects here.

Memory size reduction:

Memory size is affected predominantly by the amount of data that needs to be stored at a particular time and can be reduced by loop transformations on the algorithm. Consider the example shown in Figure 2.8a. Arrays A and C are already available in memory; when A is consumed another array B is generated; when C is consumed a scalar value, D, is produced. Memory size can be reduced by executing the j loop before the i loop (Figure 2.8b) so that C is consumed before B is generated and the same memory space can be used for both arrays.

```
for i = 1 to N do
    B[i]= f(A[i]);
for j = 1 to N do
    D = g(C[j],D);
```

(a)

$\Longrightarrow$ Loop interchange

```
for j = 1 to N do
    D = g(C[j],D);
for i = 1 to N do
    B[i] = f(A[i]);
```

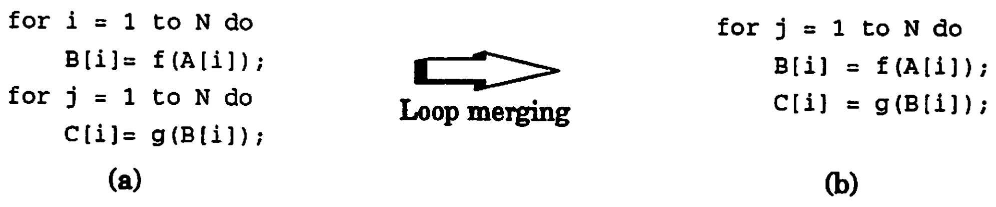
(b)

Figure 2.8. Loop interchange for reducing memory size.

Memory access reduction:

Accesses to memory are expensive since each time a data is read from or written to a memory location a large capacitance is switched. One way to reduce accesses to memory is to move variables from background memory to foreground registers as much as possible since register accesses are much cheaper in terms of power consumption. At the algorithm level, this can be enabled by reducing the lifetimes of variables.

Consider the example of Figure 2.9a consisting of two loops; the first loop consumes array A and produces array B which is consumed in the second loop and a new array C is produced. This calculation can be done “in-place” using a single n-element memory to store A, B and then C. However each memory element is accessed four times — to read A[i], write B[i], read B[i] and finally write C[i]. If the loops are merged as shown in Figure 2.9b, B[i] can be simply stored in temporary registers thus reducing the number of accesses to each memory element to two. The second implementation reduces the memory power consumption to half.



```
for i = 1 to N do
    B[i]= f(A[i]);
for j = 1 to N do
    C[i]= g(B[i]);
```

(a)

```
for j = 1 to N do
    B[i] = f(A[i]);
    C[i] = g(B[i]);
```

(b)

Figure 2.9. Loop merging for reducing memory accesses.

2.2.7. Algorithm-level techniques — lessons learned

The role of the various techniques in achieving the three basic goals of voltage, activity, and capacitance reduction are summarized in the Table 2.1.

Table 2.1. Algorithm techniques to reduce power.

Basic goals	Important techniques
Reduce voltage	Speed-up transformations: retiming, pipelining, algebraic, loop transformations, wordlength reduction. Voltage selection. Algorithm selection.
Avoid wasteful activity	Activity reducing transformations: operation reduction, operation substitution, distributivity, common sub-expression elimination, wordlength reduction. Algorithm selection. Memory transformations.
Reduce physical capacitance	Area reducing transformations: resource utilization, wordlength reduction. Algorithm selection. Memory transformations. Clock selection.

It is clear that transformations can have a large impact on the power consumption. However, the following facts must be considered before selecting a particular transformation for application on a given design:

1. Its effect is design dependent.
2. It may have a bigger impact if used in combination with other transformations.
3. It may have conflicting effects on the different power components.
4. It may have adverse effects on the area or other quality metrics of the design.

Algorithm-level power estimation mechanisms are extremely helpful in evaluating the varied effects and exploring the different options for a given design.

2.3. Architecture-level power reduction techniques

Once the algorithm is optimized for low power, it is mapped to an architecture or a register transfer level description. In this section we present an overview of architecture synthesis techniques for low power.

Since the voltage is already fixed at the algorithm level, we do not consider voltage reduction techniques. As mentioned before, the sample frequency is determined by the required throughput and the clock frequency is fixed at the algorithm level. The primary component targeted is the effective capacitance switched, C_{eff}. Power reduction at the architecture level is based on four underlying themes: activity reduction, localization, specialization, and demand driven operation.

1. Preservation of data correlations: Switching activity is dependent on correlations between successive data inputs and increasing correlations results in large power savings.
2. Distributed computing / locality of reference: Accessing global computing resources (control, datapath, memory, I/O) is expensive: the time-sharing nature of these resources requires a high switching rate, and the shared nature of such

a resource typically incurs a capacitive overhead. Distributing the accesses over many resources relieves both the switching requirements and the overhead. In particular, accesses to long global buses are costly and keeping data local reduces power consumption in data communications.

3. Application-specific processing: Specialized units consume less power than general-purpose ones due to simpler structure and reduced control required to support programmability.

4. Demand-driven operation: To avoid wasteful transitions, it is important to perform operations only when needed. Power down of memory and functional units when they are not in use is the most popular technique in this category.

These power reduction concepts will recur in the rest of this section, where we analyze the impact of the different synthesis tasks on effective capacitance and study power reduction techniques for each. The capacitance switched by each resource type — functional units, memory (including register files), interconnect (buses, multiplexors, and buffers), and control — depends on three factors: the resource's physical capacitance, the number of times it is accessed, and the correlation of the data that it operates on (the latter two determine the activity factor). While all three factors should be reduced to lower power consumption, the impact of reducing any one may depend on the values of the other factors. For example, it is more effective to reduce accesses to resources if they have a high physical capacitance.

Each of the above effective-capacitance factors for a resource is affected by decisions made during synthesis. In this section, we analyze these effects, and describe the relevant research efforts. Since the synthesis tasks are highly inter-dependent and may limit or enhance each other's effects, we present only those tasks which influence each capacitance factor most directly. Table 2.2 summarizes these influences and the remainder of this section elaborates upon them.

Table 2.2. Synthesis tasks affecting the different effective-capacitance factors.

	Physical capacitance		Resource accesses	Data correlation
Functional units	Module selection	—	—	Memory management Assignment (functional units, registers, buses) Scheduling
Memory		Memory management	Memory management	
Registers		Register allocation Memory management	Register assignment Memory management	
Muxes and buffers		Functional unit assignment	Functional unit and Bus assignment	
Buses	Placement and routing[1] Bus assignment Allocation (functional units, registers, buses)		Bus assignment	
Controller and control wiring	Assignment, Scheduling Logic synthesis[1] Placement and routing[1]			

1. Lower level tasks.

2.3.1. Reducing physical capacitance

Let us consider first the tasks affecting physical capacitance (Table 2.2, column 1).

The physical capacitance of functional units, memory, registers, multiplexors, and buffers depends on the selection of modules from the hardware library. In general, faster, more capacitive units may be needed in timing-critical situations; less capacitive units are better for cases where the timing requirements are not so critical. For example, as shown in [80], at higher voltages, a ripple-carry adder leads to more energy-efficient designs than a carry-select adder (Figure 2.10). At lower voltages, however, the ripple-carry adder may not be fast enough to meet the speed requirement and the carry-select adder can be used. Goodby [40] used module selection to speed up non-pipelinable paths to meet the timing constraint, using cheaper units for

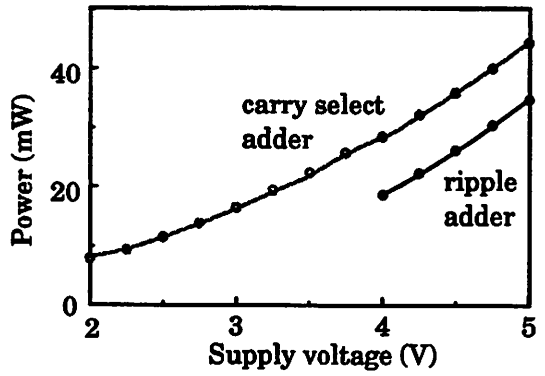


Figure 2.10. Relative power dissipation for an application running at a fixed throughput using different adders.

Another important trade-off in module selection lies in the use of specialized units instead of programmable ones. For example, it may be worthwhile to use a specialized adder instead of an ALU if there are a large number of additions in the algorithm. Specialized units consume less power for performing a particular operation but provide less flexibility.

While the physical capacitance of library based components depends mostly on resource selection and the technology used, the size of some resources, such as memories and buses, depends on the architecture mapping.

Memory elements:

For memory units, the size, and therefore the physical capacitance switched per access, is affected by memory management, while for register files, the size is mainly influenced by the register allocation. Register allocation is greatly restricted by the lifetimes of the variables. Reducing variable lifetimes by preserving temporal locality during scheduling can decrease the number of registers allocated. Since memory management decides whether variables should be stored in registers or background memory, it also influences the size of the register files and their physical capacitance.

Interconnect elements:

The physical capacitance of buses is directly related to their lengths, which are mainly determined by the number and size of the hardware units and their placement and routing. In general a large number of units with a lot of connections between them will lead to long bus lines. The number of hardware units is determined by resource (functional unit, register, and bus) allocation.

Bus assignment (also called bus merging) affects the physical capacitance since it can affect their lengths and the capacitive loading on them. Bus assignment also influences the power dissipated in the multiplexors and buffers. For example, the bus capacitance and the associated multiplexors and buffers are increased if a bus is merged with another one that has different sources and destinations.

The locality of the operations in the algorithm can be utilized during the binding of operations to hardware units to produce localized designs. By using localized communication within each localized region, bus and multiplexor power can be greatly reduced. This technique is extensively developed in this thesis (Chapter 7). Another idea presented in this thesis (Chapter 8) is based on preserving the regularity or repeated patterns of computation in an algorithm. This reduces power by enabling a simpler interconnect structure.

2.3.2. Reducing the number of accesses

Consider next the tasks affecting the number of accesses (Table 2.2, column 2).

At the architecture level, power-down of unused modules is a commonly employed approach to power reduction. Often this is achieved by disabling the clock signal to a resource (clock gating). Another way to "power-down" a resource is to prevent its inputs from switching, avoiding unnecessary transitions within the module when the resource is not in use. Two techniques based on this principle are pre-computation [2] and guarded evaluation [122]. The techniques differ in the granularity of circuits they target — pre-computation is used within logic blocks while guarded evaluation is used to disable an entire block of logic.

In the following paragraphs, we examine various approaches aimed at reducing the accesses to specific resources.

Memory elements:

Clock gating is especially popular for reducing power consumption in memories during idle cycles. Since all power-down approaches incur some overhead, its is important to cluster simultaneously-live operations and/or compact the active time-slots of a unit into consecutive intervals. In [29], Farrahi presents a memory segmentation algorithm that addresses this issue. The main idea is to partition the memory space so that memory accesses that are temporally close to each other are in the same block. In this way, only one block needs to be active for a given period of time and the other memory blocks can be shut off.

Although memory accesses depend mostly on the input algorithm, they can be substituted by accesses to foreground registers using memory management techniques. In general, accessing a value from the register file is cheaper since the size of the register file is smaller. In the example of Figure 2.11, the array A is the input to the loop and the array C is the output; array B stores intermediate values. Since only one value of B needs to be alive at a given time the array can be stored in a register eliminating the related memory accesses.

```
for j := 1 to N do
    B[i] := f(A[i]);
    C[i] := g(B[i]);
end
```

Figure 2.11. Simple loop for explaining memory access reduction.

Registers:

For register files, the accesses depend on the architecture model being used. For example, in a single centralized register file scenario, writes are determined by the algorithm (exactly equal to the number of variables) whereas for distributed register files, a single variable may need to be stored in more than one place. For a given architecture model, the number of reads from and writes to registers depends on the register assignment and the schedule.

Buffers and multiplexors:

Accesses to buffers depend heavily on the algorithm, since often each data transfer is buffered.

Operator assignment affects the amount of time-multiplexing of the functional units, which in turn affects the multiplexing of data transfers and the use of multiplexors and tristate-buffers. Accesses to multiplexors and tristate-buffers are further affected by bus assignment — if a unit needs data from two or more sources, a multiplexor may or may not be required at the inputs depending on whether the corresponding data transfers are merged onto the same bus.

Buses:

Accesses to buses depend on the total number of data transfers in the algorithm. Bus assignment further affects the accesses since, if a single variable needs to be transferred to more than one destination, one or more bus transfers may result based on whether the connections to the two units are merged into one bus. In this context, it is important to note that, if the buses are not all the same size, all accesses to buses do not consume the same amount of power and it is more important to reduce accesses to the longer buses. Preserving locality during bus assignment can reduce accesses to long global buses (Chapter 7).

Control:

The control related power includes the power consumed by the control wiring and the control logic. Wiring power depends on the wire lengths which are determined by the placement and routing. The power consumed by the control logic depends on the its functionality (determined from the assignment and schedule of the algorithm) and its implementation (logic-level optimizations). While the problem of optimizing power at the logic-level has been well studied, it is difficult to relate the controller power to high-level parameters, and therefore, to account for this component during architecture synthesis. Stochastic models described in Chapter 4, relating the control power to high-level parameters, provide a starting point in understanding various parameters and tasks affecting the control.

2.3.3. Improving signal correlations

Finally, consider the data correlation component of power (Table 2.2, column 3). Input correlations of all the components are affected by the allocation, assignment, and schedule. However the effect of these tasks on the correlations cannot be easily determined during synthesis because the correlations also depend heavily on the input data. Some research has been done to minimize switching activity during hardware assignment and scheduling. For assignment, the objective is to bind operations onto hardware so that the input signal activity is minimized. Raghunathan and Jha [102] propose an assignment scheme to minimize the average number of bit transitions on the signal inputs to hardware units (obtained from simulations).

Musoll and Cortadella [83] minimize the bit transitions for constants during scheduling. Consider, for example, the FIR filter of Figure 2.12. There are four multiplications with constants — c_0, c_1, c_2, c_3 — which can be scheduled on a single multiplier such that the transition activity at the right input of the multiplier is minimized. For the values of the constants given in the figure, the schedule $c_0 \rightarrow c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_0$ results in 26 transitions for a 12-bit implementation whereas the schedule $c_0 \rightarrow c_1 \rightarrow$

$c_3 \rightarrow c_2 \rightarrow c_0$ results in 34 transitions. Other methods suggested in [83] for increasing signal transitions include operand sharing (executing operations with common inputs in successive cycles on the same hardware), loop interchange and operand reordering. While the above techniques focus on increasing the correlations for functional units, Chang [19] proposes a register assignment scheme that reduces the activity for register files.

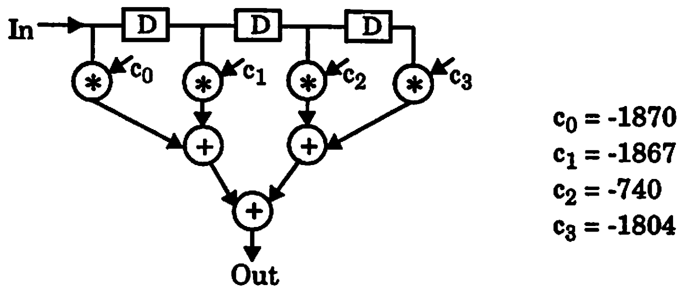


Figure 2.12. Scheduling to minimize the transitions.

The *number representation system* can have an important impact on the switching activity. Though common and simpler to implement, two's complement representation is not always the best for power purposes. A comparison between switching activity of two's complement and sign magnitude data streams is given in [65]. The latter is shown to have less switching transitions, since in going from a positive-to-negative number (or vice-versa), only the MSB switches in a sign-magnitude format while several higher-order bits are toggled in the 2's complement notation.

In cases where consecutive numbers are supplied to a resource in sequence, it may be worthwhile to code the signals in gray code since only one bit transitions when as numbers change by one. An important application of this technique is in memory addressing. Since memory locations (addresses) are often accessed in sequence, Gray coding reduces the overall switching. Up to 33% and 12% power savings were obtained in the instruction and data caches, respectively, by using this technique [119].

2.3.4. Architecture synthesis techniques — lessons learned

In this section we have presented the architecture-synthesis tasks that most directly affect the different factors comprising physical capacitance. It is important to notice that the tasks are highly interdependent and therefore the impact of each on power may be influenced by other tasks. For example, though the physical capacitance of buses depends on bus assignment, the effect of this task can be limited by functional unit assignment. If functional units are assigned such that number of destinations of each unit are low, bus assignment can result in better solutions than if they are high.

As another example, consider the number of accesses to registers in a distributed register file model described in Section 1.4.1. In this case, the number of register writes is determined by variable assignment to registers, which in turn heavily depends on the assignment of the operations that need these variables to functional units.

The last two sections have dealt with power reduction techniques. An equally important task in an overall power optimization system involves estimation or evaluation of the power dissipation which is discussed next.

2.4. Power estimation techniques at various abstraction levels

Increased interest in low-power designs has stimulated a lot of research activity in the area of power estimation. In this section, we review the different techniques that have been proposed at various abstraction levels. In general there is a direct trade-off between estimation accuracy and estimation speed at the different abstraction levels — higher-level tools are less accurate but faster than lower-level ones. Since higher-level tools provide feedback early in the design process, they are better suited for *design guidance,* while the more accurate lower-level tools are more suitable for *design validation.*

2.4.1. Circuit level

The earliest power estimation approaches SPICE [84] to simulate the circuit over a range of inputs and monitor the current drawn from the power supply. The Powermill tool from EPIC™ [26, 50] provides fast transistor-level simulation using table lookups derived from piecewise-linear transistor models and an event-driven simulation engine to achieve an order of magnitude speed-up over SPICE. A feature of this tool is that it incorporates techniques to identify "hot-spots" or power hungry parts of the design.

Circuit simulators are slow since they accurately model various device effects. For power estimation purposes, improved speeds can be obtained by modeling only the dynamic power in a circuit, which constitutes a major part of the overall dissipation as explained in Section 2.1. The dynamic power used for charging and discharging a particular node is given by the following equation:

$$Power = \frac{1}{2}\alpha C V_{dd}^2 f \qquad \text{(Eq. 2.4)}$$

where α is the transition probability or activity at the node, C is its capacitance, V_{dd} is the supply voltage, and f is the clock frequency. The switching power dissipation over an entire circuit with several nodes is then given by:

$$Power = \frac{1}{2}\left(\sum_i \alpha_i C_i\right) V_{dd}^2 f \qquad \text{(Eq. 2.5)}$$

where the summation is performed over all nodes in the circuit. For a given voltage and clock frequency, the problem of estimating this quantity reduces to evaluating $\sum_i \alpha_i C_i$. This can be estimated in switch-level simulators like IRSIM [106] where the switching activity is modeled but the leakage, short-circuit, and static currents are not.

Although circuit-level tools mentioned above provide the most accurate estimates, it is beneficial to estimate power at higher levels of abstraction, preferably before the circuit is completely designed, due to the following reasons. Firstly, circuit-level simulations are *time consuming* and secondly, these simulations *need circuit-level information* and cannot be used early in the design cycle. In the next few sections, we consider estimation techniques at higher levels of abstraction.

2.4.2. Logic level

At the logic level, the circuit is described as a network of logic gates and latches. Power estimation schemes proposed at this level fall into three main categories — simulation-based, probabilistic, and stochastic — each of which are discussed below.

2.4.2.1. Simulation based approaches

These approaches apply Equation 2.5 at the logic level, using logic simulators to extract transition probabilities at gate inputs and outputs, and gate-library information for node capacitances [8, 58]. The simulations have to be repeated for several sets of input patterns to get reliable power figures covering a typical set of input vectors. This requires complete information of all the different input sequences which makes these techniques *strongly input-pattern dependent*.

2.4.2.2. Probabilistic approaches

More recently, probabilistic techniques have been proposed that use input probabilities instead of actual inputs to encompass a whole range of possible input sequences and are hence *input-pattern independent*. These approaches are based on propagating signal probabilities through the gate-level descriptions and combining them with node capacitance values to calculate power. The various techniques proposed differ in the amount of information modeled in the probability measure.

In the simplest case [23], the following probability measures are used: the signal probability, $P_s(x)$, is defined as the average fraction of clock cycles in which a steady state value of the signal x is "1" and the transition probability, $P_t(x)$, is defined as the average fraction of clock cycles in which the steady state value of x is different from its value in the previous cycle. $P_t(x)$ is thus the activity of the signal x, disregarding any glitching. Given the signal probabilities at the primary inputs, those at all other nodes are calculated by probability propagation through the gate-level netlist. Ignoring temporal correlations, transition probabilities for all nodes are derived from their signal probabilities using:

$$P_t(x) = 2P_s(x)(1 - P_s(x)) \qquad \text{(Eq. 2.6)}$$

and power is calculated using Equation 2.5. This method ignores glitches as well as spatial and temporal correlations in the signals. An improved approach is presented in [87] which allows the user to specify the transition probabilities along with signal probabilities of primary inputs using probability waveforms, thus accounting for temporal correlations in signal sequences.

A more accurate measure for activity is given by the transition density, $D(x)$, which is defined as the average number of transitions per second at a signal x [86]. Note that this definition accounts for glitching activity. In terms of the transition density, dynamic power consumption is given by:

$$Power = \frac{1}{2}\left(\sum_i C_i D(i)\right) V_{dd}^2 \qquad \text{(Eq. 2.7)}$$

where the summation is again done over all nodes in the circuit. A technique to propagate transition densities from the primary inputs to all other nodes of the circuit is given in [86].

2.4.2.3. Statistical approaches

Another set of techniques, called *statistical* techniques, simulate the circuit with automatically generated input vectors. They use user-specified probability information to generate the vectors, and are therefore only *weakly pattern-dependent*. Statistical Monte-Carlo techniques can be used to determine when to stop the simulation to obtain a desired confidence level [13].

The techniques discussed above focus on estimating power in combinational logic circuits. For estimation approaches targeting sequential circuits, the reader is referred to [38, 125]. There is a large body of literature addressing power estimation at the logic levels and good surveys are presented in [85, 115].

2.4.3. Register transfer level

Moving up one level of abstraction, consider the architecture or register-transfer level. Here the design is described as a set of latches with combinational blocks of logic between them. The combinational blocks may either be library cells (library-based design approach) or functional logic descriptions that are later compiled to gates using logic synthesis (synthesis-based design approach). Estimation techniques for each of these design approaches discussed in this section.

2.4.3.1. Library-based estimation approaches

In the library-based approach, the internal details of the blocks are abstracted, making it impossible to get activity numbers or capacitances at the internal nodes. Also, since the internal nodes in each block have different capacitances and switching probabilities, a single activity and capacitance number for each block does not suffice. Therefore a hierarchical approach is used — the combined activity and capacitances of internal nodes is captured using the effective capacitance for each block, and the overall power consumption is computed from the usage of the various blocks as specified by the architecture. The power consumption is given by:

$$Power = \left(\sum_i N_i C_{eff}(i)\right) V_{dd}^2 f_s \quad \text{(Eq. 2.8)}$$

where the summation is done over all blocks i, N_i is the number of clock cycles the block is activated over the period of consideration and the $C_{eff}(i)$ is its effective capacitance.

The effective capacitance of a block, C_{eff}, represents the capacitance switched each time the block is used. While some of the approaches represent C_{eff} using *fixed activity* models, ignoring the effect of activity in the primary inputs of the blocks, others have proposed *activity sensitive* models that account for the activity at the block inputs.

Fixed activity models

1. The power factor approximation (PFA) technique presented in [96] uses parameterized models for the effective capacitance of blocks such as multipliers, additions, memory, I/O buffers, etc. Parameters (e.g. wordlength) are used to effectively model a large number of blocks with only few simulations. In that paper, the effective capacitances are taken from numbers previously reported in ISSCC proceedings, but these could also be obtained from library characterizations.

2. In [70], Liu and Svenson give high-level models of the different components of a circuit by analyzing the effective capacitance switched when the component is activated. For logic blocks, they use logic depth, fan-in, fan-out, and the input capacitance of a minimum sized inverter to derive a model for the effective capacitance. Similarly, their memory model is based on factors such as the number of rows, number of columns, capacitance loading of each cell on word and bit lines, number of bit lines switched per access, etc.

3. Another approach is to evaluate C_{eff} using uniform white noise inputs which have equal probability of being in the "0" and "1" states, and have no temporal or spatial correlations. This activity model, called the uniform white noise model, can be easily evaluated by simulating the block with uniform white noise inputs until a certain confidence level is attained (Monte-Carlo based approach).

Activity sensitive models

1. In [108], Sato et al. present a cycle-based simulator for a RISC microprocessor which also monitors the corresponding power consumption. Since it is based on simulation, it uses a power model for each block that is a function of the number of transitions, n, at its primary inputs.

$$P_{total} = P_{const} + nP_{change} \qquad \text{(Eq. 2.9)}$$

2. For datapath blocks, another activity-sensitive power model, called the Dual-Bit Type (DBT) model, is used in the SPA power analysis tool [65]. It is based on the observation that fixed-point two's complement data streams are characterized by two distinct activity regions. The lower-order bits (LSBs), exhibit activity similar to uniform white noise data, while the higher-order bits (MSBs) exhibit temporally correlated activity. Based on this observation, each library block is divided into three regions — the LSB and MSB regions, and the region in between them — depending on the correlation in its input data stream. While the effective capacitance of the LSBs is simply that obtained with uniform white noise input conditions, the effective capacitance of the MSBs is parameterized for different values of temporal correlation in the input data stream. In between the two regions the effective capacitance is computed by linear interpolation.

   Given a register transfer level description of a design, the tool performs a functional simulation to collect sample input streams for each datapath block. The statistical properties of the input streams at each block are used to demarcate the LSB and MSB regions for the block and also to compute the effective capacitance for the MSB region.

2.4.3.2. Behavioral estimation approaches

Recently, there has been increased interest in predicting power from a behavioral, rather than a structural, description of combinational logic blocks at the register-transfer level. Most of the techniques proposed in this realm use information theoretic measures of the computational work such as *entropy* and *informational energy* to predict the power consumption. In [88], the transition density of a signal x, is shown to be proportional to its entropy, $D(x) \propto H(x)$. A similar result relating the switching activity in a circuit to the entropy and informational energy of the input signals is given in [72].

In [88], Nemani and Najm used the following approximation to decouple the overall capacitance of the circuit from the overall activity:

$$P \propto \sum_i C_i D(x_i) \propto \sum_i C_i \sum_i D(x_i) \qquad \text{(Eq. 2.10)}$$

where C_i is the capacitance at node x_i, and $D(x_i)$ is the transition density at that node. They use a measure of the area of the circuit to estimate the total capacitance $\sum_i C_i$; the area measure used is derived from another work [21], which showed that for a boolean function with n inputs, each with a signal probability of 0.5, the output entropy, $H(y)$, can be used to predict the area of its average minimized implementation as follows:

$$A \propto \frac{2^n}{n} H(y) \qquad \text{(Eq. 2.11)}$$

Thus, they use entropy to provide relative measures for both $\sum_i C_i$ and $\sum_i D(i)$.

2.4.4. Algorithm level

In the only algorithm-level work that we are aware of, Shanbhag uses the entropy to establish a lower bound for the power dissipation required to implement a given algorithm with a certain information transfer rate [111]. He assumes the knowledge of the channel capacity of the underlying architecture and its noise power level. In Chapter 4, we identify the important issues and challenges at the algorithm level as well as present techniques for addressing them.

2.4.5. Instruction level

At a higher level of abstraction, instruction-level power estimation techniques have been proposed which can be used for making compiler optimizations for a given programmable processor. Tiwari et al. presented a scheme to measure the power consumption associated with each instruction for an Intel™ 486 processor by repeating the same instruction several times in a loop and measuring the current drawn by the chip [121]. The experiment was repeated for the Fujitsu™ Sparclite™ and Fujitsu™ DSP processors. Data gathered from such experiments was used to drive compiler optimizations for low-power.

2.4.6. Review of power estimation techniques

Over the last few years, as the power issue has gained importance, a large amount of research has concentrated on techniques to estimate power at all stages of the design flow. In this section we have reviewed some of the techniques to provide a flavor of the different approaches adopted. Table 2.3 summarizes the main techniques proposed at the various abstraction levels.

Table 2.3. Summary of estimation techniques at various abstraction levels.

Abstraction level	Power estimation technique
Circuit level	Transistor-level simulation: most accurate, models all sources of power, but slow and strongly pattern dependent
	Switch-level simulation: strongly pattern dependent
Gate level	Simulation based: strongly pattern dependent
	Probability based: pattern independent
	Stochastic techniques: weakly pattern dependent
Register transfer level	Library block based using fixed activity models
	Library block based using activity sensitive models
	Entropy or informational energy based functional models
Algorithm level	Combination of stochastic and analytic models (refer Chapter 4)
Instruction level	Measurement based

2.5. Summary

Algorithm- and architecture-level techniques result in large reductions in power dissipation. Since these techniques are highly inter-related, integrating them into a single tool is a non-trivial task. However, their application of the various techniques can be greatly enhanced by estimation mechanisms to predict their effects before they are applied, allowing the designer to select the ones most suited to the design at hand. Although a large number of power estimation schemes have been explored at various levels of abstraction, estimation at algorithm level remains relatively unexplored. Chapters 4 and 5 present an estimation and design space exploration facility that can be used to aid and guide power optimization at the highest levels.

3 Architecture Synthesis

High-level or architecture synthesis is concerned with deriving an architectural implementation of a given algorithm. The input is a behavioral description of the algorithm (e.g. in a C++ like language) and a set of performance constraints. The synthesis process involves deciding the mapping of the algorithm operations on hardware resources and determining their order of execution.

The *Hyper* synthesis system, developed at the University of California, Berkeley, has been used as a research platform for much of the work presented in this thesis. In our initial studies, the system was used to generate designs and study their power consumption. This helped us to understand the power problem and build detailed models for estimation (Chapter 4). The modular structure of Hyper allowed us to try out new ideas by implementing software modules that "hooked" into the main system. The ideas presented in this thesis were then compiled into a new low-power synthesis system, *Synergy*. Hyper was used as a base for comparisons with and evaluations of the new algorithms. For all these reasons, Hyper will be frequently referred to in this thesis. Therefore it is useful to briefly describe Hyper's design flow and underlying algorithms.

Details on the Hyper system can be found in [97]. A more general overview of the various high-level synthesis tasks is presented in [34, 126] and a number of available CAD systems for high-level synthesis are described in [131].

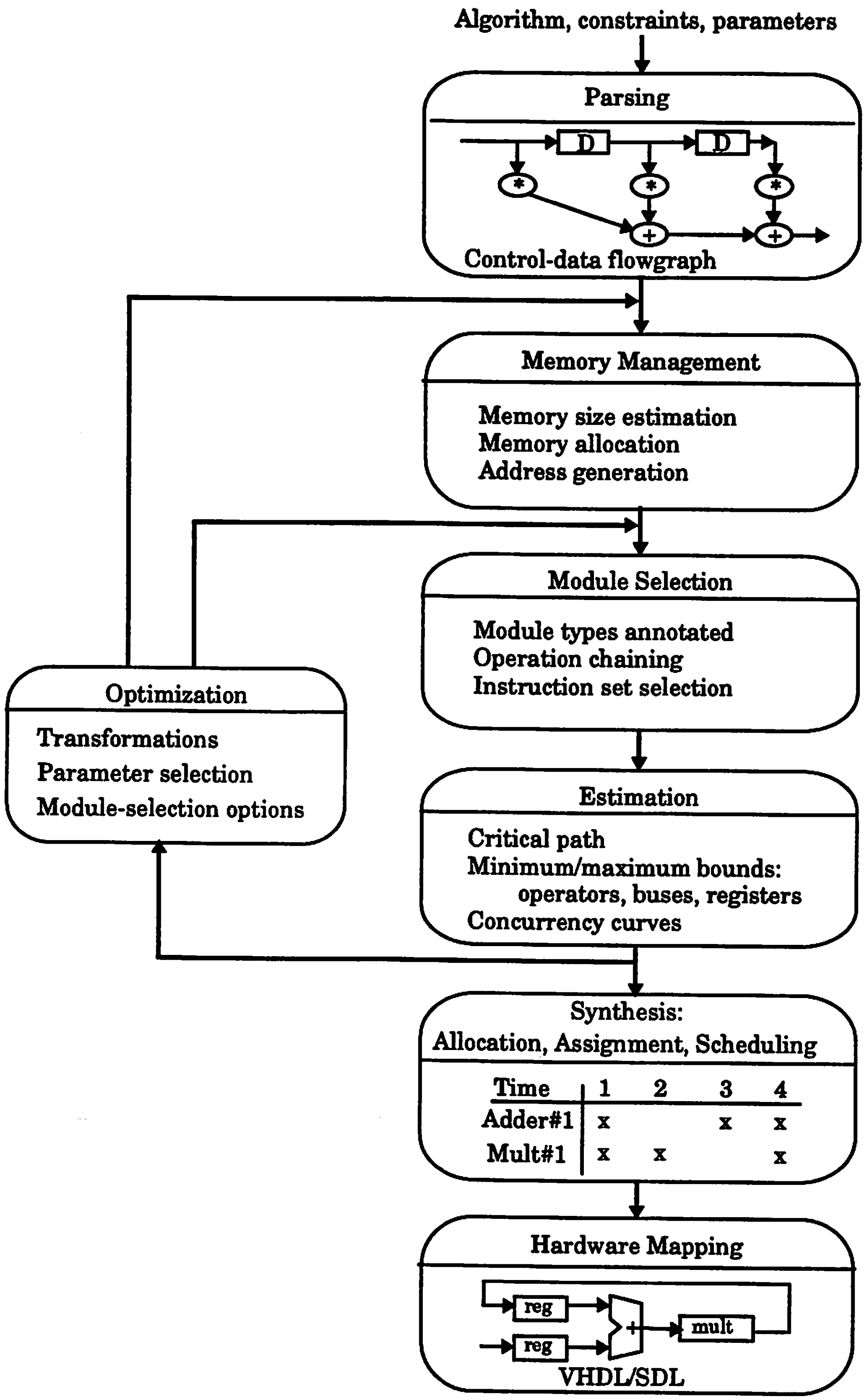


Figure 3.1. Design flow through the Hyper synthesis system.

3.1. Hyper — the base synthesis system

Figure 3.1 presents an overview of basic architecture-synthesis steps within the Hyper system. The user provides the algorithm specification, performance constraints such as the required throughput, and parameters such as the clock period and the supply voltage. The goal of Hyper's synthesis process is the minimize the area of the final implementation while meeting the throughput constraint.

The targeted application domain and architecture model are presented in Section 1.4. The input algorithm is first parsed from the Silage language [48] into the internal control data flow graph format, both of which are also detailed in Section 1.4. The rest of this section explains the main tasks in the overall synthesis flow.

3.1.1. Memory management

For memory intensive applications, an important synthesis task is memory management which includes generation of addresses, deciding whether variables should be stored in registers or background memory, allocating the number and size of the memory blocks, and assigning variables to specific memory blocks. Some of the algorithms used are presented in [128].

3.1.2. Module selection

The module selection process chooses appropriate hardware elements from a library [14, 133] to implement each operation. The "optimal hardware" for each operation is chosen according to the following strategy: for a given supply voltage and clock period, the hardware unit that can perform the operation in the lowest number of clock cycles[1] is chosen. In case alternate choices are available, the one with smaller area is selected. The module selector also determines which operations should be combined

1. The number of clock cycles required by an operation depends on its delay and the clock duty cycle. The delay includes the time required to generate the appropriate control signals, read the operand register, perform the operation, and write the result into the destination register via a bus and maybe a multiplexor.

to use more complex units such as multiply-add [22, 24]. The area, delay, and capacitance values of selected resources are annotated onto the flowgraph for use in subsequent synthesis steps.

3.1.3. Estimation

In order to optimize the algorithm at this level, one must first be able to predict the critical performance metrics. In the Hyper system, once hardware is selected, useful information such as the complexity and speed of the algorithm can be derived in the estimation step. Some metrics are computed exactly (e.g. critical path) while the minimum and maximum bounds are computed for others (hardware requirements). Bounds are important since they delimit the design space and speed up the search for the "optimal" implementation during synthesis. Also, if the bounds are close to the final solutions they provide good estimates for comparing different algorithms or different versions of the same algorithm.

As soon as possible (ASAP) and *as late as possible* (ALAP) execution times for each operation are obtained by first topologically ordering and then leveling the graph with respect to the inputs and outputs, respectively. The *critical path* of the algorithm is derived from the ASAP times of the operations and the number of clock cycles taken by each operation. It is defined as the maximum path length from any primary input or state to any primary output or state (assuming that all inputs and previous states are available at the start of the iteration). It determines the maximum throughput of the algorithm and must be no greater than the sample period for the algorithm to be feasible.

An upper bound on the number of units of each resource type (execution unit types, registers, and buses) is easily obtained by computing the maximal possible usage of that resource type (in other words, the maximal parallelism available) in each clock cycle and taking the maximum of this value over all clock periods in the sample period.

A naive lower bound on the number of units of type i, N_i, can be calculated as follows:

$$N_i \geq O_i \bullet \frac{d_i}{T} \qquad \text{(Eq. 3.1)}$$

where O_i is the number of operations of that type, d_i is the number of clock cycles it requires, and T is the total number of clock cycles available per sample period.

This bound is however too optimistic since it assumes that the flow graph contains sufficient concurrency to support a 100% utilization of each resource. Better estimates of resource utilization are obtained by first calculating a fast schedule by relaxing some of the constraints [100]. This method, called relaxation, turns the NP-complete scheduling problem into one with complexity $N \lg N$ and allows fast estimation. It is experimentally shown that the lower bound thus obtained is close to the final Hyper implementation and provides a good estimate for the number of units. The active area is estimated as the sum of the areas of the units.

3.1.4. Optimization

Even with the best synthesis techniques, the quality of the final result is limited by the initial specification of the algorithm. Transformations provide a method to overcome this limitation by changing the computational structure of the algorithm without altering its input-output behavior. Hyper provides a host of different transformations like retiming, pipelining, algebraic transformations, and loop-based transformations [93, 94, 51, 5]. Transformations fall into three main categories — *block-level transformations*, *timing* or *delay-based transformations*, and *loop transformations*.

Block-level transformations use dataflow techniques for optimizing the basic block of code. Transformations that optimize operations involving constants include constant folding and propagation (which evaluate and propagate the value of known constants at compile time), dead-code elimination, common sub-expression elimination, and constant multiplication expansion (which replaces multiplications with constants by add and shift operations). Further, algebraic transformations, based on the algebraic properties of the operators, are also used. Commonly used properties are commutativity, associativity, and distributivity. For example, since multiplications are commutative, their inputs can be switched without altering the algorithm functionality.

One of the most important timing transformations is retiming, which moves sample delays across operators taking advantage of the distributive property of delays over operations. Closely associated with retiming is the pipelining transformation. This is not a transformation in the strict sense since it does not preserve timing between the inputs and outputs; it inserts extra delays in the control flow, increasing the latency. However, it is a powerful technique for reducing the critical path and is often used.

Loop transformations [5, 120] include loop merging or jamming and loop unrolling (these increase the available parallelism by increasing the number of operations within the loop body), loop interchange (this may change the order of nested loops to reduce accesses to background memory), loop splitting, etc. Loop transformations have a large impact since they change the speed, area, and power requirements of the most computationally intensive parts of the algorithm.

3.1.5. Synthesis

The core of the synthesis process consists of allocation, assignment, and scheduling. For a given clock speed and algorithm throughput, the *allocation* task decides the number of resources of each type to be used, the *assignment* task binds operations and data transfers in the algorithm to specific hardware resources, and the *scheduling* process assigns each operation in the data-flow graph to one or more time steps.

The Hyper synthesis process [95] starts by allocating the minimum number of resources computed during estimation. Several different assignments of the operations to the allocated units are tried with this allocation. For each assignment, several schedules are tried. The scheduling uses an enhanced list-based strategy while the assignment strategy is based on a random initial assignment with iterative improvement.

If no feasible solution is found, hardware is re-allocated based on the assignment and scheduling results, and the assignment and scheduling processes are repeated. The addition or removal of hardware units is based on their "badness" which measures how a given resource type (adder, multiplier, etc.) affects the scheduling difficulty (for further details see [95]). The basic idea is to add units of the resource type with the highest badness (they are most responsible for failures in the assignment/scheduling phase) and to remove those whose badness is the lowest. To optimize area the allocation sacrifices smaller units to save on larger ones. The process is repeated several times until a feasible solution is obtained.

Register and *bus assignment* use a graph coloring algorithm. For example, in the case of buses, timing conflicts between different data transfers are represented in a conflict graph of all data transfers and a simple graph coloring heuristic is used for bus assignment. Similarly, for register assignment, overlapping variable lifetimes are represented by edges in a conflict graph of all the variables and a coloring heuristic is adopted.

3.1.6. Hardware mapping

The last step, hardware mapping [9], generates a finite state machine to control the datapath, and outputs the final architecture netlist in VHDL [52] or SDL [12]. The SDL format is suitable for silicon compilation to layout using the *Lager* system [12]. The VHDL format can be fed to commercial tools like Cadence™ and Synopsys™ for logic- and layout-level optimizations.

3.1.7. Architectural power analysis

Once the architecture is defined, the Hyper system provides a link to the SPA power analysis tool [65]. The main technique used for power estimation in this tool has been presented in Section 2.4.3.

3.2. This work

This thesis presents a new synthesis system, *Synergy*, targeted for low-power implementations. While the system uses the same flow as Hyper, several important ideas are included to address the power issue.

1. Power estimation capabilities are introduced at the algorithm level. Synergy focuses on providing average estimates because minimum and maximum power bounds can be extremely remote from the actual dissipation and are hence less meaningful for power. Further, the estimates of the hardware requirements from the Hyper system are used to build extensive models for the overall chip area. An explicit algorithm-level design exploration framework is designed to guide the user in making high-level decisions. The estimation and exploration strategies are described in Chapters 4 and 5.

2. A new set of synthesis techniques targeted for power reduction are added. Since the architecture-level optimizations have a greater effect on the interconnect power than the functional unit power, the new schemes focus on interconnect power reduction at the expense of an increase in functional unit area. Two key synthesis techniques are proposed. The first introduces a new partitioning step in the overall flow and the second proposes new allocation, assignment, and scheduling algorithms. The main ideas relating to the new synthesis schemes are presented in Chapters 6-8.

4 Algorithm-Level Power Estimation

As discussed in chapter 2, the most effective design decisions for power reduction derive from choosing and optimizing algorithms at the highest levels of abstraction. A large number of transformations and optimizations are available; however, their effect on the overall power dissipation is design dependent. A transformation may reduce the power consumption in some designs but increase it in others. Often, a technique may reduce the power in one component of the chip and increase it in another. Also, some of the techniques produce greater power reduction if used in conjunction with others rather than alone.

These factors make it difficult to select the best optimization techniques for a particular design. One option is to synthesize the design after each decision and evaluate its effect on power using an architecture- or gate-level power estimator. Given the large number of design decisions, this process is tedious and time-consuming and precludes an exhaustive exploration. As illustrated in Figure 4.1, an *algorithm-level power estimator* would greatly reduce the time associated with this process, enabling a more effective exploration of the algorithmic design space.

At the algorithm level, several implementation details cannot be accurately modeled, posing a significant challenge in realizing a high-level power estimator. However, two main points allow us to work with the reduced accuracy at these levels of abstraction. Firstly, since the power estimates are used for comparing designs and guiding design

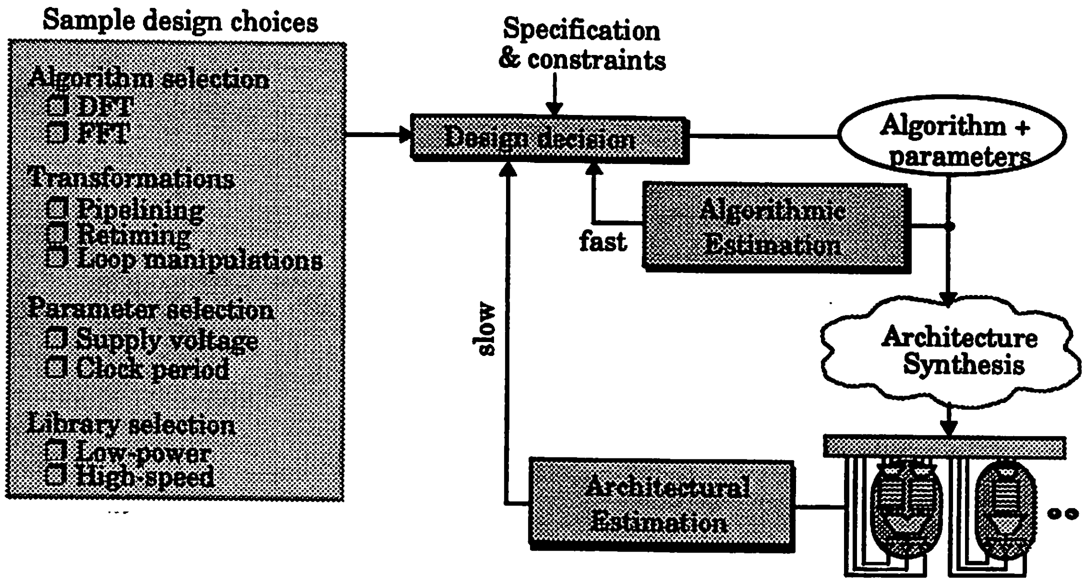


Figure 4.1. Importance of algorithm-level power estimation.

decisions, and not for obtaining precise power consumption values, only *relative* power estimates are required from these predictions. Secondly, since algorithm-level decisions can result in orders of magnitude changes in power dissipation, the accuracy requirements on the estimation tools can be relaxed, while still providing meaningful power predictions to guide high-level decisions. The work described in this chapter presents prediction tools that provide relative metrics to be judiciously used in making high-level design selections.

4.1. Algorithm-level power estimation

At the algorithm level, the design is specified by a functional description with no implementation details. The challenge in power estimation at this level is to derive useful estimates with the extremely limited information available. While no work directly addresses the algorithm-level power estimation problem, some techniques that have been proposed in other areas may be applied for this task. Below we discuss possible approaches that may be used at this level and present the core of our approach.

One approach is to predict power from purely behavioral characteristics of the algorithm and its inputs. *Information theoretic measures* of the inputs and outputs of the algorithm, like their entropy and informational energy, may be used to characterize the activity in the circuit. Shanbhag's work on lower bound calculations can be considered as a step in this direction [111]. Though this work starts with extremely abstract specifications, some assumptions are made about the architecture. *Structural properties* of the algorithm such as operation counts, concurrency, spatial locality, and regularity provide size and complexity measures [41] that could be extended for power predictions [98]. For example, algorithms with higher operation counts require more computational work resulting in higher power consumption. Also, more concurrent algorithms result in larger designs that have longer buses and more interconnect power. The degree of locality in an algorithm also affects its power consumption. Highly local algorithms lend themselves more easily to partitioning and can thus result in low-power localized architectures. The concept of spatial locality is explored in detail in Chapter 7. Another related metric is the regularity of the algorithm which refers to the occurrence of repeated patterns of computation in it. Regularity in an algorithm can be exploited to simplify the interconnect infrastructure. This concept is developed extensively in Chapter 8.

The approach mentioned above focuses on characterizing the algorithm and is elegant in the sense that the estimates obtained are general and not tied to any implementation style. However, if it to be used in a practical design environment, the algorithm rankings obtained must hold after the implementation. This may not be true due to several factors. Firstly, prioritizing and combining the various characteristics of the algorithms to produce a ranking is a non-trivial task. Secondly, the rankings will not hold if the implementation techniques do not exploit the property used to create the ranking. For example, if algorithm A is selected over algorithm B due to its higher locality but the designer ignores the locality while generating the final design, algorithm A may not result in a better implementation.

4.1.1. Our approach

We present an estimation approach to predict power from a behavioral specification of the algorithm. In addition to the specified behavior, we assume a knowledge of the underlying *architecture model* and a user specified *hardware library*. Each operation in the algorithm can be associated with a cell in the hardware library. Since our estimates are produced with an architecture model and hardware library in mind, they are *technology targeted.* The use of technology-specific information allows us to derive more meaningful estimates, while still maintaining the speed of the above methods.

The effectiveness of the approach is demonstrated by targeting the architecture model described in Section 1.4.4. Estimation schemes are extensively studied for all components in this architecture. The techniques can be extended to other architecture models, but the model must be defined in advance. An underlying hardware library (discussed in Section 1.4.5) is assumed; however, none of the schemes presented are specific to the library used.

An important characteristic of our approach is that it embodies a heterogeneous set of techniques to analyze the different components. Each architectural component is treated differently based on its specific features and the information available about it. The techniques combine dataflow analysis with the stochastic studies to estimate the overall power.

The total power consumption in an ASIC design is comprised of contributions from several components: datapath, registers, interconnect, control, and memory. These components differ fundamentally in the their dependence on the algorithm specification and therefore, in the amount of information available about them at this level. From the power consumption point of view, we divide the components into two parts: the *algorithm-inherent components* and *implementation overhead*. The algorithm-inherent components includes the functional units and background memory. These are "algorithm inherent" since the computation associated with them is necessary for the basic functionality of the algorithm. Therefore their power consumption is strongly dependent on the behavioral specification. The implementation overhead component includes the temporary storage (registers), interconnect elements (buses, buffers, and multiplexors), and control. The power dissipation in this component largely depends on the choice of the architecture and implementation.

4.2. Estimating the algorithm-inherent power dissipation

The algorithm-inherent components include the functional units and memory. A two step approach is used for power estimation in this category — characterization of the library blocks and characterization of the algorithm. While the former is indicative of the capacitance switched when a given library block is used for some computation, the latter specifies how often each block is active.

The overall power dissipation is estimated as the overall effective capacitance switched multiplied by the square of the supply voltage times sampling frequency. The capacitance switched in one iteration (one sample period) of the algorithm is given by:

$$C_{total} = \sum_{i} N_i C_i \qquad \text{(Eq. 4.1)}$$

where N_i is the number of operations of type i, C_i is the capacitance switched when that operation is executed, and the summation is performed over all operation types i (memory and arithmetic operations).

4.2.1. Characterization of the algorithm

The algorithm is characterized to obtain the counts of all the different operation types (memory and arithmetic operations). This is a measure of the activity in the corresponding functional units. For example, an addition count of 100 for a given algorithm indicates that adders will be executed 100 times in one iteration of the algorithm. Note that the *type* of an operation is determined by the functional unit selected from the hardware library for implementing it and not by its functionality. For example, 4-bit additions and 32-bit additions in a graph may use different adders for which different count measures will be generated. Also, both additions and subtractions in a graph may be implemented on ALUs in which case the number of additions and subtractions will be added to give a single count for ALUs.

For the synchronous data flow designs [66] targeted in this thesis, a static schedule of the operations can always be obtained and the operation counts are known before run-time. However, for algorithms with data-dependent control flows operation counts and memory accesses are input-dependent. In this case, either the average or worst case iteration counts must be provided. Sometimes these may be imposed by the surrounding system and can thus be specified a priori by the user. For example, certain signal processing applications require that a fixed throughput rate be sustained, enforcing a limit on the worst case iteration counts or critical paths. Some other algorithms such as speech recognition algorithms may require some average case behavior. In cases when these limits are not pre-specified, the algorithms require profiling with typical input vectors to obtain the average or worst case iteration counts for the data-dependent loops.

4.2.2. Characterization of library blocks

As described in Section 2.4, a lot of research effort has focused on abstracting the power dissipated in blocks of hardware. However, all the approaches discussed in that section presume detailed knowledge of the architecture, which is not known at the algorithm level. The most important difference lies in the fact that signal statistics cannot be estimated at the algorithm level even if sample sequences are available for the primary inputs. This is because, the signal statistics at the inputs of a particular hardware block depend on the schedule and assignment of operations — information not available until the architecture is finalized. Therefore a uniform white noise input model is used to compute the switching capacitance of different hardware blocks.

With this assumption, an algorithm with five additions consumes the same power (in the adders) irrespective of the amount of hardware sharing.

Modules in the hardware library are characterized in terms of complexity parameters such as wordlength, register size, etc. using uniform white noise inputs. Intuitively, the total power consumed by a module should be a function of its complexity (i.e. "size") since larger modules contain more circuitry and, therefore, have higher physical capacitance and activity. One would expect a 16x16 multiplication to consume more power than an 8x8 multiplication. This leads to a complexity-based capacitance model that has widely been used at the architecture level [65, 96]. Complexity models used in this work are derived from [65]. For an extensive discussion on the models and their derivation the reader is referred to that work, here we present the basic concepts through a few examples.

The effective capacitance of a ripple-carry adder depends on its wordlength, n, which is a measure of its "complexity", and is given by:

$$C_{eff} = cn \qquad \text{(Eq. 4.2)}$$

where c is a capacitive coefficient measuring the effective capacitance switched for each bit of the adder. Although several modules — adders, barrel shifters, multiplexers, registers, etc. — follow the simple linear model of Equation 4.2, complexity models are not limited to this one. For example, the capacitance model for a logarithmic shifter is given by

$$C_{eff} = c_0 n + c_1 l + c_2 nl + c_3 n^2 l + c_4 mnl + c_5 snl \qquad \text{(Eq. 4.3)}$$

where n is the wordlength, s and m are the actual and maximum shift values, while $l = \lceil log_2(m+1) \rceil$ is the number of shift stages.

In general, capacitance coefficients and complexity parameters are stored as vectors, and the overall effective capacitance is obtained from their dot product. In the rest of this thesis, this model is referred to as the *capacitance complexity model*. For the logarithmic shifter example, the capacitance coefficient vector, the complexity vector, and the overall capacitance complexity model are shown in Equations 4.4, 4.5, and 4.6, respectively.

$$\bar{c} = \begin{bmatrix} c_0 \; c_1 \; c_2 \; c_3 \; c_4 \end{bmatrix}^T \qquad \text{(Eq. 4.4)}$$

$$\bar{n} = \begin{bmatrix} n \; l \; nl \; n^2 l \; mnl \end{bmatrix}^T \qquad \text{(Eq. 4.5)}$$

$$C_{eff} = \bar{c} \cdot \bar{n} \qquad \text{(Eq. 4.6)}$$

4.3. Estimating the implementation overhead

The power consumed by the implementation overhead — registers, interconnect elements (buses, buffers, and multiplexors), and control — depends on the specific architecture platform chosen and the mapping of the algorithm operations onto specific hardware units. Since these are not essential to the basic computation in an algorithm, several estimation tools ignore their effect for high-level comparisons. However the power consumed by these components is often comparable to, if not greater than, the algorithm-inherent dissipation. This is illustrated in Figure 4.2 which shows the power breakdowns (architecture-level estimates) obtained from designs implemented using the Hyper synthesis system. Clearly, estimates of the implementation overhead must be included to obtain meaningful overall predictions.

The lack of information about these components at the algorithm level poses a significant challenge to power estimation. For example, in a high-level synthesis based design flow, the control is not specified in the algorithmic or flowgraph description of the design and is defined only after the allocation, assignment, and scheduling tasks. Similarly, the interconnect elements are not defined until very late in the synthesis process — multiplexors and buffers are added to the design description only after the

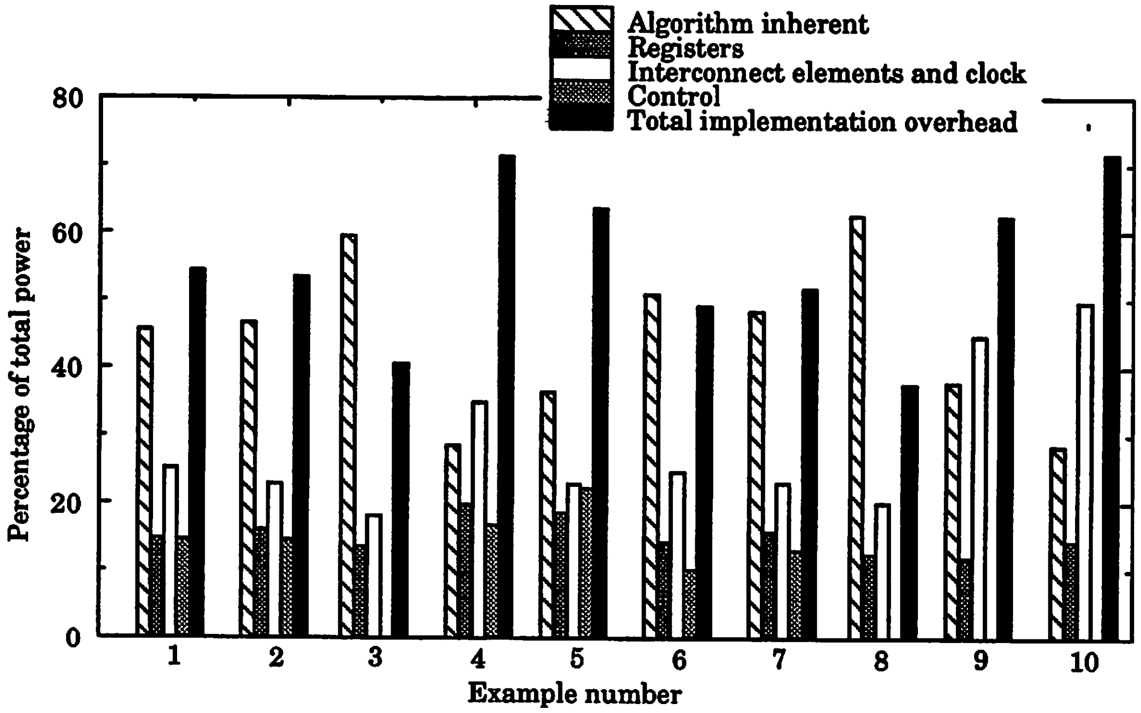


Figure 4.2. Contribution of the various components to the total power.

operations and data transfers have been assigned and scheduled; and bus capacitances (lengths) are determined during the floorplanning and layout steps.

Thus, estimates of the control and interconnect components need to account for the effects of a wide spectrum of high-level synthesis tasks, such as allocation, assignment, scheduling, and partitioning into macro blocks, as well as the effects of many low-level CAD tools, such as logic synthesis, placement, floorplanning and global and detailed routing. An extensive experimental study, followed by in-depth statistical analysis and verification is the only viable solution for efficient and accurate modeling of these components at the algorithm level.

Power estimation of the implementation overhead is further complicated by the widely varying characteristics of these components. We use analytical models for estimating the temporary storage component and stochastic techniques for the interconnect and control elements. The next section describes our methodology for stochastic modeling of these components. Specific models (both analytic and stochastic) for each of the components are described in Sections 4.3.2, 4.3.3, and 4.3.4.

4.3.1. Stochastic modeling

Stochastic models for the interconnect and control were derived using several examples which were mapped from their Silage descriptions to layout using the Hyper synthesis system and the LAGER silicon assembler [12]. Figure 4.3 shows the design flow used for algorithm mapping and data collection. The logic synthesis tool, MIS II [11] was used for optimizing the control logic. The controllers were mapped to a library of standard cells, and the remaining blocks are mapped to datapaths by tiling custom library blocks in a bit-sliced fashion using the Timlager layout generator [12]. For the control model, the capacitances switched per sample period were measured using the IRSIM switch-level simulator [106] and for the interconnect model, the areas, bus lengths, and accesses were measured from layout. A number of high-level parameters such as the number of states, the number of functional units, wordlengths, etc. are also collected from the design.

Benchmark set:

The selection of examples for building the model was guided by the goal of including as diverse and as typical examples as possible. The selected examples cover a wide variety of DSP applications including FIR filters (Wavelet, Hamming, Chebyshev), IIR filters (direct-form, cascade, parallel, continuous-function, ladder, wave-digital, polynomial and homomorphic filters), nonlinear filters (Volterra) and fast transformations algorithms (DCT, FFT, convolution). Example sizes ranged from 13 operations to more than 120 operations. Different instances of these examples were generated using different transformations (retiming, various extents of pipelining, algebraic transformations and other block-level transformations), supply voltages, clock periods and sampling rates. The examples cover a wide variety in the ratio of

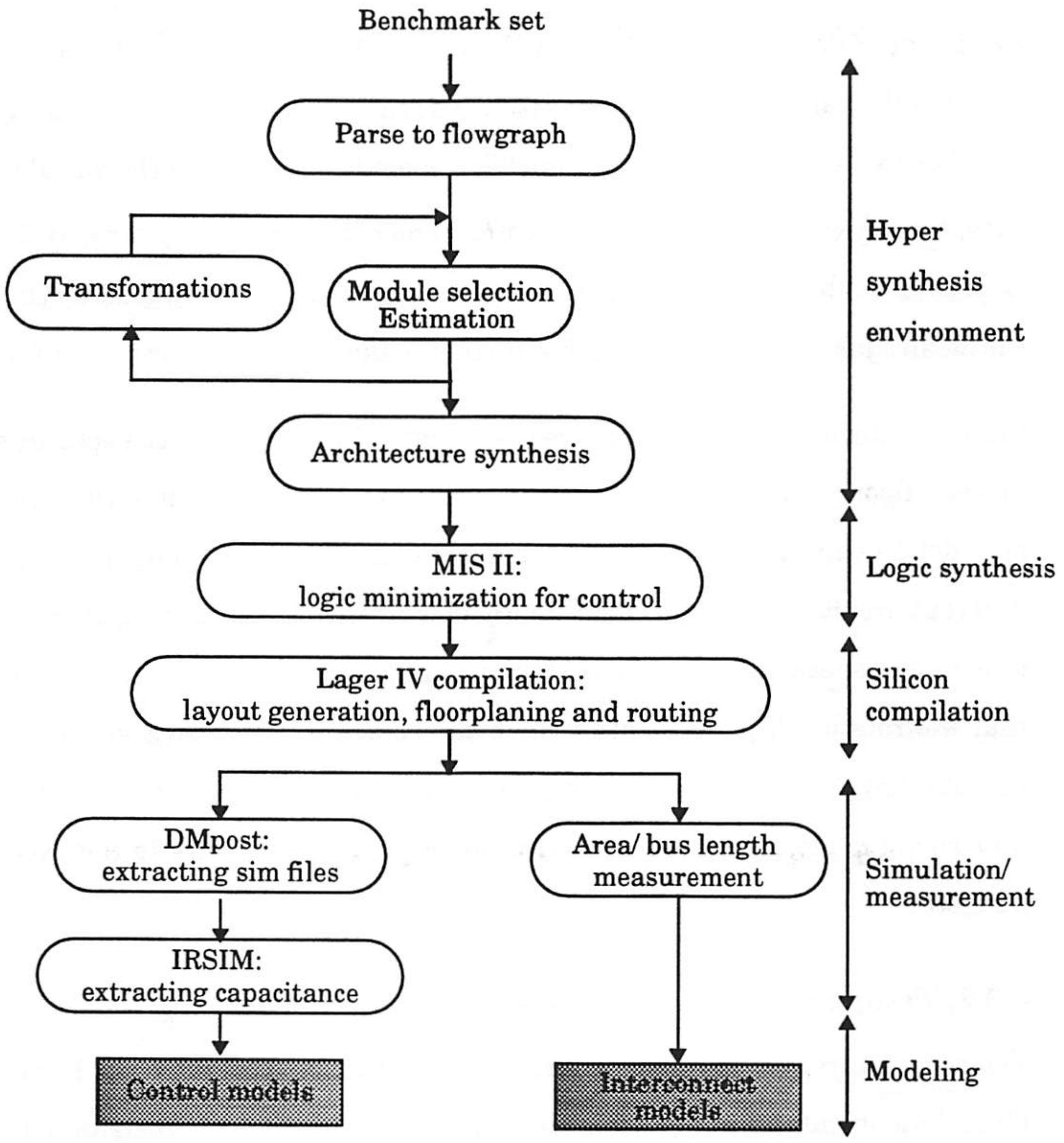

Benchmark set
Parse to flowgraph
Transformations
Module selection Estimation
Architecture synthesis
MIS II: logic minimization for control
Lager IV compilation: layout generation, floorplaning and routing
DMpost: extracting sim files
IRSIM: extracting capacitance
Area/ bus length measurement
Control models
Interconnect models
Hyper synthesis environment
Logic synthesis
Silicon compilation
Simulation/ measurement
Modeling

Figure 4.3. Design flow used for statistical modeling.

available time to critical path (also called *stress ratio*), amount of parallelism, types of operations used, level of multiplexing, and size of the final implementations.

Regression analysis:

Regression analysis (with the Matlab™ [75] and Mathematica™ [132] tools) was used to identify the important relationships and build models. The R^2 and F-ratio goodness-of-fit parameters were used to determine the significance of the models. R^2, the coefficient of determination, is defined as ratio of the variability of explained by the model to the variability of the data itself. The ratio varies between zero and one, with a higher value indicating a better model. If a model explains all the variability in the data, the R^2 would be 1, and if it explains none of the variability, R^2 is 0. The F-ratio is defined as the ratio of the mean square of the variation explained by the model to the mean square of the residuals. For further details, the reader is referred to [10, 53].

Since the stochastic models capture the nuances introduced by the specific synthesis process, floorplan, and architecture model used when generating the data, the models may not be valid for a different synthesis process or floorplanning strategy, or for a different architecture style. In our methodology, the understanding of the basic relationships is specifically emphasized while building the overall models and we believe that the relationships will hold though the constant coefficients will have be to be recalibrated. An important benefit of our approach is that it helps to understand the underlying dependencies and identify the various contributors to the overall power budget.

4.3.2. Temporary storage (registers)

Temporary variables in an application may be stored in several ways. For example, a given implementation may use a centralized register file or a distributed one; completely pipelined designs use pipeline registers instead of register files; chaining operations can get rid of registers; delay elements may or may not need registers depending the relative operation times of their producer and consumer nodes. Thus it is difficult to estimate the register power without any knowledge of the targeted implementation. We target one specific architecture model and examine the important contributors of register power. As described in Section 1.4.4, the targeted architecture model has a distributed register file organization. Each register file is attached to an input port of a functional unit and is said to be associated with the corresponding operation type.

We also assume that chained operations are merged into a single node. Thus all edges in the control data flow graph represent variables to be stored in register files and all nodes are purely combinational and have no storage components.

Overall register-power model:

Power is dissipated in registers during the read, write, and NOP operations and the overall power consumption is given by:

$$P_{reg} = \left(\sum_i \left(C^i_{eff(rd)} N^i_{rd} + C^i_{eff(wr)} N^i_{wr} + C^i_{eff(nop)} N^i_{nop} \right)\right) V^2_{dd} f \qquad \text{(Eq. 4.7)}$$

where N^i_{rd}, N^i_{wr}, and N^i_{nop} are the number of register reads, writes, and NOPs, respectively on register files associated with operation type i and $C^i_{eff\,(rd)}$, $C^i_{eff\,(write)}$, and $C^i_{eff\,(nop)}$ are the corresponding capacitances switched during these register accesses. The summation is done over all operation types in the algorithm. Computation of the various register accesses (N^i_{rd}, N^i_{wr}, and N^i_{nop}) and the associated capacitance models ($C^i_{eff\,(rd)}$, $C^i_{eff\,(rd)}$, $C^i_{eff\,(rd)}$) are discussed below.

Register access estimation:

With the assumed architecture template, the *number of register reads*, N^i_{rd}, from register files associated with operation type, i, is equal to the total number of inputs of all nodes of type i, in the graph. This is because each time an operation (specified by nodes in the CDFG) is executed, it must read inputs from its operand register files.

The *number of register writes* depends on the assignment of nodes onto specific hardware units. Figure 4.4 shows a simple case where the result of the multiply operation is used by two additions. The figure shows two different assignments of the additions. In Figure 4.4a both additions are implemented on the same adder and the variable x

is written only once, while in Figure 4.4b they are assigned to different adders and the variable is written in two register files.

An upper bound on the number of register writes to files associated with a particular operation type i, can be computed by assuming that for each node, all its outputs of type i are assigned to different units. Similarly, a lower bound can be computed by assuming that for each node, all its outputs of type i are assigned to the same functional unit. The average of the upper and lower bounds is used as an estimate of number of register-writes.

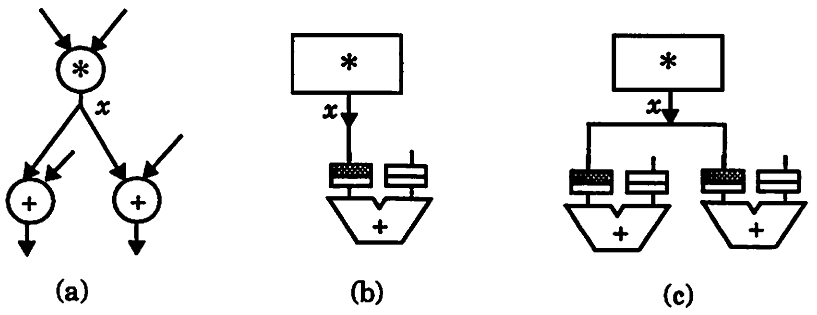


Figure 4.4. Register writes depend on assignment.

Since registers are clocked, power is consumed even when they are not reading or writing. The *number of NOPs* in each register file, is the total number of clock cycles less the number of read and write operations on it.

Register-capacitance:

The effective capacitance associated with register operations is the computed from capacitance complexity models of library blocks similar to those used for functional units. The model for a register operation — read, write, or NOP — is of the form:

$$C_{eff} = c_0 + c_1 r + c_2 n + c_3 rn \qquad \text{(Eq. 4.8)}$$

where r is the number of registers in the register file and n is the number of bits per register.

The number of bits per register is given by the wordlength of the operands of the associated operation type. The average number of registers r, in the register files associated with a particular operation type is estimated from the number of functional units of that type and the total number of registers required by them, both of which are computed using the discrete-relaxation based technique of the Hyper system (refer Section 3.1.3). Given the number of functional units F_i, the number of registers R_i, and the number of operands I_i, associated with an operation type i, average number of registers r, in the associated register files is estimated as:

$$r = \frac{R_i}{F_i \bullet I_i} \qquad \text{(Eq. 4.9)}$$

The overall power estimation scheme for registers is summarized in Figure 4.5.

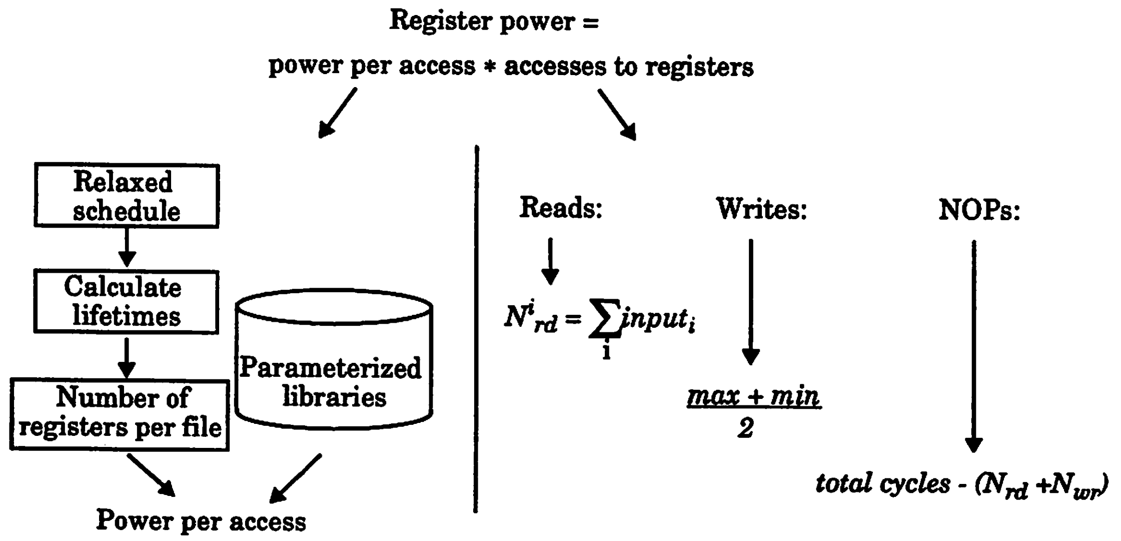

Register power =
power per access * accesses to registers
Relaxed schedule
Calculate lifetimes
Number of registers per file
Parameterized libraries
Power per access
Reads:
Writes:
NOPs:
$N^i_{rd} = \sum_i input_i$
$\frac{max + min}{2}$
total cycles - $(N_{rd} + N_{wr})$

Figure 4.5. Estimating the power consumption in registers.

4.3.3. Interconnect elements

The interconnect elements — buses, buffers, and multiplexors — are rapidly becoming a significant factor in the overall power dissipation in large designs. This section presents a stochastic model for the interconnect power and also for the clock power which is closely related.

4.3.3.1. Buses

The bus power consumption depends on the bus lengths, the capacitive load on them, and the associated activity. Models for these components are discussed below.

Bus access model:

Assuming a white noise activity model, the number of bus accesses determines their activity. In the given architecture model (Section 1.4.4), the result of each operation is written to its destinations via one or more buses. The number of the bus accesses depends directly on the number of edges in the graphs (same as the number of variables). Empirically, the following equation was found to hold:

$$B_{access} = \alpha_0 + \alpha_1 N_{edges} \qquad \text{(Eq. 4.10)}$$

The regression parameters α_0 and α_1 are 11.35 and 0.97, respectively. The model has a very high goodness-of-fit parameters with an R^2 of 0.97 and an F-ratio of 1046.21. The results of the model are shown in Figure 4.6.

Bus length model:

Bus wire capacitances are determined by their lengths, which in turn depend on the area of the overall design. In fact, bus lengths have been shown to be proportional to the square-root of the overall area [60, 117], which is the model used in this work. We first develop a stochastic model for the overall chip area, and use it to predict bus lengths.

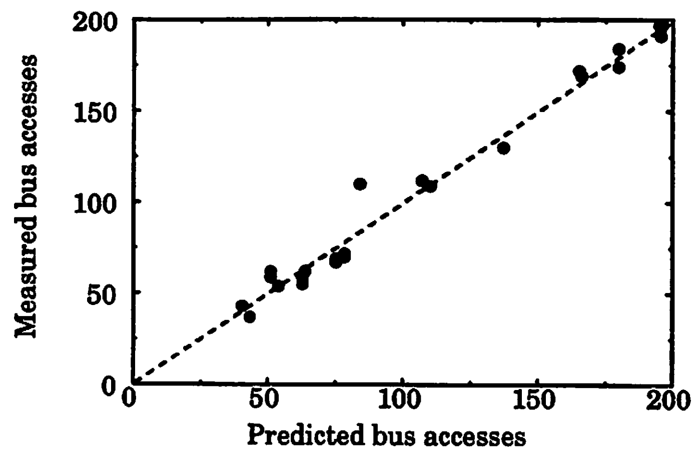


Figure 4.6. Regression model for the number of bus accesses.

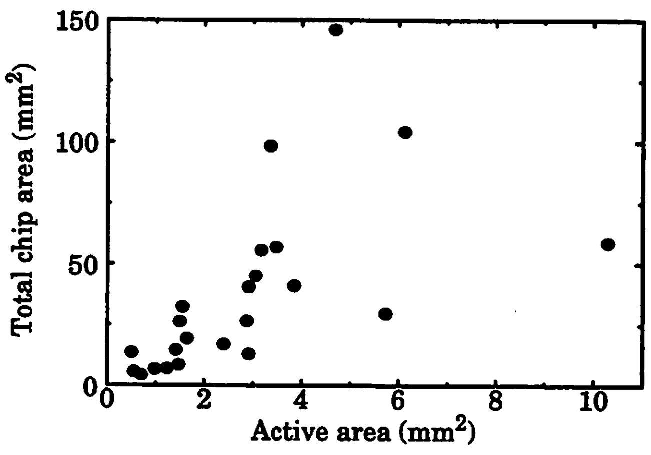


Figure 4.7. Relationship between the total chip area and the active area.

Intuitively, the most important predictor variable is the active area. This includes the combined area of all the hardware resources on the chip except the wiring — functional units, registers, buffers, multiplexors, etc. Figure 4.7 shows the total chip area versus the active area for the selected benchmark set.

Regression analysis yielded the following quadratic model of the total chip area A_{chip} in terms of the active area A_{act} (Equation 4.11).

$$A_{chip} = \alpha_0 + \alpha_1 A_{act} + \alpha_2 A_{act}^2 \qquad \text{(Eq. 4.11)}$$

However, the coefficient of determination for this model is only 0.52 which shows that the active area alone does not explain the variation in the overall area very well.

Further analysis showed that a large part of the overall area is taken by the interconnect, and two parameters — the number of buses and the wordlengths — were considered to account for it. The resulting model is specified by Equation 4.12.

$$A_{chip} = \alpha_0 + \alpha_1 A_{act} + \alpha_2 N_{bits} N_{bus} A_{act} \qquad \text{(Eq. 4.12)}$$

where A_{act} is the active area, N_{bits} is the average word length used and N_{bus} is the number of buses. The regression parameters, α_0, α_1, and α_2, are 2.302, 0.803, and 0.017 respectively and the goodness-of-fit parameters, R^2 and F-ratio, are 0.964 and 150.12, respectively.

The model accounts for the two most important components of the chip area — datapath and interconnect — represented by the second and third term respectively. The third term is explained as follows. Average bus lengths are proportional to the square root of the overall area. It is seen that the square root of the total area linearly correlates to A_{act} (Equation 4.11). Thus, in the third term, A_{act} accounts for the average length of the interconnect and $N_{bus} \cdot N_{bits}$ is the total number of interconnect wires.

Figure 4.8 shows the measured layout area versus that predicted by the model. This model gives average, median, third quartile, and worst case errors of 16.6, 14.0, 22.2, and 43.5%, respectively.

Since the values of A_{act} and N_{bus} are known only after final allocation and schedule, estimated minimum bounds for the execution units and registers are used to compute A_{act} and those for the buses are used to compute N_{bus}, at the algorithm level. These estimates are obtained using discrete relaxation and quick scheduling [100]. The

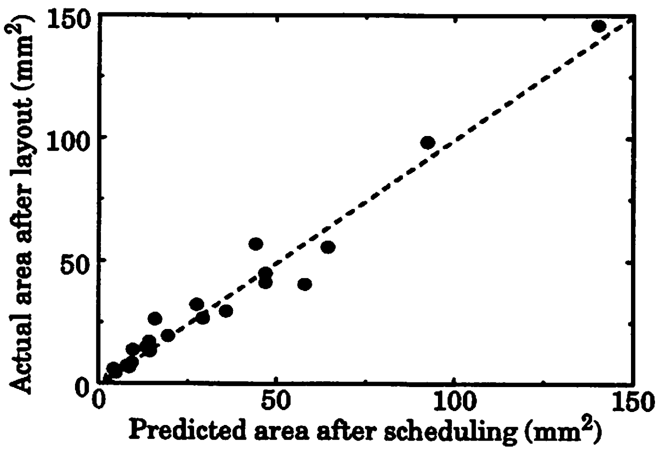


Figure 4.8. Layout area model: predictions vs. chip measurements

model is recalibrated to use estimates of A_{act} and N_{bus} instead of their exact values. Using these estimates instead of exact values reduced the accuracy of the models and the average and worst case errors increase to 42.6 and 96.1%, respectively.

The average bus length, L, is assumed to be proportional to the square-root of the chip area, and is given by:

$$L = \gamma\sqrt{Area} \qquad \text{(Eq. 4.13)}$$

with the proportionality constant, γ, determined empirically to be 0.55.

Overall bus power model:

The total bus capacitance switched per sample period is computed as the average bus capacitance times the number of accesses to buses. This is used directly in the power equation to obtain the total power dissipation in buses.

The capacitance switched per access is composed of two parts — that due to the capacitance of the wire itself and that due to the capacitive load on it. The capacitance of the wire directly depends on the wire length, which is empirically determined from Equations 4.12 and 4.13. The wiring capacitance is calculated from the wire lengths using the average capacitance per unit area and average fringe capacitance per unit length of metal1 and metal2 layers.

The loading on the buses is modeled by a fixed load of 50 fF (about 2 minimum sized inverters in 1.2 micron technology) for each fan-in and fan-out. The total number of fan-ins and fan-outs on the buses are assumed to be the half of the estimated minimum number of functional units. This is based on the assumption that each bus is connected to half of the overall units on average. The wiring and load capacitances are added to obtain the average physical capacitance per bus wire, which is then combined with the wordlength to get the average physical capacitance per bus.

In this section, empirical models have been used to compute the bus lengths and accesses. Figures 4.6 and 4.8 demonstrate that such models can achieve reasonable accuracies to yield valuable high-level estimates. Since the models have identified some important relationships, they can also be used in cost functions during the optimization and synthesis tasks. Both the area and bus access models are used later in this thesis to evaluate various options during the synthesis process.

4.3.3.2. Buffers

Since every bus access is enabled via a buffer, the number of buffer accesses is the same as the number of bus accesses which is modeled in the previous section. The effective capacitance switched per buffer access is easily derived from the capacitance complexity models similar to those used for functional units.

4.3.3.3. Multiplexors

Since multiplexors accesses are strongly dependent on the degree of hardware sharing, we do not compute this component at the algorithm level.

4.3.3.4. Clock

The chip area model presented in the previous section is also used in clock length estimation. On a set of example designs, the length of the clock routing to the datapaths was found to vary between one to three times the square root of the chip area. Based on this observations, the total length of the clock wiring is estimated by the following formula:

$$length = 2 \times \sqrt{ChipArea} \qquad \text{(Eq. 4.14)}$$

To estimate the loading on the clock, we assume that the clock is distributed to all the registers and buffers on the chip and that each offers a capacitive load that is obtained from the hardware library. and buffers are estimated using the discrete-relaxation based technique of the Hyper system (refer Section 3.1.3).

The activity or "number of accesses" on the clock wires is simply the total number of clock cycles in the sample period.

4.3.4. Controller

Recall from Section 1.4.4 that the controller consists of a global finite state machine and a set of distributed local-controllers. In this section we present power models for both the global FSM and the local controllers.

4.3.4.1. Global finite state machine

The global finite state machine is implemented as a counter with conditionals to check for the iterative states (loops). No sophisticated state minimization is performed and a straightforward minimum encoding is used for state assignment.

The relationship between the effective capacitance switched per sample period in the global FSM, C_{FSM}, and the number of states is shown in Figure 4.9. The strong linear dependency seen was used to build the following regression model:

$$C_{FSM} = \alpha_0 + \alpha_1 N_{states} \qquad \text{(Eq. 4.15)}$$

where $\alpha_0 = -22.1$, $\alpha_1 = 4.9$. This model gives negative values if there are less than five states in the machine in which case the capacitance is approximated to zero. The

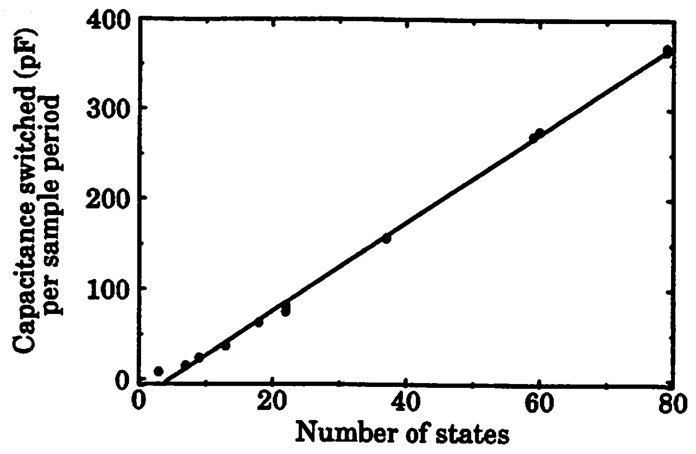


Figure 4.9. Relationship between the capacitance switched per sample period in the global FSM and the number of states.

model has a coefficient of determination of 0.999 and an F-ratio of 6511.38 indicating its high prediction potential.

The strong dependence on the number of states can easily be explained since increasing states result in more state transitions per sample period (higher activity) and also a bigger and more complex FSM (more physical capacitance). In fact, the state transitions are a linear function of the number of states. Consider, for simplicity, an FSM with no iterative loops which sequentially traverses N states. Over the entire sample period, the least significant state bit transitions each cycle (N times), the next bit transitions N/2 times, the next one N/4 times, and so on. The total state-bit transitions over the entire sample period thus show the following linear dependence on the number of states:

$$\begin{aligned} StateTransitions &= N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \ldots + 2 \\ &= N\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{2}{N}\right) \\ &\cong 2N \end{aligned} \qquad \text{(Eq. 4.16)}$$

Estimating the number of states:

For non-hierarchical graphs or hierarchical graphs with non-iterative nodes, the number of states is simply the total number of time steps available (sample period divided by the clock period). However, if the sample period is extremely large (> total number of nodes), then all the clock cycles are not used, and the resulting finite state machine has several idle cycles. In this case, the number of nodes is used as an estimate of the number of used or active states.

For hierarchical graphs with iterative nodes or loops, the number of distinct states in the FSM depend on the allocation of the total time to the different subgraphs or loops. A crude time allocation is realized by first allocating time to each hierarchy node (subgraph) equal to its critical path and distributing the remaining time over the different hierarchy nodes in proportion to their concurrency. The number of states for each hierarchy node is the time allocated to it divided by its iteration count. The total number of states is the sum of the number of states in all its constituent hierarchy nodes (since our implementation uses only a single thread of control — refer Section 1.4.4).

4.3.4.2. Local controllers

Since the global FSM is simply a conditional counter, its power is easily predicted from the number of states. The local controllers however, have several other dependencies that need to be explored. Based on the assignment of the operations to specific hardware units and their schedule, the hardware mapper determines the functionality required from the controllers and creates the truth tables for the control signals. The boolean functions are optimized using MIS II and mapped to a library of standard cell gates.

We first explored the various dependencies of the capacitance switched in the local controllers per sample period. The variables considered were the number of states in the overall machine, the number of control signals (outputs) in each local controller,

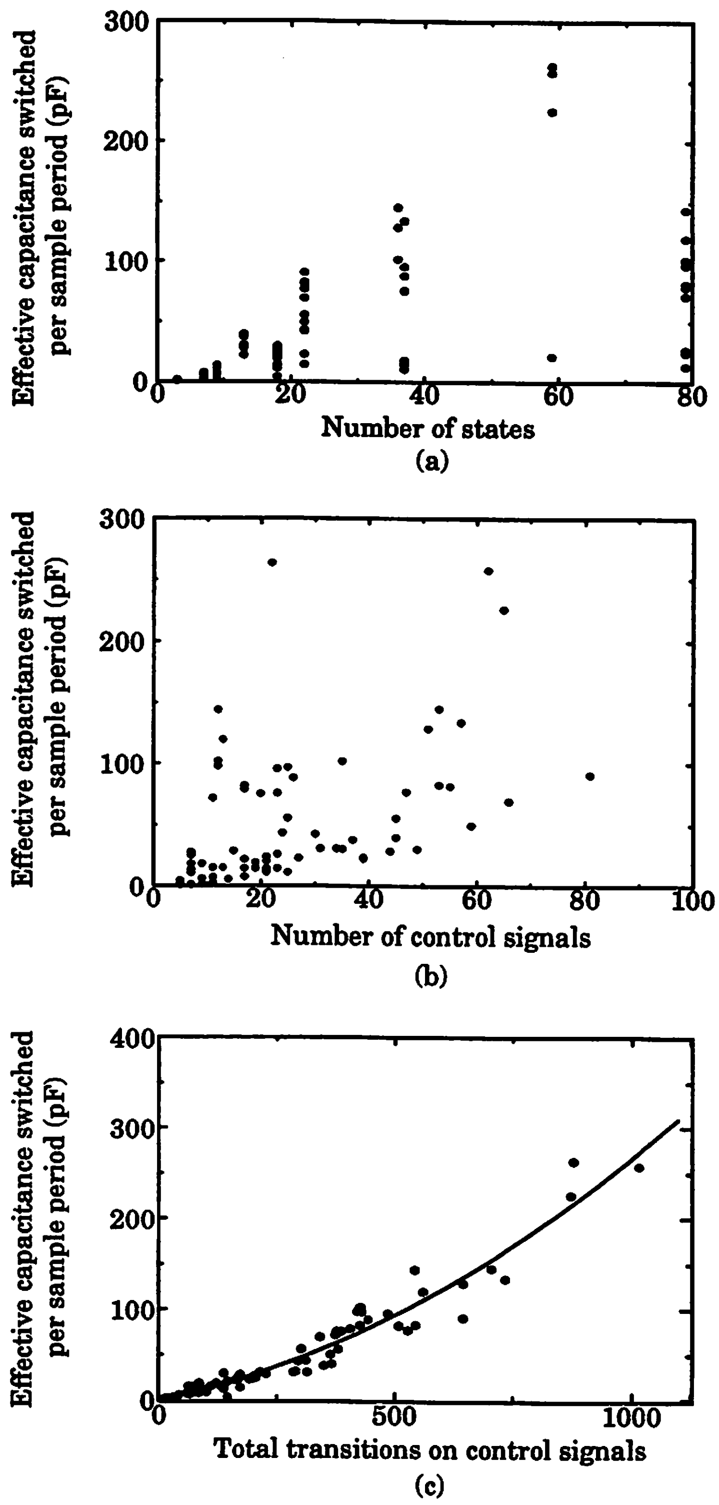


Figure 4.10. Exploring various dependencies of the capacitance of local controllers.

and the number of transitions on the control signals. These dependencies are shown in Figure 4.10. It is seen that the number of states explains part of the variations in the data; no dependence is seen on the number of control signals; and a quadratic dependence is observed on the number of control signal transitions.

The the total capacitance switched in the local controllers in one sample period, C_{lc}, was first modeled in terms of the output transitions N_{trans} using the following quadratic regression model.

$$C_{lc} = \alpha_0 + \alpha_1 N_{trans} + \alpha_2 N_{trans}^2 \qquad \text{(Eq. 4.17)}$$

This model was found to have low R^2 and F-ratio values of 0.78 and 74.0, respectively. A combined model based on the number of states, N_{states}, and the number of output transitions, N_{trans}, was then considered (Equation 4.18):

$$C_{lc} = \alpha_0 + \alpha_1 N_{trans} + \alpha_2 N_{trans}^2 + \alpha_3 N_{States} \qquad \text{(Eq. 4.18)}$$

The values of the constants α_0, α_1, α_2, and α_3 are -238.9, 0.25, 0.000025, and 154.03 respectively. This model is able to explain a large amount of the variability in the data; it has a coefficient of determination of 0.953 and an F-ratio of 423.09.

The model has an average error of 11.52% and a maximum error of 41.06% on a set of 46 examples. The correlation between the estimated and measured capacitances is shown in Figure 4.11.

Estimating intermediate parameters — number of transitions:

While the number of states can be estimated at the high level using the methods mentioned earlier in this section, the number of transitions is not easily estimated from high-level parameters. Since a large percentage of the controller outputs are the enabling/disabling signals for registers, a good correlation was found between the

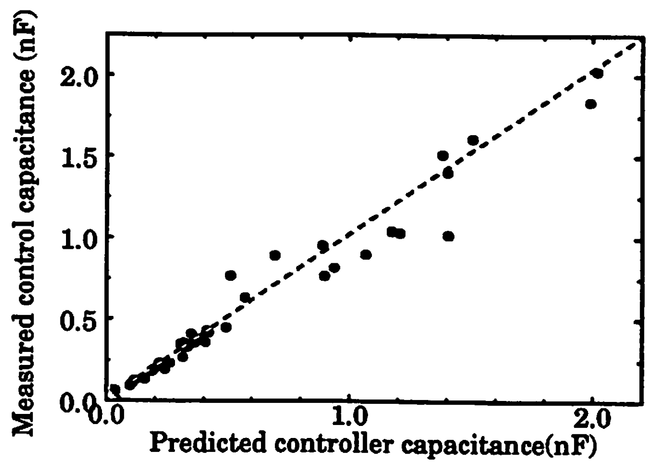


Figure 4.11. Controller model: predictions vs. chip measurements.

output transitions and register accesses as shown in Figure 4.12. The resulting linear regression model is given by Equation 4.19.

$$N_{trans} = \alpha_0 + \alpha_1 N_{regacc} \qquad \text{(Eq. 4.19)}$$

where α_0 and α_1 are 75.32 and 1.20, respectively. This model has reasonably high goodness-of-fit parameters — R^2 is 0.93 and F-ratio is 216.87 — but the problem with it is that the number of register accesses is not known at the high level and must be estimated (as explained in Section 4.3.2) resulting in further inaccuracies.

Due to redundancies in the data collected for each design, we were able to identify other higher-level parameters to predict the transitions and the following multiple regression model derived:

$$N_{trans} = \alpha_0 + \alpha_1 (N_{units} \cdot T) + \alpha_2 N_{nodes} \qquad \text{(Eq. 4.20)}$$

where N_{units} is the estimated minimum bound on the number of functional units, T is the sample period in terms of the clock cycles, and N_{nodes} is the number of nodes in

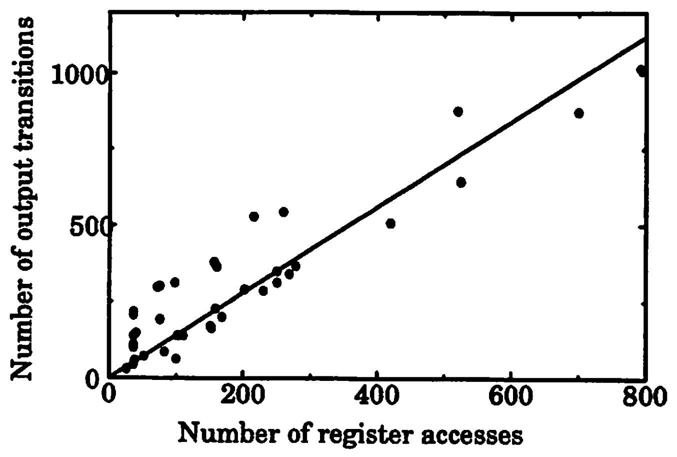


Figure 4.12. Relationship between controller-output transitions and register accesses.

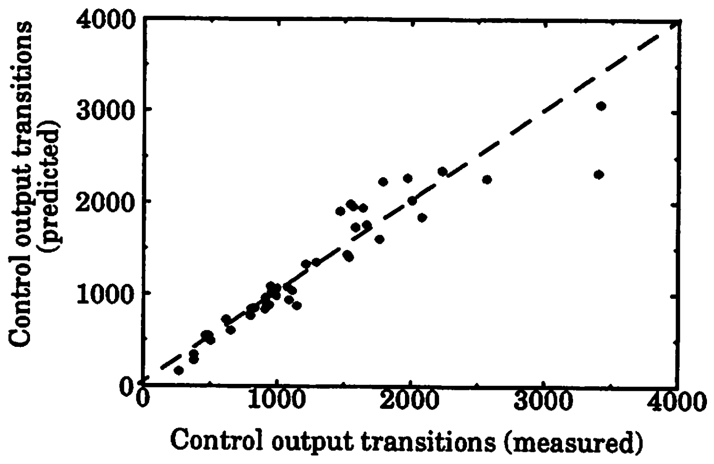


Figure 4.13. Goodness-of-fit of the output transitions model.

the algorithm. The constant values are -207.20, 3.14, and 1.20, for α_0, α_1, and α_2, respectively.

The intuition behind the choice of variables used in the model can be explained as follows. At most, each output transitions once every cycle (ignoring glitches) and thus the total number of clock cycles per iteration of the algorithm is an upper bound for the transitions on each output signal. The total number of outputs signals of the controllers increases with the number of functional units used in the design. Therefore the product of the number of functional units (estimated minimum bound) and the total clock cycles per sample period is considered as a variable. Also, the complexity of the algorithm affects the "work" done by the controllers and hence the output transitions. Therefore, the number of nodes, a measure of the algorithm's complexity was also considered as a variable.

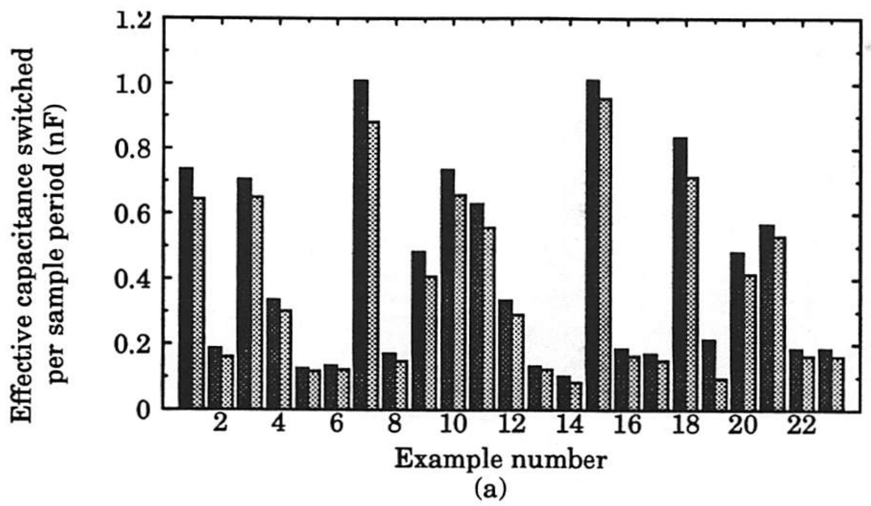
This model has reduced goodness-of-fit parameters with the coefficient of determination being 0.88 and F-ratio being 159.342. However, all the model parameters are known at the high level and no new intermediate variables are introduced. For this reason, this model is preferred over that of Equation 4.19. The goodness of fit of this model is shown in Figure 4.13.

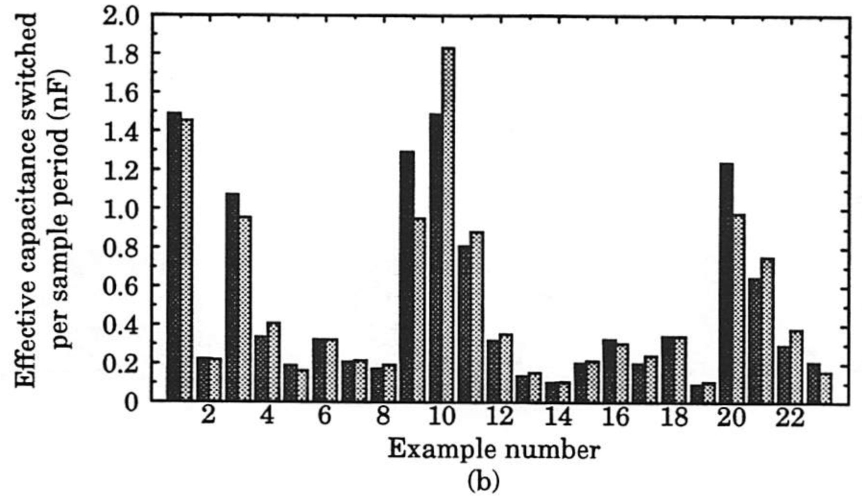
The overall capacitance model for the local controllers is given by Equations 4.18 and 4.20 combined.

4.4. Results

This section compares the algorithm-level estimates obtained using the techniques described in this chapter with the power estimates from SPA, an architecture-level power estimator [65]. The SPA estimation tool has been validated against a switch-level power estimator, IRSIM, and has been found to have errors less than 15%. At architecture level, more information is available about the design. For example, accesses to all resources — registers, buffers, buses, multiplexors — is known which is estimated using both analytical and stochastic methods at the algorithm level. Further assignment of operations to resources and their schedule is known, so that data streams can be captured at the block inputs to be used with activity sensitive block models.

Since floorplanning and routing is yet to be completed at the architecture level, bus lengths are still stochastically modeled using the model of Equation 4.12. However,





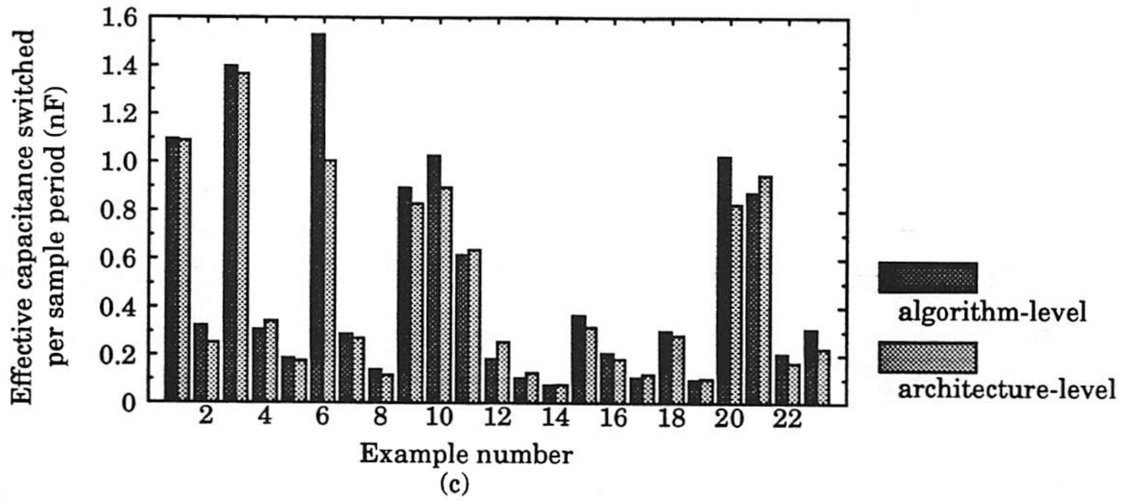
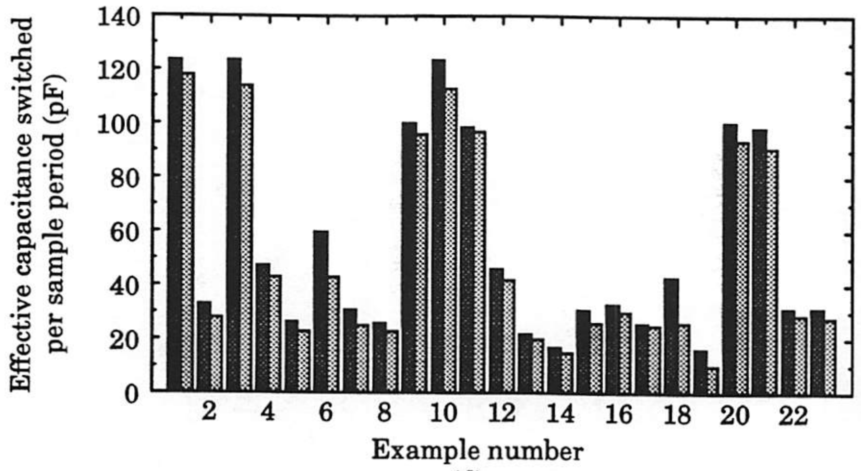
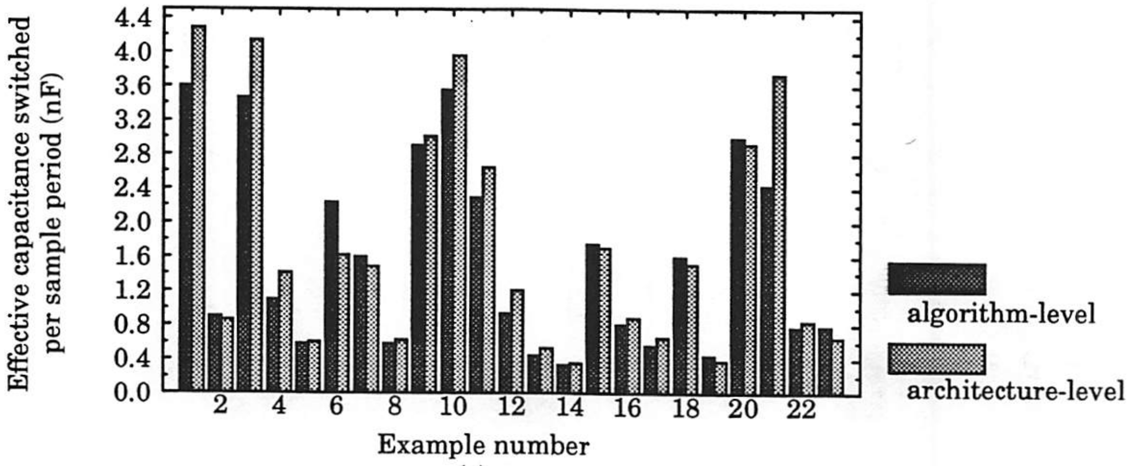


Figure 4.14. Estimates of effective capacitances switched per sample period at the algorithm and architecture levels for various components: (a) functional units, (b) registers, (c) buses, (d) buffers, (e) total.



(d)



(e)

Figure 4.14. contd. Estimates of effective capacitances switched per sample period at the algorithm and architecture levels for various components: (a) functional units, (b) registers, (c) buses, (d) buffers, (e) total..

parameters required for the model — the active area A_{act} and the number of buses N_{bus} — that are estimated at the high level are exactly known at this level. The number of bus accesses is also exactly determined.

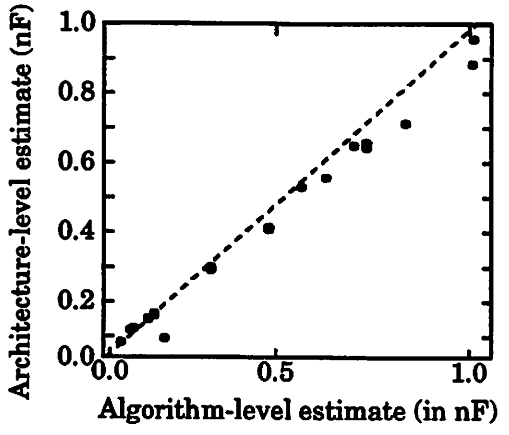
A sample set consisting of 23 designs is used for these experiments. These examples are different from those used in generating the stochastic models.

Bar graphs comparing estimates of the effective capacitances switched per sample period of the various components at the algorithm and architecture levels are shown in Figure 4.14. Notice that the algorithm-level estimates *track* the architecture-level estimates very closely. As mentioned in the beginning of the chapter, since the purpose of the facility is guide high-level decisions, relative accuracy is the main concern as opposed to absolute accuracy. In our results, relative correctness is not achieved in all cases (e.g., design #9 in Figure 4.14b and #6 in 4.14c). The percentage of cases that do achieve relative accuracy is quantified by the correlation between the estimates at the two levels. This in indicated through the correlation coefficients between the two estimates (shown in Table 4.1) and plots of the algorithm-level estimates against the architecture-level estimates (shown in Figure 4.15). A large number of points fall extremely close to the *x*=*y* line, indicating the high correlation between the two estimates.

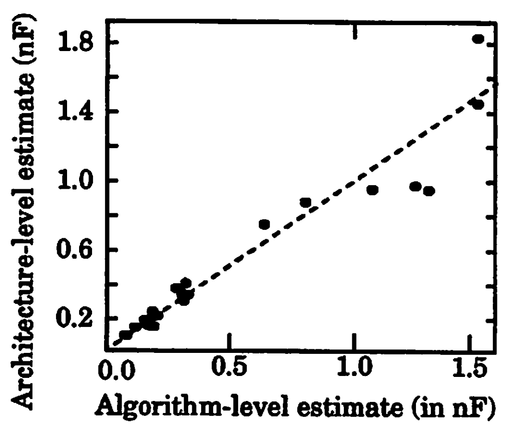
Table 4.1. Correlation between algorithm and architecture-level estimates.

	Correlation coefficient
Execution units	0.99
Registers	0.96
Buses	0.96
Buffers	0.97
Total power	0.97

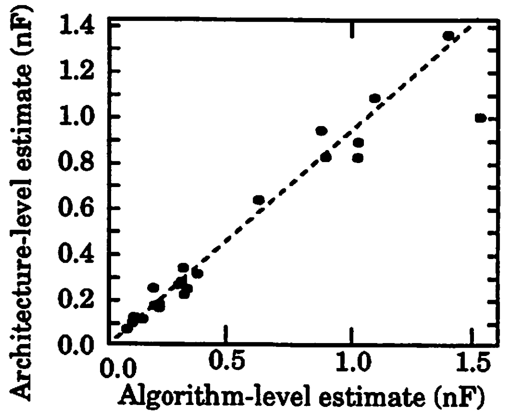
Error statistics of the algorithm-level estimates compared to the architecture-level ones are shown in Table 4.2. Along with the average and worst case errors, the table also shows the median and third quartile errors. It is seen that though the worst case errors are high, the third quartiles are below 20% in all cases showing that 75% of the designs have low errors and large errors are rare. It should be emphasized again that



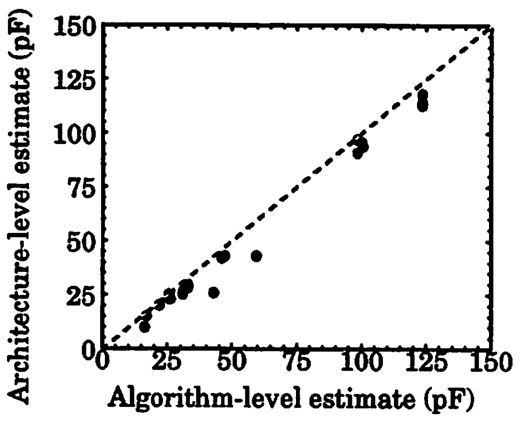
(a) Functional units



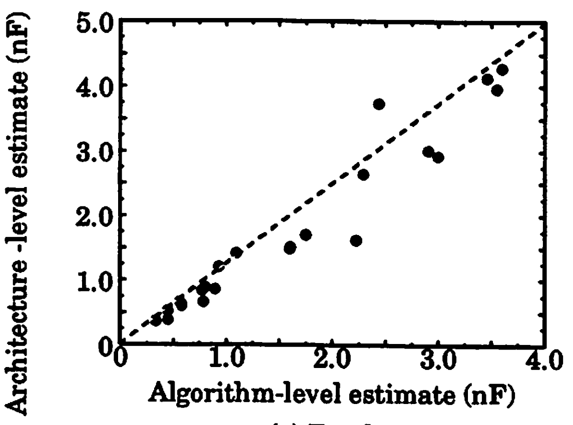
(b) Registers



(c) Buses



(d) Buffers



(e) Total

Figure 4.15. Correlation between algorithm- and architecture-level estimates of the effective capacitance switched per sample period for various chip components: (a) functional units, (b) registers, (c) buses, (d) buffers, (e) total.

the *correlation* between the two estimates is more meaningful than the percentage error between them.

Table 4.2. Error statistics of the algorithm-level estimates compared to architecture-level estimates.

	average	median	third quartile	worst case
Functional units	16.7	13.2	16.0	126.0
Registers	12.0	11.2	17.7	35.9
Buses	14.6	10.86	23.33	52.1
Buffers	15.5	9.6	17.4	65.2

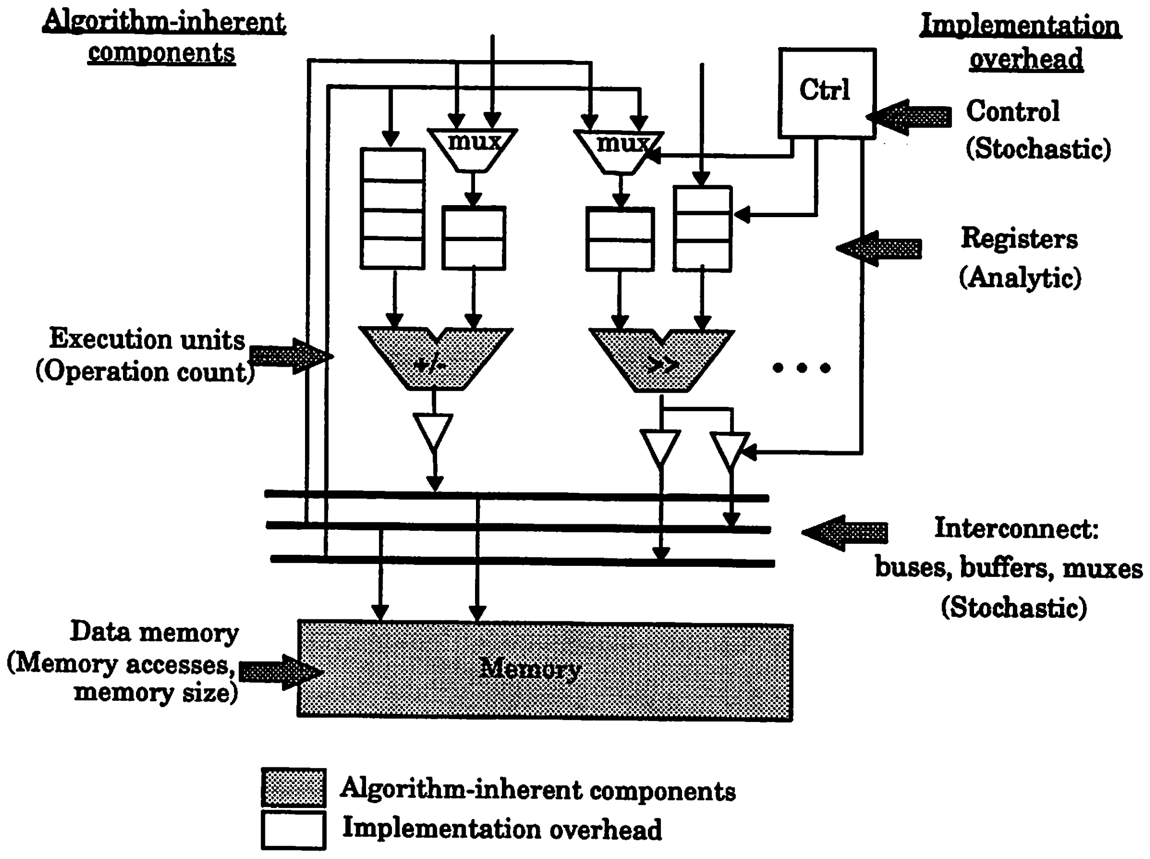

Algorithm-inherent components
Implementation overhead
Ctrl
Control (Stochastic)
mux
mux
Registers (Analytic)
Execution units (Operation count)
+/-
>>
Interconnect: buses, buffers, muxes (Stochastic)
Data memory (Memory accesses, memory size)
Memory
Algorithm-inherent components
Implementation overhead

Figure 4.16. Estimating power: various components and techniques.

4.5. Summary

In this chapter, a algorithm-level power estimation facility has been presented. In order to provide meaningful estimates with the limited information available at the behavioral level, the estimator takes a technology-targeted approach, using information from an underlying architecture model and user-specified hardware library. The various power consuming components on a chip are divided into two parts based on the amount of information available about them at the algorithm level — the algorithm-inherent part and the implementation overhead. A combination of analytic and stochastic approaches is used to address the different characteristics of these components. An overview of the different components and the corresponding estimation techniques is given in Figure 4.16 and the specific power models are summarized in Table 4.3.

The algorithm-level power estimates obtained are compared to estimates from the SPA architecture-level power estimation tool and about 20% average errors are observed. Estimates at the two levels are found to be very high correlated which emphasizes the relative accuracy of the algorithm-level estimates. Extensive use of the proposed estimation facility is demonstrated in the next chapter.

Table 4.3. Summary of the power models for various components[1].

Component	Critical parameters	Main models
Functional units and Memory	Operation counts Complexity parameters Hardware library information	$Power = \left(\sum_{i=0}^{m} N_i C_{eff}(i)\right) V_{dd}^2 f_s$ $C_{eff} = \bar{c} \cdot \bar{n}$
Registers	Hardware library Number of edges	$P_{reg} = V_{dd}^2 f_x$ $\left(\sum_i \left(C^i_{eff(rd)} N^i_{rd} + C^i_{eff(wr)} N^i_{wr} + C^i_{eff(nop)} N^i_{nop}\right)\right)$ $C_{eff} = c_0 + c_1 r + c_2 n + c_3 rn$ $N_{rd} = \sum N_{inputs}$; $N_{wr} = \frac{N_{max} + N_{min}}{2}$ $N_{NOPs} = T - (N_{rd} + N_{wr})$
Interconnect (buses)	Area Active area Wordlengths Number of buses Number of edges	$L = \gamma\sqrt{Area}$ $A_{chip} = \alpha_0 + \alpha_1 A_{act} + \alpha_2 N_{bits} N_{bus} A_{act}$ $B_{access} = \alpha_0 + \alpha_1 N_{edges}$
Interconnect (buffers)	Hardware library Wordlengths Number of edges	$C_{eff} = cn$ $B_{access} = \alpha_0 + \alpha_1 N_{edges}$
Interconnect (clock)	Area Registers/buffers	$length = 2 \times \sqrt{ChipArea}$ $C_{load} = c(N_{buffer} + N_{register})$
Control (global FSM)	Number of states	$C_{FSM} = \alpha_0 + \alpha_1 N_{states}$
Control (local controllers)	Number of states Output transitions Number of units Available time Number of nodes	$C_{gl} = \alpha_0 + \alpha_1 N_{trans} + \alpha_2 N^2_{trans} + \alpha_3 N_{states}$ $N_{trans} = \alpha_0 + \alpha_1 (N_{units} \cdot T) + \alpha_2 N_{nodes}$

1. All constants, α_i, shown in this table are regression parameters and there values are given in the appropriate section in the text.

5
Design Space Exploration

A design environment for power optimization must provide analysis and optimization tools at all levels of hierarchy. In order to facilitate this, a top-down power optimization framework in provided in the Synergy synthesis system as shown in Figure 5.1. The algorithm-level analysis mechanisms presented in the previous chapter are used to explore the behavioral design space; architecture synthesis techniques[1] provide a synthesis path to architecture; and the SPA power analysis tool validates the choices and provides more accurate power estimates at the architecture level. At all phases, power predictions are based on data from pre-characterized cell libraries. Each tool makes use of the information available to it at that level of abstraction. By applying these tools in an integrated, top-down fashion the user is able to begin with a high-level description of the desired functionality and systematically converge to the optimum low-power algorithm and architecture.

At the highest level of abstraction, several decisions like the algorithm selection, clock selection, voltage reduction, and transformations lead to large variations in the power consumption. In order to rapidly evaluate the myriad design alternatives at this level, the algorithm-level power estimates presented in the last chapter are encapsulated into an exploration tool called *Explore* which is described in the next section. Section 5.2 illustrates the use of Explore to examine various degrees of freedom in the algo-

1. The synthesis algorithms used in the Synergy synthesis system are explained in Chapters 7 and 8.

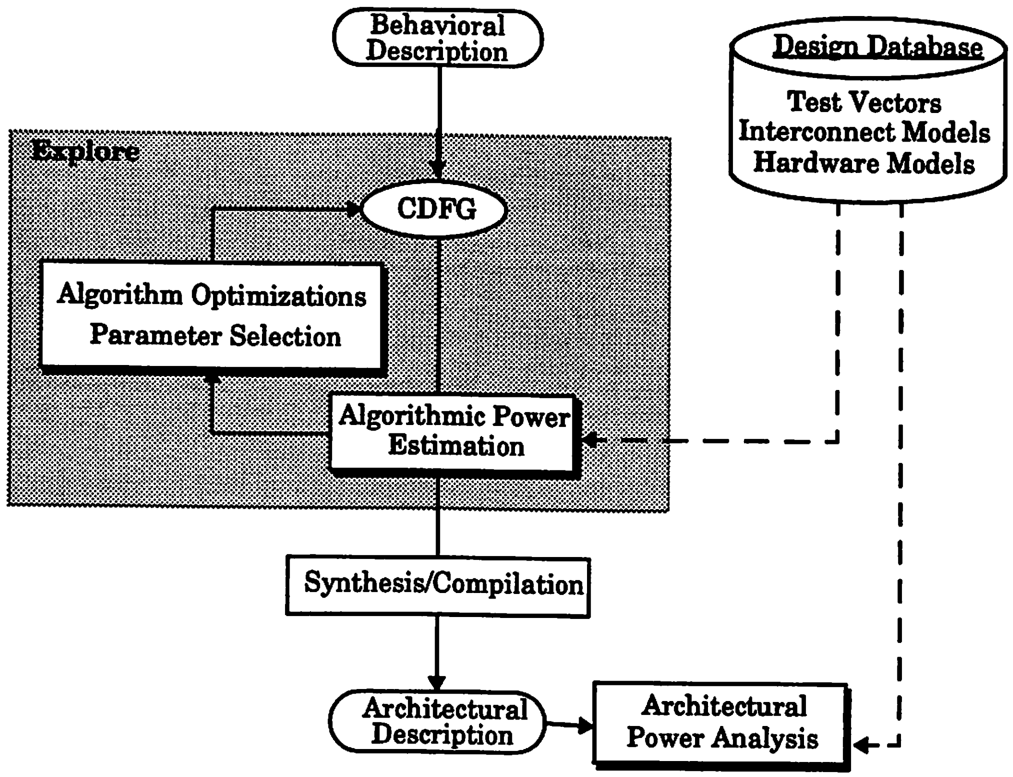


Figure 5.1. Integrated top-down power optimization approach in the Synergy synthesis system.

rithmic design space. Finally the effectiveness of the overall CAD environment is illustrated through an extensive case study in Section 5.3.

5.1. Explore

The goal of the *Explore* design-aid is to help the designer in searching through the algorithmic design space. This tool evaluates several design points for the given algorithm by varying a set of parameters and iteratively invoking the high-level power estimator. The output of the tool is a set of graphs that plot the variation in overall area and power consumption as a function of the chosen parameter.

In the current implementation, two parameters are used as independent variables, namely the voltage supply, V_{dd}, and the clock period, T_{clk}. Selecting these parameters at the high level allows an accurate evaluation of the delays of the hardware units that can be used to drive synthesis and optimization. At lower levels, when the architecture and micro-architecture are finalized, the allowed variation in the V_{dd} and T_{clk} is restricted, which limits a complete exploration of the design space and constrains the associated power savings.

The effect of these two parameters on the power and area of real-time applications is discussed in detail in Section 2.2.1. One of the important effects derives from the fact that different functional units may be optimal for implementing a given operation at different values of V_{dd} and T_{clk}. To incorporate this effect, Explore maintains a database of the functional units selected for each operation at different values of these parameters.

Each operation type has an identifier that depends on its functionality and wordlength. Thus a 4-bit addition operation is called addition#4 and a 6-bit left shift operation has left_shift#6 as identifier. A hash table indexed by the identifiers of the various operation types is used to store the hardware modules selected for them. Each entry of this table is an array indexed by the value of the independent variable (V_{dd} or T_{clk}). Each array element stores a pointer to the library cell to be used to implement the corresponding function at the given values of V_{dd} and T_{clk}. Library cell selection is based on the greedy strategy explained in Section 3.1.2. Each element is also associated with a *duration,* which is defined as the number of clock cycles required for implementing the operation on the selected library cell at the given parameter values. Operation durations affect several aspects of the implementation including the critical path of the algorithm and the hardware requirements for implementing it within the specified throughput constraint. Though they do not affect the algorithm-inherent component of power dissipation, they have a direct impact on the implementation overhead part, which is influenced by the critical path, hardware requirements, etc.

Supply-voltage variations start at the highest voltage specified (default is 5V) and continue with steps of 0.25V down to 1V. If a feasible hardware module is not found for any operation (no hardware cell can complete the operation within an allowed maximum number of clock cycles) at any voltage level, the search is terminated and lower voltage levels are invalidated.

The extremities of the clock-period variation are selected by searching for the maximum and minimum time requirements over all operations. The minimum clock period considered corresponds to the fastest speed of the fastest operation. This is because there is no gain from using a faster clock since all operations would require multiple clock cycles to execute. The maximum clock period considered is the slowest speed of the slowest operation. This is because a longer clock period will only result in wastage of the added time since none of the operations can effectively use it.

Estimation of the effective capacitance of all components is iteratively performed at each of the design points starting from the maximum clock period or supply voltage. This capacitance is appropriately combined with the operating supply voltage and user-specified sample frequency to obtain power numbers. Overall area estimates are also reported.

To speed up the exploration process, the hardware database table is analyzed for similarities across different iterations and estimates of components are computed only if they are predicted to change from the previous iteration. Estimation changes are predicted by comparing the selected hardware cells and their durations for all the operations between the two iterations. If all the hardware cells selected and their durations are the same as in the previous iteration, no estimations need to be re-performed. If the hardware cells selected are the same but the durations are changed, only the estimates of the implementation overhead components are recomputed. In other cases, all estimates are recomputed.

During the estimation process, the critical path of the algorithm is compared to the available sample period in each iteration to ensure the feasibility of the design. In the case of supply voltage variation, once a design found to be unable to execute within the given throughput constraint, the process is terminated and lower voltages are not explored since the timing requirements of an algorithm only increase at lower voltages. However, this is not the case with clock period variations since both the number of clock cycles required and the number of clock cycles available increase with reducing clock lengths. A design may therefore be infeasible at a particular clock cycle but may yield a feasible solution at longer or shorter clock periods.

As mentioned above, the output of the tool is a set of curves plotting the area and power of the application as a function of the independent variable selected. The curves help the user in selecting the implementation that offers the best trade-off between area and power. They can be used to evaluate several design aspects such as the effect of transformations, technologies, and algorithms. The two parameters can also be varied simultaneously to plot a 3-dimensional surface so that globally optimal points may be selected.

5.2. Investigating various degrees of freedom

The complex interactions of the different factors make it difficult to analytically derive the optimal design parameters. Explore offers an easy way to graphically exploit the various trade-offs. This section studies power optimization on different examples through *Explore curves*. Several degrees of freedom for design optimization are studied — clock selection, voltage selection, transformations, algorithm selection, and hardware comparison. The degrees of freedom presented here are by no means comprehensive and the purpose is to illustrate the type of design optimizations that can be explored at the high level using the estimation capability.

5.2.1. Clock period selection

Figure 5.2 shows the effect of varying the clock period on the power and area of an 8x8 discrete cosine transform (DCT). The power trend is dominated by the control component. At shorter clock periods, the number of clock cycles for the implementation (and hence the number of states) increases, leading to the greater power consumption in the controller. At longer clock periods (>80 ns), the overall power rises due to an increase in interconnect power but this effect is relatively less pronounced. The area shows the opposite trend from power and increases with the clock period due to the fewer clock cycles being available to implement the algorithm.

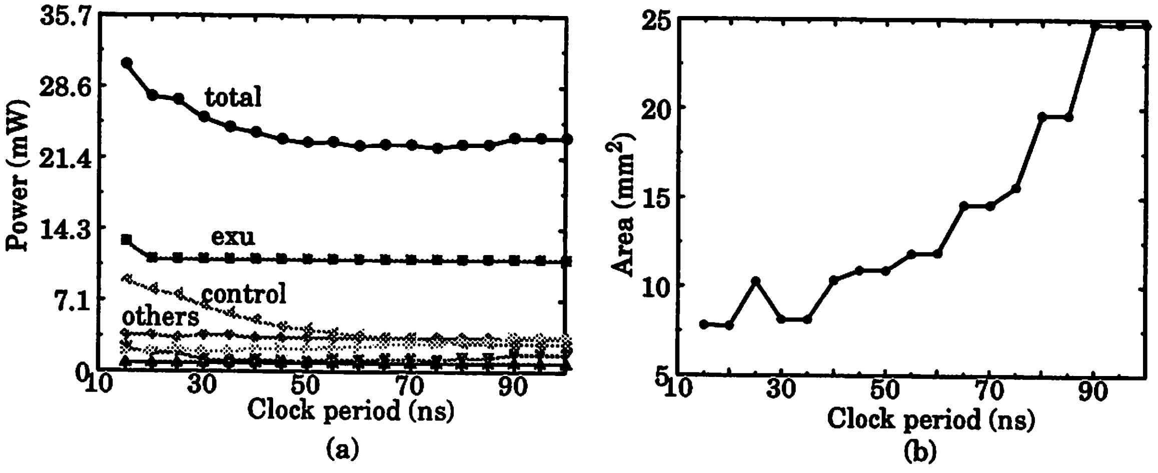
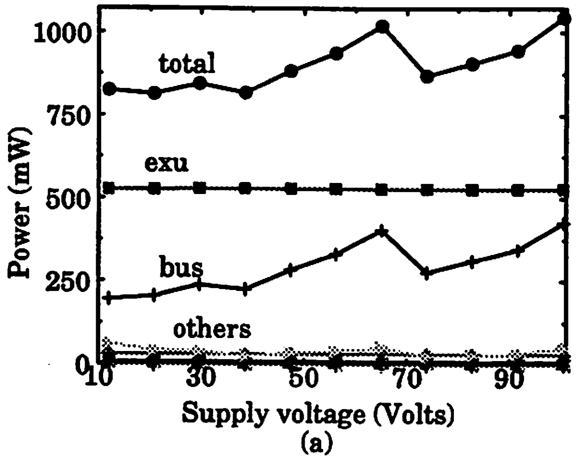


Figure 5.2. Clock period exploration for an 8x8 discrete cosine transform (DCT).

In another example, consider the clock exploration curves of a 16-point FFT shown in Figure 5.3. In this case, the power trend is dominated by the interconnect power which is highly affected by the overall area. Due to the different conflicting effects of the clock period on the overall area explained in Section 2.2.1, the area is seen to have a non-monotonic variation with the clock period.

The difference in the relative importance of different power components in the two examples can be explained by examining some of their characteristics shown in



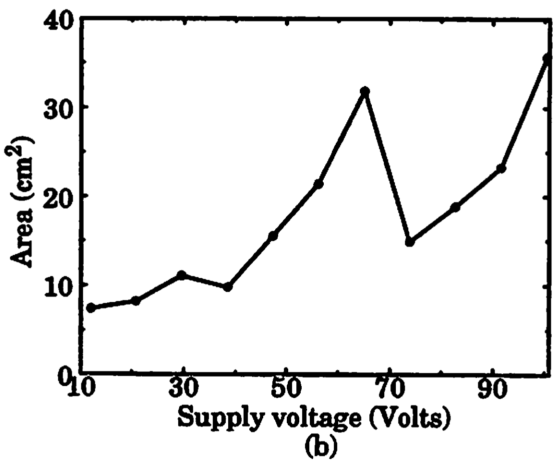


Figure 5.3. Clock exploration curves for a 16-point fast fourier transform (FFT).

Table 5.1. While their sample periods differ only by a factor of 2, their complexities, in terms of the number of operations, differ by a much larger factor. The FFT has 6.9 times more multiply operations and 4.7 times more additions and subtractions. Therefore, in a particular time period, the FFT performs about 3 times more operations than the DCT example on average, and requires a more parallel (and hence larger) implementation. Further the data transfers (measured by the number of edges) are much higher in the FFT example. Due to the larger area and more data transfers, the effect of the interconnect is greater in the FFT example.

Table 5.1. Characteristics of the two examples studied.

	Sample period (ns)	Number of multiplies	Number of additions	Number of subtractions	Number of edges
8x8 DCT	1.4	16	26	-	53
16-point FFT	0.7	110	99	23	290

5.2.2. Transformation selection

Figure 5.4 displays some of the effects of transformations on a design. The example used is a 14th-order wavelet filter with a throughput constraint of 3.33 MHz, and a 20 ns clock period. The original design consumes 133 mW power which can be reduced to 110 mW by lowering the supply voltage to 4.5 V. However, if the supply voltage is further reduced, the design fails to meet the throughput constraint. Substituting constant multiplications by shifts and adds results in a 50% decrease in power. However the lowest possible supply voltage is the same as before since the critical path of the design is unaltered.

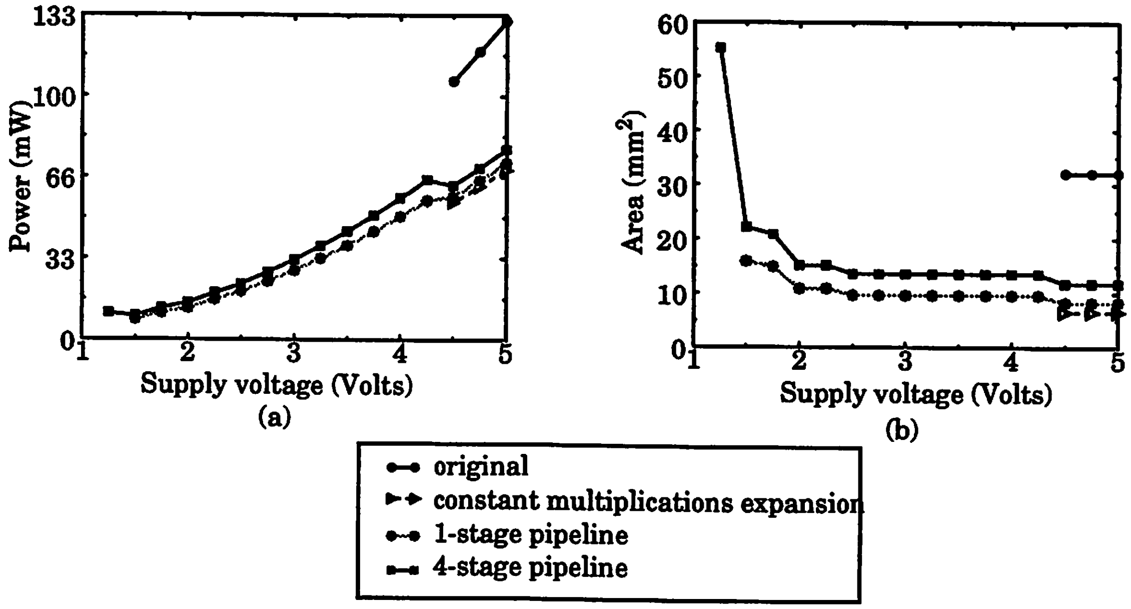


Figure 5.4. Exploring the effect of transformations on the power and area of a 14th-order wavelet filter with 3.33 MHz sampling frequency.

Pipelining the transformed version lowers the critical path of the design, making it possible to lower voltages to 1.5 V and get a further 2x power reduction. The curves also make it clear that it is not advantageous to over-pipeline a design. For a given supply voltage, 1-stage pipeline has lower area and power than a 4-stage pipeline due to the cost of extra pipeline registers. It is also seen that the supply voltage should not be indiscriminately decreased. Although using 4 pipeline stages allows voltage reduction up to 1.25 V, the power increases when the voltage is decreased from 1.5 V to 1.25 V due to a large rise in area (shown in Figure 5.4b) which causes an increase in interconnect power. The sudden increase in area is due to increased functional units needed to implement the same algorithm at a lower supply voltage.

5.2.3. Algorithm selection

For realizing a given function, the choice of the algorithm has a large impact on the quality of the final implementation. We evaluate two algorithms for comparing the mean squared distance of a vector, X, from each of two vectors, C_a and C_b. This operation is the core computation in vector quantization systems [36]. The two algorithms are given in Equations 5.1 and 5.2 [69].

$$MSE_a - MSE_b = \sum_{i=0}^{7}\left(C_{ai} - X_i\right)^2 - \sum_{i=0}^{7}\left(C_{bi} - X_i\right)^2 \qquad \text{(Eq. 5.1)}$$

$$MSE_a - MSE_b = \left(\vec{C}_a^2 - \vec{C}_b^2\right) + \sum_{i=0}^{7} 2\left(C_{ai} - C_{bi}\right) \times X_i \qquad \text{(Eq. 5.2)}$$

The energy versus supply voltage curves of the two algorithms are shown in Figure 5.5. While the first algorithm has 16 multiply, 14 add, and 17 subtract operations, the second one has 8 multiply and 9 add operations and consumes lower power in functional units. Secondly, the second algorithm has smaller area and hence shorter average interconnect lengths and dissipates less power in the interconnect. Also, due to a shorter critical path, the second algorithm has a feasible implementation at much lower supply voltages. Using the second algorithm and lowering the supply voltage to 2V reduces the power by a factor of 25 compared to the first algorithm at 4.5V.

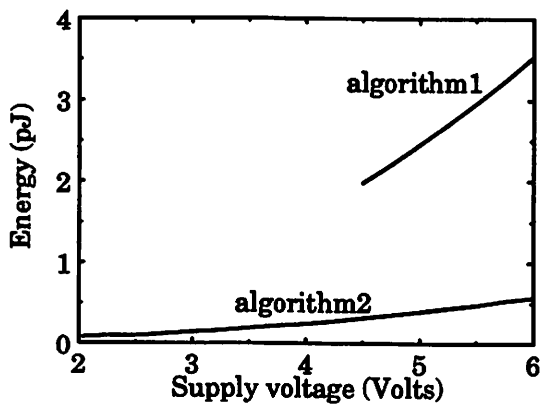


Figure 5.5. Evaluating two vector comparison algorithms.

5.2.4. Hardware comparison

Explore can also be used in studying the effects of using different hardware units for the same application. The example used here is a Volterra filter with a sampling speed of 1.9 MHz and a selected clock period of 25 ns. The filter is transformed first by replacing constant multiplications by adds and shifts to reduce the number of multiply operations and then by using associativity to reduce the critical path. Two implementations of with different adder implementations — ripple adder (RCA) and carry-select adder (CSA) — are compared.

Figure 5.6 shows the Explore curves obtained by varying the supply voltage. At 5 V, the RCA implementation performs better than the CSA implementation in terms of both power and area. However, the area of RCA implementation increases drastically at 4.5 V which impacts its bus power. As a result, it has higher power and area at voltages below 4.5 V. Further, it does not yield a feasible implementation below 2.75 V while the CSA version yields a feasible solution even at 2 V, giving about 2x power and area savings over the best RCA version.

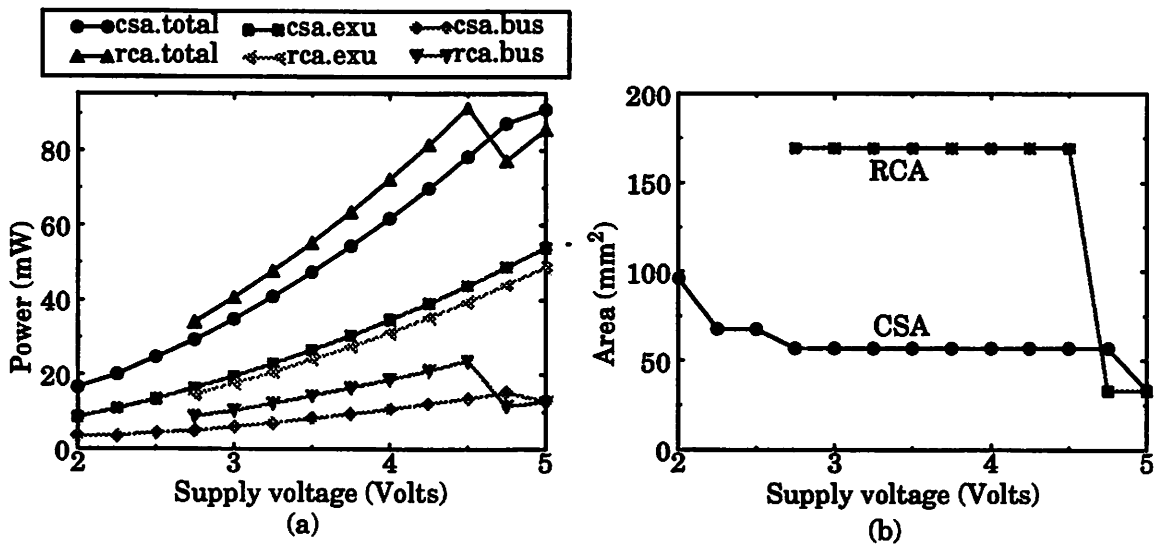


Figure 5.6. Power and area curves for implementations of a Volterra filter with carry select and ripple carry adders.

5.3. Case study: the Avenhaus filter

In this section an eighth-order bandpass Avenhaus filter example [4] is used to illustrate the top-down optimization flow, highlighting both general and specific issues at each stage. We consider the filter structures proposed in [25] to implement the Avenhaus transfer function. While these structures were further studied in [92] for complexity and area comparisons, no study has been done to compare these structures based on power consumption metrics.

We assume that the designer is required to implement an Avenhaus filter with an overall throughput constraint of 2.75 MHz imposed by the surrounding system. He/she is free to select the filter structure, the supply voltage, the clock period, the specific transformations, and the final assignment strategy. We start with a preliminary evaluation, comparing various structures that can be used to implement the required transfer function. Next, we explore the design space applying several transformations and supply voltages, accepting or rejecting them based on evaluations from the high-level estimators. After narrowing down the design space, we use architecture-level analysis to verify and refine our design decisions. We conclude with a review of the power savings achieved at each stage of optimization.

5.3.1. Preliminary evaluation

We first consider different structural implementations of the Avenhaus filter proposed by Crochiere [25] — cascade, continued-fraction, direct-form II, ladder, and parallel structures. We first estimate the power consumption of each assuming a straightforward implementation of each structure with a 5V supply voltage and clock frequency chosen so that each operation completes in one clock period. The resulting estimates are presented in Table 5.2. In order to explain the differences in power dissipation, the table also presents several key parameters that influence the power consumption — the maximum throughput (critical path and maximum sample frequency) and the complexity (operation count and required word length). The throughput is obtained from the Synergy system using the Lager low-power cell library [14]. The energy numbers presented are obtained at a frequency corresponding to the maximum possible sample frequency for each structure.

Table 5.2. Preliminary analysis of the various Avenhaus filter structures.

	Critical path (ns)	Max. sample freq. (MHz)	Word Length	Number of multiplies	Number of additions	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

It is clear that the complexity of an algorithm has a major impact on the power consumed — the cascade and parallel implementations have the lowest operation counts and the smallest word lengths and also the lowest energies. Increased wordlengths contribute to larger physical capacitance and increased activity and higher operation counts result in increased activity and may also necessitate increased hardware and routing, resulting in larger wiring capacitance.

In the current forms, only the cascade and parallel structures can meet the 2.75 MHz throughput constraint at 5V, the critical paths of the other three being too long. Since transformations can have a large impact on both the critical paths and the operation counts, we also evaluate transformed versions of each of the structures instead of simply eliminating the other three.

5.3.2. Programmable vs. dedicated hardware

One transformation that can be very useful in power reduction is expansion of multiplications with constants into additions and programmable shifts. This may result in lower power requirements since only additions and shifts corresponding to 1's in the coefficient are performed while an array multiplier implementation performs an addition for every bit of the coefficient even if it is a 0. On the other hand, the dedicated array multiplier performs shifts by hard-wired routing, while the add-shift version uses programmable shifters, latches partial products between stages, and requires additional control. In certain cases the overhead due to shifters, latches, and additional control can offset the gains from the reduced additions. The effect of this transformation on the critical path depends on the specific example being considered.

The critical path and energy values (at the corresponding maximum sample frequencies) after constant multiplication expansion are shown in Table 5.3. Not only is the power consumption increased, all the structures have longer critical paths and lower voltages cannot be tried. This transformation is therefore rejected for all the examples. It is important to note that this is not a general conclusion about constant-multiplication expansion and the trade-offs must be evaluated on a case-by-case basis.

Table 5.3. Results of constant-multiplication expansion for the Avenhaus structures.

	Critical path (ns)	Max. sample frequency (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

5.3.3. Critical path reduction and voltage scaling

In their current forms we cannot reduce the voltage for any of these designs and still meet the throughput constraint. Since decreasing the critical path opens the possibility of voltage reduction and can have a large impact on the power dissipation, we next try to decrease the critical paths using pipelining. The resulting critical paths obtained with various levels of pipelining are shown in Table 5.4. It is seen that some of the filter structures that could not meet the throughput constraint initially become feasible, while those already feasible become faster. The options that result in feasible solutions are highlighted.

We generated Explore curves with varying voltages at 2.75 MHz sampling frequency to compare these solutions. Figure 5.7 shows the curves for the designs that result in lowest power dissipation for each structure. It is seen that voltages (and energies) can be appreciably reduced for all examples (except the continued fraction) by applying pipelining. Further, the optimum level of pipelining is not always equal to the maximum pipelining due to the overhead introduced by pipeline registers. For example,

Table 5.4. Critical path reduction through pipelining[1].

	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	-	-	-

1. The critical path should be less than 364 ns to meet the 2.75 MHz throughput constraint.

the maximally pipelined version of the cascade lead to a minimum energy of 5.1 nJ, about 15% more than the two-stage pipelined version which consumes 4.4 nJ.

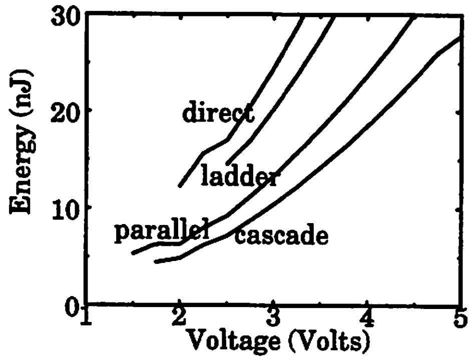


Figure 5.7. Voltage reduction (and its effect on energy) after "optimal" pipelining.

The results from the exploration curves are summarized in Table 5.5. It is apparent that the cascade and parallel versions still yield the best quality solutions. Therefore, at this point we remove the continued-fraction, direct-form, and ladder structures from consideration. We do not eliminate the parallel form at this stage since it gives results close to the cascade, and the errors inherent in the high-level estimation tools does not allow us to resolve differences that are so small.

In order to select the supply voltage we examine the area penalty associated with voltage reduction for the cascade and parallel form structures. The area exploration

Table 5.5. Effect of reducing voltage for the pipelined versions.

	# stages	Critical path	Minimum 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

curves are presented in Figure 5.8. It is seen that the minimum voltages are accompanied by large area penalties. However, at slightly higher operating voltage of about 2V, the area penalties are much less severe and, the energies are not significantly higher (Figure 5.7). The Explore curves allow the designer to evaluate the area penalties associated with parallel implementations and make informed decisions based on the area-energy trade-offs. In this case study we avoid the large area penalties by selecting a 2V supply voltage.

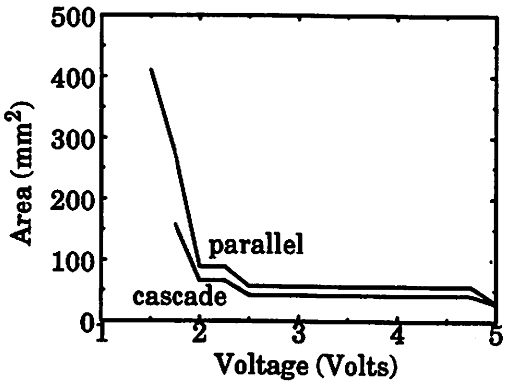


Figure 5.8. Area trade-offs for the pipelined cascade and parallel versions.

A large number of power reduction techniques are available for optimization at the algorithm level (refer Chapter 2). The purpose of this case study is to present a general methodology and show how high-level tools can be used to facilitate the selection of these techniques, rather than to enumerate all possible implementations of the Avenhaus filter. Instead of further algorithmic exploration, therefore, we now look at architecture-level optimization and analysis to refine and verify the decisions made so far.

5.3.4. Architectural exploration

Using algorithm-level power estimation and exploration tools, we have been able to narrow the design space to two filter structures, the cascade and parallel forms, with two stages of pipelining each, operating at 2.75 MHz sampling speed and 2 V supply voltage.

We now synthesize the selected algorithms and use architectural power analysis to evaluate and refine the results. Analysis tools at the architecture level provide the accuracy required to make more fine-grained choices and to evaluate the effects of the synthesis steps performed after algorithm-level exploration. We use sampled speech data as input for architecture-level simulation and power analysis. A detailed breakdown of the energy estimates at the algorithm and architecture levels is given in Table 5.6.

Table 5.6. Comparing algorithm- and architecture-level estimates for the cascade and parallel forms.

	Cascade		Parallel	
	Algorithm	Architecture	Algorithm	Architecture
Exu	2.42	1.65	3.27	2.05
Registers	0.59	0.50	0.64	0.52
Control	0.62	0.62	0.73	0.73
Buffers	0.08	0.06	0.09	0.06
Mux	-	0.18	-	0.21
Bus	1.01	1.04	1.31	1.31
Clock	0.31	0.25	0.38	0.33
Total	5.03	4.30	6.42	5.21

Since it does not account for the highly correlated nature of the speech input data, the algorithmic power estimator overestimated the power consumed by the execution units by 47% for the cascade and 60% for the parallel versions. It is important to note, however, that the errors in the algorithmic power estimates are systematic over-estimates rather than random errors since the input data used for both designs is the same, showing that relative classifications made during algorithmic design space exploration are meaningful.

Based on the accurate architecture-level estimates, we are able to select the cascade filter as our final low-power implementation. This design can be further optimized at the architecture level using techniques that will be proposed in the next few chapters of this thesis: exploiting locality and regularity of the algorithm. Here we simply present a further improvement obtained from assigning the operations to hardware units to maximize the locality of the data transfers in the final implementation. It is seen that this approach reduces the power consumption of the buses and functional units (Table 5.7). The idea of exploiting locality for low power is developed extensively in Chapter 7.

Table 5.7. Reducing power at the architecture level through local assignment.

	Default assignment	Local Assignment
Exu	1.65	1.44
Registers	0.50	0.50
Control	0.62	0.62
Buffers	0.06	0.07
Mux	0.18	0.18
Bus	1.04	0.29
Clock	0.25	0.25
Total	4.30	3.35

Further improvements can be obtained by logic and circuit level optimization of the different components in the architecture which are not discussed here.

5.3.5. Gains from design space exploration

Table 5.8 illustrates the large gains obtained (27x) by exploring the design space over different algorithm structures, transformations, and supply voltages. Selecting the correct algorithm (cascade) saved a factor of 3 in power, compared to the worst case (direct form). Moreover, the direct form (at 89.5 nJ) could not achieve the required 2.75 MHz sampling rate. If the algorithms were compared for the same throughputs, the cascade would actually be even more than 3x better. Counter to what we may expect, expanding multiplications into shifts and adds is not beneficial in this case and increased the power dissipation. Transformation for reducing critical path, e.g. pipelining, is found to further reduce the power by a factor of more than 6. Finally, local assignment helped to reduce the power by another 22%.

Table 5.8. Summary of power savings obtained by exploring the algorithmic and architectural space for the Avenhaus filter implementation.

	Inputs	Voltage (V)	Energy (nJ)	Overall power reduction
Worst algorithm (direct form)	UWN	5.0	> 89.5	1
Best Algorithm (cascade)	UWN	5.0	27.7	3x
Cascade (after constant-multiplication expansion)	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

5.4. Summary

In this chapter, we have illustrated the importance of high-level exploration and the value of an integrated top-down design environment for power optimization. An exploration framework is presented to rapidly evaluate several points in the algorithmic design space. Some of the various degrees of freedom available at the high levels are highlighted and orders of magnitude power savings are demonstrated on several examples.

Lastly, an extensive case study is presented to demonstrate the use of the overall top-down CAD environment. The case study takes the reader through the design flow, indicating the low-power techniques apropos to each step and describing how a hierarchy of analysis and optimization tools can be used to converge to a desired low-power solution. The proposed approach is shown to lead to more than an order of magnitude reduction in power. This is achieved by a thorough search of the design space, which would not have been possible without the assistance of high-level design tools.

6

Interconnect Power

The rest of the thesis focuses on architecture-synthesis approaches for power minimization. Before considering any techniques in this direction, it is worthwhile to analyze some designs to identify power bottlenecks and opportunity areas at the architecture level. In a given design flow, optimization opportunities at the highest level are unrestricted by prior decisions, while opportunities at lower levels are limited by choices made earlier in the design process. In particular, the effect of architecture-level techniques is restricted by decisions taken at the system and algorithm levels. For example, the number of operations is fixed by the algorithm and cannot be affected by the choice of the architecture. Therefore, it would be expected that the architecture synthesis tasks of allocation, assignment, and scheduling would have very little effect on the power consumed by the functional units. In fact, architecture synthesis tasks affect the functional unit power only by changing the activity of their input signals.

The key, therefore, is to *target those elements that are highly influenced by the architecture* and depend relatively less on the decisions made at higher levels. In the following section we compare manual and automated design implementations and identify the interconnect as a significant bottleneck in automated approaches. We also show that the interconnect power is greatly influenced by architecture-level techniques.

Sections 6.2 and 6.3 overview related work in interconnect optimization and estimation, respectively. In Section 6.4, we present extensive models for the interconnect power to allow us to evaluate the results of synthesis approaches presented in the next two chapters.

6.1. Interconnect — a bottleneck in overall power dissipation

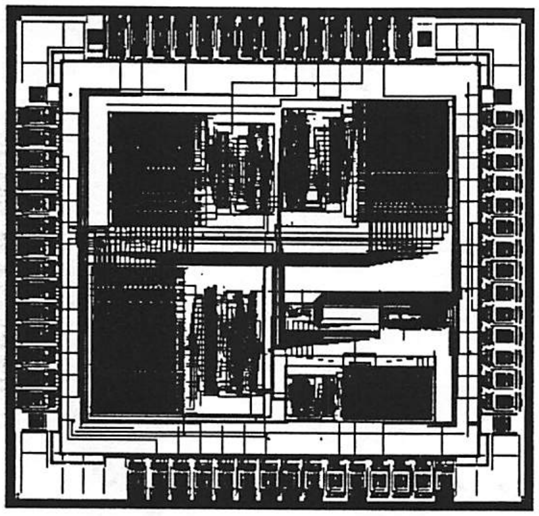
We begin with a study performed by Wu [133] that compared two different implementations — an automatically-generated maximally time-shared version and a manually-generated fully-parallel version — of a QMF sub-band coder filter. This filter is a typical example of DSP applications that are targeted in this thesis.

In the manual design, a number of optimizations were used to obtain power savings in the various components. The power consumption of both versions is documented in Table 6.1 and the layouts are shown in Figure 6.1. For the same supply voltage, an improvement of a factor of 10.5 was obtained at the expense of a 20% increase in area.

Table 6.1. Power consumption in the various components for the fully-parallel and maximally time-shared implementations.

	Time-shared	Fully-parallel	Improvement factor
Functional units	8.52	1.03	8.3
Registers	9.76	1.08	9.0
Buses	23.69	1.40	16.9
Multiplexors	3.77	0.25	15.1
Buffers	4.36	0.35	12.5
Others	23.99	2.92	8.2
Total	74.09	7.03	10.5

The breakdown of the power consumption of the two versions is shown in Figure 6.2. The interconnect elements (buses, multiplexors, and buffers) consume a large percentage of the total power (43%) in the time-shared version. Moreover, large improvement factors were obtained for the interconnect components — 16.9, 15.1, and 12.5 for buses, multiplexors, and buffers, respectively — compared to the those for other com-


(a)

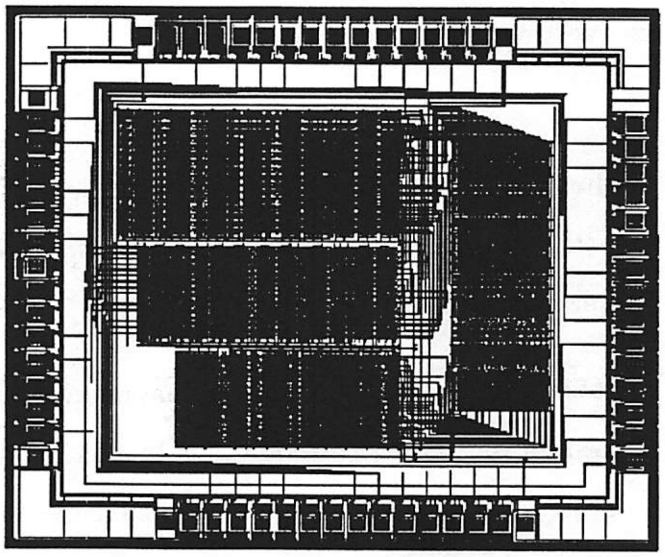

(b)

Figure 6.1. QMF sub-band coder filter [133]: (a) maximally time-shared version, (b) fully-parallel version.

ponents. These improvements are mainly due to dedicated communication and reduced usage of multiplexors and buffers in the manual design (Table 6.1). As a result of these optimizations, the contribution of the interconnect power was reduced to 28% of the total power (Figure 6.2b). The above observations point to two facts — (i) interconnect components may *consume a large percentage of the total power* and (ii)

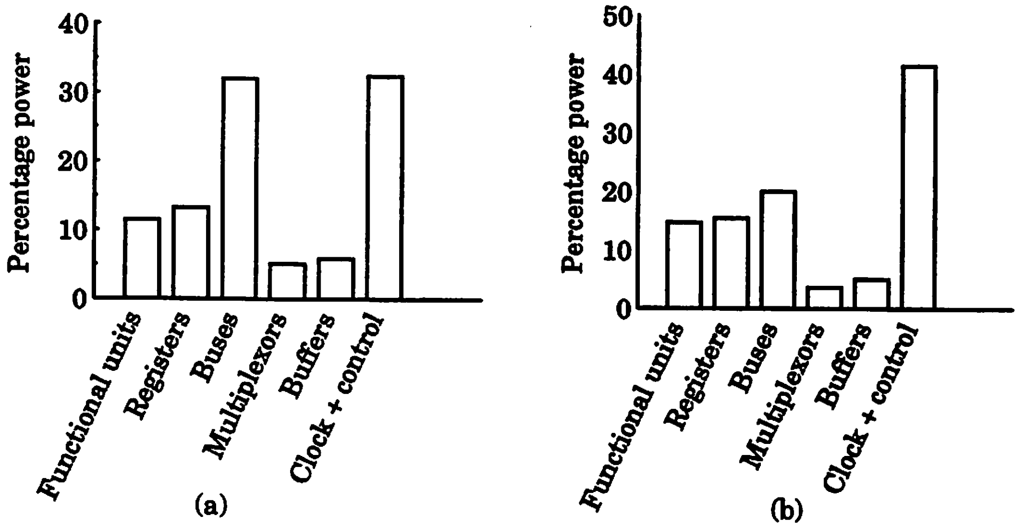


Figure 6.2. Percentage power breakdown for the QMF filter: (a) time-shared version, (b) fully-parallel version.

their power consumption is *highly influenced by architecture-level design decisions* — both of which indicate a high potential for power reduction by targeting the interconnect at the architecture level.

It is seen in the above example that maximal hardware sharing does not give the best results for power optimization. In fact, typical designs from the Hyper synthesis system have shown that buses alone consume 10 to 40% of the total power and the interconnect elements together contribute 15-50% of the total power (refer Figure 4.2). In general, area can be traded-off to a certain extent to reduce the power consumption in buses, multiplexors, and control. While in the above example, the fully-parallel implementation resulted in large power savings with low area overhead, this may not always be the case. Completely parallel implementations may be too area intensive and may not necessarily result in reduced interconnect power. If the area overhead is too high, the increase in the required bus lengths (and capacitances) may offset power savings due to other factors.

The architecture-synthesis techniques presented in the next two chapters aim to capture some of the optimizations made manually in the above example through *automated* techniques while maintaining a balance between the maximally time-shared and the fully-parallel implementations.

6.2. Related work in interconnect optimization

Originally, most high-level systems focused on functional-unit optimizations. Interconnect mapping was performed after scheduling and functional unit allocation and assignment, and usually even after register allocation. This reduced the degree of freedom for interconnect optimization. Recently however, there has been a growing interest in interconnect optimization in the high-level synthesis arena. In [76], McFarland showed that the interconnect can have a first order impact on the quality of the overall design. The interconnect trade-offs were examined in [59] by studying several examples synthesized with the Mabal synthesis system.

Several synthesis systems have incorporated interconnect minimization as one of the primary goals, accounting for it even during operator assignment. The Facet synthesis system attempts to explore the design space to minimize the functional units, registers, and interconnection units (multiplexors) using clique partitioning schemes for each of the optimizations, and user input to determine their priorities [123]. One drawback is that they do not account for the cost of buses. Mandal et al. integrate a measure of interconnect area into a well-known clique covering algorithm to allow simultaneous reduction of registers and interconnections [71]. In [118], Stok proposes an annealing-based approach incorporating interconnect costs along with functional unit costs during scheduling. Park presented an approach to interconnect optimization based on path sharing [90]. This approach is closely related to some of the interconnect optimization ideas in this thesis and is discussed in Chapter 8. Partitioning approaches for interconnect cost reduction are explore in APARTY [61] and the

BUD [77] systems. These are closely related to the techniques presented in Chapter 7, where they are discussed in further detail.

All the works mentioned above have studied the problem of interconnect optimization for area. Ours is the first work that considers the tasks of interconnect power reduction. The two problems are different in that while area reducing techniques reduce the number of interconnect wires, power reduction methods must also consider the cost of accessing them. In other words, for power minimization, it is important to reduce the accesses to the long buses rather than simply reducing the total number of buses. This topic is addressed in the next two chapters.

6.3. Related work in interconnect estimation

Several research works in the literature have addressed the problem of interconnect estimation and the closely related problem of layout area estimation from both the high and logic levels.

A classical approach is based on the Rent's rule [62] which gives an empirical relationship between the number of gates in a well placed gate array and the number of external connections (pins). The rent's parameter is determined empirically for various classes of circuits. Feuer showed that for a circuits that followed the Rent's rule, the wire length distributions $q(r)$, are of the form $q(r) = r^{-2(2-p)}$ where p was the Rent's parameter and r was the wire-length [30]. Masaki also uses a rent's rule based model for interconnect length estimation in [74].

Some models target a specific cell placement strategy. For example, Heller presented a stochastic model for wirelengths assuming a 1-dimensional placement of cells [45], and El Gamal extended this model in [33] to address a two-dimensional placement strategy. Standard cell estimation models are presented in [60, 117]

Another approach is based on adding up shape functions of the basic cells in a design and hierarchically constructing the shape functions of the bigger blocks. Interconnections between the child blocks can be incorporated in the shape function model of the parent blocks. This approach is used in [77, 137]. A hybrid approach combining the shape function based scheme with analytic models is reported in the LAST layout area estimation system [60].

We propose a layout area and bus length estimation scheme targeted at our architecture model and placement strategy. Several concepts from the techniques mentioned above are used in our estimations and are explicitly mentioned.

6.4. Bus and clock capacitance models

In this section we present models for predicting the bus and clock power at the architecture level. Since the synthesis techniques presented in the next two chapters target interconnect power reduction, good estimates of these components are important to derive meaningful conclusions. We therefore spent considerable time generating layouts for several designs to build extensive models for the bus power estimation.

The bus power consumption is determined by the physical bus capacitance and the associated activity. The capacitance switched per access is composed of two parts — that due to the capacitive load on the wire and that due to the capacitance of the wire itself. The loading on the buses is modeled by adding a fixed load of 50 fF (about 2 minimum sized inverters in 1.2 micron technology) for each fan-out and fan-in. The capacitance of the wire directly depends on the wire length, which is not determined until after placement and routing and is therefore estimated using an empirical model. Once the bus lengths are determined, the wiring capacitance is calculated using the average capacitance per unit area and fringe capacitance per unit length of metal1 and metal2 layers. The wiring and load capacitances are added to obtain the physical capacitance of the bus which is combined with the activity model to derive the power dissipation.

Bus lengths are obtained from overall area estimates which are based on the stochastic models described in Section 4.3.3. These are repeated here for convenience:

$$A_{total} = \alpha_0 + \alpha_1 A_{act} + \alpha_2 N_{bits} N_{bus} A_{act} \quad \text{(Eq. 6.1)}$$

$$L = \gamma \sqrt{A_{total}} \quad \text{(Eq. 6.2)}$$

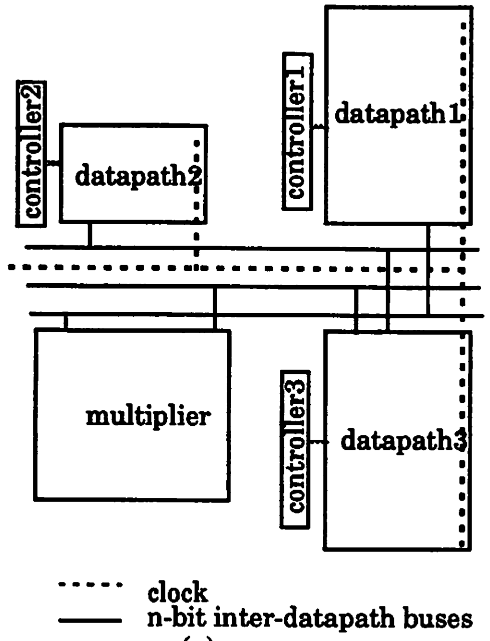
For details of these models and the values of the constants, α_0, α_1, α_2, and γ the reader is referred to Section 4.3.3. The model parameters, A_{act}, N_{bus}, and N_{bits}, that are estimated at the algorithm level are exactly known at the architecture level. Also, the bus accesses are exactly known and information on signal activity at block inputs can be obtained from simulation.

Although bus lengths are still modeled stochastically, estimates at the architecture level can incorporate some effects from the floorplanning and routing stages. In order to allow this, we use a *physical model* in addition to the architecture specification.

6.4.1. Physical model

Recall that in the architecture model, each functional unit has register files at its inputs and, if required, input multiplexors and output buffers (Figure 1.1). The functional unit, along with the associated registers, multiplexors, and buffers is called a *functional unit set*. Functional unit sets communicate with other sets via the interconnect network.

Keeping this architecture in mind, the physical model shown in Figure 6.3 is used. The model accounts for the floorplanning strategy used in the Hyper synthesis system. In this model, two or more functional unit sets may be grouped into the same datapath. While communication between datapaths occurs through global buses, within the datapaths, units are stacked in a bit-slice fashion and over-the-cell wiring is used for communication between them. Additionally the placement of the datapath is assumed to be in two rows connected by the interconnect network as shown. This



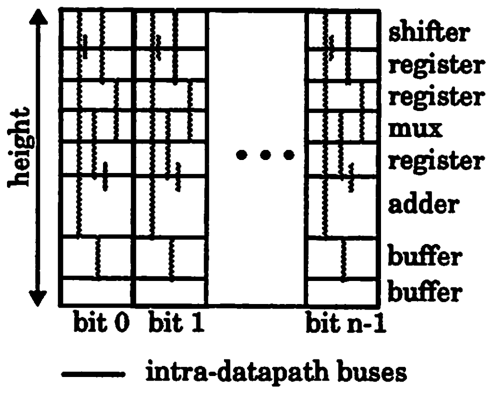


Figure 6.3. Routing strategy: (a) typical floorplan with inter-datapath buses and clock routing, (b) intra-datapath routing in a typical datapath.

inter-datapath routing strategy is depicted in Figure 6.3a and the intra-datapath routing in Figure 6.3b.

The physical model allows the consideration of several refinements to the area model to account for changes in the synthesis styles. For example, any desired clustering of functional unit sets into datapaths can be specified and the physical model can be used to distinguish between the global and local buses.

6.4.2. Incorporating effects of datapath clustering

Buses are divided into two main categories — inter-datapath and intra-datapath — and different models are used to estimate their lengths. The estimation is performed hierarchically, estimating the intra-datapath lengths first, and using this information in the calculation of the global bus lengths.

Intra-datapath connection lengths are estimated using a linear model, called the *stacked datapath* model. The average length of over-the-cell connections is estimated to be proportional to the cumulative height of the units in the datapath. The constant, γ, is empirically evaluated to be 0.3.

$$L = \gamma\left(\sum_i H_i\right) \qquad \text{(Eq. 6.3)}$$

Inter-datapath buses are modeled using Equations 6.1 and 6.2. This model is used for global buses for the entire chip and also for inter-datapath buses that are concentrated on a certain part of the chip. In each case, A_{total} refers to the region in question (whole chip or certain part).

The user is allowed to specify any clustering of functional unit sets into datapaths. If no clustering is specified, each functional unit set is assumed to be in a separate datapath and all buses are assumed to be global. If a clustering of functional unit sets is specified, it is assumed that units that are in the same cluster are placed physically close to each other. In particular, all units in a cluster, besides array elements such as memories and array elements, are merged into a single datapath and the array elements are placed nearby.

The lengths of the wires within the datapath use the stacked datapath model. Since array elements cannot be stacked onto datapaths, the wire lengths for clusters with array elements is calculated using a hybrid of the stacked and inter-datapath models. For buses between units other than the array elements, lengths are estimated using the stacked datapath model. For the connections to the array element, the inter-datapath model (Equations 6.1 and 6.2) is used where A_{total} is the estimated cluster area, the active area, A_{act}, is the sum of the areas of all the units in the cluster and the number of buses includes all buses that transfer data from the array element to any other unit in the same cluster.

Global buses between the various clusters are modeled by the inter-datapath model. In this case, A_{total} refers to the overall area of the chip, A_{act} is the sum of the total areas of all the clusters (computed as explained above) and N_{bus} is the number of global buses. The model is thus hierarchical — at the overall chip level, each cluster is a black box and only the buses in between them need to be considered; within each cluster, only modules in the cluster and the internal inter-datapath buses are considered.

Figure 6.4 compares the average measured length of the wires in the datapaths with those predicted by the stacked datapath model. Points on the dotted line indicate exact estimates.

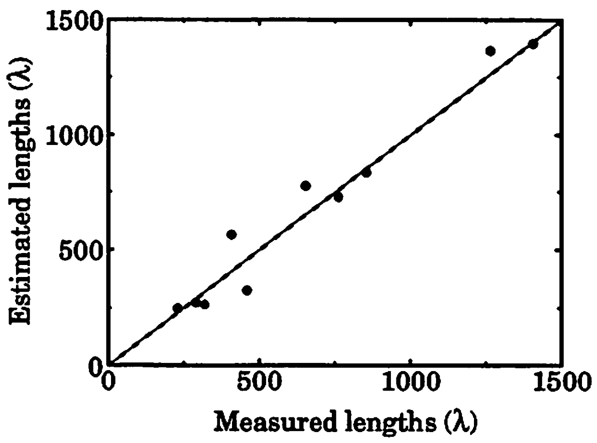


Figure 6.4. Validation of the linear model for intra-cluster buses.

The inter-datapath bus-length model was validated for clustered and non-partitioned versions of three designs (Table 6.2). Though the model was accurate to within 20% for non-clustered designs, it overestimated the bus lengths for the clustered designs. This appears to be due to the fact that unclustered implementations have many small blocks while the clustered designs have few large blocks at each level of hierarchy leading to a different behavior. This can be addressed by re-evaluating the coefficients of the inter-datapath model for clustered designs. Note that the over-estimation of bus lengths, and therefore the capacitance, in the clustered designs leads to conservative over-estimates of the power relative to unclustered designs.

Table 6.2. Measured and estimated global bus lengths for a few designs.

	Name	Measured bus lengths	Estimated bus lengths	Percentage difference
Non-partitioned designs	FFT	3.87	3.89	+0.5
	Parallel IIR	3.53	3.34	-5.4
	Wave digital	1.56	1.29	-17.3
	Cascade	2.10	2.82	+34.3
Partitioned designs	FFT	1.49	3.01	+102.0
	Parallel IIR	2.18	2.24	+2.8
	Wave digital	0.41	1.18	+188.0
	Cascade	0.95	2.73	+187.4

6.4.3. Incorporating effects of large variances in fan-out/fan-in

At the gate level, wire lengths are modeled as being directly proportional to the fan-out of the wire [74] but this effect is largely ignored in conventional architecture-level models [60, 80]. This is because in a bus-based interconnect architecture with maximal bus multiplexing, the fan-outs and fan-ins of buses show very low variances and the fan-outs of buses are assumed to be constant over the entire design.

Here we consider some scenarios in which this may not be true. In a multiplexor-based interconnect architecture, each unit has a dedicated output bus and the fan-out of each bus is the same as that of the corresponding unit. Therefore, large variances in bus fan-outs may be seen. Similarly, in a tristate-buffer based interconnect architecture, each unit has a dedicated bus at each input port and the fan-ins are determined by the sources to that input port. Also, certain design styles or synthesis schemes may optimize a selected set of buses thus producing large variances in bus fan-outs. In these cases, the constant fan-out model is not valid. In this section we present a new model that accounts for the large variances in fan-outs.

After examining several designs we found that the linear relationship between fan-out and bus length [74] holds even at the high level. The length of any bus, i, is estimated as L_{pp} times its fan-out, F_i, as given in Equation 6.4. L_{pp} represents the length of a bus with single fan-out and is constant over all buses for a given design.

$$L_i = F_i L_{pp} \qquad \text{(Eq. 6.4)}$$

The length, L_{pp}, of a single fan-out bus is assumed to be proportional to the square root of the area of the chip (Equation 6.5).

$$L_{pp} = \gamma \sqrt{A_{chip}} \qquad \text{(Eq. 6.5)}$$

Using equation 6.4, the total wiring area on the chip is given by Equation 6.6, where N_i is the number of bits in bus i, W_p is the wiring pitch and $\mathcal{F}$ is the sum of the fan-outs times the number of bits over all buses.

$$\begin{aligned} A_{wire} &= W_p \sum_i L_i N_i \\ &= W_p L_{pp} \sum_i N_i F_i \\ &= W_p L_{pp} \mathcal{F} \end{aligned} \qquad \text{(Eq. 6.6)}$$

The total area of the chip is the sum of the active area of the datapaths and the area consumed by the wires. Since our designs are targeted for datapath intensive designs we ignore the area consumed by the control. The total area is therefore given by Equation 6.7.

$$\begin{aligned} A_{chip} &= A_{act} + A_{wire} \\ &= A_{act} + W_p L_{pp} \mathcal{F} \end{aligned} \qquad \text{(Eq. 6.7)}$$

Substituting Equation 6.7 for the chip area in Equation 6.5 gives the following quadratic equation for L_{pp}:

$$L_{pp}^2 = \gamma^2 (A_{act} + W_p L_{pp} \mathcal{F}) \qquad \text{(Eq. 6.8)}$$

Solving this equation gives the following closed form expression for L_{pp}:

$$L_{pp} = \frac{\gamma^2 W_p \mathcal{F} + \sqrt{\left(\gamma^2 W_p \mathcal{F}\right)^2 + 4\gamma^2 A_{act}}}{2} \qquad \text{(Eq. 6.9)}$$

The constant in the model, γ, was found experimentally in the following way. A few algorithms were synthesized using both the Hyper and the E-template based synthesis system and layouts were created using a silicon compiler. L_{pp} was determined by summing up the lengths of all buses on each chip and dividing by the total number of fan-outs over all buses, and the active area was measured from the layout. The scalar constant, γ, was determined from Equation 6.8 to be 0.72, 0.80, 0.81, 0.88 and 0.80, 0.68 for the six chip-layouts generated. The mean value of γ, 0.78, was selected for our model.

6.4.4. Activity model

A critical difference between behavioral and architectural estimates lies in the knowledge of the activity. At the behavioral level, since hardware sharing and scheduling are not yet performed, the activity at the inputs of the hardware blocks is unknown and uniform white noise model is used. Under this abstraction, the activity in any component is given simply by the number of accesses to it. While the number of accesses to functional units is easily determined from the behavior, those to buses is not, and a stochastic model based on the number of edges was used (Equation 4.10).

At the architecture level, information of both the hardware sharing and the schedule is available and the exact number of accesses to each component (functional units, buses, registers, etc.) is available. Further, the signal statistics at component inputs can be captured by simulating the design with typical input sequences (profiling). This allows us to use an activity-sensitive model for effective capacitance instead of the white-noise one. We use the DBT activity model from the SPA [65] in our architecture-level estimations.

6.4.5. Clock models

The bus models presented in the previous section were modified to calculate the clock power. The length of the clock wire consists of two parts as shown in Figure 6.3 — the length of the routing to the border of the various datapaths and the length of the clock routing inside the datapaths. On a set of example designs, the length of the clock routing to the datapaths was found to vary anywhere between one to three times the square root of the chip area. Within each datapath, the clock wire-length was approximately the height of the datapath. Based on these observations, the total length of the clock wiring is estimated by the following formula:

$$length = 2 \times \sqrt{ChipArea} + \sum_j Height_j \qquad \text{(Eq. 6.10)}$$

where the summation is performed over all functional unit sets *j*. To estimate the loading on the clock, we assume that the clock was distributed to all the registers and buffers on the chip and each offers a 25 fF load to the clock (as obtained from our cell library).

6.5. Summary

This chapter has analyzed the importance of interconnect optimization and addressed some related issues. By studying the differences between manual and automated design implementations, it was seen that current day behavioral synthesis techniques produce architectures that are power-inefficient in the interconnect. It was also seen that there is a large opportunity for decreasing the interconnect power by working at the architecture level.

Some related research in interconnect optimization as well as interconnect and layout-area estimation are discussed. Detailed interconnect power models incorporating various architecture styles have been developed. These models are critical for assessing the savings from architecture synthesis schemes targeting interconnect power optimization which is the topic of the next two chapters.

7 Exploiting Spatial Locality

In the last chapter we saw the importance of reducing interconnect power in an automated design environment. This chapter presents a specific technique that addresses this issue. The technique is based on exploiting the spatial locality inherent in the algorithm while synthesizing to architecture. The chapter is organized as follows. Section 7.1 describes what exploiting spatial locality of an algorithm implies and discusses its impact on interconnect power, Section 7.2 reviews some of the related work, and Section 7.3 studies some of the main issues in more detail. Sections 7.4 and 7.5 present our approach for exploiting locality. Results and conclusions are presented in Sections 7.6 and 7.7.

7.1. The impact of exploiting spatial locality

The concept of using distributed or localized computing for reducing power has been used previously (e.g. memory and control partitioning). The main idea behind our approach is to apply this concept to interconnect power reduction by automatically synthesizing designs with localized communications. We achieve this by dividing the algorithm into *spatially-local clusters* and performing a *spatially-local assignment.* A spatially-local cluster is a group of algorithm operations that are close to each other in the flowgraph representation. A spatially-local assignment is a mapping of the algorithm operations to specific hardware units such that no operations in different clusters share the same hardware.

Partitioning the algorithm into spatially-local clusters ensures that the *majority of the data transfers take place within clusters* and relatively few occur between clusters. The spatially-local assignment restricts intra-cluster data transfers to buses that are local to a subset of the hardware (local buses); thus only inter-cluster data transfers use buses that are shared by all resources (global buses). In general, since *intra-cluster buses are localized to a part of the chip, they are shorter* than the buses in the non-spatially-local designs, while the global buses in the partitioned and non-partitioned designs may be comparable in length. Also, since *the local buses are connected to only a subset of the functional units, they have a lower capacitive* load on them, while the global buses are connected to more functional units and have a higher capacitive load. The combined result is that the shorter, lightly-loaded, local buses are used more frequently than the longer, heavily-loaded, highly-capacitive, global buses. Further, buffer power is reduced since smaller buffers are required to drive shorter wires. The reduced hardware sharing also results in additional power savings due to fewer accesses to multiplexors. The partitioning information is passed to the architecture-netlist generation and floorplanning tools which place the hardware units of each spatially-local cluster close together in the final layout.

Consider the example of Figure 7.1 which shows two alternative mappings of a single flowgraph to a hardware configuration consisting of two adders. The two adders are distinguished by their shadings. In Figure 7.1a, all operations of a tightly-connected group are mapped to the same hardware (for example, *a, b, e,* and *f* are all mapped to adder A_1). This does not hold in the assignment of Figure 7.1b. Considering data transfers in which a given adder outputs data to itself and to its own inputs as *local,* and those in which it outputs data to the other adder as *global,* we find that assignments of Figure 7.1a and 7.1b have 1 and 9 global data transfers, respectively (excluding input and output connections). Since global buses are long and highly capacitive compared to local ones, and also have higher capacitive load (they are connected to

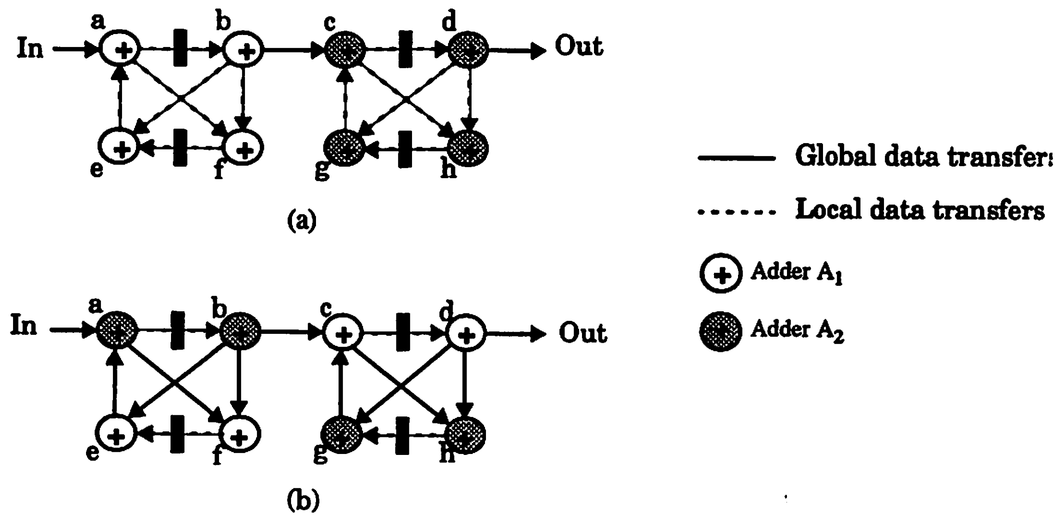


Figure 7.1. Example of spatially local and non-local assignments of a given graph: (a) local assignment, (b) non-local assignment.

two adders compared to one for the local buses), reducing accesses to global buses reduces the power dissipation.

As another example, consider a fourth-order parallel-form IIR filter. Local and non-local assignments of operations to hardware units are shown in Figures 7.2a and 7.2b, respectively (A_i are adders and M_i are multipliers). The non-local assignment is performed with the goal of minimizing the use of functional units. In Figure 7.2a, the filter is partitioned into two spatially-local clusters and the operations of each cluster are mapped to mutually exclusive hardware units (A_1, A_2, and M_1 are used for operations in cluster I and A_3, A_4, and M_2 are used for those in cluster II). As a result, all communications within cluster I take place only between hardware units A_1, A_2, and M_1 and those within cluster II take place between units A_3, A_4, and M_2. There are only two data transfers between the clusters which are global to the entire chip. In Figure 7.2b, on the other hand, operations are assigned to hardware units without

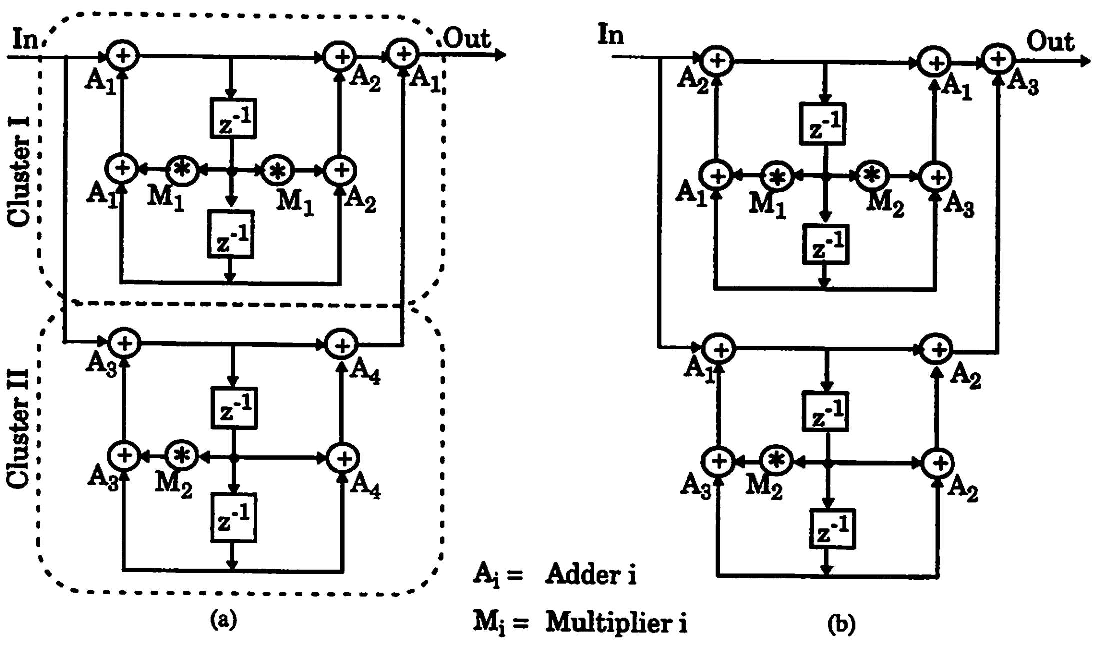


Figure 7.2. Different assignments of a fourth-order parallel-form IIR filter: (a) local assignment, (b) non-local assignment.

regard to their closeness. In this case, none of the communications are localized to a subset of hardware units and take place on global buses between all five units.

Several factors may affect the quality of designs partitioned in this way. Firstly, the local assignment may come at the cost of extra hardware. In the above example, the local version needs 4 adders and 2 multipliers whereas the non-local assignment requires just 3 adders and 2 multipliers. However, this increase in the number of functional units does not necessarily translate into a corresponding increase in the overall area since the localization of interconnect makes the design more conducive to compact layout. Secondly, reduced hardware sharing results in additional power savings in multiplexors and buffers. Thirdly, varying the number of clusters trades off local and global bus power. In particular, as the number of clusters is increased, the number of inter-cluster communications increases but the local bus lengths decrease. In the following sections we review some of the related work in partitioning, study the above issues in greater detail and present a partitioning methodology that addresses them.

7.2. Partitioning — background

The key task in exploiting algorithm locality is partitioning the algorithm into spatially local clusters of computation. In this section we present previous work in partitioning (Section 7.2.1), and review the main concepts behind spectral partitioning (Section 7.2.2) which we use in our synthesis scheme.

7.2.1. Previous work in partitioning

Previous works in partitioning for high-level synthesis have targeted area minimization, with a significant portion of the gains resulting from interconnect reduction. In the BUD system [77], pairs of nodes are repeatedly merged based on three criteria: number of common connections, possibility of executing them at the same time, and possibility of executing them on the same hardware type. In the APARTY system [61], clustering is performed in multiple stages: each stage clusters nodes based on a particular criterion such as reducing the number of control transfers, increasing hardware sharing, and decreasing data transfers.

In partitioning for low power, the goal is to reduce total chip power by reducing interconnect power. One way in which this is done is by maximizing the number of accesses to short local buses relative to long global ones. Note that in area minimization, the number of global buses is minimized. In power minimization, however, it is better to have two global buses each accessed twice each rather than one bus accessed six times.

Assuming that global buses are only used for data transfers between partitions, the number of accesses to global buses is equal to the total number of edges cut by a proposed partition, called the cut-size. Therefore, in partitioning for low-power, the goal is to minimize the cut-size. Since, in high-level synthesis, the cuts do not translate exactly into the number of buses required due to hardware sharing between units and bus sharing between data transfers, high-level area minimization schemes in do not aim at reducing the cut-size. Partitioning techniques at lower (logic, layout, and circuit) levels, on the other hand, target cut-size minimization. Thus partitioning techniques used in lower-level CAD are more useful for our purposes.

A variety of techniques have been used for partitioning at the logic, circuit, and layout levels. These include iterative improvement methods such as Kernighan and Lin [54], Fiduccia and Matheyses [31], and simulated annealing [57]; bottom-up aggregative algorithms such as [130]; top-down recursive bi-partitioning [28, 103]; and spectral partitioning techniques [3, 7, 16, 32, 42, 46, 113, 134].

We have developed a new behavioral-level partitioning method for low-power. The basic idea is to derive an ordering of the nodes by using the spectral properties of the graph and then heuristically partition this ordering. The theoretical results that form the basis of this technique are presented in the next section.

7.2.2. Key ideas in spectral partitioning

Spectral partitioning methods use eigenvectors of the Laplacian of the graph to extract a one-dimensional placement of graph nodes which minimizes the sum of squares of edge lengths. This placement is then heuristically partitioned. The key result that forms the basis of this technique was presented by Hall [43], and is given below.

Problem statement: find a one dimensional placement $x = (x_1, x_2, \ldots\ldots, x_n)$ of the nodes of a given weighted-edge graph that minimizes the weighted sum of squares of the edge lengths.

Solution: Let A be the weighted adjacency matrix of the graph, where A_{ij} is the weight of the edge between nodes i and j; $A_{ij} = 0$ if there is no edge between i and j. The cost function, z, that needs to be minimized is given by:

$$z = \frac{1}{2}\sum_{i}^{n}\sum_{j}^{n}(x_i - x_j)^2 A_{ij} \quad \text{(Eq. 7.1)}$$

The following constraint is used to normalize the placement between -1 and 1:

$$|x| = \sqrt{x^T x} = 1 \quad \text{(Eq. 7.2)}$$

Define a degree matrix, D, as the diagonal matrix in which each diagonal element is the sum of the weights of all the edges connecting to the corresponding node. The cost function z can be rewritten as $x^T(D - A)x$. The matrix $Q = (D - A)$ is called the Laplacian of the graph. The constrained cost function is given by the Lagrangian L, as:

$$L = x^T Q x - \lambda\left(x^T x - 1\right) \quad \text{(Eq. 7.3)}$$

Setting the derivative of the Lagrangian, L, to zero gives Equation 7.4.

$$(Q - \lambda I)x = 0 \quad \text{(Eq. 7.4)}$$

The solutions to Equation 7.4 are those where λ is the eigenvalue and x is the corresponding eigenvector. The smallest eigenvalue, *0*, gives a trivial solution with all nodes at the same point. The eigenvector corresponding to the second smallest eigenvalue minimizes the cost function while giving a non-trivial solution. ■

7.3. Effect of varying levels of partitioning

In the previous section the parallel-form IIR filter was partitioned into two clusters. In general a design may be partitioned into any number of clusters. In this section we study the effect of varying the number of clusters on three designs — a direct-form IIR filter, an 8-point DCT, and a fifth-order wave digital filter. For each, the power dissipation of the individual components of the interconnection network, the combined bus and clock, and the total chip are shown for different numbers of clusters (Figures 7.3-7.5). The non-partitioned case is considered as a single cluster implementation.

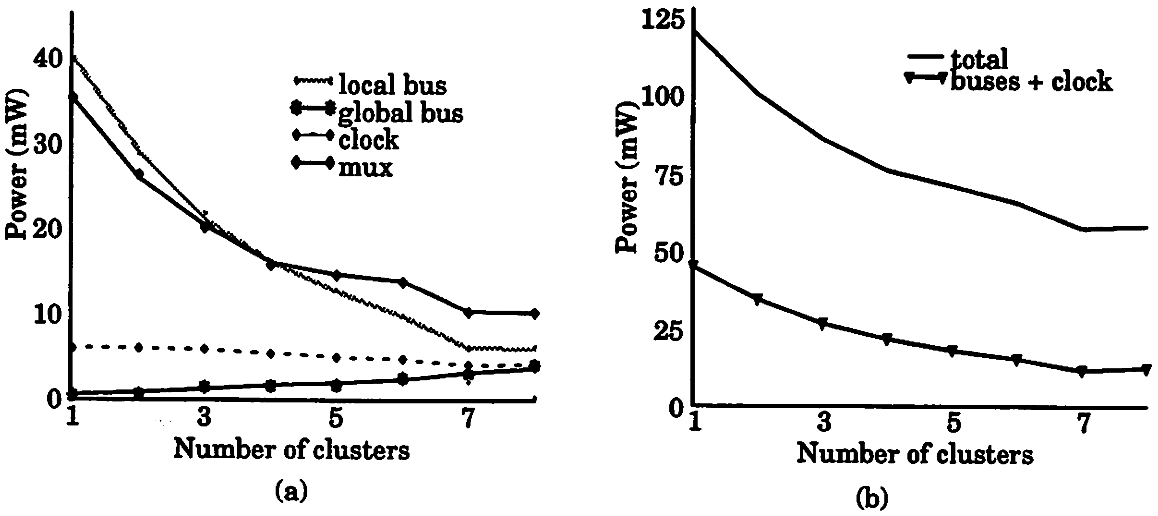


Figure 7.3. Effect of varying number of clusters on the power dissipation of the various components: direct form IIR filter.

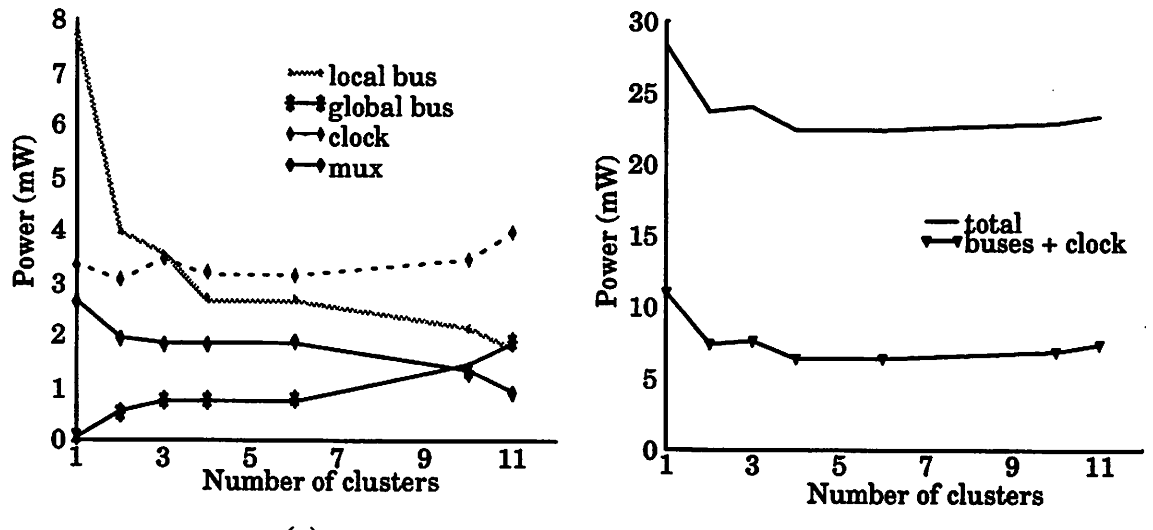


Figure 7.4. Effect of varying number of clusters on the power dissipation of the various components: discrete cosine transform (DCT).

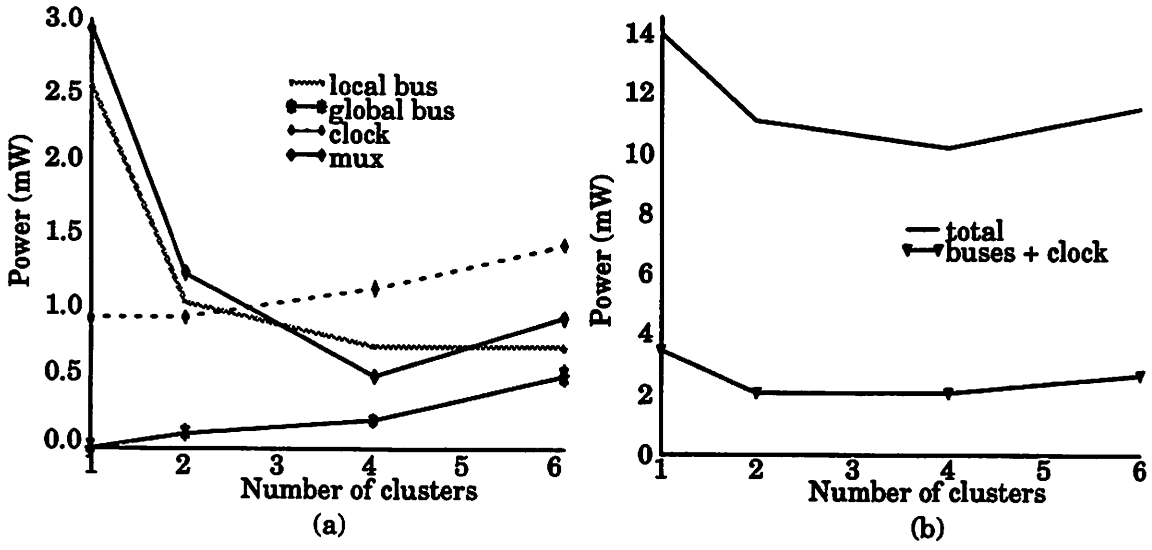


Figure 7.5. Effect of varying number of clusters on the power dissipation of the various components: wave digital filter.

With increasing number of clusters, several trends are seen. As expected, the local bus power reduces and the global bus power increases. This is because the lengths of the intra-cluster buses reduce as clusters get smaller, and the lengths of global buses increase as chip area grows. Furthermore, the number of accesses to local buses decrease while accesses to global buses increase. Another distinct trend is seen in the multiplexor power which reduces drastically due to increasingly restricted hardware sharing. The clock power remains constant or increases due to an increased number of units and therefore longer clock wiring. Although not shown, the register power reduced slightly while the power dissipation in the functional units and buffers remained the same. The reduction in register power can be explained as follows. With more clusters, the number of functional units is increased and the number of variables to be stored in the register files associated with each unit is reduced. While the total number of register accesses remain the same (determined by the number of reads and writes required). The reduced register file sizes results in lower capacitance switched per access and thus reduced register power.

The total power reduces drastically as we go from the non-partitioned design (1 cluster) to the partitioned designs. After a certain number of clusters, however, the total power starts to increase. Each example has an "optimum" number of clusters — 7, 4, and 4, for direct-form IIR, DCT, and wave digital filter, respectively. Notice that, in Figures 7.3-7.5, the combined bus and clock power tracks the total power dissipation showing an optimum at the same level of partitioning. In our partitioning methodology, we use an estimate of this value to decide the optimal number of clusters. Our partitioning methodology is explained in the next section.

7.4. Overall low-power partitioning methodology

This section describes the partitioning methodology targeted at interconnect localization for low power. First the algorithm representation and the hyperedge model are presented, then the overall methodology and the different phases are explained in detail.

7.4.1. Algorithm representation

The input algorithm is represented internally as a data-flow graph. The nodes represent operations and the edges represent data dependencies. Strictly speaking, the edges are "hyperedges" since a node may have several fan-outs. Conditionals are implemented in the datapath — all branches are executed and the conditional test is used to select the appropriate result. The representation can be hierarchical, that is, a node may itself be a graph having nodes and edges.

While connections are represented in the algorithm as hyperedges, the partitioning technique requires a representation in terms of edges in the strict sense (an edge is a connection between only two nodes). Several models to replace the hyperedge by edges were examined. All of them replace the hyperedge by edges between pairs of nodes to form a clique but differ in the weight assigned to the resulting edges.

The hyperedge problem is similar to the one discussed in layout partitioning where a net may connect several pins. In that case, the uniformly weighted clique model [67] has been widely used. This model assigns a weight of $1/(k-1)$ to all edges in the clique, where k is the number of nodes in the clique (Figures 7.6a and 7.6b). If a hyperedge is cut, its contribution to the weight of the cut is exactly one irrespective of the distribution of nodes across the net. At the layout level, the sum of weights of the edges cut by a partition under this model, one, exactly corresponds to the number of nets cut.

At the high level, however, due to hardware and bus sharing, the hyperedge does not correspond to a single bus or data transfer. Consider the hyperedge shown in Figure 7.6a. If a cut removes node z from the rest of the clique, exactly one global data transfer will be required. However, if the cut isolates node a from the rest of the clique, the number of data transfers between clusters will be anywhere from 1 to 3 depending on how the data transfers are assigned to buses. Therefore, a model that weights the edges between nodes x, y, and z less than the edges connecting a to the nodes x, y, or z may result in better solutions. We propose two new models that do this. In the first model, we assign a lower weight $1/2(k-1)$ on edges between the destination nodes leaving the weights on the other edges unchanged. In the second model, we increase the weight of the edges that join the source to the destination nodes to $2/(k-1)$ while the weights on edges between the destination nodes is unchanged. These two models are shown in Figures 7.6 c and d, respectively. Our experiments showed that different models give the best result in different situations. In our clustering methodology, we try all three hyperedge models and select the best one.

7.4.2. Overall partitioning flow

The goal of the partitioning methodology is to generate a single promising partition for the purposes of low-power synthesis. An overview of the partitioning methodology is shown in Figure 7.7. The eigenvector of the second smallest eigenvalue of the

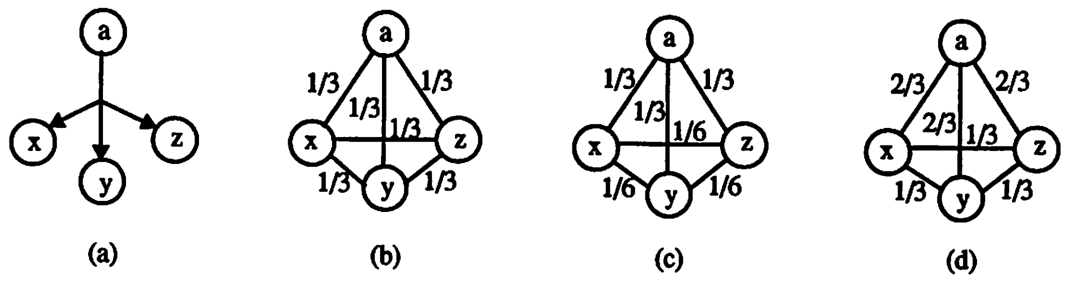


Figure 7.6. Hyperedge models: (a) initial hyperedge, (b) uniformly weighted clique model, (c) clique model with lower weights on edges between destination nodes, (d) clique model with higher weights for edges connected to the source node.

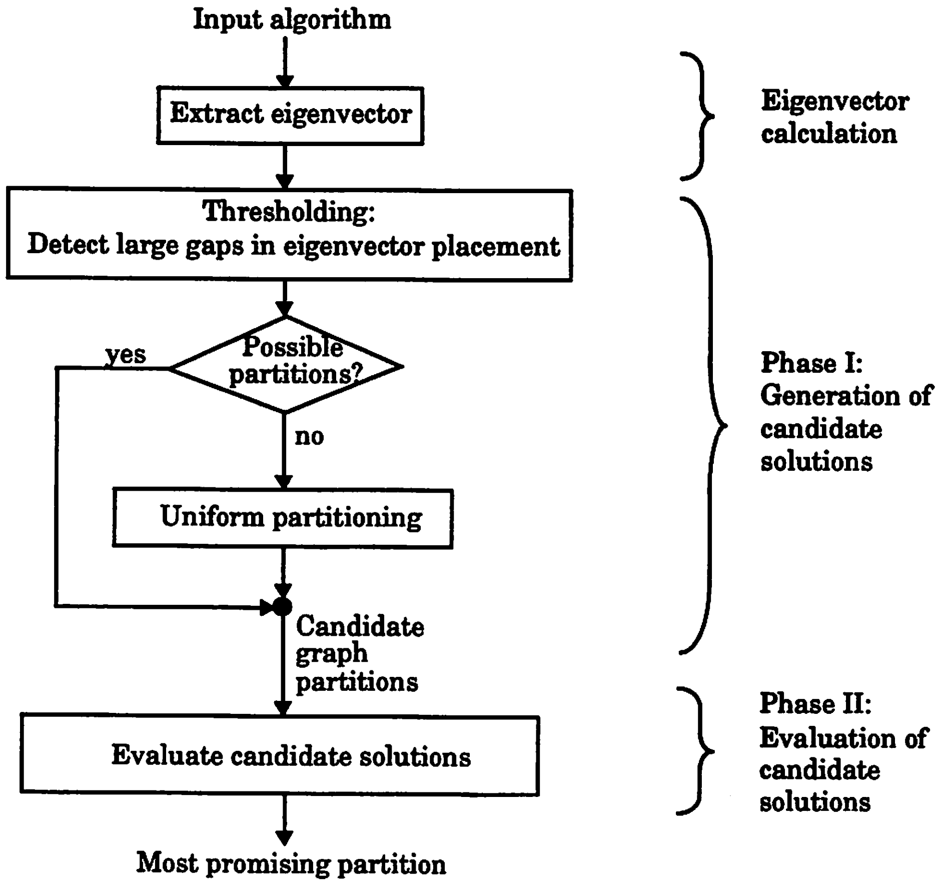
graph's Laplacian is extracted and partitioned. The partitioning is done in two phases. In phase I, several candidate graph partitions are generated. In phase II, these partitions are evaluated and the most promising one is selected.

7.4.3. Phase I: finding good partitions

The goal of this phase is to propose a few partitions that are balanced and localized. Multiple solutions are generated by using the different hyperedge models and by varying the maximum allowed number of clusters. As was seen in Section 7.3, increasing the number of clusters increases the number of global accesses and reduces the size of local buses, resulting in lower local-bus and higher global-bus power. Generating several solutions with different number of clusters explores this trade-off.

The eigenvector placement obtained as described in Section 7.2.2 forms the nucleus of the approach. It provides an *ordering* in which nodes tightly connected to each other are placed close together. Furthermore, the *relative distances* are a measure of the tightness of connections. The spectral technique is specially suited to our needs since the eigenvector is computed only once and generating several partitioning solutions from it is computationally inexpensive.

Two main techniques are used to generate partitions from this ordering. This is because graph representations of DSP algorithms can have widely varying structures



Input algorithm
Extract eigenvector
Eigenvector calculation
Thresholding: Detect large gaps in eigenvector placement
Possible partitions?
yes
no
Uniform partitioning
Candidate graph partitions
Phase I: Generation of candidate solutions
Evaluate candidate solutions
Phase II: Evaluation of candidate solutions
Most promising partition

Figure 7.7. Overview of the partitioning methodology.

in terms of spatial locality. Consider the two examples shown in Figure 7.8. In Figure 7.8a there two distinct clusters which are also clearly visible in the corresponding eigenvector placement. However, not all algorithms have a clearly clustered structure. A non-clustered algorithm may still be very partitionable in that it may be possible to partition it so that few edges are cut relative to the total number of edges. For example, the graph shown in Figure 7.8b is not clustered but is partitionable since only three edges are cut when it is divided into two equal parts. Notice that the eigenvector uniformly spaces the nodes in this case. Another good example is an FIR filter

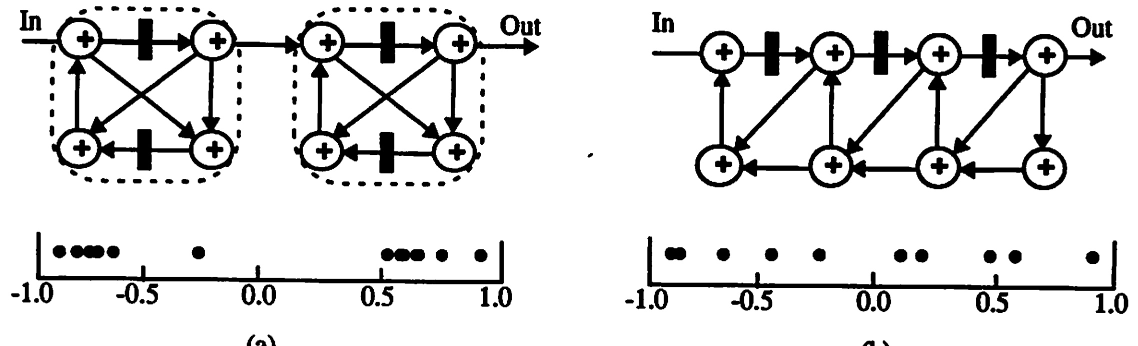


Figure 7.8. Eigenvectors for sample examples: (a) example with natural clusters, (b) partitionable example with no natural clusters.

structure which has no distinct groups of clustered operations but can be easily partitioned.

The aim of the partitioning methodology is to detect natural clusters in the clustered algorithms and find good and balanced partitions in uniform, non-clustered cases. The eigenvector placement provides a good starting point to address both cases. *Large gaps* in the placement can be used as partition points in the first case and the *order* of nodes can be used for partitioning in the second.

We first try to detect natural clusters inherent in the algorithm. Large gaps in the eigenvector placement are used to indicate good points for partitioning. Since the nodes are always placed between -1 and 1 due to the constraint $\sum_{i=1}^{n} x_i^2 = 1$ imposed by Equation 7.2, the absolute values of distances between adjacent nodes vary depending on the total number of nodes in each example. By a "large gap" we mean the distance between two adjacent nodes in the placement that is large relative to distances between other adjacent nodes in the same example. The threshold for detecting these gaps is therefore relative to the distances in the same example. Several different thresholds for identifying large gaps — $m+\sigma$, $m+2\sigma$, and $m+3\sigma$ — where m is the mean of the distances between adjacent nodes and σ is their standard deviation, were evaluated. Our experiments showed that although the $m+3\sigma$ threshold found most of the clusters, some clusters were only detected using $m+2\sigma$. In our methodology, we therefore try the $m+3\sigma$ threshold first. If no clusters are detected the threshold is reduced to $m+2\sigma$. Smaller thresholds may be tried for a more exhaustive exploration of the design space.

Solutions targeting up to 2, 3, 4, and 8 clusters are generated. This is done by varying the constraint on the number of nodes allowed in each cluster. If a maximum of x clusters is targeted, the smallest cluster size is fixed to be $\left\lceil\left(\frac{n}{(x+1)}\right)+1\right\rceil$ nodes, where n is the total number of nodes in the graph.

Partition points are inserted in the 1-dimensional placement to mark large gaps based on the $m+3\sigma$ or $m+2\sigma$ metric. These points mark boundaries between different groups of nodes. If a group detected has less nodes than allowed by the size constraint, it is merged with one of the neighboring groups. The decision regarding which neighboring group to merge with, is based on the size of the gap between the groups (as calculated from the eigenvector). The gap between two adjacent groups in the eigenvector placement is the distance from the right-most node of the left group to the left-most node of the right group. Clusters are thus identified using *thresholding and subsequent merging*.

The main goal of the thresholding is to detect natural clusters inherent in the algorithm. As mentioned before, not all algorithms have natural clusters. Therefore, if no clusters are found by thresholding, a second technique is applied in which the eigenvector placement is used simply as an ordering of the nodes. This placement is *uniformly partitioned* into the targeted number of clusters. Again solutions targeting 2, 3, 4, and 8 clusters are generated. Depending on the quality of solutions required and the time that the user wants to spend, a greater number of partitioning solutions may be tried.

7.4.4. Phase II: evaluating partitions

The generation of candidate partitions is based on minimizing the number of global bus accesses. The underlying assumption is that intracluster buses in the partitioned implementation are significantly shorter than the buses in the original non-partitioned one. This assumption may not hold for designs in which the area of any one cluster is too large. The first goal of the evaluation phase is to prune out unpromising partitions based on area estimates. Further, the studies in Section 7.3 demonstrated that as the number of clusters is increased, the local bus power reduces while the global bus power grows. Also, the different hyperedge models produce solutions of varying quality. The second goal of the evaluation phase is to explore these trade-offs. Based on these two goals, this phase first prunes out unpromising partitions and then compares the effectiveness of remaining ones based on an estimate of the bus power.

In order to prune out solutions with extremely large area overhead, we estimate the area of the unpartitioned graph and each of the partitioned solutions based on distribution graphs [91]. A distribution graph displays the expected number of operations executed in each time slot. Figure 7.9a shows a simple algorithm and the corresponding distribution graph. For an algorithm with different types of operations, the total weighted distribution graph is obtained by summing up the distribution graphs of each operation type weighted by the area of the corresponding hardware. We use the *maximum height of the total weighted distribution graph* as an estimate of the area. For clustered designs, distribution graphs are constructed for each cluster and the sum of the area estimates over all clusters is used for the total area. These estimates are used to prune out solutions with more than 100% area overhead. The allowed percentage overhead can also be modified by the user.

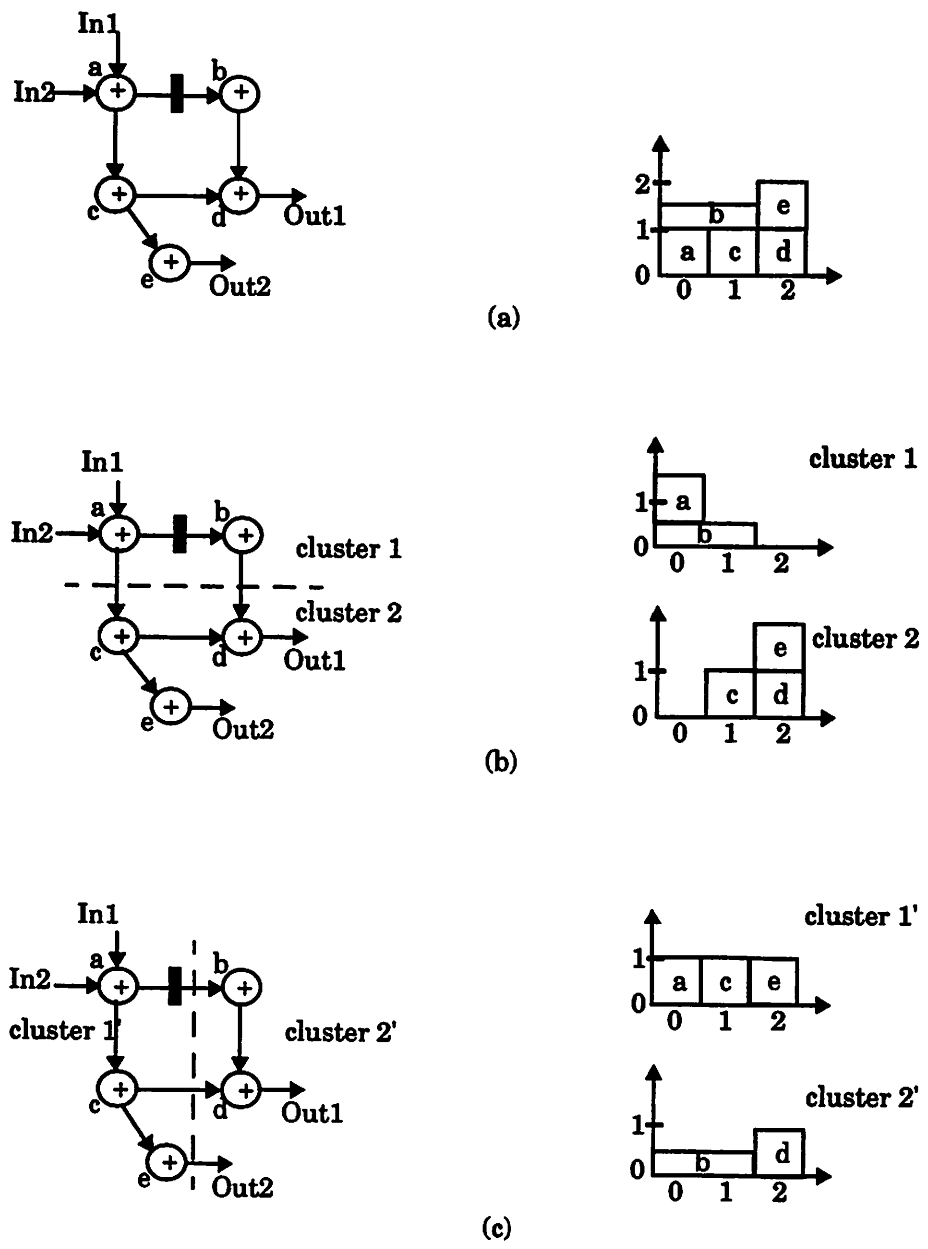


Figure 7.9. Distribution graphs for different candidate partitions of a given algorithm: (a) unpartitioned, (b) candidate partition 1, (c) candidate partition 2. Each operation's contribution is labeled on the distribution graphs.

The remaining candidate solutions are compared based on the total bus power estimated using the following heuristic measure:

$$\sum_{i=1}^{n}\left(A_i^{local}\cdot N_i^{local}\right)+A^{global}\cdot N^{global} \qquad \text{(Eq. 7.5)}$$

where A_i^{local} is the estimated area of the cluster i, N_i^{local} is the number of edges (each fan-out of a hyperedge is counted as one edge) local to the cluster, A^{global} is the overall area of the chip, N^{global} is the number of global edges, and the summation in the first term is done over all clusters. Each term inside the summation can be thought of as a measure of the local bus power in the corresponding cluster. The area of the cluster, A_i^{local}, is a measure of the lengths of the local buses in it (refer to the bus length model in Section 4.3.3) and therefore their physical capacitance, and the number of local edges, N_i^{local}, is a measure of the accesses to them (refer the bus access model in Section 4.3.3) and therefore their activity. Thus, each term in the summation represents the total capacitance switched in local buses in the cluster per sample period and is proportional to the corresponding power consumption at a given supply voltage and sampling frequency. By a similar reasoning, the last term is a measure of the power consumption in global buses, where the overall area, A^{global}, is indicative of the global bus lengths and the number of global edges, N^{global}, is a measure of the accesses to them. The cost function given by Equation 7.5 is thus indicative of the total bus power associated with a particular partition. Recall from Section 7.3 that the bus power tracks the overall power dissipation as the number of clusters is varied. Therefore, the above heuristic provides a good measure for comparing various partitioning solutions.

As mentioned before, we use the height of the distribution graph as an area metric. In the example of Figure 7.9, we see that though the number of edges cut by the partition in Figures 7.9b and 7.9c is the same (2 edges), the area penalty is higher in the first case. As a result the total number of accesses to global buses is the same but the local buses are longer and more capacitive in the first case. In this case the cost function of Equation 7.5 identifies the second partition as being more optimal.

7.5. Partitioning based synthesis

This section explains our locality-based synthesis methodology and gives an overview of the synthesis flow.

7.5.1. Design flow from algorithm to layout

Our techniques have been integrated into the Synergy[1] high-level synthesis system. The basic synthesis flow of the Synergy system is the same as that of the Hyper system (refer Chapter 3), except that a new partitioning step is added preceding the other synthesis steps and the assignment algorithms are modified to exploit spatial locality. The new synthesis flow is shown in Figure 7.10.

The core of the new system is the partitioning methodology described in Section 7.4. Once the partitioning is complete, all operations have an associated cluster number and each data transfer is classified as either global or local. The assignment technique is based on the random initial assignment with iterative improvement approach of the Hyper system with the added constraint that there is no hardware sharing between clusters. The scheduling algorithm is unchanged.

The concept of "badness" in the Hyper allocation scheme (refer Section 3.1.5) is extended to define a badness measure for each cluster. For a given type of unit and cluster, the "cluster-badness" is defined as the sum of the badnesses of all nodes of

1. The Synergy synthesis system incorporates the estimation and exploration environment explained in Chapters 4 and 5 and the two synthesis approaches developed in this and the next chapter.

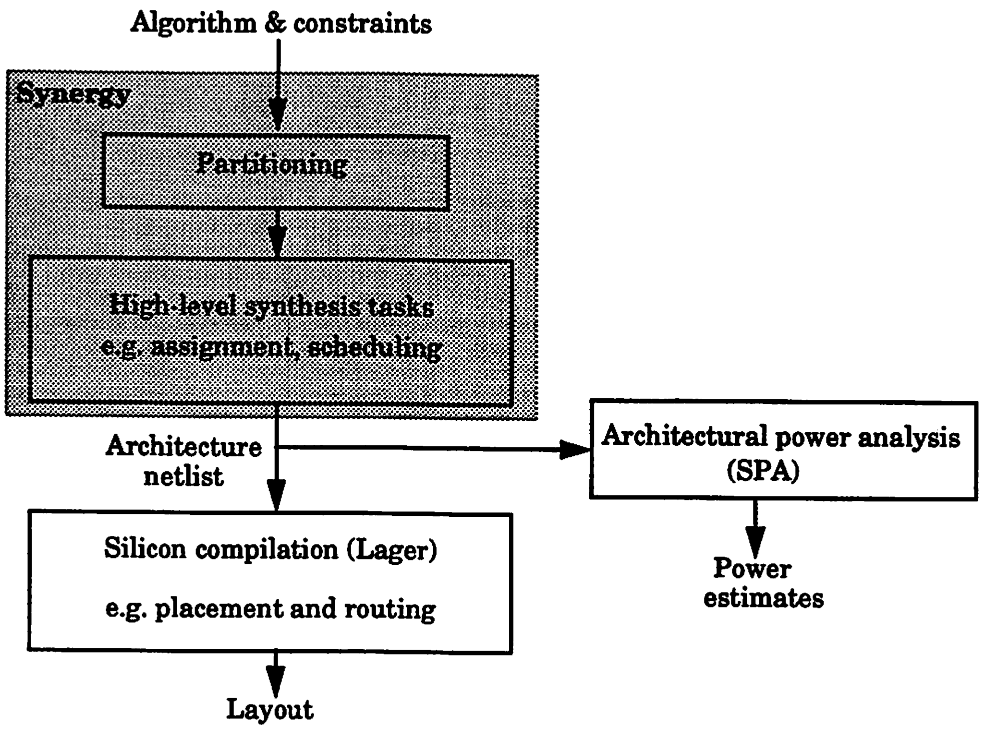

Algorithm & constraints
Synergy
Partitioning
High-level synthesis tasks
e.g. assignment, scheduling
Architecture netlist
Architectural power analysis (SPA)
Power estimates
Silicon compilation (Lager)
e.g. placement and routing
Layout

Figure 7.10. Synthesis steps from algorithm to layout.

that type in the cluster. When the scheduling/assignment process fails, new units are allocated based on the badness measure. When a new unit is allocated, it is allotted to the cluster with the highest cluster-badness. Similarly, the cluster with the lowest cluster-badness is used when resources are being removed.

Bus assignment is also modified to account for the partitioning. A given pair of data transfers are merged onto the same bus only if they are either both global transfers, or both local transfers in the same cluster. In the construction of the conflict graph for bus assignment, extra conflict edges are added to represent these partitioning constraints in addition to edges due to timing conflicts.

The architecture netlist is further compiled to layout. Silicon compilation performs a number of tasks such as tiling, placement, and routing to generate the final layout.

The Lager silicon compiler [12], is used for this purpose. The partitioning information is passed to the floorplanning tools which place hardware units of a given partition close together in the final layout. As much as possible, all units in the same cluster are placed in the same datapath. The output from Lager is a physical layout of the processor core.

7.6. Results

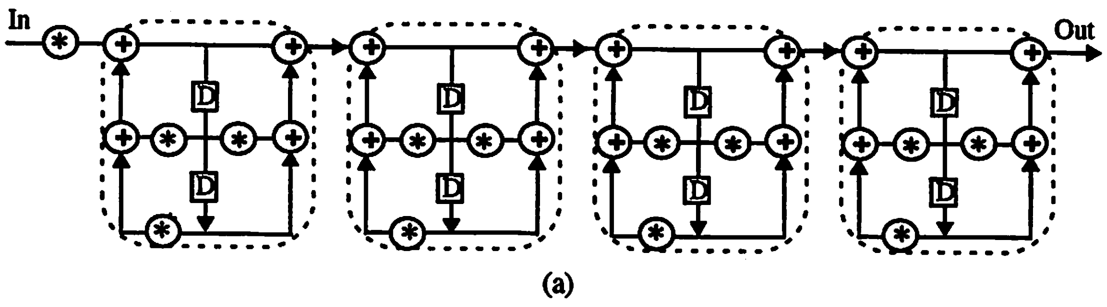
In this section we present the results of our partitioning-based synthesis scheme. Implementations generated using the Synergy and Hyper systems are compared using power estimates obtained from SPA [65]. Bus and clock power estimates are obtained using models presented in Section 6.4.2.

7.6.1. Cascade filter

The first result compares Synergy and Hyper implementations of an eighth-order cascade filter. Figure 7.11 shows the eighth-order cascade IIR filter and the corresponding eigenvector placement. The spacings between the points in the placement clearly indicate the four clusters that are evident in the structure. Using $m+3\sigma$ as the threshold to decide the points of partition, we obtain clusters delimited by the arrows. It is interesting to note that $m+2\sigma$ also works as a good threshold for this example.

Since all the multiplications in the design are multiplies with constant factors, they are converted into shift and add operations to avoid the use of area-intensive multipliers. The critical path of the resulting graph is 19 clock cycles, with both shifters and adders taking one clock cycle to execute. Given a throughput constraint of 21 clock cycles, the Hyper implementation uses four adders and three shifters while the Synergy implementation uses one adder and one shifter for each cluster resulting in a total of eight units.

Magic layouts of the two implementations (Figure 7.12) were obtained using the Lager silicon compiler and the Flint placement and routing tool [12]. In the Hyper



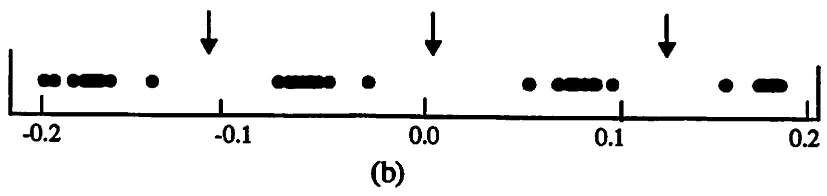
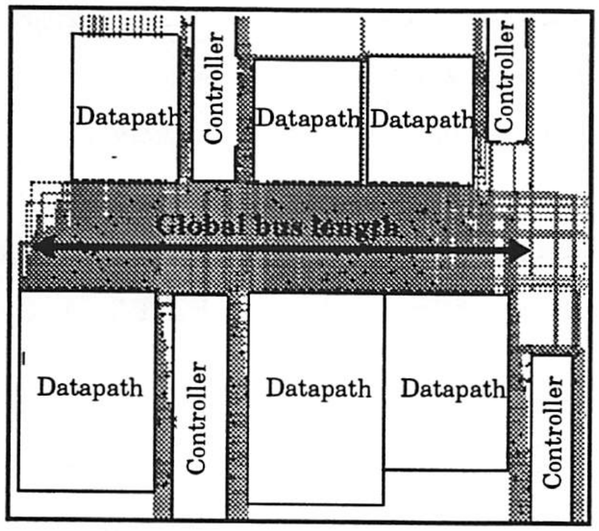


Figure 7.11. A eighth-order cascade-form IIR filter: (a) the structure, (b) corresponding eigenvector.

implementation, 2 of the 7 functional units are merged by the floorplanning tool. In the Synergy implementation the 2 units in each cluster are merged into a single datapath and the 4 datapaths in the layout correspond to the 4 clusters.

The partitioning localizes the computation enabling more compact layouts to be obtained. In the Hyper implementation, there are seven functional units that communicate heavily with each other (the layout tool has merged two units into the same datapath resulting in 6 datapaths). The Synergy implementation has eight units divided into four clusters, with heavy communication within each cluster and few connections between the clusters.

Table 7.1 compares the power dissipated in the two implementations. An overall reduction of 35% in the power consumption was realized by the Synergy approach. As opposed to 106 accesses to global buses in the Hyper implementation, the Synergy version has 95 accesses to local buses which are short (0.27 mm) and only 3 accesses to long global buses (1.48 mm). As a result, the bus power reduced 3-fold, from 2.9 mW



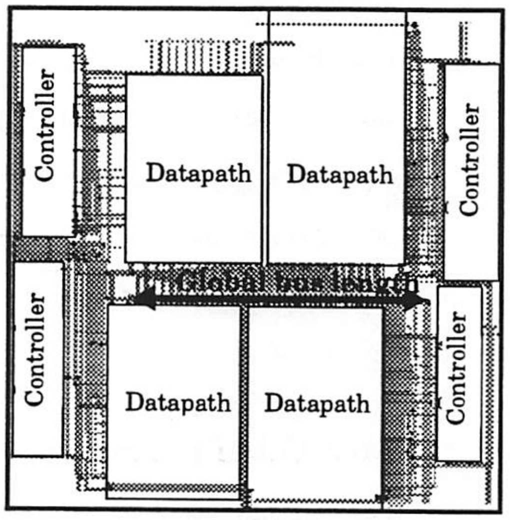


Figure 7.12. Cascade filter layouts: (a) non-local implementation from Hyper, (b) local implementation from Synergy.

Table 7.1. Comparison of power consumed in the Hyper and Synergy implementations of the cascade filter.

Component	Hyper	Synergy	Percentage reduction
Buses	2.9	1.0	65.5
Multiplexors	3.0	0.9	70.0
Clock	1.0	0.8	20.0
Buffers	0.9	0.8	11.1
Functional units	2.6	2.4	7.7
Registers	4.9	4.0	18.4
Total	15.2	9.8	35.5

to only 1.0 mW. The multiplexor power also reduced 3-fold as the reduced time-sharing of units resulted in lower usage of multiplexors. Note that the contribution of the interconnect to the total power dissipation was reduced from 44% to 27%.

For this example functional unit power also reduced. This is because signals within spatially local clusters tend to be more highly correlated than widely separated signals. Since spatially local assignment allows hardware sharing only between operations in the same cluster, the resulting implementation is likely to have more correlated inputs to each of the blocks. This results in lower activity and hence less power consumption in the functional units. Buffer and register power also decreased due to factors such as improved data correlations, reduced physical capacitance, and fewer accesses.

7.6.2. Other examples

This section summarizes our experimental results for the cascade and several other DSP filter and transform examples. Some are in their original form (DCT, FFT, and parallel-form IIR) and others are transformed using either constant multiplication expansion (cascade-form IIR, direct-form IIR, and wavelet) or retiming (wave digital filter).

Table 7.2 shows the number of accesses to buses and multiplexors, and the estimated bus lengths for both implementations of each example. The accesses to global buses is reduced drastically for all examples with very little change in the lengths of these buses. Exploiting spatial locality moves a large percentage of the bus accesses from the long global buses to intra-cluster buses whose lengths are 80% shorter on average than those of the global buses. In general, due to reduced hardware sharing, there is a decrease in the multiplexor accesses for all examples. The total number of buffer accesses (including tristated and non-tristated buffers) is unchanged because a buffer is used to drive every data transfer on a bus.

Table 7.3 shows the bus, multiplexor, and the overall power dissipation for both implementations of each example. The Synergy implementations uniformly dissipate less power than the Hyper implementations. The corresponding percentage improvements are summarized in Figure 7.13. Power consumed by buses is reduced drasti-

Table 7.2. Comparison of bus lengths and bus and multiplexor accesses in the Hyper and Synergy implementations.

Name	Hyper			Synergy					
	Bus accesses	Bus length[1]	Mux accesses	Number of clusters	Global bus accesses	Local bus accesses	Global bus length[1]	Average local bus length[1]	Mux accesses
Cascade	106	1.63	279	4	3	95	1.48	0.27	109
Direct form	202	4.93	1319	7	21	183	3.92	0.44	448
Wavelet	71	2.34	247	2	3	69	2.43	1.05	171
Wave digital	57	1.29	156	4	4	53	1.48	0.20	36
DCT	59	3.52	104	4	7	51	3.49	0.93	75
FFT	38	3.48	58	4	7	33	3.89	0.52	39
Parallel IIR	40	2.57	82	4	4	36	4.29	1.12	47

1. Bus lengths are in millimeters for a 1.2 micron technology.

Table 7.3. Power consumption in the Hyper and Synergy systems.

Design	Hyper			Synergy		
	Bus	Mux	Total	Bus	Mux	Total
Cascade	2.9	3.0	15.2	1.0	0.9	9.8
Direct form	39.5	34.7	120.2	8.2	9.9	57.4
Wavelet	7.1	7.2	33.5	4.3	4.5	27.8
Wave digital	2.5	2.9	13.9	0.9	0.5	10.1
DCT	7.7	2.6	28.4	3.3	1.8	22.4
FFT	9.6	4.5	34.8	4.3	2.5	27.4
Parallel IIR	5.2	2.9	40.7	2.9	1.3	38.1

cally in all examples (up to 80%) and large reductions are also seen in the multiplexor power (more than 70% reduction in three of the examples). The average reduction in bus, multiplexor, and total power is 57.8%, 56.0%, and 25.8%, respectively.

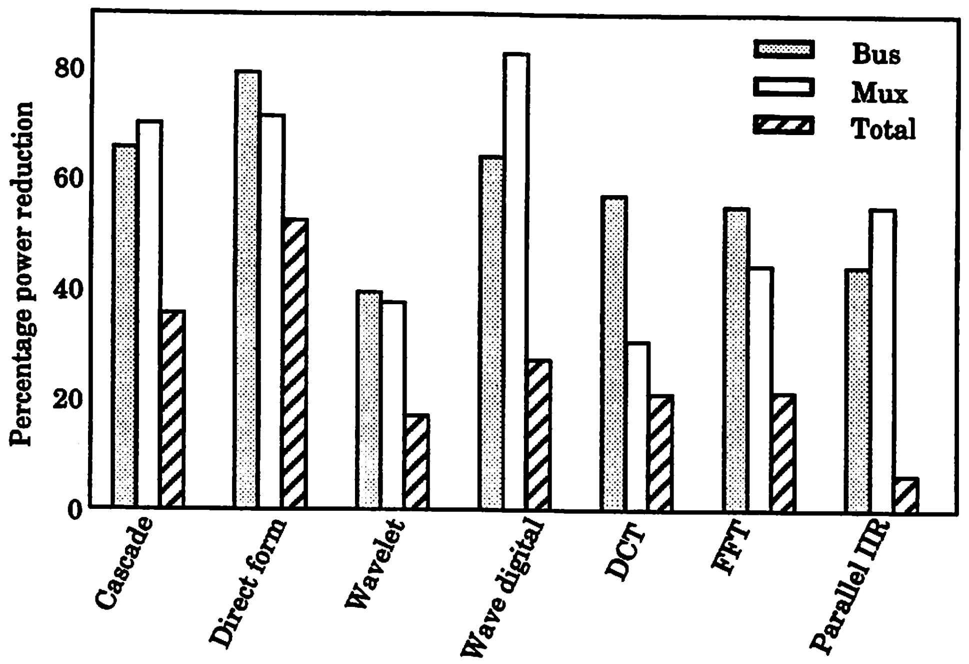


Figure 7.13. Percentage improvements in bus, multiplexor, and total power.

In partitioned implementations, we expect buffer power to decrease since smaller buffers can be used to drive the data transfers occurring on short local buses. However, our architecture-netlist generation tool currently uses fixed-sized buffers for all data transfers, regardless of bus length, and therefore, our results show negligible change in buffer power. With necessary modifications, buffer power should contribute toward further reduction in total power.

These experiments have demonstrated that restricting hardware sharing to behaviorally localized operations in the algorithm results in a large reduction in the interconnect power. However, this does not necessarily come free of cost and in several cases a penalty must be paid in terms of an increase in the number of functional units. Fortunately, an increase in the number of functional units does not necessarily translate into an equivalent increase in overall area. Since the communications across the chip are localized, the design is more conducive to compact layout. This is due to not only

Table 7.4. Comparison of the amount of hardware and the overall area required in the Hyper and Synergy implementations.

Name	Hyper					Synergy				
	Units			Estimated active area (mm^2)	Estimated total chip area (mm^2)	Units			Estimated active area (mm^2)	Estimated total chip area (mm^2)
	*	+	>>			*	+	>>		
†Cascade	-	4	3	2.32	8.79	-	4	4	1.78	7.28
Direct form	-	15	12	11.77	80.20	-	15	8	7.33	50.82
Wavelet	-	6	6	3.88	18.07	-	8	6	4.24	19.55
Wave digital	-	3	2	1.46	5.46	-	4	4	1.66	7.24
DCT	8	15	4	7.70	40.95	7	13	-	6.83	40.30
FFT	2	7	-	6.31	38.95	2	10	-	6.75	49.92
Parallel IIR	4	3	-	6.04	21.85	9	8		12.40	60.95

reduced global connections and smaller local buses but also due to fewer overhead elements such as multiplexors and buffers. As was seen for the cascade filter of Section 7.6.1, the new layout may actually be smaller than the original. Table 7.4 shows the estimated area penalty obtained in the Synergy designs. It is seen that the impact of the increase in functional units is not reflected in the total area. In most cases, therefore, the total chip area is marginally affected. For one example, the parallel form IIR filter, about 34.7% area penalty is seen. In two of the examples, the area is reduced by 47 and 9%. In fact, for the DCT example, the number of components required was also reduced! This is because partitioning the example into localized regions serves as a guidance to the assignment tool, and it is able to find a better solution. Note that the assignment tool is heuristic and therefore not guaranteed to find the global minimum.

All examples in this section have been optimized for power with no limitations on area. By varying the number of clusters, different design points with lower area penalty can be obtained at the cost of lower power savings. For example, the parallel filter implementation with 2 clusters has lower power savings (30.8%, 24.1%, and 3.7% in bus, mux, and total power, respectively) but the area penalty is much lower (30.4%) than that of the 4-cluster implementation shown in the table.

In summary, exploiting locality of algorithms during the high-level synthesis process greatly reduces interconnect power. For most examples a significant reduction in total chip power is obtained. Though the number of functional units required is increased due to restricted hardware sharing, the penalty in the total chip area is marginal.

7.7. Summary

The architecture synthesis process can have a large impact on the power dissipated in a design. In this chapter, we have presented a new technique for power reduction based on exploiting the locality in a given application. It was seen that preserving the locality improves the implementation in a variety of different ways. The predominant effect is the reduction of accesses to highly capacitive global buses. Our results showed up to 80% improvement in the power consumed in buses. Additionally, restricting hardware sharing led to reduced usage of multiplexors. Though the power savings can come at the cost of increased area, this effect is marginal. The techniques have been integrated into the Synergy system.

The concept of preserving locality is a special case of a more general class of techniques referred to as distributed computing. In general, accesses to global computing resources — controllers, buses, memory, I/O — are expensive due to increased capacitance. Dividing these resources reduces the capacitance being switched per access. This work can therefore be applied to more general applications such as memory partitioning and processor partitioning.

8 Exploiting Algorithm Regularity

The previous chapter presented a partitioning scheme for reducing interconnect power and techniques for using the partitioning information during synthesis. However the basic synthesis tasks of allocation, assignment, and scheduling were unchanged. In this chapter, we provide a new allocation, assignment, and scheduling strategy specifically aimed at reducing interconnect power. We target ASIC implementations of datapath-intensive, real-time DSP applications. The main idea behind the approach is to exploit the regularity inherent in the algorithm and derive a simplified interconnect structure. This leads to power savings in the buses, multiplexors, and buffers.

The chapter is organized as follows. In the next two sections, we define the regularity of an algorithm and elaborate on its relevance to interconnect power reduction. Section 8.3 presents related work in regularity exploitation. Section 8.4 gives an overview of our approach and Sections 8.5 and 8.6 detail our synthesis strategies. The results are presented in Section 8.7.

8.1. Algorithm regularity

Regularity in an algorithm refers to the repeated occurrence of computational patterns in it. Instances of regularity in two common DSP algorithms are presented in Figure 8.1 — Figure 8.1a shows repeating shift-add patterns in the error-prediction

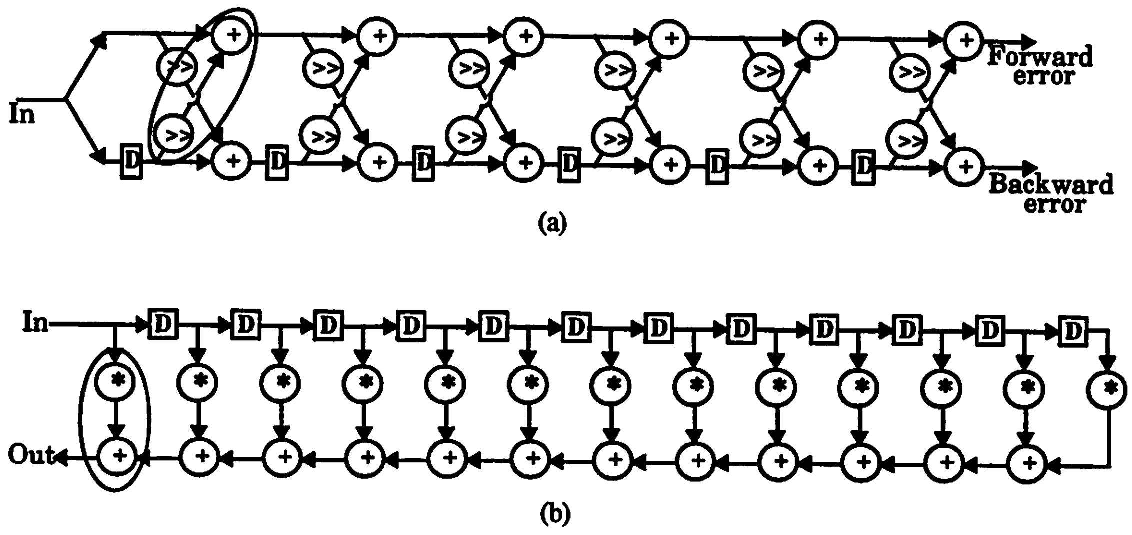


Figure 8.1. Instances of regularity: (a) error prediction filter, (b) FIR filter.

filter and Figure 8.1b shows multiply-add patterns in an FIR filter. An algorithm is said to be "more" or "less" regular depending on the degree of repetition of common patterns in it.

Recurring computational patterns may be either large-grained or fine-grained. These differ in their ease of detection by the user. Repetitions of large grain computation patterns are easily evident to the user and are usually embodied in the code as subroutines or loops. Fine-grained patterns includes smaller patterns of computations spread out through the entire program and are usually not easily evident to the user. The techniques presented here focus on automatically detecting and preserving fine-grained regularity in a given application. We rely on the user to identify the larger patterns and preserve the associated regularity during synthesis.

8.2. The impact of exploiting regularity

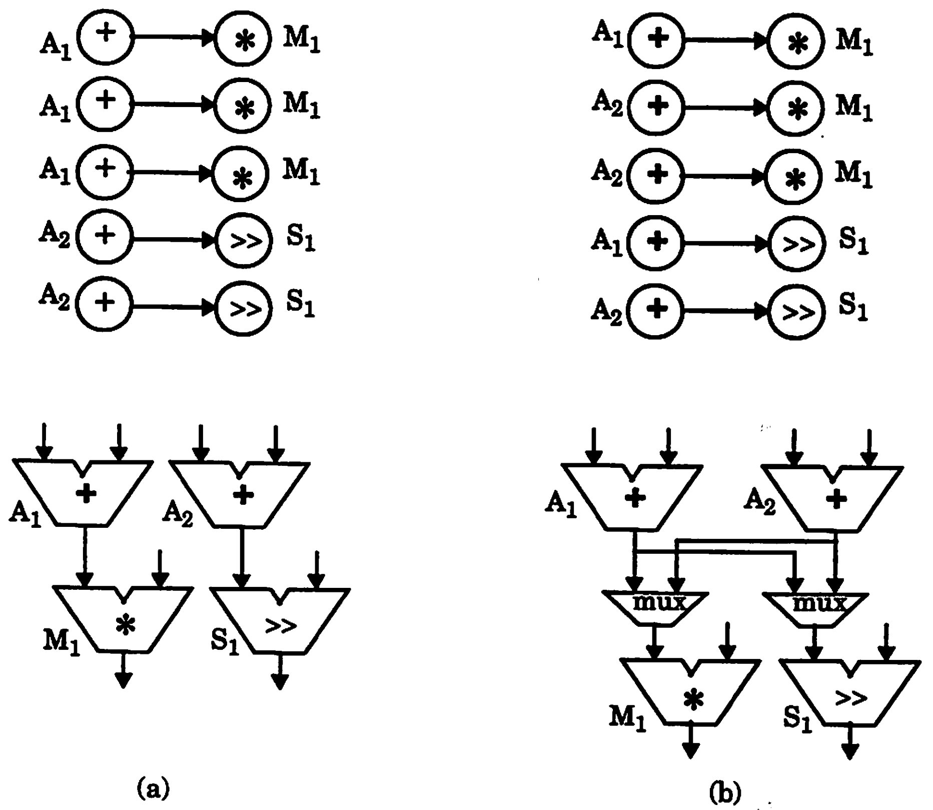


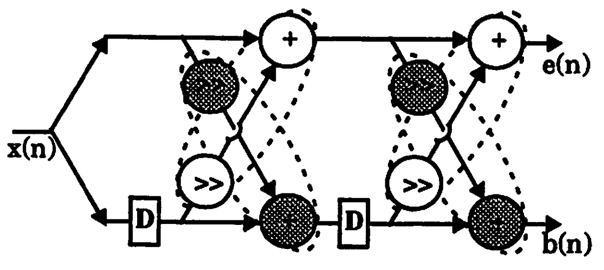
Figure 8.2. Preserving regularity leads to a simplified interconnect structure: (a) regular assignment, (b) non-regular assignment.

Exploiting regularity refers to preserving the repeated patterns of computation in the assignment of operations to hardware. This entails detecting repetitive patterns in the algorithm and mapping them such that corresponding nodes in different instances of the pattern are mapped to the same hardware unit. We refer to such an assignment as a *regular assignment*. In a regularly assigned set of pattern instances, corresponding data transfers have the same source and destination hardware units and can use the same connection without the need of extra multiplexors. As a result, the fan-outs from the output ports of the hardware units and fan-ins to their input ports are decreased and multiplexors and tristate buffers are reduced. The overall effect is a simplified interconnect structure.

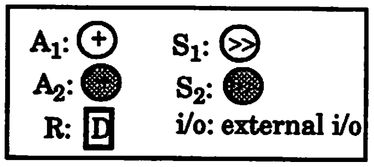
Figure 8.2 shows two different assignments for a portion of an algorithm and the corresponding hardware netlists. The first assignment is more *regular* since all instances of a given pattern (add-multiply or add-shift) are assigned to the same pair of hardware units. The assignment of Figure 8.2b, on the other hand, does not preserve regularity since additions of different add-multiply patterns are assigned to different adders. Upon examining the corresponding hardware implementations, we see that while in the first case, each adder output data only to one hardware unit and each bus has a single fan-out, in the second case, the adders outputs have two fan-outs each. Also, the second scheme requires multiplexors while the first one does not. Thus, a regular assignment leads to less fan-outs and fan-ins and lower multiplexing overhead.

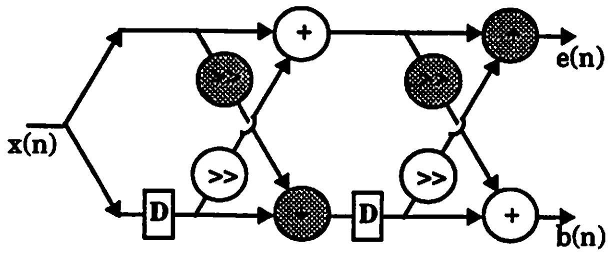
We illustrate the above ideas with the second-order error-prediction filter example shown in Figure 8.3a. Two different assignments, each using 2 adders (A_1, A_2) and 2 shifters (S_1, S2), are shown in Figures 8.3a and 8.3b. The different adders and shifters in the circuit are distinguished in the figure by their shadings. In Figure 8.3a, the shifter S_1 always outputs data to the adder A_1 and S_2 outputs to A_2 while in Figure 8.3b each shifter outputs data to both adders. The tables in the figure summarize the fan-outs and fan-ins of the output and input ports, respectively, for the four hardware units. By using a regular assignment, the total fan-outs from all output ports is reduced from 9 to 7 (Figure 8.3c). If the length of a bus is assumed to be proportional to its fan-out, the total bus-capacitance switched is reduced by a factor of 9/7 since each output bus is used twice. Further the total fan-ins to input ports of the units is also reduced (Figure 8.3d), decreasing the number of multiplexors from 5 to 2. Multiplexor accesses are reduced by a factor of 5/2 since each multiplexor is also used twice.

Power reduction in a regular implementation stems from two factors. Due to reduced fan-outs, the interconnect lines can be kept short leading to lower physical capaci-



(a)





(b)

Fan-outs

Output port	Assignment (a)	Assignment (b)
A_1.out	A_1.in2, S_2, i/o	A_2.in2, S_2.in, i/o
A_2.out	R, i/o	R, i/o
S_1.out	A1.in1	A_1.in1, A_2.in1
S_2.out	A2.in1	A2.in1, A_1.in1

(c)

Fan-ins

Input port	Assignment (a)	Assignment (b)
A_1.in1	S_1.out	S_1.out, S_2.out
A_1.in2	i/o, A_1.out	i/o, R
A_2.in1	S_2.out	S_1.out, S_2.out
A_2.in2	R	A_1.out, R
S_1.in	R	R
S_2.in	i/o, A_1.out	i/o, A_1.out

(d)

Figure 8.3. Two possible assignments and the corresponding fan-ins and fan-outs for the second-order error prediction filter: (a) more regular assignment, (b) less regular assignment, (c) fan-outs of hardware units, (d) fan-ins of hardware units.

tances. Notice that the reduced fan-out buses correspond to the data transfers in the recurring patterns and are therefore used repeatedly. This gives the desirable combination of reduced capacitance on the more active buses. Secondly, a regular implementation also has lower multiplexing overhead and dissipates less power in multiplexors.

Notice that the power improvements may be associated with increased functional units due to the lower multiplexing. Thus, a regular assignment trades-off hardware resources to obtain a simpler interconnect structure, increasing the functional unit area while saving power and area associated with the interconnect elements. The associated area costs must be carefully monitored, since the larger area may increase bus lengths and bus power.

It is interesting to note that the regularity in an algorithm can be increased by exploiting the mathematical properties of its operations. For example, operation commutativity can be used to switch the order of operands of an operation to match other computational patterns in the graph. Associativity can be used in the same way. In this thesis, we consider the structure of the graph as a given, and present techniques for exploiting the regularity inherent in it. Techniques for changing the graph structure to increase its regularity are not considered, although they would form a valuable avenue for research.

In the following sections, we review related research in regularity exploitation, and present a synthesis strategy that reduces interconnect power by applying this concept.

8.3. Related work

In high-level synthesis, the regularity issue has been addressed for several different purposes including partitioning, hierarchical scheduling, and instruction set selection. Rao and Kurdahi used regularity extraction to partition a digital system in order to minimize the overall design effort [104]. The idea was to reduce the design effort by maximally "cloning" a few repeating computational patterns or templates. At the same time, they also address the problem of reducing the critical path through inter-pattern communications in the overall design. Later they used regularity of the extracted partitions for hierarchical scheduling and allocation using the area-time characteristics of the component templates [105].

In [24], Corazao et al. used templates from a pre-defined library to match computation patterns in the algorithm for instruction set selection to maximize speed. Geurts et al. captured the concept of regularity for the synthesis of application-specific units (ASUs) for high-performance applications [37]. In their work, the algorithm is partitioned into clusters of operations and clusters that have similar structure (similar computation patterns) are implemented on the same ASU.

Some research works proposed for multiplexor minimization are also based on concepts that can easily be classified under regularity extraction even though the original work did not specifically state this. In particular, Park presented a method to reduce interconnect by sharing paths in the input algorithm [90]. This can be looked at as exploiting regularity where the patterns are limited to be paths instead of arbitrary patterns.

A quantitative measure of the regularity of a given algorithm is proposed in [41]. Regularity extraction is also studied in other fields such as code generation [1] and technology mapping [55].

Although regularity has been studied from a variety of aspects in high-level synthesis and other areas, no work has been done to exploit it during assignment for low power. We study the power aspects for the first time; though we do not directly use any of the techniques mentioned above, it would be interesting to study whether any of them can be modified to address the power issue and how it affects the results.

8.4. Overall strategy for exploiting regularity

This section explains our overall approach and provides the necessary background. We first present the targeted architecture model. Section 8.4.2 discusses the main issues and problems in regularity exploitation, Section 8.4.3 explains some terminology, and Section 8.4.4 presents the basic ideas behind our approach.

8.4.1. Architecture model

In this chapter, the multiplexor-based interconnect model is used instead of the general model with both multiplexors and tristate buffers (refer Section 1.4.4). While in the general case, the complexity of the overall interconnect is measured by the sum of the tristate buffers and multiplexors, in the multiplexor-based model, it is measured by the total multiplexors which allows us to study the improvements more easily. Note that the generalized interconnect model can be generated by trading-off multiplexors for tristate buffers; therefore, the improvements reported here for multiplexors would simply be divided over multiplexors and tristate buffers in the general case.

8.4.2. Regularity exploitation: issues

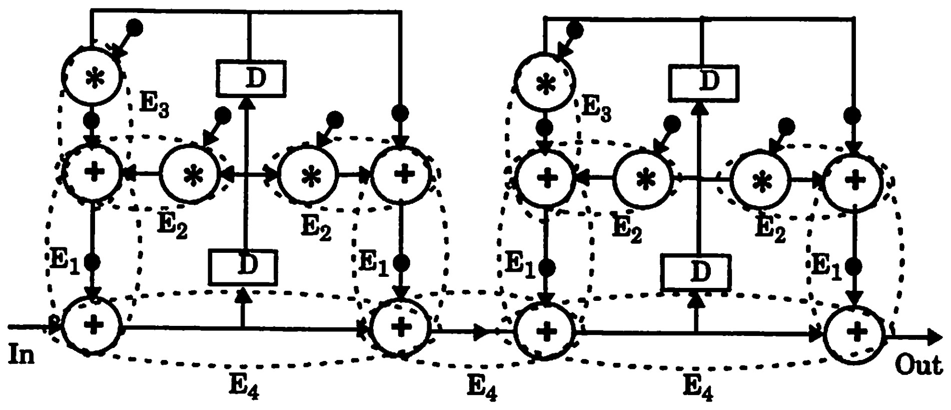
Finding a good regular mapping for a given algorithm is complicated by several factors. Detecting regularity involves *finding common computational patterns* in a given graph and *detecting all matches for each pattern*, both of which are difficult problems. Enumerating all the patterns in a graph has exponential complexity and finding all matches of a given pattern in a graph has subgraph isomorphism as a sub-problem which is known to be NP-complete [35]. Therefore it is not possible to consider and match all possible patterns; only few of the patterns can be considered. The importance of a pattern depends on its matchings since the power saving from its use in assignment depends on its occurrence in a specific design. Thus, the pattern selection and matching processes are closely inter-related.

Clearly, detecting and matching arbitrary patterns is infeasible. For a practical implementation, therefore, the search space must be limited to only a subset of patterns. In our methodology, we *detect and match only two-node patterns*. Each pattern in simply an edge in the graph with a source node and destination node, and the total number of patterns is simply the number of edges in the graph. Therefore the complexity of both the pattern detection and matching tasks is linear in the number of edges.

Given a set of computational patterns and their matches in the algorithm, the next step is to decide the patterns and matches to be used in the assignment. Since a node may be part of several different patterns, the best match (or matches) for it depends on the objective function, the overall throughput constraints, and the associated area costs, all of which must be carefully evaluated. We address these issues in detail in the following sections.

8.4.3. Terminology

As mentioned in the last section, we consider only two-node patterns. Each two-node pattern is called an *E-instance*, so named since it corresponds an edge of the graph. E-instances are classified into types, called *E-templates*, based on the type of the source node, the type of the destination node, and the input port of the destination to which the source is attached. For example, an E-template may be composed of an addition connected to the left input of a subtract operation (which is different from an E-template composed of an addition connected to the right input of a subtraction). The *coverage* of an E-template is defined as the total E-instances of that type divided by the total number of edges in the graph. This corresponds to the fraction of the graph that is covered by that E-template and hence represents its frequency of recurrence. Since each node may be part of more than one E-template, an *E-list* of a node is defined as the list of E-templates for which the node is a source or destination.



(a)

E-template	Coverage
E_1 (add → add.right)	4/26
E_2 (mult → add.left)	4/26
E_3 (mult → add.right)	2/26
E_4 (add → add.left)	3/26

(b)

Figure 8.4. Some E-templates in a fourth-order cascade filter and their coverages.

Figure 8.4a shows E-instances of different E-template types in a fourth-order cascade filter; the edges to the right input ports of each operation are indicated with a dot. The coverages of the E-templates are shown in Figure 8.4b. For example, the E-template E_1, from an add operation to the right input of an add operation, covers 4 out of the 26 edges in the graph.

8.4.4. The core approach

The main idea behind our assignment and allocation scheme is to assign E-templates as a whole in order to preserve the two-node regularity of the algorithm. Thus the data transfers associated with E-instances assigned to the same pair of hardware units can use the same bus without any extra multiplexors or buffers, and without increasing the fan-out of the bus.

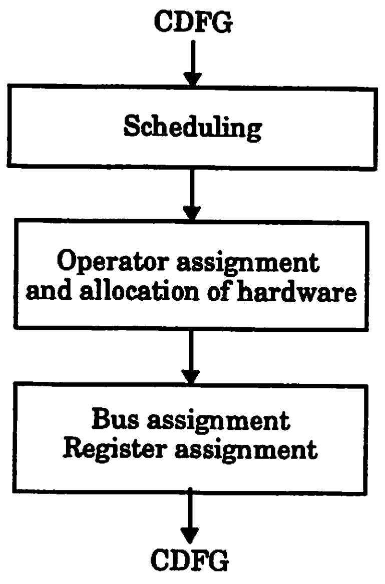


Figure 8.5. Synthesis flow.

The overall flow of our synthesis system is shown in Figure 8.5. Since the schedule is determined first, it has a large impact on the amount of regularity that can be captured during assignment. Therefore, a new scheduling algorithm which minimizes the cost of the E-templates along with the overall area is also proposed. Its aim is to derive a schedule that *enables* a regular assignment of operations to hardware. Sections 8.5 and 8.6 present the details of our allocation, assignment, and scheduling techniques, respectively.

8.5. E-template based assignment and allocation

The input to the assignment tool is the dataflow graph whose operations have been scheduled to occur in specific time steps or clock cycles in the overall sample period (refer Figure 8.5). The goal of the assignment task is to map each operation onto specific hardware resources such that nodes that are scheduled in the same time step cannot be assigned to the same hardware unit. The main idea behind our assignment strategy is to constrain the assignment process so that data transfers of the same source and destination types are mapped onto the same bus with minimum multiplexing or buffering at their sources or destinations. Resources are allocated based on the needs of the assignment scheme. We begin by briefly explaining a related assignment approach based on vertex coloring.

Vertex coloring and clique partitioning techniques have been widely used in the high-level synthesis literature for assignment tasks [103, 116]. In the vertex-coloring based approach the hardware sharing conflicts imposed by the schedule are captured in a *conflict graph* where the a node represents a graph operation and an edge between a pair of nodes represents a conflict due to which they cannot be assigned to the same hardware units. Conflicts between a pair of nodes may be due to two reasons: either the nodes require different type of hardware, or they have overlapping schedules. A minimum vertex-coloring of the conflict graph gives a valid assignment of the operations to hardware where each color represents a specific hardware unit.

While the vertex-coloring approach minimizes the number of hardware units used in the implementation, it ignores the regularity of the graph. In order to specifically exploit regularity, we propose a scheme that assigns E-instances as a whole instead of assigning the nodes individually. Analogous to the vertex-coloring approach, restrictions on the assignment are represented in conflict graphs. However, since a node can be part of several E-instances, all E-instances cannot be assigned at the same time. The coloring is done in stages; in each stage, the most promising E-template is selected and the conflict graph of its E-instances is considered for assignment.

A coloring of this conflict graph would give an assignment for all the E-instances of that E-template such that the minimum number of hardware-unit pairs are used. However this would limit hardware-sharing opportunities for the E-instances of other E-templates that share some of the nodes. Therefore, instead of a complete vertex coloring on the template's conflict graph, only the *maximum independent set* (MIS) is chosen for assignment to the same pair of hardware units. This scheme allows the rest of the E-instances to be compared with E-instances of other E-templates so that the next best E-template can be selected for assignment in the next iteration.

The maximum independent set approach concentrates on sharing between a large number of similar patterns rather than reducing the overall hardware used. This is because the maximum gains come from reusing an interconnect, once instantiated, as much as possible. When only a few instances of each template remain, they are assigned using conventional vertex coloring. This results in designs in which a few of the buses with low fan-outs are used very often while other buses that may have much higher fan-outs are used less. Notice that the goal is not to minimize the overall fan-outs (interconnect area), but to reduce fan-outs of only the highly used connections.

In Section 8.5.1 we redefine the concept of the conflict graph in this context. A technique to find its maximum independent set is presented in Section 8.5.2 and the assignment and allocation algorithm is described in detail in Section 8.5.3. Finally, Section 8.5.5 discusses some of the characteristics of the suggested technique.

8.5.1. Conflict graphs

At any stage in the assignment algorithm, the conflict graph, C_k, for E-template E_k, is derived in the following way. Each unassigned E-instance (for which at least one node, source or destination, is unassigned) of type E_k is represented by a node in the conflict graph (*conflict-node*). Each edge in the conflict graph (*conflict-edge*) represents a conflict between corresponding E-instances due to which they cannot be assigned to the same pair of hardware units.

Conflicts between E-instances are derived from conflicts between their constituent nodes, i.e., if there is a conflict between their source nodes or destination nodes. Conflicts between two nodes may be due to the following four reasons.

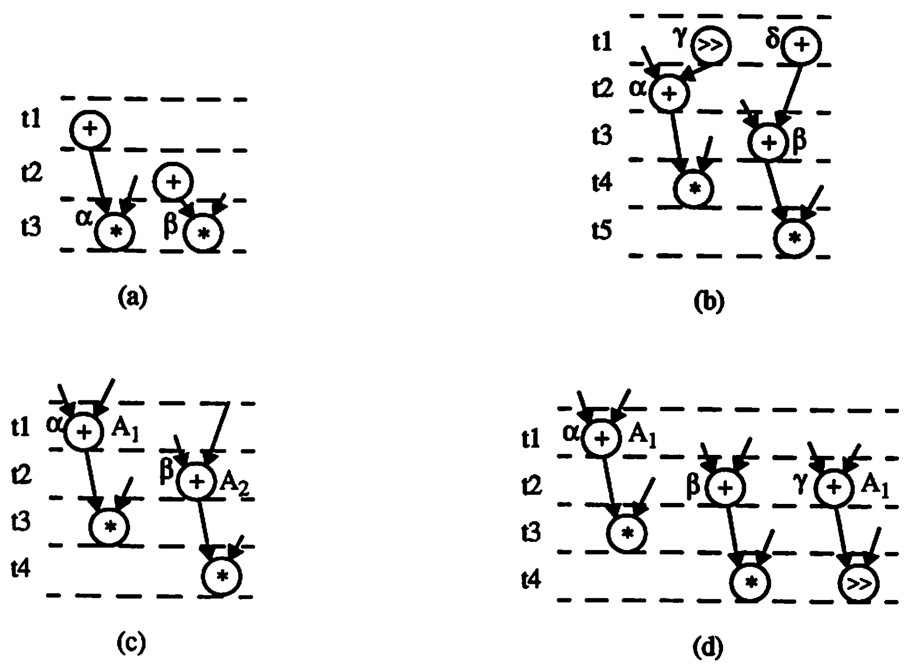


Figure 8.6. The different conflict types: (a) scheduling conflict, (b) register-bandwidth conflict, (c) assignment conflict, (d) assign-schedule conflict.

Scheduling conflict

A scheduling conflict occurs between two nodes if there is an overlap in the time slots in which they are scheduled. Figure 8.6a shows a scheduling conflict between nodes α and β.

Register-bandwidth conflict

Due to the distributed, single-ported nature of register files (refer Section 1.4.4) in our hardware model, there is a register-bandwidth conflict between two nodes if the producers of their left inputs (or right inputs) write their results in the same time step. In Figure 8.6b there is a register bandwidth conflict between nodes α and β since node γ writes into the right register file of α at the same time as δ writes into the right register file of node β.

Assignment conflict

An assignment conflict occurs if the nodes are assigned to different functional units. In Figure 8.6c there is an assignment conflict between nodes α and β since they are assigned to different adders (A_1 and A_2).

Assign-schedule conflict

An assign-schedule conflict arises between two nodes if one of them is already assigned to a hardware resource and the other has a scheduling or register-bandwidth conflict with that hardware resource. A node is said to have a scheduling or register-bandwidth conflict with a hardware resource if it has a scheduling or register bandwidth conflict with any of the nodes that are already assigned to that resource. In Figure 8.6d, there is a assign-schedule conflict between nodes α and β since β has a scheduling conflict with hardware resource that α is assigned to (A_1).

Notice that conflict edges are introduced between two E-instances only if there is a conflict between their sources or between their destination nodes. A special case occurs when the source and destination nodes of an E-template are the same type, since they can share the same hardware. In this case, two conflict graphs are generated — one that does not allow sharing between sources and destinations and one that does. In the first case, conflicts are checked only between the sources of two E-instances or between their destinations. In the latter case, additional edges are introduced to represent conflicts between the source node of one E-instance and the destination node of the other. These two conflict graphs are compared and the one with higher MIS cardinality is selected.

8.5.2. Maximum independent set of the conflict graph

The maximum independent set, or MIS, of a graph is defined as the largest subset of nodes of the graph, such that are no edges between any pair of nodes in that subset [35]. For the conflict graph defined above, the MIS is the maximum set of E-instances with no conflict edges between them and therefore represents the largest set of E-instances that can be assigned to the same pair of hardware units.

The commonality measure

Before explaining the algorithm for selecting the maximum independent set, we introduce the concept of the *commonality* of a particular E-instance I, with respect to a set of E-instances S. This is a measure of the number of common E-templates between the nodes of I and those of the elements of S.

Let I_{source} be the source node of E-instance I, *E-list*(I_{source}) be its E-list, and $N_{source}(S, E_k)$ be the number of instances in S that have the E-template, E_k, in the E-list of their source nodes. The *source commonality* of I with respect to S, $Comm_{source}(I, S)$, is defined as:

$$Comm_{source}(I,S) = \sum_{E_k \in Elist(I_{source})} N_{source}(S,E_k) \qquad \text{(Eq. 8.1)}$$

Similarly, let I_{dest} be the destination node of E-instance I, *E-list*(I_{dest}) be its E-list, and $N_{dest}(S, E_k)$ be the number of instances in S that have the E-template, E_k, in the E-list of their destination nodes. The *destination commonality* of I with respect to S, $Comm_{dest}(I, S)$, is defined as:

$$Comm_{dest}(I,S) = \sum_{E_k \in Elist(I_{dest})} N_{dest}(S,E_k) \qquad \text{(Eq. 8.2)}$$

The source (destination) commonality of I with respect to S is a measure of the common E-templates between the sources (destinations) of I and those of the elements of S. The commonality of an E-instance with respect to a set of E-instances is the sum of its source and destination commonalities with respect to that set.

MIS algorithm

While finding the optimum solution to the maximum independent set problem is NP-complete, several greedy heuristics have been proposed for it. In particular, the minimum-degree greedy heuristic is often used and has been shown to work well for bounded degree graphs. In fact, it has been proved that the results are within factors of $(2d+3)/5$ and $(d+1)/2$ of the optimal, where d is average degree of the nodes in the graph, [44, 49], and within $\Delta - 1$ where Δ is the maximum degree of the graph [114].

Fortunately, the graphs encountered in our targeted applications typically have nodes with limited fan-ins (up to three) and fan-outs (up to three). Therefore, we use the minimum-degree greedy algorithm. This algorithm iteratively selects the conflict-node with the least number of neighbors, adds it to the MIS and removes the node and its neighbors from the conflict graph. In the case of a tie, we select conflict-nodes with the highest *commonality* with respect to the conflict-nodes already selected. *This favors the selection of a maximum independent set with E-instances whose nodes are also parts of other common E-templates. Therefore, they can be chosen for hardware sharing as part of those other E-templates in future iterations of the algorithm.* This adds a degree of look-ahead to the algorithm that helps in preserving larger patterns.

This concept is illustrated through the example of Figure 8.7. Assume that the add-multiply template shown is being considered for assignment and instances C and D have a scheduling conflict between them. Assume further that E-instances A and B are already selected in the MIS and there is a tie between C and D since both have the same number of neighbors. The source commonality of C with respect to the set {A, B} is 4, while that of D with respect to {A, B} is only 2, and therefore C is selected in the MIS. This choice of the MIS maximizes the chances of further sharing of E-instance in future iterations since all the add-add instances and three of the four shift-add

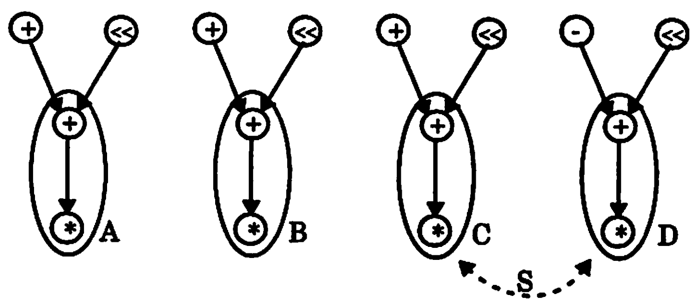


Figure 8.7. Using the commonality measure as a look-ahead technique.

instances can be shared later. The commonality measure allows the scheme to "look ahead" and exploit larger recurring patterns to a certain extent.

8.5.3. E-template based assignment strategy

Initially, all the E-templates in the given graph are detected and their coverages calculated. The overall assignment and allocation scheme is divided into two phases. In the first phase, E-instances are assigned to pairs of hardware units, while in the second phase, nodes are assigned independently using vertex coloring.

The assignment of E-instances is done iteratively. In each iteration the most promising E-template is selected based on its coverage and the MIS cardinality of its conflict graph. The E-instances corresponding to the maximum independent set of its conflict graph are assigned and then removed from the corresponding E-template list and the coverage of the E-template is recalculated. Each of these steps are explained below. A pseudo code for the overall assignment and allocation algorithm is given in Figure 8.8.

Selection of best E-template

In each iteration, the E-template with the highest coverage is selected for assignment. In the case of a tie, conflict graphs of the competing E-templates are created, their maximum independent sets are derived, and the E-template with the independent set of higher cardinality is selected. Although the MIS cardinality determines the number of E-instances that can be simultaneously assigned to

```
ETemplat_list = Make_ETemplate_list(Original_graph)
Calculate_coverage(ETemplate_list)
Remove_ETemplates_with_coverage_below_threshold(ETemplate_list)
Best_ETemplate = Select_best_ETemplate(ETemplate_list)
while (Best_ETemplate != NULL) {
    Conflict_graph = Create_conflict_graph(Best_ETemplate -> List)
    MIS_list = Max_independent_set(Conflict_graph)
    Allocate_and_assign_list(MIS_list)
    Update_ETemplates(ETemplate_list)
    Calculate_coverage(ETemplate_list)
    Remove_ETemplates_with_coverage_below_threshold(ETemplate_list)
    Best_template = Select_best_ETemplate(ETemplate_list)
}
Residual_list = Make_list_of_unassigned_nodes(Original_graph)
Vertex_coloring_based_assignment(Residual_list)
```

Figure 8.8. Pseudo-code for the overall assignment and allocation algorithm.

the same pair of hardware units, computing it for all E-templates is time consuming. Hence, we select E-templates based on the coverage, and use MIS cardinality only to resolve ties.

Assignment of E-instances

In this step, the conflict graph of the selected E-template is created and the E-instances corresponding to its maximum independent set are assigned to a pair of hardware units in the following way: the sources of the E-instances are assigned first. If any of the source nodes are already assigned, all others are assigned to the same unit. Otherwise, a new hardware unit is allocated and assigned to all the source nodes. The destination nodes are then assigned in the same way.

Notice that it is not possible for the source nodes (or the destination nodes) of a pair of E-instances in the MIS to be already assigned to different hardware units since this would have caused an assignment conflict between them. Also, if only one of them is assigned to a unit, the other node can also be assigned to the same unit since there are no assign-schedule conflicts between them.

Updating remaining E-templates

After the instance assignment step, E-instances whose source and destination nodes are assigned, are removed and the coverages of the remaining E-templates is re-calculated.

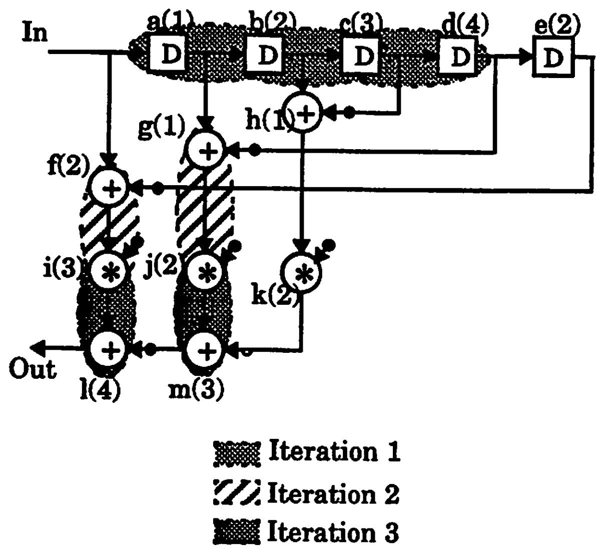
As more nodes in the original graph are assigned, the coverage of each of the remaining E-templates reduces, decreasing the inherent regularity in the remaining graph and reducing the advantages of exploiting it. Also, more assignment and assign-schedule conflicts are created, reducing the hardware sharing between pattern instances and introducing significant area overhead. Thus, in each iteration, the advantages of exploiting regularity reduce while the associated overhead grows. Therefore, when the coverages of all E-templates falls below a certain threshold, called the *coverage-threshold*, the E-template based assignment phase is terminated and a vertex-coloring based scheme is used for assignment of the remaining nodes. For implementation efficiency, E-templates whose coverages fall below the coverage-threshold are eliminated in each iteration, reducing the search space in the future iterations of the algorithm. The impact of the coverage threshold is studied in the results section.

8.5.4. Example

In this section we demonstrate the operation of the algorithm on a small example. Consider the reverse symmetric FIR filter shown in Figure 8.9a. The numbers in brackets next to the node names show the time step that the node is scheduled in. Figure 8.9b shows the E-templates and their coverages. The coverage threshold is set at 1/8. The iterations in the first phase of the algorithm are detailed below.

Iteration 1

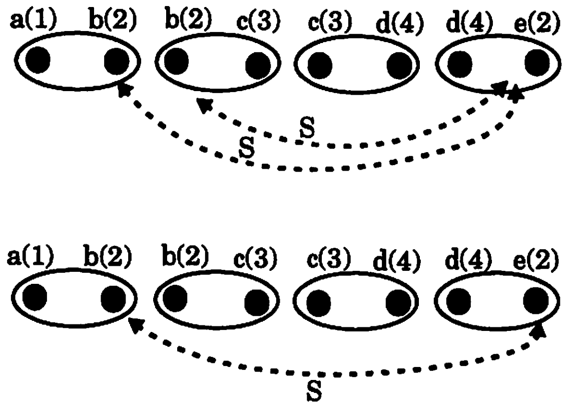
E-template E_0 is selected for assignment and its conflict graphs are shown in Figure 8.9c; there is a scheduling conflict between the nodes *b* and *e*. Since the source and destination nodes are of the same type, two conflict graphs are cre-

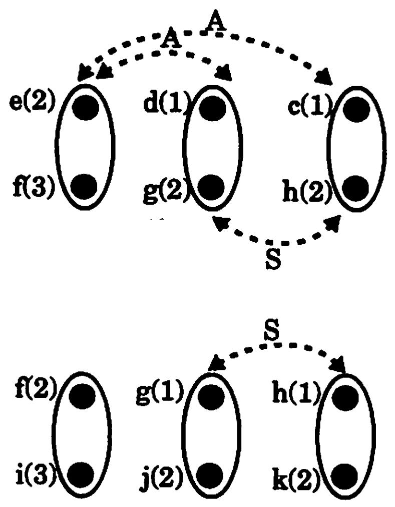


(a)

E-template name	Description	Coverage
E_0	D → D	4/16
E_1	D → add.left	2/16
E_2	D → add.right	3/16
E_3	add → mult.left	3/16
E_4	mult → add.left	2/16

(b)


a(1) b(2) b(2) c(3) c(3) d(4) d(4) e(2)
S
S
a(1) b(2) b(2) c(3) c(3) d(4) d(4) e(2)
S



(c) (d)

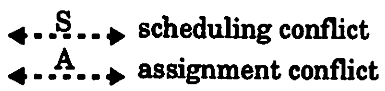


Figure 8.9. Effect of E-template based assignment on a fifth-order reverse-symmetric FIR filter: (a) E-templates assigned in each iteration, (b) E-templates and their coverages, (c) conflict graphs of E-template E_1 considered in iteration 1, allowing (above) and not allowing (below) hardware sharing between sources and destinations, (d) conflict graph of templates E_2 (above) and E_3 (below) considered in iteration 2.

ated — one that allows hardware sharing between the sources and destinations (top figure) and one that does not (bottom figure). Since both the conflict graphs have the same MIS cardinality (two), the one which allows source-destination sharing is chosen due to lower area. The selected MIS of E-instances is *a-b, b-c, c-d*. A transfer unit, T_1, is allocated to implement the delay and the source nodes, *a*, *b*, and *c* are assigned to it. As a result, some destination nodes get assigned to T_1 and therefore, the rest are also assigned it.

The assigned E-instances are removed from the graph, after which the coverage of E_0 falls below the threshold and it is eliminated from the E-template list.

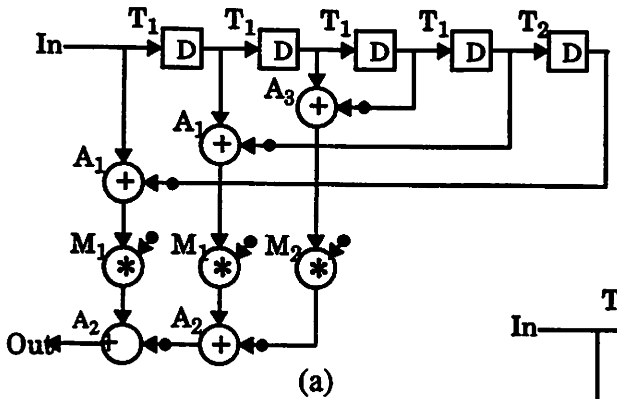
Iteration 2

E-templates E_2 (*c-h, d-g, e-f*) and E_3 (*f-i, g-j, h-k*) have the highest coverage and their MIS cardinalities are 1 (assignment conflicts between *c* & *e*, and *d* & *e*; and scheduling conflict between *g* & *h*) and 2 (scheduling conflict between *g* & *h*), respectively. The conflict graphs of these two E-templates is shown in Figure 8.9d. E_3 is selected due to its higher MIS cardinality. Since the commonality measure of *g-j* is larger than *h-k* (*g-j* has more templates in common with the selected set *f-i*), the first two instances are selected and their sources and destinations are assigned to adder A_1 and the multiplier M_1, respectively. The coverages of unassigned instances of E_1, E_2, and E_3 drop below the threshold and they are eliminated.

Iteration 3

E-template E_4 is selected and both its instances are assigned to the multiplier, M_1, and adder, A_2, pair. Note that the assignment of both instances of E_4 to the same pair of hardware units is made possible due to the consideration of commonality in the previous iteration.

At this point, all E-templates are eliminated and the remaining nodes are assigned using vertex coloring. The final assignment obtained is shown in Figure 8.10a. Figure 8.10b shows a different assignment of operations that does not consider regularity. Comparing Figures 8.10a and 8.10b, it is seen that the total number of fan-outs



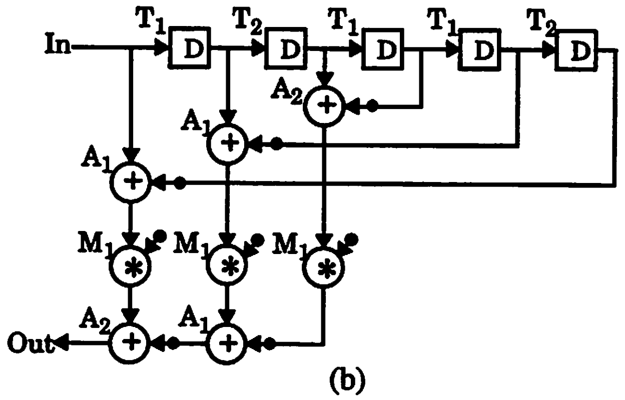


Figure 8.10. Final assignments obtained using: (a) E-template based technique, (b) vertex coloring.

is reduced from 14 to 10 by exploiting regularity — a 30% reduction. It should be noted that this is a small example with limited regularity and is used for demonstration purposes only.

8.5.5. Discussion

Although the E-template based approach has its limitations since it ignores larger recurring computational patterns and limits the amount of regularity exploited, is also has some advantages.

- While detecting and matching arbitrary patterns is NP-complete, these tasks take linear time for E-templates.

- Even if large patterns are detected, it may not be possible to map all the corresponding nodes of the pattern instances to unique hardware units either due to large area penalties involved or due to restrictions imposed by the schedule.
- The repetition of E-templates is likely to be higher than that of larger patterns and more reuse of the same interconnect is possible.
- The look-ahead scheme based on the commonality measure attempts the match larger patterns in the graph.

8.6. E-template based scheduling

In the assignment and allocation scheme presented in the previous section, it was assumed that the CDFG is already scheduled. Since the schedule is performed before assignment, it can greatly impact the amount of regularity that can be exploited during the assignment phase. This section studies this impact and proposes a modified scheduling approach that favors a regular assignment.

8.6.1. Impact of the schedule on assignment regularity

The impact of the schedule on the regularity exploitation is illustrated in this section using the force-directed scheduling technique on an example proposed by Paulin and Knight [91]. The technique is based on the concept of a distribution graph (DG) which plots the expected number of resources required as a function of time. Initially, each operation is assumed to have equal probability of being scheduled in any time-step between its as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) times with the sum of the probabilities being one. If an operation takes longer than one clock cycle, the ASAP and ALAP times correspond to its starting times (for details on how this case is addressed, the reader is referred to [91]). In any time-step, the height of the DG is a measure of the expected number of operators needed. The interval between the ASAP and ALAP times, or time frame, of each operation represents the time steps in which the operation can be scheduled. *The main idea is to schedule each operation into a time slot such that the maximum height of the DG is minimized.* The maximum height of the final DG (after scheduling) corresponds to the number of resources required to implement the algorithm with the specified schedule.

Each operation, *i*, has a force, *F(i, j)*, associated with each time step, *j*, in its time frame (a, b), that is defined as follows:

$$F(i,j) = \sum_{k=a}^{b} DG_k \bullet \Delta DG_k \qquad \text{(Eq. 8.3)}$$

where DG_k is the height of the DG at time step *k*, and ΔDG_k is the change in the height of the DG at time step *k* due to the assignment of operation *i* at time step *j*. This force represents the effect of scheduling it onto that time step; the lower the value of the force, the better it is to schedule it in the corresponding time step.

Force-directed scheduling proceeds interactively — evaluating the force associated with scheduling each node in all its possible time-steps and scheduling the node and time-step pair with the least force, in each iteration. Several enhancements to the basic force-directed scheduling algorithm have been proposed. We use the gradual time-frame reduction technique proposed by Verhaegh et al. [129] in our implementation.

Consider the impact of the force-directed technique on an example with two E-templates, E_1 and E_2, with four and two instances, respectively, as shown in Figure 8.11a. From the ASAP and ALAP times (marked next to each node), it is clear that it is possible to map the multiply operations of all multiply-add E-instances (E_1) to the same multiplier and similarly those of the shift-multiply E-instances (E_2). The initial distribution graph for multiplications (referred to in this work as *functional-unit distribution-graph* or *FDG*) is shown in Figure 8.11b and a possible schedule from the force-directed algorithm is shown in Figure 8.11c.

The most important point to notice is that nodes *b* and *c* are scheduled in the same time-slot and cannot be assigned to the same hardware resource. Given this schedule, it is not possible to map all instances of the multiply-add template onto the same pair of hardware units. The schedule thus obviates the regular assignment that was initially possible for the design based on the timing constraints alone.

8.6.2. E-template-based scheduling technique

As shown in this example, a force-directed schedule may preclude a regular assignment since it only minimizes the functional units required and ignores the cost of connections. We therefore modify the cost function to include connection costs. We do this by introducing a new distribution graph called the *connection distribution-graph* or *CDG.*

Each E-template is associated with two CDGs — one for its sources and one for its destinations. For a given E-template, E_k, the source-CDG is derived from the time distributions of the source nodes of all instances of the E-template, while the destination-CDG is derived from time distributions of the destination nodes. These distribution graphs together represent the cost of the interconnect between the source and destination nodes.

The total force on any node is the *weighted sum* of the forces from the FDG of the relevant functional unit, the source-CDGs of all the E-templates for which this node is the source node and the destination-CDGs of all the E-templates for which this node is the destination node. The weight associated with the FDG is proportional to the cost of the functional unit while the weight of each CDG is proportional to the coverage of the corresponding E-template. This weighting scheme gives preference to connections that are repeated more often. This modified cost function produces a schedule that favors the assignment of E-instances of the same E-template to the same pair of hardware units while also minimizing the total area. Since scheduling sources (or destinations) of E-instances of the same type in the same time-slot results in increased height

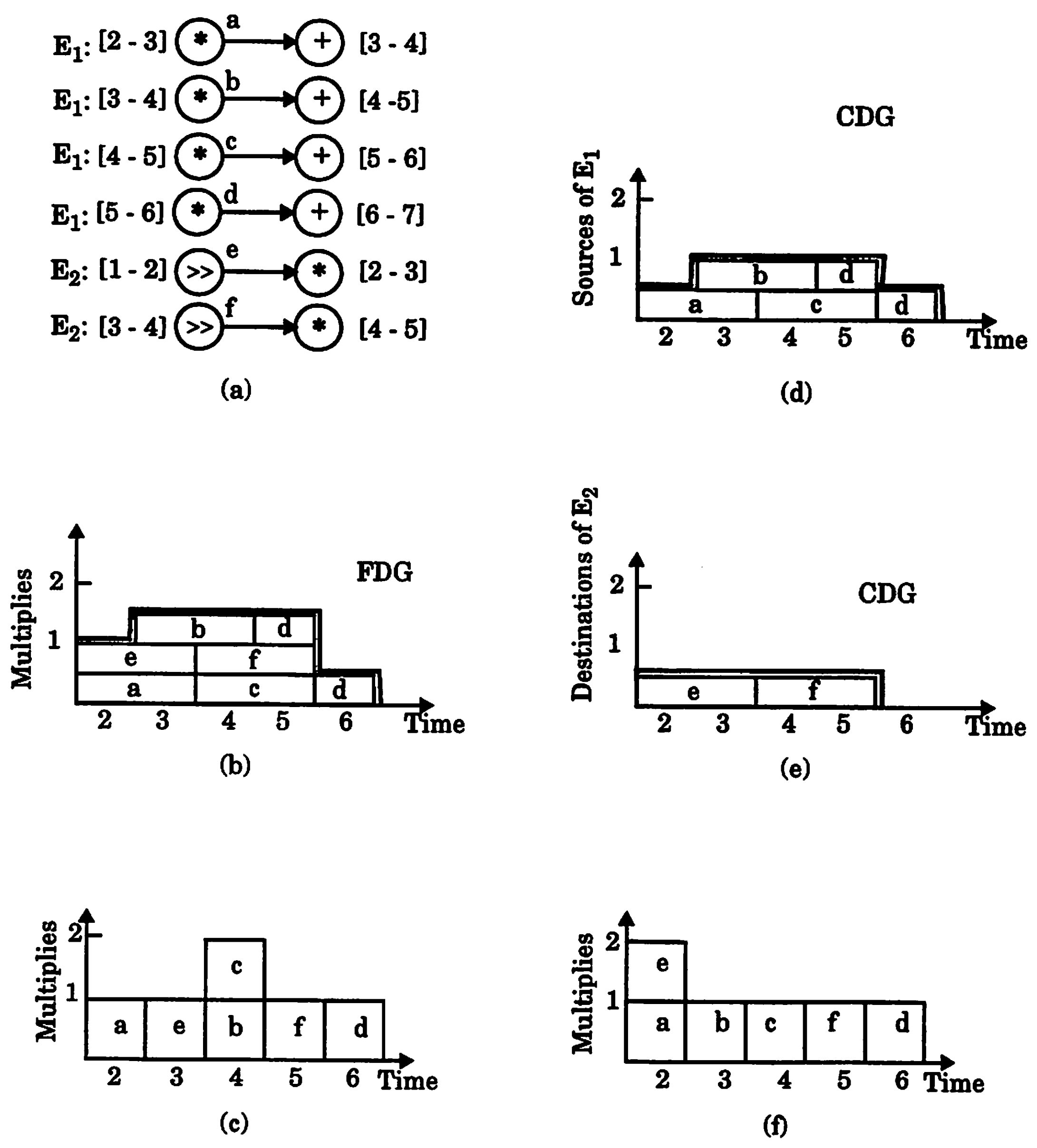


Figure 8.11. The effect of using connection distribution graphs: (a) instances of two E-templates with their ASAP and ALAP times, (b) initial FDG for multiply operations, (c) final distribution graph using only FDGs, (d) initial source-CDGs of E-template E_1, (e) initial destination-CDG of E-template E_2, (f) Final distribution graphs using FDGs and CDGs.

of the corresponding source-CDG (or destination-CDG), the scheduler tends to avoid this.

Consider the example of Figure 8.11 again. The source-CDG of E_1 and the destination-CDG of E_2 are shown in Figures 8.11d and 8.11e, respectively, and the final distribution graph that minimizes the weighted sum of the FDG and the two CDGs is shown in Figure 8.11f. In this schedule, since the scheduler tries to minimize the height of the CDGs along with that of the FDG, multiply operations of all multiply-add E-instances are scheduled at different time slots and therefore can be mapped onto the same hardware unit. Similarly the multiply operations of all shift-multiply E-instances can be mapped to the same multiplier.

In this work, we have only proposed modifications to the force-directed scheduling scheme, but it is also possible to modify other scheduling schemes to consider cost of E-template based connections. This may be done by incorporating the cost of the E-templates to the cost function used the scheduling algorithm.

8.7. Results

In this section we present the results of our synthesis approach. A set of 15 examples, consisting of different structures of FIR filters, IIR filters, and transforms were selected for experimentation. All the examples were evaluated for maximum throughput implementations (total time available equal to critical path) with no transformations. Overall power estimates were obtained from the SPA architectural power analysis tool. The bus power is computed using the model presented in 6.4.3 for fan-out optimized designs. In order to separately evaluate the effect of the fan-out and multiplexors optimizations, we use white noise capacitance models. This excludes any effects that may arise from changes in activity.

A series of experiments studying the different aspects of the E-template based synthesis approach are described. First the impact of varying the coverage threshold is studied in Section 8.7.1. Then the quality of results obtained from the E-template based synthesis methodology is compared with two other scheduling/assignment paradigms — the Hyper synthesis scheme and a force-directed scheduling followed by vertex coloring based assignment (FDS-VC). Section 8.7.2 analyzes the power improvements over these two paradigms and Section 8.7.3 presents the effects on area.

8.7.1. Effect of varying the coverage threshold

The graphs in Figure 8.12 show the effect of the coverage threshold on the power consumption of buses and multiplexors. The values were obtained on our benchmark set and normalized with respect to the base case where our technique is not used (threshold = 100%). Recall that E-instances are assigned as a whole until their coverages drop below the coverage threshold and the remainder of the graph is assigned using vertex coloring. It is seen that in most cases lowering the threshold (using E-template based assignment for a longer time) results in better solutions since lower thresholds enable more extensive exploitation of regularity.

In some cases, lower thresholds are seen to have an adverse effect on the bus power (in example 2, a 20% threshold gives the lowest bus power). This effect is due to the area overhead associated with the technique which can result in longer (more capacitive) buses and offset the gains from fan-out reduction. The multiplexors power is seen to monotonically reduce with reducing thresholds due to reduced hardware sharing. We found that further lowering the threshold changed the results of only a few examples — 2 improved and 3 worsened.

In general, the coverage threshold has a large impact on the power consumption. Lowering the coverage threshold reduces overall power dissipation up to a certain point after which the power consumption may increase. The optimum threshold is different for each example and needs to be determined separately.

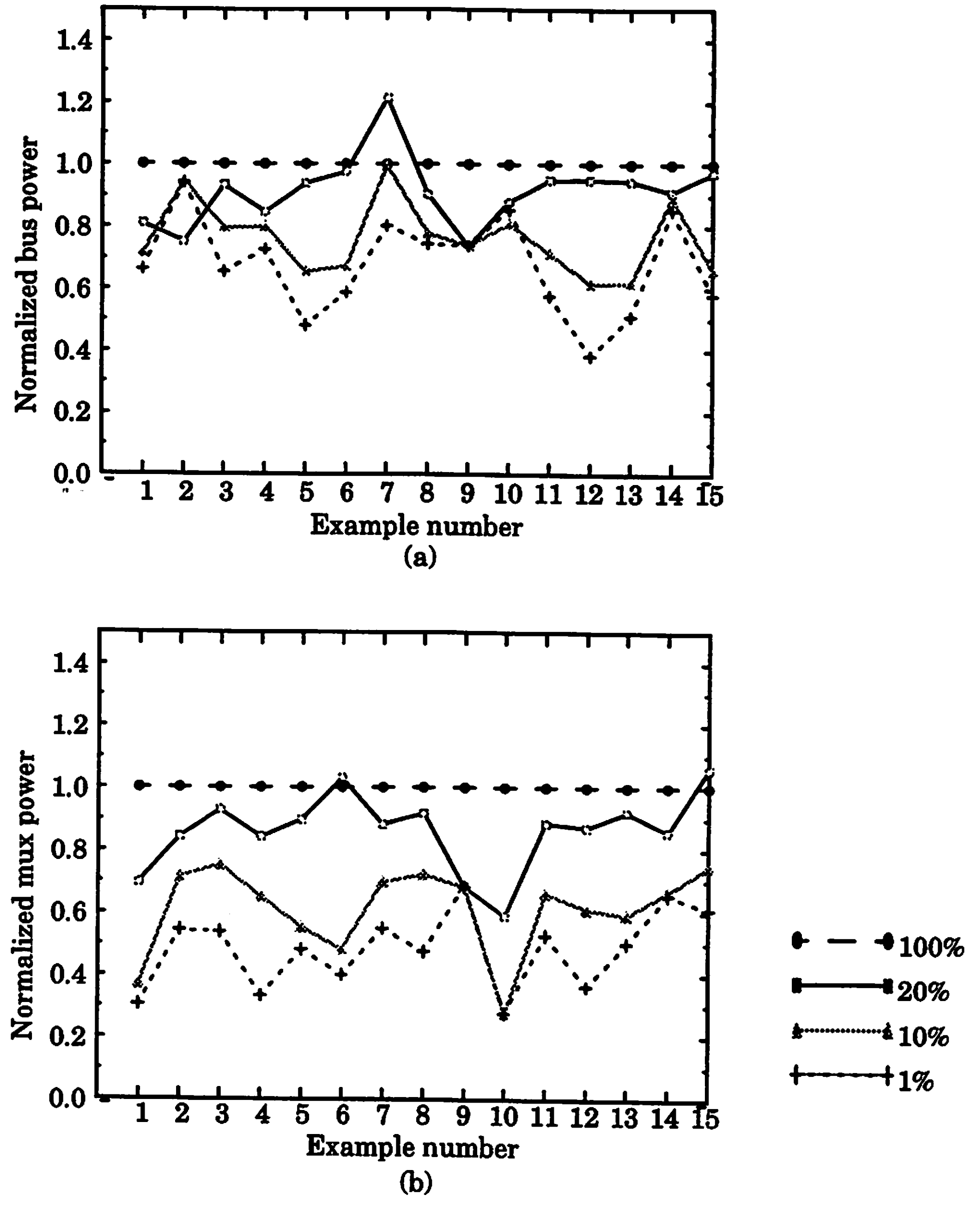


Figure 8.12. The effect of coverage threshold on the power consumption of various components: (a) buses, (b) multiplexors.

8.7.2. Power improvements

As explained before, power improvements in the new approach stem from a reduction in the fan-outs of the most frequently used buses and a decrease in a multiplexing

Table 8.1. Bus fan-outs and multiplexor accesses obtained from the Hyper, FDS-VC, and E-template based synthesis approaches.

	Hyper		FDS-VC		E-template based approach		
	Overall fan-outs	Mux accesses	Overall fan-outs	Mux accesses	Coverage threshold	Overall fan-outs	Mux accesses
parallel4	46	72	38	63	1%	32	22
df	33	55	30	57	20%	28	48
fft11	229	416	151	505	1%	133	275
fft8	49	61	52	51	1%	49	20
firD	91	270	50	190	1%	36	100
firH	70	139	41	123	1%	31	61
gm	37	147	30	210	1%	26	125
dct	97	106	77	102	0.5%	71	55
wavelet	28	96	20	80	20%	18	54
wdf9	30	61	26	47	10%	25	15
DSfir24	84	199	49	161	0.5%	35	110
DSLfir50	149	608	75	473	1%	47	157
DSfir55Mb	160	665	76	508	0.5%	51	280
cascade32	105	785	52	541	1%	50	394
parallel24	247	137	193	234	1%	162	159

overhead. Table 8.1 shows the total fan-outs and the multiplexor accesses in the three different implementation paradigms. The coverage thresholds used in the E-template based approach for each example are also given. The percentage reductions in the overall fan-outs and in multiplexor accesses are shown in the graphs of Figure 8.13. The total number of fan-outs from each unit decrease by 40% and 16% on average with respect to the Hyper and FDS-VC schemes, respectively. Notice that reduction of the *total* fan-outs is not the direct goal of our technique since we try to reduce the fan-out only on buses that are *repeatedly used*. However, a reduction in total fan-outs is seen as a result of reduced fan-outs on several of the buses. The number of multiplexor accesses is also drastically reduced (46% and 45% reduction on average compared to the Hyper and FDS-VC systems, respectively).

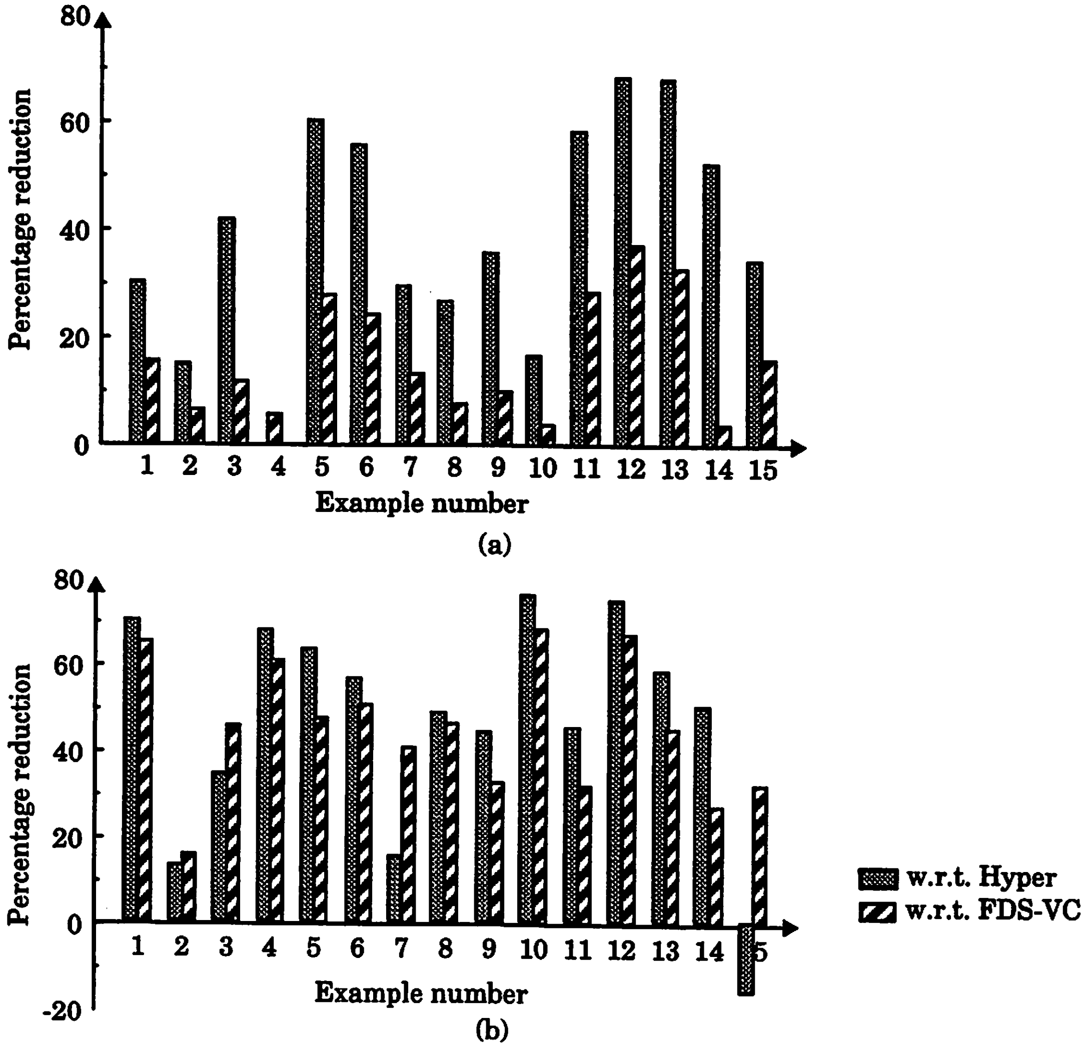


Figure 8.13. Percentage reductions with respect the Hyper and FDS-VC schemes: (a) total fan-outs, (b) multiplexor accesses.

Table 8.2 shows the total bus and multiplexor power obtained on these examples using the three different synthesis paradigms. The graphs in Figure 8.14 show the percentage improvements in bus, multiplexor, and total power compared to the Hyper and the FDS-VC implementations. It is seen that the new approach performs better that the other two in almost all cases. As compared to Hyper, an average of 47% and 49% power savings were obtained for buses and multiplexors, respectively, while compared to FDS-VC, the average reductions in these components were 39% and 49%.

Table 8.2. Bus, multiplexor, and total power consumptions (mW) for three different implementation approaches — Hyper, FDS-VC, and E-template based synthesis.

	Hyper			FDS-VC			E-template based approach		
	Bus	Mux	Total	Bus	Mux	Total	Bus	Mux	Total
parallel4	19.4	2.4	67.6	16.0	2.1	63.3	10.6	0.7	58.0
df	32.0	2.5	120.9	30.3	2.9	118.1	22.8	2.5	110.4
fft11	256.8	17.5	442.4	247.3	21.9	397.1	161.8	11.7	309.2
fft8	22.0	4.2	63.2	21.6	4.8	65.9	15.7	1.6	58.3
firD	32.9	5.9	90.0	23.4	4.1	70.0	11.3	2.0	55.5
firH	33.7	5.5	94.3	25.2	4.7	78.3	14.8	1.9	65.0
gm	9.6	2.0	36.3	8.0	2.4	35.3	6.5	1.3	32.4
dct	21.3	3.3	68.9	20.5	3.6	66.4	15.3	1.7	61.1
wavelet	14.4	2.9	68.0	11.9	1.8	63.0	8.8	1.2	59.2
wdf9	27.7	4.9	92.7	23.7	3.8	87.4	19.2	1.1	59.2
DSfir24	120.1	10.7	245.3	77.5	8.6	187.2	43.4	5.1	148.7
DSLfir50	77.6	13.3	175.7	59.1	8.2	129.0	22.7	3.0	84.4
DSfir55Mb	240.0	16.4	384.4	112.1	12.1	225.3	44.5	6.3	150.4
cascade32	66.9	8.9	134.5	28.5	5.4	84.9	24.3	3.5	80.3
parallel24	132.6	3.7	354.9	170.6	8.4	346.2	98.9	5.1	270.8

Overall average power reductions of 28% (compared to Hyper) and 17% (compared to FDS-VC) were obtained. We also expect to obtain power savings in buffers since smaller buffers can be used to drive the low fan-out, short buses. However, our automated architecture-netlist generation tool uses fixed sized buffers for all data transfers irrespective of the length of the bus being driven, and we are not able to demonstrate these savings.

8.7.3. The effect on area

Since the E-template based approach reduces bus fan-outs and multiplexors, the interconnect area can be expected to reduce. However, as discussed before, the exploitation of regularity adds extra constraints on the assignment problem and results in an increase in the number of functional units used.

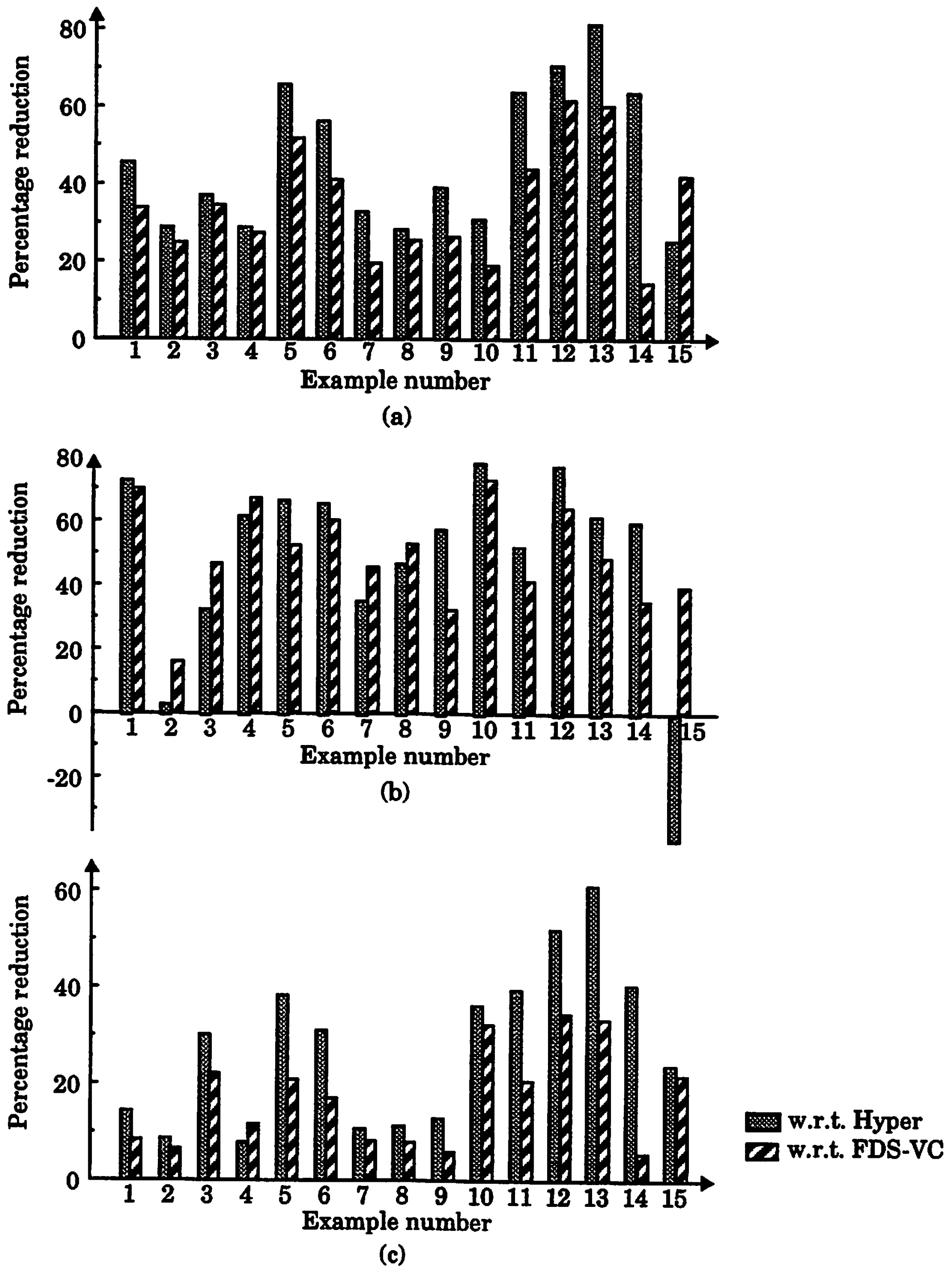


Figure 8.14. Percentage power savings of various components with respect the Hyper and FDS-VC schemes: (a) buses, (b) multiplexors, (c) total.

The graphs in Figure 8.15a and 8.15b show the percentage change in the active area and total chip area, respectively. A positive change represents an increase in area using the E-template based scheme compared to the base case. In each graph, the two lines represent comparisons with the Hyper and FDS-VC implementations. The active area includes the area of the functional units, multiplexors, buffers, and registers. Though the active area increased with respect to the FDS-VC scheme (12% increase on average), the total area reduced due to less wiring in our approach resulting in a 14% decrease in total area on average. In some examples (such as #10, #14), it was seen that the overall area increased but the power reduced.

With respect to Hyper, there is a reduction in both the active area and the total area (26% and 47% average reduction, respectively). Hyper uses an iterative approach for scheduling and allocation and can trade-off design quality versus synthesis time. Though these experiments were performed with higher iteration counts than the default settings, it may be possible to reduce the area by running them for longer times. On the other hand, these results may be due to the completely different scheduling and assignment algorithms used in the Hyper and FDS-VC synthesis schemes. Some further experiments to determine the exact cause would be useful, but are not done here since the results would not directly affect the validity of our approach.

8.8. Conclusions and future work

The allocation, assignment, and scheduling techniques presented in this chapter exploit the regularity and common computational patterns in the algorithm to reduce the fan-outs and fan-ins of the interconnect wires, resulting in reduced bus capacitances and multiplexor accesses. A simple and efficient E-template based assignment and allocation algorithm has been proposed to exploit regularity. A modified force-directed scheduling algorithm is used to produce a schedule favorable for regular assignment. The new synthesis scheme is integrated into the Synergy[1,2] synthesis system.

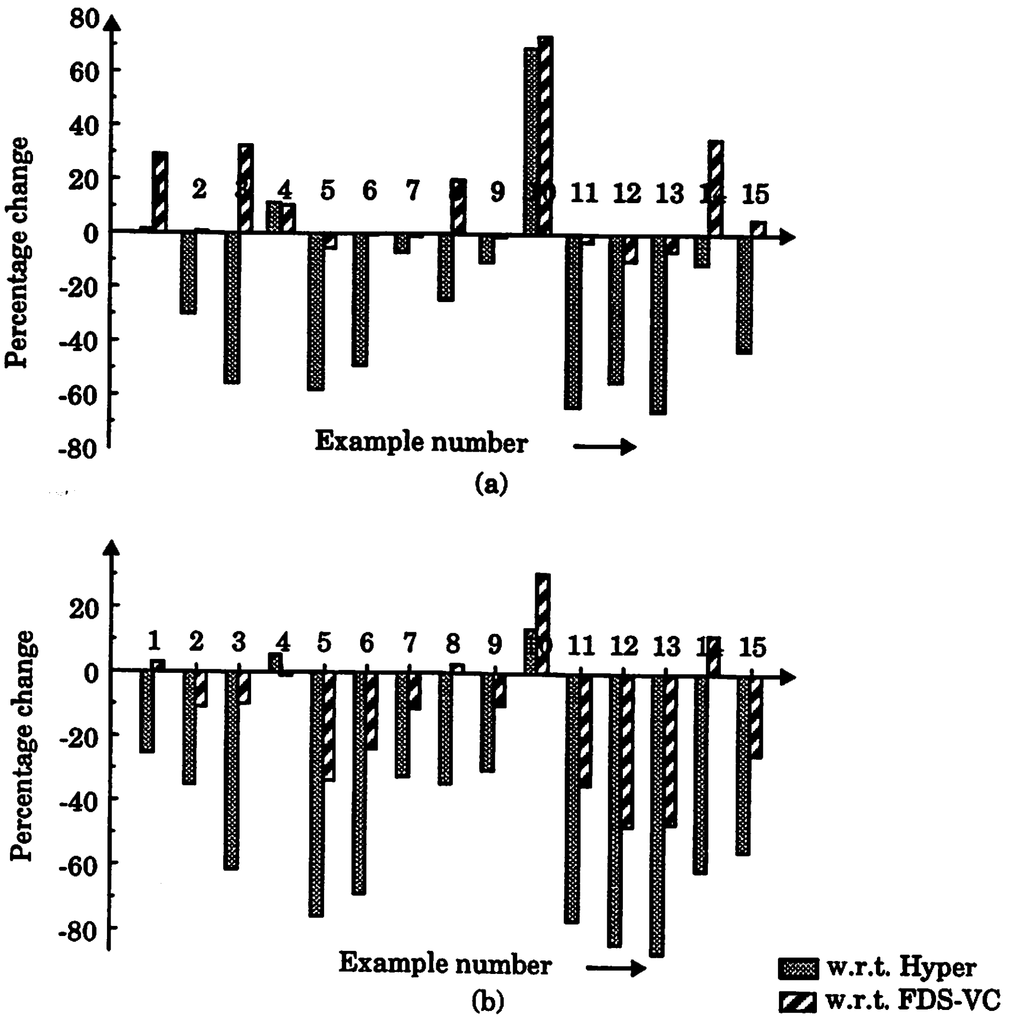


Figure 8.15. Percentage change in the area: (a) active area, (b) total chip area.

Our approach is able to capture a large amount of the regularity and results in significant reductions in bus and multiplexor power compared to both the Hyper and the FDS-VC schemes. The synthesis scheme is shown to have 47% and 35% average reduction in bus power with respect to the Hyper and the FDS-VC schemes, respectively, and about 49% average reductions in multiplexor power. Total power reduc-

1. The Synergy synthesis system incorporates the estimation and exploration environment explained in Chapters 4 and 5 and the two synthesis approaches developed.

2. Synergy allows the user to use either the locality-based synthesis scheme of Chapter 7 or the regularity-based synthesis scheme presented in this chapter.

tions of 28% and 17% were obtained on average with respect to the Hyper and FDS-VC schemes, respectively without any penalty in the overall area. The results show that there is a high potential for interconnect power improvements by exploiting the regularity inherent in the algorithm.

The effect of varying the coverage threshold on the quality of the results has been extensively studied. It was seen that different coverage thresholds are optimal for different examples. However, no automated method of determining it has been proposed. It would be useful to have an automated approach to predict a good threshold based on the properties of the algorithm at hand. Important parameters that may govern the optimal threshold are the overall regularity in the graph (total number of E-templates and there coverages) and the concurrency of the algorithm.

Also, the current implementation does not change the structure of the graph while exploiting regularity, which can potentially have a large impact on the amount of regularity available. A simple extension in this direction would be to allow permutations of inputs to commutative operations. Permuting inputs may uncover more regularity in the graph and lead to better results.

9 Conclusions

In this chapter, we summarize the work presented in this thesis and state the key contributions and conclusions. We also propose possible directions for future research.

9.1. Summary

This work has addressed various issues in low-power design. A top-down power optimization methodology is advocated and solution techniques targeting the algorithm and architecture abstraction levels are presented. The key contributions include extensive studies and models to enable power prediction at the algorithm-level, and specific synthesis techniques aimed at reducing interconnect power. The ideas presented in this thesis are implemented in a synthesis system called Synergy. Figure 9.1 overviews the system, presenting the key components and ideas.

One of the key components of Synergy is a high-level estimation tool based on a technology-targeted prediction approach. The tool exploits the knowledge of the architecture model and the hardware library along with the behavioral characteristics of the algorithm, to obtain meaningful power estimates with limited information. The different power consuming components of a chip are estimated separately using different techniques that address their specific peculiarities. The components fall into two categories based on their dependence on the algorithm — the algorithm-inherent part and the implementation overhead. The estimates use analytic techniques based on algorithm analysis as well as stochastic techniques based on data from past designs.

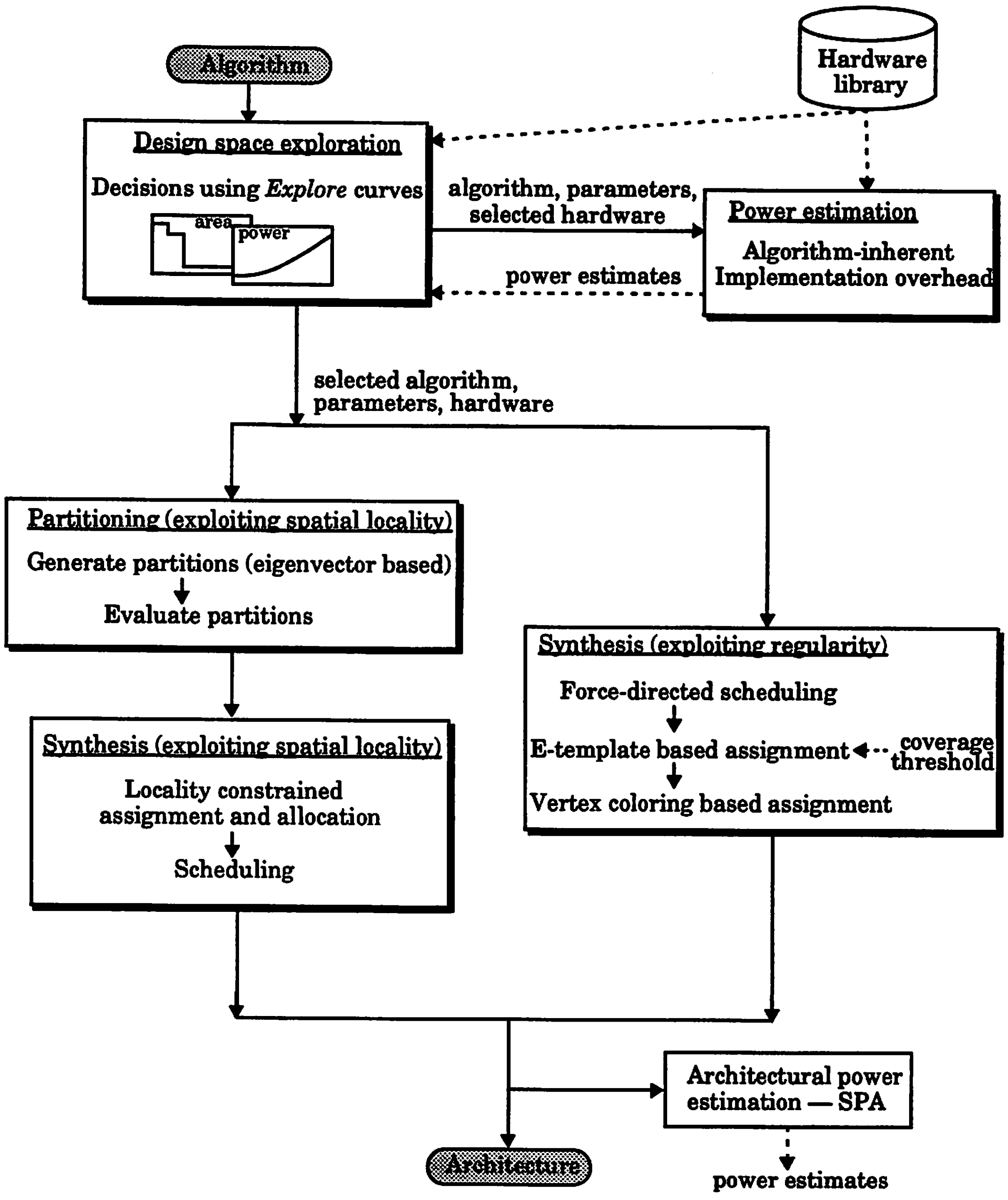

Algorithm
Hardware library
Design space exploration
Decisions using Explore curves
area
power
algorithm, parameters, selected hardware
Power estimation
Algorithm-inherent
Implementation overhead
power estimates
selected algorithm, parameters, hardware
Partitioning (exploiting spatial locality)
Generate partitions (eigenvector based)
Evaluate partitions
Synthesis (exploiting regularity)
Force-directed scheduling
E-template based assignment
coverage threshold
Vertex coloring based assignment
Synthesis (exploiting spatial locality)
Locality constrained assignment and allocation
Scheduling
Architectural power estimation — SPA
Architecture
power estimates

Figure 9.1. Synergy — the overall flow and key concepts.

The estimation tool is encapsulated into an exploration tool called Explore to facilitate a more complete search of the design space.

Another key component of the system is a set of synthesis techniques aimed at minimizing interconnect power. The interconnect is selected as the prime target for power optimization based on two observations obtained by comparing manual and automated designs — automated techniques produce designs that are power inefficient in the interconnect, and the interconnect power is greatly influenced by architecture-level decisions.

After the algorithm and its associated design parameters are selected, a partitioning scheme based on spectral techniques is used to identify spatial locality in the algorithm. The synthesis tasks that follow use this partitioning to generate a localized interconnect architecture that dissipates much less power in global interconnections.

The system also encompasses allocation, assignment, and scheduling algorithms that identify and exploit repeated patterns of computation in the graph to reduce the multiplexing overhead. The concept is termed regularity exploitation. It reduces accesses to multiplexors and tristate buffers as well as fan-outs and fan-ins of buses, decreasing the overall interconnect power.

9.2. Future directions

While the techniques and methodologies proposed in this dissertation have shown very positive results, the work in low-power design in general, and in high-level techniques for low power in particular, is far from complete. Though it is not possible to list all possible future directions in this area, we discuss some directions that are most promising and directly related to this work.

9.2.1. Power estimations

This thesis presents a technology-targeted approach to high-level power estimation, producing estimates for a targeted architecture and hardware library. A broad classification of components into algorithm-inherent and implementation overhead has been presented. Though the approach can be applied to any architecture or hardware library, the analysis in this thesis focuses on a single architecture model. The usefulness of the approach can be greatly enhanced by examining other architecture styles and verifying the validity of the proposed techniques for them.

Another dimension to be explored is the class of applications. While mainly datapath-intensive DSP applications are considered in this thesis, it would be interesting to apply these estimation techniques to more control-intensive applications.

An approach that was discussed very briefly in this thesis is the technology-independent estimation scheme. As discussed, ideas of using entropy [72, 88, 111] and structural properties of algorithms [41, 98] are definitely steps in this direction. Still, a large amount of work remains to be done to link these behavioral characteristics to concrete power numbers.

Recently, there has been increased interest in automated system-level techniques for dealing with the increasing complexity of electronic systems being designed. Estimating the power at this stage allows the user to appropriately budget the system power requirements and identify bottlenecks that need to be targeted for optimization. The spreadsheet-based power estimation approach presented in [68] addresses some of the issues.

9.2.2. Low-power synthesis

This dissertation has presented techniques for exploiting two algorithm properties for reducing interconnect power. Several other properties such as concurrency, connectivity, etc. may influence the power consumptions of some components, and are worth examining.

Though the locality and regularity properties have been studied here in the context of architecture synthesis from algorithm specifications, the concepts are general and may be applied more broadly. In fact, these concepts can be used in several other areas that have large components of their power consumption in the interconnect. Some good applications include field-programmable gate arrays, memories, and large blocks of control, where a large amount of power is consumed in the global interconnections.

Also, it is possible to use these concepts for system-level power optimization. Locality may be used to guide system-level partitioning either into different hardware blocks or into software and hardware components. This is especially useful since the arguments of greater power savings at higher abstraction levels hold as we move to the system level.

References

[1] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang, "Code generation using tree generation and dynamic programming," *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 3, pp. 491-516, June 1989.

[2] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Transactions on VLSI Systems*, Vol. 2, No. 4, pp. 426-436, Dec. 1994.

[3] C. J. Alpert and A. B. Kahng, "Geometric embeddings for faster and better multi-way netlist partitioning," *Proceedings of the Design Automation Conference*, pp. 743-748, June 1993.

[4] E. Avenhaus, "On the design of digital filters with coefficients of limited word length," *IEEE Transactions on Audio and Electroacoustics*, Vol. 20, pp. 206-212, Aug. 1972.

[5] F. Balasa, F. Catthoor, and H. De Man, "Dataflow-driven memory allocation for multi-dimensional signal processing systems," *Proceedings of the International Conference on Computer Aided Design*, pp. 518-521, Nov. 1994.

[6] J. Bardeen and W. Brattain, "The transistor, a semiconductor triode," *Physics Reviews*, Vol. 74, pp. 230, July 1948.

[7] E. R. Barnes, "An algorithm for partitioning the nodes of a graph," *Siam Journal of Algorithms and Discrete Methods*, Vol. 3, No. 4, pp. 541-549, 1994.

[8] L. Benini, M. Favalli, P. Olivio, and B. Ricco, "A novel approach to cost-effective estimate of power dissipation in CMOS IC's," *Proceedings of the European Design Automation Conference*, pp. 354-363, April 1993.

[9] O. Bentz, "A hardware mapper for the Hyper high-level synthesis system," *Master's Thesis*, University of California, Berkeley, Memorandum No. UCB/ERL 96/97, Dec 1993.

[10] G. Box, W. Hunter, and S. Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, John Wiley and Sons, New York, 1978.

[11] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: a multiple-level logic optimization system", *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. CAD-6, No. 6, pp. 1062-1081, Nov. 1987.

[12] R. W. Brodersen, ed. *Anatomy of a Silicon Compiler*, Kluwer Academic Publishers, Boston, 1992.

[13] R. Burch, F. N. Najm, P. Yang, and T.N. Trick, "A Monte Carlo approach for power estimation," *IEEE Transactions on VLSI Systems*, Vol. 1, No. 1, pp. 63-71, Mar. 1993.

[14] T. Burd, "Low-power CMOS library design methodology," *Master's thesis* University of California, Berkeley, Memorandum No. UCB/ERL M94/89, Nov. 1993.

[15] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low-power signal processing systems," *VLSI Signal Processing VII*, pp. 178-187, Oct. 1994.

[16] P. K. Chan, M. D. F. Schlag, and J. Zien, "Spectral K-way ratio-cut partitioning and clustering," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol 13, No. 9, pp. 1088-1096, Sept. 1994.

[17] A. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen, "Design of portable real-time DSP applications," *Proceedings of the Custom Integrated Circuits Conference*, pp. 259-266, May 1994.

[18] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 14, No. 1, pp. 12-31, Jan. 1995.

[19] J. M. Chang and M. Pedram, "Register allocation and binding for low power," *Proceedings of the Design Automation Conference*, pp. 29-35, June 1995.

[20] A. Chatterjee and R. Roy, "Synthesis of low power linear DSP circuits using activity metrics," *Proceedings of the International Conference of VLSI Design*, pp. 265-270, Jan. 1994.

[21] K. T. Cheng and V. D. Agarwal, "An entropy measure for the complexity of multi-output Boolean functions," *Proceedings of the Design Automation Conference*, pp. 302-305, June 1990.

[22] C. Chu, "Hardware mapping and module selection in the Hyper synthesis system," *Ph.D. Thesis*, University of California, Berkeley, Memorandum No. UCB/URL M92/46, May 1992.

[23] M. A. Cirit, "Estimating dynamic power consumption of CMOS circuits," *Proceedings of the International Conference on Computer Aided Design*, pp. 534-537, Nov. 1987.

[24] M. Corazao, M. Khalaf, L. M. Guerra, M. Potkonjak, and J. M. Rabaey, "Instruction set mapping for performance optimization," *Proceedings of the International Conference on Computer Aided Design*, pp. 518-521, Nov. 1993.

[25] R. Crochiere and A. Oppenheim, "Analysis of linear networks," *Proceedings of the IEEE*, Vol. 63, No. 4, pp. 581-595, April 1975.

[26] A-C. Deng, "Power analysis for CMOS/BiCMOS circuits," *Proceedings of the International Workshop on Low-Power Design*, pp. 3-8, April 1994.

[27] S. Devadas and S. Malik, "A survey of optimization techniques targeting low-power VLSI circuits," *Proceedings of the Design Automation Conference*, pp. 242-247, June 1995.

[28] W. E. Donath, "Logic partitioning," *Physical Design Automation of VLSI Systems*, Edited by B. Preas and M. Lorenzetti, Benjamin/Cummings, pp. 65-86, 1988.

[29] A. H. Farrahi, G. E. Tellez, and M. Sarrafzadeh, "Memory segmentation to exploit sleep mode operation," *Proceedings of the Design Automation Conference*, pp. 36-41, June 1995.

[30] M. Feuer, "Connectivity in random logic," *IEEE Transactions on Computers*, Vol. C-31, No. 1, pp. 29-33, Jan. 1982.

[31] C. M. Fiduccia and R. M. Matteyses, "A linear-time heuristic for improving network partitions," *Proceedings of the Design Automation Conference*, pp. 175-181, June 1982.

[32] J. Frankle and R. M. Karp, "Circuit placement and cost bounds by eigenvector decomposition," *Proceedings of the International Conference on Computer Aided Design*, pp. 414-417, Nov. 1986.

[33] A. El Gamal, "Two-dimensional stochastic model for interconnections in master slice integrated circuits," *IEEE Transactions on Circuits and Systems*, Vol. CAS-28, No. 2, pp. 127-38, Feb. 1981.

[34] D. D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Boston, Kluwer Academic Publishers, 1992.

[35] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Company, New York, 1979.

[36] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston 1992.

[37] W. Geurtz, "Synthesis of accelerator datapaths for high-throughput signal processing applications," *Ph. D. Thesis*, Katholieke Universiteit Leuven, Belgium, Mar. 1995.

[38] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," *Proceedings of the Design Automation Conference*, pp. 253-259, June 1992.

[39] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol 23, no. 1, pp. 5-48, Mar. 1991.

[40] L. Goodby, A. Orailoglu, and P. M. Chau, "Micro-architectural synthesis of performance constrained, low-power VLSI designs," *Proceedings of the International Conference on Computer Design*, pp. 323-326, Oct. 1994.

[41] L. Guerra, M. Potkonjak, and J. Rabaey, "System-level design guidance using algorithm properties", *VLSI Signal Processing VII*, pp. 73-82, Oct. 1994.

[42] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 11, No. 9, pp. 1074-1085, Sept. 1992.

[43] K. M. Hall, "An r-dimensional quadratic placement algorithm," *Management Science*, Vol. 17, No. 3, pp. 219-229, Nov. 1970.

[44] M. M. Halldorsson and J. Radhakrishnan, "Greed is good: approximating independent sets in sparse and bounded-degree graphs," *Proceedings of the Symposium on the Theory of Computing*, May 1994, pp. 439-448.

[45] W. Heller, W. F. Mikhail, and W. E. Donath, "Prediction of wiring space requirements for LSI," *Proceedings of the Design Automation Conference*, pp. 20-22, June 1977.

[46] B. Hendrickson and R. Leland, "The Chaco user's guide, V. 1.0," *Technical Report SAND93-2339, Sandia National Lab*, Oct. 1993.

[47] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quatitative Approach*, Morgan Kaufman Publishers Incorporated, San Francisco, Califiornia, 1996.

[48] P. Hilfinger, "A high-level language and silicon compiler for digital signal processing applications," *Proceedings of the Custom Integrated Circuits Conference*, pp. 213-216, May 1985.

[49] D. Hochbaum, "Efficient bounds for the stable set, vertex cover, and set packing problems", *Discrete Applied Mathematics*, Vol. 6, pp. 243-254, 1983.

[50] C. X. Huang, B. Zhang, A-C. Deng, and B. Swirski, "The design and implementation of PowerMill," *Proceedings of the International Symposium on Low-Power Design*, pp. 105-109, April 1995.

[51] S-H. Huang, J. M. Rabaey, "An integrated framework for optimizing transformations," *VLSI Signal Processing IX*, pp. 263-272, Oct. 1996.

[52] IEEE, "IEEE standard VHDL language reference manual," IEEE standard 1076-1987, New York, NY.

[53] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, New York, 1991.

[54] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, Vol. 49, pp. 291-307, Feb. 1970.

[55] K. Keutzer, "DAGON: technology binding and local optimization," *Proceedings of the Design Automation Conference*, pp. 341-347, Nov. 1987.

[56] K. Keutzer and P. Vanbekbergen, "Impact of CAD on the design of low power digital circuits," *Proceedings of the Symposium on Low Power Electronics*, pp. 42-45, Oct. 1994.

[57] S. Kirkpatrick, C. D. Gelatt, and M. P. Velatt, "Optimization by simulated annealing," *Science*, 220(4598), pp. 671-680, May 1983.

[58] T.H. Krodel, "PowerPlay — fast dynamic power dissipatiojn based on logic simulation," *Proceedings of the International Conference on Computer Design*, pp. 96-100, Oct. 1991.

[59] K. Kucukcakar and A. C. Parker, "Data path tradeoffs using MABAL," *Proceedings of the Design Automation Conference*, pp. 511-516, June 1990.

[60] F. J. Kurdahi and C. Ramachandran, "Evaluating layout area trade-offs for high level synthesis applications", *IEEE Transactions on VLSI systems*, Vol. 1, No. 1, pp. 46-55, Mar. 1993.

[61] E. Lagnese and D. E. Thomas, "Architectural partitioning for system level synthesis of integrated circuits," *IEEE Transactions on Computer Aided Design*, Vol. 10, No. 7, pp. 847-860, July 1991.

[62] B. Landman and R. Russo, "On a pin versus block relationship for partition of logic graphs," *IEEE Transactions on Computers*, Vol. C-20, No. 12, pp. 1469-1479, Dec. 1971.

[63] P. Landman, "High quality, low bit-rate speech coding for low-power VLSI implementation," *Masters thesis*, University of California, Berkeley, Memorandum No. UCB/ERL M94/47, May 1991.

[64] P. Landman, R. Mehra, and J. Rabaey "An integrated CAD environment for low power design," *IEEE Design and Test of Computers*, summer 1996.

[65] P. Landman and J. Rabaey, "Architectural power analysis: the dual bit type method," *IEEE Transactions on VLSI Systems*, Vol.3, No.2, pp. 173-87, June 1995.

[66] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, Vol. 36, No. 1, pp. 24-35, Jan. 1987.

[67] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Chichester, U.K.: Wiley-Teubner, 1990.

[68] D. Lidsky and J. Rabaey, "Early power exploration — a world wide web application," *Proceedings of the Design Automation Conference*, pp. 27-32, June 1996.

[69] D. Lidsky and J. Rabaey, "Low power design of memory intensive functions. Case study: vector quantization," VLSI Signal Processing VII, pp. 378-387, Oct. 1994.

[70] D. Liu and C. Svenson, "Power consumption estimation in CMOS VLSI chips," *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 6. pp. 663-670, June 1994.

[71] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Register-interconnect optimization in data path synthesis," *Microprocessing and Microprogramming*, Vol 33, No. 5, pp. 279-288, Aug. 1992.

[72] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic approaches of energy consumption at the register transfer level," *Proceedings of the International Symposium on Low Power Design,"* pp. 81-86, April 1995.

[73] R. Marculescu, D. Marculescu, and M. Pedram, "Switching activity analysis considering spatiotemporal correlations," *International Conference on Computer Aided Design*, pp 294-299, Nov. 1994.

[74] A. Masaki, "Possibilities of deep-submicrometer CMOS for very-high-speed computer logic," *Proceedings of the IEEE*, Vol. 81, No. 9, pp. 1311-1324, Sept. 1993.

[75] *MATLAB, quick reference*, MathWorks, Incorporated, Natick, MA, 1995.

[76] M. C. McFarland, "Re-evaluating the design space for register-transfer level hardware synthesis," *Proceedings of the International Conference on Computer Aided Design*, pp. 262-265, Nov. 1987.

[77] M. McFarland and T. Kowalski, "Incorporating bottom-up design into hardware synthesis," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 9, No. 9, pp. 938-949, Sept. 1990.

[78] R. Mehra, L. Guerra, and J. Rabaey, "Low power architectural synthesis and the impact of exploiting locality", *Journal of VLSI Signal Processing*, Vol. 13, No. 2-3, pp. 239-258, Aug.-Sept. 1996.

[79] R. Mehra, D. Lidsky, A. Abnous, P. Landman, and J. Rabaey, "Algorithm and architecture level methodologies", *Low Power Design Methodologies*, J. M. Rabaey and M. Pedram, Ed., Kluwer Academic Publishers, 1996.

[80] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," *Proceedings of the International Workshop on Low-Power Design*, pp. 197-202, April 1994.

[81] J. D. Meindl, "Low power microelectronics: retrospect and prospect," *Proceedings of the IEEE*, Vol. 83, No. 4, April 1995.

[82] J. Montanaro, R. Wilek, K. Anne, A. Black, E. Cooper, D. Dobberpuhl, P. Donahue, J. Eno, A. Farell, G. Hoeppner, D. Kruckemyer, T. Lee, P. Lin, L. Madden, D. Murray, M. Pearce, S. Santhanam, K. Snyder, R. Stephany, S. Theirauf, "A 160 MHz 32b 0.5W CMOS RISC microprocessor," *Digest of Technical Papers, International Solid State Circuits Conference*, pp. 214-215, Feb. 1996.

[83] E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," *Proceedings of the International Symposium on Low-Power Design*, pp. 99-104, April 1995.

[84] L. W. Nagel, "SPICE2: a computer program to simulate semiconductor circuits," *Technical Report ERL-M520*, University of California, Berkeley, May 1975.

[85] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Transactions on VLSI Systems*, Vol. 2, No. 4, pp. 446-455, Dec. 1994.

[86] F. N. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. 12, No. 2, pp. 310-323, Feb. 1993.

[87] F. Najm, R. Burch, P. Yang, and I. Hajj, "CREST — a current estimator for CMOS circuits," *Proceedings of the International Conference on Computer Aided Design*, pp. 204-207, Nov. 1988.

[88] M. Nemani and F. N. Najm, "Towards a high-level power estimation capability," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 15, No. 6, pp. 588-598, June 1996.

[89] R. Norman, J. Last, and I. Haas, "Solid-state micrologic elements," *Digest of Technical Papers, International Solid State Circuits Conference*, pp. 82-83, Feb. 1960.

[90] N. Park and F. J. Kurdahi, "Module assignment and interconnect sharing of pipelined datapaths," *Proceedings of the International Conference on Computer Aided Design*, pp. 16-19, Nov. 1989.

[91] P. G. Paulin and J. P. Knight, "Force-directed scheduling for behavioral synthesis of ASIC's," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 8, No. 6, pp. 661-679, June 1989.

[92] M. Potkonjak and J. M. Rabaey, "Exploring the algorithmic design space using Hyper," *VLSI Signal Processing VI*, pp. 123-131, Nov. 1993.

[93] M. Potkonjak and J. M. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," *Proceedings of the International Conference on Computer Aided Design*, pp. 304-308, Nov. 1992.

[94] M. Potkonjak and J. M. Rabaey, "Optimizing the resource utilization using transformations," *Proceedings of the International Conference on Computer Aided Design*, pp. 88-91, Nov. 1991.

[95] M. Potkonjak and J. M. Rabaey, "Scheduling algorithms for hierarchical data control flow graphs," *International Journal of Circuit Theory and Applications*, Vol. 20, No.3, pp. 217-33, May-June 1992.

[96] 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

[97] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test of Computers*, pp. 40-51, June 1991.

[98] J. M. Rabaey, L. Guerra, and R. Mehra "Design Guidance in the Power Dimension," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 2837-2840, May 1995.

[99] J. Rabaey and M. Pedram, Ed., *Low Power Design Methodologies*, Kluwer Academic Publishers, Boston, 1996.

[100] J. M. Rabaey and M. Potkonjak, "Complexity estimation for real-time application specific circuits," *Proceedings of the European Solid State Circuits Conference*, pp. 201-204, Sept. 1991.

[101] A. Raghunathan and N. K. Jha, "An iterative improvement algorithm for low power data path synthesis," *Proceedings of the International Conference on Computer Aided Design*, pp. 597-602, Nov. 1995.

[102] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low-power," *Proceedings of the International Conference on Computer Design*, pp. 318-322, Oct. 1994.

[103] J. V. Ranjan, "Automatic synthesis of microprocessors," *Ph.D. Thesis*, Carnegie Mellon University, Dec. 1988.

[104] D. S. Rao and F.J. Kurdahi, "Partitioning by regularity extraction", *Proceedings of the Design Automation Conference*, pp. 235-238, June 1992.

[105] D. S. Rao and F.J. Kurdahi, "An approach to scheduling and allocation using regularity extraction", *Proceedings of the European Design Automation Conference*, pp. 557-561, Feb. 1993.

[106] A. Salz and M. Horowitz, "IRSIM: An incremental MOS switch-level simulator," *Proceedings of the Design Automation Conference*, pp. 173-178, June 1989.

[107] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez, "Thermal management system for high performance PowerPC Microprocessors," *Proceedings of the COMPCON*, pp. 325-330, Feb. 1997.

[108] T. Sato, Y. Ootaguro, M. Nagamatsu, and H. Tago, "Evaluation of architecture-level power estimation for CMOS RISC processors," *Proceedings of the Symposium of Low-Power Electronics*, pp. 44-45, Oct. 1995.

[109] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," *Proceedings of the Design Automation Conference*, pp. 57-62, June 1972.

[110] Semiconductor Technology Association, "The national technology roadmap for semiconductors," San Jose, CA, 1995.

[111] N. R. Shanbag, "Lower bounds on power dissipation for DSP algorithms," *Proceedings of the International Symposium for Low Power Electronics and Design*, pp. 43-48, Aug. 1996.

[112] W. Schockley, "The theory of pn junctions in semiconductors and pn-junction transistors," *BSTJ*, Vol. 28, pp. 435, 1949.

[113] H. D. Simon, "Partitioning of unstructured problems for parallel processing," *Computing Systems in Engineering*, Vol. 2, No. 2/3, pp. 135-148, 1991.

[114] H. U. Simon, "On approximate solutions for combinatorial problems," *Siam Journal of Discrete Mathematics*, Vol. 3, No. 2, pp 294-310, May 1990.

[115] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, and T. Mozden, "Power conscious CAD tools and methodologies: a perspective," *Proceedings of the IEEE*, Vol. 83, No. 4, pp. 570-594, April 1995.

[116] D. Springer and D. E. Thomas, "New methods for coloring and clique partitioning in data path allocation," *Integration, The VLSI Journal*, Vol.12, No.3, pp. 267-292, Dec. 1991.

[117] G. Sorkin, "Asymptotically trivial global bus routing: a stochastic analysis," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. CAD-6, pp. 820-827, Sept. 1987.

[118] L. Stok, "Interconnect optimization for multiprocessor architectures," *Proceedings of the International Conference on Computer Systems and Software Engineering*, pp. 461-465, May 1990.

[119] C-L. Su and A. Despain, "Cache design trade-offs for power and performance optimization: a case study," *Proceedings of the International Symposium on Low-Power Design*, pp. 63-68, April 1995.

[120] M. van Swaaij, F. Franssen, F. Catthoor, and H. De Man, "Modeling data flow and control flow for high level memory management," *Proceedings of the European Conference on Design Automation*, pp. 8-13, Mar. 1992.

[121] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *Proceedings of the International Conference on Computer Aided Design*, pp. 384-390, Nov. 1994.

[122] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," *Proceedings of the International Symposium on Low Power Design*, pp. 221-226, April 1995.

[123] C-J Tseng and D. P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. CAD-5, No. 3, pp. 379-395, July 1986.

[124] E. Tsern and T. Meng, "A low-power video-rate pyramid VQ decoder," *Digest of Technical Papers, International Solid State Circuits Conference*, pp. 214-215, Feb. 1996.

[125] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, and A.M. Despain, "Power estimation methods for sequential logic circuits," *IEEE Transactions on VLSI* systems, Vol. 3, No. 3, pp. 404-416, Sept. 1995.

[126] J. Vanhoof, K. V. RomPaey, I. Bolsens, G. Goosens, and H. DeMan, *High-level Synthesis for Real-time Digital Signal Processing*, Kluwer Academic Publisher, Boston, 1993.

[127] 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, Aug. 1984.

[128] I. Verbauwhede, C. Scheers, and J. M. Rabaey, "Analysis of multidimensional DSP specifications," *IEEE Transactions on Signal Processing*, Vol.44, No.12, pp. 3169-3174, Dec. 1996.

[129] W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, J. L. van Meerbergen, and A. van der Werf, "Improved force-directed scheduling in high throughput digital signal processing," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 14, No. 8, pp. 945-960, Aug. 1995.

[130] G. Vijayan, "Partitioning logic on graph structures to minimize routing cost," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, pp. 1326-1334, Dec. 1990.

[131] R. A. Walker and R. Camposano, *A Survey of High Level Synthesis Systems*, Boston, Kluwer Academic Publishers, Boston, 1991.

[132] S. Wolfram, *The Mathematica Book*, Cambridge University Press, 1996

[133] S. Wu, "A hardware library representation for the Hyper synthesis system," *Masters' Thesis*, University of California, Berkeley, Memorandum No. UCB/ERL M94/47, June 1994.

[134] Y. C. Wei and C. K. Cheng, "Ratio cut partitioning for hierarchical designs," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, Vol. 10, pp. 911-921, July 1991.

[135] M. E. Wolf and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471, Oct. 1991.

[136] A. Yeung and J. Rabaey, "A 2.4 GOPS data-driven reconfigurable multiprocessor IC for DSP," *Digest of Technical Papers, International Solid State Circuits Conference*, pp. 108-109, Feb. 1995.

[137] G. Zimmerman, "A new area and shape function estimation technique for VLSI layouts," *Proceedings of the Design Automation Conference*, pp. 60-65, June 1988.