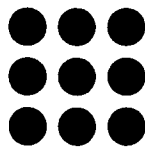

HIGH-LEVEL AND LOGIC SYNTHESIS TECHNIQUES FOR LOW POWER

Enric Musoll Cinca

UPC. Universitat Politècnica de Catalunya

Barcelona (Spain). July, 1996



UPC

A THESIS SUBMITTED IN FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
Doctor en Informàtica

Signatures

To all my family.

ACKNOWLEDGEMENTS

First of all, I would like to stress that this work would not have been possible without my teacher and now advisor, Professor Jordi Cortadella. Although a fearless and full-time devoted leader to the design of asynchronous circuits, he has always been available for help, always putting me on the right track, always seeing the bright side of my problems.

I would like to thank Enric Pastor, Fermín Sánchez, Oriol Roig, Marco A. Peña and Gianluca Cornetta, all members of the CAD-VLSI group where I have been working these last years, for their help and support. I would like to thank also my colleagues at the Computer Architecture Department.

Special thanks to Rosa M. Badía and Tomás Lang, who have reviewed some parts of this work, and to Joan Figueras. Their technical discussions and suggestions have definitely improved the quality of this work. To all of them, my sincere thanks.

This work has been supported in part by the *Ministerio de Educación y Ciencia* of Spain under contracts CYCIT TIC94-0531-E and TIC95-0419. My bank account is definitely indebted to the *Departament d'Ensenyament de la Generalitat de Catalunya* for their help financing my doctorate and recognizes the *Amics de Gaspar de Portolà* association for providing the funds to present part of this work in a conference in California.

I would like to acknowledge the secretary staff of the Computer Architecture Department for coping so well with the administrative paperwork. My sincere thanks also to the systems group people, who have perfectly managed the advanced computing resources provided by the Computer Architecture Department.

During the last years, this work has absorbed all my time, even the so-called free time. I would like to thank the people who have tried, most of the times unsuccessfully, to put for a while my eyes out of the books and my hands out of the keyboard. Special thanks to my best friends and the UPC Rowing team for this work.

Last, but not least, thanks to my parents, Pilar and Pere, and my brother Ramon, for their unconditional patience and support. Thank you all.

CONTENTS

ACKNOWLEDGEMENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES	xiii
1 INTRODUCTION	1
1.1 Why low-power design techniques?	2
1.1.1 Reliability	2
1.1.2 Chip package	2
1.1.3 Batteries	2
1.2 Applications	4
1.3 Low-power design approach	5
1.3.1 Overall power reduction	6
1.4 Research objectives	7
1.5 Overview	8
2 STATE-OF-THE-ART IN LOW-POWER DESIGN	9
2.1 Power-consumption sources	10
2.1.1 Static dissipation	10
2.1.2 Dynamic dissipation	11
2.1.3 Defining power consumption	13
2.2 System layer	14
2.2.1 System shutdown	14
2.2.2 System partitioning	15
2.2.3 Algorithm selection	15
2.3 Architecture layer	16
2.3.1 Parallel and pipelined processing	17
2.3.2 Compiler transformations	20

2.3.3	Cache design	22
2.3.4	Data representation	22
2.4	Logic layer	24
2.4.1	Combinational circuits	24
2.4.2	Sequential circuits	29
2.5	Circuit layer	33
2.5.1	Logic style	33
2.5.2	Synchronous vs. asynchronous design	35
2.5.3	Design style	37
2.5.4	Adiabatic computing	38
2.6	Layout layer	39
2.7	Technology layer	41
2.7.1	Technology scaling and SOI process	41
2.7.2	Packaging technology	42
2.8	Low-power techniques summary	42
2.9	Conclusions	43
3	HIGH-LEVEL SYNTHESIS TECHNIQUES FOR LOW POWER	
		45
3.1	Introduction	46
3.2	Previous work	47
3.2.1	Estimating power consumption	48
3.2.2	Reducing power consumption	49
3.3	Power-consumption model of functional units	51
3.4	Register-transfer level techniques for low power	55
3.4.1	Loop interchange	56
3.4.2	Operand reordering	61
3.4.3	Operand sharing	66
3.4.4	Operand retaining	70
3.4.5	Operand similarity	72
3.4.6	Summary of targeted circuit domains	75
3.5	Scheduling and register binding for low power	76
3.5.1	High-level synthesis	76
3.6	Previous work on high-level synthesis for low power	78
3.6.1	Overview	80

3.6.2	Scheduling for low power	81
3.6.3	Register binding for low power	85
3.7	Conclusions	89
4	ARCHITECTURAL TECHNIQUE FOR LOW POWER	91
4.1	Introduction	92
4.2	Parallel multipliers	93
4.3	Previous Work	94
4.4	Potential power reduction in array multipliers	95
4.5	Transition-Retaining Barriers	98
4.6	Implementation of the TRB	100
4.7	Results	102
4.8	Conclusions	104
5	LOGIC/CIRCUIT TECHNIQUE FOR LOW POWER	107
5.1	Introduction	108
5.1.1	Motivation examples	108
5.2	Previous work and overview	112
5.2.1	Overview of our approach	113
5.3	Power consumption of CMOS gates	115
5.3.1	Definitions and overview	115
5.3.2	Transition density	116
5.3.3	Extended power-consumption model	116
5.4	Power-optimization algorithm	121
5.4.1	Algorithm overview	122
5.4.2	Monotonic characteristic	122
5.4.3	Equilibrium probabilities	123
5.4.4	Exhaustive exploration of gate configurations	123
5.5	Results	126
5.5.1	Scenarios for the experiments	126
5.5.2	Discussion	127
5.6	Conclusions	128
6	CONCLUSIONS AND FUTURE RESEARCH	131
6.1	Introduction	132
6.2	Contributions	133

6.2.1 System layer	134
6.2.2 Architecture layer	135
6.2.3 Logic/circuit layer	136
6.3 Future research	138
A CIRCUIT DESIGN AND ANALYSIS TOOLS	141
A.1 Introduction	142
A.2 Ocean	142
A.3 SLS	145
A.4 SIS	146
REFERENCES	149

LIST OF FIGURES

Chapter 1

1.1	State-of-the-art in battery technology ([Pow95]).	3
1.2	Power-consumption trend ([Mat96]).	3
1.3	Different layers in the design flow of a circuit.	6
1.4	Power breakdown in the ThinkPad notebook ([Ike95]).	7

Chapter 2

2.1	Static CMOS inverter: schematic, transistor and switch representations.	10
2.2	Short-circuit power dissipation.	11
2.3	(a) Input waveform; (b) charging and (c) discharging the capacitance C_{load} .	12
2.4	(a) Data-path circuit at 5V; (b) Parallel and (c) pipelined versions at 2.9V. ([BCS92])	17
2.5	Implementing polynomial $X^3 + AX^2 + BX + C$ (a) straightforward and (b) Horner's scheme.	20
2.6	Substituting one multiplication by one addition.	21
2.7	(a) Unbalanced and (b) balanced data-flow graph of the addition of four operands.	21
2.8	Delaying signals to eliminate hazards.	25
2.9	Different subject graphs implementing a 4-input NAND function.	28
2.10	(a) Subject graph; (b) available gates; (c) low-area mapping; (d) low-power mapping.	28
2.11	(a) FSM example; (b) two different codings with different switching activity among them ([RP93]).	29
2.12	Sequential circuit (a) before and (b) after re-timing.	30
2.13	Comparator circuit (a) without and (b) with disabled input registers.	32

2.14	(a) STG and (b) FSM with the registers enabled/disabled by signal f_a .	32
2.15	(a) Static and (b) dynamic implementations of the boolean function $Y = \overline{(A_1 + A_2)}(B_1 + B_2)$.	34
2.16	(a) Original synchronous system; (b) asynchronous system with adaptive scaling of the supply voltage ([NS94]).	36
2.17	Design time vs. power consumption of some design styles.	37
2.18	Clock tree driving schemes: (a) buffers at the clock source and (b) distributed buffers.	40

Chapter 3

3.1	Activity of a 16-bit data stream for different temporal correlations (data approximated from [LR94a]).	48
3.2	(a) Average activity in a multiplier as a function of the constant value (data approximated from [CR94]). (b) A parallel and serial implementations of an adder tree.	49
3.3	Transformations for reducing (a) activity at the address bus and (a) number of memory references.	50
3.4	(a) Coarse-grained and (b) fine-grained power-consumption model for an 8×8 -bit Booth multiplier.	52
3.5	(a) Data-flow graph; (b) and (c) are two possible schedules and functional-unit bindings.	54
3.6	The input data order in (b.1) leads to a higher operand AHD than in (b.2). Big arrows indicate the input data. Thin arrows indicate the AHD between two consecutive input data.	55
3.7	Example of application of the loop-interchange technique.	57
3.8	(a) Motion estimation algorithm and (b) motion-estimation algorithm with two loop interchanges.	58
3.9	Data-path for executing (a) the motion-estimation algorithm and (b) the matrix-vector product algorithm.	59
3.10	Matrix-vector product algorithm when referencing matrix A (a) by rows and (b) by columns.	60
3.11	(a) MAC operator for $p = 4$ and (b) DFG of the 4th-order LMS adaptive filter.	62
3.12	Different input-reusing graphs for: (a) the MAC operator and (b) the symmetric matrix-vector product operator.	65
3.13	(a) DFG of the AR filter; (b,c) two possible schedule and bindings.	67

3.14	(a) Original code and (b) after loop unrolling.	68
3.15	(a) Low-pass image filter algorithm; (b) DFG of the inner loop of (a) (without division by nine) and (c) DFG after loop unrolling.	70
3.16	(a) DFG of the 4th-order FIR filter; (b,c) two possible schedule and bindings.	74
3.17	DFG of the Differential Equation Solver.	75
3.18	High-level synthesis process.	76
3.19	Example of peak-power and average-power optimization during the scheduling and functional-unit binding tasks; (a) DFG; (b) adder characteristics; (c-e) different schedules and bindings and (d) results.	78
3.20	Register binding for low power.	80
3.21	List-scheduling algorithm.	82
3.22	Low-power list-scheduling algorithm.	83
3.23	(a) DFG of the LMS filter and (b) schedule and FU binding.	86
3.24	Register-binding algorithm for low power.	87
Chapter 4		
4.1	(a) Circuit structure with glitching; (b) block C presents more activity than block A.	92
4.2	5×5-bit array multiplier.	96
4.3	(a) FA and (b) HA structure.	96
4.4	Signal transitions per operation for each basic block in a 16×16-bit array multiplier.	97
4.5	Useful and total number of signal transitions for an 8×8, 16×16, 32×32 and 64×64-bit array multiplier.	98
4.6	(a) Possible locations for one TRB and (b) transitions per input vector for different locations of one TRB in a 16×16-bit array.	99
4.7	Signal transitions per input vector in a 16×16-bit array multiplier. (a) General view of the design with no TRBs and side view with (b) no TRBs, (c) one TRB and (d) two TRBs.	100
4.8	(a) Comparison between Static CMOS and C ² MOS and (b) avoiding direct-path currents.	101
4.9	4×4-bit array with two TRBs. The four types of delay-cells (TA-TD) are shown.	102
4.10	Design of a FA.	102

4.11	Layout of an 8×8 -bit array multiplier in a sea-of-gates design style.	103
------	---	-----

Chapter 5

5.1	(a) Four implementations of function $y = \overline{(a1 + a2)} b$ and (b) relative power consumption for two different input activity scenarios.	109
5.2	SPICE simulations of configurations (A) and (D) in Figure 5.1 for two different input activity scenarios.	110
5.3	Transistor-reordering technique in a 3-input NAND gate (example from [SLW95]).	111
5.4	(a) Static CMOS 3-input NAND gate (b) relative power consumption for different input activities (without varying the equilibrium probabilities of the inputs).	114
5.5	(a) Static CMOS gate representation and (b) algorithm to obtain function H_{nk} .	117
5.6	Two BDD representations of function $f = abc + \bar{b}d + \bar{c}d$ using different variable order.	120
5.7	Capacitances in a MOS transistor.	121
5.8	Optimization algorithm.	122
5.9	Exhaustive exploration algorithm.	124
5.10	Execution example of the exhaustive exploration algorithm.	125
5.11	A gate ($y = \overline{ad + be + bcd + ace}$) without a series-parallel representation.	126
5.12	The two scenarios considered.	127

Chapter 6

Appendix A

A.1	(a) Schematic and (b) gate description of a full adder.	142
A.2	Sea-of-gates layout architecture.	143
A.3	(a) Full-adder netlist description and (b) layout produced by Ocean.	144
A.4	(a) Command file and (b) graphical output of the SLS simulator.	145
A.5	(a) Library and (b) circuit description in SIS.	147

LIST OF TABLES

Chapter 1

1.1	Power dissipation of high-performance microprocessors ([BE95]).	5
1.2	Power dissipation of high-performance low-power microprocessors (updated from [BE95]).	5
1.3	Power breakdown for different processors ([Keu94]).	7

Chapter 2

2.1	Switching power in different types of circuits [Tiw94].	13
2.2	Number of multiplications and additions of the 8×8 -matrix DCT execution for different algorithms [BCS92].	16
2.3	Normalized area and power for the different architecture designs of the dat-apath example in Figure 2.4 ([BCS92])	19
2.4	Two's complement and sign-magnitude representation for the values -3 to 3 using 8 bits.	23
2.5	Area/delay/power/design simplicity comparison of some logic styles.	33
2.6	Technology scale factors for some parameters.	41
2.7	Power reductions obtained by different techniques.	43

Chapter 3

3.1	Factors of the coarse-grained model.	53
3.2	Two different input reorderings for the 4-input MAC unit. M_j represent the multiplications of the MAC unit.	63
3.3	Results obtained by applying the operand-sharing technique.	69
3.4	Idle time spent by the functional units.	71
3.5	Features of the targeted circuit domains.	75
3.6	Results obtained with the LPLS algorithm.	85
3.7	Comparison between the traditional resource-binding algorithm (TRB) and its low-power version (LPRB).	88

Chapter 4

- 4.1 Comparison between the original design and the new designs with TRBs. 104

Chapter 5

- 5.1 SPICE and SLS comparison for the configurations in Figure 5.2. Numbers in parenthesis show the power-consumption ranking. 110
- 5.2 Number of different configurations for some standard gates obtained by reordering its transistors. 124
- 5.3 Results obtained for several MCNC benchmarks for both scenarios considered. 129

Chapter 6**Appendix A**

INTRODUCTION

Historically, power consumption has not been a priority issue in the semiconductor industry. Until recently, the two leading parameters that a designer had to take into account when designing a chip were area and delay.

The area of a circuit has become smaller with the increasing levels of integration and the performance has improved with higher clock frequencies. By the year 2001, the frequency and integration density is expected to increase by a factor of 2 and 2.5 respectively with respect to 1995.

One of the consequences of these high levels of integration and performance is the increase in power consumption. Nowadays, and for certain applications, the power consumption is a design parameter as important as area or delay. So far the designer had to find the best design in a two-dimensional space (area/delay). Now, there is a three-dimensional space (area/delay/power) to dive in, with the corresponding increase in complexity.

This chapter presents a motivation for low-power design and the research objectives of this work.

1.1 WHY LOW-POWER DESIGN TECHNIQUES?

Several factors have pushed the concern for low-power as a critical design constraint. The three most important are: reliability, chip package cost and battery life in portable systems.

1.1.1 Reliability

The power dissipated by a circuit turns into heat. The excessive heat may cause a decrease of the circuit's reliability, i.e., failure mechanisms such as silicon interconnect fatigue, package related failure, electrical parameter shift, electro-migration, junction fatigue, etc. may occur with the generation of on-chip high-temperature. In fact, every 10°C increase in the on-chip temperature doubles the failure's rate [Sma94].

1.1.2 Chip package

The dissipated heat in chips causes a cost increase in packaging. Packages are characterized by the thermal resistance (in $\Delta^{\circ}\text{C}$ per watt), which means that one watt will raise the temperature by $\Delta^{\circ}\text{C}$. For chips consuming a few watts a plastic package ($40\text{-}50^{\circ}\text{C}/\text{watt}$) may be used. For more power hungry chips, a ceramic package ($15\text{-}30^{\circ}\text{C}/\text{watt}$) is needed [WE93]. This may increase the overall cost of the chip in US\$ 5-10. Moreover, high-cost packaging might include forced air or liquid cooling through tiny ducts, increasing even more the cost of the final chip.

1.1.3 Batteries

Portable battery-driven applications such as notebook and laptop computers represent the fastest growing segment of the computer industry and the driving factor for low-power design concern. In the Figure 1.1 we observe the state-of-the-art in batteries [Pow95]. For example, Li ion batteries currently present 60 Wh/Kg . Li ion technology will probably improve a 25% over the next 10 years [War95]. These improvements can not meet the high increase in power consumption of the circuits (a factor of four every three years as shown in Figure 1.2).

For example, consider the hypothetical case of a portable application based on a DEC Alpha processor (30-50 Watts). Using NiCd batteries, we would need roughly 0.6 Kg of batteries per hour of operation. The problem is not solved by using more powerful batteries (with Li ion technology, we reduce the weight to 0.4 Kg, still too

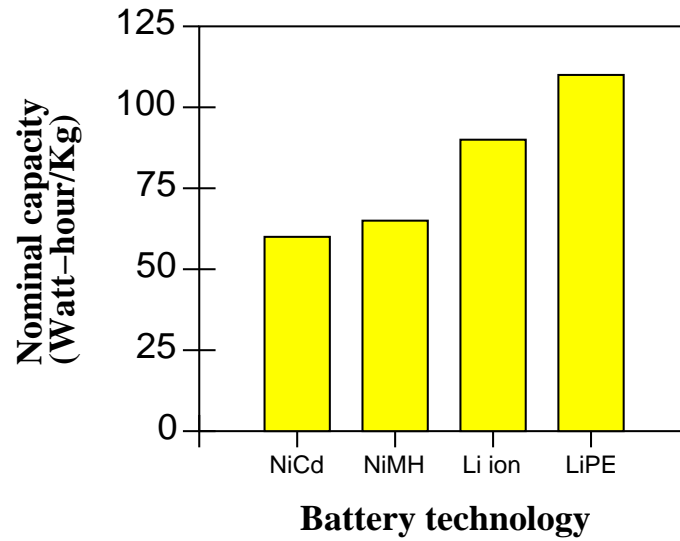


Figure 1.1 State-of-the-art in battery technology ([Pow95]).

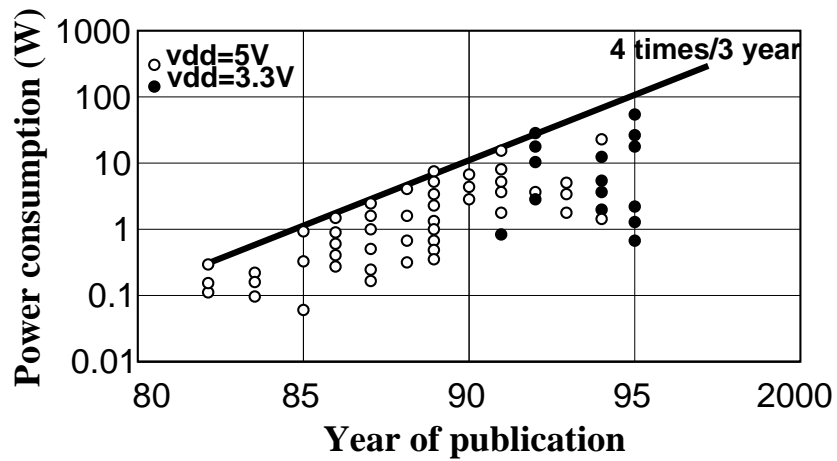


Figure 1.2 Power-consumption trend ([Mat96]).

heavy). The solution relies on applying also design techniques for low power during the design of the processor.

1.2 APPLICATIONS

Low-power design techniques are suitable for portable applications, where the life of the battery is a primary design constraint, and high-performance computing, where the power has increased and is increasing at a high pace.

Battery-operated applications

Battery-operated applications clearly benefit the most from the low-power design techniques. Examples of these applications may be found in different fields:

- computing: notebook and laptop computers.
- personal communications: the current generation of digital cellular phones and the Personal Digital Assistants (pocket-sized devices with multimedia access supporting full motion digital video and control via hand-write or speech recognition [CSB90]).
- consumer electronics: wrist watches (targeted long ago for low power), digital TV, pocket calculators.
- medicine: hearing aids, implantable cardiac equipments.
- military: night visors, surveillance systems.

Many of the above applications, specially those related to personal communications, demand the same computation capabilities as found in desktop machines since they execute complex speech and data compression algorithms.

High-performance computing

High-performance systems may also benefit from low-power design techniques. With large integration density and improved speed of operation, processors dissipating 30 to 50 Watts are emerging.

Table 1.1 shows some high-performance processors with their associated power consumption. We observe that, for example, the DEC Alpha 21064 dissipates 30W. With the integration level expected by the year 2000, several DEC Alphas may be integrated in just one chip, but the power dissipated will certainly have to be reduced.

Therefore, low-power design techniques are needed to implement either new versions of these processors or completely new designs that consume less. Some examples

Processor	Clock (MHz)	Technology (μm)	V_{dd} (V)	Power Peak (W)	Ref.
Intel Pentium	66	0.80	5.0	16	[Rep93]
DEC Alpha 21064	200	0.75	3.3	30	[ea92]
DEC Alpha 21164	300	0.50	3.3	50	[ea95b]
Power PC 620	133	0.50	3.3	30	[ea95a]
MIPS R10000	200	0.50	3.3	30	[MIP94]
UltraSparc	167	0.45	3.3	30	[ea95a]

Table 1.1 Power dissipation of high-performance microprocessors ([BE95]).

Processor	Clock (MHz)	Technology (μm)	V_{dd} (V)	Power Peak (W)	Ref.
Power PC 603	80	0.5	3.3	2.2	[ea94a]
IBM 486SLC2	66	0.8	3.3	1.8	[ea94b]
MIPS R4200	80	0.64	3.3	1.8	[YSS ⁺ 94]
ARM 710	33	0.8	5	0.5	[Gwe93a]
AT&T Hobbit	20	0.9	3.3	0.25	[Gwe93b]

Table 1.2 Power dissipation of high-performance low-power microprocessors (updated from [BE95]).

are shown in Table 1.2. For instance, the Power PC 603 is the low-power version of the Power PC 620. The frequency has been reduced to 80 MHz. This frequency reduction reduces power by itself, but the great amount of reduction obtained with the Power PC 603 processor (more than a factor of 10) has been obtained applying low-power techniques during its design.

1.3 LOW-POWER DESIGN APPROACH

The power dissipated by a circuit may be tackled at all the different layers of the design process: from the system layer, where the specifications of the circuit are defined, till the layout and technology layers (see Figure 1.3).

In general, larger power reductions are obtained in the early layers of the design process, i.e. in the architecture and system layers. But the low-power techniques in each layer are orthogonal among them. For example, the power savings obtained

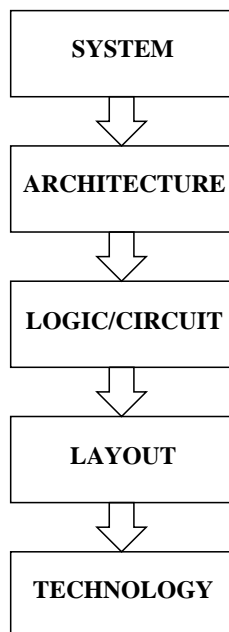


Figure 1.3 Different layers in the design flow of a circuit.

at the logic/circuit layer will add up to those already obtained at the architecture layer.

Therefore, it is important to find low-power techniques at each of the layers of the design process of a circuit.

1.3.1 Overall power reduction

An application is not only composed of digital circuits. There are other components that dissipate power. For example, Figure 1.4 shows the power distribution within the ThinkPad notebook [Ike95]. The processor unit accounts for the 40% of the total power budget. In a portable communications terminal, the processing units may account for the 50% of the power [CSB92b].

Moreover, power may be further broken down in the processor unit as shown in Table 1.3 for some processors. We observe that, for example, the logic power component of an *average* processor is 27%. Assuming that this processor is integrated in the ThinkPad notebook¹, the complete elimination on the logic power only draws an approximate 10% overall power reduction. Therefore, efforts should be aimed at

¹In fact, it is a Pentium 75.

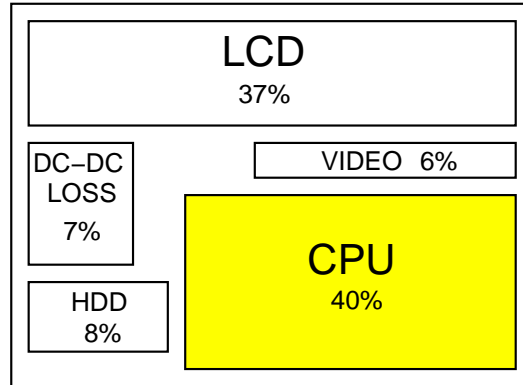


Figure 1.4 Power breakdown in the ThinkPad notebook ([Ike95]).

Processor	Logic	Clock	Memories	Interface
DEC Alpha	30%	50%	10%	10%
DEC NVAX	30%	40%	20%	10%
MIPS-X	25%	30%	35%	10%
TORCH	20%	25%	40%	10%
<i>average</i>	27%	36%	27%	10%

Table 1.3 Power breakdown for different processors ([Keu94]).

reducing the power consumption of those components that have the highest impact on reducing the overall power consumption.

1.4 RESEARCH OBJECTIVES

The goal of this work is to contribute to the actual state of low-power design with techniques at the higher layers of the design process, namely, the logic/circuit, architecture and system layers. These techniques will mainly focus on data-path architectures, such as those used in high-performance, real-time systems in telecommunications, speech, video and image processing. A proper power-consumption model for the system layer will be proposed to evaluate the techniques described. Some of these techniques will be automated to study the area/delay/power trade-off. In these cases, the power savings obtained will be evaluated with existent circuit simulators.

1.5 OVERVIEW

This work is divided into 6 chapters and one appendix. This chapter has presented an introduction to the growing concern in system design houses related to the power consumption of integrated circuits.

Chapter 2 briefly shows the state-of-the-art in low-power design: some of the existing techniques for low power will be outlined for each layer of the design process. The goal of this chapter is not to provide a thorough description of these techniques, but rather to give a flavour of the state-of-the-art in low-power design.

The remainder of the chapters are devoted to the contribution of this work. In Chapter 3, some techniques for reducing power during the high-level synthesis process are presented. These techniques rely on reducing the activity of the circuit, i.e. reducing the number of transitions at its internal signals. To evaluate the power savings obtained with these techniques, a proper power-consumption model is presented for the functional units. In the same chapter, some of the techniques are automated. The algorithms are presented and the area/delay/power trade-off is evaluated.

In Chapter 4, a technique at the architecture layer that reduces the generation and propagation of the unnecessary activity of the circuit is described. This technique is specially appropriated for array multiplier circuits. The power reductions for some of these circuits are evaluated and again the area/delay/power trade-off is studied.

The effect of the transistor-reordering technique in power consumption is studied in Chapter 5. A power-consumption model for a static CMOS gate that takes into account its internal capacitances is presented. An optimization algorithm that finds the best order of the transistors for each gate of the circuit is described. This algorithm relies on the power-consumption model of the CMOS gate. The power reductions obtained with this technique and the increase in delay are evaluated.

Finally, Chapter 6 concludes this work summarizing the main conclusions and indicating future research work in the field of low-power design.

In appendix A, the tools used to design and evaluate the circuits in this work are briefly reviewed.

STATE-OF-THE-ART IN LOW-POWER DESIGN

This chapter presents an overview of the state-of-the-art in low-power design. The approach described in the last chapter will be followed: some of the existing techniques for low power will be outlined for each layer of the design process. A more thorough insight will be given for the logic, circuit, architecture and system layers since this work focus on these steps of the design process.

Some of the techniques have been classified in different layers by other authors. It is specially unclear the boundary between system and architecture layers. Clearly, decisions taken at the system layer affect the architectural considerations. Moreover, some authors further split or merge some of the layers. Nevertheless, the classification presented in this chapter agrees with the majority of classifications done by other authors.

During optimization for low power, it is necessary to know the power distribution within a circuit. Therefore, the first section of this chapter is devoted to analyze the power-consumption sources in digital circuits.

Sections 2.2 to 2.7 describe some of the existing techniques for low-power design for each layer of the design process. Section 2.8 presents a summary of the techniques described in the previous sections. Section 2.9 states the conclusions of this chapter.

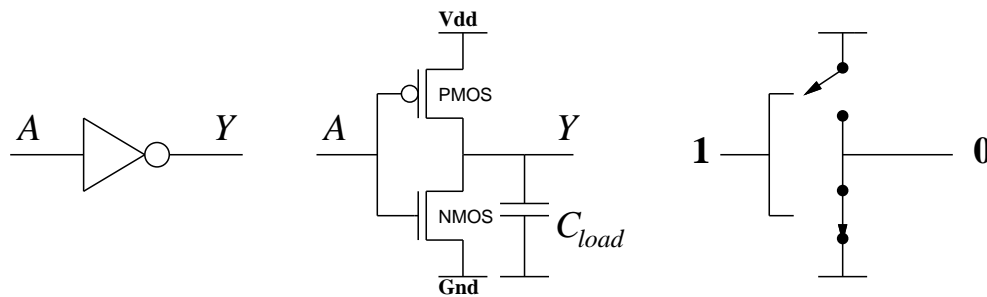


Figure 2.1 Static CMOS inverter: schematic, transistor and switch representations.

2.1 POWER-CONSUMPTION SOURCES

To minimize the power consumption (measured in Watts) of a CMOS circuit, the different power components and their effect must be identified. The power dissipated in a CMOS circuit is divided into two components [WE93]:

- static dissipation caused by leakage and other static currents.
- dynamic dissipation caused by the switching transient current (or short-circuit current) and the charging and discharging of load capacitances.

To describe these power-consumption sources, a static CMOS inverter will be used as example. Figure 2.1 shows its schematic, transistor and switch representations.

2.1.1 Static dissipation

CMOS circuits ideally present no static power dissipation because there is no direct path from power supply to ground. In practice, however, transistors behave as non-perfect switches, thus allowing the generation of some leakage currents that compose the static component of CMOS power dissipation.

The static dissipation is divided into two components: the leakage current, caused by the parasitic diodes, and the static current which is a function of the input voltage and the threshold voltage of the transistors.

The contribution of the static dissipation is less than 2% of the total power. But this contribution increases with the scaling of the technology. An important prerequisite of scaling down the device dimensions is the reduction of power supply because the

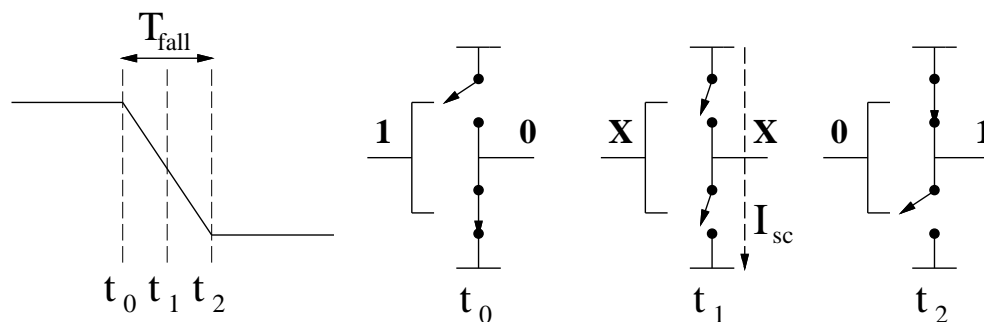


Figure 2.2 Short-circuit power dissipation.

higher voltage undermine device reliability. Reducing the power supply leads to the reduction of the threshold voltage of the transistors, implying an increase of the static dissipation [Mei95].

2.1.2 Dynamic dissipation

This component is caused by the short-circuit current and the current used for charging and discharging of load capacitances.

Short-circuit power dissipation

Since the transistors do not behave as ideal switches, a short-circuit current arises when the input changes its value as shown in Figure 2.2. At time t_0 , the input to the inverter is stable at 1 and the power dissipation (static) is negligible. At time t_1 , the input is changing its value and, therefore, it takes an undefined value (X). This value causes both transistors to be turned on and, therefore, a current (I_{sc}) is generated from power supply to ground. At time t_2 the input is again stable.

The power consumption produced by current I_{sc} depends on the time that both transistors are on. Therefore, if the input changes slowly, the power increases. With careful design for balanced input and output rise times this power component can be kept below 10-15% of the total power [Vee84]. Moreover, this component highly depends on the circuit type, as will be shown in the next section.

Charging/discharging capacitances

The power component resulting from the charging and discharging of parasitic capacitances (switching power) in the circuit dominates the total power consumption.

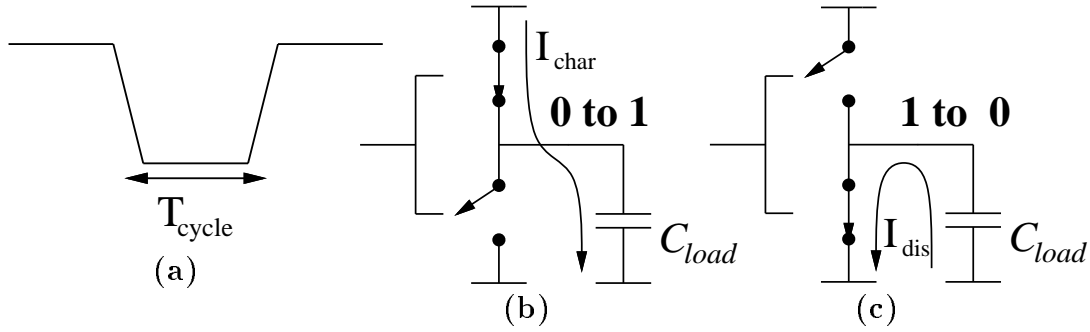


Figure 2.3 (a) Input waveform; (b) charging and (c) discharging the capacitance C_{load} .

The situation is illustrated in Figure 2.3, where the capacitance C_{load} represents all the different parasitic capacitances.

Let us consider the case of one complete cycle of operation with two transitions at the input signal (Figure 2.3(a)). When the input signal changes from 1 to 0, a current (I_{char}) is generated and the capacitance C_{load} is charged (Figure 2.3(b)). It can be easily derived that the energy (measured in Joules) delivered by the power supply is:

$$E = C_{\text{load}} V_{DD}^2 ,$$

where V_{DD} is the power supply.

Half of this energy is dissipated through the PMOS transistor as a result of the flow of current I_{char} . The other half is stored in the capacitance C_{load} . When the input signal changes to 1, a current (I_{dis}) is generated and the capacitance C_{load} is discharged. The energy stored in C_{load} during the previous input transition is now dissipated through the NMOS transistor; the power supply does not deliver new energy. Therefore, the average energy of a single output transition is:

$$E_{\text{avg}} = \frac{1}{2} (C_{\text{load}} V_{DD}^2) .$$

Assuming that the output of the inverter changes every cycle, the average power consumption produced by the charge and discharge of parasitic capacitances is:

$$W = \frac{1}{2} (C_{\text{load}} V_{DD}^2 f) ,$$

where f is the operating frequency.

Circuit	Switching (mA)	Total (mA)	Switching as % of total power
16-bit parallel multiplier	13.3	15.3	87%
32-bit Manchester adder	3.33	3.92	85%
Floating-point processor	231	307	75%
Signal processor	20.1	27.2	74%
Sense amplifier portion of static RAM	12.3	22.6	54%
Micro-controller	7.91	16.9	47%
Static RAM	70.2	201	32%
Dynamic RAM	14.5	76.5	19%

Table 2.1 Switching power in different types of circuits [Tiw94].

Since not always the outputs of the gates switch, the above expression is extended as:

$$W_{avg} = \frac{1}{2} (C_{load} V_{DD}^2 \alpha f) , \quad (2.1)$$

where α is the number of times that the capacitance C_{load} is charged or discharged. The product $C_{load} \alpha$ is called the *switched capacitance*.

Again, the contribution of this power component to the total power depends on the type of the circuit. In Table 2.1 we observe that for data-path circuits (i.e. multipliers, adders, signal processors, etc.) this component accounts roughly for the 80% of the total power [Tiw94].

2.1.3 Defining power consumption

Since this work mainly focus on data-path circuits, the power consumption will be defined by equation 2.1. This equation reveals the three degrees of freedom when designing for low power (considering a fixed operating frequency): voltage, physical capacitance and activity.

With its quadratic relationship to power, voltage reduction offers the most promising means for reducing power. Section 2.3 will address the technique of reducing the power supply to decrease power.

Reducing physical capacitances can be achieved by using small devices and short interconnection wires. Sections 2.7 and 2.6 will briefly address this issue.

But once the supply voltage and the device dimensions have been fixed, it is the switching activity of the signals of the circuit that will ultimately determine its power dissipation. Techniques that reduce this activity will be explained in Sections 2.2 to 2.4. Moreover, this is the degree of freedom targeted by the techniques presented as contribution of this work.

2.2 SYSTEM LAYER

The least explored layer of the design process is the system layer. Although the greatest power reductions may be obtained through a proper algorithm selection or system partitioning, the designer has to trade off among contradictory parameters, and their impact is mostly unknown in the early phases of the design process.

The most well-known low-power design strategies at the system layer are: system shutdown, system partitioning and algorithm selection.

2.2.1 System shutdown

An obvious mechanism for saving power is to implement power management techniques to shut down parts of the system hardware that are idle. An idle part still consumes power because of the clock line and the meaningless input data, which generates some activity. Three system-shutdown strategies are divided into non-predictive and predictive.

Non-predictive shutdown

The non-predictive approaches are based on shutting the system down after the processor has been idle for a certain time interval. This interval is fixed or set by the operating system. The system remains in a power-saving mode until an external event asserting one of the wake-up signals will bring the processor out of that mode (external interrupts, system management interrupts, reset, etc.). The system may be partially shut down, thus still performing some operations while in one of the power-saving modes. As an example, the Power PC 603 implements the following software-programmable power-saving modes [GIG⁺94]:

- *doze mode*: the cache coherence is maintained.

- *nap mode*: all logic except the time base/decrementer is disabled. Since cache coherence cannot be maintained in this mode, a pair of handshake signals enable the system to ensure that the processor goes into this mode only when cache coherency will no longer be a problem.
- *sleep mode*: in this mode, the maximum power savings are obtained. The clocks to all units are disabled.

Predictive shutdown

The length of the idle time may be predicted based on the computation history, and the processor is shutdown if the predicted length of idle time justifies the cost of shutting down. In [CSB94] a predictive system shutdown has been implemented for a portable X-terminal device.

2.2.2 System partitioning

System partitioning may be considered at different levels of granularity:

- to decide which components of the system will be realized in hardware and which will be implemented in software (hardware/software co-design). Some preliminary research has been done in estimating the power consumption of a software program [TMW94].
- to decide which system functions go to which components. In a traditional system partitioning, the primary design goal is to keep the chip count low. In a low-power system partition, the new primary design goal of low-power may favor a different partitioning. Furthermore, the new system requires power management functions that were not required in earlier systems. An example is found in [Wol95].
- to decide whether the system functions are distributed or not. Distributed processors, memories and controllers can lead to significant power savings. The drawback is the increase in area. One example is found in [LR94b], where a non-distributed and a distributed design of a vector quantizer is implemented and the power compared.

2.2.3 Algorithm selection

The choice of the algorithm used to implement the application may have a dramatic impact on the power consumption. The most straightforward decision is to select,

DCT Algorithm (8×8)	Multiplications	Additions
Brute Force	4096	4096
Row-Col. DCT	1024	1024
Chen's Algorithm	256	416
Lee's Algorithm	192	464
Feig's Algorithm	54	462

Table 2.2 Number of multiplications and additions of the 8×8 -matrix DCT execution for different algorithms [BCS92].

from the different algorithms that implement the same function, the one with less number of operations to be executed. This statement may not be true if the algorithm has different types of operations. Since, for example, multiplications typically consume much more power than additions, the algorithm with less multiplications is a candidate.

As an example, consider the Discrete Cosine Transform (DCT) of an 8×8 matrix. Table 2.2 shows five different algorithms that implement the DCT and all of them present different number of additions and multiplications [BCS92]. Clearly, we will choose Feig's DCT algorithm since it has the smallest number of multiplications.

Other features that a low-power algorithm should present are:

- regularity: to minimize the power in the control hardware and the interconnection network.
- modularity: to exploit data locality through distributed processing units, memories and control.
- few memory references: since references to memories are expensive in terms of power.

2.3 ARCHITECTURE LAYER

At the architecture layer, four topics will be studied: parallel and pipelined processing along with voltage reduction, compiler-related techniques, cache design and data representation.

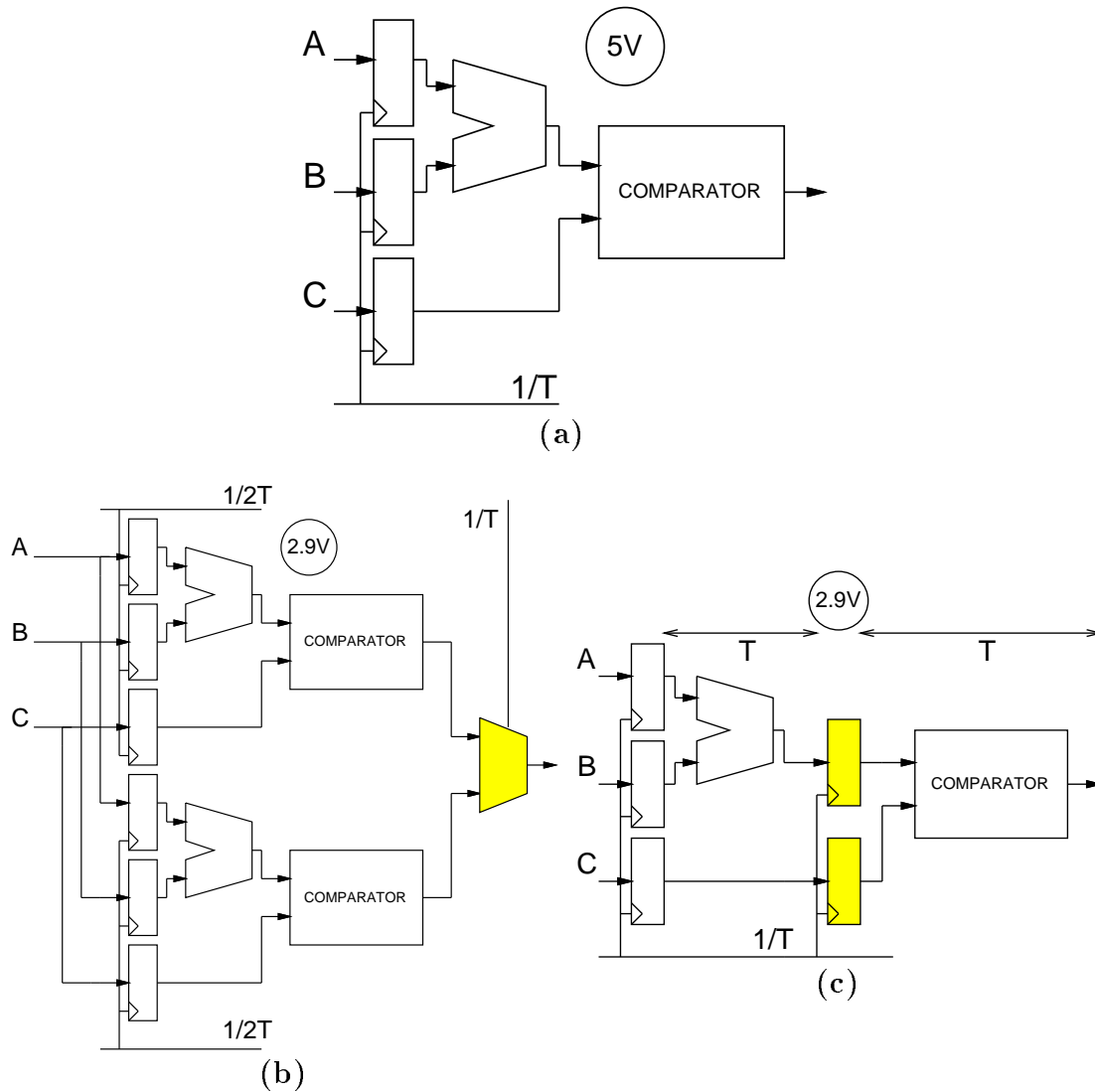


Figure 2.4 (a) Data-path circuit at 5V; (b) Parallel and (c) pipelined versions at 2.9V. ([BCS92])

2.3.1 Parallel and pipelined processing

The most well-known technique for reducing power, i.e. decreasing the power supply, is combined at the architecture layer with parallel and pipelined processing to overcome the loss of performance produced by the reduced power supply.

The power consumption depends quadratically on the power supply V_{DD} (refer to equation 2.1 in Section 2.1). Therefore, a reduction in V_{DD} leads to a significant power-consumption reduction. Moreover, reducing V_{DD} leads to an increase

of the delay of the system since the delay is proportional to the capacity being charged/discharged and inversely proportional to $V_{DD}/(V_{DD} - V_{th})^2$, begin V_{th} the threshold voltage [CB95].

Through parallel and pipelined processing, this loss of performance is overcome. To illustrate this, the simple data-path circuit of Figure 2.4(a) will be used. This circuit compares an operand C and the result of an operation between A and B . The power consumption of this circuit is:

$$W = \frac{1}{2} (C_{switched} V_{DD}^2 f) ,$$

where f is the operating frequency and $C_{switched}$ is the capacitance of the circuit signals being switched (i.e. the product C_{load} and α in equation 2.1).

Let us assume that the V_{DD} of this circuit is 5V and that if we reduce it to 2.9V the delay is doubled.

Parallel processing

In Figure 2.4(b) the circuit is duplicated and powered at 2.9 V, therefore allowing each part to work at half the original rate. Since there are two units producing a result at half the original rate, the same throughput is maintained.

The power consumption of the parallel version of the data-path circuit is:

$$W_{par} = \frac{1}{2} \left((2 + \xi_{par}) C_{switched} \left(\frac{2.9}{5} V_{DD} \right)^2 0.5 f \right) .$$

The switched capacitance is at least the double of the original one (since the circuit has been duplicated) plus some overhead because of the additional logic (shadowed multiplexer in Figure 2.4(b)). Ideally (i.e. with $\xi_{par} = 0$), $W_{par} = 0.34W$, therefore achieving a power reduction of 66%.

Pipelined processing

The same ideal power reduction of 66% can be obtained if the original circuit is pipelined as shown in Figure 2.4(c). Now, both states can operate at half the speed, and thus the supply voltage can again be reduced to 2.9V with no loss of throughput.

The power consumption of the pipelined version is:

Architecture	Voltage	Area	Power
Original data-path	5V	1	1
Pipelined data-path	2.9V	1.3	0.39
Parallel data-path	2.9V	3.4	0.36
Pipeline-Parallel data-path	2.0V	3.7	0.2

Table 2.3 Normalized area and power for the different architecture designs of the dat-apath example in Figure 2.4 ([BCS92])

$$W_{pip} = \frac{1}{2} \left((1 + \xi_{pip}) C_{switched} \left(\frac{2.9}{5} V_{DD} \right)^2 f \right) .$$

The operation frequency remains the same and there is an overhead increase in the switched capacitance because of the additional registers. If $\xi_{pip} = 0$, then $W_{pip} = 0.34W$.

Overhead circuitry

The 66% power reduction obtained previously has neglected the overhead circuitry needed to parallelize or to pipeline the original circuit. In [BCS92] this data-path example has been implemented and the overhead has been taken into account. The results are shown in Table 2.3.

In this table we observe that the actual power reductions obtained are 61% and 64% for the pipelined and parallel versions respectively. Although higher power savings are obtained with the parallel version, its large area overhead makes the pipelined approach more appealing. Note, however, that the higher the level of pipelining is, the higher is the power associated to clock the registers.

Both techniques of pipelining and parallelizing can be used together as shown in the last row of Table 2.3 to further decrease the power supply and the power consumption.

There is a limit in reducing the power supply. At low V_{DD} voltages, the static power consumption of the circuit becomes important and equation 2.1 is no longer valid to model the power consumed by the circuit. In Section 2.7 this problem will be addressed.

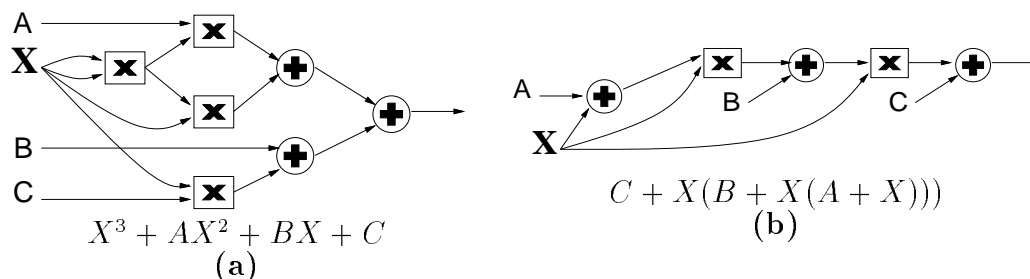


Figure 2.5 Implementing polynomial $X^3 + AX^2 + BX + C$ (a) straightforward and (b) Horner's scheme.

2.3.2 Compiler transformations

Several well-known compiler techniques used for optimizing area or throughput may reduce the power of the synthesized circuit. These techniques transform the data-flow graph of the application in a manner that the input/output behavior is preserved.

One straightforward use of these transformations is the following: once the throughput design constraint has been met, further optimize for throughput. Afterwards, the power supply can be reduced to trade the over-optimized throughput for power.

The following compiler transformations may reduce power [CPM⁺95]:

- *operand reduction*: for example, using the Horner's scheme to implement polynomials. Two multiplications are eliminated on a third-order polynomial without increasing the critical path as shown in Figure 2.5.
- *operation substitution*: costly operations in power may be substituted for other less expensive operations. This is often used in software compilers [ASU86]. The drawback of this transformation is that it often implies an increase in the critical path length. An example is shown in Figure 2.6, where a multiplication has been traded off in Figure 2.6(b) for an addition at the expense of an additional adder delay in the critical path.

When speed is not a major issue, a significant reduction in hardware complexity is achieved by performing multiplications over several clock cycles as a series of shifts and adds. The average number of shifts and adds per multiplication can be minimized by rounding precise coefficients to the nearest canonical signed digit coefficients [Sam89].

- *reducing logic-depth*: reducing logic depth leads to a decrease in useless activity (see Section 2.4.1). An example of this fact is shown in Figure 2.7, where

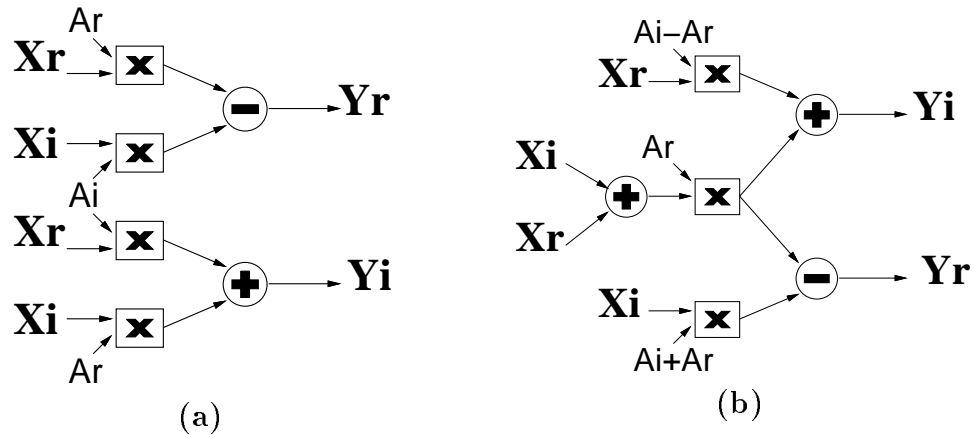


Figure 2.6 Substituting one multiplication by one addition.

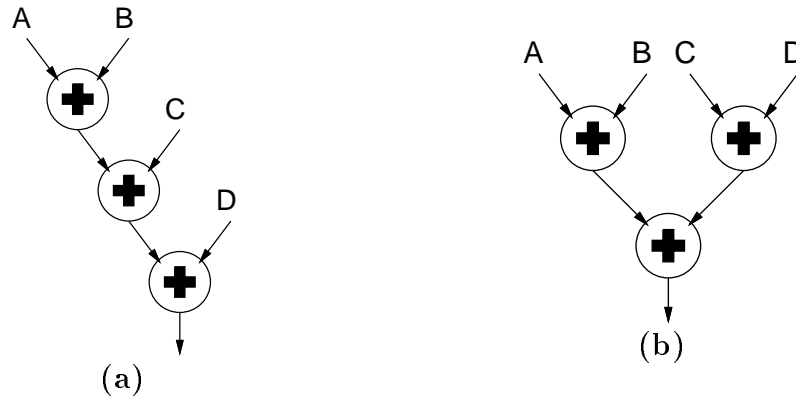


Figure 2.7 (a) Unbalanced and (b) balanced data-flow graph of the addition of four operands.

two different implementations of the addition of four operands are shown. All operands are assumed to arrive at the same time. The implementation in 2.7(a) presents more activity because of the unbalanced paths: for example, the second adder first computes the unnecessary addition of C and the previous output of the first adder. When the correct value of the first adder finally propagates, the second adder recomputes the sum. This unnecessary activity is not presented in Figure 2.7(b).

- *resource utilization*: by distributing more uniformly the operations over the available time, the scheduling algorithm may reduce the required amount of hardware (i.e. increases the time-sharing of the resources) while preserving the number of control steps [PR91]. Reducing the amount of hardware implies less

useless activity but the amount of interconnection units and control logic may increase.

2.3.3 Cache design

A cache improves the system performance by reducing memory references. Since memory references are very power consuming, the existing techniques to reduce the miss rate and, therefore, increase performance (associativity, buffering, sub-array access, etc.) also reduce power [RP96]. However, some guidelines should be followed for the design of low-power caches:

- power and hit rate increase with the cache size. Therefore, an increase in size that only obtains a small reduction of the miss rate should be carefully considered since the power cost may outweigh the power savings obtained with the lower miss rate.
- implement the array part of the cache using a six-transistor SRAM cell design instead of a four-transistor cell. The latter, although it presents less area, consumes more static power.

2.3.4 Data representation

The representation chosen for the operands also affects the power dissipation since one representation may inherently present more bit toggles between two consecutive words.

Arithmetic representation and data encoding are two examples where a proper data representation for low power may be used.

Arithmetic representation

There are several arithmetic representations for data: two's complement, sign-magnitude and canonical signed-digit. Among them, two's complement is the most widespread. In this representation, the most significant bits are sign bits (1 for negative and 0 for positive). In the sign-magnitude representation, however, only the most significant bit has the information of the sign. Therefore, if the application usually has to operate with small values but needs to keep a large bit width for some operations involving large values, a high activity may arise if the values present a large number of sign transitions.

Two's complement	Value	Sign-magnitude
11111101	-3	10000011
11111110	-2	10000010
11111111	-1	10000001
00000000	0	00000000
00000001	1	00000001
00000010	2	00000010
00000011	3	00000011

Table 2.4 Two's complement and sign-magnitude representation for the values -3 to 3 using 8 bits.

For example, consider the two's complement and sign-magnitude representation for the values -3 to 3 using 8 bits as shown in Table 2.4. Any transition from a negative value to a positive (or vice-versa) implies an average number of 7 bit transitions when using the two's complement representation, whereas only 2 with the sign-magnitude representation. The drawback in using sign-magnitude representation is the additional overhead required to perform the operation since it may depend on the signs of the operands. This results in a sequence of decisions that have to be made, implying additional logic and execution time [Kor93].

Data encoding

Data encoding is specially useful for low power when a large capacitance is driven by these data. This occurs, for example, at the input/output pads (I/O) of the chip. Typically, the load capacitances of the I/O pads are one to three orders larger than in the internal circuit.

The Gray encoding is one example of a low-power address encoding. It takes advantage of the sequential memory access, which occur often in a general processor for executing consecutive instructions in basic blocks. For example, for the sequence of values from 0 to 16, there are 31 bit switches when the values are encoded in binary representation while only 16 bit switches when they are represented in Gray code. For random access patterns, Gray code and binary code have similar number of bit switches. The main feature of the Gray code is that contiguous Gray-coded numbers only differ in one bit [Hay88].

To implement a Gray-code addressing system, the instruction field of the target address in a branch instruction is modified in such a way that the calculated target

address is the correct target address in Gray-code addressing system. Moreover, a Gray-code counter is needed for incrementing the program counter [STD94].

2.4 LOGIC LAYER

Several low-power design techniques exist at the logic layer (or gate layer). All the techniques reviewed in this section aim at reducing the activity of the synthesized circuit. Again, power is traded off, in some cases, for area and delay. The techniques target combinational and sequential circuits.

2.4.1 Combinational circuits

The power consumed by a combinational circuit can be reduced by a proper path balancing of the signals and during the multi-level logic synthesis and technology mapping process.

Path balancing and gate sizing

Equation 2.1 in Section 2.1 pointed out that once the supply voltage and the device dimensions have been fixed, it is the switching activity that will determine the power consumption of the circuit. The switching activity of a circuit may be divided into:

- *useful activity*: those signal transitions needed to change the gate outputs of the circuit from the previous cycle (a function of the previous primary inputs) to the current cycle (a function of the current primary inputs).
- *useless activity* (also called *hazard activity*): the rest of the circuit activity.

As an example, consider the circuit in Figure 2.8(a). Assume that the critical signal is B . At time 0, the previous value of the output Y is 1 and at time T is the same. Since the correct value of Y is at time T , any double transition of Y is considered to be a hazard.

Hazards are generated because of the different arrival times of the signals to the inputs of the gates. One of the reasons of this unmatched delay is the different propagation times of the gates (which vary depending on the input) and interconnections among gates¹. Moreover, these propagation times vary inside each gate depending

¹At sub-micron design, the metal interconnect contributes most of the net's capacitance. The narrow interconnect offers significant resistance, while the smaller transistors switch faster and cause much less capacitive loading.

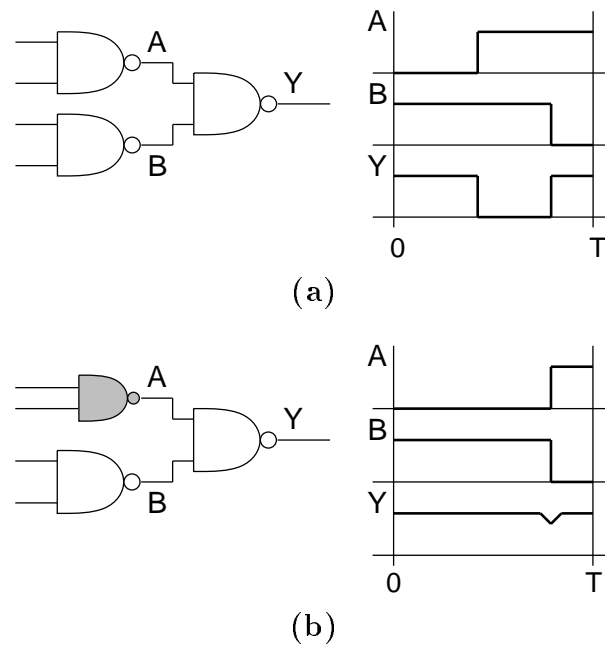


Figure 2.8 Delaying signals to eliminate hazards.

on the input. Therefore, the amount of useful activity in a gate-level description of a circuit can be calculated considering that the gates follow a *zero-delay* model, that is, the propagation delays of the gates are 0.

The power dissipated because of hazards strongly depends on the circuit topology and the applied inputs, but clearly accounts for a significant fraction of the overall power consumption (15%-20% [BFR94] or even higher (70%) [SGDK92]).

Hazards at the output of a gate can be minimized if all the paths in the circuit feeding that gate have approximately the same delay. This implies that any potential hazard at its output will be eliminated since the delay of the gate acts as a filter. Thus, balancing path delays may cause a reduction of the power consumption. One method to balance the paths is by *gate resizing* [LM93], i.e. by replacing some gates of the circuit with others implementing the same logic function but having smaller area. A smaller gate takes longer in charging/discharging its output load, thus delaying its output signal. In the example of Figure 2.8(a), signal *A* can be delayed as shown in Figure 2.8(b). Now, signal *Y* presents no hazards.

Resizing a gate implies resizing its transistors. Since smaller devices present less capacitance, an efficient low-power strategy is to use minimum size devices whenever

possible. But care has to be taken since smaller gates have larger transition times at their outputs. A large transition time at the input of a gate affects the width of the short-circuit region (see Section 2.1) of the gate and hence increases the power consumption [BIO95]. Anyway, along the critical path devices should be sized up to meet performance constraints.

Multi-level logic synthesis

Traditionally, logic optimization based on identifying logic shared among different nodes of a circuit has provided an efficient method for minimizing a cost function (the area) of a boolean network. The area is estimated by computing the number of literals in the factored form of the network [BM82]. The same process of logic synthesis can be applied to minimize the power of the network simply by incorporating the power measure into the cost function [RP93, IP95].

As an example, consider the synthesis of the following boolean functions [RP93]:

$$f_1 = ad + bcd + ae ,$$

$$f_2 = a + bc + dh + eh .$$

Assume that the signal probabilities of all input signals is 0.5 and that the activities, i.e. the number of transitions per cycle, are:

$$\alpha_a = 0.1, \alpha_b = 0.6, \alpha_c = 3.6, \alpha_d = 21.6, \alpha_e = 129.6, \alpha_h = 3.6 .$$

The cost function mentioned above is simply the switched capacitance ($C_{load} \alpha$ product in equation 2.1 in Section 2.1). Therefore, the number of transitions at each node of the circuit should be calculated. The *transition density measure* [Naj91] will be used for this purpose. This measure will be explained in Chapter 5.

There are several ways to synthesize the boolean functions. For example, a straightforward two-level implementation has an area of 11 (5 literals for f_1 plus 6 literals for f_2). The power cost obtained with the transition density measure is 503.4. A possible multi-level implementation is:

$$f_3 = a + bc ,$$

$$f_1 = \boxed{f_3} d + ae ,$$

$$f_2 = \boxed{f_3} + dh + eh ,$$

with an area of 11 and a power cost of 476.5. Another multi-level implementation is:

$$f_3 = d + e ,$$

$$f_1 = \boxed{f_3} a + bcd ,$$

$$f_2 = a + bc + \boxed{f_3} h ,$$

with an area of 12 and a power cost of 423.1. This implementation has more area but less power.

Technology mapping

Once the logic optimization has been done, the logic functions obtained are transformed into a technology-dependent circuit with minimized power consumption.

The graph-covering-based technology-mapping process can be summarized in the following three steps [WE93]:

1. the logic equations are converted into a graph (subject graph) where each node is restricted to one of a set of base functions (e.g. 2-input NAND gates and inverters).
2. each library gate is also represented by a graph (pattern graph) where each node is restricted to one of the base functions.
3. find a minimum cost covering of the subject graph.

Low-power techniques can be applied in steps 1 and 3 [TAM93, TPD94] during step 1, a proper subject graph should be found that minimizes the activity at the outputs of the gates. As an example, consider the subject graph of Figure 2.9(a), implementing a 4-input NAND function. The input signals are assumed to be independent. With the probabilities and activities of the input signals shown in the figure, the amount of activity at the output nodes is 1.505. The subject graph in Figure 2.9(b), however, present less activity (2.095) and, therefore, should be chosen. In this example, the cost function has been considered to be only the activity. A more accurate cost function may be the switching capacitance.

Further power reductions during the covering step are obtained based on the fact that internal nodes of a gate have less capacitance than output nodes. Therefore,

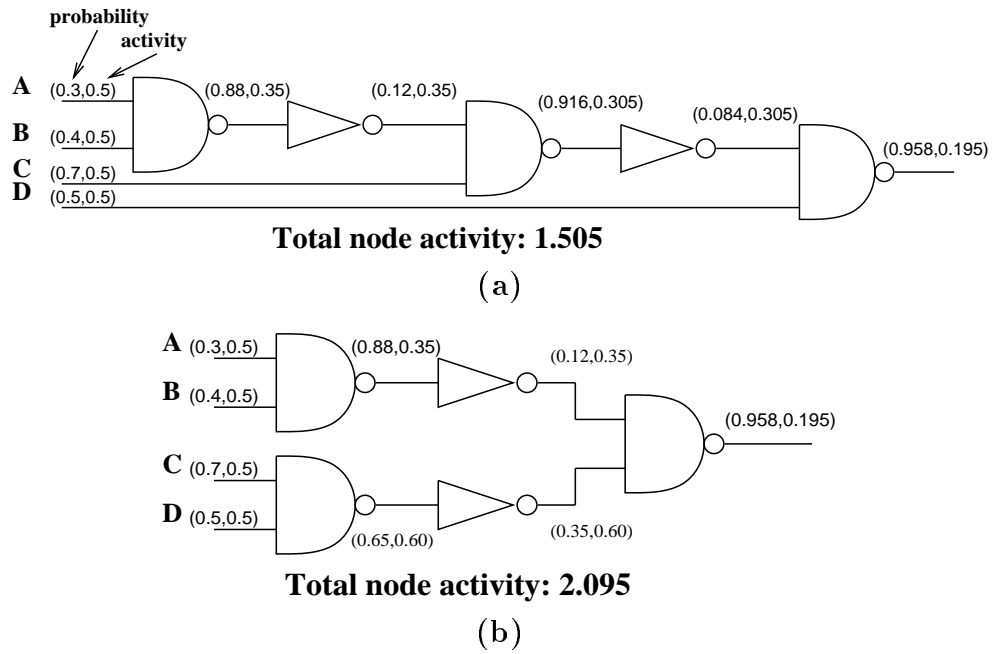


Figure 2.9 Different subject graphs implementing a 4-input NAND function.

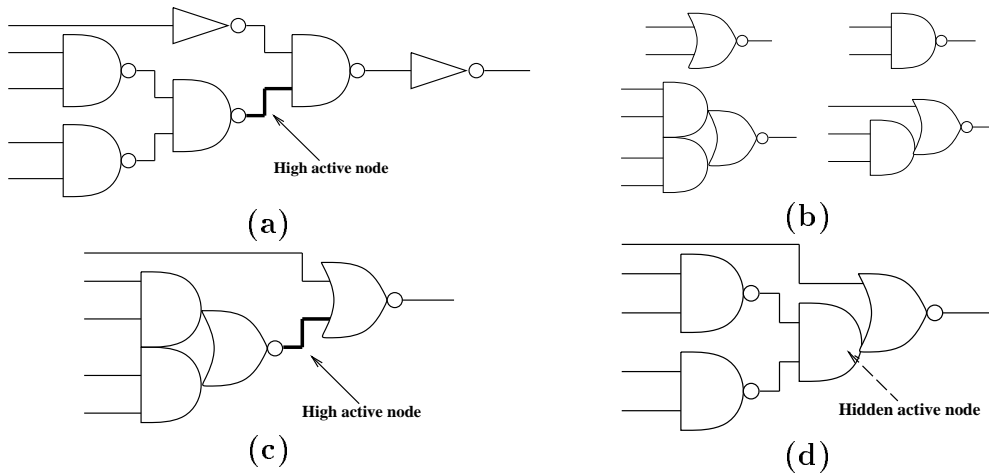


Figure 2.10 (a) Subject graph; (b) available gates; (c) low-area mapping; (d) low-power mapping.

if a node in the subject graph presents a high activity, it should be *hidden* as an internal node of a gate of the targeted library.

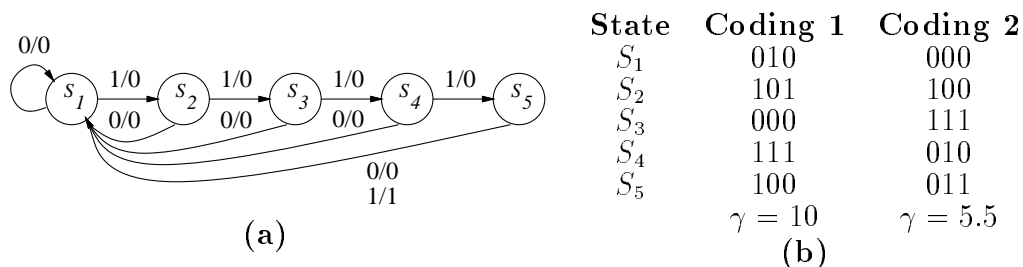


Figure 2.11 (a) FSM example; (b) two different codings with different switching activity among them ([RP93]).

For example, consider the subject graph of Figure 2.10(a) and the available library gates of Figure 2.10(b). The mapping of Figure 2.10(c) presents less area (12 transistors in static CMOS), but the high active node still remains as an output node. The mapping of Figure 2.10(d) presents more area (14 transistors) but the high active node has been hidden on a gate.

2.4.2 Sequential circuits

The techniques addressed for low-power sequential circuits cover the finite-state machine (FSM) state assignment, the circuit re-timing and the disabling of input registers when not needed.

FSM state assignment

The idea behind the FSM state assignment for low power is to assign similar codes to states S_i and S_j if the probability of state transition between these two states is high.

The difference between two codes is the number of non-equal bits. This measure is called the Hamming Distance. Therefore, the goal of the FSM state assignment for low power is to minimize a value γ defined as [RP93]:

$$\gamma = \sum_{\text{over all edges}} p_{ij} H(S_i, S_j),$$

where $H(S_i, S_j)$ is the Hamming Distance between codes S_i and S_j and p_{ij} is the conditional state transition probability (the probability that the next state is S_j given that the FSM is in state S_i).

The FSM that produces a 1 whenever there is sequence of five 1's at its input is shown in Figure 2.11(a) as an illustrative example. The probability of the input is

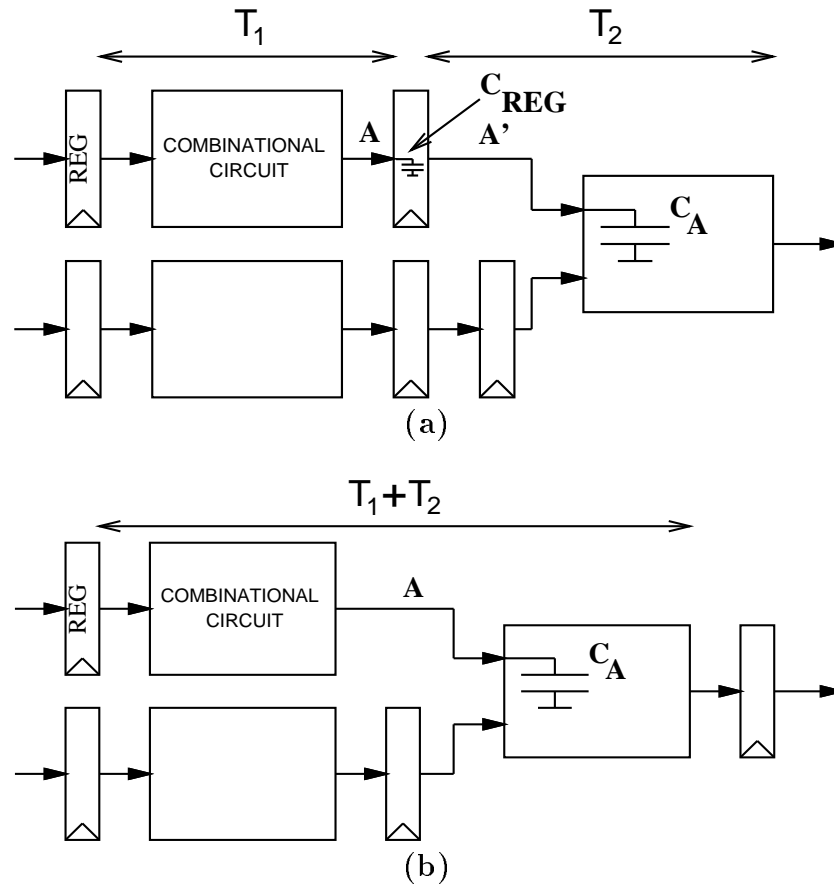


Figure 2.12 Sequential circuit (a) before and (b) after re-timing.

assumed to be 0.5. In Table 2.11(b), two different state codings are shown. With the first coding, value γ is 10; with the second coding, γ is reduced to 5.5. Both codings can be implemented with the same area [RP93].

Re-timing

The re-timing technique [LS83] goal is to move registers across portions of combinational logic in order to minimize the cycle time or the number of registers while maintaining the behavior of the circuit. For example, the sequential circuit in Figure 2.12(a) can be re-timed to eliminate one register as shown in Figure 2.12(b). Moreover, the re-timing technique can be applied along with combinational synthesis techniques to optimize the portions of combinational logic between register boundaries [MSBSV91].

The re-timing technique for low power is based on the fact that the switching activity at register outputs in a sequential circuit can be significantly less than the activity at the register inputs because only the last signal transition at the input of a register may propagate to the output when the register is clocked. A register may, then, save power specially if its input is very active and its output has a high load capacitance.

Therefore, once the constraint on performance has been met, the re-timing technique can be used to decrease power by selecting the set of nodes which, by having a register placed at their output, lead to the minimization of the switching activity-capacitance product in the circuit. For example, in the circuit of Figure 2.12(a), the power consumption associated to signals A and A' is proportional to: $\alpha_A C_{REG} + \alpha_{A'} C_A$. Assuming that $\alpha_{A'}$ is significantly lower than α_A and C_A is higher than C_{REG} , the power of both nodes A and A' is approximated by $\alpha_{A'} C_A$. The power associated to signal A in Figure 2.12(b) is proportional to $\alpha_A C_A$. Since $\alpha_{A'} < \alpha_A$, power is saved by not removing this register during the re-timing process.

A drawback of the re-timing technique is that the critical path of the circuit may increase. The circuit in Figure 2.12(b) has a critical path of $T_1 + T_2$ whereas in Figure 2.12(a) it is $\max(T_1, T_2)$.

Disabling input registers

In synchronous circuits some useless power dissipation can be eliminated by disabling the clock in parts of the circuit where useful computation is not being performed. It is the same idea of the system shutdown described in Section 2.2 but at the logic level.

One approach is to add a redundant logic to disable the registers at some of the inputs if the operation that the combinational block has to perform does not depend on those inputs [AMD⁺94]. Therefore, fewer activity is generated in the combinational block, thus saving power, and also some power is reduced in the disabled registers.

As an example, consider a comparator circuit in Figure 2.13(a) performing the operation $A < B$ being A and B the n -bit two's complemented operands ($\langle a_{n-1}, \dots, a_0 \rangle, \langle b_{n-1}, \dots, b_0 \rangle$). Clearly, if $a_{n-1} = 1$ and $b_{n-1} = 0$ (B greater than A) or vice-versa (A greater than B) the $\langle a_{n-2}, \dots, a_0 \rangle$ and $\langle b_{n-2}, \dots, b_0 \rangle$ bits have no information to calculate the operation $A < B$. Therefore, the input registers of these bits may be disabled. Let f_a be the boolean function that implement these conditions. If $f_a = 1$, the registers should be disabled (Figure 2.13(b)).

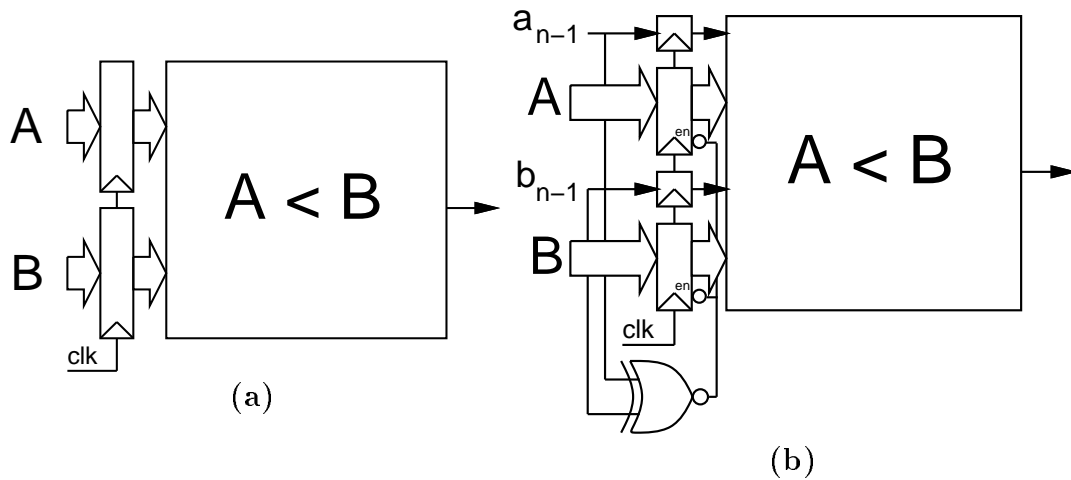


Figure 2.13 Comparator circuit (a) without and (b) with disabled input registers.

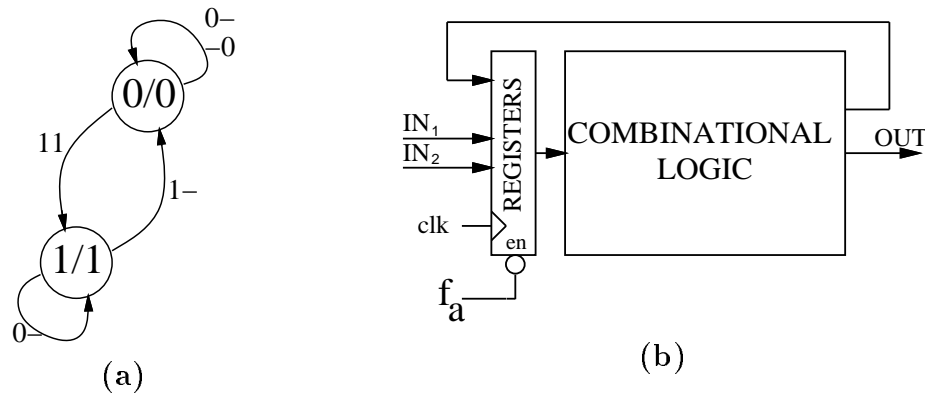


Figure 2.14 (a) STG and (b) FSM with the registers enabled/disabled by signal f_a .

Another approach is to disable all the input registers in a finite-state machine (FSM) when a self-loop in the signal-transition graph (STG) has been detected [BSdM94, BM95]. In a self-loop, the state does not change. Therefore, by disabling the input registers, no switching activity is generated in the combinational block and power is saved.

As an example, consider the simple STG of Figure 2.14(a) with two self-loops. One is produced when the FSM is in state 1 (with output 1) and the first input is 0. The other self-loop is in state 0 (with output 0) and either the first input is 0 or the second input 0. Again, a boolean function f_a is needed to implement the self-loop conditions.





































Logic Style	Area	Delay	Power	Design
Static CMOS				
BICMOS				
BINMOS				
Pseudo-NMOS				
N-P Domino Logic				
CMOS PTL				
CPL				
DPL				
SRPL				

Table 2.5 Area/delay/power/design simplicity comparison of some logic styles.

In all these approaches of disabling the input registers, the area of the synthesized circuit increases because the logic to implement the enabling/disabling function f_a . Moreover, the delay also increases since this logic usually is in the critical path.

2.5 CIRCUIT LAYER

At the circuit layer four topics related to low-power design will be addressed: the logic style, the asynchronous design, the design style and the adiabatic computing.

2.5.1 Logic style

Table 2.5 presents a comparison in area, delay, power and design simplicity among different logic styles (or logic families). The first part of the table shows some static and dynamic logic styles. The second one presents several pass-transistor logic families [BE95]. A more accurate comparison in term of power is reported in [LKB94] for some of these logic styles.

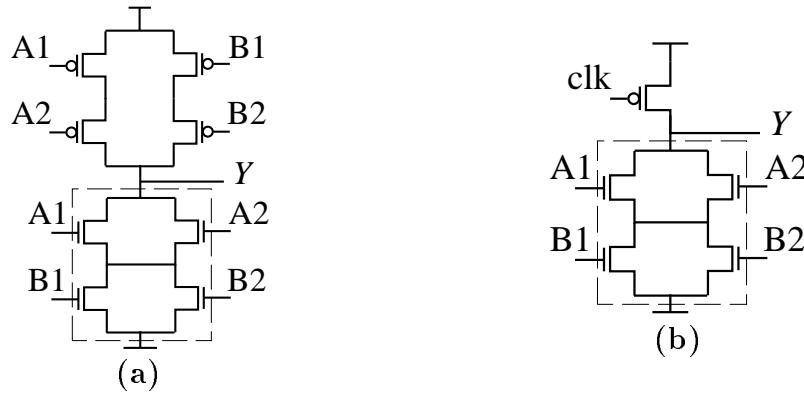


Figure 2.15 (a) Static and (b) dynamic implementations of the boolean function $Y = (A_1 + A_2)(B_1 + B_2)$.

Static vs. dynamic logic style

Static CMOS logic design is very popular for its design simplicity and it also presents a good low-power behavior because of its negligible static power dissipation. However, it suffers from a high area and low speed. The BICMOS logic style overcomes the limited current drive of the CMOS when the load capacitance is high. This comes at the expense of an increase in area, power (specially for small loads) and design simplicity. The BINMOS logic family presents less power consumption than BICMOS but is still higher than in CMOS. The Pseudo-NMOS logic style offer a good design simplicity and low area since the P-block of the static CMOS is replaced by a single PMOS transistor. The problem with this logic style is the non-zero static power dissipation.

To reduce area and improve the speed of CMOS circuits, the dynamic logic style may be used. The N-P Domino logic in Table 2.5 is a dynamic logic family. A clock is needed in dynamic logic as shown in Figure 2.15, where a static CMOS gate is compared to its dynamic version. When the clock is low, the output node is precharged; when the clock is high, the output node may be discharged depending on the logic performed by the N-logic block. During the evaluation phase, the output node maintains the value stored in its load capacitance if it is not discharged.

In general, the dynamic logic families are not suitable for low power applications. The most important reasons are [Lan94]:

- the clocking scheme adds more capacitance to the clock distribution, increasing the power and skew.

- the unnecessary activity generated at the output of the gates.

For example, the standard cells and data-path element libraries of the Power PC 603 [GIG⁺94] use fully static logic for design simplicity and avoid the higher power dissipation of dynamic logic caused by the pre-charge phase. Moreover, a dynamic design also requires saving the state of the chip if clocks are disabled.

Other drawbacks concerning the robustness and charge sharing are presented in dynamic logic families [Lan94, BE95]. Nevertheless, dynamic implementation might actually achieve low-power consumption in some specific applications such as PLAs.

Pass-transistor logic style

The most promising logic families for low power are those based on the pass-transistor logic style. Some of these logic families are shown in the second part of Table 2.5. The most important drawback in using a pass-transistor logic family is the difficulty in designing circuits. There is not much commercial CAD support for the synthesis of pass-transistor gate designs.

2.5.2 Synchronous vs. asynchronous design

Asynchronous circuits are typically designed on the basis of an explicit *self-timed* protocol, which implies that the delay of an asynchronous circuit is data dependent.

Ideally the asynchronous circuits should drastically reduce the power consumption of their synchronous counterparts since there is no global clock and any signal transition generated in the circuit is useful (i.e. the circuit only dissipates when and where necessary).

The main drawback with asynchronous circuits is the area overhead to implement the handshake and completion protocols needed for a correct operation. In most of the cases the power consumption of this overhead overcomes the power savings obtained with the elimination of the clock signal.

However, CAD tools for asynchronous design are still under research and improvements are expected in the next years in the asynchronous arena. Some of the results reported so far seem to indicate that it is possible to obtain power reductions using asynchronous techniques, but it depends on the type and characteristics of the circuit. For example, in [KGG95] a dynamic synchronous and asynchronous adder are compared. The asynchronous version is 20% more power consuming than the

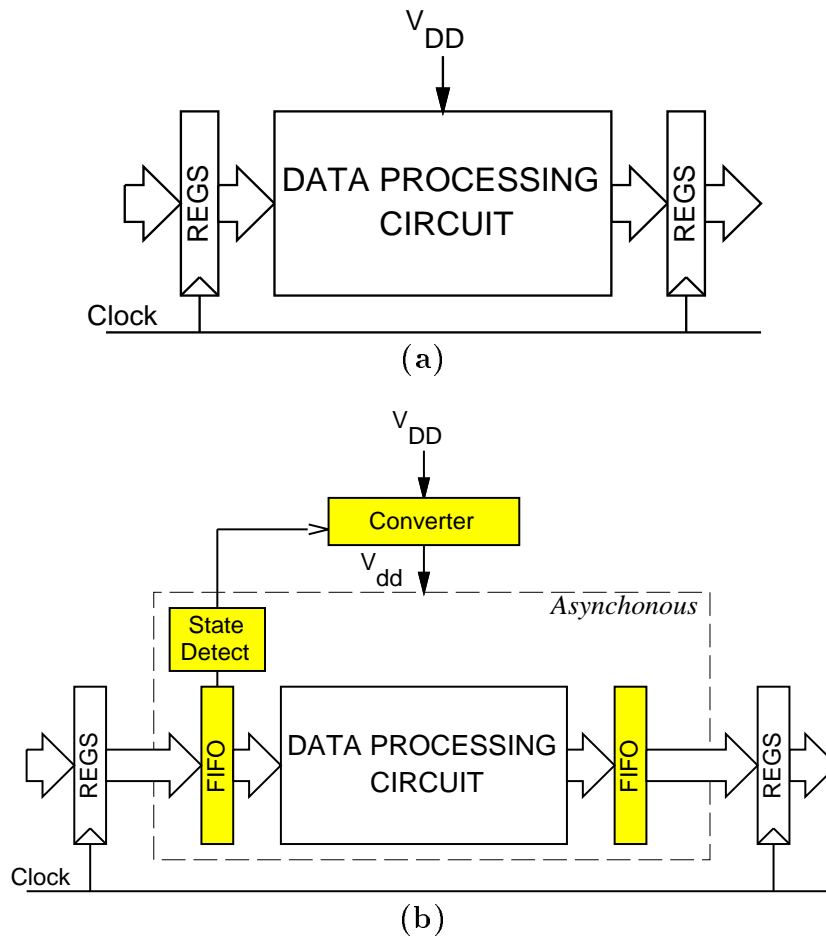


Figure 2.16 (a) Original synchronous system; (b) asynchronous system with adaptive scaling of the supply voltage ([NS94]).

synchronous one. However, in [vBBK⁺94] the asynchronous implementation of a DCC error corrector reduces the power 80% with respect the synchronous one.

Asynchronous design with adaptive scaling of the supply voltage

The data dependent delay of an asynchronous circuit can be combined with the reduction of the power supply to obtain a low-power circuit that operates in a synchronous environment [vBN93, NS94].

Figure 2.16(a) shows the original synchronous system working at a frequency $1/T_{max}$. T_{max} is defined so that the response time of the data processing circuit is guaran-

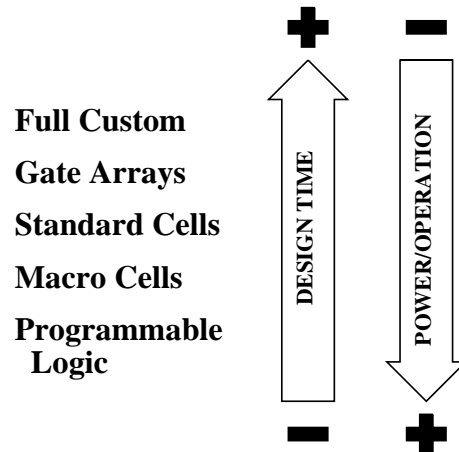


Figure 2.17 Design time vs. power consumption of some design styles.

ted for the worst-delay case. An asynchronous design of the circuit has an average response time of T_{avg} . The delay overhead, $T_{max} - T_{avg}$, can be converted into corresponding power savings by reducing the power supply until the computational delay of the asynchronous circuit plus a safety margin is T_{max} .

The asynchronous system with adaptive scaling of the supply voltage is shown in Figure 2.16(b). The system consists of the data processing circuit itself, two FIFO buffers, a state detecting circuit and a DC/DC converter for scaling down the supply voltage. The converter can be anything from a resistive device to a more sophisticated lossless device [SBS94]. The scaling circuit may consist of a D/A converter, which scales the supply voltage linearly depending on the number of data words in the input FIFO.

Since the system operates in a synchronous environment, the input (output) buffers never should full (empty). Therefore, synchronization problems will not occur at the synchronous/asynchronous interface. The state detecting circuit monitors the state of one of the buffers, for example the input buffer. If it is running empty, it indicates that the circuit is operating too fast, and the supply voltage can be reduced. If, on the contrary, the buffer is running full, the supply voltage must be increased.

2.5.3 Design style

There is a range of design style options to implement a circuit: full custom, gate arrays, standard cells, macro cells, programmable logic structures. These design

style options have contradictory design-time and power-consumption characteristics as shown in Figure 2.17. For example, in a full-custom design, techniques such as transistor sizing can be optimally applied to individual circuits in an ad-hoc basis. Unfortunately, this comes at a low design productivity. The gate array (and sea-of-gates) design style consist of predefined transistors and the designer needs only to wire the different transistors using metalization and contacts. In a standard-cell design style, the cells are already predefined. Care has to be taken in the routing (discussed in the next section) to reduce the power. Moreover, different designs of the same cell with different sizes should be available for low-power design.

In general, the more flexible design styles present more power per operation than the least flexible. For example, in a programmable logic structure there is a higher capacitance to be switched per operation than in full custom because of the overhead needed to provide this flexibility.

2.5.4 Adiabatic computing

The idea of adiabatic or charge-recovery circuits is to recover some of the energy needed to change the capacitor's voltage. This is possible since the energy dissipated by a transistor when there is a one-step voltage of V across it is N times the energy when this voltage step presents N sub-steps [RP96]. The adiabatic computing becomes attractive only when the delay is not critical since power is traded for delay [ea94c].

As an example, consider the energy dissipation by the charging of a capacitance C through a transistor (modeled as a resistor R):

$$E_{diss} = \int_0^T P(t) dt = \int_0^T \frac{V^2}{R} e^{-\frac{2t}{RC}} dt = \frac{1}{2} CV^2 (1 - e^{-\frac{2T}{RC}}) ,$$

where V is the voltage swing at the capacitance. The same result is obtained for the discharging process.

The asymptotic energy dissipation, i.e. the limit as T approaches to infinity, is $\frac{1}{2} CV^2$, which draws a power consumption of $\frac{CV^2}{T}$ for a charge and discharge of the capacitance. This value coincides with the one obtained in Section 2.1.2.

It is important to point out that current, power and energy are exponential in time relative to the circuit's RC time constant. Current and energy asymptotically approach zero and $\frac{1}{2} CV^2$ respectively. Assuming that, for example, after $3RC$ the circuit has reached its final output value, the transition time could be doubled (to

$6RC$) and apply a voltage step of $V/2$ from 0 to $3RC$ and a voltage step of V from $3RC$ to $6RC$:

$$E_{diss} = \int_0^{3RC} P(t) dt + \int_{3RC}^{6RC} P(t) dt = \frac{1}{4} CV^2 .$$

Therefore, doubling the transition time T and halving the maximum voltage drop reduces the energy by a factor of two.

2.6 LAYOUT LAYER

The power consumption can be tackled at the layout layer during the floorplanning, placement and routing tasks.

The floorplanning task consists of the allocation of space on a chip for a given set of modules. A module may have its dimensions fixed or it may present different shapes. The goal is to find a suitable implementation for each module such that the total area is minimized while meeting a floorplan topology. Now, the power consumption adds a new constraint. A floorplanning for low power has to decrease the power by [CW94]:

- selecting the less power-consuming modules.
- distributing the power consumption rather than to have hot spots.

The traditional placement algorithm tries to minimize the wire capacitance among the different modules. The low-power placement algorithm has to decrease the switched capacitance [VP93].

The most well-known techniques for a low-power routing algorithm are:

- keep the high switching activity wires short.
- use the lower parasitic capacitance layer for the highest active wires.

Clock distribution

A particular critical signal to route is the clock. The clock signal has to drive large loads with fast rise/fall times. For example, in the case of the DEC Alpha chip [ea92] the clock load is $3.2 nF$, and this capacity has to be charged/discharged

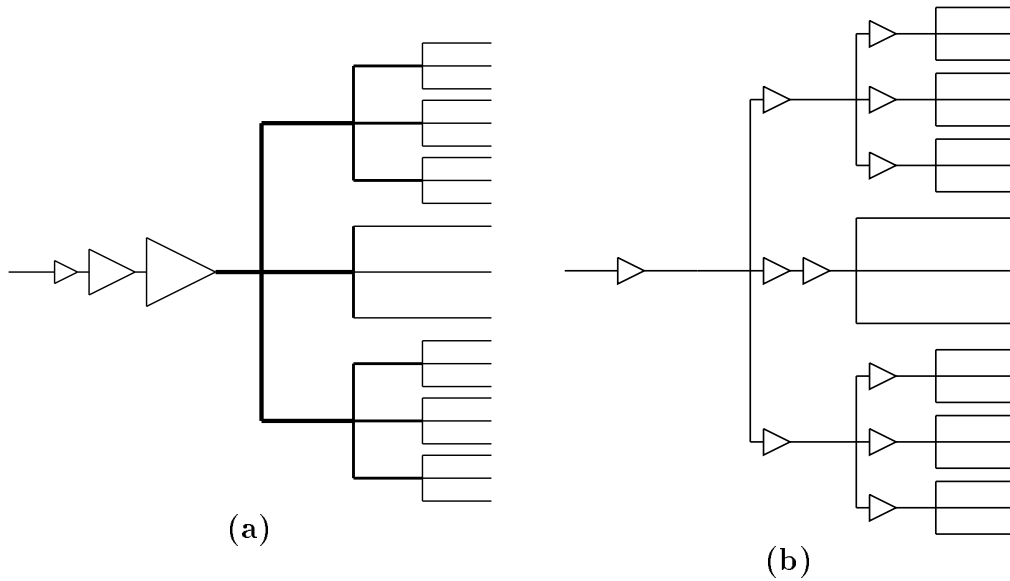


Figure 2.18 Clock tree driving schemes: (a) buffers at the clock source and (b) distributed buffers.

every 5 ns in rise/fall times of 0.5 ns. This draws a power consumption for the clock of 7 W, which corresponds to the 23% of the total power consumption of the chip (30 W).

Aside from using a low capacitance layer for routing the clock signal, low-swing drivers at the top level or at intermediate levels of the clock tree distribution may be devised [KaKS94]. This scheme reduces the power a 75% at the expense of an increase in delay.

To obtain the desired fast rise/fall times in the clock signal, buffers are needed to drive the large load capacitance. These buffers may be grouped at the clock source (Figure 2.18(a)) or may be distributed along the clock signal (Figure 2.18(b)). The first option has the advantage of avoiding the adjustment of intermediate buffer delays. The second has the advantage that relatively small buffers are used and they can be flexibly placed across the chip to save area layout. This option should be used because both wiring capacitance and driver power dissipation are reduced [RP96].

Circuit characteristic	Scale factor
Dynamic and static power consumption	$1/\delta^2$
Delay	$1/\delta$
Power-delay product	$1/\delta^3$
Gate area	$1/\delta^2$
Power density	1

Table 2.6 Technology scale factors for some parameters.

2.7 TECHNOLOGY LAYER

The equation 2.1 of power consumption in Section 2.1 showed that the three degrees for power reduction are: power supply, capacitance and activity. In the last sections the reduction of the activity has been targeted by the majority of the low-power techniques. At the technology layer, reducing power supply and capacitance is the main goal.

Three topics related to power consumption are addressed at the technology layer: technology scaling, SOI (Silicon-on-Insulator) process and MCMs (Multi-Chip Modules).

2.7.1 Technology scaling and SOI process

One straightforward way to drastically reduce the power consumption is through technology scaling. Let us suppose that the power supply and the dimensions of the transistors are scaled down by a factor δ ($\delta > 1$). Table 2.6 shows what happens to some parameters of the resulting circuit [WE93]. The power consumption is significantly reduced (although the power density is maintained since the integration level has increased by δ^2).

The scaling behavior of Table 2.6 does not apply when the technology is scaled below the half-micron. The delay is proportional to $\frac{1}{V_{DD}-V_{th}}$, where V_{th} is the threshold voltage of the transistor. When V_{DD} is near V_{th} , the delay increases dramatically. Therefore, to maintain the delay scaling by δ at below the half-micron, the threshold voltage of the transistor needs also to be scaled down. The problem of scaling down the threshold voltage is that the static power consumption caused by sub-threshold leakage currents is no longer negligible [BE95].

It is necessary, then, a proper technology process for low-power and low-voltage applications. One such technology is a class of Silicon-On-Insulator (SOI), names thin-film SOI, which presents a low cost and improvement of its performance at low voltages. The performance improvement of SOI is mainly caused by the reduction of parasitic capacitances and body effect [SNDD94].

2.7.2 Packaging technology

The off-chip capacitances are approximately one to three orders larger than those inside the chip. Therefore, the interconnections among the different chips may account for a significant part of the overall power consumption of the system.

To reduce the off-chip wiring capacitance, an obvious solution is to integrate as many chips as possible into a single module. These modules, called Multi-Chip Modules (MCM) are designed to accommodate 4 to 200 chips and combine the function of the single-chip package and the printed circuit board [TRDH89]. In an MCM, all the chips comprising the system are mounted on a single substrate. The major problem associated to MCM is the power supply distribution and it is caused by the material properties of the substrate.

2.8 LOW-POWER TECHNIQUES SUMMARY

The previous sections have reviewed some of the most well-known techniques for low power. In this section we give an overview of the power reductions obtained with these techniques. Table 2.7 shows, for the system, architecture, logic and circuit layers of the design process, some results obtained by the low-power research community. We have skipped the lower layers since they are out of the scope of this work.

The power reduction values reported on the table should not be superficially compared to decide which technique is more powerful. Among the reasons:

- the power reduction values have been mainly obtained by applying the techniques reported in Table 2.7. However, some other low-power approaches are applied together with the main technique to further reduce the power consumption.
- the type of comparison performed by the researchers. For example, larger power reductions may be obtained by a low-power scheduling technique if it is compared to an initial, poorly optimized register-transfer level description than if

Layer	Technique	Reductions	References
System	System shutdown	41%-99%	[CSB94, GIG ⁺ 94]
	System partitioning	61%	[Wol95]
	Algorithm selection	33%	[OY94]
Architecture	Voltage scaling with pipel./parallel processing	61%-80%	[CSB92a]
	Cache design	20%-85%	[BAF94, PR95] [KBN95]
	Data representation	21%-58%	[STD94]
	Compiler transformations	44%-94%	[CPM ⁺ 95, WCF ⁺ 94]
Logic	Path balancing	9%-38%	[BCH ⁺ 94]
	Logic synthesis	< 67%	[IP94, RP93] [TMA95, SGDK92]
	Technology mapping	< 47%	[LM93, TAM93] [TPD93]
	State assignment	< 66%	[TPCD94, HdlR94]
	Re-timing	< 16%	[MDG93]
	Disabling input registers	< 75%	[BFR94, AMD ⁺ 94]
Circuit	CPL vs. static CMOS	28%	[YYN ⁺ 90]
	Asynchronous vs. synchronous	-20%-80%	[KGG95, vBBK ⁺ 94]
	Asynchronous with adaptive supply voltage	29%-90%	[NS94]
	Adiabatic computing	77%	[SK94]

Table 2.7 Power reductions obtained by different techniques.

compared to an already optimized (for area or delay) description. In the first case, some power reductions may be obtained by simply applying the traditional techniques for low area and high performance.

Nevertheless, we believe that Table 2.7 provides an overview of the power reductions obtained at the system, architectural, logic and circuit layers of the design process. We observe that the highest power savings are obtained at the architecture and system layers. However, significant power reductions are also achieved at the logic and circuit layers, which add up to those already obtained at the higher layers.

2.9 CONCLUSIONS

The power-consumption sources in a CMOS circuit have been described in this chapter. Dynamic power consumption accounts for almost all the power in data-path circuits and, in general, it is the major source in the rest of the circuits.

Three degrees of freedom are considered when designing for low power: supply voltage, capacitance and activity. Once the supply voltage and the device dimensions

have been fixed, it is the switching activity of the signals of the circuit that ultimately determine its power consumption. Reducing the activity is the goal of the techniques proposed in the next chapters as the contribution of this work.

This chapter has also reviewed some state-of-the-art low-power design techniques covering all the layers of the design hierarchy ranging from the technology to the system layer. A more thorough insight has been given for the logic, circuit, architecture and system layers. The highest power savings are obtained at the architecture and system layers. However, significant power reductions are also achieved at the logic and circuit layers, which may add up to those already obtained at the higher layers.

HIGH-LEVEL SYNTHESIS TECHNIQUES FOR LOW POWER

Decisions taken at the earliest steps of the design of an electronic circuit may have a significant impact on the characteristics of the final implementation. This chapter illustrates how power consumption issues can be tackled at the system and architecture layer during the design of application specific integrated circuits (ASICs) in an embedded system scenario. A set of RTL transformations aiming at reducing power consumption are proposed and the potential benefits evaluated.

The common idea behind the transformations is to reduce the activity of the data-path functional units (e.g. adders, multipliers) by minimizing the switching activity of their input operands. Functional units highly contribute to the power consumption of the data-path. Preliminary evaluations obtained by simulation show that significant improvements can be achieved.

Finally, this chapter demonstrates how some of the presented transformations can be automated and incorporated in high-level synthesis tools.

3.1 INTRODUCTION

An increasing demand for devices executing complex tasks, such as notebook computing, video compression and speech recognition, has emerged. These tasks are often implemented with application specific integrated circuits (ASICs) that interact with general purpose processors and memories in an embedded system. In this chapter we focus on estimating and reducing the power consumption of those tasks that are mainly data driven such as, for example, digital signal processing applications.

We present a set of register-transfer level (RTL) techniques for power reduction, bearing in mind that decisions taken at the highest layers of the design process, i.e. architecture and system, can have a significant impact on the quality of the final implementation.

One of the main problems of proposing RTL techniques is the estimation of power consumption. At this level, there is a lack of information on the capacitance and the switching activity of the signals of the circuit. The first part of the chapter (Section 3.3) is devoted to propose power-consumption models for the functional units of the data-path, based on the activity of their operands. Rather than accurately estimating the power consumption of the final implementation, we focus on fairly compare the relative benefits of different RTL descriptions.

Section 3.4 presents five techniques for power reduction. Some of them have already been proposed by other authors but focusing on different targets (e.g. increasing data locality). We believe that these techniques can be also applied to reduce the power consumption of the synthesized design. All the techniques presented in this chapter aim at reducing the power consumption of the functional units by means of reducing the activity of the input operands. The benefits in power savings of the different techniques are evaluated.

Thus, the objective of this chapter is to help to the designer in deciding which specification of the circuit is more appropriate in terms of power consumption rather than fully implement all the techniques.

The following example illustrates the spirit of the techniques.

Assume the average energy dissipated by an array multiplier is 1 unit/operation. If two multiplications, say $a \times b$ and $c \times d$, are executed at consecutive cycles, the dissipated energy will be 2 units. On the other hand, if the second multiplication is again $a \times b$ (instead of $c \times d$), the switching activity of the multiplier will be null during the second multiplication and, therefore, no energy will be dissipated. Our

estimations also state that, if one of the operands remains unchanged during the second multiplication, say $a \times d$ (the first operand does not change), the average energy is 0.65 units.

Now assume the following two execution orders of a sequence of independent operations executed in the same multiplier:

	Order 1	Energy	Order 2	Energy
Cycle 1	$x = a \times b$	1.00	$x = a \times b$	1.00
Cycle 2	$y = c \times d$	1.00	$z = a \times d$	0.65
Cycle 3	$z = d \times a$	1.00	$y = c \times d$	0.65
<i>Total</i>		3.00		2.30

By applying simple RTL transformations to the first order (*Order 1*), a less power-consuming description can be obtained (*Order 2*). Two techniques have been applied in the example: *operand reordering* on the operation $z = d \times a$ (see Section 3.4.2) and *operand sharing* to swap the operations $y = c \times d$ and $z = d \times a$ (see Section 3.4.3). Thus, the multiplier sees no activity of its first operand between cycles 1 and 2 and of its second operand between cycles 2 and 3. A 23% power savings is estimated.

Each of the techniques presented in the chapter is evaluated with some benchmarks in which there is a clear evidence that significant benefits can be obtained. The techniques are complementary among them and not every technique produces a tangible improvement on every benchmark.

To cope with the high complexity of the problem of power reduction, all the techniques are based on heuristics, which trade, in some cases, power for area and performance.

Finally, some of the transformations for low power have been automated and incorporated in high-level synthesis algorithms and their complexity along with the hardware and delay overhead of the synthesized circuit are given. In particular, a list scheduling algorithm and a clique partitioning approach for register binding aiming at reducing power are presented and evaluated in Section 3.5.

3.2 PREVIOUS WORK

The efforts in higher levels of the design process for low-power design have been devoted to estimate the power consumption and to reduce it through transformations.

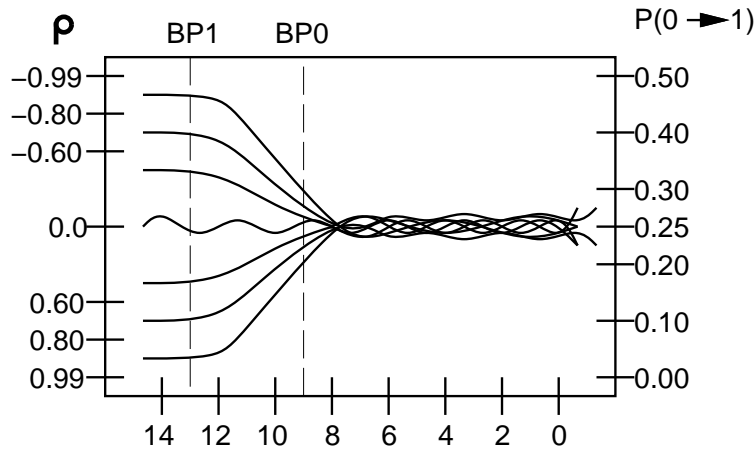


Figure 3.1 Activity of a 16-bit data stream for different temporal correlations (data approximated from [LR94a]).

3.2.1 Estimating power consumption

Landman has proposed two power-estimation models: in [LR94a], a model that accounts for the random behavior of the LSB bits and the correlated behavior of the MSB bits is presented for data-path circuits. Figure 3.1 shows the transition activity for several different two's complement data streams for each bit. Each curve corresponds to a different temporal (i.e. between successive data) correlation (ρ). When the data stream is white noise, $\rho = 0$ and $P(0 \rightarrow 1) = P(1 \rightarrow 0) = 0.25$. When the stream is highly correlated, $\rho \rightarrow \pm 1$. The MSB region corresponds to the sign bit and, consequently, the transition probabilities of these bits clearly depend on the correlation of the data stream. $\rho > 0$ corresponds to a lower activity for positively correlated signals; $\rho < 0$ corresponds to a higher activity for negatively correlated signals. Three bit regions are observed in Figure 3.1. The MSB and LSB regions present a constant transition activity. The middle region may be modeled by linear interpolation. Values BP0 and BP1 are determined from the data stream statistics. In [LR95], this model is extended to control-path circuits.

In [BB95], different processor models that account for the energy for the major modes of computation are described: fixed throughput, maximum throughput and burst throughput. Techniques for trading off throughput and energy consumption are derived for these modes.

Recently, entropy-based approaches at register-transfer level have been proposed in [DMP95, Naj95]. The idea is that the switching activity of a circuit can be

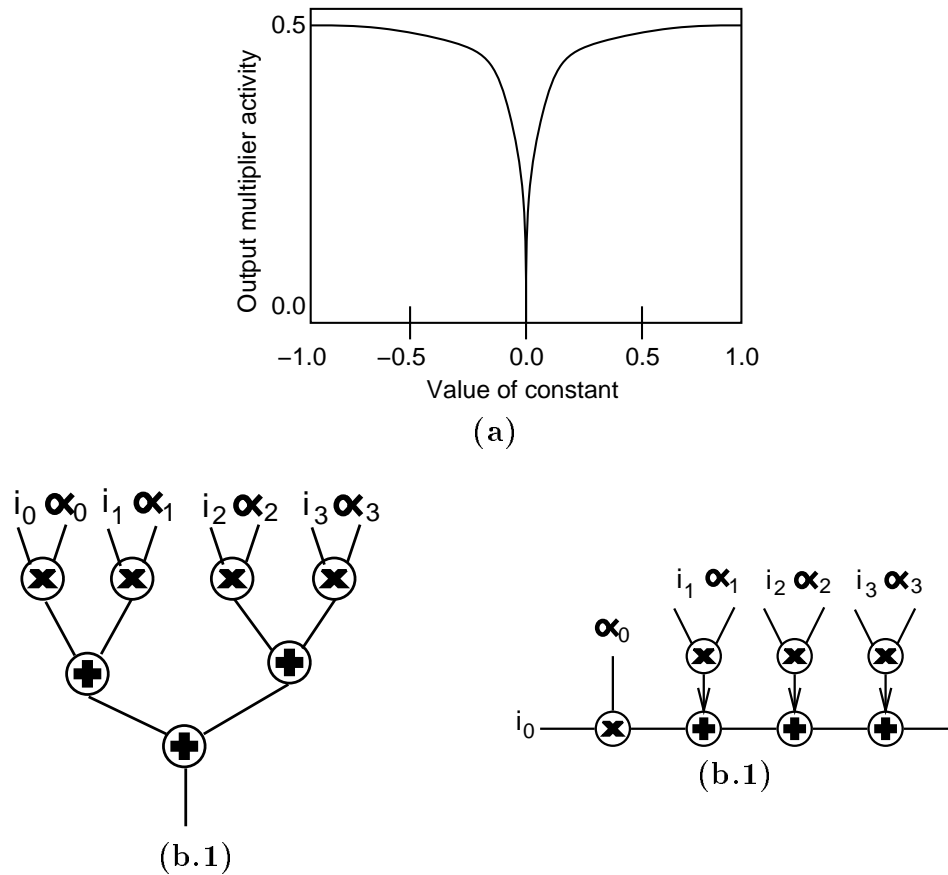


Figure 3.2 (a) Average activity in a multiplier as a function of the constant value (data approximated from [CR94]). (b) A parallel and serial implementations of an adder tree.

predicted without simulation using either entropy or informational energy averages; only the characteristics of the input sequence and some knowledge about the composition of the circuit are used to obtain power estimations.

3.2.2 Reducing power consumption

Few authors have addressed the set of transformations at the system and architecture layer to obtain lower-power designs. In [CR94], the power consumption of additions and constant multiplications as a function of the operand activity is studied. The average activity at the output of the multiplier monotonically increases with the absolute value of the multiplier constant (Figure 3.2(a)). The average activity at the output of an adder may be approximated by the maximum of the activities of the operands. From this study, a data flow graph transformation is derived for the typ-

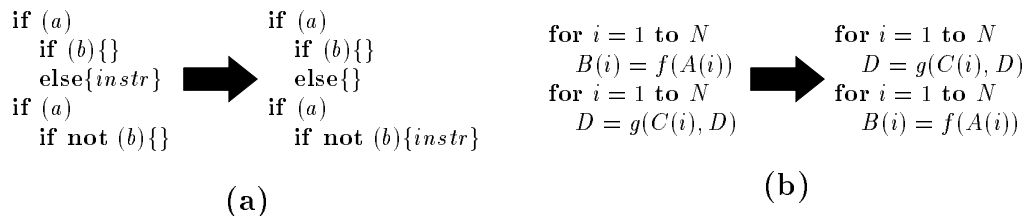


Figure 3.3 Transformations for reducing (a) activity at the address bus and (a) number of memory references.

ical operations in signal processing applications shown in Figure 3.2(b). Values α_i are constants. The conclusion is that the minimum average switching activity over all the modules (adders and multipliers) of the data-flow graph in Figure 3.2(b.1) is obtained when either $\alpha_0 \geq \alpha_1 \geq \alpha_2 \geq \alpha_3$ or $\alpha_0 \leq \alpha_1 \leq \alpha_2 \leq \alpha_3$. Similar results are obtained when the additions are performed serially (Figure 3.2(b.2)). In this case, the minimum activity is obtained when $\alpha_0 \leq \alpha_1 \leq \alpha_2 \leq \alpha_3$.

In [WCF⁺94], some memory transformations for low-power systems are hinted. The aim of these transformations is to reduce both the activity of the address lines and the number of off-chip references. One transformation that reduces the address transitions consists of reordering memory references so that few bits as possible toggle between successive addresses supplied to the memory. For example, the instruction *instr* in the left-hand side code of Figure 3.3(a) may be moved from the scope of the first *if* instruction to the scope of the second one (or vice versa) if the switching activity at the address bus is reduced. This transformation is only allowed if the instructions between the original and final position of the moved instruction are independent.

A transformation that reduces the amount of off-chip references consists also on reordering memory references so that locality is exploited. As an example, consider the left-hand side code of Figure 3.3(b). If we know that both the *A* and *C* arrays are available in a local memory because of previous operations and that only array *C* is not needed after the execution of the code, the references to array *B* in the first *for* loop may throw some values of array *C* out of the local memory, values that will have to be fetch again in the second *for* loop. This can be prevented if both loops are reordered as shown in the right-hand side code of Figure 3.3(b).

In [CPRB92], the traditional transformations for faster and smaller circuits are applied in order to evaluate the power consumption savings. Some of these transformations are the following: critical-path balancing and reduction, operation count reduction, optimization of data-path and control logic, operation substitution (e.g.

simple multiplications by shift-add operations) and bit-width optimization. Whenever the resulting circuit is faster than the required throughput, power-supply reduction can be applied to take advantage of its quadratic impact on consumption.

3.3 POWER-CONSUMPTION MODEL OF FUNCTIONAL UNITS

In high-level synthesis, a fast and accurate power-consumption estimation tool is necessary to narrow down the design process space and to finally obtain the design to be implemented. The power-consumption model described in Section 2.1 of Chapter 2 is not applicable to high-level synthesis because it would require the very time-consuming task of, first, mapping down each implementation to the gate level and, second, apply the model to obtain the power estimation.

Power consumption in the data-path accounts for a large fraction of the overall power budget. Among the different types of units, functional units (adders, multipliers, etc.) highly contribute to the power consumption of the data-path. For example, consider a Newton-Raphson divider based on the Newton-Raphson iterative algorithm. The power consumption of the functional units of the divider has been estimated to be approximately the 85% of the total power of the circuit whereas only a 15% is because of the registers, wire, control and multiplexers [LR95].

We use a high-level power-consumption estimation based on power profiles of different units, with special insight in functional units. It is possible to obtain a profile of the power consumption of each unit by simulating it and averaging the power consumed over many input samples. This method provides a fast way of estimating power consumption and it has been used previously by other authors [MK95, PC91, LR94a, KKR95].

Two power-consumption models of functional units will be described. Both consider the operand variability of the inputs of the functional unit. In one of the models, the *activity* of the operands is taken into account whereas the other model is based on the *repetition* of the operands.

Operand activity relates to the variability of the bit-pattern of one operand from one operation to the next. Power consumption is somehow related to the Hamming distance of consecutive bit-patterns. *Operand repetition* relates to the coarse-grained variability of the operand, i.e. the operand may or may not change between two consecutive operations.

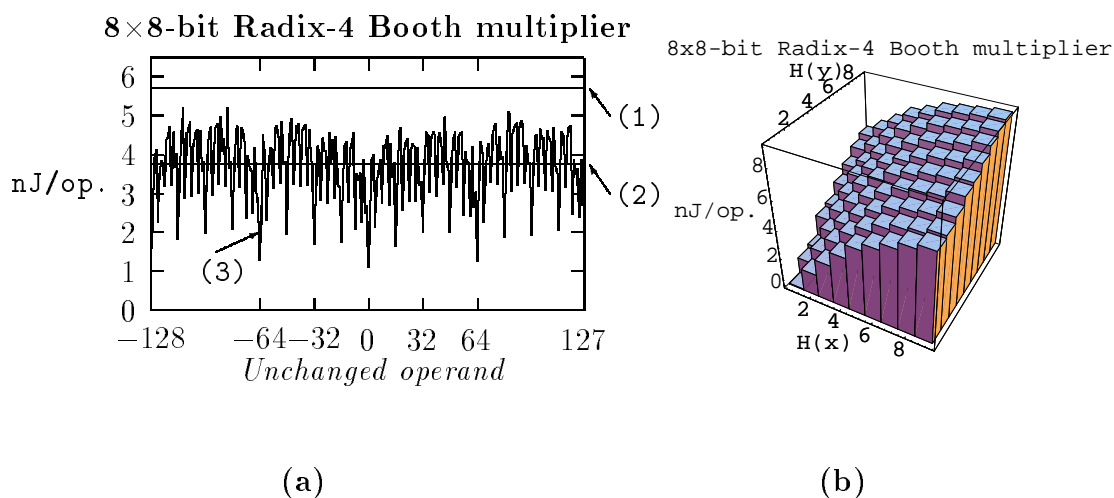


Figure 3.4 (a) Coarse-grained and (b) fine-grained power-consumption model for an 8×8 -bit Booth multiplier.

Henceforth, we will call *coarse-grained model* the model based on the operand repetition and *fine-grained model* the one based on the operand activity.

Since these power-consumption models target at data-path circuits, whenever we refer to the *power consumption* of a circuit we mean the *energy per operation* executed by that circuit. Data-path circuits have a fixed throughput and, therefore, the energy/operation metric is the most appropriate to quantify the energy efficiency for these type of circuits [BB95].

Coarse-grained model

To illustrate how this model works, consider Figure 3.4(a). Plot (3) represents the energy of an 8×8 -bit Booth multiplier [Kor93] in *nJ/operation* when one operand remains unchanged (x axis) with respect to the previous operation executed on the multiplier and the other operand varies randomly. Line (2) is the average energy of plot (3) and line (1) is the average energy when both operands vary randomly with respect to the previous operation. Comparing lines (1) and (2), we observe that the average power consumption of the multiplier is approximately 35% less when one operand remains unchanged. It is interesting to notice the influence of the bit-pattern for the unchanged operand over the power consumption (e.g. operands with many zeroes reduce the activity of the multiplier).

Factor	Definition	8-bit	12-bit	16-bit
α_{rca}	P_{rca1}/P_{rca2}	0.74	0.74	0.74
α_{cla}	P_{cla1}/P_{cla2}	0.74	0.75	0.75
α_{amul}	P_{amul1}/P_{amul2}	0.74	0.79	0.82
α_{booth}	P_{booth1}/P_{booth2}	0.65	0.65	0.68
$\beta_{rca/amul}$	P_{rca2}/P_{amul2}	0.1	0.049	0.029
$\beta_{rca/booth}$	P_{rca2}/P_{booth2}	0.04	0.025	0.016
$\beta_{cla/amul}$	P_{cla2}/P_{amul2}	0.152	0.074	0.056
$\beta_{cla/booth}$	P_{cla2}/P_{booth2}	0.061	0.039	0.027

Table 3.1 Factors of the coarse-grained model.

Absolute measures of the power consumption are very time-consuming in high-level synthesis whereas relative measures give satisfactory results. Power consumption estimations obtained with the coarse-grained model are relative to the number of operands that remain unchanged at the inputs of the functional units. Relative power estimations are expressed with the α and β factors shown in Table 3.1. The notation followed in this table is the following: P_{fu2} is the average energy spent by functional unit fu when its two operands vary randomly at each cycle; P_{fu1} is the energy spent when one operand remains unchanged and the other varies randomly; rca , cla , $amul$ and $booth$ stand for ripple-carry adder, carry-look-ahead adder, array multiplier and Booth multiplier respectively.

Factors α hardly change with the operand bit-width, but factors β clearly change because of the larger increase in power of the multipliers respect to the adders.

The factors of both models have been obtained by means of switch-level simulations (see Appendix A, Section A.3) of functional units implemented with the library cells provided with the Ocean system (see Appendix A, Section A.2).

As a simple example of how the coarse-grained model is applied, assume that the data-flow graph (DFG) shown in Figure 3.5(a) is a part of a larger DFG scheduled with two carry-look-ahead adders ($A1$ and $A2$) and two Booth multipliers ($M1$ and $M2$). Assume also an operand bit-width of 12 for all functional units.

Two schedules and functional-unit bindings for the DFG of Figure 3.5(a) are shown in Figures 3.5(b) and 3.5(c). The power consumption because of the schedule and binding in Figure 3.5(b) is expressed as $2 P_{cla2} + 2 P_{booth2} = 2 (1 + \beta_{cla/booth})$, whereas in the schedule and binding in Figure 3.5(c) it is expressed as $P_{cla2} + P_{cla1} + P_{booth2} + P_{booth1} = (1 + \alpha_{booth}) + \beta_{cla/booth} (1 + \alpha_{cla})$. Thus, the power reduction estimated

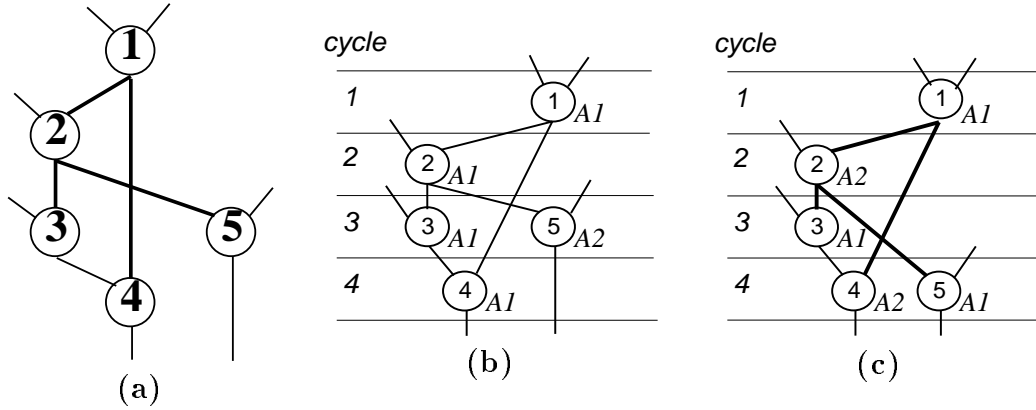


Figure 3.5 (a) Data-flow graph; (b) and (c) are two possible schedules and functional-unit bindings.

obtained with the schedule and binding in Figure 3.5(c) with respect to the one in Figure 3.5(b) is:

$$\frac{1 + \beta_{cla/booth} (1 - \alpha_{cla}) - \alpha_{booth}}{2 (1 + \beta_{cla/booth})},$$

which draws a power reduction of 17%.

The coarse-grained model provides a fast estimation of the power consumption when no information of the activity of the input data to the functional units is available. Power consumption highly depends on the input data activity. As an example, consider a 12×12 -bit Booth multiplier with one operand remaining unchanged and the other varying randomly (coarse-grained model). In this case, the multiplier observes an average of 6-bit changes on the varying operand. But if the varying operand presents an average of 4-bit changes, the power in the multiplier is reduced a 19%. Thus, whenever the information of the input data activity is known, a more accurate model should be used.

Fine-grained model

The activity of an operand can be measured with the Average Hamming Distance (AHD) defined as:

$$AHD(x) = \lim_{n \rightarrow \infty} \frac{\sum_{i=1}^n H(x_i, x_{i-1})}{n},$$

where: $H(p, q)$ is the Hamming distance between p and q ; x_i is the value of operand x in iteration i of the algorithm and n is the total number of iterations.

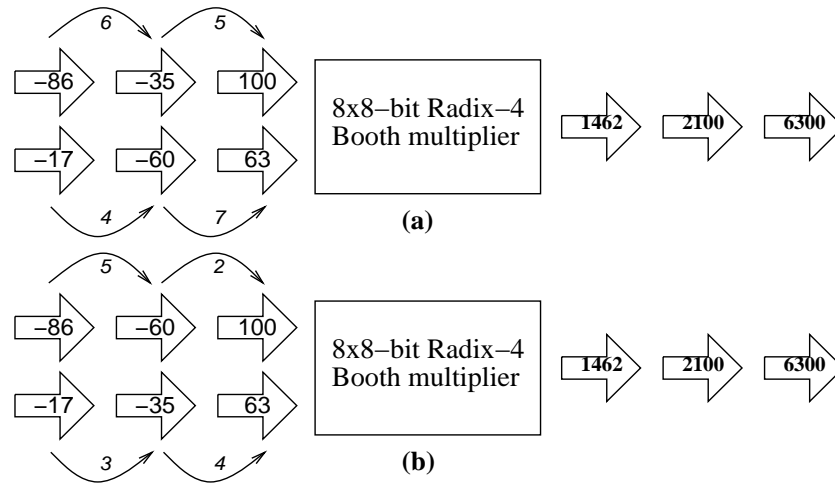


Figure 3.6 The input data order in (b.1) leads to a higher operand AHD than in (b.2). Big arrows indicate the input data. Thin arrows indicate the AHD between two consecutive input data.

Figure 3.4(b) illustrates the effect of the operand activity on the power consumption of an 8×8 -bit Booth multiplier. Obviously, power consumption tends to zero when the AHD of both operands tend to zero.

A simple example of how the fine-grained model is applied is shown in Figure 3.6(b). The multiplier in the figure has to perform three operations (Figure 3.6(b.1)). Taking advantage of the commutative property of the multiplication operation, a better input data order is obtained in Figure 3.6(b.2), where the AHD of both operands has decreased from 5.5 to 3.5. This decrease in the AHD of the operands makes the multiplier a 22% less power consuming when executing the last two operations.

3.4 REGISTER-TRANSFER LEVEL TECHNIQUES FOR LOW POWER

This section proposes a set of register-transfer level synthesis techniques that focus on reducing the power consumption of functional units. Implementation of some of these techniques is presented in Section 3.5.

The techniques proposed are summarized as follows:

- **loop interchange:** takes advantage of data locality to reduce the activity of the inputs of the functional units.
- **operand reordering:** seeks an appropriate operand order for commutative operations to reduce the switching activity.
- **operand sharing:** attempts to schedule and bind operations to functional units in such a way that the activity of the input operands is reduced.
- **operand retaining:** attempts to minimize the useless power consumption of the idle units and
- **operand similarity:** uses the information of the similarity among the operands in the scheduling and register-binding steps.

All the benchmarks used to evaluate the techniques have only additions and multiplications. The power reduction estimations obtained with these techniques are estimated as a function of α_{mul} , α_{add} and β with values α_{booth} , α_{cla} and $\beta_{cla/booth}$ respectively in Table 3.1 for 12-bit-wide functional units.

3.4.1 Loop interchange

The loop-interchange technique has been traditionally implemented in compilers to obtain dependency graphs with a higher degree of parallelism, to reduce stride or to increase data locality (and, thus, register reuse) [ASU86, AK84].

As an example, consider the algorithm in Figure 3.7(a.1) and its associated *dependence graph* in Figure 3.7(a.2). The compiler creates a dependence graph to capture the dependency information for a piece of code. Each node in the graph typically represents one statement. An arc between two nodes indicates that there is a dependence between the computation they represent. The dashed line shows the execution order of the iterations of the algorithm.

With algorithm in Figure 3.7(b.1), the execution order varies as it is shown in Figure 3.7(b.2). If matrix A is laid-out in memory in column-major form, execution order in 3.7(a.2) implies more cache misses than the execution order in 3.7(b.2). Thus, the compiler chooses algorithm 3.7(b.1) to reduce the running time.

Notice that in this example the loop interchange is a legal transformation since iterations in 3.7(b.2) are all executed after iterations they depend upon. But in general, the compiler has to verify that the transformation does not change the semantics of the program [Pug91, SFCM95].

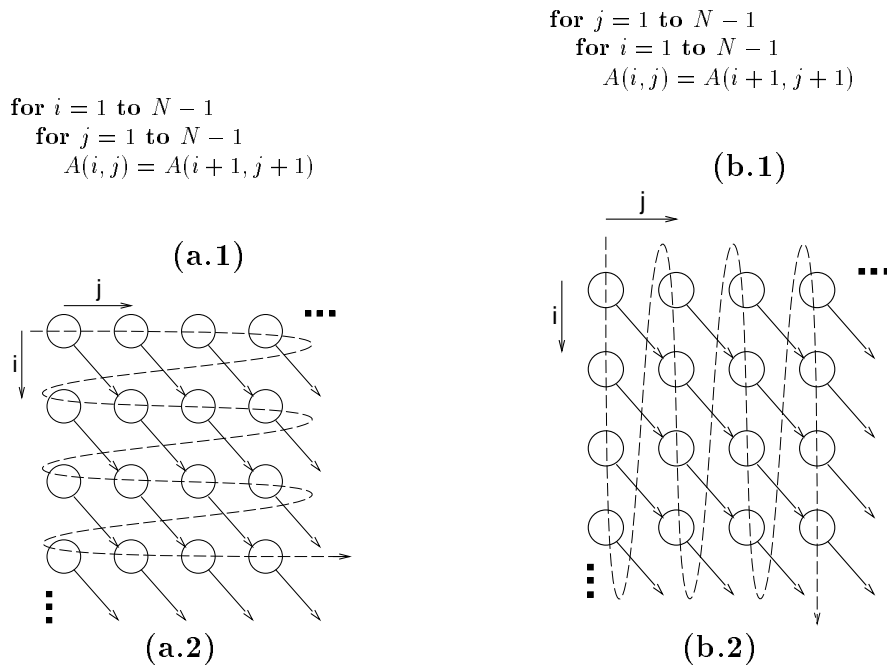


Figure 3.7 Example of application of the loop-interchange technique.

We apply loop interchange with the goal of minimizing the number of operand changes of the inputs of the functional unit. This technique will be applied to the *motion-estimation* algorithm for image compression [LK84] (Figure 3.8(a)) and to the matrix-vector product algorithm to illustrate its efficiency.

Application of loop interchange

In the algorithm of Figure 3.8(a) we observe three operations in the inner loop: absolute value, addition and subtraction. We assume the data-path depicted in Figure 3.9(a) for the execution of this algorithm. For simplicity, we will consider a subtraction to be the same as an addition in terms of power consumption.

The absolute value in 2's complement arithmetic is computed in two steps: (a) to check whether a number is negative and (b) complement the number and add 1 in this case. The first step represents negligible contribution to the total power consumption: just check if the MSB bit is one. In average, the second step will be executed half of the times.

In the algorithm in Figure 3.8(a) we also observe that: (1) both operands of the accumulation usually change with respect to the previous iteration of the algorithm

P, L : bit-length and bit-width of the current frame
 M, N : max. horizontal and vertical vector coord
 m, n : bit-length and bit-width of the current block
 CV, RV : current and reference image frame value
 CF, RF : current and reference frame
 $MV(g, h)$: motion vector of block (g, h)

```

for  $g = 0$  to  $\lceil \frac{P}{m} \rceil - 1$ 
  for  $h = 0$  to  $\lceil \frac{L}{n} \rceil - 1$ 
     $\Delta_{optimal}(g, h) = \infty$ 
    for  $i = -\lfloor \frac{M}{2} \rfloor$  to  $\lfloor \frac{M-1}{2} \rfloor$ 
      for  $j = -\lfloor \frac{N}{2} \rfloor$  to  $\lfloor \frac{N-1}{2} \rfloor$ 
         $\Delta_{part}(i, j) = 0$ 
        for  $k = 0$  to  $m - 1$ 
          for  $l = 0$  to  $n - 1$ 
             $CV = CF(m \cdot g + k, n \cdot h + l)$ 
             $RV = RF(m \cdot g + i + k, n \cdot h + j + l)$ 
             $\Delta_{part}(i, j) = \Delta_{part}(i, j) + |CV - RV|$ 
          if  $\Delta_{part}(i, j) < \Delta_{optimal}(g, h)$  then
             $\Delta_{optimal}(g, h) = \Delta_{part}(i, j)$ 
             $MV(g, h) = [i, j]^T$ 

```

(a)

```

for  $i = -\lfloor \frac{M}{2} \rfloor$  to  $\lfloor \frac{M-1}{2} \rfloor$ 
  for  $j = -\lfloor \frac{N}{2} \rfloor$  to  $\lfloor \frac{N-1}{2} \rfloor$ 
     $\Delta_{part}(i, j) = 0$ 
    for  $g = 0$  to  $\lceil \frac{P}{m} \rceil - 1$ 
      for  $h = 0$  to  $\lceil \frac{L}{n} \rceil - 1$ 
        for  $k = 0$  to  $m - 1$ 
          for  $l = 0$  to  $n - 1$ 
             $CV = CF(m \cdot g + k, n \cdot h + l)$ 
            for  $i = -\lfloor \frac{M}{2} \rfloor$  to  $\lfloor \frac{M-1}{2} \rfloor$ 
              for  $j = -\lfloor \frac{N}{2} \rfloor$  to  $\lfloor \frac{N-1}{2} \rfloor$ 
                 $RV = RF(m \cdot g + i + k, n \cdot h + j + l)$ 
                 $\Delta_{part}(i, j) = \Delta_{part}(i, j) + |CV - RV|$ 
             $\Delta_{optimal}(g, h) = \infty$ 
            for  $i = -\lfloor \frac{M}{2} \rfloor$  to  $\lfloor \frac{M-1}{2} \rfloor$ 
              for  $j = -\lfloor \frac{N}{2} \rfloor$  to  $\lfloor \frac{N-1}{2} \rfloor$ 
                if  $\Delta_{part}(i, j) < \Delta_{optimal}(g, h)$  then
                   $\Delta_{optimal}(g, h) = \Delta_{part}(i, j)$ 
                   $MV(g, h) = [i, j]^T$ 
             $\Delta_{part}(i, j) = 0$ 

```

(b)

Figure 3.8 (a) Motion estimation algorithm and (b) motion-estimation algorithm with two loop interchanges.

and (2) both operands of the subtraction inside the absolute value operator also

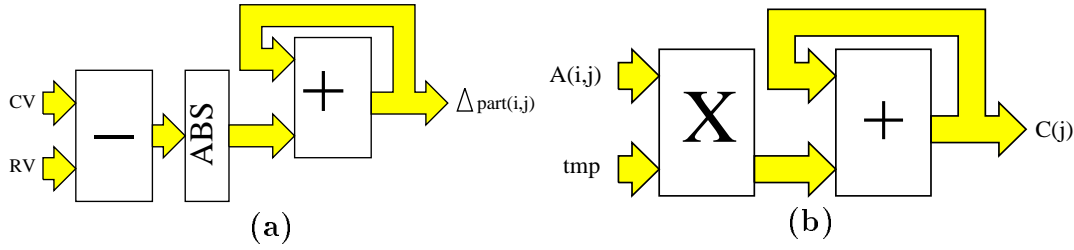


Figure 3.9 Data-path for executing (a) the motion-estimation algorithm and (b) the matrix-vector product algorithm.

change because both are fetched from memory in the inner loop (where the absolute value operation is executed).

But in the algorithm in Figure 3.8(b) we find out that one operand of the subtraction inside the absolute value operator remains the same during $M \times N$ iterations.

With the notation in Table 3.1, the power-consumption estimation of algorithm (a) is roughly:

$$P_a = PLMN \left(2 P_{add2} + \frac{P_{abs}}{2} \right),$$

and the power-consumption estimation of algorithm (b) is:

$$P_b = PLMN \left(P_{add2} + P_{add1} + \frac{P_{abs}}{2} \right).$$

We have estimated, by simulation, the average power consumption of the increment operation executed on an adder as $P_{abs} \approx 0.45 P_{add2}$. Thus, the estimated reduction factor on power consumption is:

$$R(\alpha_{add}) = \frac{1 - \alpha_{add}}{2.225},$$

obtaining a reduction of the power consumption of 11%.

Power consumption has only been estimated for the functional units of the data-path.

With algorithm (b), both the control logic and the off-chip reference order have changed. Of course, a wise use of local registers is expected to minimize off-chip references. This is important particularly in the motion-estimation algorithm, where

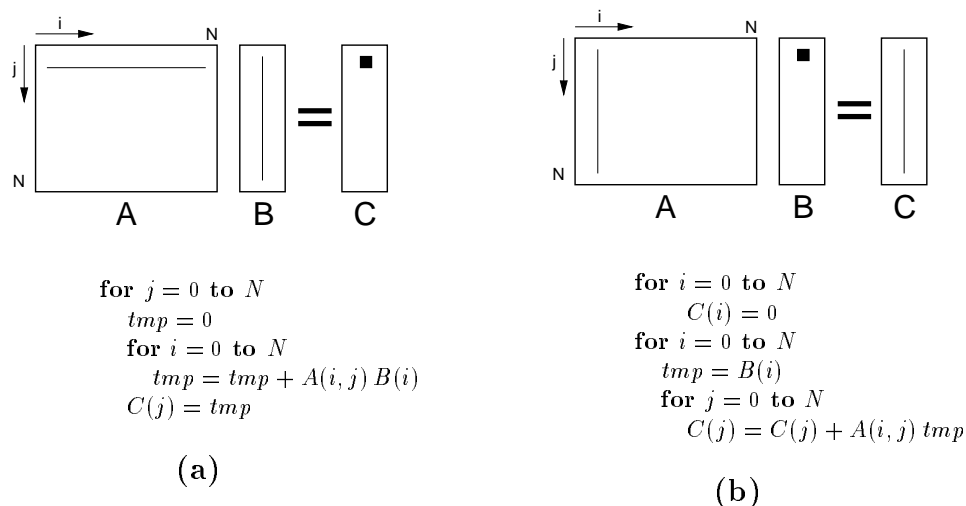


Figure 3.10 Matrix-vector product algorithm when referencing matrix A **(a)** by rows and **(b)** by columns.

its data working-set is considerably large. In order to minimize off-chip references, the most frequently referenced data can be stored in an internal cache. This implies that the algorithm must adapt its structure to the size of this internal cache to properly exploit data locality.

Moreover, the loop-interchange technique should be applied together with transformations that optimize the number of off-chip references [WCF⁺94]. Memory related power consumption is an important factor specially in multi-dimensional signal processing systems.

The loop interchange has also been applied to the matrix-vector product algorithm. Two different implementations of the algorithm can be devised depending on how the matrix is referenced: by rows (see Figure 3.10(a)) or by columns (see Figure 3.10(b)). If the matrix is referenced by rows, N^2 multiplications are needed with both operands not necessarily equal. But if the matrix is referenced by columns, only N of those multiplications are necessary; the rest have one operand fixed (which corresponds to the different values of the vector). This draws a power reduction of 33% when referencing the matrix by columns rather than by rows. We assume the data-path depicted in Figure 3.9(b) for the execution of this algorithm.

3.4.2 Operand reordering

The goal of this technique is to find an appropriate input operand order for commutative operators in such a way that switching activity is reduced. This will be achieved by reusing the same input operands to the operator among consecutive iterations of the algorithm implemented by the operator.

The operand-reordering technique can be implemented using existing techniques integrated in compilers for the manipulation of dependence graphs [ASU86].

In order to estimate its efficiency, this technique will be applied to the multiply-accumulate (MAC) operator and to the low-pass image-filter operator.

The MAC operator

Digital filters are basic components in digital signal processing systems. A typical substructure of a filter is the MAC operator, which performs the operation $\sum_{i=0}^{p-1} x_i y_i$, where p multiplications and $p - 1$ additions are executed.

One possible DFG of a 4-input MAC operator is shown in Figure 3.11(a). Three adders and four multipliers are used to implement this MAC operator. There are other ways to reorganize the additions, but the balanced structure of Figure 3.11(a) implies less power consumption [CPRB92].

Figure 3.11(b) shows a 4th-order LMS adaptive filter [TJL87]. In the LMS filter, and in some other digital filters (g.e. FIR and IIR filters), the MAC operator plays an important role and, therefore, minimizing its power consumption will decrease the total power consumption of the filter.

Application of operand reordering

For power consumption purposes, the MAC operator is classified into three cases: (a) both the x and y values change from one iteration to the next one (the general case); (b) either x or y values are constant and (c) either the x or y values of iteration i are the same as those of iteration $i - 1$ but shifted one position. The IIR and FIR filters follow cases (b) and (c). In the LMS filter, the MAC operator follows case (c).

In order to propose a better operand reordering for cases (a) and (b), the *activity* of the operands is taken into account whereas for case (c) the *repetition* of the operands (see Section 3.3) will determine the new operand reordering.

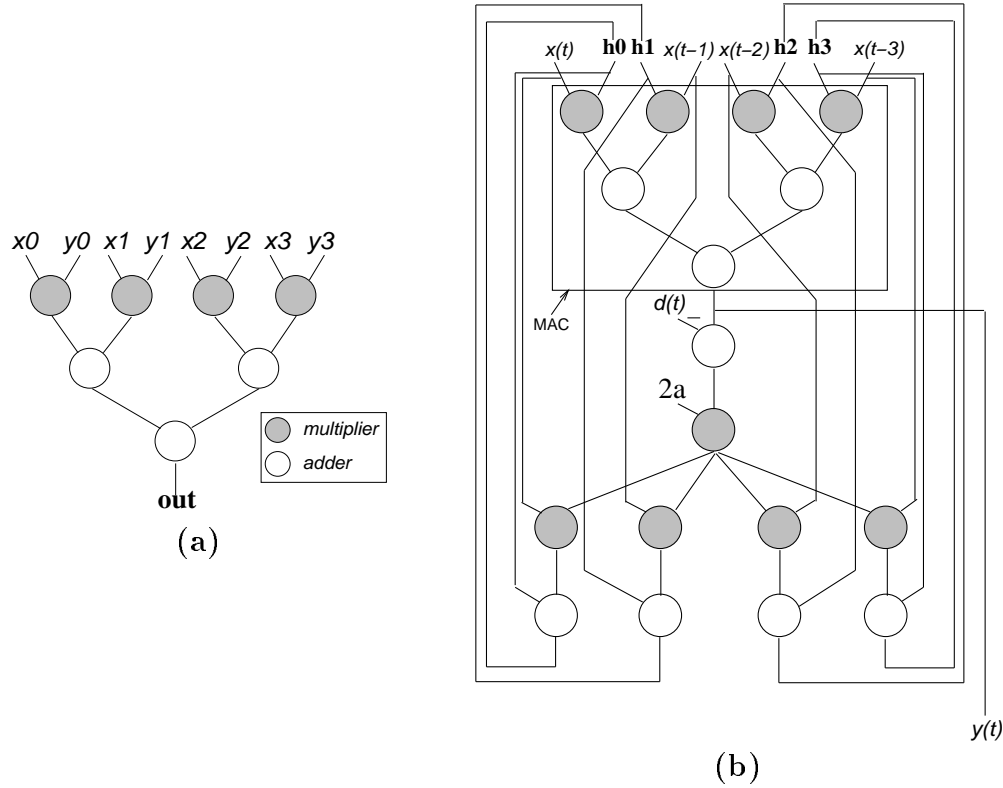


Figure 3.11 (a) MAC operator for $p = 4$ and (b) DFG of the 4th-order LMS adaptive filter.

Case (b) has been addressed in [CR94] and the conclusion is that the minimum average activity over all nodes of the balanced MAC operator is obtained when the constant operands (say the y values) satisfy $y_0 \leq y_1 \leq \dots \leq y_n$ or $y_0 \geq y_1 \geq \dots \geq y_n$.

We will focus on case (c). As previously explained, operand repetition will determine the new reordering. In the MAC operator of the LMS filter of Figure 3.11(b) we observe that all multipliers receive different operands at each iteration: the x values are shifted one position to the left and the first position is the new operand value; the h values are recalculated at each iteration and, therefore, are different. This fact is clearly shown in Table 3.2 (Reordering A).

Table 3.2 (Reordering B) shows a different operand reordering that takes advantage of the shift-wise behavior of the x values. With this new reordering, each multiplier will have one fixed operand (the x value) during four consecutive iterations.

iter.	Reordering A			
	M_0	M_1	M_2	M_3
i	(x_i, h_i^0)	(x_{i-1}, h_i^1)	(x_{i-2}, h_i^2)	(x_{i-3}, h_i^3)
$i+1$	(x_{i+1}, h_{i+1}^0)	(x_i, h_{i+1}^1)	(x_{i-1}, h_{i+1}^2)	(x_{i-2}, h_{i+1}^3)
$i+2$	(x_{i+2}, h_{i+2}^0)	(x_{i+1}, h_{i+2}^1)	(x_i, h_{i+2}^2)	(x_{i-1}, h_{i+2}^3)
$i+3$	(x_{i+3}, h_{i+3}^0)	(x_{i+2}, h_{i+3}^1)	(x_{i+1}, h_{i+3}^2)	(x_i, h_{i+3}^3)
$i+4$	(x_{i+4}, h_{i+4}^0)	(x_{i+3}, h_{i+4}^1)	(x_{i+2}, h_{i+4}^2)	(x_{i+1}, h_{i+4}^3)
$i+5$	(x_{i+5}, h_{i+5}^0)	(x_{i+4}, h_{i+5}^1)	(x_{i+3}, h_{i+5}^2)	(x_{i+2}, h_{i+5}^3)
$i+6$	(x_{i+6}, h_{i+6}^0)	(x_{i+5}, h_{i+6}^1)	(x_{i+4}, h_{i+6}^2)	(x_{i+3}, h_{i+6}^3)
$i+7$	(x_{i+7}, h_{i+7}^0)	(x_{i+6}, h_{i+7}^1)	(x_{i+5}, h_{i+7}^2)	(x_{i+4}, h_{i+7}^3)
iter.	Reordering B			
	M_0	M_1	M_2	M_3
i	(x_i, h_i^0)	(x_{i-1}, h_i^1)	(x_{i-2}, h_i^2)	(x_{i-3}, h_i^3)
$i+1$	(x_i, h_{i+1}^1)	(x_{i-1}, h_{i+1}^2)	(x_{i-2}, h_{i+1}^3)	(x_{i+1}, h_{i+1}^0)
$i+2$	(x_i, h_{i+2}^2)	(x_{i-1}, h_{i+2}^3)	(x_{i+2}, h_{i+2}^0)	(x_{i+1}, h_{i+2}^1)
$i+3$	(x_i, h_{i+3}^3)	(x_{i+3}, h_{i+3}^0)	(x_{i+2}, h_{i+3}^1)	(x_{i+1}, h_{i+3}^2)
$i+4$	(x_{i+4}, h_{i+4}^0)	(x_{i+3}, h_{i+4}^1)	(x_{i+2}, h_{i+4}^2)	(x_{i+1}, h_{i+4}^3)
$i+5$	(x_{i+4}, h_{i+5}^1)	(x_{i+3}, h_{i+5}^2)	(x_{i+2}, h_{i+5}^3)	(x_{i+5}, h_{i+5}^0)
$i+6$	(x_{i+4}, h_{i+6}^2)	(x_{i+3}, h_{i+6}^3)	(x_{i+6}, h_{i+6}^0)	(x_{i+5}, h_{i+6}^1)
$i+7$	(x_{i+4}, h_{i+7}^3)	(x_{i+7}, h_{i+7}^0)	(x_{i+6}, h_{i+7}^1)	(x_{i+5}, h_{i+7}^2)

Table 3.2 Two different input reorderings for the 4-input MAC unit. M_j represent the multiplications of the MAC unit.

The estimated power consumption of the MAC operator with reordering A after p iterations is:

$$P_A(\beta) = p(p P_{mul2} + (p-1) P_{add2}) = p P_{mul2}(p + \beta(p-1)),$$

and the estimated power consumption with reordering B after p iterations is:

$$P_B(\alpha_{mul}, \beta) = p(p P_{mul1} + (p-1) P_{add2}) = p P_{mul2}(p \alpha_{mul} + \beta(p-1)).$$

Thus, the estimated power-consumption reduction factor from reordering A to B is

$$R(\alpha_{mul}, \beta) = \frac{p(1 - \alpha_{mul})}{p(1 + \beta) - \beta} \approx \frac{1 - \alpha_{mul}}{1 + \beta},$$

achieving a 34% power-consumption reduction.

The operand-reordering technique has also been applied to the low-pass image-filter operator using 8 adders (its algorithm is shown in the next section). In this operator, the accumulation of 9 values of the image is the main operation. Two consecutive iterations of the algorithm have 6 out of the 9 input operands in common. Thus,

an input operand reordering can be devised where three of the nine adders will have the same operands during three consecutive cycles, which draws an estimated reduction in power of 27%.

Implementation of the operand reordering

An appropriate operand reordering can be obtained by manipulating (using existing compiler techniques [ASU86]) a graph (similar to a dependence graph) associated to the operator. In this graph, an arc between two nodes indicates that an input operand can be reused between the computations they represent. Let us call this graph an *input-reusing graph*.

To illustrate how a compiler can manipulate an input-reusing graph, consider again the MAC operator. The input reordering shown in Table 3.2 (Reordering A) corresponds to the input-reusing graph in Figure 3.12(a.1). The y -axis represents the iterations and the x -axis corresponds to the functional units being considered (in this case, the four multipliers of the MAC operator). A node in the graph corresponds to a multiplication.

The compiler has to obtain the best mapping of multiplications into the multipliers. This is accomplished by maximizing the number of input-reusing dependencies among the multiplications executed in each multiplier, i.e. by maximizing the number of vertical input-reusing dependencies. Figure 3.12(a.2) shows the optimum input-reordering which corresponds to the one represented in Table 3.2 (Reordering B).

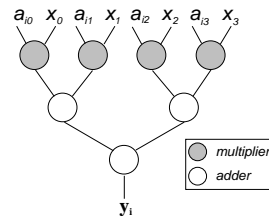
Another example is the matrix-vector product operator when the matrix has the symmetry property (see Figure 3.12(b.1)). Figure 3.12(b.2) shows the data-flow graph for one iteration of this operator using four multipliers and three adders. Again, we will focus on the multipliers for reducing power. Figure 3.12(b.3) shows a possible input-reusing graph with no input-reusing dependencies among the multiplications executed in each multiplier (i.e. all possible input-reusing dependencies are shown in dashed arcs, meaning that they are not achieved by the compiler). Figure 3.12(b.4) obtains an improved input reordering with three input-reusing dependencies.

The compiler can also obtain more input-reusing dependencies if another multiplier is available in the targeted architecture, as it can be observed in Figure 3.12(b.5).

To synthesize the input reordering obtained by the compiler, it is necessary to route the operands to the desired input of the functional unit. This implies, in some cases,

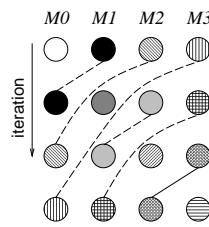
$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{01} & a_{11} & a_{12} & a_{13} \\ a_{02} & a_{12} & a_{22} & a_{23} \\ a_{03} & a_{13} & a_{23} & a_{33} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

(b.1)



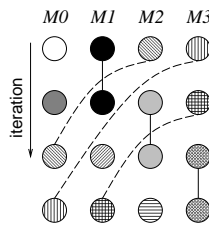
(b.2)

Multipliers



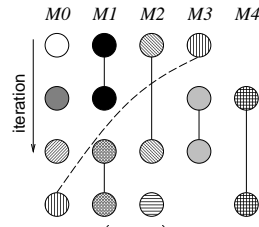
(b.3)

Multipliers

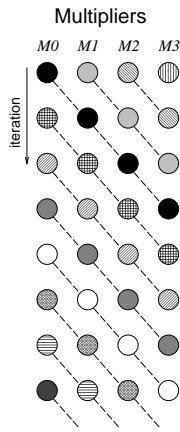


(b.4)

Multipliers

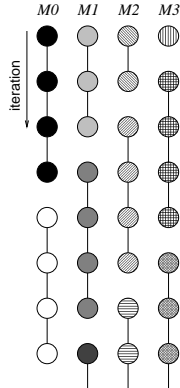


(b.5)



(a.1)

Multipliers



(a.2)

Figure 3.12 Different input-reusing graphs for: (a) the MAC operator and (b) the symmetric matrix-vector product operator.

an area overhead in terms of interconnection units and an increase in complexity in the control unit.

3.4.3 Operand sharing

The operand-sharing technique attempts to schedule and bind operations to functional units in such a way that the activity of the input operands is reduced.

Operations sharing the same operand are bound to the same functional unit and scheduled in such a way that the function unit can reuse that operand.

This technique is efficient when it is applied to a DFG with variables used by more than one operation. The AR filter [Kun84] will be used to illustrate this technique. The DFG of the AR filter is presented in Figure 3.13(a).

Figure 3.13(b) shows a possible schedule of the AR filter with two adders (one cycle) and one pipelined multiplier (two cycles). We observe that there are some operations which result in the input for more than one operation (thick lines in Figure 3.13(a)). For example, the result of addition 5 is an input for multiplications 10 and 11. Assume we bind multiplications 10 and 11 to the same unit U . Assume also that between the execution of multiplication 10 and 11 there is no other use of unit U . Then, one of the operands of unit U will not change from multiplication 10 to multiplication 11. Henceforth, we will call operand reutilization (OPR) the fact that an operand is reused by two operations consecutively executed in the same functional unit. In Figure 3.13(a), 4 OPRs can be potentially obtained in a multiplier.

An alternative schedule and unit binding with the same latency as the one in 3.13(b) is presented in Figure 3.13(c) with all 4 OPRs achieved. In the schedule and unit binding of Figure 3.13(b) no OPRs are obtained.

Thus, the estimated power consumption of one iteration in schedule (b) is:

$$P_b(\beta) = 12 P_{add2} + 16 P_{mul2} = P_{mul2} (16 + 12 \beta) ,$$

and the estimated power consumption of one iteration in schedule (c) is:

$$P_c(\alpha_{mul}, \beta) = 12 P_{add2} + 12 P_{mul2} + 4 P_{mul1} = P_{mul2} (12 + 4 \alpha_{mul} + 12 \beta) .$$

The estimated power-consumption reduction is:

$$R(\alpha_{mul}, \beta) = \frac{1 - \alpha_{mul}}{4 + 3 \beta} ,$$

achieving an 8.5% reduction.

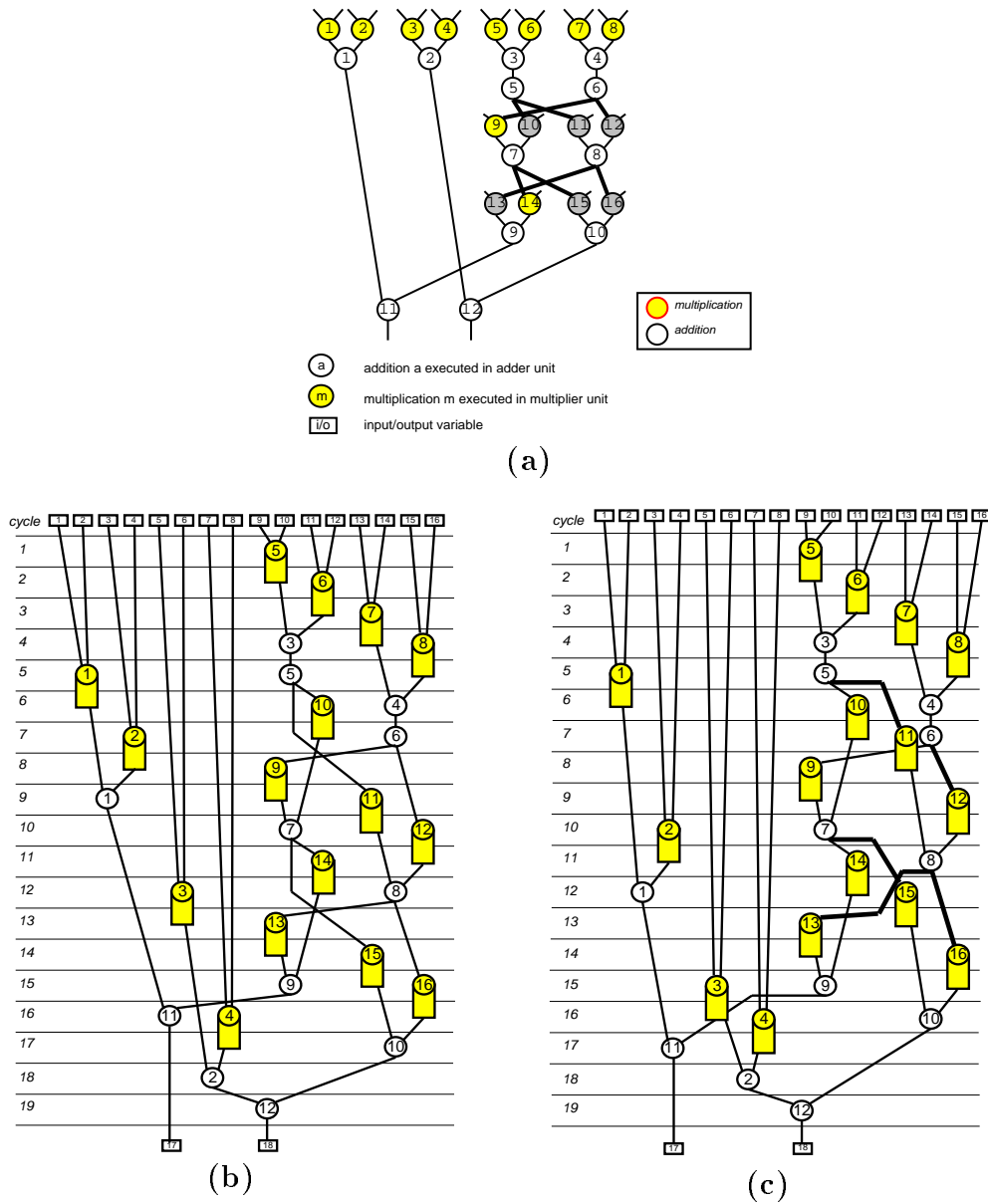


Figure 3.13 (a) DFG of the AR filter; (b,c) two possible schedule and bindings.

Application of loop unrolling for operand sharing

The operand-sharing technique is applied when some operations share the same operand in the same iteration of the algorithm. But it can also be applied even if operands feed more than one operation in different iterations. We just need to unroll the loop [BGS93]. The technique of loop unrolling replicates the body of a

<pre> for $i = 2$ to $N - 1$ $A(i) = A(i) + A(i - 1) A(i + 1)$ </pre> <p style="text-align: center;">(a)</p>	<pre> for $i = 2$ to $N - 2$ step 2 $A(i) = A(i) + A(i - 1) A(i + 1)$ $A(i + 1) = A(i + 1) + A(i) A(i + 2)$ </pre> <p style="text-align: center;">(b)</p>
--	---

Figure 3.14 (a) Original code and (b) after loop unrolling.

loop some number of times (unrolling factor u) and then iterates by step u instead of step 1. This transformation reduces the loop overhead, increases the instruction parallelism and improves register, data cache or TLB locality. Because of these properties, this technique has been implemented in compilers to optimize the code for performance.

Figure 3.14 shows an example of loop unrolling. The original code in Figure 3.14(a) is unrolled once in Figure 3.14(b). Value N is assumed to be even. Loop overhead is cut in half because two iterations are performed in each iteration. If array elements are assigned to registers, register locality is improved because $A(i)$ and $A(i + 1)$ are used twice in the loop body. Instruction parallelism is increased because the second assignment can be performed while the results of the first are being stored and the loop variables are being updated.

The low-pass image filter [Lim90] will be used to illustrate how the operand sharing and loop unrolling techniques can be combined for power optimization. A DFG of the low-pass image filter is shown in Figure 3.15(b). We see that no OPR is possible. But if we unroll the inner loop twice (the loop body contains now three iterations), the DFG of Figure 3.15(c), with 9 potential OPRs are possible, is obtained.

With one adder, the schedule of the DFG in Figure 3.15(c) has a 24-cycle latency and the one in 3.15(b), 8. Therefore, the total latency of the algorithm is the same in both schedules. Moreover, all 9 OPRs are achieved.

The estimated power-consumption reduction is now:

$$R(\alpha_{add}) = \frac{3}{8}(1 - \alpha_{add}),$$

obtaining a reduction of 9.4%.

There are some drawbacks in unrolling a loop that may diminish the power savings achieved on the functional units. First, the lifetime of the variables may increase. A side effect is the possible increase in the number of registers used. This is not a problem (since an increase in the number of registers does not imply an increase in

Benchmark	+/*	\otimes, \oplus	Power red.
<i>5th-order Elliptic filter [DDN85]</i>	26/8	$1 \otimes (2 \text{ cycles}), 2 \oplus (1 \text{ cycle})$	12%
<i>4th-order Daubechies filter [PTVF92]</i>	12/12	$1 \otimes (2 \text{ cycles}), 1 \oplus (1 \text{ cycle})$	21%
<i>SHARF [TJL87]</i>	11/12	$1 \otimes \text{p. } (2 \text{ cycles}), 2 \oplus (1 \text{ cycle})$	10%
<i>1-D 8-input Lee DCT [RY90]</i>	29/13	$2 \otimes (2 \text{ cycles}), 2 \oplus (1 \text{ cycle})$	6%
<i>1-D 8-input Chen DCT [RY90]</i>	26/16	$2 \otimes (2 \text{ cycles}), 2 \oplus (1 \text{ cycle})$	19%
<i>4 × 4 matrix multiplier</i>	4/8	$2 \otimes (2 \text{ cycles}), 1 \oplus (1 \text{ cycle})$	26%

Table 3.3 Results obtained by applying the operand-sharing technique.

power consumption if the variables of the algorithm are properly assigned to those registers - see Section 3.23) unless

- the number of registers in the targeted architecture is fixed and it is exceeded by the number of registers used. In this case, the contents of some of them have to be temporarily stored in memory and loaded back when needed (register spillage) and
- the number of registers in the targeted architecture is not fixed (we can choose it since we design the architecture) but there is an area constraint.

Second, the complexity of the control unit is increased because more cycles are needed to schedule the operations of the unrolled loop.

Application of the technique to other benchmarks

Table 3.3 shows the results obtained when applying the operand-sharing technique to other high-level synthesis benchmarks.

In all benchmarks except for the Elliptic filter, the results have been obtained by comparing the power-consumption estimation of the schedule with fewest achieved OPRs and the schedule with the largest number of achieved OPRs, having both schedules the lowest possible latency.

In the Elliptic filter we have detected a tradeoff between the speed and the consumption of the final design: it is possible to obtain a design with a longer latency (23 cycles vs. 21 cycles) but also with a higher number of achieved OPRs.

```

for  $i = 0$  to  $M$ 
  for  $j = 0$  to  $N$ 
     $out =$ 
       $(A[i-1][j-1]+$        $/* a0 */$ 
       $A[i-1][j]+$          $/* a1 */$ 
       $A[i-1][j+1]+$       $/* a2 */$ 
       $A[i][j-1]+$         $/* b0 */$ 
       $A[i][j]+$           $/* b1 */$ 
       $A[i][j+1]+$         $/* b2 */$ 
       $A[i+1][j-1]+$       $/* c0 */$ 
       $A[i+1][j]+$         $/* c1 */$ 
       $A[i+1][j+1])/9$     $/* c2 */$ 

```

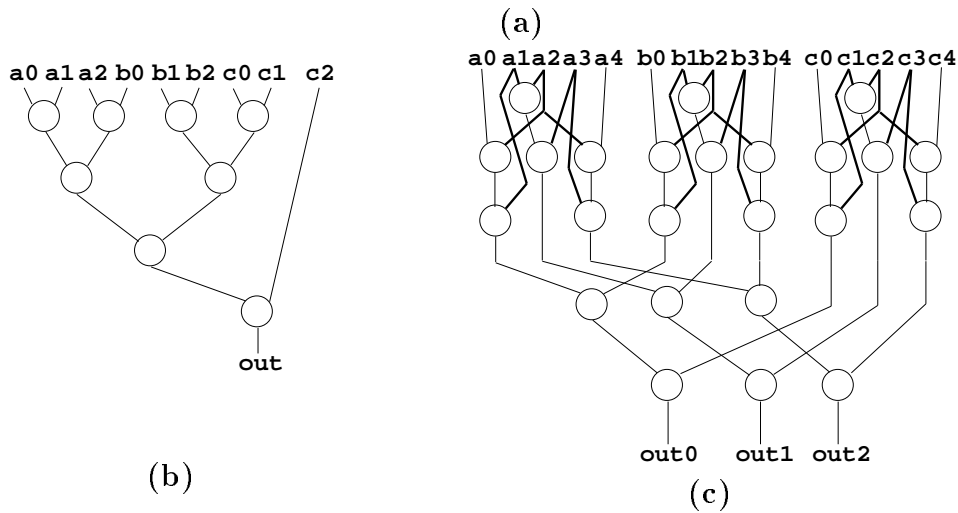


Figure 3.15 (a) Low-pass image filter algorithm; (b) DFG of the inner loop of (a) (without division by nine) and (c) DFG after loop unrolling.

3.4.4 Operand retaining

Not all resources of a data-path are always used during all cycles. Some remain idle when no operation is available for them. A functional unit consumes *useful* power when it is executing an operation and consumes *useless* power when there is an input operand variation while the functional unit is idle. One of the reasons for this possible variation of the input operands is that the control unit is usually synthesized using *don't care* values to minimize area or increase speed. Thus, an idle functional unit may have input operand changes because of the variation of the selection signals of multiplexers.

The operand-retaining technique attempts to minimize the useless power consumption of the idle functional units. Useless power is specially important in data-paths

	Benchmark	+/*	\oplus, \otimes	Idle time	Power reduc.
(1)	<i>AR filter [Kun84]</i>	12/16	1 p, \otimes (2 c), 1 \oplus (1 c)	27%	12%
(2)	<i>4th-order Daubechies filter [PTVF92]</i>	12/16	1 \otimes (2 c), 1 \oplus (1 c)	37%	9%
(3)	<i>1-D 8-input Lee DCT [RY90]</i>	28/13	4 \otimes (2 c), 3 \oplus (1 c)	40%	38%
(4)	<i>4 \times 4 matrix multiplication</i>	4/8	2 \otimes (2 c), 1 \oplus (1 c)	33%	15%
(5)	<i>unrolled low-pass image filter (3.13)</i>	24/0	4 \oplus (1 c)	25%	20%
(6)	<i>LMS adaptive filter [TJL87]</i>	8/9	2 \otimes (2 c), 1 \oplus (1 c)	45%	34%
(7)	<i>pixel interpolation [BS95]</i>	5/0	3 \oplus (1 c)	44%	34%
(8)	<i>5th-order Elliptic filter [DDN85]</i>	26/8	1 \otimes (2 c), 1 \oplus (1 c)	33%	21%

Table 3.4 Idle time spent by the functional units.

synthesized from *sparse* schedules. A schedule is said to be *sparse* if its unit occupation is relatively low. Table 3.4 presents the idle time spent by the functional units for some high-level synthesis benchmarks for a schedule with the functional units of the fourth column. The total number of operations for each benchmark is shown in the third column.

Some approaches to minimize the useless power consumption of idle units are: (a) with a proper register binding that minimizes the activity of the functional units (this technique is addressed in Section 3.4.5); (b) by wisely defining the control signals of the multiplexors during the idle cycles in such a way that the changes at the inputs of the functional units are minimized (this may result in defining some of the *don't care* values of the control signals) [RJ94] and (c) latching the operands of those units that will be often idle.

In this section, approach (c) is evaluated. It consists of the insertion of latches at the inputs of the functional units to store the operands only when the unit requires them. Thus, in those cycles in which the unit is idle no consumption is produced. The control unit has to be redesigned accordingly, in such a way that input latches become transparent during those cycles in which the corresponding functional unit must execute an operation.

The outcome of this technique in terms of power savings is somehow similar to the well-known technique of putting the functional units (or other larger parts of the circuit) to sleep when they are not needed through gated-clocking strategies [CSB94, BSdM94, AMD+94].

The operand-retaining technique has been evaluated to several benchmarks. For example, the least-mean square filter (LMS) has been scheduled with two multipliers (two cycles) and one adder (one cycle). In each iteration of this algorithm, the adder becomes idle during 8 cycles and the multipliers during 14 cycles.

The power consumption generated by the idle units (*useless* consumption) is:

$$P_{useless}(\alpha_{add}, \alpha_{mul}, \beta) = 8 P_{add1} + 7 P_{mul1} = 8 \alpha_{add} \beta + 7 \alpha_{mul} ,$$

and the power consumption because of the useful calculations (*useful* consumption) is:

$$P_{useful}(\alpha_{add}, \alpha_{mul}, \beta) = 8 P_{add2} + 9 P_{mul2} = 8 \beta + 9 .$$

The estimated reduction in power consumption is:

$$R(\alpha_{add}, \alpha_{mul}, \beta) = \frac{P_{useless}}{P_{useless} + P_{useful}} ,$$

achieving a reduction of 34%. This estimation does not take into account the power consumption overhead because of latches. For simplicity in the evaluation (and to avoid synthesizing every control unit), we have assumed that, in average, only one of the operands changes in each idle unit at each cycle. This assumption may be optimistic or pessimistic depending on the final implementation. A more realistic assumption would basically depend on the register binding done for the variables of the algorithm.

More results obtained by applying the operand-retaining technique are reported in the last column in Table 3.4.

3.4.5 Operand similarity

In the techniques previously presented, the main idea was to maximize the operand locality or, in other words, the *operand repetition* in the functional units. The operand-similarity technique takes into account the *operand activity*¹. This technique uses the information of the similarity among the variables and constants of the algorithm in the scheduling and register-binding steps.

We will show how the activity of the input operands affect the power consumption of the design. Two examples will be presented to illustrate this technique: a low-power schedule for a finite impulse response filter (FIR filter) [TJL87] and a low-power register binding for the Differential Equation Solver [GDWL92].

A profiling of the algorithm to be synthesized is needed in order to determine the similarity (measured with the AHD) among its variables and constants.

¹See Section 3.3 for the definition of *operand repetition* and *operand activity*.

To verify the effect of the input operand activity in power consumption, we have fed the LMS filter of Figure 3.11(b) with a random signal and with a waveform calculated as the sum of two sines. The AHD of both signals differ in 2.5 and this difference leads to a 22.6% power reduction when the LMS filter is fed with the sum-of-two-sines signal.

Example 1: scheduling of the FIR filter

A FIR filter follows the equation $\sum_{i=0}^{p-1} x_i c_i$ where c_i are constants.

When speed is not a major issue, a significant reduction in hardware complexity is achieved by performing multiplications over several clock cycles as a series of shift-add operations. When speed is important, the multiplications must be executed by multipliers. We will focus on this case and will show how a different multiplication execution order can influence over the final power consumption.

As an example, assume $p = 4$, the values -1870, 1867, -740 and -1804 for the constants c_0 to c_3 and a bit-width of 12. Assume also that the input data is a waveform calculated as a sum of two sines. If the DFG of the 4th-order FIR filter (Figure 3.16(a)) is scheduled with one multiplier and one adder, different minimum-latency schedules are possible with different multiplication execution order. In one of those schedules (Figure 3.16(b)), the multiplier observes the following changes in one of its operands (numbers over the arrows indicate the AHD between constants): $c_0 \xrightarrow{10} c_1 \xrightarrow{7} c_2 \xrightarrow{6} c_3 \xrightarrow{3} c_0 \xrightarrow{10} \dots$ (i.e. an AHD of 6.5) whereas in the another schedule (Figure 3.16(c)), it may observe the following changes: $c_0 \xrightarrow{10} c_1 \xrightarrow{11} c_3 \xrightarrow{6} c_2 \xrightarrow{7} c_0 \xrightarrow{10} \dots$ (i.e. an AHD of 8.5).

By means of switch-level simulations, the calculated power consumption for the functional units associated to the first schedule is 6.3% less than the one associated to the second. This reduction has been achieved only with the change of the schedule of two multiplications.

Example 2: register binding for the Differential Equation Solver

The experiments done before for the LMS filter of Figure 3.11(b) further showed that the AHD among the variables h_i is lower than among the other variables. The same occurs for other groups of variables.

This information can be used in register-binding algorithms to obtain a register set where activity of individual registers is minimized. As a side effect, those idle units that observe the changes in the registers will also reduce its consumption. Further-

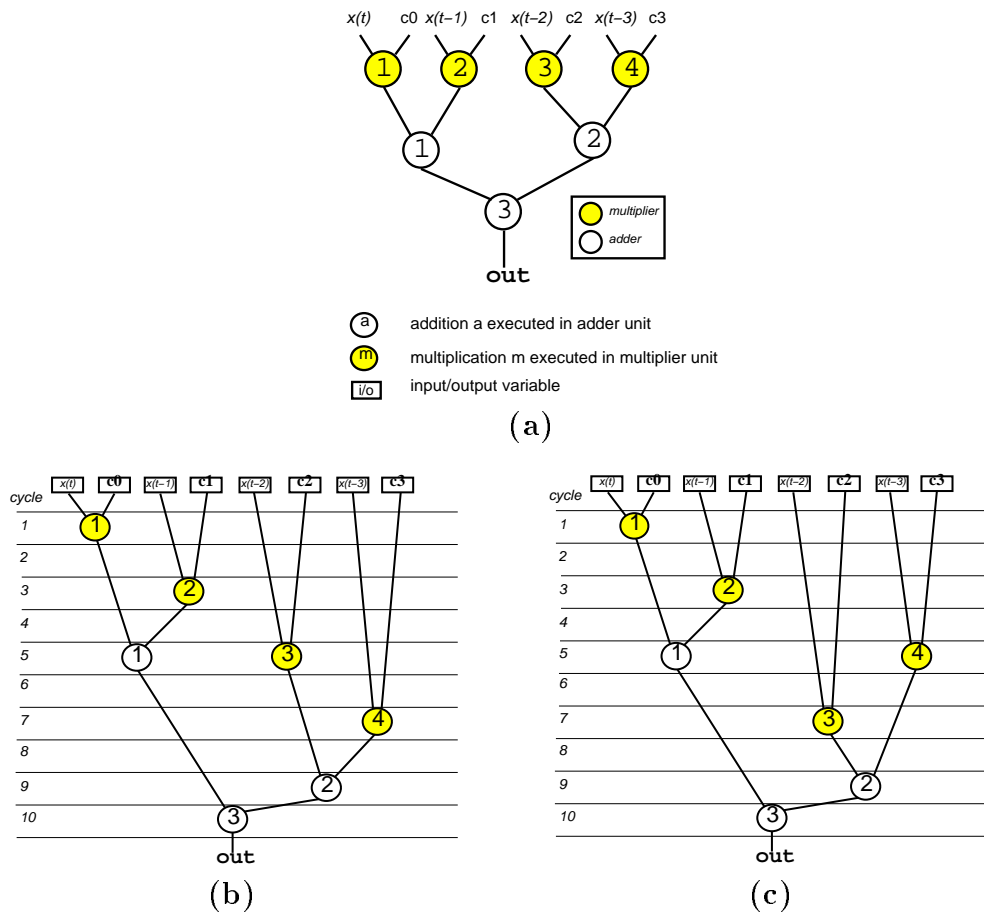


Figure 3.16 (a) DFG of the 4th-order FIR filter; (b,c) two possible schedule and bindings.

more, the operand-similarity information along with the commutative property of some operations can be used also to decrease the power consumption in the non-idle functional units by swapping their operands.

The Differential Equation Solver [GDWL92] data-flow graph (see Figure 3.17) has been scheduled with one adder (one cycle) and two multipliers (two cycles). The AHD between all pairs of variables has been obtained by means of simulations of the algorithm with different input data. The final AHD used has been obtained as the average of all simulations.

Two different register bindings (A and B) have been obtained. Both bindings use 5 registers. The reduction of the register activity of binding B produces an average power savings of 7.5% in the functional units with respect to A. This is obtained by

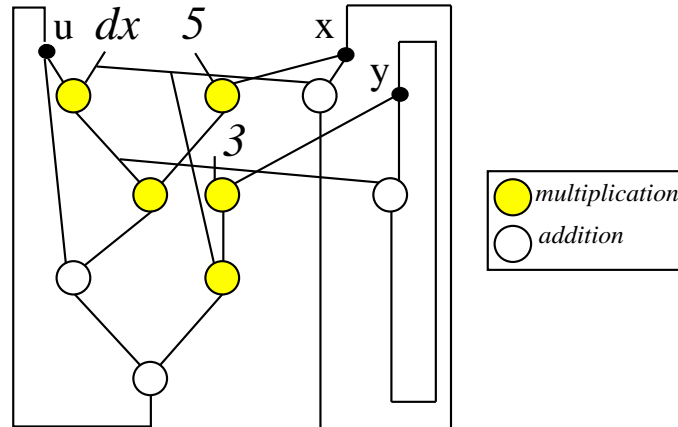


Figure 3.17 DFG of the Differential Equation Solver.

Technique	Applicability range
<i>Loop interchange</i>	Multi-dimensional signal processing (e.g. image processing)
<i>Operand reordering</i>	Descriptions with operations sharing input operands between consecutive cycles (e.g. digital filters)
<i>Operand sharing</i>	Descriptions with operations sharing input operands in the same cycle
<i>Operand retaining</i>	Descriptions where the functional units are idle a large fraction of the time
<i>Operand similarity</i>	Descriptions with a high similarity among the operands of the operations

Table 3.5 Features of the targeted circuit domains.

the reduction of the operand activity at the inputs of the functional units during the idle cycles.

3.4.6 Summary of targeted circuit domains

Not all the techniques produce a tangible improvement in all type of circuits. Table 3.5 shows the applicability range targeted by each technique.

All the techniques are complementary among them. There is only an exception: if the operand-retaining technique (i.e. the operands of the functional units are latched

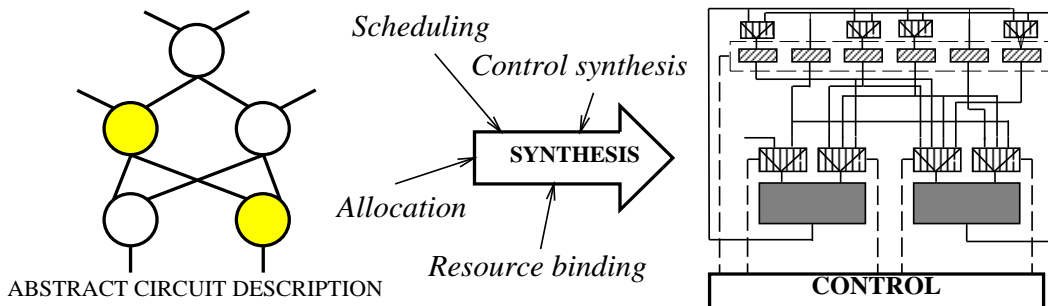


Figure 3.18 High-level synthesis process.

to avoid useless power consumption) is applied, then there is no need to use the correlation information to reduce the power during the idle cycles (see section 3.23).

The power savings that each of the techniques would obtain if applied alone to the same circuit do not add up when they are applied together to that circuit. For example, if both operand-reordering and operand-sharing techniques are separately applied to the same circuit and we observe an $A\%$ and $B\%$ power reduction respectively, we can not assume an $(A+B)\%$ power reduction when both techniques are applied together. If the operand-reordering technique is first applied, it may happen that the operand-sharing technique can not achieve a $B\%$ power reduction because the number of operations that share an operand may have been reduced as a consequence of the transformation performed by the operand-reordering technique. Therefore, the designer should first apply the techniques that obtain the largest power savings.

3.5 SCHEDULING AND REGISTER BINDING FOR LOW POWER

In this section, algorithms for the scheduling and register-binding steps of high-level synthesis are presented. These algorithms implement some of the concepts described in the last three techniques in Section 3.4, therefore targeting at the reduction of power consumption only in the functional units. They do not address the reduction of power in I/O, clocks or data transfers.

3.5.1 High-level synthesis

High-level synthesis is the design task of converting an abstract description of a digital system into a register-transfer level (RTL) design implementing that behavior (Figure 3.18). This abstract description may be composed of programs, algorithms, flowcharts, data-flow graphs, instruction sets, generalized FSMs, etc. The final RTL design consists of functional units (ALUs, multipliers), storage units (memories, register files) and interconnection units (multiplexers, buses).

The high-level synthesis system compiles the abstract description into an internal representation. All synthesis tasks work from this representation. A data-flow graph is used to represent descriptions with a large number of arithmetic operations. To represent reactive or embedded systems, in which the control sequence is based on external conditions, the DFG is extended with the control flow (CDFG).

The high-level synthesis process is traditionally decomposed into four different yet interdependent tasks [GR94]: allocation, scheduling, binding and control synthesis.

The *allocation* task determines the type and quantity of resources used in the RTL design. It also determines the clocking scheme, memory hierarchy and pipelining style. The goal of allocation is to make appropriate trade-offs between the design's cost and performance. To perform the required trade-offs, the allocation task must determine the exact area and performance values. The number of functional units is a first approximation of the cost in area. The number of control steps indicates the performance.

The *scheduling* task schedules operations and memory references into clock cycles. If the number of clock cycles (number of resources) is a constraint, the scheduler has to produce a design with the fewest functional units (fewest clock cycles).

The *binding* task assigns operations and memory references within each clock cycle to available hardware units. A resource can be shared by different operations if they are mutually exclusive, i.e. they will never execute simultaneously.

Furthermore, the binding task is decomposed into functional, storage and interconnection unit steps, all of them tightly related to each other. They are usually ordered and executed sequentially because of the high complexity of the binding task.

The final task, *control synthesis*, consists of reducing and encoding states and deriving the logic network for next-state and control signals in the control unit.

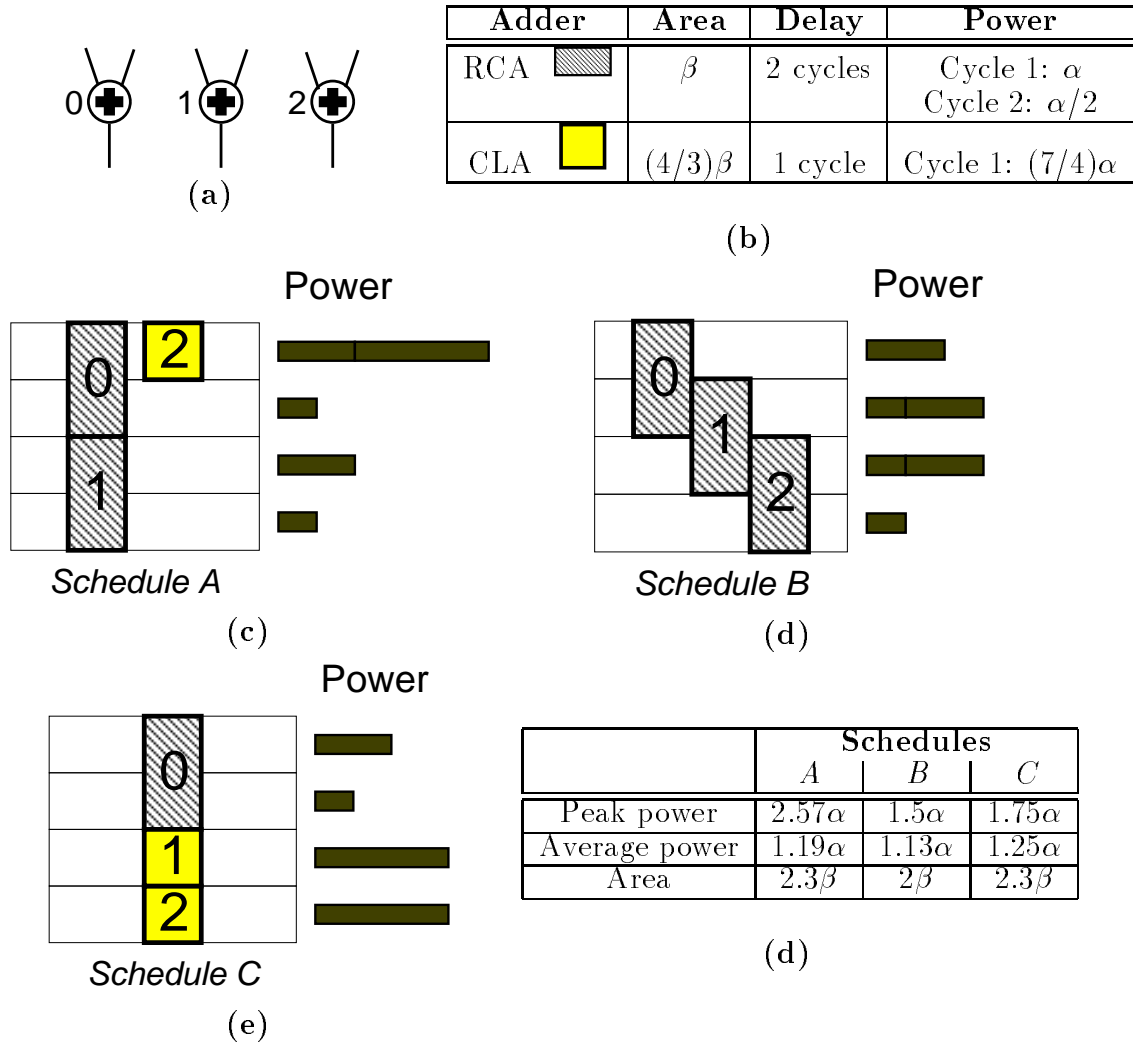


Figure 3.19 Example of peak-power and average-power optimization during the scheduling and functional-unit binding tasks; (a) DFG; (b) adder characteristics; (c-e) different schedules and bindings and (d) results.

3.6 PREVIOUS WORK ON HIGH-LEVEL SYNTHESIS FOR LOW POWER

High-level synthesis for low power has been addressed in [Mar95, RJ94, DK95, CP95, KKR95, RJ95].

In [Mar95], the scheduling and functional-unit binding tasks are applied to reduce the average and the peak-power consumption. Peak power reduction is important to avoid sudden high currents that may cause voltage drops, electron-migration

and other failure mechanisms. A simple example is shown in Figure 3.19. Figure 3.19(a) shows the data-path to be synthesized. Figure 3.19(b) presents the characteristics (area, cycles and average power) of the two types of adders that can be used (the power values are a simplification of the values presented in [Mar95] for the sake of clarity). The ripple-carry adder (RCA) presents more power in the first cycle because there is activity throughout it from the very beginning. In the second cycle, the least-significant part has stabilized and only there is activity in the most-significant part because of carry propagation. Therefore, the average power consumption in the second cycle is less. Figures 3.19(c) to 3.19(e) show different schedules with the same delay (4 cycles). We observe in Table 3.19(d) that *Schedule B* presents both the lowest peak and average power.

In [RJ94], an scheduling method that attempts to reduce the transition activity of the synthesized design is presented. The scheduling approach is driven by the activity at the inputs of the functional units; it selects a sequence of operations (variables) for a module (register) such that the transition activity is reduced. The same authors present in [RJ95] a linear integer programming formulation for the problem.

Moreover, in [RJ94] a method for optimizing the control is described. The controller generates the control signals (load signals for registers, select signals for multiplexers). The inputs to functional units and registers during idle periods do not affect the behavior of the circuits, i.e. they are *don't cares* that are used to prevent the load of a register in those cycles it is idle. The *don't cares* can also be used to select the registers with least activity for the idle units, so that the useless power consumption is minimized. Power reductions applying both scheduling and controller optimization for low power range from 5 to 33%. The study in [RJ94] is the closest work to the one presented in this chapter as contribution.

In [DK95], a scheduling and binding technique for reducing the activity in the buses is described. This is accomplished by properly multiplexing the data transfers onto the buses. Busses are high-capacitance signals; therefore, reducing their activity implies decreasing significantly the power of the RTL design. An average reduction of 45% is achieved for three well-known high-level synthesis benchmarks at the expense of a very high average increase in delay of 31%.

In [CP95], a similar register binding algorithm to the one presented in this chapter as contribution is described. It is also formulated as a clique covering of a compatibility graph. Unfortunately, no power reduction results are reported for a comprehensive set of benchmarks.

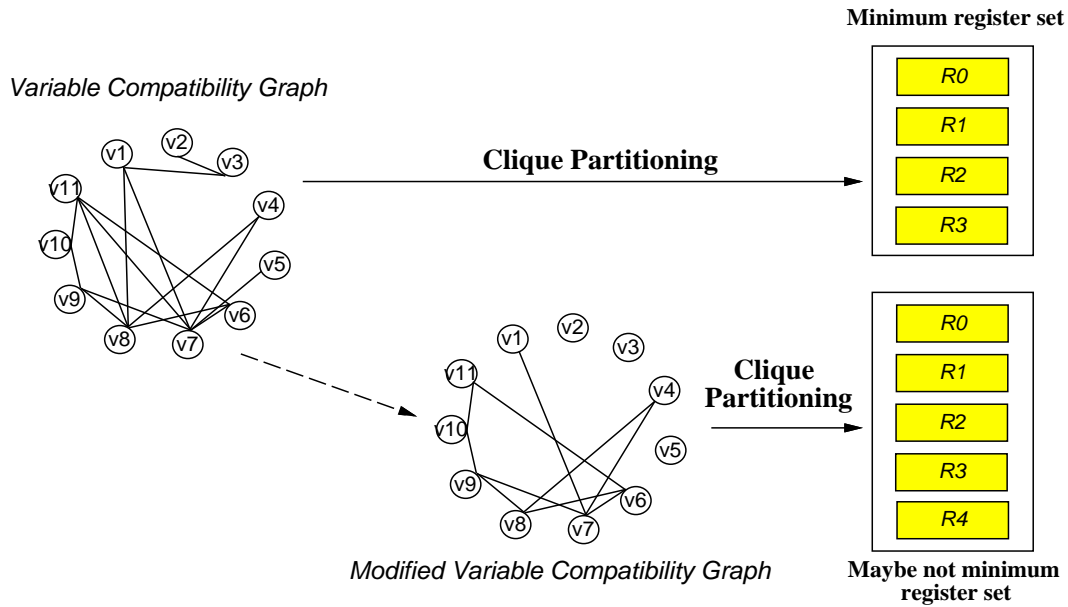


Figure 3.20 Register binding for low power.

Other works related to high-level synthesis for low power are found in [KKRV95, RJ95].

3.6.1 Overview

Two traditional approaches for the scheduling and register-binding tasks modified to obtain lower-power designs are presented in the following sections.

The scheduling algorithm for low power uses a list-scheduling approach where the priorities of the operations of the ready-operation queue are set in such a way that operations sharing the same operand are bound to the same functional unit and scheduled so that the functional unit can reuse that operand.

The register-binding algorithm for low power is based on a clique partition of a restricted variable-lifetime compatibility graph to obtain a register set that, for each functional unit, reduces the power consumption during idle cycles. A drawback of this algorithm is that larger register sets may be obtained (Figure 3.20). Power consumption in the functional units during non-idle cycles is decreased by taking into account the AHD among the variables of the behavioral description and the commutative property of some operations.

Although the scheduling and register-binding techniques for low power are compatible and complementary, the scheduling technique will obtain better improvements if applied to dense schedules (e.g. schedules where the functional unit occupation is high) whereas the register-binding technique is more suitable to sparse schedules.

3.6.2 Scheduling for low power

The goal of the scheduling algorithm for low power is to increase the potential for a functional unit (FU) to reuse an operand. Henceforth, we will call operand reutilization (OPR) the fact that an operand is reused by two operations consecutively executed in the same FU. The OPR concept has already been explained in Section 3.4.3.

A traditional list-scheduling algorithm (LS) will be compared with a modified list-scheduling targeting at low power (LPLS). In both cases, the FU-binding task is based in a clique partition of a variable-lifetime compatibility graph but it differs in the number of edges of this graph as will be shown later.

LPLS also trades off latency for OPRs: assume that an operation happens to be in the critical path of the DFG but it is scheduled later than indicated by LS because an OPR is then achieved. In this case, the latency of the synthesized circuit has increased.

List-scheduling algorithm

The list-scheduling algorithm [GDWL92] is a popular method for scheduling operations when the number of resources is a constraint. The algorithm is shown in Figure 3.21.

The list-scheduling algorithm maintains a priority list of ready operations. An operation is ready if it has all predecessors already scheduled. During each iteration of the algorithm (which corresponds to a control step) the operations in the beginning of the ready list are scheduled till all the resources get used. The priority list is always sorted with respect to a priority function. One such a function is the mobility range, which indicates in how many different cycles the operation may be scheduled considering no restriction on the number of resources. A smaller value of mobility indicates a higher urgency for scheduling an operation since the schedule will run out of alternative control steps earlier. The mobility range can be easily calculated with an ASAP and ALAP scheduling (refer to [GDWL92] for more details). Thus, the priority function resolves the resource usage among the ready operations; the

V is the set of operations.
 $PList_{t_k}$ is the priority list for each operation type $t_k \in T$.
 C_{step} is the current control step.
 m is $|T|$.
 N_{t_k} is the number of FUs performing operations of type t_k .
 $S_{current}$ is the current schedule.

```

INSERT_READY_OPS ( $V, PList_{t_1}, PList_{t_2}, \dots, PList_{t_m}$ );
 $C_{step} = 0$ ;
while (( $PList_{t_1} \neq \emptyset$ ) or ... or ( $PList_{t_m} \neq \emptyset$ )) do
   $C_{step} = C_{step} + 1$ ;
  for  $k = 1$  to  $m$  do
    for  $funit = 1$  to  $N_k$  do
      if  $PList_{t_k} \neq \emptyset$  then
        SCHEDULE_OP ( $S_{current}, FIRST(PList_{t_k}), C_{step}$ );
         $PList_{t_k} = DELETE(PList_{t_k}, FIRST(PList_{t_k}))$ ;
      endif
    endfor
  endfor
  INSERT_READY_OPS ( $V, PList_{t_1}, PList_{t_2}, \dots, PList_{t_m}$ );
endwhile
  
```

Figure 3.21 List-scheduling algorithm.

operation with higher priority gets scheduled. Operations with lower priority will be deferred to the next or later control steps.

The algorithm uses a priority list $PList$ for each operation type ($t_k \in T$). These lists are denoted by the variables $PList_{t_1}, PList_{t_2}, \dots, PList_{t_m}$. The operations in these lists are scheduled into control steps based on N_{t_k} which is the number of functional units performing operation of type t_k . The function `INSERT_READY_OPS()` scans the set of operations, V , determines if any of the operations in the set are ready, deletes each ready operation from the set V and appends it to one of the priority lists based on its operation type. The function `SCHEDULE_OP($S_{current}, o_i, s_j$)` returns a new schedule after scheduling the operation o_i in control step s_j . The function `DELETE($PList_{t_k}, o_i$)` deletes operation o_i from the specified list.

Initially, all operations that do not have any predecessor are inserted into the appropriate priority list by the function `INSERT_READY_OPS()`, based on the priority function. The *while* loop extracts operations from each priority list and schedules them into the current step until all the resources are exhausted in that step. Scheduling an operation in the current step makes other successor operations ready. These ready

```

ASS = CREATE_ALL_SS (V);
INSERT_READY_OPS (V, PListt1, PListt2, ..., PListtm);
Cstep = 0;
while ((PListt1 ≠ ∅) or ... or (PListtm ≠ ∅)) do
  Cstep = Cstep + 1;
  for k = 1 to m do
    UPDATE_PRIORITIES (PListtk);
    while PListtk ≠ ∅ do
      op = FIRST (PListtk);
      if IS_OSS (ASS, op) then
        ifnot SS_HAS_RESERVED_FU (SS) then
          funit = GET_FREE_AND_NOT_RESERVED_FU (SS);
          RESERVE_FU_IN_SS (SS, funit);
        endif
        schedule_operation = TRUE;
      else
        funit = GET_FREE_AND_NOT_RESERVED_FU (SS);
        if funit = ∅ then
          schedule_operation = FALSE;
        else
          schedule_operation = TRUE;
        endif
      endif
    if schedule_operation = TRUE then
      SCHEDULE_OP (Scurrent, op, Cstep);
    endif
    PListtk = DELETE (PListtk, op);
  endwhile
endfor
INSERT_READY_OPS (V, PListt1, PListt2, ..., PListtm);
endwhile

```

Figure 3.22 Low-power list-scheduling algorithm.

operations are scheduled during the next iterations of the loop. These iterations continue until all the priority lists are empty.

Low-power list-scheduling algorithm

The traditional list-scheduling algorithm shown in Figure 3.21 has been modified by means of including some heuristics to achieve as many OPRs as possible. A simplified version of its algorithm is presented in Figure 3.22. Both algorithms follow the notation in [GDWL92].

Those operations that share an operand are grouped in *operand-sharing sets* (henceforth named SS) (CREATE_ALL_SS()). All operations of a group are able to be executed on the same FU. An operation of an SS (IS_OSS()) is able to reserve the FU where it is going to be assigned for the rest of its SS in case it has not one re-

served yet (`RESERVE_FU_IN_SS()`). Given an SS and its reserved FU, in the best case $|SS| - 1$ OPRs can be obtained. All these consecutive OPRs on the same FU are called an *operand-sharing chain*. LPLS attempts to schedule as many operations as possible of the SS on its reserved FU. It also attempts not to execute other operations on this FU in order to prevent breaking the operand-sharing chain (`OBTAIN_FREE_AND_NOT_RESERVED_FU()`). The scheduling of the operation of an SS is guided by giving more priority to the operations in the operand-ready queue whose SS has already a reserved FU (`UPDATE_PRIORITIES()`). The priority of an operation is decreased (i.e. will be scheduled later) if it is going to be assigned to an FU not reserved by its SS. If the operation scheduled in a later cycle happens to be in the critical path, the final latency is increased.

All the information about achieved OPRs gathered during the execution of LPLS is transferred to the FU-binder as a set of binding constraints. The FU-binder first complies with all these constraints (i.e. achieves all OPRs already obtained by LPLS) and after that proceeds as the traditional FU-binder with weights to minimize the number of interconnection units (multiplexers).

LS has a complexity of $O(n)$, where n is the number of operations. LPLS has a complexity of $O(n^2m)$, where m is the number of unit types.

Results

The power consumption of the designs synthesized with LS are compared with the power consumption of those obtained with LPLS. To estimate power consumption, the coarse-grained model (see Section 3.3) is used and 12-bit-wide FUs are assumed.

It is important to notice that LS already achieves many of the possible OPRs because the FU binder already forces some OPRs in its attempt to minimize the number of multiplexers (e.g. interconnections).

Several results are shown in Table 3.6. The benchmarks have been scheduled with the resources reported in Table 3.4. The fourth column in Table 3.6 indicates the maximum possible number of OPRs. The fifth and sixth columns show the number of OPRs (for both type of FUs) achieved with LS and LPLS.

The last column of Table 3.6 accounts for the savings in power consumption in the FUs because of the increment of achieved OPRs obtained with LPLS. The power consumption of an operation of the benchmark depends on the type of FU where this operation is scheduled and on how many operand changes that FU has when executes the operation. A 17% of power reduction is achieved in the Daubechies

Bench.	LS Latency	LPLS Latency	Max. OPRs	OPRs with LS	OPRs with LPLS	Power reduc.
(1)	20	20	4 \otimes	3 \otimes	4 \otimes	2%
(2)	20	22	10 \otimes	7 \otimes	10 \otimes	17%
(3)	14	15	11 \oplus /3 \otimes	5 \oplus /2 \otimes	5 \oplus /2 \otimes	0%
(4)	11	11	6 \otimes	0	4 \otimes	7%
(5)	9	9	9 \oplus	2 \oplus	9 \oplus	10%
(6)	17	17	3 \otimes	3 \otimes	3 \otimes	0%
(7)	5	5	2 \oplus	1 \oplus	1 \oplus	0%

Table 3.6 Results obtained with the LPLS algorithm.

filter and a 7% in the 4×4 matrix multiplication. The rest of the benchmarks present a small or null power-consumption reduction because of the following reasons: (a) the maximum number of possible OPRs is too small compared to the number of operations of the benchmark and (b) the null or little increase in OPRs achieved by LPLS with respect to LS.

3.6.3 Register binding for low power

Once the scheduling and FU-binding tasks have been done, the register-binding algorithm for low power (LPRB) attempts to further reduce the useful and the useless power consumption of FUs (see Section 3.4.4).

LPRB assumes that the control unit maintains, for each FU, the same registers at its inputs during idle cycles.

The LMS benchmark (see Figure 3.23(a) for its DFG) will illustrate how LPRB works.

Reducing useless power consumption in FUs

One way to tackle the reduction of useless power consumption is by building up a register set that minimizes the number of input changes at the idle units. LPRB algorithm follows this approach (first part of the algorithm in Figure 3.24).

A traditional approach for building up a register set is the clique-partitioning method. After applying this method to a lifetime compatibility graph for the variables (CG), each clique of the partition corresponds to one register. LPRB uses the same traditional approach but applied to a different variable-compatibility graph (LPCG). To build up the LPCG, the register-binding for low power first constructs

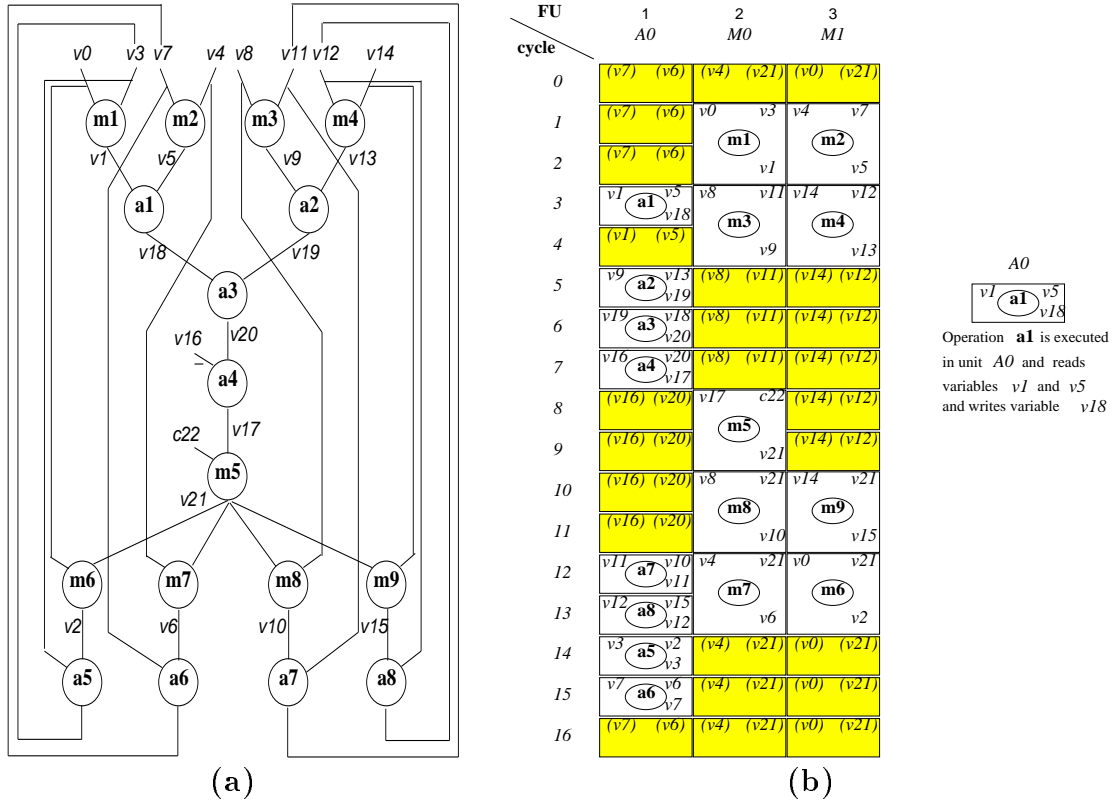


Figure 3.23 (a) DFG of the LMS filter and (b) schedule and FU binding.

the CG (`CREATE_CG()`). In a second step, a set of edges of the CG are removed (`REMOVE_EDGE()`). Each edge removed from the CG connects two compatible variables with the following property: should both be assigned to the same register, an idle FU would have an input change.

Figure 3.23(b) illustrates this concept. It shows the schedule and FU binding of the DFG of Figure 3.23(a) with one adder (one cycle) and two multipliers (two cycles). The shadowed slots represent the cycles in which the FUs are idle. For each FU, the control unit maintains, during idle cycles, the same registers (containing the variables in parenthesis) on its inputs. Let us consider what happens with FU *A0* in cycle 10. An input change will occur at the inputs of idle unit *A0* if, for example, variables *v16* and *v21* are assigned to the same register because multiplier *M0* will modify the value of that register in cycle 9. The same happens with variable pair (*v20* – *v21*). But not all the variables of these two pairs have compatible lifetime between them. In this example, only the pair (*v20* – *v21*) does. Thus, for the FU *A0* in cycle 10, this edge is removed from the CG. If the same procedure is applied to all the idle slots of Figure 3.23(b), 6 edges will be removed.


```

/* reduce power consumption in idle functional units */
CG = CREATE_CG (V);
for c = 1 to MAX_CYCLES do
  for fu = 1 to MAX_FUs do
    if IDLE (fu, c) then
      for each operation op whose result
        is in cycle (c - 1) MOD MAX_CYCLES do
        op_source = OPERATION_IN_FU (fu);
        REMOVE_EDGE (CG, <VAR_DEST(op_source), VAR_A(op) >);
        REMOVE_EDGE (CG, <VAR_DEST(op_source), VAR_B(op) >);
      endforeach
    endif
  endfor
endfor
REGISTER_BINDING(CG);
/* reduce power consumption in non - idle functional units */
for each FU fu do
  OBTAIN_BEST_VARIABLE_ORDER (fu, AHD);
endforeach
INTERCONNECTION_UNIT_BINDER();

```

Figure 3.24 Register-binding algorithm for low power.

The drawback in removing edges is the possibility to obtain a larger register set, as it will be confirmed later with the results.

Not all the useless power consumption in the idle FUs is eliminated with this technique. As an example, let us consider FU *A0* in cycle 16 in Figure 3.23(b). Because the previous operation executed in FU *A0* has variable *v7* as an operand and as the result, FU *A0* has in cycle 16 an input change.

Further reducing useful power consumption in FUs

By taking into account the commutative property of some operations and the average Hamming distance (AHD) among the variables of the design, useful power consumption in FUs may be further reduced (it was first reduced in the scheduling task). This reduction has to be done after the register set has been derived. This process is shown in the second part of the algorithm in Figure 3.24.

As an example, consider additions *a1* and *a2* of Figure 3.23(b). With the variable input order shown, the FU *A0* has an AHD on one of its inputs of $H(v1, v9)$ and on the other input of $H(v5, v13)$. Recall from Section 3.5 how the power consumption of an FU depends on the AHD of its inputs. If the AHD information among the variables is available, the reduction in power can be evaluated if the variable order in addition *a2* is changed. The problem of obtaining the best variable order for all

Bench.	LPLS and TRB						LPLS and LPRB						Power Red.
	(1)	(2)	(3)	(4)	(5)	(6)	(1)	(2)	(3)	(4)	(5)	(6)	
(3)	21	14.9	65	33.2	73.0/182.1	303.2	23	14.7	67	34.2	48.5/184.0	281.4	7%
(6)	12	4.6	23	13.3	1.4/104.3	123.6	12	4.5	24	13.9	0.2/98.4	117.0	5%
(7)	6	1.21	5	0.85	0.33/1.45	3.84	7	1.38	4	0.68	0.0/1.46	3.52	8%

Table 3.7 Comparison between the traditional resource-binding algorithm (TRB) and its low-power version (LPRB).

operations requires an exhaustive exploration. Thus, for simplicity, LPRB follows a greedy approach (`OBTAIN_BEST_VARIABLE_ORDER()`).

By defining a variable order, the degrees of freedom for the interconnection-unit binder are reduced because the correct variable order (which implies the correct register order) has to be satisfied. This implies that the number of multiplexers will be larger than the number obtained if no useful power is reduced.

Results

TRB is compared with its low-power version LPRB over three data-path benchmarks for which we have representative input data². The AHD among the variables has been obtained by means of profiling the benchmarks. In all of them, the scheduling and FU-binding tasks have been done with the low-power methods described in Section 3.6.2. The benchmarks have been scheduled with the resources reported in Table 3.4. To estimate power consumption, the fine-grained model (see Section 3.5) is used and 12-bit-wide FUs are assumed.

For both register-binding algorithms, useful and useless power consumption of FUs, and the number of registers and multiplexers³ along with estimations of their power consumption are reported in Table 3.7. All power estimations are in $nJ/iteration$. Columns (1) to (6) are, respectively, number of registers, power of registers, number of multiplexers, power of multiplexers, useless/useful power of FUs and total data-path power.

In the 1-D 8-input Lee DCT and pixel interpolation benchmarks, no improvement has been observed when applying the algorithm for reducing the useful power con-

²It is important to notice that the AHD among the variables highly depends on the input data. The AHD of the benchmarks related to image processing has been obtained using the well-known Lena picture. We have observed that the AHD values converge fast (in approximately 500 iterations of the algorithm).

³The equivalent number of 2-input multiplexers.

sumption in FUs. The greedy method used did not change the variable order for any FU.

In the pixel interpolation benchmark, only two adders are used. This implies that the power consumption of the registers and multiplexers plays an important role in this benchmark.

It is worth noticing the area-power trade-off: in two benchmarks the number of registers and multiplexers has increased when applying LPRB. Although the total area has increased, the power consumption has been reduced.

3.7 CONCLUSIONS

In this chapter, several strategies to tackle the problem of power consumption at register-transfer level have been presented. The common idea behind these strategies is to reduce the activity of functional units by reducing the switching activity of their input operands. The potential benefits have been evaluated in different DSP examples. The techniques have been evaluated with a novel two-level power consumption model of functional units. Up to 34% power reductions have been obtained.

Algorithms that reduce the activity of the functional units by minimizing the changes of their input operands have been presented for the high-level synthesis tasks of scheduling and register binding. A significant power-consumption reduction is obtained in the scheduling task with little increase or no increase at all in latency. Further power reduction is achieved in the idle functional units during the register-binding task by increasing the number of storage and interconnection units and taking into account both the commutative property of some operations and the average Hamming distance among the variables of the data-flow graph to be synthesized.

ARCHITECTURAL TECHNIQUE FOR LOW POWER

This chapter presents a method for reducing the power dissipation produced by useless activity that only dissipates power and do not contribute to the calculation of the final result of the circuit. This method is specially indicated for the class of circuits that present a layered structure, where each layer is driven by primary inputs from the very beginning, thus generating a high amount of useless activity. Arithmetic circuits, such as parallel multipliers, belong to this class of circuits.

The technique proposed to reduce the useless activity is based on the insertion of self-timed controlled latches on the high active paths of the circuit.

In this chapter, this technique has been implemented and evaluated for several configurations of array multipliers.

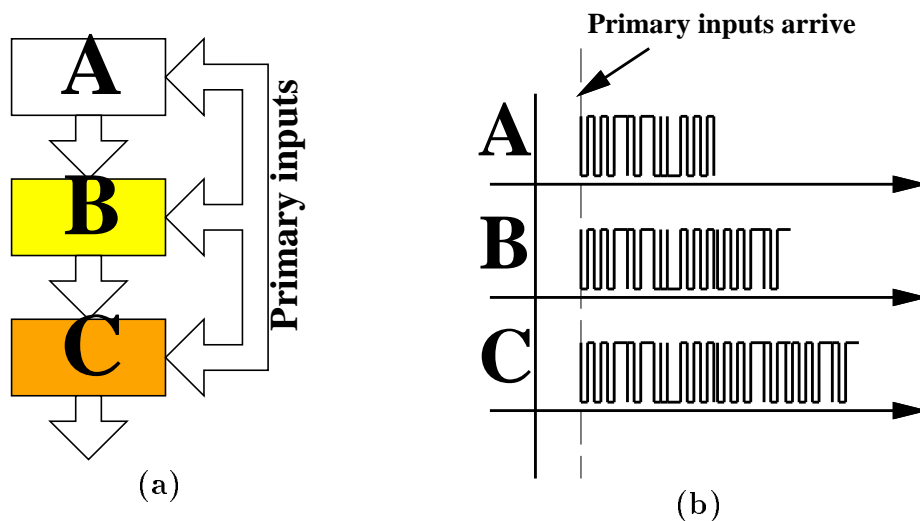


Figure 4.1 (a) Circuit structure with glitching; (b) block C presents more activity than block A.

4.1 INTRODUCTION

The switching activity of a circuit may be divided into:

- useful activity: those signal transitions needed to change the gate outputs of the circuit from the previous cycle to the current cycle.
- useless activity: the rest of the circuit activity. Consumes power and does not generate correct outputs.

The amount of useless activity depend on the structure of the circuit. It can range from 15 to 70% [BFR94, SGDK92]. Useless activity is generated because of the different arrival times of the signals to the inputs of the gates. The different propagation times of the gates is one reason of these unmatched delays. Reducing the logic depth of the circuit and equalizing paths through gate sizing techniques decrease the amount of useless activity.

However, some circuits present a regular structure with a fixed logic depth. In these circuits, other more efficient techniques for reducing the useless activity should be devised. For example, the block C in Figure 4.1(a) will start computing when the primary inputs arrive (Figure 4.1(b)). However, block C only needs to compute its output then the correct output of block B is available.

In this simple example, a large amount of useless activity may be eliminated by delaying the primary inputs to blocks B and C till the outputs of, respectively, blocks A and B have been calculated. This solution is not appropriate if the circuit presents several blocks and the primary inputs go to different blocks since the area overhead produced by the delay components may be prohibited. Moreover, the delay components consume power and may overcome the power savings obtained with the reduction of the useless activity.

Array multipliers belong to this type of highly regular circuits. In this chapter, a technique that reduces the useless activity in array multipliers is proposed. It is based on the insertion of self-timed controlled latches on some high-active signals of the circuit.

This chapter is organized as follows. In Section 4.2, an overview of parallel multiplier circuits is given. In Section 4.3, previous studies in the area of low-power arithmetic circuits in general and multipliers in particular are presented. In Section 4.4, the potential power reduction in array multipliers is analyzed. In Section 4.5, the *transition-retaining barrier* (TRB) technique is proposed. In Section 4.6, the implementation of the TRBs is described. Finally, the results obtained are presented in Section 4.7. In Section 4.8, some conclusions are presented.

4.2 PARALLEL MULTIPLIERS

Multipliers are basic arithmetic circuits for many DSP applications. Among multipliers, parallel multipliers are the fastest and they are widely used in order to achieve high execution speed. Parallel multipliers present a good regularity but also a high dissipation that increases dramatically with the operand bit-width. Techniques for decreasing the dissipation of this type of circuits are thus suitable and in some cases necessary if power dissipation is a major issue.

Multipliers are classified into two basic types: sequential and parallel. The parallel ones are further classified into two subtypes:

- *tree multipliers*: all partial products are generated in parallel and afterwards a multi-operand adder is used to obtain the final result. The delay of a parallel multiplier is $O(\log n)$, being n the bit-width of the input data.
- *array multipliers*: these multipliers do not use two separate circuits for partial product generation and accumulation. A single cell is used in an array struc-

ture where the new partial products and the partial accumulation are produced simultaneously. The delay of an array multiplier is $O(n)$.

Schemes for parallel multipliers have been studied for decades trying to reduce the number of logical levels of the multiplier [BW73, Pez71].

Tree multipliers are faster than array multipliers because the former reduce the number of logical levels; however, they also suffer from a bad layout regularity. This bad regularity implies a large routing area.

4.3 PREVIOUS WORK

The design of arithmetic circuits aiming at minimizing power dissipation has basically focused on multipliers. Other arithmetic circuits have been rarely studied. The power consumption of adders has been estimated in [CS93, KGG95, NMO94]. Some low-power arithmetic designs have been proposed in [EL94, KBL95, EL95].

In [SLB95] the delays in the full adder cells of the multiplier are balanced by implementing a few delay buffer circuits and, therefore, power savings are obtained since the unnecessary activity is reduced.

In [YYN⁺90], a comparison is presented between the design of a static CMOS multiplier and the design using Complementary Pass-Logic (CPL) [WE93]. Because of the fewer transistors used in the CPL design and their lower input capacitance, a 28% of power dissipation reduction is achieved with a frequency of operation of 100 MHz and a power supply of 4V.

In [SA95], an architecture to implement the data compression in a 52×52 -bit Booth multiplier [WE93] based on tally functions is described. The authors derive a reduction in power of 35% in the compression stage of the multiplier compared with that of a conventional multiplier.

The characterization of unnecessary activity in an array multiplier has been studied in [OF94]. It is shown that the unnecessary activity in a 16×16 -bit array multiplier is the 50% of the total activity.

In [MT95], the switching activity of multipliers is studied as a function of the multiplication algorithm, the compressor circuitry and the circuit design style. A 8×8 -bit modified Booth multiplier is designed where the activity is estimated to be reduced by a factor of 2.

In [LvMJ95], the transition activity in array and tree (Wallace) multipliers is analyzed. This study shows that the number of total transitions in a 16×16 -bit Wallace multiplier is less than half of those in the array design. Moreover, the ratios useless/useful transitions are 3.26 and 0.16 in the array and Wallace multipliers, respectively.

The switching activity during the partial product generation in a radix-4 Booth multiplier is reduced in [ZA95]. This is done by converting the multiplicand from two's complement into sign-magnitude representation. Significant bit transitions are saved because negating the multiplier in sign-magnitude representation implies only one bit toggle. Unfortunately, no power reductions are reported for the entire multiplier in this work.

Asynchronous multipliers have also been studied in [HvBPS93, SNNS93]. The authors of [HvBPS93] conclude that for small multipliers, a synchronous design is better in terms of power consumption than its asynchronous counterpart. The take-over point for asynchronous is predicted to be near an input width of 16. Three small (input width of 4 bits) asynchronous multipliers are also compared to the synchronous design in [SNNS93] showing that the energy per operation of the synchronous design is less than one third of the asynchronous designs.

Finally, in [LS94, SR95] a similar technique as the one proposed in this paper is studied. In [LS94], latches are inserted in a 16×16 -bit radix-4 Booth multiplier to reduce the switching activity. The authors claim that a 40% of power savings is achieved with a frequency of operation of 50 MHz and a power supply of 3V. In [SR95], a design of a Booth encoded array multiplier is presented that uses dynamic CMOS logic together with a self-timed evaluate signal in such a way that each carry-save adder within the array evaluates only after all of its inputs have stabilized. No power reductions obtained with this technique are reported.

4.4 POTENTIAL POWER REDUCTION IN ARRAY MULTIPLIERS

The array multiplier structure studied is depicted in Figure 4.2. This array multiplier computes the $2n$ -bit product of two n -bit unsigned numbers. The basic building blocks of the array multiplier are full-adders (FA) and half-adders (HA) (see Figure 4.3).

Figure 4.4 plots the logic transitions per operation that take place in the interconnections among the FA and HA in a 16×16 -bit array. These results have been

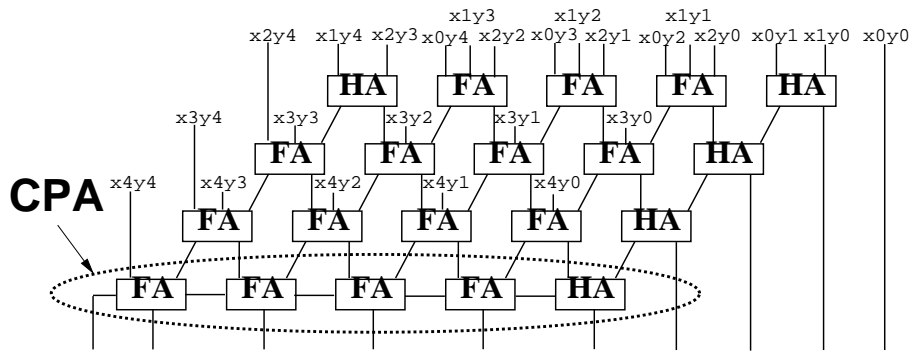


Figure 4.2 5×5-bit array multiplier.

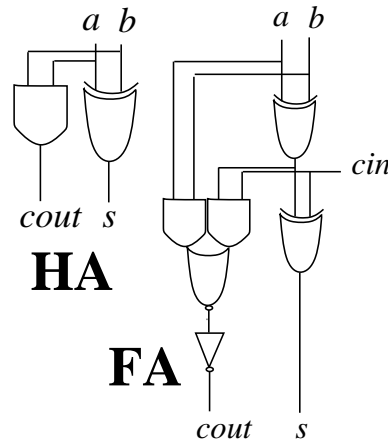


Figure 4.3 (a) FA and (b) HA structure.

obtained with a gate-level simulator for a totally inertial gate-delay model¹. Each block in the figure represents the sum of the s and $cout$ signal transitions of the corresponding FA or HA.

¹A gate with a delay t_i (from input i to the gate output) has an *inertial degree* d if the gate filters all those glitches of length g_i such that $g_i < d \times t_i$. If $d = 0$ we say that the gate follows a *pure* delay model (i.e., no glitch is filtered). If $d = 1$, we say that the gate follows a *totally inertial* delay model (i.e., all glitches shorter than the gate delay t_i are filtered). If $0 < d < 1$, we say that the gate follows a *partially inertial* model.

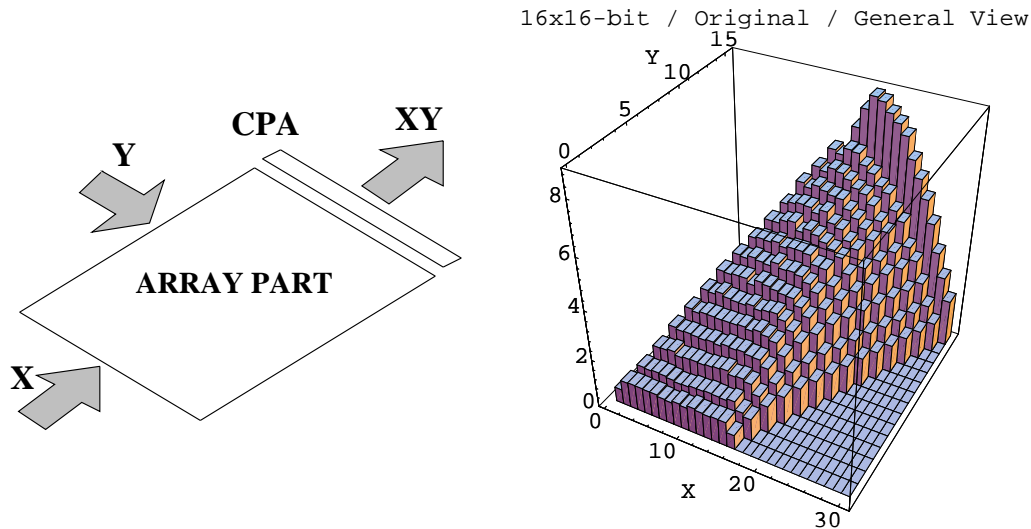


Figure 4.4 Signal transitions per operation for each basic block in a 16×16 -bit array multiplier.

The layout of the figure has been rotated to allow an appropriate observation of the transitions at each cell. The tallest bars of the plot correspond to the carry-propagate adder (CPA)².

Since the average power dissipation of a CMOS circuit is dominated by the switching activity, *useless* signal transitions (i.e. transitions that only dissipate power and do not contribute to the calculation of the final result) should be prevented.

In the layered structure of array multipliers, where each layer is excited from the very beginning, many useless transitions are generated. These transitions, in their turn, generate more useless transitions, thus creating a *snow-ball* effect. We observe in Figure 4.4 the consequence of this effect. The deepest layers of the array (i.e. those closest to the CPA) are the most power consuming.

The potential power reduction in array multipliers increases dramatically with the operand bit-width. Consider Figure 4.5. Both useful and total number of signal transitions for an 8×8 , 16×16 , 32×32 and 64×64 -bit array multiplier are shown. It is clear that the larger the array multipliers is, the larger is the possibility for reducing its power dissipation.

²CPA is used here for its simplicity and inherent low-power (although high power-delay product) [NMO94].

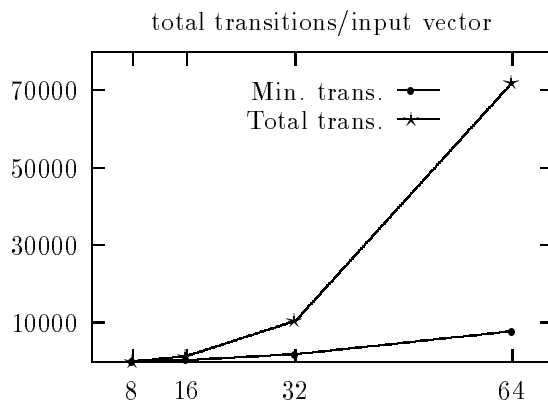


Figure 4.5 Useful and total number of signal transitions for an 8×8 , 16×16 , 32×32 and 64×64 -bit array multiplier.

It is interesting to notice that the potential for reducing power consumption in tree multipliers (e.g. Wallace) is lower than in array multipliers. The reason is that parallel multipliers present less generation of useless transitions because of the more balanced paths to the basic blocks that compose the multiplier. We have observed that, for example, the number of signal transitions in a 32×32 -bit array multiplier doubles the number in its Wallace counterpart.

4.5 TRANSITION-RETAINING BARRIERS

A strategy to avoid useless transitions is to insert self-timed controlled latches on the paths where these transitions occur. A possible solution is to latch every output of each cell. This configuration has some drawbacks that make it not very appealing: area and delay (and even power dissipation) increase as a result of the latches.

A trade-off is to find, for instance, a barrier of latches that divides the array into two parts (the upper and bottom halves). Thus, the useless transitions that are generated in the upper half of the array will not cross the barrier until the barrier switches into transparent mode. Henceforth, this barrier of latches is called a *transition-retaining barrier* (TRB).

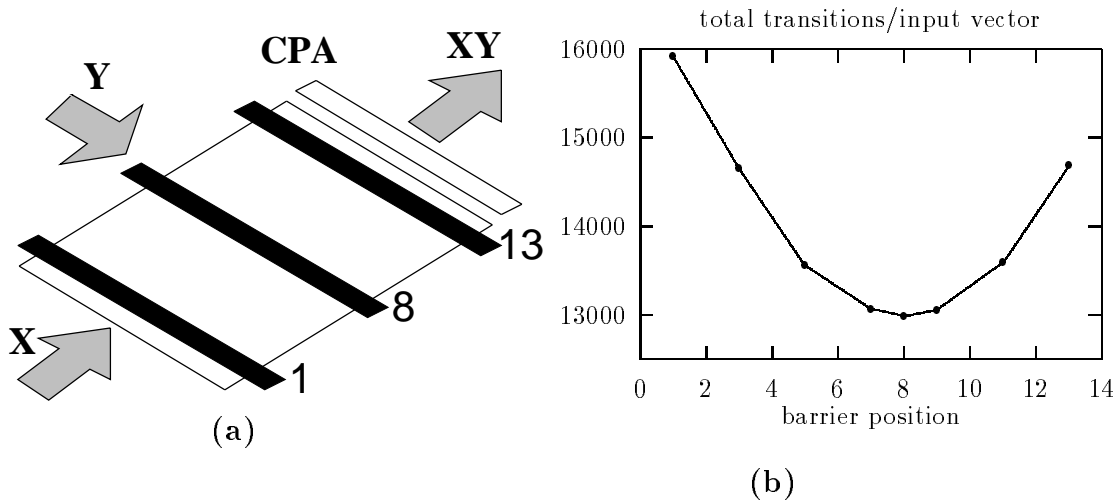


Figure 4.6 (a) Possible locations for one TRB and (b) transitions per input vector for different locations of one TRB in a 16×16 -bit array.

For VLSI arrays, a regular implementation of this barrier is sought. The easiest way to insert latch barriers in the array is by taking advantage of the layered structure of the array.

The enabling signal of the latch barriers is generated by delaying an input signal (for example, the clock signal). The delay should be no longer than the critical path from the inputs of the array to the barrier (thus not increasing the delay of the multiplier) and as similar as possible to the critical path (thus no useless transitions are allowed to cross the barrier too early).

It is obvious that the best location for the array is the middle of the array because we balance the useless transitions between the two partitions. Figure 4.6(b) shows an analysis of the number of transitions per input vector in the 16×16 -bit array with different locations for the TRB. The optimum partition is in row 8 of Figure 4.6(a).

We can evaluate also the switching activity if two or more TRBs are inserted. Similarly, the minimum is obtained when the array is divided into equal parts.

Figure 4.7 shows how the transitions in the array are reduced when inserting one and two TRBs.

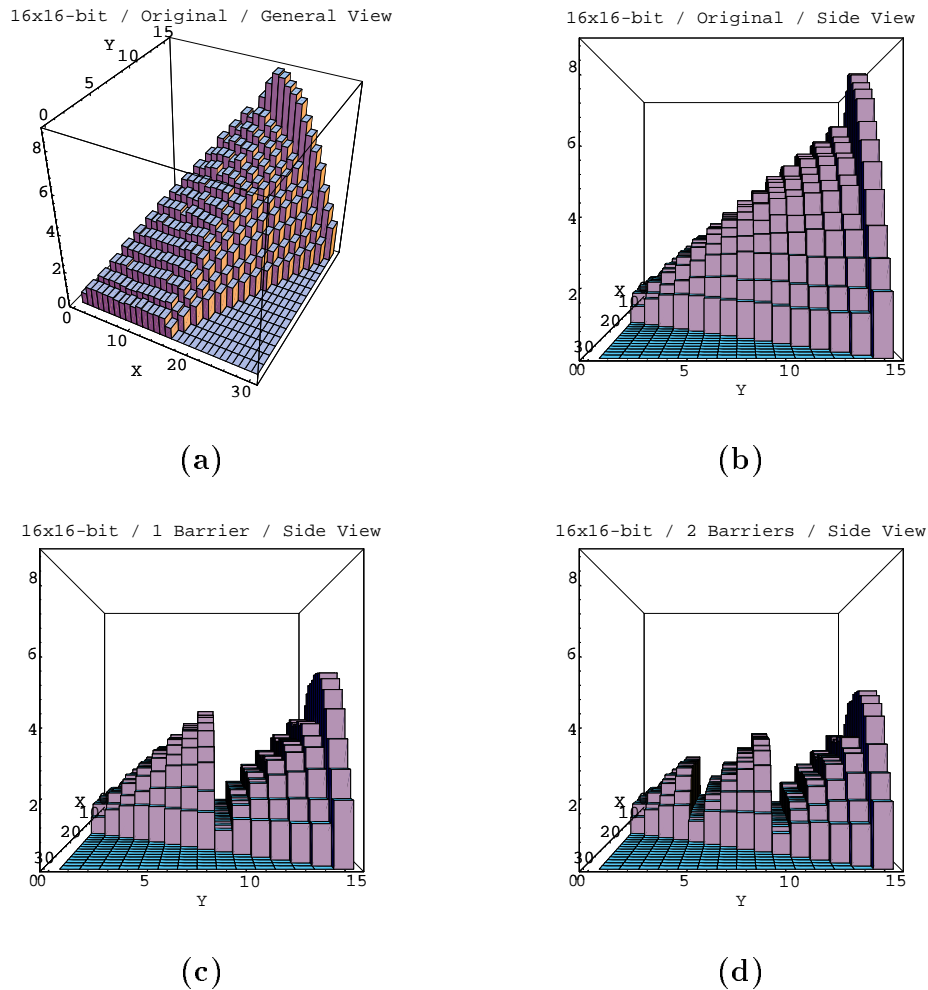


Figure 4.7 Signal transitions per input vector in a 16×16 -bit array multiplier. (a) General view of the design with no TRBs and side view with (b) no TRBs, (c) one TRB and (d) two TRBs.

4.6 IMPLEMENTATION OF THE TRB

Since the cost of latches may be too high (both in area and power dissipation), other structures that perform the same transition-retaining function at the outputs of the multiplier cells have to be devised.

The designer can choose between a static or dynamic implementation of the latch. The most appropriate dynamic latch structure in terms of area is the Clocked-CMOS (or C^2 MOS) latch [WE93]. The area overhead consists of four transistors per cell (two for each output of the full-adder cell). Figure 4.8(a) depicts the C^2 MOS latch.

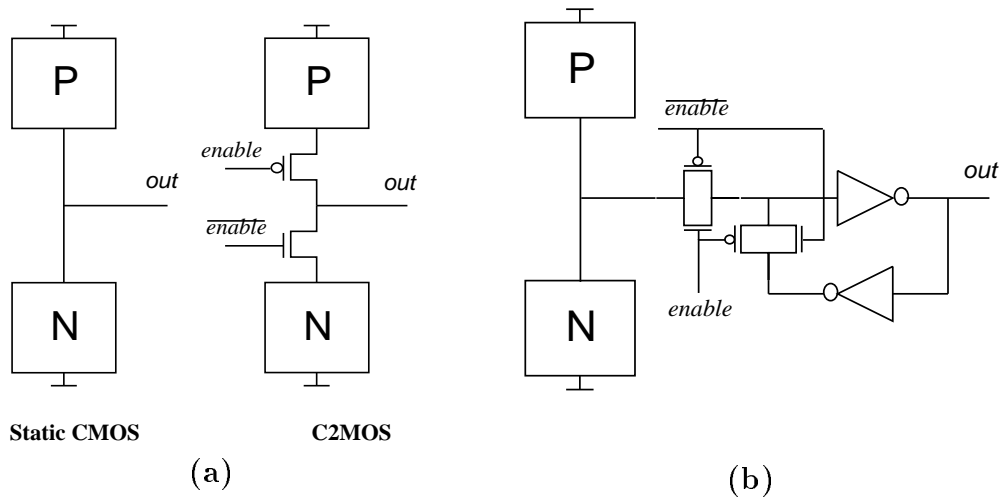


Figure 4.8 (a) Comparison between Static CMOS and C²MOS and (b) avoiding direct-path currents.

Related work with power dissipation using C²MOS has been presented in [SNK73, BOI95].

The main problem with the use of C²MOS structures is that capacitive coupling may arise at the *out* node in Figure 4.8(a), causing direct-path currents in the gates fed by this node. To avoid this problem, the static structure depicted in Figure 4.8(b) should be used.

The delay mentioned in Section 4.5 is implemented now as a chain of inverters. We distinguish 4 types of delay-cells:

- TA simulates the delay of the first layer of the array
- TB simulates the delay of a full-adder
- TC simulates the delay of a full-adder and controls the TRB and
- TD is the same as TC but has no other delay-cell as fanout

Figure 4.9 depicts a 4×4-bit array with two TRBs where the four types of delay-cells can be observed. All the controlling signals are obtained by delaying the input signal *start*.

The number of inverters in each delay-cell depends on the delay of the inverter, the input capacitance of the inverter and the capacitance of the C²MOS signals (*enable*

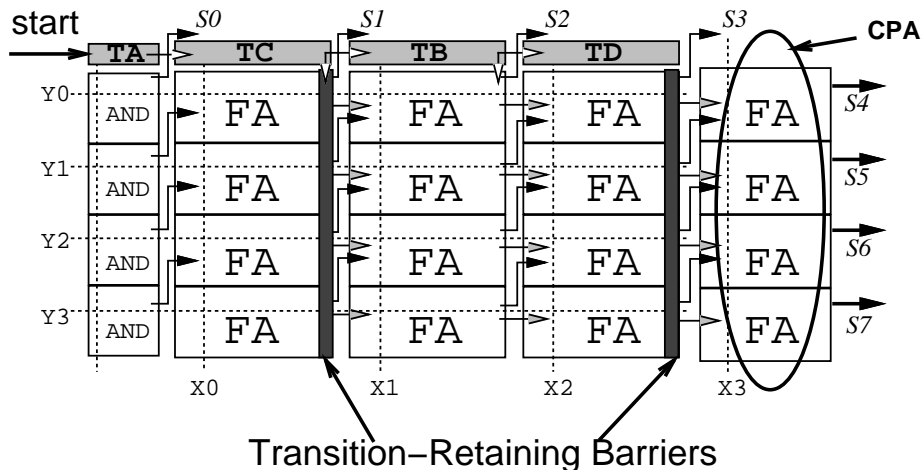


Figure 4.9 4×4-bit array with two TRBs. The four types of delay-cells (TA-TD) are shown.

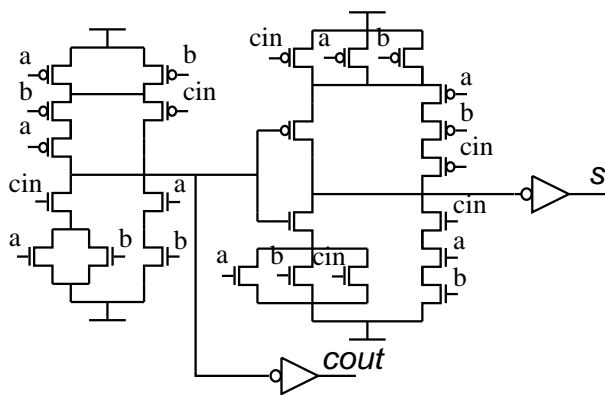


Figure 4.10 Design of a FA.

and \overline{enable} of Figure 4.8(b)). The number of inverters in TC and TD delay-cells also depends on the operand bit-width of the array.

4.7 RESULTS

Four configurations of array multipliers have been implemented for the evaluation of power dissipation, area and delay. The original array multiplier is compared to designs with one, two and three TRBs.

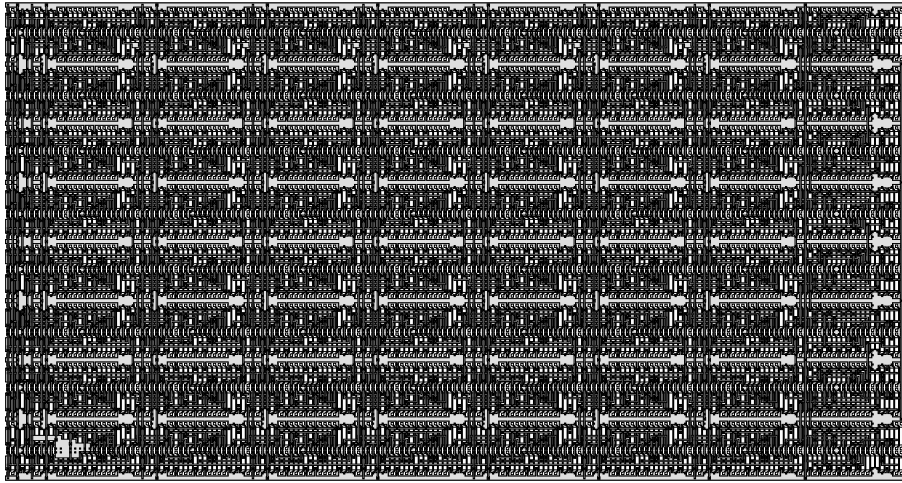


Figure 4.11 Layout of an 8×8 -bit array multiplier in a sea-of-gates design style.

The configurations are implemented in a sea-of-gates (SoG) design style using the Ocean system (see Appendix A, Section A.2). A layout of an 8×8 -bit array multiplier is shown in Figure 4.11. The full-adder cell is implemented as in [WE93] (see Figure 4.10). The evaluation of the power dissipation has been obtained with a switch-level simulator [vG89] that uses an RC model for timing calculations.

The scenario of the simulation is the following:

- input patterns are randomly generated with probability of logical 1 at the primary inputs being 0.5
- all inputs to the multiplier arrive at the same time and
- between one pseudo-random input vector and the following one there is enough time for the multiplier outputs to settle.

The results are presented in Table 4.1. In the 8×8 -bit multipliers with one and two TRBs, the overhead of the additional logic generates more power than it reduces. Hence, the overall power dissipation is higher than in the original design.

A trade-off exists between the delay of the new designs and their power dissipation. If the delay of the delay-cells is greater than the delay of the full-adder cell, the multipliers will present less power but will be slower. Thus, the increase in the delay of the designs with barriers is because of the additional transistors of the latches and the longer delay of the delay-cells.

Array Multiplier	Area	Area Incr.	Delay	Delay Incr.	Energy	Energy Reduc.
8×8-bit original	1	-	1	-	1	-
8×8-bit 1 TRB	1.144	14.4%	1.042	4.2%	1.056	-5.6%
8×8-bit 2 TRB	1.154	15.4%	1.119	11.9%	1.060	-6.0%
16×16-bit original	1	-	1	-	1	-
16×16-bit 1 TRB	1.071	7.1%	1.029	2.9%	0.927	7.3%
16×16-bit 2 TRB	1.078	7.8%	1.049	4.9%	0.873	12.7%
16×16-bit 3 TRB	1.082	8.2%	1.180	18%	0.795	20.5%
24×24-bit original	1	-	1	-	1	-
24×24-bit 1 TRB	1.047	4.7%	1.059	5.9%	0.901	9.9%
24×24-bit 2 TRB	1.049	4.9%	1.089	8.9%	0.825	17.5%
24×24-bit 3 TRB	1.054	5.4%	1.144	14.4%	0.745	25.5%
32×32-bit original	1	-	1	-	1	-
32×32-bit 1 TRB	1.033	3.3%	1.021	2.1%	0.870	13.0%
32×32-bit 2 TRB	1.037	3.7%	1.044	4.4%	0.780	22.0%
32×32-bit 3 TRB	1.039	3.9%	1.076	7.6%	0.700	30.0%

Table 4.1 Comparison between the original design and the new designs with TRBs.

We have the best results in power dissipation reduction when three TRBs are inserted. However, it is important to notice that the greatest decrease in power is achieved when inserting one TRB with regard to the original design rather than when inserting two TRBs with regard to inserting one TRB.

The increase in area when inserting TRBs is small, and more negligible as the bit-width increases (there is an increase of 15.3% when inserting three TRBs in the 16×16-bit and just a 7.6% in the 32×32-bit with also three TRBs). The overhead in power dissipation, area and delay as a result of the insertion of TRBs has been taken into account in the results in Table 4.1.

4.8 CONCLUSIONS

In this chapter we have studied the power consumption of a class of circuits (array multipliers) that present a high power consumption because of the generation and propagation of a large amount of useless activity.

The technique of inserting *transition-retaining barriers* (TRB) in order to decrease the propagation of the *useless* transitions in array multipliers has proved effective in reducing the power dissipation of this type of circuits. With three TRBs we achieve

a reduction in power of 30% in a 32×32 -bit array multiplier with an increase in area and delay of 3.9% and 7.6% respectively.

The final multiplier remains combinational and highly regular. This implies that the technique presented in this chapter can be easily integrated in existing CAD tools for the generation of array multipliers.

This technique should be evaluated for other type of circuits that present a high amount of useless activity.

LOGIC/CIRCUIT TECHNIQUE FOR LOW POWER

This paper addresses the optimization of a circuit for low power using transistor reordering, a technique that has been traditionally used to improve the performance. In this chapter this technique is used to find a proper order of the transistors of a gate so that the switching activity at the internal nodes of the gate is minimized.

A stochastic power-consumption model that depends on the switching activity and the signal probabilities of the inputs of the gate is presented. This model takes into account the power at the internal nodes of the gate.

An optimization algorithm that relies on the stochastic model is described. It performs an exhaustive exploration of the different configurations of a gate that are obtained by reordering its transistors. Thus, the best configuration for low power of each gate is selected and the overall power consumption of the circuit is reduced. Power-reduction results are reported for several benchmarks.

5.1 INTRODUCTION

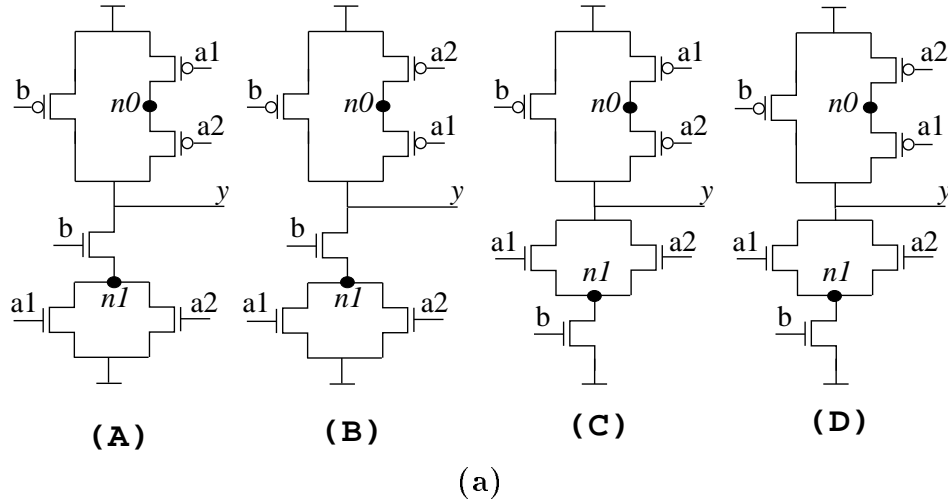
This chapter addresses the optimization of a circuit for low power using transistor reordering from a gate-level description. The optimization algorithm uses a power-consumption model of a static CMOS gate that takes into account the power of the internal nodes of the gate. This model allows the exploration of the different configurations of a gate that are obtained by reordering its transistors. Thus, the best configuration of each gate is selected and the overall power consumption of the circuit is decreased.

We focus on combinational multi-level circuits, where it has been shown that the power consumption of useless signal transitions (i.e. those transitions that do not contribute to the final result of the circuit) accounts for a large fraction of the overall dynamic power consumption of the circuit. Thus, it is necessary to incorporate the switching activity of the input signals into the power consumption of the gate.

5.1.1 Motivation examples

To illustrate why it is important to incorporate the switching activity information to the power estimation of a gate, consider the four possible configurations of the gate in Figure 5.1(a) that implement function $y = \overline{(a1 + a2)} \overline{b}$. Different switching activity at the inputs (D_{a1} , D_{a2} and D_b) results in a different optimal transistor reordering for the gate as it is shown in Table 5.1(b). The equilibrium probability (i.e. the probability for a signal to be '1') of all input signals has been set to 0.5. Table 5.1(b) shows the power consumption for two different input switching activity scenarios (cases **(1)** and **(2)**) of the different configurations relative to configuration **(D)** in case **(1)**. Time intervals between two consecutive transitions at input signal k follow an exponential distribution with average $1/D_k$. In case **(1)** the best transistor ordering is given by configuration **(A)** of Figure 5.1(a); the power consumption is decreased by 12% with respect to configuration **(D)**. In case **(2)**, the power is decreased by 17% if configuration **(D)** is taken instead of **(A)**. The reason of these power savings is the reduction of the switching activity at the internal nodes of the gate. This internal activity depends on the switching activity and probability of the inputs and the order of the transistors.

The power-consumption values in the last experiment have been obtained with an accurate switch-level simulator (named SLS, see Chapter A, Section A.3). Throughout this work we will rely on switch-level simulations instead on SPICE [JQN⁺91] simulations because we will deal with relatively large benchmarks and a large number of input vectors for each one. A SPICE simulator is characterized by high



	Activity (trans./sec)	(A)	(B)	(C)	(D)	Reduction
(1)	$D_{a1} = 10K$ $D_{a2} = 100K$ $D_b = 1M$	0.81	0.84	0.98	1.0	19%
(2)	$D_{a1} = 1M$ $D_{a2} = 100K$ $D_b = 10K$	0.58	0.53	0.53	0.48	17%

(b)

Figure 5.1 (a) Four implementations of function $y = \overline{(a1 + a2)} b$ and (b) relative power consumption for two different input activity scenarios.

accuracy but long simulation time. Simulation time is typically proportional to N^m , where N is the number of non-linear devices in the circuit, and m is between 1 and 2 [WE93].

However, we have performed an experiment using SPICE simulations to validate the use of switch-level simulations in this work. Consider gate configurations (A) and (D) in Figure 5.1(a) with an equivalent load capacitance of an inverter. Two new scenarios are considered that produce the same output waveform. The current drawn by the power supply ($i_{DD}(t)$) for each scenario and configuration is shown in Figure 5.2. We observe significant differences in current supply when the output load has to be charged (at times 10ns and 20ns). Table 5.1 shows a comparison of the power-consumption estimations provided by SPICE and SLS. The power

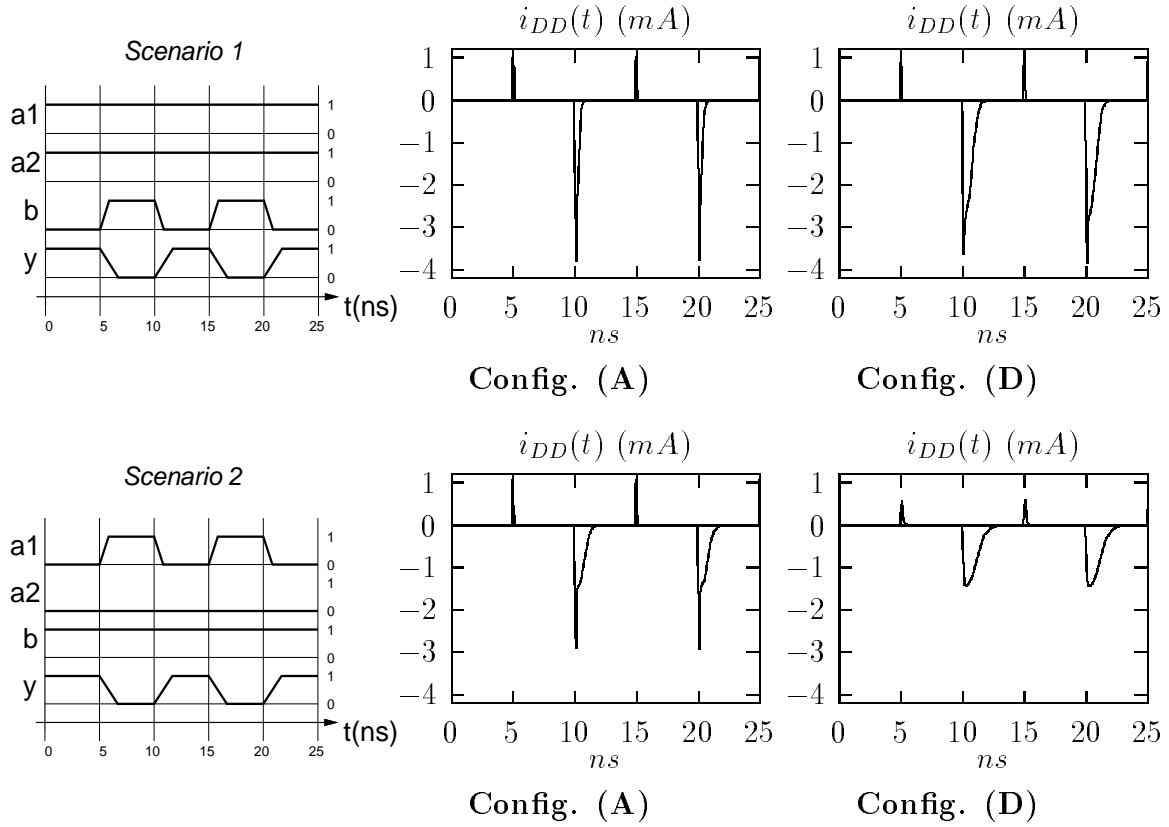


Figure 5.2 SPICE simulations of configurations (A) and (D) in Figure 5.1 for two different input activity scenarios.

	Config. (A)	Config. (D)
<i>Scenario 1</i>	SPICE: $0.305mW$ SLS: $0.334mW$ Error: -10% (1)	SPICE: $0.757mW$ SLS: $0.667mW$ Error: 12% (4)
<i>Scenario 2</i>	SPICE: $0.438mW$ SLS: $0.390mW$ Error: 11% (2)	SPICE: $0.558mW$ SLS: $0.544mW$ Error: 3% (3)

Table 5.1 SPICE and SLS comparison for the configurations in Figure 5.2. Numbers in parenthesis show the power-consumption ranking.

consumption in SPICE is calculated as:

$$W = \frac{1}{T} \int_{t=0}^T i_{DD}(t) \times V_{DD} dt ,$$

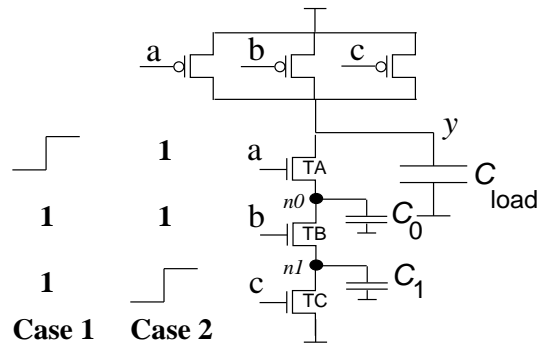


Figure 5.3 Transistor-reordering technique in a 3-input NAND gate (example from [SLW95]).

where T is the cycle time ($10ns$) and V_{DD} is the power supply voltage, while in SLS it is obtained as explained in Chapter A. In this experiment, relative errors of SLS with respect to SPICE are less than 12%. However, it is important to notice that both simulators agree on the power-consumption ranking for the four cases (shown in parenthesis in Table 5.1).

We conclude from this experiment that, indeed, power consumption depends on the order of the transistors and the switching activity of the inputs to the gate and that SLS simulations are valid to discern the lowest power-consuming configuration.

The ripple-carry adder is another example in which the equilibrium probability does not give enough information to optimize gates for low power. Consider an adder implemented as a chain of full-adders that has to calculate the addition of two n -bit operands with equal equilibrium probability for all bits. The equilibrium probabilities of all inputs of the full-adders is 0.5, but it is clear that the switching activity of the inputs of the full-adders corresponding to the operands is low (0.5 transitions per operation) whereas the switching activity of the input corresponding to the propagated carry is higher (specially in those full-adders that compute the most-significant bits) because of the generation and propagation of useless signal transitions.

To show how transistor reordering affects the number of transitions at the internal nodes of a gate, consider the 3-input NAND gate in Figure 5.3. In this particular example, the techniques of input reordering and input interchange (i.e. interchange those inputs to the gate that are logically equivalent) coincide. In Case 1, inputs b and c are “1” and a is initially “0”. The capacitances associated to internal nodes

n_0 and n_1 (C_0 and C_1) are discharged. The output capacitance (C_{load}) is charged. When input a changes to “1”, only the output capacitance is discharged. In Case 2, inputs a and b are “1” and input c is initially “0”. All capacitances are charged. When input c changes to “1”, all capacitances will discharge, thus consuming more power than in Case 1. Therefore, if the probabilities of two of the inputs are close to 1 and the other input is very active, we should assign this input to the transistor TA to minimize the power consumption at the internal nodes of the gate.

5.2 PREVIOUS WORK AND OVERVIEW

Transistor reordering is a technique that does not significantly affect layout area and it may decrease the propagation delay of the gate.

Carlson [CC93] hinted the possibility to use the transistor-reordering technique to decrease power consumption and he presented an algorithm for delay/power/area optimization where high speed was synonym of high power consumption. This approach of measuring power consumption is not sufficiently accurate since it does not consider the probability and switching activity of signals. No power-consumption reductions are reported in [CC93].

Input reordering conform a subset of transistor-reordering techniques. For example, by reordering the inputs in a 3-input NAND gate, 6 different configurations of the gate are obtained; the same occurs if we apply transistor reordering. But in the gate represented in Figure 5.1(a), only two different configurations are obtained by applying input reordering and four are obtained with transistor reordering. Input reordering has been used in [TA94, BIO95] along with transistor sizing to reduce power consumption. It is not clear in those works which is the contribution of the input-reordering technique by itself. This is true specially in [BIO95], where the largest power consumption reduction (15%) is achieved in the benchmark with the largest number of high-fanout gates. Input (or transistor) reordering techniques obtain better results when applied to low-fanout gates because these techniques focus on reducing the power consumption of the internal nodes of the gate. If the output capacitance is increased, the overall gate power reduction obtained is decreased [HZA94].

A study of the potential benefits of transistor reordering for a specific library is presented in [GBJ95], where an average power-consumption reduction of 13% and up to 17% for complex gates is obtained. A novel binary-decision diagram is used to represent the structure of the gate. An algorithm is presented to obtain all possible

transistor reorderings of the gate. No model for the power consumption of a CMOS gate is presented.

In [HZA94], input-reordering techniques are applied to a large variety of CMOS NAND gates ranging from 2 to 8 inputs. The major contribution in [HZA94] is the description of a new model for the power consumption of a MOSFET chain. This model accounts for the consumption of internal nodes and it depends only on the equilibrium probabilities of the inputs of the gate.

The closest works to ours are [PR94, SLW95]. In [PR94], a clear example of the effect of transistor reordering on the power consumption of a circuit is shown. An estimated average of 6% in power reduction is obtained for some MCNC benchmarks.

Finally, in [SLW95] the authors propose a set of simple transistor reordering rules for both basic and complex CMOS gates to minimize the switching activity at the internal nodes. These rules are:

- For NAND gates,
 - place the input signal with high signal probability near ground
 - place the input signal with low switching activity near ground
- For NOR gates,
 - place the input signal with low signal probability near power supply
 - place the input signal with low switching activity near power supply
- For complex gates,
 - transform the complex gates into NOR or NAND structures
 - apply the rules of NAND and NOR gates to the transformed gate

Whenever the rules conflict, heuristics are applied based on probability/activity ratios. Average reductions of 9% are reported for several circuits.

5.2.1 Overview of our approach

The goal of this chapter is to evaluate the power reduction that can be obtained by reordering the transistors that compose the different gates of a circuit. Our work accounts for the transistor-reordering technique by itself. We maintain the size of the transistors constant when they are reordered.

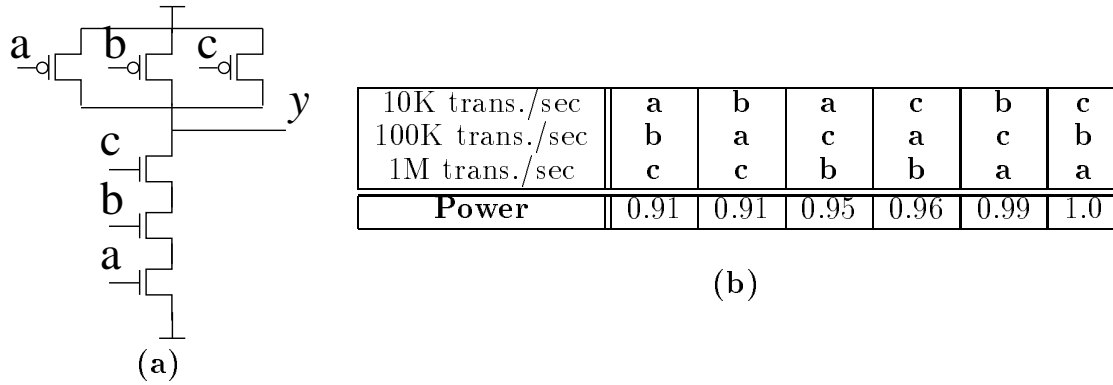


Figure 5.4 (a) Static CMOS 3-input NAND gate (b) relative power consumption for different input activities (without varying the equilibrium probabilities of the inputs).

It has been shown that the arrival times of the input signals affect the consumption of the gate [CC93]. However, we do not take into account the arrival times because we are more concerned about the switching activity of an input during all the cycle time than just the last transition that settles the correct value at the output of the gate.

We present a power-consumption model of a static CMOS gate that depends on both the static probability and the switching activity of its inputs. Our power-consumption model differs from the one presented in [HZA94] in that we take into account the switching information of the input signals. (our model is based on the *transition density* measure of activity in digital circuits proposed by Najm [Naj91]). As an example, consider a 3-input NAND gate (Figure 5.4(a)). Consider equilibrium probabilities of 0.4, 0.5 and 0.6 for its inputs **a**, **b** and **c** respectively. Table 5.4(b) shows the relative power consumption of the gate for different input switching activity scenarios. Our model discerns, among the six possible input reorderings, those that give the maximum and minimum power consumption. The model in [HZA94], on the contrary, can not discern any of the reorderings and it provides the same power estimation for all of them.

An algorithm that traverses the gate-level description of the circuit and uses the gate model mentioned above is presented and results are reported for a wide range of MCNC benchmarks. A cell library with different instances of a single gate to obtain all its possible transistor reorderings has been implemented in a sea-of-gates design style. The results are based on switch-level simulations and show that power-consumption reductions of up to 20% may be obtained when applying the transistor-reordering technique.

This chapter is organized as follows: in Section 5.3, a power-consumption model of a static CMOS gate that takes into account the power consumption of the internal nodes is presented. In Section 5.4, an algorithm that traverses the gate-level description of the circuit and generates an optimized circuit for low power is described. Results are presented for several MCNC benchmarks in Section 5.5. In Section 5.6, some conclusions are drawn for this chapter.

5.3 POWER CONSUMPTION OF CMOS GATES

5.3.1 Definitions and overview

Definition 5.3.1 (Stochastic process) *Let $x(t)$ be the value of a system characteristic being observed at time t . In most situations, $x(t)$ is not known before time t and may be viewed as a random variable. A stochastic process is a description of the relation between the random variables $x(t)$.*

Definition 5.3.2 (Stationary Markov process) *A stationary Markov process is a stochastic process where the probability law relating the next period's state to the current state does not change (or remains stationary) over time.*

Definition 5.3.3 (Equilibrium probability) *Let $x(t), t \in (-\infty, +\infty)$, be a 0-1 stationary Markov process with random transition times. The probability that it takes the value "1" at any given time t is the expected value $E[x(t)]$ at that time and it is independent of time. This value is called the equilibrium probability of $x(t)$ and is denoted as $P(x)$.*

Definition 5.3.4 (Switching activity) *The switching activity of a 0-1 stochastic process $x(t)$ is the number of 0-to-1 transitions and 1-to-0 transitions of $x(t)$ in a time unit.*

Henceforth, we will model logic signals of a circuit as 0-1 stationary Markov processes as in [Naj91] to derive a power-consumption model of a static CMOS gate that includes the switching activity at the output and internal nodes of the gate. The model depends on both the equilibrium probabilities and the switching activity of the inputs of the gate.

The switching activity in both input and output nodes of a gate is measured with the *transition density* technique described in [Naj91]. This technique is briefly reviewed in the next section.

5.3.2 Transition density

The *transition density* is a compact measure of the switching activity in digital circuits. The transition density of a node is the average number of signal transitions per time unit at that node and it is defined as

$$D(y_j) = \sum_{i=0}^{n-1} P\left(\frac{\partial y_j}{\partial x_i}\right) D(x_i) ,$$

where $P(z)$ is the equilibrium probability of a signal or logic function, $D(z)$ is the transition density of a signal, x (y) are the n (m) gate inputs (outputs). $\frac{\partial y_j}{\partial x_i}$ is named the *boolean difference* and it is a boolean function that may depend on all x_p , $p = 1 \dots n$, $p \neq i$. The boolean difference $\frac{\partial y_j}{\partial x_i}$ is defined as

$$y_j |_{x_i=1} \oplus y_j |_{x_i=0} = y_j(x_i) \oplus y_j(\overline{x_i}) .$$

If $\frac{\partial y_j}{\partial x_i} = 1$, then all the transitions at input x_i are propagated to output y_j . In other words, $\frac{\partial y_j}{\partial x_i}$ expresses the sensibility of y_j with respect to input x_i .

Thus, the transition density provides a fast way of propagating switching activity from the primary inputs to the outputs of the gates that compose a circuit. The main drawback of the transition density is that it does not take into account the spatial and temporal correlation of the input signals to the gate (although it considers the signal correlations within the logic module).

As an example, the transition density at the output of a 2-input AND gate is:

$$D(y) = D(a)P(b) + D(b)P(a) ,$$

where y is the output of the gate and a, b are the two inputs. However, when $a = \overline{b}$ ($P(a) = 1 - P(b)$), y is always “0” and, therefore, $D(y) = 0$. However, using the above transition density measure, we obtain:

$$D(y) = D(a)P(b) + D(b)(1 - P(b)) = D(b) ,$$

which is not correct. This drawback, caused by the simultaneous switching of the inputs, can be overcome using conditional probabilities [CRP94].

5.3.3 Extended power-consumption model

Notation

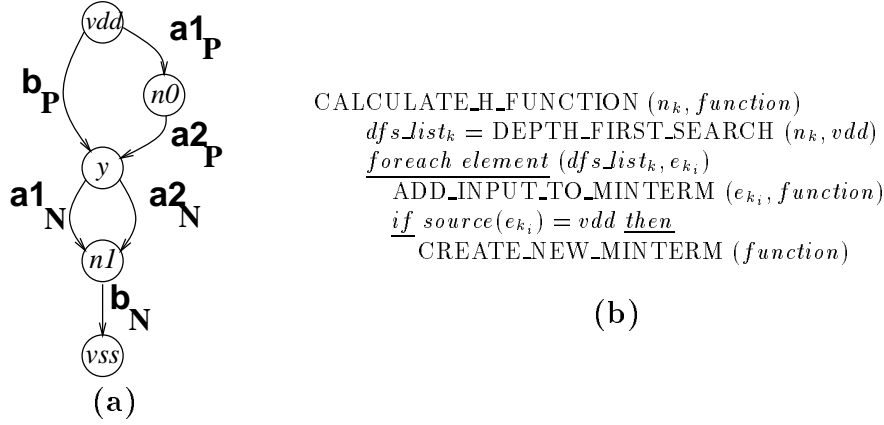


Figure 5.5 (a) Static CMOS gate representation and (b) algorithm to obtain function H_{n_k} .

We represent a static CMOS gate as a directed acyclic graph (V, E) . $V = \{n_0 \dots n_{p-1}, y, vdd, vss\}$ is the set of nodes representing the p internal nodes of the gate ($n_0 \dots n_{p-1}$), the output node (y) and the power and ground nodes (vdd, vss). The set of edges representing the $2q$ transistors (q of type P and q of type N) that connect nodes in V is $E = \{e_{0_P} \dots e_{q-1_P}, e_{0_N} \dots e_{q-1_N}\}$. Figure 5.5(a) shows the graph representation of gate (C) in Figure 5.1(a). Note that this representation retains the transistor order information of the gate.

The power consumption of a node (output or internal) of a gate is potentially affected by all the inputs of the gate. In particular, the power consumption of node n_k produced by input x_i ($W_{n_k} | x_i$) is:

$$\frac{\frac{1}{2} V_{DD}^2 C_{n_k} T_{x_i \rightarrow n_k}}{T_{cyc}},$$

where $T_{x_i \rightarrow n_k}$ is the number of transitions at node n_k because of input x_i . In other words, $T_{x_i \rightarrow n_k}$ expresses how many signal transitions of x_i (D_{x_i}) are propagated to node n_k . C_{n_k} is the capacitance of node n_k .

Henceforth, we assume that a node is charged (discharged) only when there is a direct path from power (ground) supply to the node, i.e. we do not consider charge sharing among the nodes of the gate.

Computation of the model

To compute $T_{x_i \rightarrow n_k}$, the boolean function that represents all possible paths from power supply to node n_k needs to be calculated. Let us call H_{n_k} this function;

similarly, G_{n_k} is the boolean function that represents all possible paths from n_k to ground¹.

The algorithm to obtain the H_{n_k} function is depicted in Figure 5.5(b). Function H_{n_k} is obtained by generating a minterm for each possible path from node n_k to supply node vdd . A path from node n_k to vdd is a set of r edges e_{k_i} so that $dst(e_{k_0}) = n_k, dst(e_{k_1}) = src(e_{k_0}), \dots, dst(e_{k_{r-1}}) = src(e_{k_{r-2}})$ and $src(e_{k_{r-1}}) = vdd$, where $src(e_k)$ ($dst(e_k)$) is the source (destination) node of edge e_k . Using a depth-first-search approach [CLR90], a list of all edges to visit is created (*depth_first_search_list_k*). Afterwards, the edges of this list are added to the current minterm of the H_{n_k} boolean function (`ADD_INPUT_TO_MINTERM()`) until an edge e_{k_j} is reached so that $src(e_{k_j}) = vdd$. In this case, a new minterm is created (`CREATE_NEW_MINTERM()`), sharing with the last created minterm all its edges but the last one visited.

In the example of Figure 5.5(a), the four minterms generated when calculating H_{n_1} are $\{a1\bar{b}, a1\bar{a}2, \bar{a}1, a2\bar{b}, a2\bar{a}2a1\}$, leading to $H_{n_1} = \bar{b}(a1 + a2)$. Similarly, G_{n_k} can also be derived. In Figure 5.5(a), $G_{n_1} = b$. The time complexity of these algorithms is linear in the number of transistors of the gate.

Afterwards, the boolean difference of function H_{n_k} with respect to input x_i (that is $\frac{\partial H_{n_k}}{\partial x_i}$) and the equilibrium probabilities of node n_k need to be calculated. The boolean function $\frac{\partial H_{n_k}}{\partial x_i}$ is calculated as explained in Section 5.3.2. The equilibrium probability of node n_k is obtained as follows [HZA94]: the probability of node n_k of being “1” at a given instant of time ($P(n_k)|_c$) is the probability that n_k was “1” in the instant before ($P(n_k)|_b$) and it is not discharged ($\overline{P(\frac{\partial G_{n_k}}{\partial x_i})}$) or that it was “0” ($\overline{P(n_k)|_b}$) and it is charged ($P(\frac{\partial H_{n_k}}{\partial x_i})$), i.e.:

$$P(n_k)|_c = P(n_k)|_b \overline{P(\frac{\partial G_{n_k}}{\partial x_i})} + \overline{P(n_k)|_b} P(\frac{\partial H_{n_k}}{\partial x_i}),$$

where $\overline{P(f)} = 1 - P(f)$.

Since all signals are assumed to be 0-1 stationary Markov processes, the steady state value of $P(n_k)$ can be derived as:

$$P(n_k) = \frac{P(\frac{\partial H_{n_k}}{\partial x_i})}{P(\frac{\partial H_{n_k}}{\partial x_i}) + P(\frac{\partial G_{n_k}}{\partial x_i})}.$$

¹Note that G_{n_k} and H_{n_k} are complementary functions only when n_k is the output node (y) of the gate.

We conclude that $W_{n_k} |_{x_i}$ is:

$$\frac{\frac{1}{2} V_{DD}^2 C_{n_k} D(x_i) (P(\frac{\partial H_{n_k}}{\partial x_i}) \overline{P(n_k)} + P(\frac{\partial G_{n_k}}{\partial x_i}) P(n_k))}{T_{cyc}} .$$

If the contributions of all nodes (output and internal) are taken into account, the power estimation of the gate is obtained as:

$$P_{gate} = \sum_{k=0}^{p-1} (\sum_{i=0}^{n-1} W_{n_k} |_{x_i}) + \sum_{i=0}^{n-1} W_y |_{x_i} ,$$

where p is the number of internal nodes and n is the number of inputs of the gate. The power consumption of a circuit is, then, the sum of the power consumption of its gates.

Computing functions H_{n_k} and G_{n_k}

Boolean functions H_{n_k} and G_{n_k} that appear in the model in the last section may be symbolically calculated using Binary Decision Diagrams (BDDs) [Bry86].

There exist several representations of boolean functions (truth table, Karnaugh's map, canonical sum, canonical product, product of sums, sum of products, multi-level form, etc.). Among these, the canonical representations present the desirable property that if two functions are equivalent, their representations are the same, and vice-versa. Therefore, the equivalence test is straightforward. A form is canonical if the representation of a function in that form is unique. The problem of canonical representations is that they are large (typically exponential in the number of variables). A BDD is a canonical and compact (for many functions) of a boolean function.

Given a boolean function $f(\dots, x_{j+1}, x_j, x_{j-1}, \dots)$ the positive and negative cofactors of f with respect to x_j are defined by $f_{x_j} = f(\dots, x_{j+1}, 1, x_{j-1}, \dots)$ and $f_{\overline{x_j}} = f(\dots, x_{j+1}, 0, x_{j-1}, \dots)$, respectively. A BDD of a boolean function is a directed acyclic graph based on the successive Shannon decompositions of f given a sequence of splitting variables x_i . Each non-terminal N_{x_j} corresponds to a splitting variable x_j and is the origin of two arcs: $\text{low}(N_{x_j})$ and $\text{high}(N_{x_j})$. The subgraph rooted at $\text{low}(N_{x_j})$ represents the negative cofactor $f_{\overline{x_j}}$, and the graph rooted at $\text{high}(N_{x_j})$ represents the positive cofactor f_{x_j} .

Figure 5.6(a) shows a BDD that represents the function $f(a, b, c, d) = abc + \overline{b}d + \overline{c}d$. Edges are assumed to go from the top to the bottom terminal. A negative cofactor is

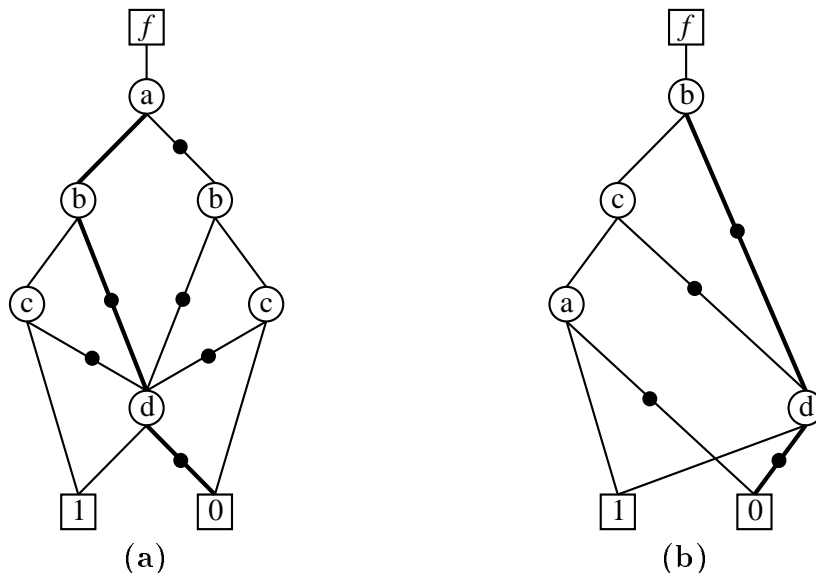


Figure 5.6 Two BDD representations of function $f = abc + \bar{b}d + \bar{c}d$ using different variable order.

shown with a circle in the edge. There are one root node (the name of the function) and two terminal nodes, “0” and “1”. Therefore, to evaluate function f for some variable values, we start at the root node and, for each variable, we choose the edge representing the positive cofactor (if the variable is 1) or the negative cofactor (if it is 0). The explored path for $f(a, b, c, d) = (1, 0, 1, 0)$ is shown in bold in Figure 5.6(a), obtaining the value “0”.

The complexity in terms of node count and graph depth strongly depends on the choice of the splitting variables. The same function f is represented in Figure 5.6(b) using a different variable order. A good reordering is essential for obtaining a small graph. Some heuristics have been proposed to find a good ordering for a BDD [FFK88, MWBSV88]. An ordered BDD (OBDD) is a BDD where variables appear in the same order along all paths from the root to the terminal nodes.

Moreover, an OBDD can be reduced by iteratively applying the reductions:

- identification of isomorphic subgraphs
- elimination of redundant nodes

to obtain a reduced OBDD (ROBDD). An ROBDD is a canonical form.

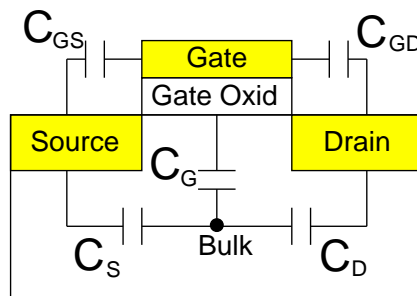


Figure 5.7 Capacitances in a MOS transistor.

In this work, we have used the subroutines provided with the SIS package (see Chapter A, Section A.4) for handling BDDs.

Node capacitances

Several intrinsic capacitances are associated to a MOS transistor (Figure 5.7). The values of these capacitances (and, therefore, the capacitance of an internal node of a gate) clearly depend on the technology and design style used. For example, most gates may be designed using an unbroken row of transistors in which abutting source-drain connections are made. But because of transistors with the same signal on its gate terminal should be vertically aligned to reduce internal routing, the layout algorithm must find the minimum set of transistor chains in order to minimize the number of diffusion gaps. This leads to the fact that two internal nodes with the same number of source-drain connections may actually have different capacitances. Furthermore, source and drain diffusions are different in PMOS and NMOS-type transistors.

Moreover, it is clear that gate output nodes are more capacitive than internal nodes because of the additional routing and transistor-gate (fanout) capacitances.

Because of the task of modeling the capacitances of the nodes of a gate is difficult, these capacitances should be extracted and stored for all gates of the library whenever it is possible. This is the approach followed in this work.

5.4 POWER-OPTIMIZATION ALGORITHM

In this section, an algorithm that traverses the gate description of the circuit is presented. For each gate, it finds the best transistor reordering based on the power-

```

OBTAIN_PROBABILITIES (circuit)
gate_list = DEPTH_FIRST_TRAVERSE (circuit)
for each gate gate in gate_list do
    info_inputs = OBTAIN_PROB_AND_DENS (gate, circuit)
    FIND_BEST_REORDERING (info_inputs, gate, circuit)
    info_output = CALCULATE_DENS (info_inputs, gate)
    UPDATE_CIRCUIT_INFORMATION (info_output, circuit)

```

Figure 5.8 Optimization algorithm.

consumption model explained in Section 5.3 and an exhaustive exploration of all possible reorderings.

5.4.1 Algorithm overview

Finding the best transistor reordering implies an exhaustive exploration of each gate. Since most gates only have a small number of transistors in series, an exhaustive exploration is feasible. The algorithm to obtain all possible transistor reorderings of a gate will be addressed later.

A simplified algorithm of the optimization approach for low power is shown in Figure 5.8. The probabilities for all output nodes of the gates of the circuit are computed in `OBTAIN_PROBABILITIES()`. `DEPTH_FIRST_TRAVERSE()` returns the list of gates (*gate_list*) of the circuit (*circuit*) ordered in a depth-first fashion [CLR90] from the outputs, i.e. every gate appears somewhere after all of its transitive fan-in gates. For each *gate* of this list, the probability and transition density information for all of its inputs is obtained from the circuit (`OBTAIN_PROB_AND_DENS()`). Afterwards, the best reordering is derived for *gate* (`FIND_BEST_REORDERING()`). Finally, the transition density of the output node of the gate is calculated (`CALCULATE_DENS()`) and this information is transferred to the circuit (`UPDATE_CIRCUIT_INFORMATION()`).

5.4.2 Monotonic characteristic

The algorithm takes advantage of the following property of the model explained in Section 5.3: *the reduction of the power in an individual gate always decreases the power of the circuit*. The reason of this monotonic behavior is that all possible transistor reorderings of a gate lead to the same probability and transition density at its output node if the model explained in Section 5.3 is used to compute them. Since:

1. the model precisely relies on the probability and transition density of the inputs of a gate to decrease its power consumption and
2. the power of the circuit is the sum of the power of its gates,

it is clear that the reduction of the power in an individual gate always decreases the total power of the circuit. This monotonic behavior may not correspond to the actual behavior of a circuit, but our results show that this local (greedy) approach results in an overall power reduction for the whole circuit.

Thus, with only one traversal of the circuit, the optimal reordering (always with respect to the model) for all gates is obtained.

5.4.3 Equilibrium probabilities

To apply the power-consumption model explained in Section 5.3, the equilibrium probabilities of the inputs of the gate need to be known. The calculation of the probabilities for all nodes in a circuit that has re-convergent fanout (i.e. there is correlation among the signals) is an NP-hard problem [Naj91]. Without properly partitioning the circuits ([CRP94, Kap94]), the method may not be applicable to very large circuits.

Since having correct equilibrium probabilities is important to obtain a better accuracy, we have implemented the algorithm proposed in [PM75] and suitable to compute probabilities when re-convergent fanout exists.

This algorithm traverses the circuit starting at its inputs and computes the logical expression for the output of each gate as a function of its inputs expressions. Once an expression has been calculated, all its exponents are suppressed to obtain the correct probability expression for that signal.

5.4.4 Exhaustive exploration of gate configurations

Table 5.2 shows the number of possible configurations obtained by reordering the transistors of some standard gates. These configurations are obtained by *pivoting* on the internal nodes of the gate. More formally, this process of obtaining a new configuration is described as follows: let BS be the subgraph (of the graph representing the PMOS or NMOS blocks of a gate) composed of all edges whose source is node n_k and whose destination node is either node n_{bs} or another node n_{dummy} . The same definition is recursively applied to node n_{dummy} until all destination nodes are n_{bs} .

Gate	#Conf.	Gate	#Conf.	Gate	#Conf.
inv	1	aoi21[A,B]	4	oai21[A,B]	4
nand2	2	aoi22	8	oai22	8
nand3	6	aoi31[A,B]	12	oai31[A,B]	12
nand4	24	aoi32[A,B]	24	oai32[A,B]	24
nor2	2	aoi33	72	oai33	72
nor3	6	aoi211[A,B,C]	12	oai211[A,B,C]	12
nor4	24	aoi221[A,B,C]	24	oai221[A,B,C]	24
		aoi222	48	oai222	48

Table 5.2 Number of different configurations for some standard gates obtained by reordering its transistors.

```

PIVOTE_AND_SEARCH (gate_graph, visited_reords, current_node)
  gate_graph = PIVOTING_ON_NODE (gate_graph, current_node)
  if not VISITED (gate_graph, visited_reorderings) then
    visited_reords = ADD_TO_VISITED_REORDS (gate_graph)
    for index = 1 to number_of_internal_nodes do
      if index ≠ current_node then
        PIVOTE_AND_SEARCH (gate_graph, visited_reords, index)

FIND_ALL_REORDERINGS (gate_graph)
  visited_reords ← ∅
  for index = 1 to number_of_internal_nodes do
    PIVOTE_AND_SEARCH (gate_graph, visited_reords, index)

```

Figure 5.9 Exhaustive exploration algorithm.

Similarly, let TS be the subgraph with its associated node n_{ts} . A new configuration of the gate is obtained by interchanging subgraphs BS and TS.

For example, in any configuration of Figure 5.1(a), new configurations are obtained by pivoting on node n_1 , being n_{ts} and n_{bs} , in all cases, y and vss respectively.

For the graph representation of the gate explained in Section 5.3.3, an algorithm that finds all possible transistor reorderings of a gate is presented in Figure 5.9. The strategy of pivoting on an internal node to obtain a new reordering of transistors leads to the generation of repeated reorderings. A dynamic programming approach with memoization [CLR90] is used to avoid the generation of overlapping subproblems.

The algorithm recursively points to an internal node (*current_node*) and *pivots* on it to obtain a new reordering (PIVOTING_ON_INTERNAL_NODE()). Further searching for new reorderings is pruned if the reordering obtained has already been visited (VISITED()).

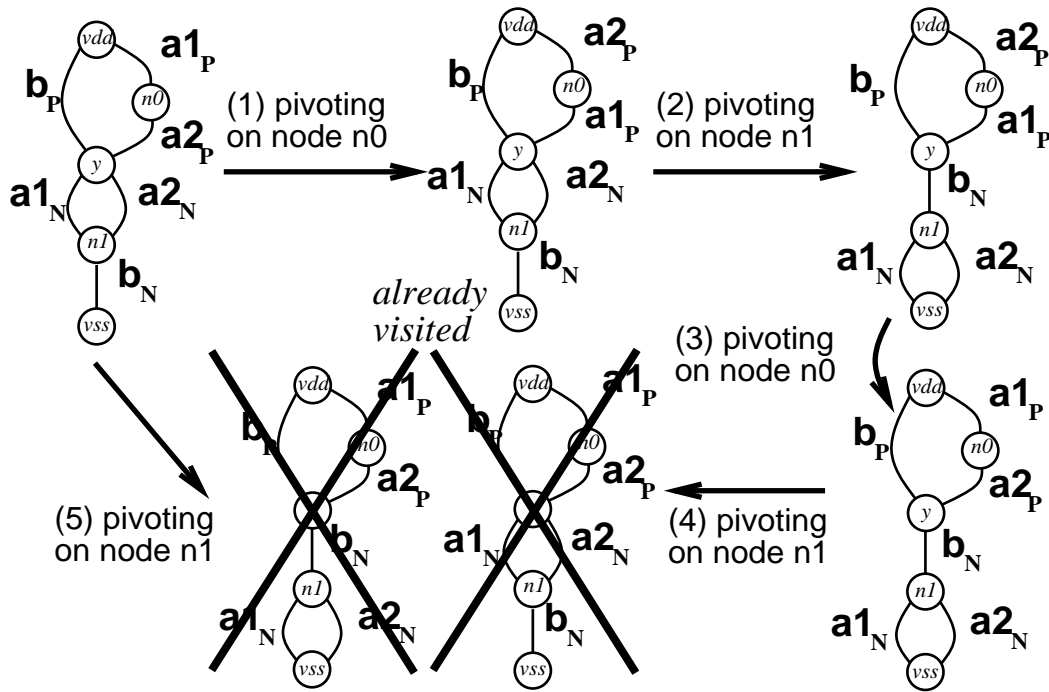


Figure 5.10 Execution example of the exhaustive exploration algorithm.

If it has not been visited, it is added to the set of transistor reorderings of the gate already visited (`ADD_TO_VISITED_REORDERINGS()`) and the algorithm is called again for all internal nodes of the gate except the current one (this is so to prevent the generation of a reordering that we know beforehand that we have already visited).

The algorithm in Figure 5.9 works for gates that can be represented with a series-parallel graph. Almost all the gates of typical libraries can be represented with this type of graphs. Figure 5.11 shows a gate that does not admit a series-parallel representation. The authors are currently working on the formal demonstration that all possible transistor reorderings of a static CMOS gate are generated with this algorithm.

To illustrate how this algorithm works, it has been applied to the gate implementing the function $y = \overline{(a1 + a2)} b$. Figure 5.10 shows the execution. The starting graph representation of the gate is the one in Figure 5.5(a). We observe that all four possible reorderings (those already seen in Figure 5.1(a)) are generated.

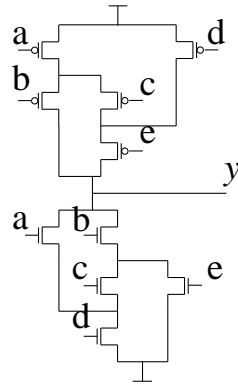


Figure 5.11 A gate ($y = \overline{ad + be + bcd + ace}$) without a series-parallel representation.

5.5 RESULTS

5.5.1 Scenarios for the experiments

A wide range of MCNC [Yan91] circuits have been used as benchmarks. They have been mapped into the gate library shown in Table 5.2. In some cases, to obtain all transistor reorderings of a gate, it is necessary to have more instances of that gate. For example, there are two instances of gate **oai21**: **oai21[A]**, which is able to implement configurations **(A)** and **(B)** of Figure 5.1(a) and **oai21[B]**, which is able to implement configurations **(C)** and **(D)**. All instances of the gates in Table 5.2 have been implemented in a sea-of-gates design style (see Chapter A, Section A.2).

Two scenarios have been considered to evaluate the power-consumption savings obtained with the transistor reordering technique (see Figure 5.12(a)). In **Scenario A**, the circuit is considered to be embedded in a larger digital system. Thus, the equilibrium probability and specially the transition density of the inputs of the circuit may take very different values. In this scenario, the probabilities and transition density of the primary inputs of each circuit are randomly set with a uniform distribution. Probabilities range from 0 to 1 and transition densities range from 0 to 1 million transitions per second. In **Scenario B**, the circuit is considered to be the combinational part of a sequential system. In this scenario, both the probability and transition density of the primary inputs of the circuit may take values between 0 and 1. We have set both values to 0.5 and 0.5 transitions per cycle, respectively.

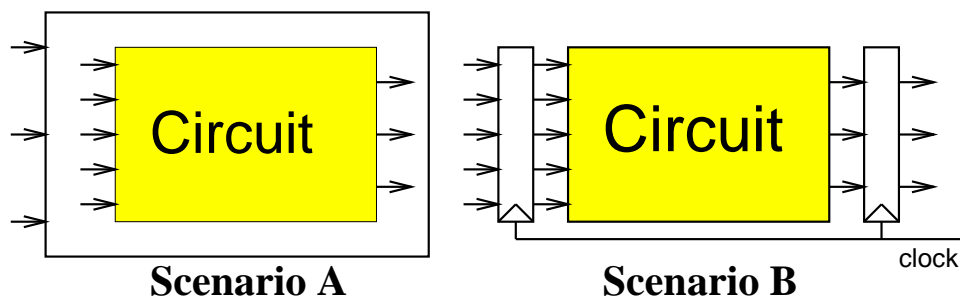


Figure 5.12 The two scenarios considered.

In both scenarios, the optimization algorithm has been applied to the original gate-level description of the circuits to obtain, for each gate, the best instance and, for each instance, the best input reordering. Because of all instances of the same gate have the same area, the total area of the optimized circuit remains the same.

For each scenario and circuit, two new gate-level descriptions have been created. One of them contains the best transistor reordering for low power for all gates whereas the other contains the worst one. A switch-level simulator (see Appendix A, Section A.3) extracts the power consumption of each description. Thus, the maximum power reduction for each scenario is obtained. The input signals to the circuits used by the switch-level simulator have been generated with an exponential distribution, i.e. time intervals between two consecutive transitions of input signal k to the gate follow an exponential distribution with average $1/D_k$, being D_k the transition density of input signal k . The delay information has been obtained with the SIS system (see Appendix A, Section A.4).

5.5.2 Discussion

Table 5.3 shows the results obtained. Columns **Model** and **SLS** show the power-consumption reduction (best case compared to worst case for low power) obtained with the model and with switch-level simulations respectively. Column **Delay** shows the increase in delay (best case for low power compared to a mapping into the original cell library). The delay increases in most of the benchmarks because not always the best transistor reorderings of a gate for low power and low delay coincide. In fact, the rule of thumb that states that the critical transistor should always be placed near the output terminal to obtain a fast gate contradicts the low power rule

of placing it close to the ground node as can be observed in the motivation example (case **(2)**) in Section 5.1.1 and in [SLW95].

It is shown that the average improvement in power consumption in scenario **A** is 12% with an average increase in delay of 4%. The estimated average improvement is 9%. The reason of this lower value in the estimated improvement is that the model, in general, overestimates the power consumption by an offset, thus leading to a lower estimated reduction.

The power reduction in scenario **B** is roughly half the one in scenario **A** because of the smaller amount of the circuits in scenario **B**. The power and delay of latches and the clock line in scenario **B** has not been included in the results. In both scenarios there is a small average increase in delay.

Thus, significant power consumption reduction can be obtained in both scenarios with little average increase in delay and it is possible to achieve power reductions without increasing the delay of the circuit (see, for example, benchmarks *c8* and *too_large* in Table 5.3).

5.6 CONCLUSIONS

This chapter has shown that the performance optimization technique called transistor reordering may be also used to reduce the power consumption of a circuit. Average power reductions of 12% at the expenses of 4% increase in delay have been obtained for a wide range of benchmarks.

An optimization algorithm that uses a stochastic power-consumption model of a static CMOS gate has been presented. This novel power-consumption gate model takes into account both the probabilities and the switching activity of the inputs of the gates that compose the circuit to estimate the activity at the internal nodes of the gate.

The results obtained suggest that:

- current libraries may be upgraded with more instances of the gates with different transistor reorderings, so that an optimization algorithm can choose the best instance for power reduction and
- it is also possible to obtain power reductions without increasing the delay of the circuit.

Circuit name	Gates	Scenario A			Scenario B		
		Model	SLS	Delay	Model	SLS	Delay
alu2	224	9.5	14.6	1.2	5.7	6.4	-0.2
c432	148	6.2	14.1	-0.7	4.3	8.2	-3.7
c499	316	2.8	4.6	-1.6	1.7	3.9	-3.0
c8	99	11.5	13.7	-15.5	4.1	5.1	-2.9
cht	117	9.9	9.3	4.4	3.7	6.9	2.5
cm150a	43	7.8	11.9	1.1	2.7	5.5	10.8
cm85a	24	15.4	20.0	0.0	8.7	15.3	-1.2
comp	94	5.8	10.2	17.5	2.2	3.7	7.4
cordic	64	6.5	11.9	-2.7	1.8	1.6	-0.7
i5	244	10.2	12.5	9.4	3.8	7.4	14.4
mux	55	9.7	12.5	13.3	4.1	5.5	-2.6
my_adder	128	4.3	12.3	6.2	1.7	0.7	5.4
parity	45	2.8	5.9	1.1	0.6	0.1	0.0
too_large	459	10.5	11.0	-8.0	4.6	3.2	-2.6
x1	192	11.2	12.4	11.7	5.3	4.9	10.2
x4	313	11.0	13.3	-2.4	5.1	6.4	-1.9
pcl	47	8.4	13.6	11.8	6.4	10.0	15.3
pcler8	64	7.7	16.8	13.8	6.4	11.5	14.1
frg1	67	12.2	15.2	8.4	4.9	8.3	4.4
sct	62	13.0	13.8	1.5	5.5	3.6	8.1
unreg	49	8.4	3.8	0.0	0.8	0.1	-14.0
z4ml	41	10.7	15.4	-5.3	3.0	1.7	-4.9
f51ml	73	13.8	15.0	7.5	4.5	5.4	-4.3
symml	84	12.7	12.6	3.7	3.0	3.3	-4.9
apex7	155	7.9	11.5	11.7	4.7	8.3	5.9
count	80	11.8	18.2	4.0	6.1	9.6	5.3
c1355	540	2.1	2.9	2.3	1.9	4.6	2.2
c1908	401	5.4	9.0	1.0	3.5	5.0	-2.0
c880	235	8.6	13.6	-0.4	4.5	10.2	-6.4
alu4	424	7.9	12.0	3.6	5.7	7.7	4.3
apex6	442	7.5	12.3	0.2	4.2	7.1	-3.4
example2	222	6.2	8.4	17.2	2.1	2.5	16.4
i6	284	7.7	7.6	8.1	1.6	4.6	-6.1
i7	411	8.4	7.9	9.6	1.2	2.2	10.9
i9	516	5.6	3.3	1.3	4.5	5.5	0.3
rot	408	8.8	12.3	13.5	4.6	7.0	15.2
term1	206	12.9	13.7	6.0	6.5	5.7	-4.7
ttt2	132	12.5	13.4	-11.2	7.1	5.7	-9.2
x3	485	11.0	13.3	-2.4	7.0	7.1	17.2
Average		8.9	11.7	3.6	4.1	5.7	2.3

Table 5.3 Results obtained for several MCNC benchmarks for both scenarios considered.

CONCLUSIONS AND FUTURE RESEARCH

This chapter presents the conclusions for the contributions of this work, namely some low-power design techniques at the system, architecture and logic/circuit layers of the design process of a circuit. In addition, some research directions derived by these techniques are discussed.

6.1 INTRODUCTION

Concern about the increasing power consumption of the circuits has grown in the recent years among the semiconductor industries. Issues like battery life, chip package and cooling devices, reliability, etc. are becoming key factors for the successful outcome of the products for a large sector of the microprocessor market.

The main factor that has driven most of the efforts in low-power design is the increasing demand of portable battery-operated applications such as notebook and laptop computers or PDAs (Personal Digital Assistants). In these applications, one of the primary design constraints is the battery life.

Thus, circuit designers face a new dimension (power) in the traditional two-dimensional (area and delay) search space, with the associated increase in complexity. There exist few CAD tools that take into account the power consumption during the design process; therefore, new low-power design techniques are needed to help the designer in the complex task of meeting area, delay and power constraints.

There are two main research fields in the low-power design area: power estimation and low-power design. Both are tightly related since a previous analysis of the power consumption of the design to be implemented is needed to understand where the power goes, i.e. which are the most power-consuming parts that need to be targeted by the low-power design techniques. In this work we have focused on low-power design techniques.

The power consumed by a circuit is divided into static and dynamic. Static power is consumed when there is no activity. Transistors do not behave as perfect switches and, therefore, leakage currents (caused by parasitic diodes) and static currents (a function of the input voltage and the threshold voltage of the transistors) arise. The contribution of the static consumption is less than 2% of the total power and it is usually neglected.

Dynamic power is consumed when there is activity. This component of the power is divided into:

- short-circuit power consumption: a short-circuit current arises when there is a path between power supply and ground where all transistors are on. The power associated to this effect can be kept below 10-15% of the total power for data-path circuits.
- capacitance charge/discharge power consumption: whenever a circuit node changes its value, its associated capacitance has to be charged or discharged, thus

consuming power. For data-path circuits, this component accounts roughly for the 80% of the total power.

Considering only the capacitance charge/discharge power component, the power consumption of an output node i of a gate is defined as:

$$\frac{1}{2} (C_i V_{DD}^2 \alpha f) ,$$

where V_{DD} is the power supply, f is the operating frequency and α is the number of times that the capacitance C_i is charged or discharged per cycle. This equation reveals the three degrees of freedom when designing for low power: voltage, physical capacitance and activity. Once the supply voltage and the device dimensions have been fixed, it is the switching activity of the signals of the circuit that will ultimately determine its power dissipation.

Techniques for reducing this activity are the contribution of this work. Several low-power design techniques have been reviewed covering all the layers of the design hierarchy:

- system (system shutdown, system partitioning, algorithm selection)
- architecture (parallel and pipelined processing with voltage scaling, compiler transformations, cache design, data representation)
- logic (path balancing and gate sizing, multi-level logic synthesis, technology mapping, FSM state assignment, re-timing, disabling registers)
- circuit (logic style, asynchronous design, design style, adiabatic computing)
- layout (floorplanning, placement and routing)
- technology (technology scaling and SOI process, packaging technology).

The highest power savings are obtained at the architecture and system layers. However, significant power reductions are also achieved at the logic and circuit layers, which may add up to those already obtained at the higher layers.

6.2 CONTRIBUTIONS

This work contributes to the design of low-power at the system, architecture and logic/circuit layers of the design process. The contributions have been published in several international conferences [MC95a, MC95b, MC95c, MC96].

6.2.1 System layer

At the system layer, a set of high-level synthesis techniques (loop interchange, operand reordering, operand sharing, operand retaining, operand similarity) that tackle the problem of power consumption at the register-transfer level (RTL) have been presented and evaluated using an appropriate high-level power-consumption model. Rather than accurately estimating the power consumption of the final implementation, we focus on fairly compare the relative benefits of different RTL descriptions.

The common idea behind these techniques is to reduce the activity at the inputs of the functional units (the most power-consuming units in an RTL architecture). To cope with the high complexity of the problem of power reduction, all the techniques are based on heuristics which trade, in some cases, power for area and performance. Each of the techniques presented has been evaluated with some benchmarks in which there is a clear evidence that significant benefits can be obtained. Up to 34% power reductions in the functional units have been evaluated. The techniques are complementary among them and not every technique produces a tangible improvement on every benchmark. Moreover, the implementation of these techniques may cause an increase in area. In the evaluation of the power-consumption reductions, we have not taken into account the power of this additional circuitry. However, we believe that it is negligible compared to the power consumption of the functional units.

The high-level synthesis tasks of scheduling and register binding have been automated and power/area and power/delay trade-offs have been studied. The scheduling algorithm for low power uses a list-scheduling approach where the priorities of the operations of the ready-operation queue are set in such a way that operations sharing the same operand are bound to the same functional unit and scheduled so that the functional unit can reuse that operand. Significant power reductions are obtained in the scheduling task (up to 17%) with little increase or no increase at all in latency.

The register-binding algorithm for low power is based on clique partitioning of a restricted variable-lifetime compatibility graph to obtain a register set that, for each functional unit, reduces the power consumption during its idle cycles. Power consumption in the functional units during non-idle cycles is decreased by taking into account the average Hamming distance among the variables of the behavioral description and the commutative property of some operations. The power reductions obtained (5-8%) consider all units of the RTL design, not only the functional units.

Although the scheduling and register-binding techniques for low power are compatible and complementary, the scheduling technique obtains better improvements if applied to dense schedules (e.g. schedules where the functional unit occupation is high) whereas the register-binding technique is more suitable to sparse schedules.

Both algorithms of scheduling and register binding for low power trade off, in some cases, latency and area for power. They should be modified to achieve, first, the same performance or area as the original algorithms and, later, as much power reduction as possible. Moreover, the algorithms proposed are modifications of two traditional approaches for scheduling (list-based) and binding (clique covering). Other approaches for the scheduling and binding tasks (force-directed, greedy, integer programming) should be investigated.

The techniques presented only focus on functional units since we target at data-intensive applications. For memory-intensive circuits, different techniques should be devised that reduce the amount of memory references.

6.2.2 Architecture layer

Useless activity does not contribute to the calculation of the final result of the circuit and only dissipates power. Useless activity is the main power-consumption source in a class of circuits that present a regular, layered structure, where each layer is driven by primary inputs from the very beginning. Moreover, the useless/useful activity ratio increases with the bit width of the inputs of the circuit. Array multipliers belong to this class of circuits.

The technique proposed to reduce the useless activity is based on the insertion of self-timed controlled latches on the high-active paths of the circuit. The enabling signals for these latches are generated by delaying an input signal. This technique has been implemented and evaluated for several configurations of array multipliers.

An implementation of this technique that retains the regularity of the array structure consists of inserting the latches following the layered structure of the array, i.e. latches are inserted acting as a transition-retaining barrier (TRB) for the useless activity.

Two different trade-offs have been studied:

- area/power: the more number of latches are inserted, the more number of useless transitions are eliminated. However, latches and their associated delay cells

increase the total area and consume power. As an example, the insertion of a TRB in an 8×8 -bit array multiplier increases the power instead of reducing it.

- delay/power: if the delay of the delay-cells used for the enabling of the TRBs is greater than the delay of the basic cell of the array, the multipliers will present less power but will be slower. If not, the TRBs will switch into transparent mode too early, allowing some useless transitions to pass through and generate more useless activity in the next part of the array.

The best results are obtained when three TRBs are inserted. However, it is important to notice that the greatest decrease in power is achieved when inserting one TRB compared to the original design rather than when inserting two TRBs compared to inserting just one. There is an optimal (for power consumption) number of TRBs. This number is a complex function of the size of the multiplier, the useless activity generated and the power consumption of the TRBs and delay cells. The evaluation of this optimal number has not been addressed in this work and it is left as future research.

With three TRBs, we achieve a reduction in power of 30% in a 32×32 -bit array multiplier with an increase in area and delay of 4% and 8% respectively. Further power savings are obtained with a proper design of the basic building block of the multiplier (the full-adder cell) with its balanced paths so that the generation of useless transitions is minimized.

The final multiplier remains combinational and highly regular. This implies that the TRB technique can be easily integrated in existing CAD tools for the generation of array multipliers.

6.2.3 Logic/circuit layer

Some complex logic functions accept several static CMOS implementations that present a different order of the transistors. The transistor reordering technique has already been proposed to decrease the propagation delay of the gate. We have used this technique to find a proper order of the transistors of a gate so that the switching activity at the internal nodes of the gate is minimized.

The power consumption at the internal nodes of a gate depends on the signal probabilities and the amount of switching activity at the inputs of that gate, and the order of the transistors that compose the gate. Therefore, an stochastic power-consumption model of a gate that takes into account these parameters has been proposed. One drawback of this model is that it is based on the transition density

measure of the activity. This measure, although fast to evaluate, does not take into account the correlation (spatial or temporal) of the input signals to the gate. Other more accurate (but more time consuming) models that take into account the signal correlation should be used to evaluate the power reductions.

This model is the core of an optimization algorithm that traverses that gate description of a circuit and, for each gate, it finds the best order of the transistors for low power. This algorithm presents the following property: the power reduction in an individual gate always decreases the power of the circuit. The reason of this monotonic behavior is that all possible transistor reorderings of a gate lead to the same probability and switching activity at its output node if the model mentioned before is used to compute them.

Moreover, this algorithm works for gates that can be represented with a series-parallel graph. Almost all the gates of typical libraries can be represented with this type of graphs.

Finding the best transistor reordering implies an exhaustive exploration of each gate. Since most gates only have a small number of transistors in series, an exhaustive exploration is feasible. An algorithm that performs this exploration is presented and it is based on a graph representation of a CMOS gate that preserves the order of the transistors.

Two scenarios have been considered to evaluate the power-consumption savings obtained with the transistor-reordering technique. The main difference between these scenarios is the different amount of switching activity that the primary inputs present. A standard gate library has been implemented with all the necessary instances of a gate to obtain all transistor reorderings for that gate.

The results obtained show that:

- the power consumption is always decreased.
- an average power reduction of 12% and up to 20% may be achieved.
- lower power reductions (6% in average) are obtained for the scenario with less switching activity at the primary inputs.
- the delay increases in most of the benchmarks because the best transistor reordering for low power usually does not coincide with the best one for performance.
- in some circuits, both power and delay may be reduced at the same time.

- the reductions predicted by the model underestimate the results obtained with a switch-level simulator because the model, in general, overestimates the power consumption by an offset, thus leading to a lower estimated reduction.

6.3 FUTURE RESEARCH

The techniques presented as the contribution of this work may be improved to obtain better power reductions and extend their applicability domain. Some of the future research lines are outlined for each of the three parts of the contributions of this work.

With regard to the high-level synthesis techniques,

- evaluate the area and delay overhead and implement those techniques that have not been automated (loop interchange, operand reordering).
- focus on the power consumption of memory-intensive applications and propose techniques to reduce the number of memory references (for example, study a mechanism to map the arrays to memory so that the activity at the address bus is minimized).
- modify the scheduling and register binding algorithms for low power to achieve the same performance and area as the original algorithms.
- investigate other approaches (force-directed, greedy, integer programming) to the scheduling and resource binding for low power.

With regard to the transition-retaining barrier technique,

- use a full-adder design with balanced paths so that the generation of useless transitions is minimized.
- reduce the area overhead by not inserting those latches that would eliminate a small amount of useless activity.
- experimentally calculate the optimal (for power) number of TRBs.
- evaluate this technique for other types of circuits.

And with regard to the transistor reordering technique,

- upgrade other standard libraries with more instances of the gates with different transistor reorderings and evaluate again the technique.
- modify the power-optimization algorithm to apply the transistor-reordering technique only to those gates that are not in the critical path of the circuit, thus not increasing the delay.
- extend the stochastic power-consumption model of the static CMOS gate to be applicable to other logic families and to static CMOS gates that do not admit a series-parallel representation.
- use other more accurate measure of the switching activity that takes into account the spatial and temporal correlation of the input signals to the gate.

CIRCUIT DESIGN AND ANALYSIS TOOLS

The designs proposed in this work have been implemented and evaluated with three circuit design and analysis tools: Ocean, SLS and SIS. In this chapter, these tools are reviewed. The design and analysis of a full-adder circuit is presented as an example.

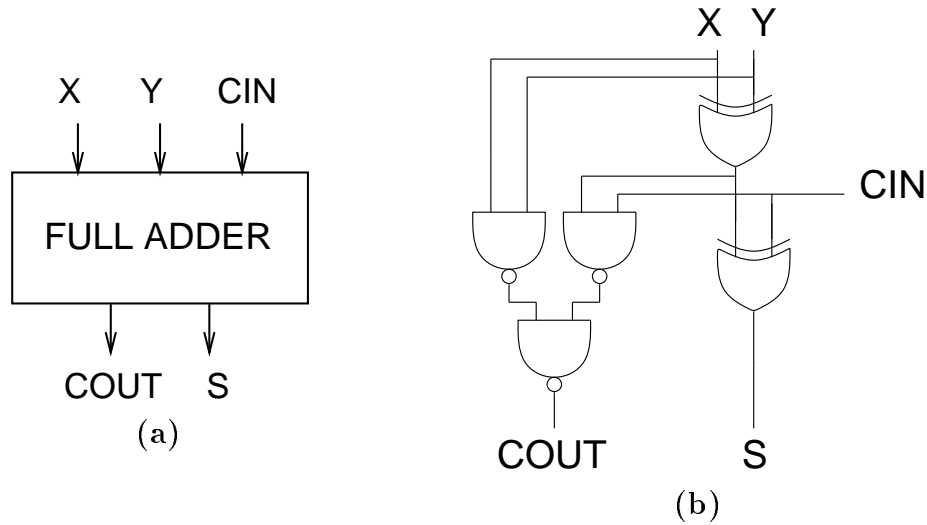


Figure A.1 (a) Schematic and (b) gate description of a full adder.

A.1 INTRODUCTION

In this work three tools have been used to implement and evaluate the proposed designs. All the tools have been developed at universities.

The **Ocean** design system [GS93] has been used to implement, in a sea-of-gates design style, some of the cells and circuits proposed in this work. It has provided also an accurate estimation of the area at the layout level.

The **SLS** switch-level simulator [vG89] has been used to obtain the power consumption of the circuits either implemented with the Ocean system implemented using its gate libraries. Also timing analysis is provided with this simulator.

The **SIS** synthesis system [SSL⁺92] has been used to calculate the critical path of a gate-level circuit description.

In this chapter we review some of the features of these three tools. The full-adder circuit of Figure A.1 is implemented and analyzed with the tools.

A.2 OCEAN

Ocean [GS93] is a chip design package developed at Delft University of Technology, the Netherlands. It includes a set of tools for the synthesis and verification of semi-custom sea-of-gates and gate-array [WE93] chips. Its main features are:

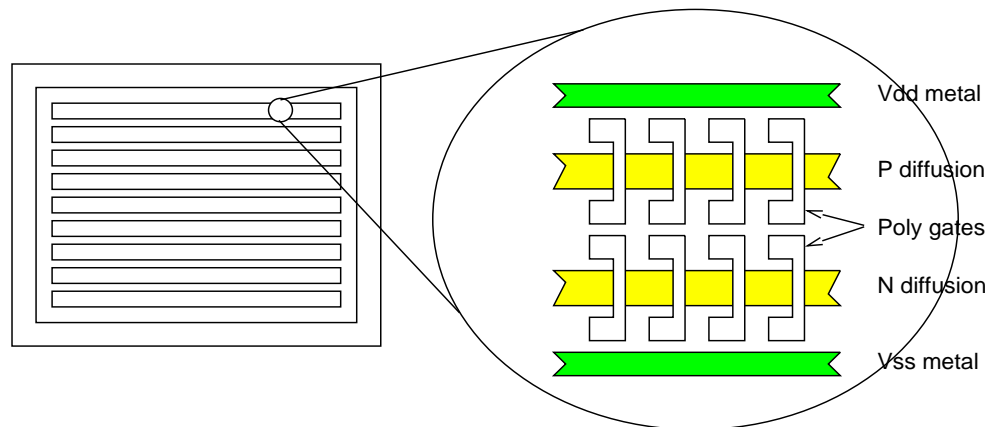


Figure A.2 Sea-of-gates layout architecture.

- hierarchical (full-custom like) layout style on sea-of-gates.
- automatic tools for placement, routing and extraction.

Sea-of-gates [WE93] is a design option where the design cost of the integrated circuit is reduced since the designer only has to define the connections with metals, contacts and vias. The core of the chip already contains a continuous array of N- and P-transistors. Those rows are repeated vertically (Figure A.2). Routing channels are formed by routing over the top of unused transistors. In contrast to the conventional gate-arrays, a sea-of-gates image does not have pre-defined routing channels. This enables a much more compact implementation of structured circuits such as processors.

Most designs choose equally sized transistors to equalize rise and fall times. The absolute size of transistors is a trade-off between drive capability, fan-in loading, and the array density required. The size of the transistors also affects the routing tracks.

The package is provided with some standard gate libraries which have been used to implement the circuits throughout this work. Whenever needed, we have implemented our own gates. The full-adder circuit in Figure A.1 may be described with a network description language as shown in Figure A.3(a). Gates **na210** and **ex210** belong to the standard digital library provided with the Ocean package.

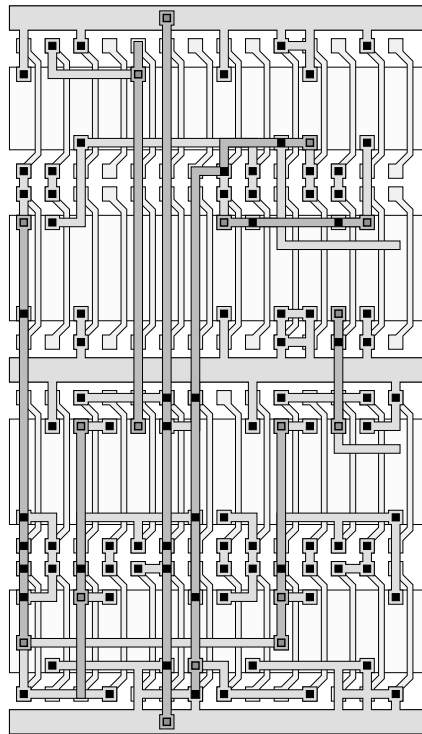
Once this description of the circuit is compiled and entered into a database, the Ocean system is able to perform an automatic placement and routing. The resulting

```

extern network na210 (terminal A,B,Y,vss,vdd)
extern network ex210 (terminal A,B,Y,vss,vdd)
network full_adder (terminal X,Y,CIN,S,COUT,vss,vdd)
{
  ex210 (X,Y,V0,vss,vdd);
  ex210 (V0,CIN,S,vss,vdd);
  na210 (X,Y,V1,vss,vdd);
  na210 (V0,CIN,V2,vss,vdd);
  na210 (V1,V2,COUT,vss,vdd);
}

```

(a)



(b)

Figure A.3 (a) Full-adder netlist description and (b) layout produced by Ocean.

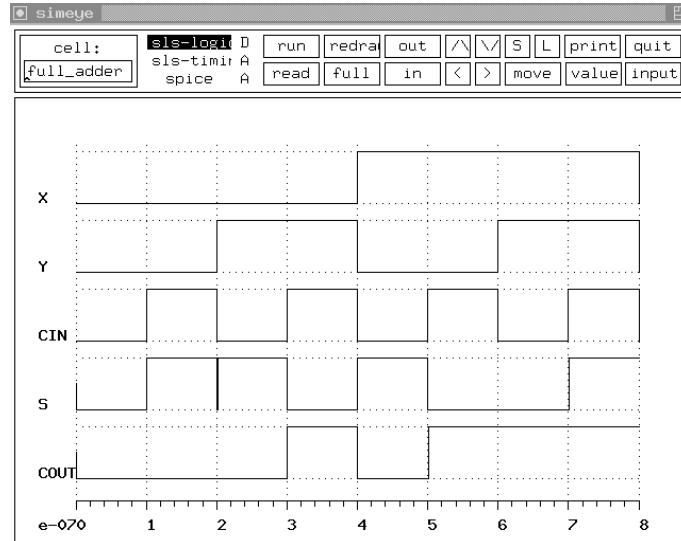
circuit is shown in Figure A.3(b). The transistors and capacitances of this layout can be extracted and saved into the database for later simulations.

```

print X,Y,CIN,S,COU
option level = 3
option sigunit = 100e-09
dissipation
set vdd = h*
set vss = l*
set X = (l*4 h*4)*
set Y = (l*2 h*2)*
set CIN = (l*1 h*1)*

```

(a)



(b)

Figure A.4 (a) Command file and (b) graphical output of the SLS simulator.

A.3 SLS

SLS [vG89] is a switch-level simulator developed at Delft University of Technology, the Netherlands. It can be used for simulating the logical and timing behavior of digital MOS circuits at three levels:

- level 1: purely logic without considering the actual circuit parameters.
- level 2: logic simulation based on actual circuit parameters (transistor dimensions and interconnection resistances and capacitances).
- level 3: same as level 2 but with timing. To find the delay times, the simulator uses approximating piece-wise-linear voltage waveform that are found by performing RC constant calculations.

One important feature of the SLS simulator is the possibility to perform mixed-level simulations for transistor-level, gate-level and function-level circuits. Therefore, a network may consist of some functional blocks that will be functionally simulated. A functional block is a software model of a digital design. It manipulates its inputs and produces outputs. The language in which the function blocks is described is mainly the C programming language.

A separate file (the command file) provides the simulation control commands. A possible command file for the full-adder example that explores all possible combi-

nation at its inputs is shown in Figure A.4(a). With the *dissipation* command, the simulator is able to obtain the total dynamic dissipation if the circuit is described at the transistor level. It may also provide the average dissipation for different time intervals. In our example, the power dissipation can be obtained since we are simulating the layout of Figure A.3(b).

The total dissipation is obtained by adding the values $0.5 C_i V_{step_i}^2$ for all node transitions in the circuit. C_i is the capacitance of a node i where a voltage change occurs and V_{step_i} is the value of the voltage change at that node i . The total dissipation is divided by the simulation time.

The delay model (simulation at level 3) causes that more accurate values for the dissipation are obtained than no delay model (simulation at level 1 and 2). At level 3, even the size of transient effects (glitches) is taken into account to compute the dissipation. At level 1 and 2, glitches are also considered to compute the dissipation, but they may be much less realistic. In this work, we have always simulated the circuits at level 3.

The simulation outputs can be visualized either in graphic or text form. Figure A.4(b) shows the graphical results of the simulation of the full-adder circuit using the command file in Figure A.4(a).

A.4 SIS

SIS [SSL⁺92] is an interactive tool for synthesis and optimization of sequential circuits developed at the University of California, Berkeley. Although it focuses on sequential systems, it implements techniques implemented to handle the combinational blocks of these systems.

The SIS package has several features:

- signal-transition graph (STG) manipulation (state minimization, state assignment, STG extraction).
- combinational optimization (node simplification, kernel and cube extraction, test pattern generation, technology mapping, restructuring for performance, delay analysis).
- sequential optimization (re-timing and re-synthesis, technology mapping, state enumeration).

```

# LSILOGIC 0.5u 3.3V
GATE nd2 1.0 O = !(a * b);
  PIN a INV 0.9 999.0 0.06 0.035 0.08 0.025
  PIN b INV 1.0 999.0 0.06 0.035 0.08 0.025
GATE eo 3.0 O = ((!a * b) + (a * !b));
  PIN a UNKNOWN 2.0 999.0 0.16 0.035 0.14 0.025
  PIN b UNKNOWN 1.1 999.0 0.30 0.035 0.29 0.025

```

(a)

```

.model full_adder
.inputs X Y CIN
.outputs S COUT
.gate eo a=X b=Y O=V0
.gate eo a=V0 b=CIN O=S
.gate nd2 a=X b=Y O=V1
.gate nd2 a=V0 b=CIN O=V2
.gate nd2 a=V1 b=V2 O=COUT
.end

```

(b)

Figure A.5 (a) Library and (b) circuit description in SIS.

- asynchronous synthesis (hazard-free synthesis with unbounded gate-delay, synthesis with bounded gate-delay, removing hazards with bounded wire-delay, ensuring complete state coding and persistency, etc.)

that have been built on top of the MISII [BRSVW87] system.

In this work we have mainly used the delay analysis and technology mapping features for the combinational circuits. To perform the technology mapping, a description of the targeted gate library has to be provided to SIS. The **genlib** format is used for this purpose. For example, Figure A.5(a) shows the description of a two-input NAND and XOR gate of the LSILOGIC library [LSI94]. The logic function, area and delay information is provided.

The **blif** format is used to describe a logic-level hierarchical circuit. This format is different from the one used with the Ocean package. The same description of the full adder in **blif** format is found in Figure A.5(b).

Once the SIS system has read both descriptions of the library and circuit, delay analysis may be performed to obtain, among other information, the critical time of the circuit.

REFERENCES

- [AK84] J.R. Allen and K. Kennedy. Automatic loop interchange. In *Proc. of the SIGPLAN Symp. on Compiler Construction*, pages 233–246, 1984.
- [AMD⁺94] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Paefthymiou. Precomputation-based sequential logic optimization for low power. *IEEE Transactions on VLSI Systems*, 2(4):426–436, December 1994.
- [ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, 1986.
- [BAF94] J. Bunda, W.C. Athas, and D. Fussell. Evaluating power implications of CMOS microprocessor design decisions. In *Proc. International Workshop on Low Power Design*, pages 147–152, April 1994.
- [BB95] T.D. Burd and R.W. Brothersen. Energy efficient CMOS microprocessor design. In *Proc. 28th Annual Hawaii International Conf. on System Sciences*, January 1995.
- [BCH⁺94] R.I. Bahar, H. Cho, G.D. Hachtel, E. Macii, and F. Somenzi. A symbolic method to reduce power consumption of circuits containing false paths. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 368–371, November 1994.
- [BCS92] R. Brodersen, A. Chandrakasan, and S. Sheng. Low-power signal processing systems. In *Proc. of the IEEE VLSI Signal Processing Workshop*, 1992.
- [BE95] A. Bellaouar and M.I. Elmasry. *Low-power digital VLSI design: circuits and systems*. Kluwer Academic Publishers, 1995.
- [BFR94] L. Benini, M. Favalli, and B. Riccò. Analysis of hazard contributions to power dissipation in CMOS ICs. In *Proc. International Workshop on Low Power Design*, pages 27–32, April 1994.
- [BGS93] D.F. Bacon, S.L. Graham, and O.L. Sharp. Compiler techniques for high-performance computing. Technical Report UCB/CSD-93-781, Computer Science Division (EECS), UCB, November 1993.
- [BIO95] M. Borah, M.J. Irwin, and R.M. Owens. Minimizing power consumption of static CMOS circuits by transistor sizing and input reordering. In *Proc. of the International Conference on VLSI Design*, pages 294–298, January 1995.

- [BM82] R. Brayton and C. McMullen. The decomposition and factorization of boolean expressions. In *Proc. International Symposium on Circuits and Systems*, pages 49–54, 1982.
- [BM95] L. Benini and G. De Micheli. Transformation and synthesis of FSMs for low-power gated-clock implementation. In *Int. Symposium on Low Power Design*, pages 21–26, April 1995.
- [BOI95] M. Borah, R.M. Owens, and M.J. Irwin. High-throughput and low-power DSP using clocked-CMOS circuitry. In *Int. Symposium on Low Power Design*, pages 139–144, April 1995.
- [BRSVW87] 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*, 6(6):1062–1081, November 1987.
- [Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [BS95] C.W. Brown and B.J. Shepherd. *Graphics File Formats: reference and guide*. Prentice-Hall, 1995.
- [BSdM94] L. Benini, P. Siegel, and G. de Micheli. Saving power by synthesizing gated clocks for sequential circuits. *IEEE Design & Test of Computers*, pages 32–41, winter 1994.
- [BW73] C.R. Baugh and B.A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22(12):1045–1047, December 1973.
- [CB95] A.P. Chandrakasan and R.W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.
- [CC93] B.S. Carlson and C.Y.R. Chen. Performance enhancement of CMOS VLSI circuits by transistor reordering. In *Proceedings of the 30th Design Automation Conference*, pages 361–366, 1993.
- [CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
- [CP95] J-M. Chang and M. Pedram. Register allocation and binding for low power. In *Proceedings of the 32nd Design Automation Conference*, pages 29–35, 1995.
- [CPM⁺95] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R.W. Brodersen. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design*, 14(1):12–31, January 1995.
- [CPRB92] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R.W. Brodersen. HYPER-LP: A system for power minimization using architectural transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 300–303, November 1992.

- [CR94] A. Chatterjee and R.K. Roy. Synthesis of low power linear DSP circuits using activity metrics. In *Proc. of the International Conference on VLSI Design*, pages 265–270, January 1994.
- [CRP94] T-L. Chou, K. Roy, and S. Prasad. Estimation of circuit activity considering signal correlations and simultaneous switching. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 300–303, 1994.
- [CS93] T.K. Callaway and E.R. Swartzlander. Estimating the power consumption of CMOS adders. In *Proc. of the Custom Integrated Circuit Conference*, pages 210–216, 1993.
- [CSB90] A. Chandrakasan, S. Sheng, and R.W. Brodersen. Design considerations for a future multimedia terminal. In *WINLAB Workshop*, October 1990.
- [CSB92a] A. Chandrakasan, S. Sheng, and R. Brodersen. Low power CMOS digital design. *IEEE Transactions on Solid-State Circuits*, 27(4):473–483, April 1992.
- [CSB92b] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power techniques for portable real-time DSP applications. *VLSI Design*, pages 203–208, 1992.
- [CSB94] A.P. Chandrakasan, M.B. Srivastava, and R.W. Brodersen. Energy efficient programmable computation. In *Proc. of the International Conference on VLSI Design*, pages 261–264, January 1994.
- [CW94] K-Y. Chao and D.F. Wong. Low power considerations in floorplan design. In *Proc. International Workshop on Low Power Design*, pages 45–50, April 1994.
- [DDN85] P. Dewilde, E. Deprettere, and R. Nouta. *Parallel and pipelined VLSI implementation of signal processing algorithms*, chapter 15, pages 257–264. VLSI and Modern Signal Processing. Prentice-Hall, Inglewood Cliffs, NJ, 1985.
- [DK95] A. Dasgupta and R. Karri. Simultaneous scheduling and binding for power minimization during microarchitectural synthesis. In *Int. Symposium on Low Power Design*, pages 69–74, April 1995.
- [DMP95] R. Marculescu D. Marculescu and M. Pedram. Information theoretic measures of energy consumption at register transfer level. In *Int. Symposium on Low Power Design*, pages 81–86, April 1995.
- [ea92] D.W. Dobberpuhl et al. A 200-MHz 64-b dual-issue CMOS microprocessor. *IEEE Journal of Solid-State Circuits*, 27(11):1555–1567, November 1992.
- [ea94a] G. Gerosa et al. A 2.2 W 80 MHz superescalar RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 29(12):1440–1454, December 1994.

- [ea94b] R. Bechade et al. A 32-b 66 MHz microprocessor. In *International Solid State Circuits Conference*, pages 208–209, February 1994.
- [ea94c] T. Indermaur et al. Evaluation of charge recovery circuits and adiabatic switching for low power CMOS design. In *Int. Symposium on Low Power Electronics*, October 1994.
- [ea95a] D. Bearden et al. A 133 MHz 64-b four-issue CMOS microprocessor. In *International Solid State Circuits Conference*, pages 174–175, February 1995.
- [ea95b] W.J. Bowhill et al. A 300 MHz 64-b quad-issue CMOS RISC microprocessor. In *International Solid State Circuits Conference*, pages 182–183, February 1995.
- [EL94] M.D. Ercegovac and T. Lang. Reducing transition counts in arithmetic circuits. In *Int. Symposium on Low Power Electronics*, pages 64–65, October 1994.
- [EL95] M.D. Ercegovac and T. Lang. Low-power accumulator (correlator). In *Int. Symposium on Low Power Electronics*, pages 30–31, October 1995.
- [FFK88] M. Fujita, M. Fujisawa, and N. Kawato. Evaluation and improvements of boolean comparison method based on binary decision diagrams. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 2–5, November 1988.
- [GBJ95] A.L. Glebov, D. Blaauw, and L.G. Jones. Transistor reordering for low power CMOS gates using an SP-BDD representation. In *Int. Symposium on Low Power Design*, pages 161–166, April 1995.
- [GDWL92] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-level synthesis: introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
- [GIG⁺94] S. Gary, P. Ippolito, G. Gerosa, C. Dietz, J. Eno, and H. Sánchez. PowerPC 603TM, a microprocessor for portable computers. *IEEE Design & Test of Computers*, pages 14–23, winter 1994.
- [GR94] D.D. Gajski and L. Ramachandran. Introduction to high-level synthesis. *IEEE Design & Test of Computers*, 2(4):44–54, winter 1994.
- [GS93] P. Groeneveld and P. Stravens. Ocean: The Sea-of-Gates design system. Technical report, Delft University of Technology, 1993.
- [Gwe93a] L. Gwennap. ARM7 cuts power, increases performance. *Microprocessor Report*, 7(15):12–13, November 1993.
- [Gwe93b] L. Gwennap. Hobbit enables personal communications. *Microprocessor Report*, 14:15–21, October 1993.
- [Hay88] J. P. Hayes. *Computer architecture and organization*. 1988.

- [HdlR94] G.D. Hachtel and M. Hermida de la Rica. Re-encoding sequential circuits to reduce power dissipation. In *Proc. International Workshop on Low Power Design*, pages 69–74, April 1994.
- [HvBPS93] Jaco Haans, Kees van Berkel, Ad Peeters, and Frits Schalijs. Asynchronous multipliers as combinational handshake circuits. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 149–163. Elsevier Science Publishers, 1993.
- [HZA94] R. Hossain, M. Zheng, and A. Albicki. Reducing power dissipation in serially connected MOSFET circuits via transistor reordering. In *Proc. of the IEEE International Conference on Computer Design*, pages 614–617, October 1994.
- [Ike95] T. Ikeda. Thinkpad low-power evolution. In *Int. Symposium on Low Power Electronics*, pages 6–7, October 1995.
- [IP94] S. Iman and M. Pedram. Multi-level network optimization for low power. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 371–377, November 1994.
- [IP95] S. Iman and M. Predram. Logic extraction and factorization for low power. In *Proceedings of the 30th Design Automation Conference*, pages 248–253, 1995.
- [JQN⁺91] B. Johnson, T. Quarles, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE3 Version 3e User's Manual*. Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley, April 1991.
- [KaKS94] H. Kojima and S. Tanaka ad K. Sasaki. Half-swing clocking scheme for 75% power saving in clocking circuitry. In *Symp. on VLSI Circuits*, pages 23–24, June 1994.
- [Kap94] B. Kapoor. Improving the accuracy of circuit activity measurement. In *Proceedings of the 31st Design Automation Conference*, pages 734–739, 1994.
- [KBL95] U. Ko, P.T. Balsara, and W. Lee. Low-power design techniques for high-performance CMOS adders. *IEEE Transactions on VLSI Systems*, 3(2):327–333, June 1995.
- [KBN95] U. Ko, P.T. Balsara, and A.K. Nanda. Energy optimization of multi-level processor cache architectures. In *Int. Symposium on Low Power Design*, pages 45–49, April 1995.
- [Keu94] K. Keutzer. The impact of CAD on the design of low power digital circuits. In *Int. Symposium on Low Power Electronics*, pages 42–43, October 1994.

- [KGG95] D.J. Kinniment, J.D. Garside, and B. Gao. A comparison of power consumption in some CMOS adder circuit. In *Proc. of the Int. Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)*, pages 106–118, 1995.
- [KKRV95] N. Kumar, S. Katkooi, L. Rader, and R. Vemuri. Profile-driven behavioral synthesis for low-power VLSI systems. *IEEE Design & Test of Computers*, pages 70–84, fall 1995.
- [Kor93] I. Koren. *Computer Arithmetic Algorithms*. Prentice-Hall, 1993.
- [Kun84] S.Y. Kung. On supercomputing with systolic/wavefront array processor. In *Proc. of the IEEE*, pages 867–884, July 1984.
- [Lan94] P. Landman. *Low-power architectural design methodologies*. PhD thesis, College of Engineering, University of California at Berkeley, August 1994. UCB/ERL M94/62.
- [Lim90] J.S. Lim. *Two-Dimensional Signal and Image Processing*. Signal Processing Series. Prentice-Hall, 1990.
- [LK84] C. Lin and S. Kwatra. An adaptive algorithm for motion compensated colour image coding. *IEEE Globecom*, 1984.
- [LKB94] W. Lee, U. Ko, and P.T. Balsara. A comparative study on CMOS digital circuit families for low-power applications. In *Proc. International Workshop on Low Power Design*, pages 129–132, April 1994.
- [LM93] B. Lin and H. De Man. Low-power driven technology mapping under timing constraints. In *Proc. of the IEEE International Conference on Computer Design*, 1993.
- [LR94a] P.E. Landman and J.M. Rabaey. Black-box capacitance models for architectural power analysis. In *Proc. International Workshop on Low Power Design*, pages 165–170, April 1994.
- [LR94b] D.B. Lidsky and J.M. Rabaey. Low-power design of memory intensive functions. In *Int. Symposium on Low Power Electronics*, pages 16–17, October 1994.
- [LR95] P.E. Landman and J.M. Rabaey. Activity-sensitive architectural power analysis for the control path. In *Int. Symposium on Low Power Design*, pages 93–98, April 1995.
- [LS83] C.E. Leiserson and J.B. Saxe. Optimizing synchronous circuitry by retiming. In *Third Caltech Conf. on VLSI*, 1983.
- [LS94] C. Lemmonds and S.S.M. Shetti. A low power 16 by 16 multiplier using transition reduction circuitry. In *Proc. International Workshop on Low Power Design*, pages 139–142, April 1994.
- [LSI94] LSI LOGIC. *LCA500K Preliminary Design Manual*, November 1994.

- [LvMJ95] J. Leijten, J. van Meerbergen, and J. Jess. Analysis and reduction of glitches in synchronous networks. In *Proc. European Conference on Design Automation (EDAC)*, March 1995.
- [Mar95] R.S. Martin. *Optimizing power consumption, area and delay in behavioral synthesis*. PhD thesis, Department of Electronics, Faculty of Engineering, Carleton University, March 1995.
- [Mat96] A. Matsuzawa. Low-power portable design. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 1996. Invited lecture.
- [MC95a] E. Musoll and J. Cortadella. High-level synthesis techniques for reducing the activity of functional units. In *Int. Symposium on Low Power Design*, pages 99–104, April 1995.
- [MC95b] E. Musoll and J. Cortadella. Low-power array multipliers with transition-retaining barriers. In *Proc. of the Int. Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)*, pages 227–238, October 1995.
- [MC95c] E. Musoll and J. Cortadella. Scheduling and resource binding for low power. In *Int. Symposium on System Synthesis*, pages 104–109, September 1995.
- [MC96] E. Musoll and J. Cortadella. Optimizing CMOS circuits for low power using transistor-reordering. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 219–223, March 1996.
- [MDG93] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. In *Proc. of the IEEE International Conference on Computer Design*, pages 398–402, November 1993.
- [Mei95] J.D. Meindl. Low-power microelectronics: retrospect and prospect. *Proceedings of the IEEE*, 83(4):619–635, April 1995.
- [MIP94] MIPS. *MIPS Press release*. 1994.
- [MK95] R.S. Martin and J.P. Knight. Power-profiler: Optimizing ASICs power consumption at the behavioral level. In *Proceedings of the 32nd Design Automation Conference*, 1995.
- [MSBSV91] S. Malik, E.M. Sentovich, R.K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: optimizing sequential networks with combinational techniques. *IEEE Transactions on Computer-Aided Design*, 10(1):74–84, January 1991.
- [MT95] V.G. Moshnyaga and K. Tamary. A comparative study of switching activity reduction techniques for the design of low-power multipliers. In *Proc. International Symposium on Circuits and Systems*, pages 1560–1563, April 1995.

- [MWBSV88] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 6–10, November 1988.
- [Naj91] F.N. Najm. Transition density, a stochastic measure of activity in digital circuits. In *Proceedings of the 28th Design Automation Conference*, pages 644–649, 1991.
- [Naj95] F.N. Najm. Towards a high-level power estimation capability. In *Int. Symposium on Low Power Design*, pages 87–92, April 1995.
- [NMO94] C. Nagendra, U.K. Mehta, and R.M. Owens. A comparison of the power-delay characteristics of CMOS adders. In *Proc. International Workshop on Low Power Design*, pages 231–236, April 1994.
- [NS94] L.S. Nielsen and J. Sparsø. Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. In *Proc. International Workshop on Low Power Design*, pages 99–104, April 1994.
- [OF94] M.A. Ortega and J. Figueras. Extra power consumed in static CMOS circuits due to unnecessary logic transitions. In *IV Int. Design Automation Workshop (RUSSIAN WORKSHOP'94)*, June 1994.
- [OY94] P-W. Ong and R-H. Yan. Power-conscious software design - a framework for modeling software on hardware. In *Int. Symposium on Low Power Electronics*, pages 36–37, October 1994.
- [PC91] S.R. Powell and P.M. Chau. A model for estimating power dissipation in a class of DSP VLSI chips. *IEEE Transactions on Circuits and Systems*, 38(6):646–650, June 1991.
- [Pez71] S.D. Pezaris. A 40ns 17-bit by 17-bit array multiplier. *IEEE Transactions on Computers*, C-20(4):442–447, April 1971.
- [PM75] K.P. Parker and E.J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers*, C-24:668–670, June 1975.
- [Pow95] R.A. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, April 1995.
- [PR91] M. Potkonjak and J. Rabaey. Optimizing the resource utilization using transformations. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 88–91, November 1991.
- [PR94] S.C. Prasad and K. Roy. Circuit optimization for minimization of power consumption under delay constraint. In *Proc. International Workshop on Low Power Design*, pages 15–20, April 1994.
- [PR95] R. Panwar and D. Rennels. Reducing the frequency of tag compares for low power i-cache design. In *Int. Symposium on Low Power Design*, pages 57–62, April 1995.

- [PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.
- [Pug91] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proc. Supercomputing'91*, 1991.
- [Rep93] Special Report. The new contenders. *IEEE Spectrum*, pages 20–25, December 1993.
- [RJ94] A. Raghunathan and N.K. Jha. Behavioral synthesis for low power. In *Proc. of the IEEE International Conference on Computer Design*, pages 318–322, October 1994.
- [RJ95] A. Raghunathan and N.K. Jha. An iterative improvement algorithm for low power data path synthesis. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, 1995.
- [RP93] K. Roy and S.C. Prasad. Circuit activity based logic synthesis for low power reliable operations. *IEEE Transactions on VLSI Systems*, 1(4):503–513, December 1993.
- [RP96] J.M. Rabaey and M. Pedram, editors. *Low power design methodologies*. Kluwer Academic Publishers, 1996.
- [RY90] K.R. Rao and P. Yip. *Discrete Cosine Transform*. Academic Press, 1990.
- [SA95] M. Song and K. Asada. Design methodology for low power data compressors based on a window detector in a 54×54 bit multiplier. In *Proc. International Symposium on Circuits and Systems*, pages 1568–1571, April 1995.
- [Sam89] H. Samueli. An improved search algorithm for the design of multiplierless FIR filters with power-of-two coefficients. *IEEE Transactions on Circuits and Systems*, 36:1044–1047, July 1989.
- [SBS94] A.J. Strakatos, R.W. Brodersen, and S.R. Sanders. High-efficiency low-voltage dc-dc conversion for portable applications. In *Proc. International Workshop on Low Power Design*, pages 105–110, April 1994.
- [SFCM95] H. Samsom, F. Franssen, F. Catthoor, and H. De Man. System level verification of video and image processing specifications. In *Int. Symposium on System Synthesis*, pages 144–149, September 1995.
- [SGDK92] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, 1992.
- [SK94] L. “J”. Svenson and J.G. Koller. Adiabatic charging without inductors. In *Proc. International Workshop on Low Power Design*, pages 159–164, April 1994.

- [SLB95] T. Sakuta, W. Lee, and P.T. Balsara. Delay balanced multipliers for low power/low voltage DSP core. In *Int. Symposium on Low Power Electronics*, pages 36–37, October 1995.
- [SLW95] W.Z. Shen, J.Y. Lin, and F.W. Wang. Transistor reordering rules for power reduction in CMOS gates. In *ASP-DAC*, July 1995.
- [Sma94] C. Small. Shrinking devices put the squeeze on system packaging. *EDN*, 39(4):41–46, February 1994.
- [SNDD94] G.G. Shahidi, T.H. Ning, R.H. Dennard, and B. Davari. SOI for low-voltage and high-speed CMOS. In *Int. Conf. on Solid-State Devices and Materials*, pages 265–267, 1994.
- [SNK73] Y. Susuki, K. Nogami, and M. Kakumu. Clocked CMOS calculator circuitry. *IEEE Journal of Solid-State Circuits*, SC-8(6):462–469, December 1973.
- [SNNS93] J. Sparsø, C. D. Nielsen, L. S. Nielsen, and J. Staunstrup. Design of self-timed multipliers: A comparison. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 165–179. Elsevier Science Publishers, 1993.
- [SR95] G.E. Sobelman and D.L. Raatz. Low-power multiplier design using delayed evaluation. In *Proc. International Symposium on Circuits and Systems*, pages 1564–1567, April 1995.
- [SSL+92] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report Memorandum No. UCB/ERL M92/41, University of California, Berkeley, 1992.
- [STD94] C-L. Su, C-Y. Tsui, and A.M. Despain. Saving power in the control path of embedded processors. *IEEE Design & Test of Computers*, pages 24–30, winter 1994.
- [TA94] C.H. Tan and J. Allen. Minimization of power in VLSI circuits using transistor sizing, input ordering, and statistical power estimation. In *Proc. International Workshop on Low Power Design*, pages 75–80, April 1994.
- [TAM93] V. Tiwari, P. Ashar, and S. Malik. Technology mapping for low power. In *Proceedings of the 30th Design Automation Conference*, pages 74–79, 1993.
- [Tiw94] G. Tiwary. Below the half-micron mark. *IEEE Spectrum*, pages 84–87, November 1994.
- [TJL87] J.R. Treichler, C.R. Johnson, Jr., and M.G. Larimore. *Theory and Design of Adaptive Filters*. New York: John Wiley & Sons, 1987.

- [TMA95] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: pushing power management to logic synthesis/design. In *Int. Symposium on Low Power Design*, pages 221–226, April 1995.
- [TMW94] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.
- [TPCD94] C-Y. Tsui, M. Pedram, C-A. Chen, and A.M. Despain. Low power state assignment targeting two- and multi-level logic implementations. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, November 1994.
- [TPD93] C.Y. Tsui, M. Pedram, and A.M. Despain. Technology decomposition and mapping targeting low power dissipation. In *Proceedings of the 30th Design Automation Conference*, pages 68–73, June 1993.
- [TPD94] C.Y. Tsui, M. Pedram, and A.M. Despain. Power efficient technology decomposition and mapping under extended power consumption model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1110–1122, September 1994.
- [TRDH89] I. Turkik, A. Reisman, R. Darveaux, and L-T. Hwang. Multichip packaging for supercomputers. In *Proc. of the NEPCON West'89*, pages 280–287, March 1989.
- [vBBK⁺94] K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalijs, and A. Peeters. Asynchronous circuits for low power: a DCC error corrector. *IEEE Design & Test of Computers*, pages 22–32, summer 1994.
- [vBN93] C.H. van Berkel and C. Niessen. An apparatus featuring a feedback signal for controlling a powering voltage for asynchronous electronic circuitry therein. *European Patent Application 92203949.0*, June 1993.
- [Vee84] H.J.M. Veendrick. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, pages 468–473, August 1984.
- [vG89] A.J. van Gerenden. SLS: An efficient switch-level timing simulator using min-max voltage waveforms. In *Proc. VLSI 89 Conf.*, pages 79–88, August 1989.
- [VP93] H. Vaishnav and M. Pedram. PCUBE: A performance driven placement algorithm for low power design. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 72–77, 1993.
- [War95] C.A. Warwick. Trends and limits in the “talk time” of personal communicators. *Proceedings of the IEEE*, 83(4):681–686, April 1995.
- [WCF⁺94] S. Wuytack, F. Catthoor, F. Franseen, L. Nachtergaele, and H. De Man. Global communications and memory optimizing transformations for low power. In *Proc. International Workshop on Low Power Design*, pages 203–208, April 1994.

- [WE93] N. Weste and Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 2nd edition, 1993.
- [Wol95] A. Wolfe. A case study in low-power system level design. In *Proc. of the IEEE International Conference on Computer Design*, October 1995.
- [Yan91] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical report, Microelectronics Center of North Carolina, Research Triangle Park, NC, January 1991.
- [YSS⁺94] N.K. Yeung, Y-H. Sutu, T.Y-F. Su, E.T. Pak, C-C. Chao, S. Akki, D.D. Yau, and R. Lodenquai. The design of a 55SPECint92 RISC processor under 2w. In *International Solid State Circuits Conference*, pages 206–207, February 1994.
- [YYN⁺90] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu. A 3.8-ns CMOS 16x16-b multiplier using complementary pass-transistor logic. *IEEE Journal of Solid-State Circuits*, 25(2):388–395, April 1990.
- [ZA95] M. Zheng and A. Albicki. Low power and high speed multiplication design through mixed number representations. In *Proc. of the IEEE International Conference on Computer Design*, October 1995.