

Design of Low-Power Reduction-Trees in Parallel Multipliers

by

Saeed Tahmasbi Oskuii

Dissertation submitted to the
Norwegian University of Science and Technology
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy



Department of Electronics and Telecommunications
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

Multiplications occur frequently in digital signal processing systems, communication systems, and other application specific integrated circuits. Multipliers, being relatively complex units, are deciding factors to the overall speed, area, and power consumption of digital computers. The diversity of application areas for multipliers and the ubiquity of multiplication in digital systems exhibit a variety of requirements for speed, area, power consumption, and other specifications. Traditionally, speed, area, and hardware resources have been the major design factors and concerns in digital design. However, the design paradigm shift over the past decade has entered dynamic power and static power into play as well.

In many situations, the overall performance of a system is decided by the speed of its multiplier. In this thesis, parallel multipliers are addressed because of their speed superiority. Parallel multipliers are combinational circuits and can be subject to any standard combinational logic optimization. However, the complex structure of the multipliers imposes a number of difficulties for the electronic design automation (EDA) tools, as they simply cannot consider the multipliers as a whole; i.e., EDA tools have to limit the optimizations to a small portion of the circuit and perform logic optimizations. On the other hand, multipliers are arithmetic circuits and considering arithmetic relations in the structure of multipliers can be extremely useful and can result in better optimization results. The different structures obtained using the different arithmetically equivalent solutions, have the same functionality but exhibit different temporal and physical behavior. The arithmetic equivalencies are used earlier mainly to optimize for area, speed and hardware resources.

In this thesis a design methodology is proposed for reducing dynamic and static power dissipation in parallel multiplier partial product reduction tree. Basically, using the information about the input pattern that is going to be applied to the multiplier (such as static probabilities and spatiotemporal correlations), the reduction tree is optimized. The optimization is obtained by selecting the power

efficient configurations by searching among the permutations of partial products for each reduction stage. Probabilistic power estimation methods are introduced for leakage and dynamic power estimations. These estimations are used to lead the optimizers to minimum power consumption. Optimization methods, utilizing the arithmetic equivalencies in the partial product reduction trees, are proposed in order to reduce the dynamic power, static power, or total power which is a combination of dynamic and static power. The energy saving is achieved without any noticeable area or speed overhead compared to random reduction trees. The optimization algorithms are extended to include spatiotemporal correlations between primary inputs. As another extension to the optimization algorithms, the cost function is considered as a weighted sum of dynamic power and static power. This can be extended further to contain speed merits and interconnection power. Through a number of experiments the effectiveness of the optimization methods are shown. The average number of transitions obtained from simulation is reduced significantly (up to 35% in some cases) using the proposed optimizations.

The proposed methods are in general applicable on arbitrary multi-operand adder trees. As an example, the optimization is applied to the summation tree of a class of elementary function generators which is implemented using summation of weighted bit-products. Accurate transistor-level power estimations show up to 25% reduction in dynamic power compared to the original designs.

Power estimation is an important step of the optimization algorithm. A probabilistic gate-level power estimator is developed which uses a novel set of simple waveforms as its kernel. The transition density of each circuit node is estimated. This power estimator allows to utilize a global glitch filtering technique that can model the removal of glitches in more detail. It produces error free estimates for tree structured circuits. For circuits with reconvergent fanout, experimental results using the ISCAS85 benchmarks show that this method generally provides significantly better estimates of the transition density compared to previous techniques.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (Ph.D.) at the Department of Electronics and Telecommunications, Norwegian University of Science and Technology (NTNU). My main supervisor has been Associate Professor Per Gunnar Kjeldsberg, while my co-supervisors have been Associate Professor Lars Lundheim and Professor Geir Øien, all of whom are with the Department of Electronics and Telecommunications at NTNU. I spent about four months at Electronics Systems group at Linköping University for research cooperation under the supervision of Assistant Professor Oscar Gustafsson. The last six months I have worked at the University of Oslo.

The studies have been carried out in the period from January 2004 to March 2008. The work includes the equivalent of half a year of full-time course studies and approximately one year of teaching duties. The latter included being a teaching assistant in several graduate courses, as well as being an advisor for one student working on his master thesis.

The work included in this thesis has been funded by the the Department of Electronics and Telecommunications, NTNU.

Acknowledgments

I have a theory that when I am happy, time passes more quickly. It has been over four years now from the day I started as a Ph.D. student at Norwegian University of Science and Technology and these four years have passed extremely quick. Indeed, these years have been nothing but happiness and excitement and I owe this to many people who have helped me in this part of my life and in the course of my research throughout these years.

First of all, I would like to thank my supervisor, Associate Professor Per Gunnar Kjeldsberg. I could not have imagined having a better advisor for my Ph.D., and without his constant encouragement, perceptiveness and advice over these years I could ever come this far. I am very grateful for his encouragement, common-sense and being available to help me whenever I needed it. I am also very grateful to my co-supervisors Associate Professor Lars Lundheim and Professor Geir Øien for the valuable discussions and insightful advise.

I am thankful to Assistant Professor Oscar Gustafsson who was hosting me during my stays at Linköping University. I am very grateful for his fruitful discussions, valuable comments, and feedback. My special thanks also go to Kenny Johansson for his contributions to this thesis. I am amazed by his punctuality and his ability to find small errors. I am also grateful to Professor Jim Tørresen at ROBIN group, Department of Informatics, University of Oslo, for letting me have an office at his group.

I would also like to thank my colleagues and friends at the Department of Electronics and Telecommunications, NTNU and especially those of the Circuits and Systems and Signal Processing groups. There are many people I should thank for making these years so memorable, and by naming a few of them I will certainly run the risk of forgetting others. Therefore, I simply want to thank all the people who have made these years so enjoyable.

I would like to express my gratitude to my treasured family for their support and unconditional love throughout all these years. And last but most importantly, I am greatly indebted to my sweet Jeiran, my wife, for all support and encouragement. She has been the greatest source of love and inspiration.

Contents

Abstract	i
Preface	iii
Acknowledgments	iv
List of Abbreviations	xv
1 Introduction	1
1.1 Sources of Power Consumption	2
1.1.1 Static Power Dissipation	2
1.1.2 Dynamic Power Dissipation	3
1.1.2.1 Short-Circuit Power Dissipation	3
1.1.2.2 Switching Power Dissipation	4
1.1.3 Power Consumption in Multipliers	5
1.2 Probabilistic Treatment of Multipliers	6
1.3 Outline of the Thesis	9
1.4 Main Contributions	11
2 Multiplication Schemes	13
2.1 Classification of Multipliers	16
2.1.1 Sequential Multipliers	16
2.1.2 Parallel Multipliers	16
2.1.3 Array Multipliers	17
2.2 Parallel Multipliers	17
2.2.1 Partial Product Generation	17
2.2.1.1 Higher Radix Multipliers	20
2.2.1.2 Recoding Techniques	21

2.2.2	Partial Product Accumulation	23
2.2.2.1	Carry-Save Adders	25
2.2.2.2	Wallace Reduction Tree	27
2.2.2.3	Dadda Reduction Tree	31
2.2.2.4	Modified Dadda-Wallace Reduction Tree	32
2.2.2.5	Generalized Parallel Counters	34
2.2.2.6	Parallel Multipliers with Redundant Full-Adders	34
2.2.3	The Final Carry Propagate Adder	37
2.3	Flexibility in the Reduction Tree	39
2.3.1	Optimizing for Area, Delay and Hardware Resources	39
2.3.2	Optimizing for Power	40
2.4	Choice of the Multiplier Structure	41
3	Power Estimation in Combinational Circuits	43
3.1	Dynamic Power Estimation	43
3.1.1	Power Estimation using Simple Waveform Set	44
3.1.1.1	Simple Waveform Set	46
3.1.1.2	Computation of Simple Waveform Sets	48
3.1.1.3	Dealing with interdependencies	50
3.1.1.4	Glitch filtering	52
3.1.1.5	Experiments	55
3.1.2	High-Level Power Estimation	59
3.2	Static Power Estimation	60
4	Transition-Activity Aware Design of Reduction-Trees	63
4.1	Construction of the Reduction Tree	63
4.2	Method 1: Optimization of Complete Reduction Tree	66
4.2.1	Power Estimation	71
4.2.2	Notation of the multipliers	72
4.2.3	Experiments	73
4.3	Method 2: Progressive Design of Reduction Tree	78
4.3.1	Progressive PPRT Design and the SWS Power Estimator	81
4.3.2	Experiments	83
4.3.3	Runtime and Complexity	90
5	PPRT Optimization in Presence of Highly Correlated Inputs	95
5.1	Spatiotemporal Correlation of Inputs	96
5.1.1	Modeling Spatial Correlations	97

5.1.2	Modeling Temporal Correlations	98
5.2	Design Methodology	101
5.3	Experiments	103
5.3.1	Temporal correlations of input words	103
5.3.2	Spatial correlations of input words	105
5.3.3	MAC-based FIR filter	109
6	Reducing the Static Power for the PPRT	113
6.1	Progressive Reduction-Tree Design	114
6.1.1	Experiments	114
6.2	Reducing Total Power for the PPRT	118
6.2.1	Experiments	119
7	Optimization of Multipliers Operating with Variable Word-Length	123
7.1	Variable Word-Length	125
7.2	Experiments	128
7.2.1	General-Purpose Multiplier	129
7.2.2	FFT Processor with Variable Word-Length	132
8	Function Generation using a Weighted Sum of Bit-Products	137
8.1	Elementary Function Generation	139
8.1.1	Approximation using a Weighted Sum of Bit-Products	140
8.1.2	Architecture	142
8.2	Optimized Summation-Tree Generation	143
8.3	Experiments	145
9	Conclusions	151
9.1	Low-Transition Reduction-Tree Generation	151
9.2	Low-Leakage Reduction-Tree Generation	152
9.3	Generalized Multi-Operand Adders	152
9.4	Probabilistic Gate-Level Power Estimator	153
9.5	Directions for Future Work	153
9.5.1	Power Estimation with Realistic Delay Models	153
9.5.2	Including Speed and Interconnect Power	154
9.5.3	Different PP Generation and Reduction Schemes	154
9.5.4	Other Search Algorithms	154
9.5.5	Pipelining	154
9.5.6	Optimization of Synthesized Designs	155

Appendix	155
A Computation of static probabilities	157
A.1 Pairwise Correlation Coefficients	158
A.1.1 An Example	163
B Examples of Computing SWSs	165
B.1 Tree-Structured example	166
B.2 A 2-to-1 MUX Example	173
Bibliography	192

List of Figures

1.1	Dynamic and static power in Watt for microprocessor chips	3
1.2	Computation of static probabilities	8
2.1	Shift-and-Add multiplication scheme	14
2.2	Block diagram for a shift-and-add multiplier	15
2.3	The dot-diagram representation of an 8×8 -bit multiplier	15
2.4	Signed multiplication	19
2.5	8×8 -bit Radix-4 multiplication in dot-diagram and its structure	20
2.6	Overlapping multiple bit scanning in radix-4 MBR	22
2.7	Accumulating PPs using two-operand adders	23
2.8	Carry propagation and carry-free operation	24
2.9	A half-adder(a) and a full-adder(b)	26
2.10	A typical gate-level implementation of full-adder and half-adder	27
2.11	The relationship between RCA and CSA	28
2.12	Wallace's scheme for a 12×12 -bit unsigned multiplier	30
2.13	Using Dadda's strategy for a 12×12 -bit multiplier	33
2.14	A reduced-area 12×12 -bit unsigned multiplier [13]	35
2.15	A $(10; 4)$ counter and its possible implementation	36
2.16	Examples of generalized parallel counters	36
2.17	The arrival time profile for output bit-vectors of the dadda PPRT	38
3.1	The target circuit architecture for power estimator	45
3.2	Examples of simple and non-simple waveforms	47
3.3	A 2-to-1 MUX example	48
3.4	Generation and decomposition of non-simple waveforms	49
3.5	Algorithm I for computing SWSs	50
3.6	Algorithm II for computing SWSs with glitch filtering	53
3.7	An example of glitch filtering	54

3.8	Propagation delays in Full-adders and Half-adders	55
3.9	An example tree-structured circuit	56
3.10	Average node error, average number of waveforms per node and runtime versus p_{th} for C1355	58
3.11	A NAND gate with possible input values	62
4.1	The multiplier generation algorithm	64
4.2	The PPM, FA and HA matrices for a 12×12 -bit unsigned multiplier	65
4.3	The multiplier optimization algorithm - Method 1	67
4.4	Sorting PPs based on their estimated transition densities	70
4.5	Worst-case multiplier algorithm - Method 1	71
4.6	Log-normal signal distribution	77
4.7	One-probabilities of input bits with log-normal distribution	77
4.8	The multiplier optimization algorithm - Method 2	79
4.9	Progressive reduction-tree design using SWS power estimator	82
4.10	The worst-case multiplier generation algorithm	84
4.11	Log-normal signal distribution	89
4.12	One-probabilities of input bits with log-normal distribution	89
4.13	Runtime versus number of iteration in SA	92
4.14	Average number of transitions versus number of iterations in SA	92
4.15	Cooling factor	93
5.1	Transition activity vs. bit positions with varying ρ	99
5.2	The generalized design methodology	102
5.3	Illustration of lag-one temporal correlation ratio vector (\mathcal{R}) for $\Phi_{0.99}^{16}$	105
5.4	Illustration of pairwise correlation coefficient matrix (\mathcal{C})	106
5.5	Illustration of pairwise correlation coefficient matrix (\mathcal{C})	107
5.6	FIR filter implementation	110
5.7	Multiply accumulate architecture	110
5.8	Illustration of pairwise correlation coefficient matrix (\mathcal{C})	112
6.1	Progressive reduction-tree design using static power estimator	115
6.2	Progressive PPRT design for total power reduction	120
7.1	Correlated and uncorrelated input pattern	127
7.2	Visualization of the word-length probability density matrix	129
7.3	One-probabilities for primary input bits in example 1	132
7.4	Visualization of the correlation coefficient matrix	133
7.5	Lag-one temporal correlation ratios for primary input bits	134

8.1 Different patterns for partial products in multi-operand adders . . . 138

8.2 Architecture used for approximating elementary functions 142

8.3 Progressive reduction-tree design algorithm for function generators 144

8.4 The primary PP pattern for the (a) COSINE and (b) SINE functions 146

A.1 Calculating output one-probability of the basic logic gates 160

A.2 Calculating correlation coefficients for basic logic gates 162

A.3 An example of static probability computation 163

B.1 An example tree-structured circuit 166

B.2 A 2-to-1 MUX example 173

List of Tables

2.1	Booth's recoding algorithm	21
2.2	Radix-4 modified Booth's recoding algorithm	22
2.3	Truth-table and behavioral representation of half-adder	26
2.4	Truth-table and behavioral representation of full-adder	26
2.5	The Dadda sequence for minimal number of reduction stages	31
3.1	Power estimation error for ISCAS'85 benchmark circuits with fanout delay assignment. All errors are in percentage.	57
3.2	Runtime and computation complexity for ISCAS'85 benchmark circuits with fanout delay assignment. All errors are in percentage.	59
3.3	The normalized leakage current for the logic gates	61
4.1	Number of search alternatives in one column	69
4.2	The search alternatives of a 7-bit column	70
4.3	Estimated energy per operation for different multipliers [pJ]	74
4.4	Estimated energy per operation for optimized multipliers [pJ]	74
4.5	Estimated energy per operation for multipliers with log-normal inputs	78
4.6	Average number of transitions for different multipliers	85
4.7	Average number of transitions with realistic delay model	86
4.8	Estimations of energy per operation from Synopsys Power Compiler for $0.35\mu m$ CMOS library	86
4.9	Estimations of energy per operation from Synopsys Power Compiler for a $65nm$ CMOS library	86
4.10	Average number of transitions for different multipliers	88
4.11	Average number of transitions for different multipliers	90

5.1	Average number of transitions for different multipliers with temporally correlated inputs	104
5.2	Average number of transitions for different multipliers with spatially correlated inputs	108
5.3	Average number of transitions for MAC-based FIR filter	111
6.1	Unsigned multipliers with independent input bits	117
6.2	Signed 16×16 multipliers with correlated input bits	118
6.3	16×16 multipliers optimized for total power	121
7.1	Non-zero elements in the word-length probability density matrix (\mathcal{W}) in example 1	130
7.2	Average number of transitions for different multipliers	130
7.3	Word-length probabilities for an FFT processor with fading channel	135
7.4	Average number of transitions for different multipliers	135
8.1	Coefficient values in terms of function values	141
8.2	Function definitions	147
8.3	Average transition activity per clock cycle	149
8.4	Estimated energy per operation from NanoSim [pJ]	149
B.1	Average transition activity for nodes in Figure B.1	166
B.2	The simple waveform sets for the example tree-structured circuit .	172
B.3	Average transition activity for nodes in Figure B.2	173
B.4	The simple waveform sets for the 2-to-1 MUX example	174

List of Abbreviations

Abbreviation	Definition	Page
ASIC	Application Specific Integrated Circuit	1
CMOS	Complementary Metal-Oxide-Semiconductor	2
CORDIC	COordinate Rotation DIgital Computer	139
CPA	Carry-Propagate Adder	23
CSA	Carry-Save Adder	25
CSD	Canonical Signed Digit	22
CSNR	Channel Signal to Noise Ratio	132
CUT	Circuit Under Test	44
DBT	Dual Bit Type	98
DCT	Discrete Cosine Transform	139
DDFS	Direct Digital Frequency Synthesizer	139
DFT	Discrete Fourier Transform	139
DSP	Digital Signal Processor	1
EDA	Electronic Design Automation	1
FA	Full-Adder	25
FFT	Fast Fourier Transform	139
FIR	Finite-length Impulse Response	103
HA	Half-Adder	25
GA	Genetic Algorithm	154
LN	Log-Normal distribution	116
LSB	Least Significant Bit	37
MAC	Multiply-Accumulate	95
MBR	Modified Booth's Recoding	22
MSB	Most Significant Bit	37
MSD	Minimum Signed Digit	142

Abbreviation	Definition	Page
OBDD	Ordered Binary Decision Diagram	61
OFDM	Orthogonal Frequency Division Multiplexing	132
OPT	Optimized	72
PDF	Probability Density Function	6
PP	Partial Product	13
PPG	Partial Product Generation	17
PPM	Partial Product Matrix	14
PPRT	Partial Product Reduction Tree	27
RCA	Ripple-Carry Adder	23
RFO	Reconvergent FanOut	50
RND	Randomly generated multiplier	72
ROBDD	Reduced Ordered Binary Decision Diagram	157
ROM	Read Only Memory	140
RT	Reduction Tree	27
SA	Simulated Annealing	80
SNR	Signal to Noise Ratio	132
SWS	Simple Waveform Set	46
TDM	Three Dimensional Minimization	40
TPS	Tagged Probabilistic Simulation	43
UD	Uniform Distribution	116
UWN	Uniform White Noise	98
VHDL	VHSIC Hardware Description Language	73
VHSIC	Very High Speed Integrated Circuits	xvi
VLSI	Very Large Scale Integration	1
VMA	Vector Merge Adder	37
WC	Worst-Case	72
WSS	Wide Sense Stationary	6

Chapter 1

Introduction

Multipliers have been important since the introduction of the digital computers. Multiplication occurs frequently in Digital Signal Processing (DSP) systems, communication systems and other Application Specific Integrated Circuits (ASICs). Because of the significance of multiplication in scientific and engineering computations, this area has received much attention in the past decades which has led to a number of implementation techniques for multiplication. These are surveyed in many textbooks such as [50, 93, 105, 141, 195, 199]. The vast variety of application areas for multipliers exhibits different requirements for speed, area, power consumption and other specifications. Based on these requirements, which are imposed from the system that the multiplier will be operating in, different characteristics of the multiplier will be given different priorities. It is the designers task to choose a suitable multiplication algorithm and implementation method according to these priorities. Traditionally, the design priorities have been given to speed and area. However, the fast evolution of digital systems has caused a major paradigm shift in the past years. Now, other parameters like low-power, flexibility, testability and reliability have entered into the play. Power dissipation has become an important constraint in the design of digital systems. This is even more important for battery-powered applications where the energy budget is extremely limited. Low-power design has become a new area in VLSI technology and power-aware design is inevitable in the new Electronic Design Automation (EDA) tools.

Multipliers are generally computationally heavy circuit parts. Basically a large number of transistors with high transition activities has to be devoted to perform the multiplication. More transistors with high transition activities mean more internal capacitance, more overall switching and consequently more power dissi-

pation. Also the total leakage current is expected to be large in a multiplier because of its large active area. Multipliers are among the main contributors of area and power consumption in a DSP system and, more importantly, they are usually placed in the critical paths of such systems.

Throughout this thesis, a design methodology is proposed for reducing the dynamic and static power dissipation in parallel multipliers. It is assumed that the multiplier will be operating in real-time systems where high-speed is essential. Therefore, among the variety of implementation methods, high-speed parallel implementation methods are addressed. The proposed optimization method in this thesis, which will be elaborated in the following chapters, is an interconnection re-ordering algorithm based on the input data characteristics. It is applicable directly on all full-adder based parallel multipliers. With some changes the optimization method is applicable for majority of the reduction schemes. The optimization only modifies the interconnects between logic gates and the logic gates and the architecture remains unchanged.

1.1 Sources of Power Consumption

The total power dissipation in CMOS circuits results from a combination of dynamic and static sources:

$$P_{Total} = P_{Static} + P_{Dynamic} \quad (1.1)$$

Figure 1.1 shows a scaling scenario throughout past years for both the dynamic and the static power consumption [125]. It can be seen that the problem of static power due to the leakage has emerged in the 1990s. Both fractions of the total power consumption grow exponentially but the leakage power with a much bigger rate. From the trajectory of the evolutions in dynamic and static power consumption, it can be expected that in future the static power will not be negligible anymore.

1.1.1 Static Power Dissipation

Ideally, the static power consumption of CMOS gates should equal to zero as the PMOS and NMOS devices are never on simultaneously in steady-state operation. However, the static power consumption in CMOS circuits results from imperfect cut-off of the transistors and causes power dissipation even without any switching activity. The leakage currents are always present, flowing through the reverse

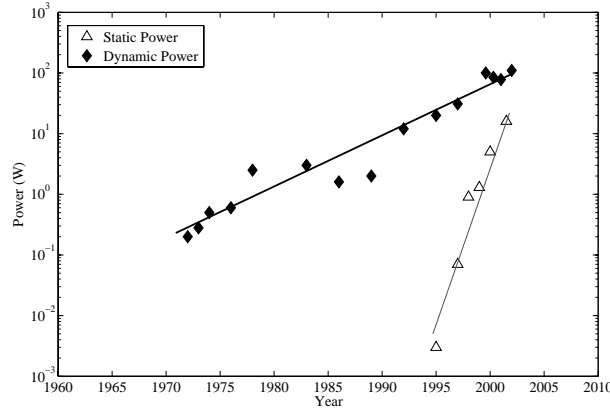


Figure 1.1: Dynamic and static power in Watt for microprocessor chips

biased diode junctions of the transistors located between the source or drain and the substrate. Another important source of leakage current is subthreshold current of the transistors. A CMOS transistor can experience a drain-source current, even when the gate-source voltage is below the threshold voltage. The subthreshold leakage current increases as the threshold voltage becomes smaller.

With an increasing number of transistors both the total capacitance as well as the total channel width which is relevant for leakage currents grows [135]. Thus an exponential growth of the amount of devices, according to Moore's law, directly results in exponentially increased leakage power dissipation. It is worth mentioning that the junction leakage currents are by thermally generated carriers. Therefore, their value increases exponentially with increasing junction temperature.

1.1.2 Dynamic Power Dissipation

The dynamic power, $P_{Dynamic}$, in Eq. 1.1 can be divided into two parts:

$$P_{Dynamic} = P_{Switching} + P_{Short-Circuit} \quad (1.2)$$

1.1.2.1 Short-Circuit Power Dissipation

$P_{Short-Circuit}$ is due to direct-path short-circuit current which arises when both NMOS and PMOS transistors are simultaneously active, conducting current directly from voltage supply to ground. The short-circuit power dissipation can be

written as:

$$P_{Short-Circuit} = V_{DD}I_{SC} \quad (1.3)$$

where V_{DD} is the supply voltage and I_{SC} is the short-circuit current that flows between V_{DD} and ground during the switching of the gates. The power dissipation due to short circuit currents shows a strong dependency on the supply and threshold voltage [138, 190, 191]. Short circuit power consumption can be kept within bounds by careful design and tuning the switching characteristics (slope engineering) of complementary logic.

1.1.2.2 Switching Power Dissipation

$P_{Switching}$ is due to charging and discharging of the load capacitances in the circuit. Each time the load capacitor at node i is charged through a transistor, part of the energy is dissipated in the transistor while the remainder of the energy drawn from power supply is stored on the load capacitance. The amount of dissipated energy per transition is equal to [151]:

$$\text{Energy per transition} = \frac{1}{2}V_{DD}^2C_i \quad (1.4)$$

where V_{DD} is the supply voltage and C_i is the load capacitance at node i . Therefore, the average power dissipation at node i due to switching will be equal to:

$$P_i^{sw} = \frac{1}{2}V_{DD}^2D_iC_i \quad (1.5)$$

where D_i is the transition density at node i . The notion of transition density which is the average number of transitions per second is proposed in [130] (DENSIM). D_i is defined as:

$$D_i = \lim_{T \rightarrow \infty} \frac{n_i(T)}{T} \quad (1.6)$$

where $n_i(T)$ is the number of transitions at node i in a time interval of length T . Average number of transitions in one clock cycle can be defined as:

$$\bar{n}_i = \frac{D_i}{f_{clk}} \quad (1.7)$$

where f_{clk} is the clock frequency. The average switching power consumed in a combinational circuit is then given by:

$$P_{total}^{sw} = \sum_{i=1}^N P_i^{sw} = \frac{1}{2}V_{DD}^2 \sum_{i=1}^N C_i D_i = \frac{1}{2}f_{clk}V_{DD}^2 \sum_{i=1}^N C_i \bar{n}_i \quad (1.8)$$

where N is the total number of nodes, C_i is the total load capacitance at node i , D_i is the transition density for node i and \bar{n}_i is the average number of transitions at node i per clock cycle.

1.1.3 Power Consumption in Multipliers

As will be discussed in Chapter 2, multipliers can in general be implemented as sequential and combinational circuits. In this thesis the focus is on the parallel multipliers which are purely combinational circuits. Therefore, the three sources for power consumption in general circuits (i.e., switching power, short-circuit power and leakage power) are present in parallel multipliers as well.

Parallel multipliers are fairly large circuit portions with high transistor density. Large active area directly leads to large leakage power dissipation in parallel multipliers. As will be elaborated in Chapter 2, parallel multipliers have three computation steps: partial product generation, partial product accumulation and vector merge addition. The accumulation of partial products is a computationally heavy task and in general this part of the multiplier dictates the overall computation delay, area and power consumption in the parallel multipliers. A number of techniques, that will be discussed in Chapter 2, has been proposed to move parts of the computation burden from the partial product accumulation step to the partial product generation step. However, even after applying such techniques, the partial product accumulation step is the dominating portion for delay, area, and power. Comprehensive discussions about the power consumption in different multiplier structures are given in [24, 25, 118]

Imbalanced signal paths within the structure of the parallel multipliers lead to high density of spurious transitions. The steady-state transitions, also referred to as functional activities, are the transitions that are necessary to perform the computation task. On the other hand, spurious (hazardous) transitions or glitches dissipate power without producing any useful computations. The ratio of the hazardous component to the total power depends strongly on the logic depth and the signal arrival time balance. In a 32-bit multiplier, the power dissipation due to glitches is about three times higher than that due to functional activities [143].

The glitch pattern is particularly dependent on the gate delays. Indeed, two identical circuits at gate-level with different gate delays may have a totally different glitch pattern and power consumption. Current power estimation techniques often handle both zero delay (no glitch) and real delay models. In the first model, it is assumed that all changes at the circuit inputs propagate through the internal gates of the circuits instantaneously. The latter model assigns each gate in the

circuit a finite delay and can thus account for the hazards in the circuit. A real-delay model significantly increases the computational requirements of the power estimation techniques while improving the accuracy of the estimates. In the context of parallel multipliers, taking the glitches into account is extremely important, because glitches consume a considerable amount of power. Consequently, considering gate delays is essential in optimization of parallel multipliers.

1.2 Probabilistic Treatment of Multipliers

When the behavior of a digital system becomes so complicated that it cannot be described deterministically, probabilistic treatment of the system is useful to quantify the behavior. In many applications, the input signals to a multiplier can be modeled as discrete time random processes. The discussion here is restricted to a short description of random processes. A more specific definition of random processes is available in probability textbooks such as [7, 122].

A discrete random process X at any specific time n is a random variable, $X[n]$. The first order distribution function of $X[n]$ is:

$$F_{X[n]}(\mathcal{X}; n) = p\{X[n] \leq \mathcal{X}\} \quad (1.9)$$

where \mathcal{X} is a real number ($\mathcal{X} \in \mathbb{R}$). The corresponding probability density function (PDF) is:

$$f_{X[n]}(\mathcal{X}; n) = \frac{\partial F_{X[n]}(\mathcal{X}; n)}{\partial \mathcal{X}} \quad (1.10)$$

The second order distribution function is:

$$F_{X[n_1]X[n_2]}(\mathcal{X}_1, \mathcal{X}_2; n_1, n_2) = p\{X[n_1] \leq \mathcal{X}_1, X[n_2] \leq \mathcal{X}_2\} \quad (1.11)$$

Similarly, the second order PDF is:

$$f_{X[n_1]X[n_2]}(\mathcal{X}_1, \mathcal{X}_2; n_1, n_2) = \frac{\partial^2 F_{X[n_1]X[n_2]}(\mathcal{X}_1, \mathcal{X}_2; n_1, n_2)}{\partial \mathcal{X}_1 \partial \mathcal{X}_2} \quad (1.12)$$

Note that two random variables $X[n_1]$ and $X[n_2]$ can be correlated to each other.

Throughout this thesis, the input data words applied to the multiplier are denoted by random processes A and B . Furthermore, it is assumed that A and B are ergodic. Ergodicity is defined for wide sense stationary (WSS) random processes. A random process is WSS if the ensemble average and the autocorrelation

function are invariant to a time shift. The ensemble average of X is:

$$E[X[n]] = \int_{-\infty}^{+\infty} \mathcal{X} f_X(\mathcal{X}; n) d\mathcal{X} \quad (1.13)$$

The auto correlation is defined as:

$$R_{XX}(n_1, n_2) = E[X[n_1]X[n_2]] = \iint_{-\infty}^{+\infty} \mathcal{X}_1 \mathcal{X}_2 f_{X_1, X_2}(\mathcal{X}_1, \mathcal{X}_2; n_1, n_2) d\mathcal{X}_1 d\mathcal{X}_2 \quad (1.14)$$

Therefore a process is WSS if $E[X[n]]$ and $R_{XX}(n, n + \eta)$ are independent of n . $E[X[n]]$ is constant for all values of n , and $R_{XX}(n, n + \eta)$ can be denoted as $R_{XX}(\eta)$.

A WSS random process is ergodic if the time average of any realization of the random process is equal to the ensemble average of the variable. The time average of X is defined as:

$$\bar{X} = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N X[n] \quad (1.15)$$

A WSS random process X is ergodic in mean if $\bar{X} = E[X]$. It is ergodic in autocorrelation if

$$R_{XX}(\eta) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N X[n]X[n + \eta] \quad (1.16)$$

The inputs of $M \times N$ -bit multiplier, $A[n]$ and $B[n]$, are quantized and represented with M and N bits respectively. The i :th and j :th input bit in the first and second operands are denoted as $a_i[n]$ and $b_j[n]$, respectively ($a_{N-1}[n]$ and $b_{M-1}[n]$ are the most significant bits). The values of A and B must be limited to the maximum and minimum numbers that can be represented using M and N bits, respectively. The one-probabilities (and zero-probabilities) of individual bits in the data words can be computed by integrating the PDFs for the intervals where the bits are 1 (or 0). The one-probability for a bit is referred to as the static probabilities of that bit as well. For example let us assume a real random variable X with values in the interval $[0, 2^N)$. X will be uniformly digitized (truncated) using an unsigned N -bit word. Figure 1.2(a) depicts an example distribution function for the real valued random variable X . Figure 1.2(b-d) illustrate the parts of the distribution function where the N :th, $(N - 1)$:th and $(N - 2)$:th bits are one,

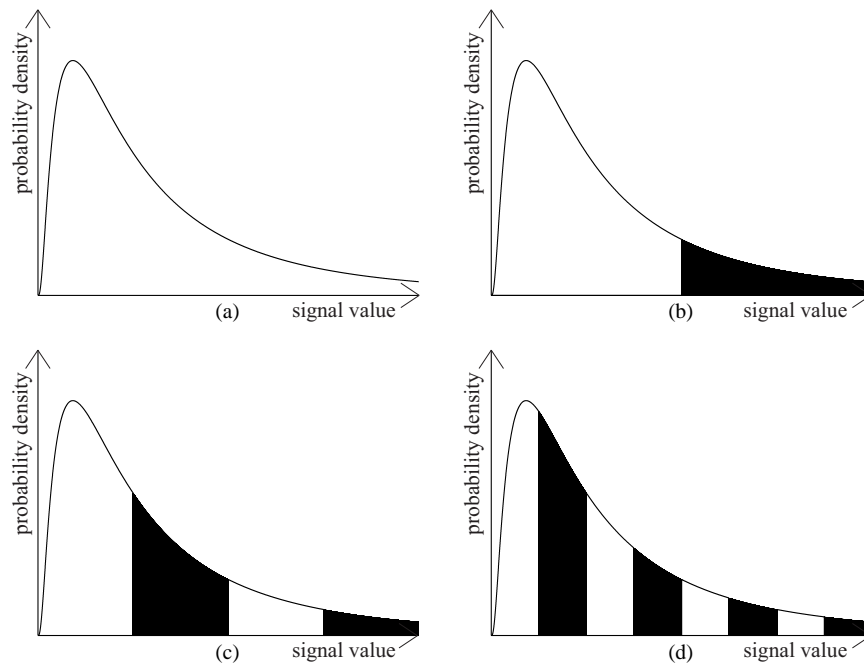


Figure 1.2: Computation of static probabilities
 (a) The probability density function at word-level
 (b) The interval resulting a 1 at the MSB
 (c) The intervals resulting a 1 at the $(N - 1)$:th bit
 (d) The intervals resulting a 1 at the $(N - 2)$:th bit

respectively. By integrating over these intervals the one-probabilities can be computed. The one-probability for the MSB can be written as:

$$p_{x_{N-1}} = \int_{2^{N-1}}^{2^N} f_{X[n]}(\mathcal{X}; n) d\mathcal{X} \quad (1.17)$$

The one-probability for i :th bit is:

$$p_{x_{i-1}} = \sum_{r=1}^{2^{N-i}} \left(\int_{(2r-1)2^{i-1}}^{r2^i} f_{X[n]}(\mathcal{X}; n) d\mathcal{X} \right) \quad (1.18)$$

For many applications the distribution of input words are unknown. However, high-level simulations of the system can provide realistic input streams for the multiplier inputs. Let us assume that a realistic input stream with N_S samples is available for both input operands. Using this input stream the static probabilities of the input bits can be approximated as:

$$p_{x_i} = \lim_{N_S \rightarrow +\infty} \frac{1}{N_S} \sum_{n=0}^{N_S-1} x_i[n] \quad (1.19)$$

1.3 Outline of the Thesis

Chapter 1 has presented a brief introduction to the concepts and importance of low-power design. It has also discussed the sources of power consumption in the multipliers. The probabilistic treatment of multipliers introduced in this chapter will be a key consideration for the subsequent chapters.

Chapter 2 presents the relevant background information about multipliers in the following order:

- Classification of multipliers into sequential, parallel and array multipliers
- The purpose of basic steps in parallel multipliers: partial product generation, partial product accumulation and the final carry propagate adders
- Prevalent methods for designing the partial product generation step, the partial product accumulation step, and the final carry propagate adder in parallel multipliers
- The arithmetic flexibilities in the partial product reduction-tree structure

- The multiplier structure that will be used in the experiments in the subsequent chapters

Chapter 3 presents the power estimation methods used in this thesis. A probabilistic gate-level power estimation technique presented in Section 3.1.1 and a high-level power macro model based estimator presented in Section 3.1.2 will be utilized as dynamic power estimators in the optimization algorithms in Chapters 4 through 8. A leakage power estimator presented in Section 3.2 will be utilized as the static power estimator in Chapter 6.

Chapter 4 presents two optimization algorithms for designing low-power partial product reduction-trees. The first method optimizes the complete reduction-tree using the power macro model based estimator presented in Section 3.1.2. The second method designs the reduction-tree progressively using the power estimator presented in Section 3.1.1. Each method is evaluated using a number of experiments.

Chapter 5 extends the progressive partial product reduction tree design algorithm in Chapter 4 to include the potential correlations between the primary inputs. The correlations are classified into two sources: temporal correlation and spatial correlations. Methods for including the temporal and spatial correlations in the power estimator are presented. This is followed by experiments for multipliers with highly correlated inputs.

Chapter 6 introduces a change in the progressive partial product reduction tree design algorithm in Chapter 4 to minimize the static power instead of the dynamic power. The dynamic power estimator is replaced by the static power estimator presented in Chapter 6. Additionally, Chapter 6 presents a method to optimize the reduction-tree for static power and dynamic power simultaneously. In this method a weighted sum of static power and dynamic power will be used in the optimization algorithm as the cost function.

Chapter 7 extends the optimization algorithms, presented in Chapters 4 and 5, on multipliers with variable word-length. Experiments on two scenarios, where the multiplier may operate with variable word-length, are presented in this Chapter.

Chapter 8 presents a different application area for the introduced optimization algorithms in Chapters 4 and 5. While Chapters 4 through 7 focus on the parallel multipliers, Chapter 8 extends the power reduction technique to a special class of elementary function generators. This class of elementary function generators is built using a summation-tree which can be optimized in a similar way as the reduction-trees in the parallel multipliers.

Chapter 9 presents the conclusions of this thesis, and recommendations for future work. This is followed by appendices and the bibliography.

1.4 Main Contributions

This thesis has led to several contributions, with the principal ones listed here below:

- Power-optimized partial product reduction tree design algorithm using power estimation based on power macro model and suggestions for limiting the search space
- Power-aware reduction tree design methodology using the progressive reduction tree design algorithm
- Extending the progressive reduction tree design algorithm to include spatiotemporal correlations between primary inputs
- Extending the progressive reduction tree design algorithm to optimize for static power or total power which is a combination of static and dynamic power
- Generalizing the application area to arbitrary multi-operand adder trees and applying the progressive reduction tree design algorithm to elementary function generators build using weighted sum of bit-products
- Probabilistic dynamic power estimation technique for combinational logic circuits using the simple waveform set

The proposed algorithms and the automated VHDL generation describing the optimized reduction trees are implemented in C++ and MATLAB as part of the work behind this thesis.

Chapter 2

Multiplication Schemes

This chapter briefly reviews a number of widely-used high-speed multiplication techniques, focusing mainly on the parts that will be addressed in the following chapters. More elaborate discussions about the multiplication techniques are given in [45, 50, 93, 105, 141]. Moreover, the discussions of this chapter are only limited to fixed-point multipliers. In fact, the floating-point multipliers consist of a fixed-point multiplier for the significands, plus peripheral and support circuitry to deal with the exponents and special values ($0, \pm\infty$, etc.) [4, 16, 36, 62, 124]. Therefore the optimization methods discussed in the following chapters are also applicable for floating-point operators.

A fixed-point multiplication involves two basic steps: generating partial products (PPs) and accumulating the generated PPs. The diverse multiplication schemes differ in the generation and/or accumulation methods. Consequently, speed-up in the multiplication process is achieved in two ways: generating less number of PPs in the first step or accelerating their accumulation in the second step. The simplest scheme for multiplication, known as shift-and-add scheme, consists of cycles of shifting and adding with hardware or software control loops. Let us use the following notation to illustrate this scheme for an unsigned $M \times N$ -bit multiplier:

	a_{M-1}	a_{M-2}	\dots	a_1	a_0	Multiplicand
	b_{N-1}	b_{N-2}	\dots	b_1	b_0	Multiplier
	$a_{M-1}b_0$	$a_{M-2}b_0$	\dots	a_1b_0	a_0b_0	
	$a_{M-1}b_1$	$a_{M-2}b_1$	\dots	a_1b_1	a_0b_1	
	\vdots	\vdots	\vdots	\vdots	\vdots	Partial Products
	$a_{M-1}b_{N-1}$	$a_{M-2}b_{N-1}$	\dots	a_1b_{N-1}	a_0b_{N-1}	
	p_{N+M-1}	p_{N+M-2}	\dots	p_1	p_0	Product

Figure 2.1: Shift-and-Add multiplication scheme

		Binary representation
Multiplicand	$A = \sum_{i=0}^{M-1} a_i 2^i$	$(a_{M-1} a_{M-2} \dots a_1 a_0)_2$
Multiplier	$B = \sum_{j=0}^{N-1} b_j 2^j$	$(b_{N-1} b_{N-2} \dots b_1 b_0)_2$
Product ($P = A \times B$)	$P = \sum_{k=0}^{N+M-1} p_k 2^k$	$(p_{N+M-1} p_{N+M-2} \dots p_1 p_0)_2$

Figure 2.1 illustrates the generation, shifting and summing of partial products for an unsigned $M \times N$ -bit multiplier and Figure 2.2 shows the block diagram of a shift-and-add multiplier in which the PPs are implemented using a multiplexer. The PPs can also be implemented using logical AND gates. The multiplication is performed in N cycles of shifting and adding. The product is given in Eq. 2.1.

$$P = \sum_{k=0}^{N+M-1} p_k 2^k = \left(\sum_{i=0}^{M-1} a_i 2^i \right) \left(\sum_{j=0}^{N-1} b_j 2^j \right) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} a_i b_j 2^{i+j} \quad (2.1)$$

The PPs are more conveniently illustrated using a dot-diagram. A dot-diagram is useful when only the positioning and alignment of bits, rather than their values, are important. Figure 2.3(a) illustrates the PPs for an unsigned 8×8 -bit multiplier. Figure 2.3(b) is an alternative dot-diagram representation for the same multiplier. Each dot in the diagram is representing a single bit which can be a zero or one. In the dot-diagram representation, only the horizontal position of the PPs has significance. By relaxing the vertical positions of PPs, the dot-diagram can be altered as shown in Figure 2.3(b).

The partial product matrix (PPM) is another representation tool for PPs. The PPM is a boolean matrix that represents a PP with the weight j by placing a 1 or TRUE in column j . The elements in the PPM that do not correspond to a PP are

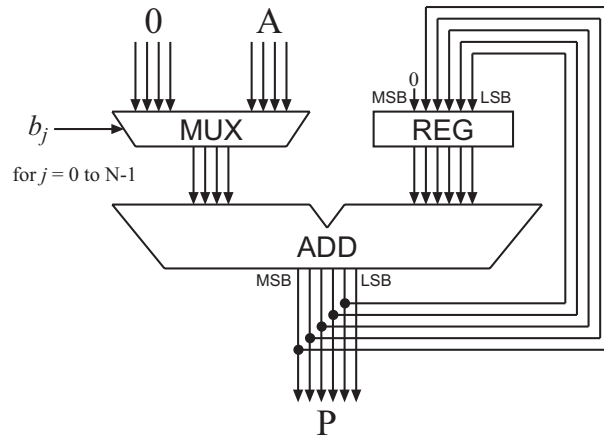
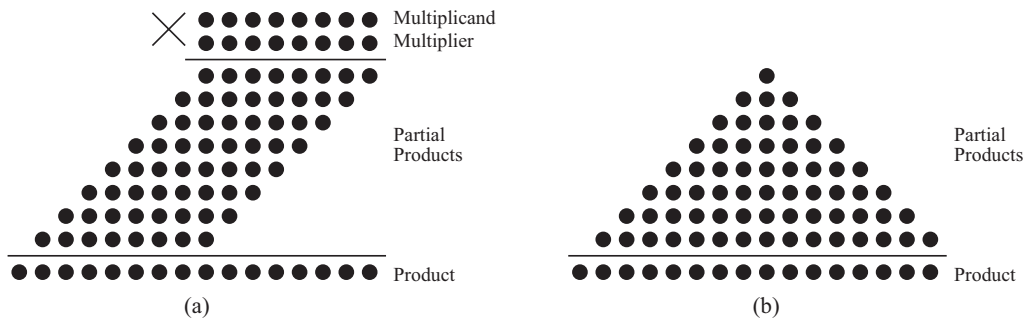


Figure 2.2: Block diagram for a shift-and-add multiplier



$$\begin{bmatrix}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}$$

(c)

Figure 2.3: The dot-diagram representation of an 8×8 -bit multiplier
 (a) Dot-diagram for shift-and-Add multiplication scheme
 (b) An alternative dot-diagram representation
 (c) The corresponding partial product matrix

represented with a 0 or FALSE. Therefore, the PPM is equivalent to dot-diagram representation of the PPs. Note that the size of the PPM can be arbitrary large as long as it contains all PPs. However, conventionally, the minimum size PPM that contains all PPs is chosen; i.e. both the first column and the last column in the PPM must include a PP (a TRUE value). The PPs are normally accumulated through several stages, resulting in a number of intermediate PPMs. Throughout this thesis the initial PPM is denoted as \mathbb{M}_0 and contains all initial PPs in the column corresponding to their weight. \mathbb{M}_i denotes the partial product matrix after i th stage of reduction. The operator $\|\mathbb{M}_i\|$ is also defined so that it returns the maximum height of the partial product matrix; i.e., the largest number of PPs with equal weight for \mathbb{M}_i .

2.1 Classification of Multipliers

Multiplication schemes are commonly classified in three general types: sequential, parallel and array multipliers [93, 106, 195]. This is not a universal classification and some hybrid multiplication schemes do not fall into exactly one of these categories. For example, as a compromise between sequential and parallel multipliers, partially combinational multipliers are introduced to achieve higher performance but still keep the hardware small. Examples of such multipliers are described in [4, 63, 107, 162]

2.1.1 Sequential Multipliers

The sequential multipliers generate the PPs sequentially and add each newly generated PP to the previously accumulated sum. The sequential multipliers were popular when the hardware was expensive and bulky. They are still in use in applications where the speed is not critical or the high parallelism achieved by multiple operating sequential multipliers compensates for the low speed. Shift-and-Add multiplication is an example of sequential multipliers.

2.1.2 Parallel Multipliers

The parallel multipliers generate all PPs in parallel and then use fast multi-operand adders for their accumulation. Parallel multipliers are the main focus of this thesis; Therefore the structure of parallel multipliers will be discussed more elaborately in Section 2.2.

2.1.3 Array Multipliers

Array multipliers iteratively utilize (almost) identical cells that generate new PPs and accumulate them simultaneously and therefore there is no separate circuit for PP generation and for their accumulation. In this way, the overhead that is due to the separate controls of these two steps is avoided. Examples of array multipliers are presented in [33, 49, 109, 110, 137].

2.2 Parallel Multipliers

Today parallel multipliers are popular because of the need for high speed operators. In addition to the high-speed requirement, more area is available nowadays, compared to the past. As mentioned earlier, the speed enhancement in multipliers are achieved in two ways: reducing the number of generated PPs and using faster techniques to accumulate the PPs. In order to investigate the speed enhancement methods in multipliers, the partial product generation (PPG) and partial product accumulation methods must be investigated.

2.2.1 Partial Product Generation

The PPs are generated using multiplexers or AND gates in an unsigned radix-2 shift-and-add multiplication. For multiplication of signed-magnitude numbers, the unsigned multiplication core can be used for the magnitude part of the inputs, with an extension that the sign bit is computed separately by checking the two input operands' sign bits. Multiplication of signed values with complement representation is a bit more complex. One way is to complement the negative operands, multiply the unsigned values and then complement the result if necessary; i.e. when only one of the input operands are negative. Such a pre-complement and post-complement method is suitable for 1's complement numbers but is too complicated for 2's complement numbers. For a 2's complement multiplier to yield correct product of its inputs, sign extension is needed on PPs. The PPs must be sign-extended to the width of the final product. Robertson in [158] suggests a more efficient procedure for 2's complement numbers compared to pre- and post-complement approach. Robertson classifies the multiplication in 2 cases. Case 1 is when the multiplier is positive. In this case the original shift-and-add approach (with right-shifts) can be utilized with only one consideration that the right-shifts are arithmetic shifts rather than logic shifts; i.e. when the multiplicand is neg-

ative, 1 will be entered in MSB instead of 0. Case 2 is when the multiplier is negative. In this case a correction step is required in the last step of shift-and-add multiplication. That is, the multiplicand will be subtracted instead of being added.

For 2's complement multiplication a number of direct methods are developed [6, 15, 70, 145]. Let us consider the following 2's complement representation for the multiplicand, the multiplier and the product:

$$\begin{aligned}
 \text{Multiplicand} \quad A &= -2^{M-1} a_{M-1} + \sum_{i=0}^{M-2} 2^i a_i \\
 \text{Multiplier} \quad B &= -2^{N-1} b_{N-1} + \sum_{j=0}^{N-2} 2^j b_j \\
 \text{Product } (A \times B) \quad P &= -2^{N+M-1} p_{N+M-1} + \sum_{k=0}^{N+M-2} 2^k p_k
 \end{aligned}$$

Therefore, the product will be computed as:

$$\begin{aligned}
 P &= \left(-2^{M-1} a_{M-1} + \sum_{i=0}^{M-2} 2^i a_i \right) \times \left(-2^{N-1} b_{N-1} + \sum_{j=0}^{N-2} 2^j b_j \right) \\
 &= \sum_{i=0}^{M-2} \sum_{j=0}^{N-2} a_i b_j 2^{i+j} + a_{M-1} b_{N-1} 2^{N+M-1} \\
 &\quad - \left(\sum_{j=0}^{N-2} a_{M-1} b_j 2^{j+M-1} + \sum_{i=0}^{M-2} a_i b_{N-1} 2^{i+N-1} \right) \quad (2.2)
 \end{aligned}$$

Figure 2.4(a) depicts the product computation for 2's complement numbers. In order to avoid the complication that subtraction or sign extension will introduce in most applications, Baugh and Wooley in [6] proposed an efficient method for 2's complement multiplication that handles subtractions by taking the 2's complement of the terms to be subtracted. In 2's complement representation negation of an integer word X can be obtained by the formula $-X = \text{comp}(X) + 1$; where $\text{comp}(X)$ is bitwise inversion of the bits in word X . Figure 2.4(b) illustrates the PPM formation using Baugh-Wooley's method. Baugh-Wooley's strategy increases the maximum column height by 2 and this can potentially increase the accumulation delay. Hatamian and Cash in [70] modified Baugh-Wooley multiplier and reduced the maximum number of PPs to its previous value as in the unsigned multiplier. Figure 2.4(c) illustrates the modified 2's complement multiplier.

					a_4	a_3	a_2	a_1	a_0
					b_4	b_3	b_2	b_1	b_0
					\times				
						$-a_4b_0$	a_3b_0	a_2b_0	a_1b_0
						a_3b_1	a_2b_1	a_1b_1	a_0b_1
					$-a_4b_2$	a_3b_2	a_2b_2	a_1b_2	a_0b_2
					a_3b_3	a_2b_3	a_1b_3	a_0b_3	
					$-a_4b_3$	a_3b_4	a_2b_4	a_1b_4	a_0b_4
					a_3b_4	a_2b_4	a_1b_4	a_0b_4	
					$-a_4b_4$	$-a_3b_4$	$-a_2b_4$	$-a_1b_4$	$-a_0b_4$
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

(a)

					a_4	a_3	a_2	a_1	a_0
					b_4	b_3	b_2	b_1	b_0
					\times				
						$a_4\overline{b_0}$	a_3b_0	a_2b_0	a_1b_0
						$a_3\overline{b_1}$	a_2b_1	a_1b_1	a_0b_1
						$a_4\overline{b_2}$	a_3b_2	a_2b_2	a_1b_2
						$a_3\overline{b_3}$	a_2b_3	a_1b_3	a_0b_3
						$a_4\overline{b_4}$	a_3b_4	a_2b_4	a_1b_4
						$a_3\overline{b_4}$	a_2b_4	a_1b_4	a_0b_4
						$\overline{a_4}$	a_4		
						$\overline{b_4}$	b_4		
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

(b)

					a_4	a_3	a_2	a_1	a_0
					b_4	b_3	b_2	b_1	b_0
					\times				
						$\overline{1}$	$\overline{a_4b_0}$	a_3b_0	a_2b_0
						$\overline{a_4b_1}$	a_3b_1	a_2b_1	a_1b_1
						$\overline{a_4b_2}$	a_3b_2	a_2b_2	a_1b_2
						$\overline{a_4b_3}$	a_3b_3	a_2b_3	a_1b_3
						$\overline{a_4b_4}$	a_3b_4	a_2b_4	a_1b_4
						$\overline{a_3b_4}$	a_2b_4	a_1b_4	a_0b_4
						$\overline{a_2b_4}$	a_1b_4	a_0b_4	
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

(c)

Figure 2.4: Signed multiplication

(a) 2's complement multiplication with positive and negative partial products.

(b) Baugh-Wooley multiplier 2's complement multiplier

(c) Modified Baugh-Wooley 2's complement multiplier

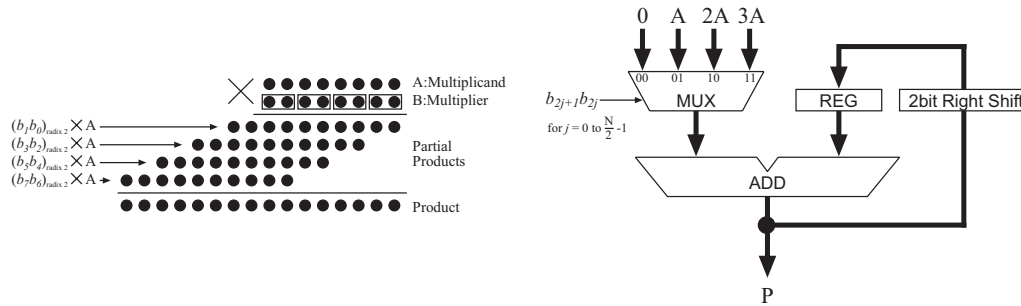


Figure 2.5: 8×8 -bit Radix-4 multiplication in dot-diagram and its structure

In addition to the Baugh-Wooley and modified Baugh-Wooley multiplication methods, an alternative method to handle 2's complement multiplication is proposed by Pezaris in [145]. Pezaris's method employs different types of full-adders depending on whether they have positive or negative weight.

2.2.1.1 Higher Radix Multipliers

Reduction in the number of generated partial products is favorable because it reduces the complexity of the following accumulation steps; thus it decreases the execution and amount of hardware involved. One way to reduce the number of PPs is to examine two or more bits of the multiplier at a time. Although higher representation radix leads to fewer digits, it requires generation of multiples of the multiplicand. For example in radix-4 multiplication each digit can assume the values 0, 1, 2 or 3; therefore 0, A , $2A$ and $3A$ are needed, where A is the multiplicand. In fact a part of the complexity is moved from the PP accumulation step to the PP generation step. This method is efficient specially when these multiples of the multiplicand can be precomputed [141, Chapter 10].

As illustrated in Figure 2.5, the number of partial products are reduced by a factor 2 for radix-4 multiplication. The multiples A and $2A$ are easy to obtain ($2A$ is simply the shifted version of A), but $3A$ is a hard multiple and requires an addition operation ($3A = 2A + A$). This addition can be combined with the addition required with each shift-and-add step. Doing so there will be a need for a 3-operand adder. As will be discussed later in Section 2.2.2.1, carry-save adders can be used to reduce the number of operands from three to two. The time required to perform this computation is approximately equal to the delay of one full-adder cell (i.e. 2 XOR gates) regardless of the number of bits in the operands.

Table 2.1: Booth's recoding algorithm

Current input bit b_j	Previous input bit b_{j-1}	Recoded bit b'_j	Explanation
0	0	0	Consecutive zeros
0	1	1	End of a string of ones
1	0	$\bar{1}$	Start of a string of ones
1	1	0	Consecutive ones

2.2.1.2 Recoding Techniques

Various algorithms for reducing the number of generated partial products have been proposed. Many of these algorithms benefit from the fact that a group of consecutive zeros or ones may generate fewer PPs. A group of i consecutive ones (or zeros) can be represented by the difference between two bit sequences each having a single nonzero bit. The string property is the base for such algorithms:

$$2^{j+i-1} + 2^{j+i-2} + \dots + 2^{j+1} + 2^j = 2^{j+i} - 2^j \quad (2.3)$$

The longer the sequence of ones or zeros, the larger is the achieved saving. In order to benefit from this transformation, the binary number representation needs to be changed from the conventional representation in digit set $\{0,1\}$ to the signed-digit representation in digit set $\{-1,0,1\}$ [5]. The process of converting the representation digit set to another digit set is referred as recoding in the literature.

Although the use of digit set $\{-1,0,1\}$ dates back to 1897 [60], Booth's recoding in [15] was one of the first algorithms to benefit from the property described above to accelerate serial multiplication. Booth proposed a simple recoding algorithm described in Table 2.1. that examines the current bit b_j and the previous bit b_{j-1} of the multiplier $b_{N-1}b_{N-2}\dots b_1b_0$ in order to generate the i th bit b'_i of the recoded multiplier $b'_{N-1}b'_{N-2}\dots b'_1b'_0$. Booth's recoding algorithm may reduce the number of non-zero bits in the multiplier but it has two main drawbacks. First, the number of non-zero digits is variable and in synchronous applications the worst-case maximum number of required addition/subtractions should be considered which is the same as the original multiplier width. Second, in some cases this algorithm may increase the number of non-zero bits. An example of such inefficient recoding is the string $(001010101)_2$ which is recoded to $(01\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1})_2$. The original word has 4 non-zero bits while the recoded word has 8. The maximum

Table 2.2: Radix-4 modified Booth's recoding algorithm

b_{2j+1}	b_{2j}	b_{2j-1}	b'_{2j+1}	b'_{2j}	increment
0	0	0	0	0	0
0	0	1	0	1	$+A$
0	1	0	0	1	$+A$
0	1	1	1	0	$+2A$
1	0	0	$\bar{1}$	0	$-2A$
1	0	1	0	$\bar{1}$	$-A$
1	1	0	0	$\bar{1}$	$-A$
1	1	1	0	0	-0

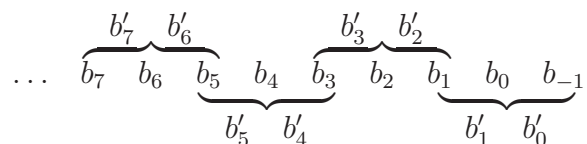


Figure 2.6: Overlapping multiple bit scanning in radix-4 MBR

number of non-zero bits can be reduced using overlapping multiple bit scanning. Modified Booth's recoding (MBR) proposed by MacSorley in [107] examines three overlapping bits of the multiplier at a time and recodes them into two digits (Figure 2.6). The radix-4 MBR algorithm is summarized in Table 2.2. Using the MBR can reduce the maximum number of required additions/subtractions to $\frac{n}{2}$ for an $n \times n$ -bit multiplier at the cost of somewhat increased complexity for each iteration [159]. However, the MBR is simpler than Reitwiesner's algorithm [154] for canonical signed digit (CSD) conversion, which guarantees the minimum number of non-zero digits. The CSD conversion using Reitwiesner's algorithm requires carry propagation and therefore it has to generate the multiplier bits sequentially. The canonical recoding has application in constant multiplications and modular exponentiation [77]. The MBR algorithm can be extended to higher radices but this will require hard multiples like $\pm 3A$, adding more complexity to the PP generation step [108]. Flynn and Oberman in [55] present methods for simpler generation of these hard multiples. Extensions to radices-8 and -16 are described in [51, 161, 206]. Seidel et al. in [166] present multipliers in radix-32 and radix-256.

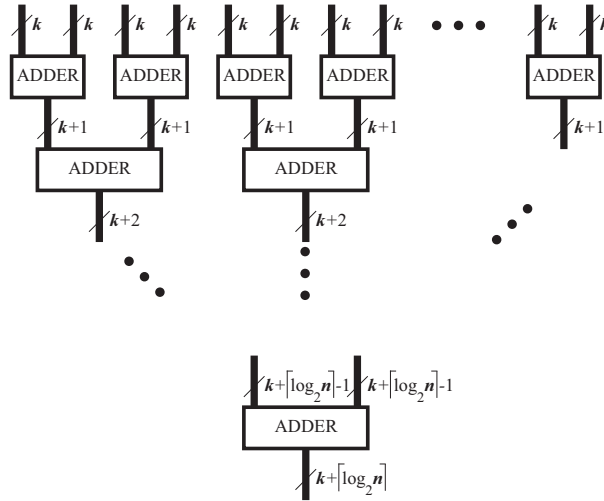


Figure 2.7: Accumulating PPs using two-operand adders

2.2.2 Partial Product Accumulation

After the generation of the PPs, they must be accumulated to obtain the final product. In the multiplication operation, this is the most time consuming process. The accumulation of the PPs, also referred to as reduction of the PPs, is performed using two main approaches: Reduction by rows and reduction by columns. The building modules are called adders and counters for reduction by rows and reduction by columns, respectively. A straightforward method in the reduction by rows is to use multiple two-operand carry propagate adders (CPAs). Using logarithmic time two-operand CPAs, the accumulation time needed to accumulate n words of width k bits will be equal to [141]:

$$\begin{aligned}
 T_{fast\ two\ operand\ adders} &= O(\log n + \log(n+1) + \dots + \log(n + \lceil \log_2 k \rceil - 1)) \\
 &\approx O(\log k \log n + \log k \log \log k)
 \end{aligned} \tag{2.4}$$

The accumulation process is illustrated in Figure 2.7. $\lceil n/2 \rceil$ adders of width k bits compute outputs of width $k+1$ bits each. Then $\lceil n/4 \rceil$ adders of width $k+1$ compute outputs of width $k+2$. This process is continued until the final product is computed. Parhami in [141] shows that using two-operand ripple carry adders (RCAs) instead of fast logarithmic adders results in computation time equal to:

$$T_{ripple\ carry\ adders} = O(n + \log k) \tag{2.5}$$

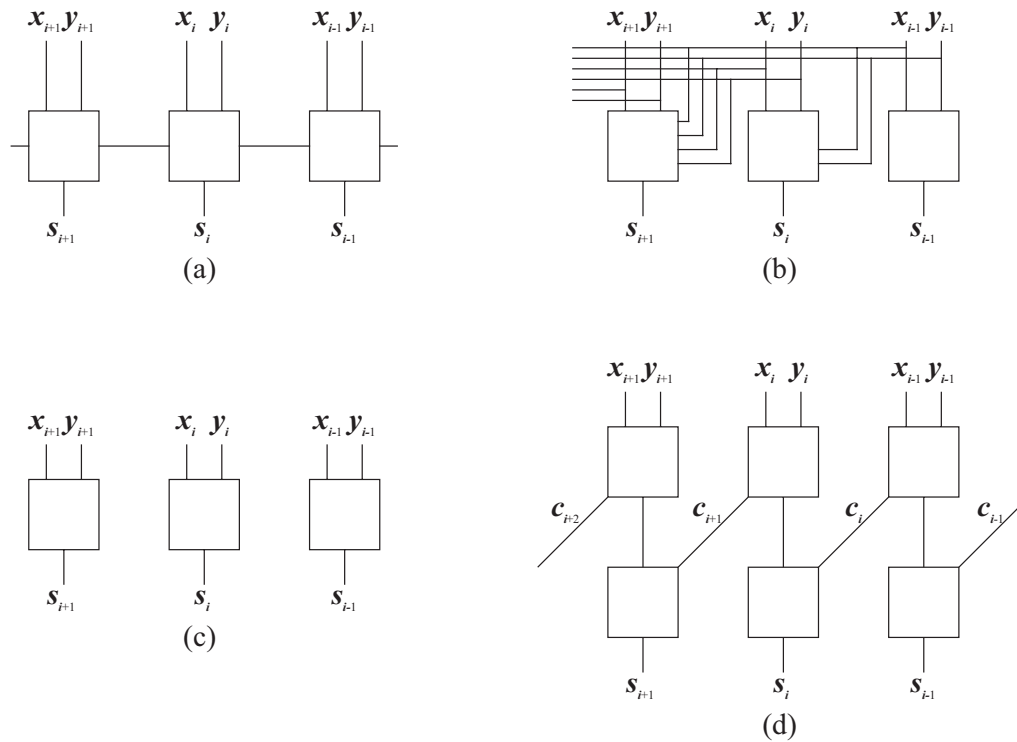


Figure 2.8: Carry propagation and carry-free operation

- (a) Single-stage operation with carry propagation
- (b) Single-stage operation with look-ahead
- (c) Single-stage carry-free operation
- (d) Two-stage carry-free operation

This is because the carry propagation in each level lags one time unit behind the preceding level. In fact accumulation time using RCAs can be less than the accumulation time using fast adders for large k . The absolute minimum time required for accumulation is $O(\log(nk)) = O(\log n + \log k)$, where nk is the total number of bits to be processed by the multi-operand adder, which is composed of constant-fan-in logic gates. This minimum is achievable using redundant multi-operand adders with redundant outputs in carry-save form [141]. The objective of adopting redundant representation is to reduce the computation time by reducing the maximum carry propagation chain.

Propagation of the carry in arithmetic operations is time-consuming and slow (Figure 2.8(a)). Look-ahead techniques are usually expensive and unaffordable

(Figure 2.8(b)). Redundancy enables performing carry-free arithmetic operations. Although the idealistic single-stage carry-free operation in Figure 2.8(c) is impossible to realize with fixed digit set, the two-stage carry-free operations are feasible using the redundant representation illustrated in Figure 2.8(d). Metze and Robertson in [121] presented the first example of using carry-save numbers for fast addition of a sequence of binary operands.

2.2.2.1 Carry-Save Adders

Carry-save adders (CSAs) are efficient operators when three or more operands are to be added. Their efficiency is due to the fact that the addition is performed without propagating carries. Carry-save addition was introduced by Estrin et al. in the context of sequential multiplication [53]. In the simplest implementation of a CSA, the basic elements of the CSA are full-adders (FAs) and half-adders (HAs).

The half-adder in Figure 2.9(a) adds two binary inputs, A and B . The result is 0, 1 or 2, so two bits are required to represent the value. They are called the sum and carry-out denoted by S and C_{out} . The C_{out} output has double the weight of the other bits. The full-adder in Figure 2.9(b) has a third input called C_{in} . The truth-table and behavioral representation of the HA and FA are given in Tables 2.3 and 2.4. The notations \oplus , \cdot and $+$ in Tables 2.3 and 2.4 represent logical XOR, AND and OR operators, respectively. Typical gate-level implementations of HA and FA are depicted in Figure 2.10(a) and Figure 2.10(b). A number of efficient transistor-level FA implementations are surveyed in [28, 168]. From behavioral perspective, all three inputs of the FA are equivalent, because the FA is symmetric with respect to its inputs (Table 2.4). However, in the actual implementation of FAs the structure is not symmetric with respect to the inputs. Different permutations of the inputs to a FA or a HA may result in different overall computation delays, dynamic power consumption, or leakage current. For instance, in the implementation shown in Figure 2.10(b), transitions at the inputs A and B have to pass through two XOR gates to reach the output S , while it is only one XOR gate for the input C_{in} .

A row of full-adders can be viewed as a mechanism to reduce three numbers to two numbers. An RCA turns into a CSA if carries are saved rather than propagated. Figure 2.11 shows the relationship between an RCA and a CSA. Figure 2.11 illustrates also the dot-diagram of the inputs and outputs to RCA and CSA. For RCA, three dots in the rightmost column represent the inputs to the least significant bit FA. The remaining FAs have two dots in one column and the carry

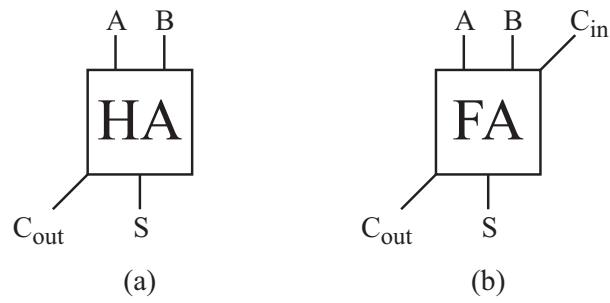


Figure 2.9: A half-adder(a) and a full-adder(b)

Table 2.3: Truth-table and behavioral representation of half-adder

A	B	S	C _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C_{in} = A \cdot B$$

Table 2.4: Truth-table and behavioral representation of full-adder

A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A \oplus B \oplus C$$

$$C_{in} = A \cdot B + B \cdot C_{in} + C_{in} \cdot A$$

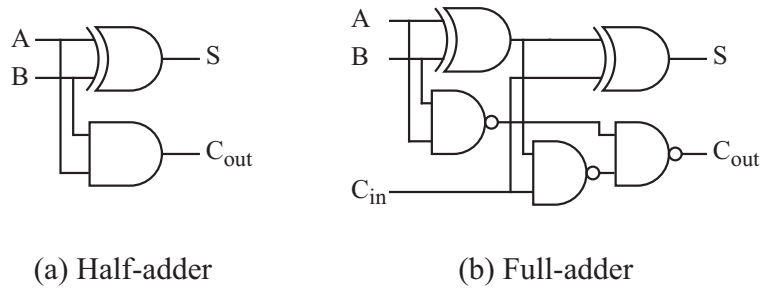


Figure 2.10: A typical gate-level implementation of full-adder and half-adder

out from the FA to its right. For a CSA, each FA has three dots in one column as inputs. For the CSA, a sum output will result in a dot with the same weight as the inputs. A carry output will result in a dot in the column to its left; i.e. one order of magnitude higher. Each dashed line in the dot-diagram represents the delay of one full-adder.

Let X , Y and Z be the input bit-vectors to a three-operand CSA, and let C and S be the carry and sum output bit-vectors, respectively. Then it can be written:

$$X + Y + Z = S + 2C \quad (2.6)$$

A row of full-adders produces a reduction from three bit-vectors to two bit-vectors and generates weighted outputs. Therefore a row of full-adders is also referred to as a $[3 : 2]$ adder. Early schemes for reduction by rows using $[3 : 2]$ adders are described in [107, 193]. MacSorley in [107] utilized CSAs in the structure of the radix-8 sequential multiplier. Wallace reduction method described in [193] accommodates a number of parallel CSAs. This method will be elaborated in Section 2.2.2.2.

The reduction of PPs can be performed using the generalized form of CSAs referred to as $[p : 2]$ adders which reduce the p bit-vectors to 2 bit-vectors [50]. A reduction-tree composed of $[4 : 2]$ adders is described in [197]. The generalized $[p : 2]$ adders are referred to as parallel compressors in [58]. Implementation of such adders are discussed in, among others, [96, 129, 148].

2.2.2.2 Wallace Reduction Tree

Wallace in [193] proposed a method for reduction by rows using CSAs in parallel. The partial product reduction tree (PPRT) obtained from this method is called a Wallace tree. In each stage of the reduction, Wallace performs a preliminary

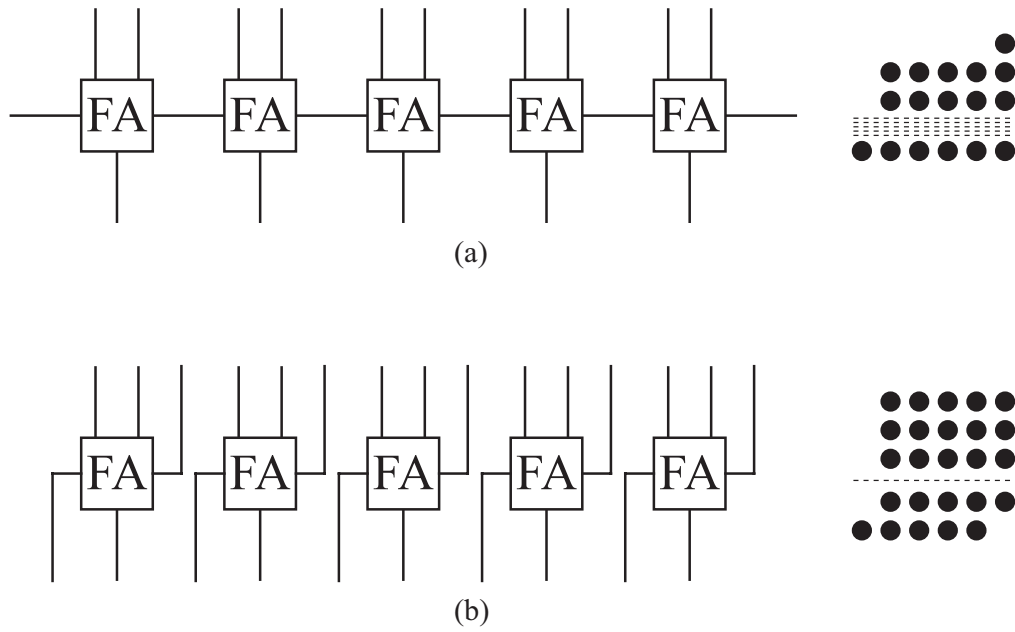


Figure 2.11: The relationship between RCA and CSA
 The structure and dot-diagram of (a) A ripple-carry adder
 (b) A carry-save adder

grouping of rows into sets of three. Within each three-row set, $[3 : 2]$ adders are employed to reduce columns with three bits to two bits. Half-adders are employed to reduce columns with only two bits. Rows which are not part of any three-row sets are transferred to the next stage without modification. The bits of these rows will be reduced in later stages. Figure 2.12 illustrates the reduction process for a 12×12 -bit unsigned multiplier. As mentioned, a sum output, S , from a full- or half-adder at one stage places a dot in the same column at the next stage. A carry output, C_{out} , from a full- or half-adder at one stage places a dot in the column to its left, i.e., one order of magnitude higher, at the next stage. As it is shown in Figure 2.12, in the second stage of reduction for example, the last two rows are left to the following stage. Recall the operator $\|\mathbb{M}_i\|$ which returns the maximum height of PPM in the i th stage of reduction. $\|\mathbb{M}_i\|$ can be recursively computed from the formula:

$$\|\mathbb{M}_i\| = 2 \lfloor \|\mathbb{M}_{i-1}\| / 3 \rfloor + (\|\mathbb{M}_{i-1}\| \bmod 3) \quad (2.7)$$

where $\|\mathbb{M}_0\|$ is equal to $\min(N, M)$ in an $M \times N$ -bit multiplier. When $\|\mathbb{M}_i\|$ is equal to 2, i is the last stage of reduction (denoted as l) and the reduction should be followed by a CPA to add the two final vectors. Each level of CSAs reduces the number of PPs by a factor of $3/2$. Thus the total number of the required stages is roughly equal to:

$$l \approx \lceil \log_{\frac{3}{2}} \left(\frac{\min(N, M)}{2} \right) \rceil \quad (2.8)$$

The total number full-adders in the Wallace's scheme is equal to [14]:

$$n_{FA} = N \cdot M - 2(N + M) + l + \eta \quad (2.9)$$

where η is 1, 2 or 3 depending on the values of N and M . The number of half-adders is at least $\min(N, M) - 1$, although it is often much larger than N and M . The word-length of the final CPA using Wallace's scheme is

$$w_{CPA} = N + M - l - \nu \quad (2.10)$$

where ν is either 1 or 2 depending on the values of N and M . For the 12×12 -bit multiplier in Figure 2.12, in order to reduce 12 bit-vectors to 2 bit-vectors, 5 stages of reduction is required. The number of full-adders and half-adders is equal to 102 and 34 respectively. The CPA length is 18 bits in this case.

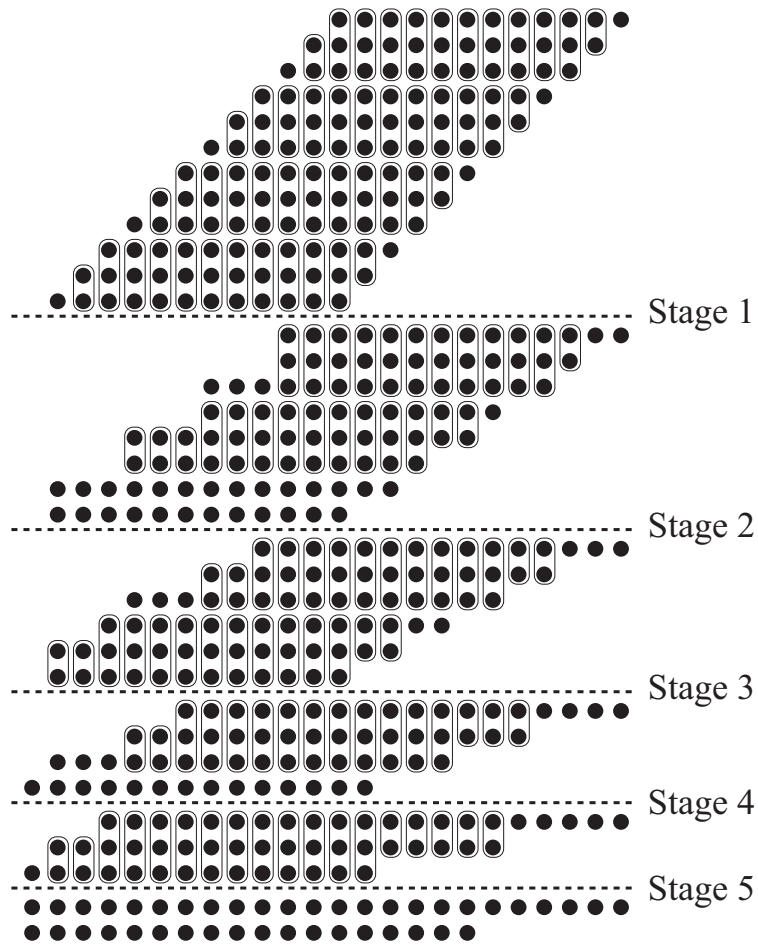


Figure 2.12: Wallace's scheme for a 12×12 -bit unsigned multiplier

Table 2.5: The Dadda sequence for minimal number of reduction stages

i	s_i	i	s_i	i	s_i	i	s_i
1	2	6	13	11	94	16	711
2	3	7	19	12	141	17	1066
3	4	8	28	13	211	18	1599
4	6	9	42	14	316	19	2398
5	9	10	63	15	474	20	3597

2.2.2.3 Dadda Reduction Tree

Following the Wallace's scheme for reduction by rows, Dadda in [38, 39] introduced a scheme for reduction by columns. Similar to Wallace's scheme, full-adders and half-adders are the building blocks of the PPRT, but they are referred to as counters. In the Wallace's method, the PPs are reduced as soon as possible. In contrast, Dadda's method carries out the minimum reduction necessary at each stage to perform the reduction in the same number of stages as required by a Wallace multiplier. In order to perform the minimum necessary reduction, a sequence of intermediate PPM heights is defined that provides the minimum number of reduction stages for a given multiplier size. This sequence is computed recursively as:

$$s_i = \lfloor \frac{3}{2} s_{i-1} \rfloor \quad (2.11)$$

where $s_1 = 2$. In other words, the sequence is determined by working back from the final two bit-vectors and limits the height of each intermediate matrix to the largest integer that is no more than $\frac{3}{2}$ times the height of its successor. The values of s_i for $1 \leq i \leq 20$ are given in Table 2.5.

Dadda's reduction scheme is as follows [176]:

1. Generate the PPs and initialize the partial product matrix \mathbb{M}_0 .
2. Find the largest i such that $\|\mathbb{M}_0\| > s_i$.
3. Employ full-adders and half-adders to obtain a reduced matrix with no more than s_i elements in any column.
4. Let $i = i - 1$ and repeat step 3 until a matrix with only two rows is generated.

The total number of full-adders for an $N \times M$ -bit multiplier using the Dadda PPRT is equal to [14]:

$$n_{FA} = N \cdot M - 2(N + M) + 3 \quad (2.12)$$

The total number of half-adders is $\min(N, M)$. The word-length of the final CPA using Dadda's scheme is

$$w_{CPA} = N + M - 2 \quad (2.13)$$

Dadda and Wallace multipliers are compared in [12, 66, 185]. Dadda's scheme is optimum in the number of required full-adders and half-adders but it requires a larger CPA. It has generally been assumed that compared to a Dadda multiplier, a Wallace multiplier yields a slightly faster design due to its smaller CPA. However, Townsend et al. in [185] show that this assumption may be incorrect, as the CPA can start the computation before all the bits of the final vectors are computed. In fact, using the 9-gate full-adder implementation given in [175] which consists of AND, OR and NOT gates, a Dadda multiplier is smaller and faster than a Wallace multiplier. Figure 2.13 illustrates the dot diagram of the reduction steps for a 12×12 -bit multiplier using Dadda's scheme. For the 12×12 -bit multiplier in Figure 2.13, similar to Wallace's scheme, 5 stages of reduction is required. The number of full-adders and half-adders is equal to 99 and 11 respectively. The CPA length is 22 bits in this case.

2.2.2.4 Modified Dadda-Wallace Reduction Tree

Bickerstaff et al. in [13] introduced a refinement of Wallace's and Dadda's methods referred to as reduced area multiplier. The modified Dadda-Wallace reduction scheme differs from Wallace's and Dadda's methods in that the maximum number of full-adders is used in the PPRT as soon as possible, while a number of half-adders are carefully placed to reduce the word size of the CPA. Doing so, the number of half-adders will be kept small (inheriting this property from Dadda's scheme). On the other hand, the word-length of the final CPA will be kept small similar to Wallace's scheme. Additionally, the modified Dadda-Wallace reduction scheme is also claimed to require minimum number of registers for a pipelined multiplier [13]. This scheme constructs the PPRT as follows:

For each stage, the maximum number of full-adders are used, i.e. $\lfloor \frac{p_j}{3} \rfloor$, where p_j is the number of bits in column j . Half-adders are used only (a) when required to reduce the number of bits in a column to the number of bits specified by the Dadda sequences (Table 2.5), or (b) to reduce the rightmost column containing exactly two bits.

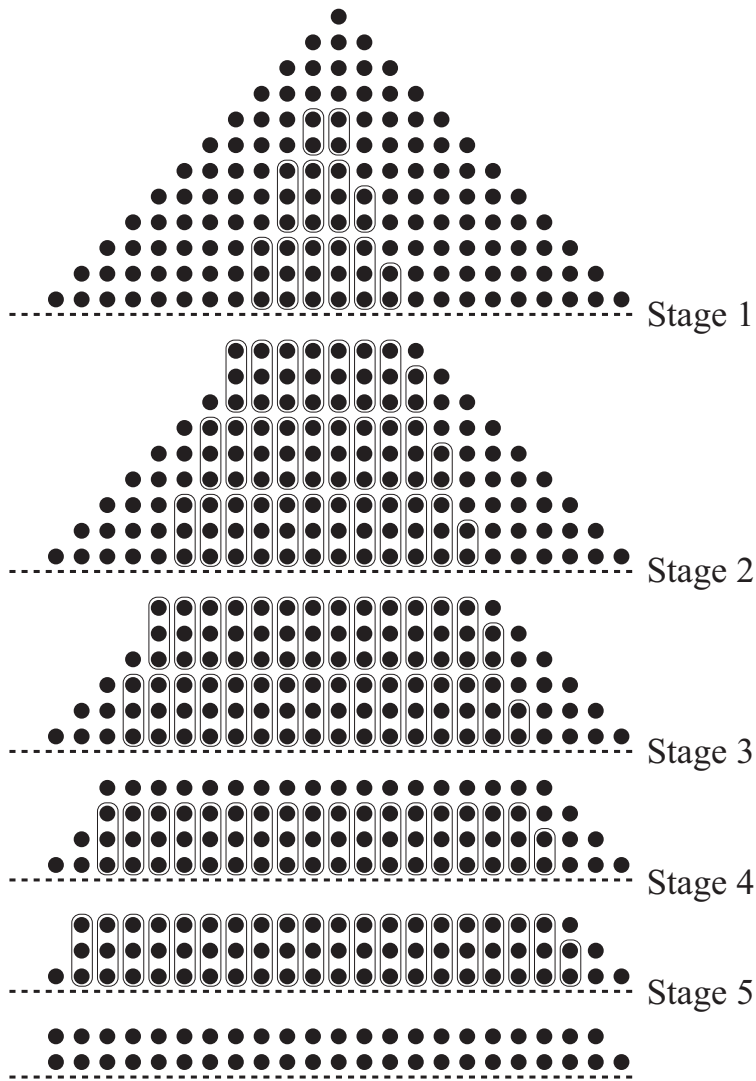


Figure 2.13: Using Dadda's strategy for a 12×12 -bit multiplier

Using this scheme, the number of required full-adders for an $N \times M$ -bit multiplier is equal to:

$$n_{FA} = N \cdot M - 2(N + M) + 3 + l \quad (2.14)$$

The total number of half-adders is $\min(N, M)$ and the word-length of the final CPA is

$$w_{CPA} = N + M - 2 - l \quad (2.15)$$

Figure 2.14 illustrates the dot diagram of the reduction steps for a 12×12 -bit multiplier using modified Dadda-Wallace scheme. For the 12×12 -bit multiplier in Figure 2.14, similar to Dadda and Wallace trees, 5 stages of reduction are required. The number of full-adders and half-adders is equal to 104 and 11 respectively. The CPA length is 17 bits in this case.

2.2.2.5 Generalized Parallel Counters

The reduction by columns can be done using parallel counters that add several equal-weight bits in one column and produce a binary output representing the number of ones among these bits. Full-adders and half-adders in Dadda's reduction scheme are the simplest forms of the parallel counters. A full-adder, represented as a $(3; 2)$ counter, counts the number of ones among its three inputs and represents the result as a 2-bit number. The counter modules can be generalized to a $(p; \lceil \log_2(p + 1) \rceil)$ counter that has p equal-weight input bits and produces a $\lceil \log_2(p + 1) \rceil$ -bit number at the output [173]. The $(p; q)$ counters can be implemented using full-adders and half-adders [56]. Figure 2.15 illustrates a $(10; 4)$ counter and a possible implementation of this counter. A parallel counter can be generalized further by allowing multi-column compression. A generalized parallel counter receives a number of PPs (not necessarily in a single column) and converts the PPs to another pattern (not necessarily one in each column). For example, a $(4, 6; 4)$ illustrated in Figure 2.16(a) receives 6 bits of weight 1 and 4 bits of weight 2 and outputs their sum in the form of a 4-bit binary number. The generalized parallel counters and their implementations are discussed in [47, 48, 91, 119, 140].

2.2.2.6 Parallel Multipliers with Redundant Full-Adders

The concept of carry-free reduction of partial products using redundant representation is useful in eliminating the intermediate carry propagation from Least

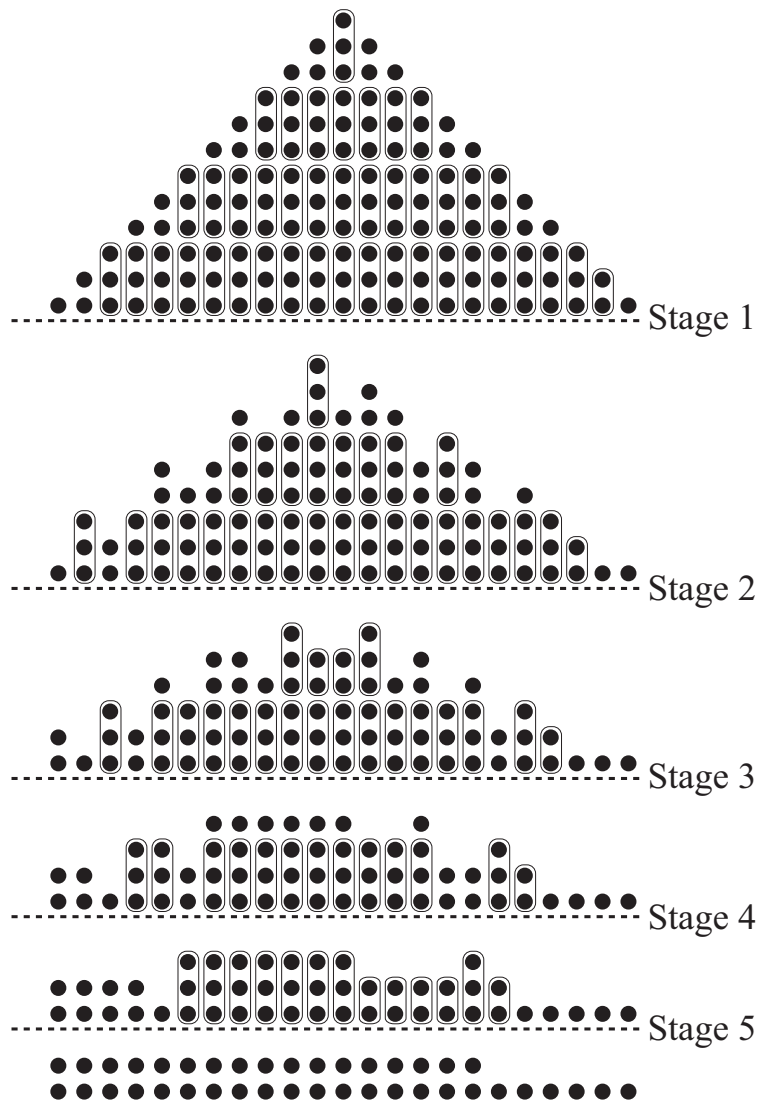


Figure 2.14: A reduced-area 12×12 -bit unsigned multiplier [13]

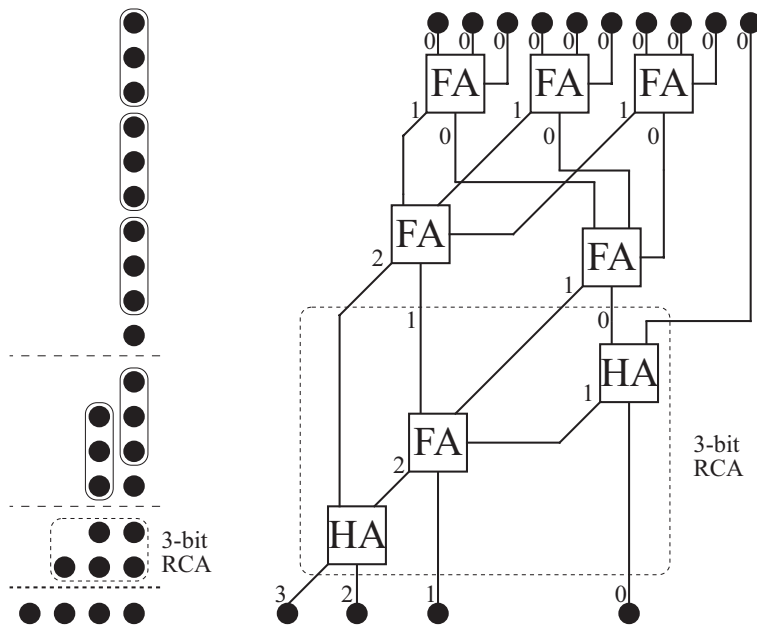


Figure 2.15: A (10; 4) counter and its possible implementation

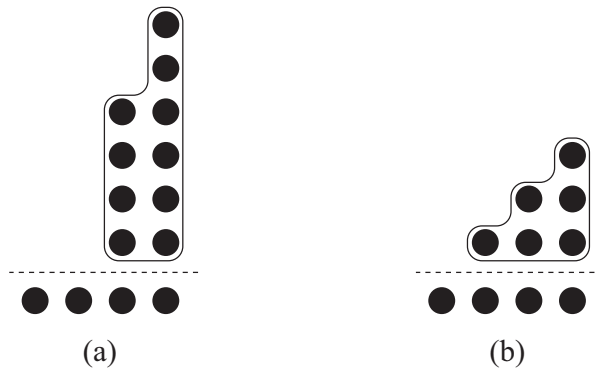


Figure 2.16: Examples of generalized parallel counters
 (a) The dot-diagram representation for a (4, 6; 4) counter
 (b) A (1, 2, 3; 4) counter.

Significant Bit (LSB) to Most Significant Bit (MSB). The CSA-based PPRTs were discussed earlier in this chapter. In addition to CSA-based PPRTs, a number of PPRTs are introduced in the past which utilize redundant full-adders [111, 183]. The parallel multipliers based on redundant full-adders are originally worse than CSA-based multipliers both in terms of delay and power consumption [43]. Huang et al. in [74] introduced a new encoding method for converting two's complement representation to signed digit representation, a development that has enhanced the performance of multipliers using redundant binary adders.

2.2.3 The Final Carry Propagate Adder

Accumulating the PPs in redundant form is fast and can result in the logarithmic delay growth for the reduction step, as discussed earlier in this chapter. Algorithms for multiplication with redundant form at inputs and outputs have been proposed [54, 55]. However, the majority of applications require inputs and outputs in conventional representation. Therefore after computation of the product in redundant form, it has to be converted into conventional form. For CSA-based PPRT, the conversion is performed using a carry propagate adder also referred as a vector merge adder (VMA). This step is also called assimilation of the carries. A number of alternatives exists for the final CPA, varying in the speed, area and complexity. Similar to PP generation algorithm and PP reduction scheme, the priorities like speed, area and power consumption must be considered simultaneously in selection of the CPA to achieve an optimal solution. An RCA or a Manchester adder [84] for example have the area and worst-case delay proportional to the adder's length. On the other hand, carry-lookahead adders [103, 198] offer logarithmic delay growth if they are constructed of constant-fan-in logic gates. Alternative solutions are parallel prefix adders, such as Kogge-Stone [92] and Brent-Kung [17], which offer close to logarithmic delay.

Most adders are built assuming that all inputs will arrive at the same time; i.e., all the input signals are stable at the beginning of the computation. However, the arrival times of the PPRT outputs are not uniform. The arrival time profile for a 100×100 -bit multiplier with Dadda reduction scheme under the fanout delay model assumption is depicted in Figure 2.17. The automated multiplier design program, used to design this 100×100 -bit multiplier, will be presented in Section 4.1. Figure 2.17 is the result of timing analysis for the generated 100×100 -bit multiplier. Under the fanout delay model assumption, the delay of each logic gate is set equal to the number of its fanouts. The vertical axis is the arrival

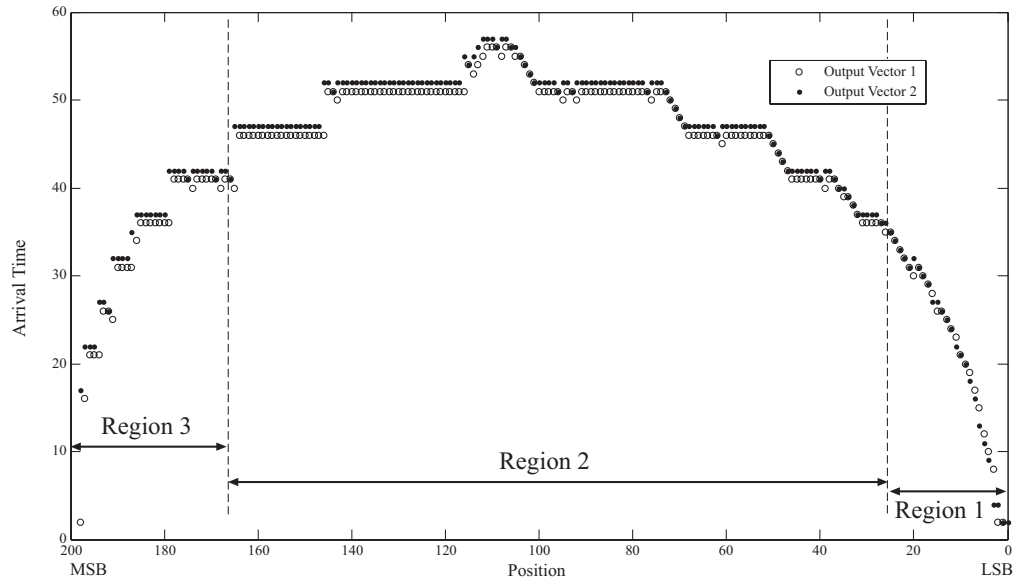


Figure 2.17: The arrival time profile for output bit-vectors of the dadda PPRT

time and the horizontal axis is the position of the output bits. A similar arrival time pattern is reported in [171].

Based on the timing characteristics of the output bit-vectors from the reduction step, the final adder structure can be optimized [32, 102, 170–172, 205]. The addition in region 1 (shown in Figure 2.17) can be done using a slow adder, such as an RCA. In region 2, a fast adder is required because the bits of this region arrive late. A logarithmic time adder is often used in this region. The RCA's length should be such that the carry-out is ready when the LSB of the adder in the second region arrives. Finally, the addition in region 3 can be done using a simple carry select adder. The carry select adder computes two outputs, assuming zero and one carry-in values. Depending on the carry-out value from the adder in the second region one of these values will be selected. The length of the carry select adder should be such that the outputs are ready when the carry-out from region 2 is ready.

The optimization of the final adder in a high-speed multiplier reduces the power consumption and area. High speed adders require large area, and then by using slower and smaller adders for region 1 and 3, the area of the whole multiplier is improved. Meanwhile, the use of a structure that computes as late as possible avoids some spurious transitions.

As mentioned earlier in Section 2.2.2.4, the modified Dadda-Wallace reduction scheme produces smaller output bit-vectors. Therefore the adder in region 1 (Figure 2.17) is absorbed in the PPRT.

2.3 Flexibility in the Reduction Tree

Multipliers and multi-operand adders can be subject to any standard combinational logic optimization. Such circuits are inherently more difficult to optimize because they are normally large and have a prevalence of exclusive-OR operations in their logic relations. Therefore, considering the arithmetic relations in the structure of multipliers and multi-operand adders can be extremely useful and can result in larger savings.

Arithmetic properties such as commutativity, associativity and retiming entail a large freedom in the structure of multi-operand adders. This freedom can be exploited to optimize area, delay, dynamic power and static power. The bases of the flexibilities that are used throughout this thesis are the commutativity and associativity of addition. The arithmetic equivalency inferred from this property is discussed in Chapter 4. In this section related works are reviewed.

2.3.1 Optimizing for Area, Delay and Hardware Resources

The word-level area and delay optimization of multi-operand adders using CSAs is explored in [87]. Various forms of arithmetic optimizations in word-level have been proposed in [146, 157] using arithmetic properties such as commutativity, associativity and retiming.

The timing and area trade off for CSA implementations has been studied extensively [83, 88, 89, 189, 202]. The bit-level optimization proposed by Khoo et al. in [83] introduces the concept of a relaxed carry-save adder and the flexibility of choosing among a number of arithmetically equivalent implementations of a multi-operand adder where the input operands have unequal word-length. The target architecture in [83] is CSA adders and the optimization target is a weighted cost function which is affected by the overall number of FAs, HAs and registers. In [83] authors conclude that the order of inputs to the CSA has a slight impact on the optimization as their cost function does not include power consumption and delay. In this thesis, it will be shown that the order of inputs to the FAs and HAs has a large impact on dynamic and static power consumption.

Oklobdzija et al. in [139] developed a three-dimensional minimization (TDM) algorithm to reduce the delay in the reduction tree. [139] benefits from unequal delays in a FA.

An essential difference between the optimizations in [83, 88, 89, 189, 202] and the method in this thesis is that, these optimizations are performed on the PPRT building blocks. However, in this thesis, a reduction scheme that is claimed to have small area [12] is chosen and the optimization is performed without altering the building blocks and only by reorganization of the interconnects. In fact, the proposed optimization in this thesis can be applied as a postprocess on the optimization results from majority of these works.

2.3.2 Optimizing for Power

In order to optimize the power dissipation of digital systems, and specifically multipliers, low-power strategies should be applied throughout the design process from system-level to process-level, while realizing that the performance is still essential. The different levels that can be considered for a design, from highest to lowest level of abstraction, are: system and algorithm level, architecture level, logic/circuit level and device/process level. Basically the majority of low-level power optimization methods for universal circuits can be applied for multipliers in particular. Examples of such low-level power optimizations are voltage scaling, layout optimization, transistor reordering and sizing [40, 42], using pass-transistor logic [2, 101], timing balancing [118, 128, 160] and node polarity optimization [118].

There has been substantial work on power optimization of multipliers in algorithm/architecture level. As discussed briefly in this chapter, a large variety of multiplication algorithms, partial product generation techniques and partial product reduction techniques have been proposed in the past. In addition to different multiplication algorithms, high-level power reduction strategies have been applied to multipliers. Among them are signal gating techniques and dynamic strategies for improving power [11, 19, 20], caching higher part of the operands to avoid redundant recomputations [68], bypassing parts of circuit based on the input values [41].

Optimization of power using the flexibilities induced from arithmetic properties are proposed among others in [100, 201]. Larsson and Nicol in [100] exploited redundancies in the structure of $[4 : 2]$ compressors and optimized the power consumption by selecting a proper boolean function for the compressors. Yu et al. in [201] proposed a reorganization method on Booth-encoded multipliers. Es-

estimates of switching probabilities based on the static probabilities of the nodes are used for reorganization. [201] introduces a number of guidelines for reducing the power consumption; For example, to assign those PPs having high switching probabilities a short signal path. This leads to a CSA array structure that adds the partial-products sequentially in the order of non-decreasing switching probabilities. Based on the experiments that will be discussed in Chapter 4, sorting the PPs based on their transition probabilities can lower the power consumption but the improvements are not significant because the spurious transitions are completely ignored in the transition probability quantifier.

A number of high-level optimization techniques are proposed in [75] using input data characteristics.

2.4 Choice of the Multiplier Structure

High-speed is essential in many real-time applications, and multipliers are typically in the critical path of such systems. Hence, strict requirements on speed are present for the multipliers. Among the large number of the existing implementation methods for multiplication, the parallel multipliers are proven to outperform others in speed. Tree multipliers such as Wallace and Dadda are shown to have substantially better performance compared to array multipliers [25, 66, 118, 185, 186]. A comparison between the power consumption of a CSA array multiplier, Wallace-tree multiplier and Booth multiplier is presented in [71]. According to [71], the average power consumption in Wallace-tree multiplier is lowest among these three structures. In this thesis the proposed optimization methods are applied to modified Wallace/Dadda PPRT [12]. However, with minor changes in the reduction tree generation algorithm, the proposed optimization methods are directly applicable on Wallace and Dadda PPRT. The optimization algorithm can be generalized to include redundant full-adder based PPRT and generalized counter based PPRT. As the optimizations are performed on the PPRT, the partial product generation method is chosen to be simple and general. Throughout the experiments in this thesis, the PP generation step for signed multipliers is modified Baugh-Wooley's scheme [6, 70]. For unsigned multipliers simple 2-input AND gates between the operand bits are used. The optimization method is directly applicable in presence of other partial product generation methods.

Chapter 3

Power Estimation in Combinational Circuits

Power estimation is an important step in the power-aware design. In order to optimize the PPRT, estimates of different solutions are needed. This chapter presents the power estimation techniques that will be used within the PPRT optimization algorithms in subsequent chapters. The techniques used for power estimation, especially the method introduced in Section 3.1.1, are among the contributions of this thesis. Some of the materials in Section 3.1.1 are published in [179].

3.1 Dynamic Power Estimation

Several power estimation techniques have been proposed in the past. These are surveyed in [133, 136, 144, 152]. Gate-level power estimation techniques can be divided into two main categories: Statistical sampling and probabilistic estimations. Statistical sampling approaches explicitly simulate the circuit with typical input streams [76, 81, 188]. To generate the input streams efficiently, Monte Carlo simulation techniques have been exploited [23]. Probabilistic approaches on the other hand, rely on the signal probabilities and spatiotemporal correlations between signals to estimate the power [10, 46, 59, 72, 73, 131]. The concept of probability waveforms, introduced in [132], is a compact representation of logic waveforms and consists of the signal probability and a sequence of events happening at different time instances. Tagged probabilistic simulation (TPS) [46, 187] is also an estimation technique based on the notion of tagged waveforms which models the set of all possible events at the output of each circuit node. The origin of

the estimation errors in the probabilistic power estimation methods is the glitch filtering and interdependency issues. In the original TPS a simple glitch filtering approach is followed. That is, if a rising transition appears close enough to a falling transition in a tagged waveform, they will both be removed from computations. The glitch filtering of TPS has been improved in [72] by introducing a dual-transition method to consider different combinations of tagged waveforms at a certain node, and filter out extra transitions if necessary. The dual transition method is improved further in [73] by reusing supergate notion from [167] (enclosing reconvergent fanouts) and improving interdependency issues in a circuit. Simple Waveform Set (SWS) is a novel power estimation method which will be presented in Section 3.1.1. It improves glitch filtering by taking into account the successor nodes when a glitch is generated, as a glitch might be filtered out in some of the successor nodes. Unlike the previous methods, which consider the glitch filtering locally, SWS considers the glitch filtering globally. This results in accurate estimations for tree-structured circuits and relatively smaller errors for general circuits with reconvergent fanouts.

3.1.1 Power Estimation using Simple Waveform Set

As discussed in Section 1.1.2.2, the average dynamic power consumed in a combinational circuit is given by:

$$P_{av} = \frac{1}{2} V_{DD}^2 \sum_{i=1}^N C_i D_i$$

where V_{DD} is the supply voltage, N is the total number of nodes, C_i is the total load capacitance at node i and D_i is the transition density at node i defined in Eq. 1.6. Blocks of combinational logic within a synchronous system are very common in digital circuits (Figure 3.1). In the context of parallel multipliers, it is assumed that the primary input operands arrive simultaneously at a clock edge and that the circuit under test (CUT) is built of pure combinational logic. The transitions on the primary combinational inputs then appear on the clock edge and the total delay of the combinational circuit is required to be less than the clock period. With these assumptions the transition density D_i in a pure combinatorial circuit can be rewritten as:

$$D_i = f_{clk} \bar{n}_i \quad (3.1)$$

where \bar{n}_i is the average number of transitions at node i in one clock cycle and f_{clk} is the clock frequency.

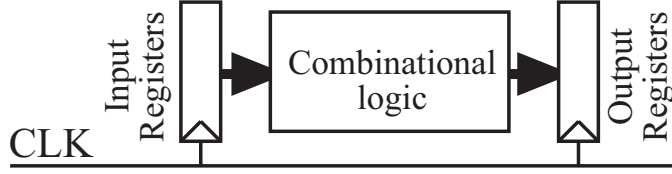


Figure 3.1: The target circuit architecture for power estimator

Deterministic delay assignments for the gates lead to a finite set of transition times for each node. The power estimator is restricted to deterministic delay assumptions. In addition, a node can only assume binary values ($\{0, 1\}$). As can be seen from Eq. 3.1, the computations are limited to one clock cycle. The clock period is assumed larger than the maximum delay of the combinational logic. The clock's active edge is assumed to appear at time 0. This infers that all nodes have settled to their final values after the previous clock edge at time 0.

Let us assume that primary inputs of the CUT change from X_u to X_v at time $t = 0$; where X_u and X_v are bit-vectors of N_b bits (N_b is the number of primary input bits). Therefore, $X_u, X_v \in \mathfrak{X}$, where \mathfrak{X} the set of all possible primary input values; i.e., $\mathfrak{X} = \{0, 1\}^{N_b}$. This transition at primary inputs will create a waveform at node i , denoted by $W_i^{X_u \rightarrow X_v}(t)$. The transition times set of $W_i^{X_u \rightarrow X_v}(t)$ is defined as:

$$\mathcal{T}_i^{X_u \rightarrow X_v} = \left\{ \tau \in [0, T_{clk}) \mid W_i^{X_u \rightarrow X_v}(\tau^+) = \neg W_i^{X_u \rightarrow X_v}(\tau^-) \right\} \quad (3.2)$$

That is, the time instances that $W_i^{X_u \rightarrow X_v}(t)$ has changed its values from 0 to 1, or vice versa. The occurrence probability of a transition time, t , at node i is equal to:

$$p_i(t) = \begin{cases} \sum_{(X_u, X_v) \in \mathcal{C}_i(t)} p(X_u \rightarrow X_v) & \mathcal{C}_i(t) \neq \emptyset \\ 0 & \mathcal{C}_i(t) = \emptyset \end{cases} \quad (3.3)$$

where $p(X_u \rightarrow X_v)$ is the probability of the transition X_u to X_v at the primary inputs and $\mathcal{C}_i(t)$ is the subset of all input transitions that will cause a transition at time t ; i.e.:

$$\mathcal{C}_i(t) = \left\{ (X_u, X_v) \in \mathfrak{X}^2 \mid t \in \mathcal{T}_i^{X_u \rightarrow X_v} \right\} \quad (3.4)$$

Under temporal independence assumption, $p(X_u \rightarrow X_v) = p(X_u)p(X_v)$ where $p(X_u)$ and $p(X_v)$ are the occurrence probabilities of inputs X_u and X_v ,

respectively. Thus, the average number of transitions at node i in one clock cycle, \bar{n}_i , can be expressed in terms of $p_i(t)$ s.

$$\bar{n}_i = \sum_{\tau \in \check{\mathcal{T}}_i} p_i(\tau) \quad (3.5)$$

where

$$\check{\mathcal{T}}_i = \left\{ \tau \in [0, T_{clk}] \mid p_i(\tau) > 0 \right\} \quad (3.6)$$

The objective is to approximate the set of all possible transition times ($\check{\mathcal{T}}_i$) and their corresponding probabilities. Once this is computed for all nodes, the dynamic power consumption can easily be calculated using Eq. 3.5 and Eq. 1.8. The Simple Waveform Set (SWS) is a method for representing and approximating the set of all possible transition times and their occurrence probabilities.

3.1.1.1 Simple Waveform Set

The SWS is build on waveforms with a special meaning. A waveform W is assumed to have exactly one rising edge and one falling edge. Thus, a waveform is denoted as a pair of time stamps $\Psi(t_r, t_f)$, where t_r and t_f are the times of the rising edge and falling edge respectively. An occurrence probability is also associated with each waveform denoted as $p(W)$ for waveform W . Figure 3.2 shows some example waveforms. The zero-holding and one-holding are in SWS method represented as $\Psi(+\infty, -\infty)$ and $\Psi(-\infty, +\infty)$, respectively. $+\infty$ shows that the transition appears after any imaginable time and will not be considered in power computations. Similarly $-\infty$ shows that the transition appears before any imaginable time and will not be considered in power computations. A waveform W is simple if either t_r or t_f is $+\infty$. A waveform is non-simple if both t_r and t_f are non-infinite times. A non-simple waveform is originally a glitch. A Simple Waveform Set \mathcal{S} is a set of simple waveforms and is complete if the sum of the waveform occurrence probabilities is equal to 1, i.e:

$$\sum_{W \in \mathcal{S}} p(W) = 1 \quad (3.7)$$

In order to keep SWS small, waveforms with equal rise and fall times are combined, i.e., the occurrence probabilities are summed.

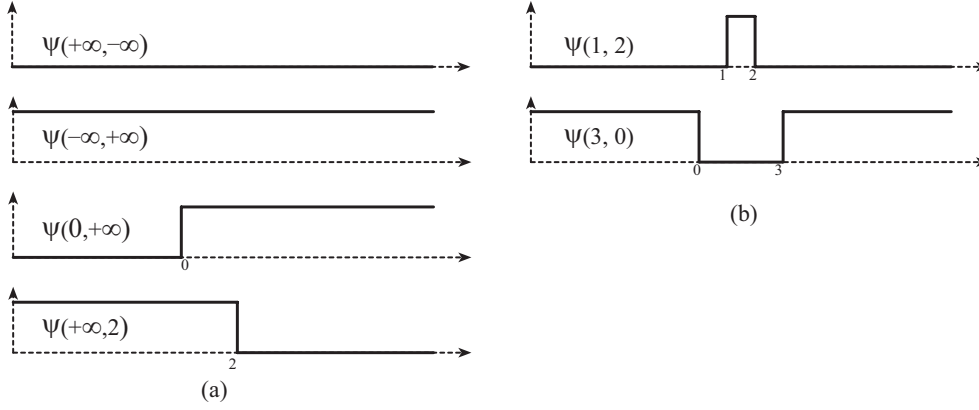


Figure 3.2: Examples of simple and non-simple waveforms
(a) Examples of simple transitions (b) Examples of non-simple transitions

The probabilistic power estimation commences by assigning SWSs to input nodes. The SWSs at primary inputs consist of four waveforms holding one, holding zero, zero-one transition and one-zero transition:

$$\left\{ \begin{array}{l} W_{11} = \Psi(-\infty, +\infty) \\ W_{00} = \Psi(+\infty, -\infty) \\ W_{01} = \Psi(0, +\infty) \\ W_{10} = \Psi(+\infty, 0) \end{array} \right\} \quad (3.8)$$

Under the temporal independence assumption for input values before and after time 0, occurrence probabilities can be assigned to each waveform based on their static probabilities. Let p be the one-probability at the corresponding input node. Then $1 - p$ will be zero-probability at this node. Hence,

$$\left\{ \begin{array}{l} p(W_{11}) = p \times p = p^2 \\ p(W_{00}) = (1 - p) \times (1 - p) = 1 - 2p + p^2 \\ p(W_{01}) = (1 - p) \times p = p - p^2 \\ p(W_{10}) = p \times (1 - p) = p - p^2 \end{array} \right\} \quad (3.9)$$

Once the SWSs are assigned to the inputs of a certain logic gate, the SWS at the output of that gate can be computed by computing the output waveform for each combination of the input waveforms. In some cases the output waveform may contain two non-infinite transition edges which cannot be considered as simple waveforms. In such cases the waveform will be decomposed into simple waveforms.

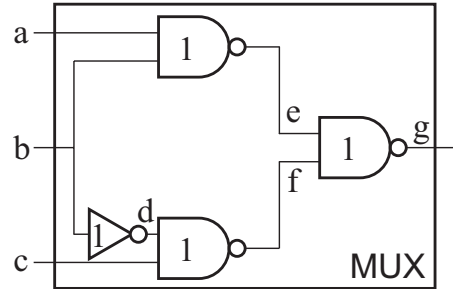


Figure 3.3: A 2-to-1 MUX example

Definition - Assume that $W = \Psi(t_r, t_f)$ is a non-simple waveform ($t_r \neq t_f$). $\text{decomp}(W)$ decomposes W into three simple waveforms:

If $t_r < t_f$, these simple waveforms are $W_1^* = \Psi(t_r, +\infty)$, $W_2^* = \Psi(+\infty, t_f)$ and $W_3^* = \ominus\Psi(-\infty, +\infty)$. If $t_r > t_f$, they are $W_1^* = \Psi(t_r, +\infty)$, $W_2^* = \Psi(+\infty, t_f)$ and $W_3^* = \ominus\Psi(+\infty, -\infty)$.

Occurrence probabilities for all decomposed simple waveforms are equal to the occurrence probability $p(W)$ of the original non-simple waveform. Waveforms marked by \ominus will be considered with negated occurrence probabilities and they are used to remove the extra transitions generated by the other two waveforms. The negative sign for the occurrence probability nullifies the extra waveforms, as two waveforms with equal rise edge and fall edge can be combined by adding the occurrence probabilities.

An example of decomposition is illustrated in Figure 3.4. The numbers inside each gate in Figure 3.3 represent the inertial delay of that gate. When $\Psi(+\infty, 2)$ and $\Psi(1, +\infty)$ are applied to inputs f and e , respectively, the output at node g will be $\Psi(3, 2)$. This waveform is non-simple and should be decomposed to simple waveforms $\Psi(3, +\infty)$, $\Psi(+\infty, 2)$, and $\ominus\Psi(-\infty, +\infty)$.

3.1.1.2 Computation of Simple Waveform Sets

For a given node in a combinational circuit, the output SWS is computed using its input SWSs as shown in Algorithm I in Figure 3.5. The computation for a node can commence if all input SWSs are computed earlier. The list of available nodes is initialized to the primary inputs in the beginning. After completion of the computations for each node, the list of available nodes should be updated by adding the new nodes that can be computed and removing the computed node. After computation of the SWS of a given node i , its transition density can be

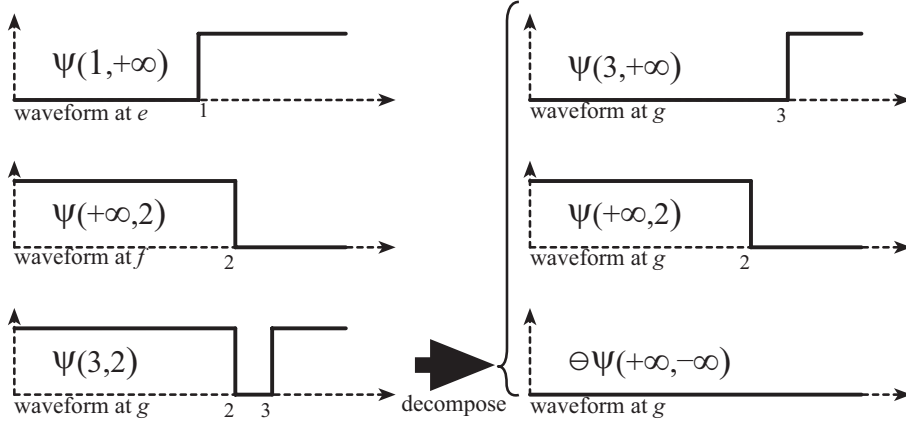


Figure 3.4: Generation and decomposition of non-simple waveforms

calculated. When SWSs are computed for all nodes having node i as input, the SWS for node i can be removed from memory to reduce the estimator's run-time memory requirement.

The waveform $W = \Psi(t_r, t_f)$ resulting from applying W_1 and W_2 to a logic gate f , has the occurrence probability of $p(W_1 \wedge W_2)$; i.e. the joint probability of two waveforms. If two waveforms are independent, the joint probability is equal to $p(W_1)p(W_2)$; where $p(W_1)$ and $p(W_2)$ are the occurrence probabilities of W_1 and W_2 , respectively.

The inertial and transport delay of the logic gate f , denoted as d_i^f and d_t^f respectively, are taken into account by adding the delay values d_i^f and d_t^f to the time stamps of the resulted waveform's rise and fall edges, i.e. t_r and t_f . For non-simple output waveform, if the glitch length $|t_r - t_f|$ is smaller than the inertial delay d_i^f , it will be replaced by constant zero or one depending on the waveform value at $+\infty$. This is because such a non-simple waveform (or glitch) cannot pass through a gate with the inertial delay d_i^f . Although the proposed technique is not limited to a particular delay model, a fanout delay model is utilized. The delays of the gates are assumed to be proportional to their fanout number. The simplify() procedure in Algorithm I locates waveforms with equal rise and fall times and equal values at $t = -\infty$ and $t = +\infty$ within the given SWS and replaces them with a waveform with the same rise and fall times and an occurrence probability equal to the sum of their occurrence probabilities.

```

function ComputeSWS ( $\mathcal{S}_1, \mathcal{S}_2$ )
{
   $\mathcal{O} = \emptyset$ ;
  for all  $W_1 \in \mathcal{S}_1$ 
  {
    for all  $W_2 \in \mathcal{S}_2$ 
    {
      compute  $W = f(W_1, W_2)$ ;
      if  $W$  is simple then
        add  $W$  to  $\mathcal{O}$ ;
      else
        {
          [ $W_1^*, W_2^*, W_3^*$ ] = decomp( $W$ );
          add  $W_1^*, W_2^*$  and  $W_3^*$  to  $\mathcal{O}$ ;
        }
    }
  }
  simplify( $\mathcal{O}$ );
  return  $\mathcal{O}$ ;
}

```

Figure 3.5: Algorithm I for computing SWSs

3.1.1.3 Dealing with interdependencies

The problem of interdependencies between different nodes arises in the presence of reconvergent fanouts (RFOs) in a combinational logic circuit. Indeed these interdependencies are time-variant under real delay models. For circuits with RFO paths, the exact computation of transition probabilities is a problem of NP-complexity. Several methods try to overcome this problem. [132] uses the supergate concept introduced in [167] and tries to estimate spatial dependencies, while [46] uses macroscopic correlations. Correlation coefficients can also be approximated using local ordered binary decision diagrams (LOBDDs) [82, 174]. Markov Chain models are utilized in [113, 164, 184] to capture correlations. In [113], Marculescu et al. describe a mechanism for computation of temporal and spatial correlations. The exact computation of spatiotemporal correlations is computationally heavy. In the SWS power estimator, similar to [46], macroscopic correlations are employed. Macroscopic correlations are time-independent correlations between the different nodes for fixed input values and zero-delay model. Since the values of nodes are assumed to be settled and delay independent at times $+\infty$ and $-\infty$, these values are used for capturing macroscopic correlations. These macroscopic correlations are encapsulated by correlation coefficients, which are defined as:

$$\kappa_{A,B}^{a,b} = \frac{p(A = a \wedge B = b)}{p(A = a)p(B = b)} \quad (3.10)$$

where A and B are two logic nodes in the circuit and a and b are two logic values ($a, b \in \{0, 1\}$). These correlation coefficients can be computed under the zero-delay model. In [52], a procedure for propagating signal probabilities from the circuit inputs toward the circuit outputs using only pairwise correlations between circuit lines and ignoring higher order correlation terms is described. The computation method is described in Appendix A. This method approximates joint probabilities using only pairwise correlation coefficients of all involved signals and ignores higher orders of correlations. This approximation, however, introduces small errors to the estimations [52].

In the SWS power estimation method, the joint probabilities of the waveforms are approximated using the correlation coefficients. Assume that W_A and W_B are two waveforms at inputs A and B of gate f . The occurrence probability for the waveform W at the output of f is $p(W) = p(W_A \wedge W_B)$ which is approximated by $\kappa p(W_A)p(W_B)$ where

$$\kappa = \kappa_{A,B}^{W_A|+\infty, W_B|+\infty} \cdot \kappa_{A,B}^{W_A|-\infty, W_B|-\infty} \quad (3.11)$$

Ercolani's method is utilized for computation of correlation coefficients (see Appendix A).

As an example let us consider the multiplexer circuit in Figure 3.3 again. Waveforms $W_1 = \Psi(1, +\infty)$ and $W_2 = \Psi(+\infty, 2)$ are applied to inputs e and f , respectively. A one-probability assignment of 0.5 at the primary inputs gives occurrence probabilities of each of these waveform of $\frac{3}{16}$. The RFO from node b to node g means that the waveforms at the inputs of node g are not independent. Using Ercolani's method gives the following correlation coefficients (more detailed computations in Appendix A):

$$\kappa_{e,f}^{1,1} = \frac{8}{9}, \kappa_{e,f}^{1,0} = \kappa_{e,f}^{0,1} = \frac{4}{3} \text{ and } \kappa_{e,f}^{0,0} = 0 \quad (3.12)$$

The waveform W_1 at node e is equal to 0 at $t = -\infty$ and is 1 at $t = +\infty$ since it rises at time 1. Similarly, the waveform W_2 at node f is equal to 1 and 0 at $t = -\infty$ and $t = +\infty$ respectively. This gives the following coefficient κ :

$$\kappa = \kappa_{e,f}^{1,0} \kappa_{e,f}^{0,1} = \frac{16}{9} \quad (3.13)$$

The occurrence probability of the resulting waveform $W = \Psi(3, 2)$ will therefore be

$$p(W) = \kappa p(W_1)p(W_2) = \frac{1}{16} \quad (3.14)$$

As discussed earlier, the waveform W is non-simple and should be decomposed to simple waveforms.

3.1.1.4 Glitch filtering

A logic gate will remove short glitches due to its inertial delay. Overestimation of power is resulted if glitch filtering is ignored or is imperfect. A glitch will be removed at a certain gate if the glitch length is shorter than the inertial delay of that gate. Glitches generated in a certain node of the circuit might be filtered out in some of the successor nodes. Due to differences in the inertial delays of the successor nodes, a situation can occur where the glitch is filtered out in some, but not all, of the successor nodes. In order to include glitch filtering in the SWS power estimator, a mask tag is assigned to each one of the waveforms in the SWS. Let M^W be the mask tag for waveform W . This tag is an array of boolean with the size of the total number of nodes in the system. M_i^W , the i :th element in array M^W , is `FALSE` if the waveform W will reach node i in the future and will be filtered out at node i ; otherwise it is `TRUE`. Several changes in Algorithm I are needed in order to include glitch filtering. Algorithm II in Figure 3.6 takes into account the glitch filtering. The `&` operators used for mask computations in Algorithm II are bitwise `AND` operations on two vectors.

If $W = \Psi(t_r, t_f)$ is a non-simple waveform, `decomp2(W)` decomposes the non-simple waveform, W , into simple waveforms similar to `decomp(W)`. However `decomp2(W)` generates five waveforms $W_{1..5}^*$. If $t_r < t_f$, they are $W_1^* = \Psi(t_r, +\infty)$, $W_2^* = \Psi(+\infty, t_f)$, $W_3^* = \ominus\Psi(-\infty, +\infty)$, $W_4^* = \ominus\Psi(+\infty, -\infty)$ and $W_5^* = \Psi(+\infty, -\infty)$. And if $t_r > t_f$, the decomposed waveforms are $W_1^* = \Psi(t_r, +\infty)$, $W_2^* = \Psi(+\infty, t_f)$, $W_3^* = \ominus\Psi(+\infty, -\infty)$, $W_4^* = \ominus\Psi(-\infty, +\infty)$ and $W_5^* = \Psi(-\infty, +\infty)$.

The `simplify()` procedure is altered to take into account the mask tags as well. Two waveforms with equal rise and fall times and equal values at $t = -\infty$ and $t = +\infty$ can be combined only if they have equal mask tags for the successor nodes of the current node.

The call `computemask(W)` computes a new mask, M^W , for the non-simple waveform W . M_i^W is `FALSE` if W can reach node i but cannot pass this node; otherwise M_i^W will be `TRUE`. Basically, `computemask(W)` has to traverse all successor nodes where the non-simple waveform W can reach; if the inertial delay of the node is larger than the glitch size of W , the corresponding node will block W . The new computed mask affects waveform $W_{1..4}^*$ from the decomposed waveforms, but not waveform W_5^* . Assume that node j and node k are successors of node i and that node j will pass the non-simple wave W , while node k will filter it out. As a result of Algorithm II, the j node mask tags of the decomposed waveforms $W_{1..5}^*$ will all be `TRUE`. Therefore, at node j , waveform W_4^* and W_5^* will be

```

function ComputeSWS ( $S_1, S_2$ )
{
   $\mathcal{O} = \emptyset$ ;
   $i = \text{current node ID}$ ;
  for all  $W_1 \in S_1$ 
  {
    for all  $W_2 \in S_2$ 
    {
      if ( $M_i^{W_1} \& M_i^{W_2}$ )
      {
        compute  $W = f(W_1, W_2)$ ;
        if  $W$  is simple then
        {
           $M^W = M^{W_1} \& M^{W_2}$ ;
          add  $W$  to  $\mathcal{O}$ ;
        }
        else
        {
           $M^W = \text{computemask}(W)$ ;
          [ $W_1^*, W_2^*, W_3^*, W_4^*, W_5^*$ ] =  $\text{decomp2}(W)$ ;
           $M^{W_{1..4}^*} = M^{W_1} \& M^{W_2} \& M^W$ ;
           $M^{W_5^*} = M^{W_1} \& M^{W_2}$ ;
          add  $W_{1..5}^*$  to  $\mathcal{O}$ ;
        }
      }
    }
  }
  simplify( $\mathcal{O}$ );
  return  $\mathcal{O}$ ;
}

```

Figure 3.6: Algorithm II for computing SWSs with glitch filtering

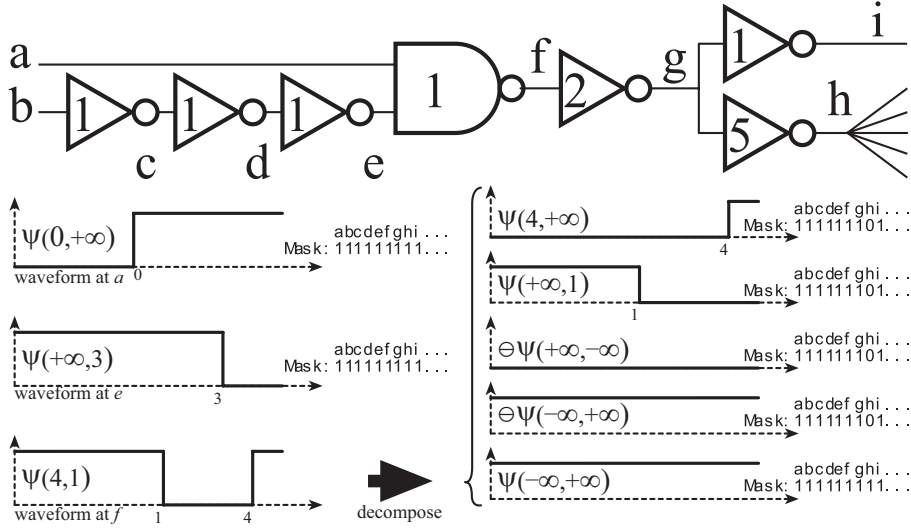


Figure 3.7: An example of glitch filtering

removed because of the negative sign in the occurrence probability of W_4^* . Only W_1^* , W_2^* and W_3^* will be left which is exactly as algorithm I. However, the node k mask tags are FALSE for waveforms $W_{1..4}^*$ so that they will be removed, leaving only waveform W_5^* . The glitch is consequently filtered out. Let us consider the example shown in Figure 3.7. A glitch with length 3 will be created at node f due to unequal delays of input paths of node f . The successor nodes i and h have different inertial delays. The glitch with length 3 created in node f will pass through node g and i , while it will be filtered out in node h . Therefore the node h mask tag of waveform $W_{1..4}^*$ generated at node f will be FALSE while all other mask tags will be FALSE. During the SWS computation for node h , all waveforms with FALSE mask tag value for node h will be removed.

The complete waveform sets, resulted from the SWS power estimator, for two example circuits are listed in Appendix B. The SWS power estimator described in this section will be applied to parallel multiplier PPRT with FAs and HAs as basic building blocks. The full-adder based PPRT exhibits special properties with respect to generated glitches and their filtering. Therefore it can be optimized for such structure to enhance the speed of estimations. Each PP (if not connected to the final CPA) is connected to exactly one FA or HA. Thus each PP is connected to exactly one NAND/AND gate and one XOR gate. An AND gate in the HA can be considered as a NAND gate followed by a NOT gate; hence, they have

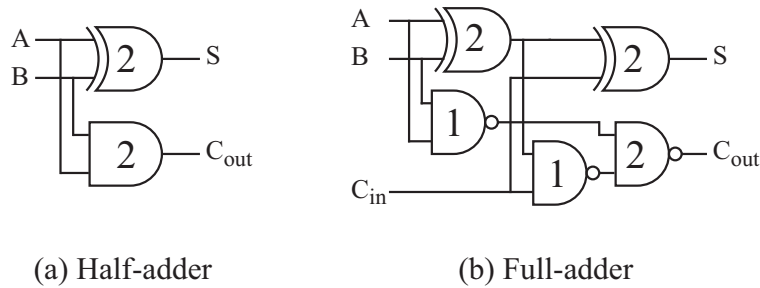


Figure 3.8: Propagation delays in Full-adders and Half-adders

similar input capacitance. The structure of the PPRT hence does not accommodate high-fanout nodes, and the maximum fanout is two. Figure 3.8 illustrates the gate propagation delays of a FA and a HA with the fanout delay model. With the fanout delay model assumption, the only gates that can generate a glitch that might be filtered are two NAND gates in the FA with unit delay. With this observation the glitch filtering in the PPRT can be simplified. The estimator will therefore be faster if it is not required to track the successor nodes for glitch filtering analysis.

3.1.1.5 Experiments

The proposed technique for power estimation is implemented as stand-alone software in C++. The power estimation tool includes the procedure for signal probability and correlation coefficient estimation. The inputs to the tool are the netlist, the delay model and one-probabilities of primary inputs. The delay model is chosen to be the fanout delay model. It should be emphasized that any realistic delay model can be used in the estimation system and it not restricted to the fanout model. However, using realistic delay models may increase the number of waveforms in the SWS because the delays are not integer numbers anymore. For the experiments in this section, the one-probabilities for primary inputs are in the experiments set to 0.5, but can be any arbitrary value between 0 and 1. In addition, the primary inputs are assumed to be temporally and spatially uncorrelated. Therefore, as discussed in Section 3.1.1.3, the pairwise correlation coefficients between primary inputs are initialized to one. In Chapter 5, a modification to this power estimator will be presented that enables the power estimation when primary inputs have spatiotemporal correlations. These modifications involve the initialization of the correlation coefficients and the waveform occurrence probabilities for primary input nodes. The reference transition densities for the experiments

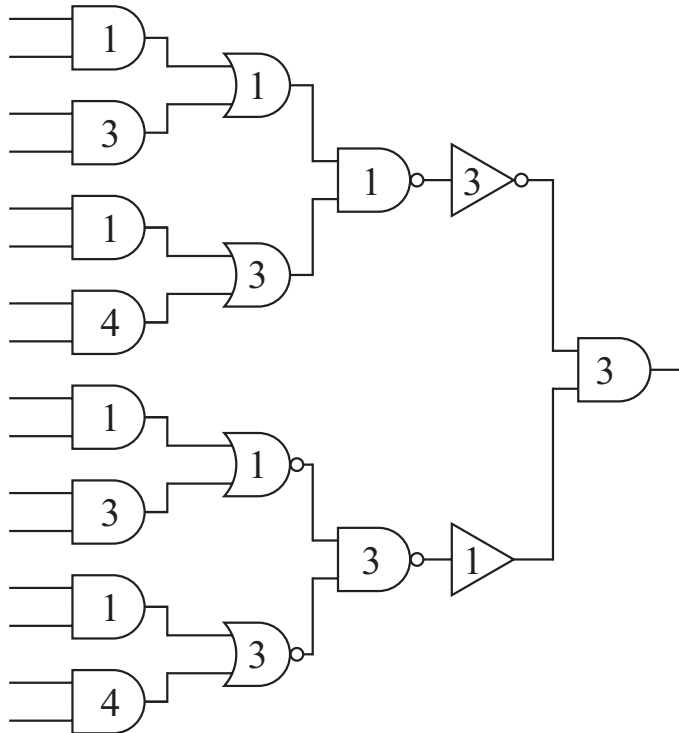


Figure 3.9: An example tree-structured circuit

in this section are acquired from a circuit logic simulation with 100000 random input vectors that satisfy the one-probabilities of the primary inputs (i.e, 0.5).

A sample tree structured circuit, illustrated in Figure 3.9, is analyzed in [72] and computation errors are reported. Node errors are up to 42% for the probabilistic simulation method [132] and up to 23% for the TPS method [46]. Dual-transition glitch filtering [72] reduces the computation error to maximum 3%. As opposed to these methods, the computation error for tree structured circuits (with no reconvergent fanouts) using SWS estimation technique is zero. This is because of the complete glitch filtering consideration for tree-structured circuits.

In the presence of RFOs errors are introduced in the estimations due to imperfect consideration of interdependencies between signals. More experiments are carried out using ISCAS'85 benchmark circuits. In order to be able to compare the results with those previously published, the fanout delay model is utilized; i.e., the delay of each gate is assumed proportional to its number of fanouts. For each circuit average node error (E_{av}), standard deviation (σ) and total power error

Table 3.1: Power estimation error for ISCAS’85 benchmark circuits with fanout delay assignment. All errors are in percentage.

Circuit	TPS			TPS-DT			TPS-EDT			SWS		
	E_{av}	σ	E_{tot}	E_{av}	σ	E_{tot}	E_{av}	σ	E_{tot}	E_{av}	σ	E_{tot}
C17	2.3	2.6	0.1	2.3	2.6	0.1	2.3	2.6	0.1	0.7	1.1	0.7
C432	29.9	38.8	35.8	9.5	11.8	6.5	11.5	16.6	11.5	5.2	9.6	2.8
C499	6.8	14.0	7.0	3.6	8.2	0.6	2.3	3.0	3.0	1.0	1.6	0.3
C880	8.3	15.3	1.6	8.0	15.7	5.2	4.8	9.0	0.0	4.4	7.9	2.7
C1355	24.2	31.6	32.9	5.8	11.2	5.4	5.0	9.5	0.5	6.3	9.5	0.2
C1908	15.0	23.1	4.1	17.7	27.9	11.2	7.0	16.3	2.0	7.8	11.1	2.3
C2670	16.6	29.8	7.2	16.7	28.3	9.9	13.2	23.6	6.2	9.7	16.7	2.9
C3540	13.8	26.3	9.8	10.3	25.6	2.4	10.5	26.4	3.7	7.3	17.9	1.5
C5315	11.8	24.4	2.3	13.4	31.5	10.1	11.3	27.0	3.4	7.1	11.5	2.1
C6288	27.4	27.5	32.1	15.7	18.8	4.1	12.7	15.4	0.2	15.7	20.1	4.9
C7552	14.5	27.5	3.2	14.8	31.4	7.8	14.1	27.6	1.3	11.5	27.7	0.9
AVE.	15.5	23.7	12.4	10.7	19.4	5.8	8.6	16.1	2.9	7.0	12.2	1.9

(E_{tot}) are reported in Table 3.1. E_{av} is the average relative error in node transition density estimation and E_{tot} is relative error in total power. The errors are reported in percentage. The error numbers for TPS, dual transition TPS (referred as TPS-DT) and Enhanced dual transition TPS (referred as TPS-EDT) from [46], [72] and [73] are reported for comparison. On average, and for most cases better results are achieved using SWS technique. One measurement of quality of an estimate is average error. Another important measure is standard deviation (σ), which measures the uncertainty of the estimate. The improvement over TPS-EDT, namely σ equals 12.2% versus 16.1%, is significant, because large errors contribute more to the sigma value. TPS-EDT provides better estimation results in some cases which is mainly because of better dependency coverage in this method. From Table 3.1 and 3.2 it can be seen that SWS (as well as TPS, TPS-DT and TPS-EDT) in general gives better estimates for circuits with small logic depth. The waveform set at the output of a gate is computed based on its input waveform sets, therefore errors might propagate and accumulate from earlier stages to later stages. However, one should note that practical circuits are typically optimized to avoid excessively large logic depths, so the circuits with large logic depths are mainly used to evaluate the performance of the estimators.

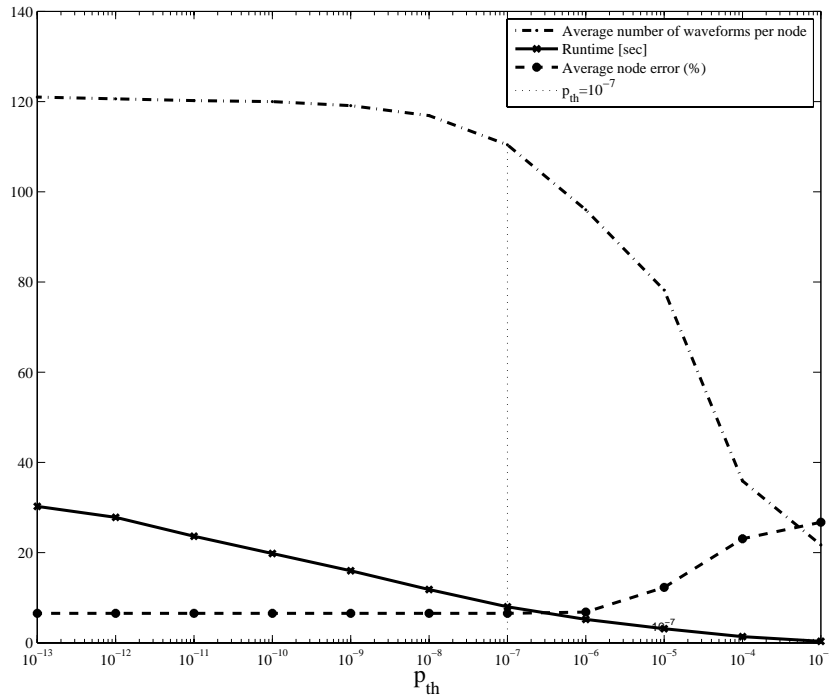


Figure 3.10: Average node error, average number of waveforms per node and runtime versus p_{th} for C1355

During the computation phase, waveforms with extremely small occurrence probabilities can be ignored in order to achieve higher speed. All the generated waveforms are compared to an experimentally determined threshold p_{th} ; if the occurrence probability is smaller than p_{th} this waveform will be ignored. The value of p_{th} affects the runtime and accuracy of computations. Figure 3.10 shows the variation of the runtime, the average number of waveforms, and the average error for the C1355 circuit. For the experiments in this section $p_{th} = 10^{-7}$ is chosen since this value provides a very small degradation from the case with $p_{th} = 0$ while still achieving a relatively short runtime.

Table 3.2 summarizes the estimation errors from computing correlation coefficients as discussed in Sec. 3.1.1.3. E_{av2} and E_{tot2} in Table 3.2 are average and total relative estimation errors respectively where the correlation coefficients are acquired from zero-delay logic simulations. In this case the correlation coefficients are error-free. However, errors still exist in the power estimations because microcorrelations are approximated by macrocorrelations. E_{av1} and E_{tot1} in Ta-

Table 3.2: Runtime and computation complexity for ISCAS’85 benchmark circuits with fanout delay assignment. All errors are in percentage.

	Runtime (sec)	Ave. # of waveforms	Ave. logic depth	Maximum logic depth	Number of nodes	E_{av1}	E_{tot1}	E_{av2}	E_{tot2}
C17	0.003	8.4	1.1	3	11	0.7	0.7	0.7	0.7
C432	7	130.6	11.2	29	252	5.2	2.8	5.1	1.0
C499	0.3	23.1	5.9	13	287	1.0	0.3	0.9	0.1
C880	7	110.7	8.7	30	495	4.4	2.7	4.5	2.0
C1355	8	110.4	12.7	26	631	6.3	0.2	6.3	0.4
C1908	77	168.6	16.1	44	1090	7.8	2.3	6.4	1.2
C2670	205	125.8	8.4	39	1633	9.7	2.9	9.3	2.5
C3540	881	275.2	13.4	56	2033	7.3	1.5	5.8	0.7
C5315	570	141.1	9.5	52	3151	7.1	2.1	6.8	2.0
C6288	3984	1268.3	42.1	124	2448	15.7	4.9	14.9	3.5
C7552	1201	154.4	11.1	45	4249	11.5	0.9	10.0	0.8

Table 3.2 are average and total relative estimation errors when correlation coefficients are computed using the method in [52]. The computation complexity of the SWS method is highly dependent on the circuit structure. Table 3.2 summarizes the runtime and average number of waveforms per node for the ISCAS’85 benchmark circuits. An Intel Xeon processor 3GHz is used for computations. These experiments show that the product of the total number of nodes and the average logic depth can in general give a relatively good estimation of the runtime of the power estimation algorithm.

The memory requirement can grow for large circuits due to the large mask tags and large number of waveforms. However, the memory size can be suppressed significantly by dynamically removing the SWSs for the nodes that are not required for future computations. This reduces the maximum required memory for the C6288 circuit from about 2.1GB to about 200MB.

3.1.2 High-Level Power Estimation

In some situations fast power estimators are required. An example of such situations will be discussed in Section 4.2 where the power estimator is particularly

utilized within an optimization loop for estimating the power consumption of the different PPRTs. A power macro model based high-level power estimator can be utilized in these situations. Such power estimation techniques characterize every component in the high-level design library by simulating it under pseudo random data and fitting a multi-variable regression curve (i.e. the power macro-model equation) to the power dissipation results using a least mean square error fit. The macro-model equation is then used to estimate the power for random logic [8, 97, 144, 147]. The power estimator estimates the transition densities of each node using the static probabilities and transition densities of the preceding nodes. This estimation of transition densities is performed using data interpolation of the estimations from the Synopsys Power-Compiler for elementary gates that are used in the PPRT circuits, i.e. XOR, NAND and AND gates.

The inputs to Power-Compiler are static probabilities and transition densities of circuit's primary inputs. According to Power-Compiler's documentation, Power-Compiler estimates the power statistically using random simulation vectors that satisfies the given static-probabilities and transition densities [177]. The zero delay simulator uses the functionality of the design cells and the random vectors to obtain the switching activity on unannotated cell outputs.

For AND, NAND and XOR gates, a large number of estimations have been performed using Power-Compiler for different values of static-probabilities and transition densities. The target technology used in these experiments is a typical $0.35\mu\text{m}$ CMOS library. Information about the load capacitances is also extracted for the target technology.

The estimation results from Power-Compiler have been used in the development of a faster estimator, which estimates the power by interpolation of the collected data for each gate and estimating the transition activities at the gate outputs. The transition densities are propagated through the circuit from primary inputs to the outputs. The experimental results using this power estimator is reported and compared in Section 4.2.

3.2 Static Power Estimation

The amount of leakage current in a logic gate strongly depends on its input values [67, 135]. Figure 3.11 illustrates a static CMOS 2-input NAND gate. The total leakage current shows large variations when the input values vary (Figure 3.11.a-d). In Table 3.3 the estimates of leakage currents for a 2-input NAND gate are given in a 65nm CMOS library. The estimates are computed using SPICE mod-

Table 3.3: The normalized leakage current for the logic gates

Input Values		NAND	AND	XOR
L	L	1	5.3	17.9
L	H	5.9	10.2	17.9
H	L	7.1	11.4	9.1
H	H	4.5	14.5	9.1

els of a low power 65nm process with standard threshold transistors. Standard cells with driving force of one are used. The sum of gate leakage, reverse biased diode leakage and sub-threshold leakage is extracted for different states. It is observed that the total leakage current is dominated by the sub-threshold leakage, followed by the reverse bias diode leakage and the gate leakage. The values are normalized and are for room temperature ($T = 27^\circ C$). From Table 3.3, the ratio between maximum total leakage current and minimum total leakage currents for a NAND, an AND and an XOR gate are about 7.1, 2.7 and 2 respectively. These differences have earlier been used to reduce the leakage current through input vector control [1, 3, 67, 85]. If the values of leakage currents for the logic gates are known and the inputs are specified, computing the leakage power is simple. The total leakage current can be approximated using sum of the gate leakage currents. This can be generalized to build a probabilistic static power estimator, where instead of using exact values for the different gate inputs, the corresponding one-probabilities are used. The static power estimation is obtained using the lookup-tables (Table 3.3) for the basic logic cells (AND, NAND and XOR gates). Let us assume a gate with inputs X and Y . The static power for this gate will be

$$P_{static} = \sum_{x \in \{0,1\}} \sum_{y \in \{0,1\}} p(X = x \wedge Y = y) \text{LUT}(x, y) \quad (3.15)$$

where x and y are logic values, and $\text{LUT}(x, y)$ is the value from the lookup-table for x and y as inputs. The joint probability of the events $X = x$ and $Y = y$ is denoted by $p(X = x \wedge Y = y)$. Therefore, as a part of a probabilistic static power estimator, a static probability estimator needs to be implemented.

As discussed in Section 3.1.1, various methods exist for computing the static probabilities in a combinational logic network [26, 52, 95, 142]. Methods based on Bayesian networks [37] and ordered binary decision diagrams (OBDDs) [21]

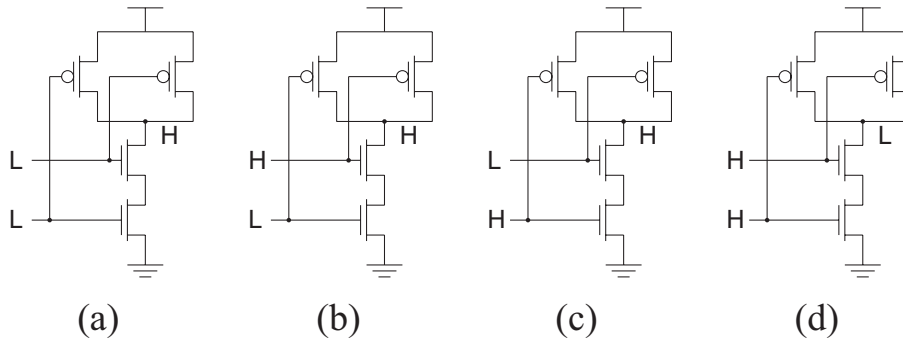


Figure 3.11: A NAND gate with possible input values

have become widespread in recent years and they provide efficient computational procedures for exact signal probability estimation. In order to estimate the static probabilities the method in [52] is utilized because it can easily encapsulate the spatial correlation between primary inputs. This method is summarized in Appendix A. When spatial correlations are present between primary inputs, they can also be included in the form of correlation coefficients between primary inputs.

The static power estimator presented in this section is implemented in C++ and it will be utilized in the optimization algorithm introduced in Chapter 6. Chapter 6 also presents the experiments on multipliers using this power estimator. As briefly discussed in Chapter 1.1, the total power consumption of a digital circuit consists of dynamic power consumption and static power consumption. The estimation methods introduced in Section 3.1 and Section 3.2 can be employed concurrently to estimate the total power consumption of the circuit. This will be discussed and experimented further in Chapter 6.2.

Chapter 4

Transition-Activity Aware Design of Reduction-Trees

This chapter presents two new techniques for designing the reduction-trees. First, the construction of the PPRT and the possible flexibilities in the PPRT structure are discussed in Section 4.1. The optimization methods will be discussed in Sections 4.2 and 4.3.

4.1 Construction of the Reduction Tree

The full-adder based PPRT is in fact a multi-operand adder with a large number of inputs. The inputs are bit-products with different weights, conveniently represented in matrix form (the partial product matrices \mathbb{M}_i). It is assumed that pointers to the actual location of the PPs are also embedded in the PPMs; i.e., from a 1 in the PPM it is possible to track which PP it is corresponding to. The left-top corner of these matrices corresponds to the LSB bit. A pseudo-code for generating the multiplier is given in Figure 4.1. A C++ and MATLAB program with this behavior has been implemented as a part of the work behind this thesis. First, \mathbb{M}_0 is generated using a suitable PPG method, as discussed in Chapter 2. Then, using a suitable reduction scheme (for example Wallace, Dadda or modified Wallace/Dadda) the number of full-adders (FAs) and half-adders (HAs) are specified for the current stage. The reduction scheme specifies a number of FAs/HAs to be connected to the PPs in \mathbb{M}_i . All inputs of a FA (or HA) have equivalent weight; i.e., all inputs are selected from a unique column in \mathbb{M}_i . The order of this column is referred to as the order of the FA (or HA). The call-routine `reduction_scheme()`,

```

generate_multiplier(operand sizes)
{
  TheMultiplier=0;
  M0=generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  i=0;
  while (||Mi|| > 2)
  {
    [Fi, Hi]=reduction_scheme(Mi);
    Mi+1=apply_FA_HA(Mi, Fi, Hi);
    TheMultiplier.append(Mi, Fi, Hi);
    i=i+1;
  }
  TheMultiplier.append(final_CPA(Mi));
}

```

Figure 4.1: The multiplier generation algorithm

takes the current PPM, \mathbb{M}_i , and returns two vectors, \mathbb{F}_i and \mathbb{H}_i , which contain the number of FAs and HAs required in the current stage. The j :th element in the vector \mathbb{F}_i is the number of FAs with order j in the i :th stage. Similarly, the j :th element in the vector \mathbb{H}_i is the number of HAs with order j in the i :th stage. The call-routine `apply_FA_HA()` applies the FAs and HAs specified by \mathbb{F}_i and \mathbb{H}_i to \mathbb{M}_i and generates the PPM for the next stage (\mathbb{M}_{i+1}). Each FA with order j removes three PPs from column j in \mathbb{M}_i and places two new PPs in j :th and $(j+1)$:th column of \mathbb{M}_{i+1} . The new PPs in columns j and $j+1$ correspond to the FA's S and C_{out} outputs, respectively. Similarly, each HA with order j removes two PPs from column j in \mathbb{M}_i and places two new PPs in j :th and $(j+1)$:th column of \mathbb{M}_{i+1} for S and C_{out} outputs, respectively. The reduction continues until the maximum height of the current PPM, $\|\mathbb{M}_i\|$, is equal to 2. The reduction tree is followed by the final CPA which produces the final multiplication product. Figure 4.2 gives the intermediate matrices and vectors for an unsigned 12×12 -bit multiplier. The modified Wallace/Dadda's reduction scheme is utilized.

The reduction scheme gives the number of FAs/HAs and their order but it does not specify the order of the inputs to the FAs. If column j in \mathbb{M}_i has m_j bits, and there are f_j FAs and h_j HAs with the order j , then the number of possible ways the inputs to FAs and HAs can be connected is:

$$P(m_j, f_j, h_j) = \frac{m_j!}{(m_j - 3f_j - 2h_j)! f_j! h_j!} \quad (4.1)$$

If a FA is assumed to treat all three inputs equally and a HA is assumed to treat

$$\begin{array}{l}
\mathbb{M}_0 : \begin{bmatrix} 11111111111111111111 \\ 01111111111111111110 \\ 001111111111111111100 \\ 0001111111111111111000 \\ 00001111111111111110000 \\ 000001111111111111100000 \\ 00000011111111111000000 \\ 0000000111111110000000 \\ 00000000111111100000000 \\ 000000000111111000000000 \\ 000000000011111000000000 \\ 0000000000010000000000 \end{bmatrix} & \mathbb{F}_0 : [00111222333433322211100] \\
& \mathbb{H}_0 : [010000000000000000000] \\
\\
\mathbb{M}_1 : \begin{bmatrix} 11111111111111111111 \\ 001111111111111111110 \\ 0001111111111111111010 \\ 000010111111111110000 \\ 0000001111111111010000 \\ 0000000101111110000000 \\ 00000000011111010000000 \\ 0000000000101000000000 \\ 0000000000010000000000 \end{bmatrix} & \mathbb{F}_1 : [00011112122232221111010] \\
& \mathbb{H}_1 : [001000000000000000000] \\
\\
\mathbb{M}_2 : \begin{bmatrix} 11111111111111111111 \\ 000111111111111111101 \\ 000010111111111110100 \\ 0000001011111111010000 \\ 00000000101111101000000 \\ 0000000001001000000000 \end{bmatrix} & \mathbb{F}_2 : [00001011112112111110100] \\
& \mathbb{H}_2 : [0001000000011000000000] \\
\\
\mathbb{M}_3 : \begin{bmatrix} 11111111111111111111 \\ 0000111111111111111011 \\ 0000010011111111011000 \\ 00000000100111111000000 \end{bmatrix} & \mathbb{F}_3 : [0000010011111111011000] \\
& \mathbb{H}_3 : [000010000000000000000] \\
\\
\mathbb{M}_4 : \begin{bmatrix} 11111111111111111111 \\ 0000011111111111101111 \\ 00000010000111111100000 \end{bmatrix} & \mathbb{F}_4 : [00000010000111111100000] \\
& \mathbb{H}_4 : [0000010111100000000000] \\
\\
\mathbb{M}_5 : \begin{bmatrix} 11111111111111111111 \\ 0000001111111111111111 \end{bmatrix}
\end{array}$$

Figure 4.2: The PPM, FA and HA matrices for a 12×12 -bit unsigned multiplier

both inputs equally, then the number of possible interconnections:

$$C(m_j, f_j, h_j) = \frac{m_j!}{(m_j - 3f_j - 2h_j)!f_j!h_j!6^{f_j}2^{h_j}} \quad (4.2)$$

Let us consider the the FA and HA implementations in Figure 2.10. If AND, XOR and NAND gates are assumed to be symmetric with respect to their inputs, then HA and FA will be symmetric for inputs A and B . However, the FA is asymmetric for input C_{in} . With these assumptions, the number of possible interconnections will be:

$$S(m_j, f_j, h_j) = \frac{m_j!}{(m_j - 3f_j - 2h_j)!f_j!h_j!2^{f_j+h_j}} \quad (4.3)$$

Indeed, interchanging the PPs that belong to one column of the PPM does not change the functionality of the structure. The commutativity and associativity properties of the addition allow any permutation of the PPs with equal weight to be connected to the adder's inputs. Although all permutations of the PPs result in the same final output, they may differ in other properties of the circuit such as the computation delay, the dynamic power dissipation, the leakage currents, and so on. The freedom of choosing among these possibilities can therefore be used to reduce the computation delay, the dynamic power, and the total leakage current.

In this chapter two methods for optimizing the PPRT with respect to the dynamic power consumption are examined. In the first method, the complete reduction tree is constructed in the beginning of the optimization procedure. Then the optimization is performed on the complete PPRT. In the second method, however, the reduction tree is progressively designed such that at the end a low-power PPRT is achieved.

4.2 Method 1: Optimization of Complete Reduction Tree

In the first attempt for optimizing the PPRT, the optimization of the complete PPRT is proposed. Most of the material in this section is published in [181]. The proposed optimization algorithm is summarized in Figure 4.3. A high-level power estimator developed in MATLAB, `estimate_power()`, estimates the transition activities at internal nodes based on the primary input static probabilities. This power estimator is described in Section 4.2.1. The complete PPRT is initially

```

optimized_multiplier(input static probabilities)
{
  generate_multiplier(operand sizes);
  P_min=estimate_power(TheMultiplier, input static probabilities);
  i=0;
  while ((||M_i|| > 2)
  {
    for current_col=1 to number of columns in M_i do
    {
      possible_permutations=list of all useful permutations of the PPs in current_col of M_i;
      current_perm= the current permutation of the PPs in current_col of M_i;
      for all m ∈ possible_permutations
      {
        reorder_interconnects(TheMultiplier, M_i, current_col, m);
        P=estimate_power(TheMultiplier, input static probabilities);
        if (P < P_min)
        {
          P_min=P;
          current_perm=m;
        } else
        {
          reorder_interconnects(TheMultiplier, M_i, current_col, current_perm);
        }
      }
    }
    i=i+1;
  }
}

```

Figure 4.3: The multiplier optimization algorithm - Method 1

constructed using random interconnection permutations. In this step all intermediate PPMs (\mathbb{M}_i s), FA assignment vectors (\mathbb{F}_i s) and HA assignment vectors (\mathbb{H}_i s) are computed. A brute force approach to find the optimal solution is to perform an exhaustive search of all implementation alternatives and estimate their power consumption. As demonstrated in Table 4.1, this is obviously impossible because of the huge number of possibilities. Therefore utilizing simpler methods and heuristics are inevitable. Estimated transition density values from the power estimator can be used to focus the search at the interesting parts of the solution space. As defined in Eq. 1.6, the transition density at node i is :

$$D_i = \lim_{T \rightarrow \infty} \frac{n_i(T)}{T}$$

where $n_i(T)$ is the number of transitions at node i in a time interval of length T . Based on the experiments with the PPRT, in order to limit the number of search alternatives, the following considerations are applied in the optimization method. The experimental results show that, even with limiting the search alternatives, significant reduction in power consumption is achieved.

The search is limited to only one column in one stage at a time. The ex-

periments show that performing the optimization of columns from LSB to MSB and from first stage to last stage gives better results. The explanation can be that the generated carry bits from adders propagate from LSB towards MSB. The optimization from LSB to MSB and from first stage to last stage ensures that the optimized parts of the circuit are not altered when later optimizations are being performed.

It is assumed that optimization of columns in early stages of the reduction can be performed without taking the organization of interconnection in later stages into account. This localization of search might in general result in suboptimal structures because of the fact that the structure of later stages may influence the optimal solution for earlier stages. However, this degradation from the optimal solution reduces the search space dramatically. Localization of the search space has been employed in numerous optimization methods, especially in optimization algorithms for hardware design [57]. The observations below are provided under this assumption of independence between columns.

Glitches and spurious transitions spread in the reduction stage after a few layers of combinational logic. To avoid them is not feasible in most cases. Therefore it seems beneficial to assign partial products having high switching activity a short path and thus perform addition of bits with lower switching activities in the earlier stages of the reduction tree and keep the ones with higher switching activity for the later stages. The partial products first added will affect larger parts of the reduction tree.

The experiments in this section, as well as other work that has employed the above mentioned assumption [201], show that by sorting the PPs based on their transition densities and assigning bits to full-adders and half-adders in this order, a noticeable power saving can be achieved.

As discussed earlier in this chapter, the inputs to a FA are asymmetric. The path of the carry-in input of a FA (C_{in} in Figure 2.10) is shorter than the other two inputs. A transition on this input will therefore result in less activity. The input with the highest transition density among the three inputs of the FA should therefore be connected to the C_{in} input. For a FA with inputs (A , B , C_{in}), $\max(D_A, D_B) < D_{C_{in}}$. It is assumed that the FA is symmetric with respect to inputs A and B .

In a FA, a transition in input C_{in} affects only one XOR gate while a transition in A or B affects two XOR gates to reach the S output. This will minimize the power consumption of such an addition under the single column assumption.

The transition densities of the outputs of FAs are in general close to the largest input transition density. Thus the transition densities of the three inputs should

Table 4.1: Number of search alternatives in one column

Number of bits	Exhaustive search	Reduced search
3	6	1
6	360	3
9	6.048×10^4	12
12	1.996×10^7	55
18	8.892×10^{12}	1428
24	1.539×10^{19}	4.326×10^4

be as similar as possible in order to minimize the overall transition activity of the multiplier. For any two FAs x and y with inputs $(x_A, x_B, x_{C_{in}})$ and $(y_A, y_B, y_{C_{in}})$ respectively, the following property should hold. If $D_{x_{C_{in}}} < D_{y_{C_{in}}}$ then $\max(D_{x_B}, D_{x_A}) < \min(D_{y_B}, D_{y_A})$.

From the above assumptions, it can be easily inferred that if $D_{x_{C_{in}}} < D_{y_{C_{in}}}$, then D_{x_B} and D_{x_A} are smaller than $D_{x_{C_{in}}}$, D_{y_A} , D_{y_B} and $D_{y_{C_{in}}}$.

Using these four assumptions, it is possible to prune away a large number of uninteresting solutions. Table 4.1 compared the total number of search alternatives (Eq. 4.1) and the number of permutations after applying the four above-mentioned assumptions. Although the numbers of search alternatives are still factorially growing, the growth rate is much slower. Note that the numbers of alternatives for the exhaustive search are obtained with independence assumption between different columns and stages of the PPRT (the first assumption). Without the independence assumption between different columns and stages of the PPRT, the numbers of alternatives for the exhaustive search will be much bigger.

Figure 4.4 gives an example where 7 bits with the same order of magnitude are to be added. It is taken from the second stage in a 12×12 -bit multiplier in Figure 2.14. According to the rules of designing the reduction tree, 2 FAs are needed and one bit will be kept for the next stage. Using the observations in this section, the bit with the highest transition density will be kept for the next stages and the other bits are assigned to the FAs. The three interesting permutations of these seven bits that will be examined in the optimization algorithm are given in Table 4.2. For comparison, a full search of these permutations would require examination of 2520 alternatives. The ordered transition density D_i^* s in Table 4.2 are found through a sorting of the transition densities D_i s of the 7 bits, as illustrated in Figure 4.4.

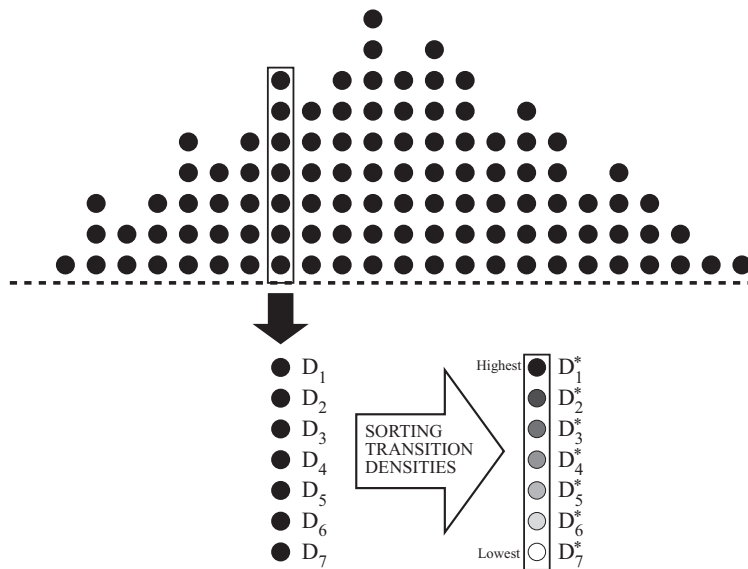


Figure 4.4: Sorting PPs based on their estimated transition densities

Table 4.2: The search alternatives of a 7-bit column

#	To next stage	Full-adder 1			Full-adder 2		
		A	B	C_{in}	A	B	C_{in}
1	D_1^*	D_4^*	D_3^*	D_2^*	D_7^*	D_6^*	D_5^*
2	D_1^*	D_5^*	D_3^*	D_2^*	D_7^*	D_6^*	D_4^*
3	D_1^*	D_5^*	D_4^*	D_2^*	D_7^*	D_6^*	D_3^*


```

worst_case_multiplier(input static probabilities)
{
  generate_multiplier(operand sizes);
  Pmax=estimate_power(TheMultiplier, input static probabilities);
  i=0;
  while (||Mi|| > 2)
  {
    for current_col=1 to number of columns in Mi do
    {
      possible_permutations=list of all useful permutations of the PPs in current_col of Mi;
      current_perm= the current permutation of the PPs in current_col of Mi;
      for all m ∈ possible_permutations
      {
        reorder_interconnects(TheMultiplier, Mi, current_col, m);
        P=estimate_power(TheMultiplier, input static probabilities);
        if (P > Pmax)
        {
          Pmax=P;
          current_perm=m;
        } else
        {
          reorder_interconnects(TheMultiplier, Mi, current_col, current_perm);
        }
      }
    }
    i=i+1;
  }
}

```

Figure 4.5: Worst-case multiplier algorithm - Method 1

4.2.1 Power Estimation

After assigning a new configuration set to the bits of a particular column, the power consumption of these configurations is needed. The power estimation is placed in the core of search loops (Figure 4.3) and the power estimation is performed for the complete PPRT. Therefore the estimation time is crucial and the power estimator needs to be fast. The power macro model based high-level power estimator introduced in Section 3.1.2 is utilized as `estimate_power()`. This estimator runs on the MathWorks MATLAB platform at a speed about 80 times faster than Power-Compiler. Using a Pentium IV 1.4GHz machine one power estimation run for a 12×12 multiplier is performed in 0.09 second with this tool while it takes 8 seconds using Power-Compiler. For a 24×24 multiplier the times are 0.29 and 31 seconds respectively. The run-time differences between the tools are important since the estimates are performed a large number of times during an execution of the optimization algorithm.

4.2.2 Notation of the multipliers

In order to demonstrate the effect of the optimization, the optimized structure generated by algorithm `optimize_multiplier()` are compared with the worst case structure generated by another algorithm similar to `optimize_multiplier()` but aiming at getting the maximum possible switching activity when rearranging the interconnections in the reduction stages (algorithm `worst_case_multiplier()` shown in Figure 4.5). A random multiplier version, achieved if transition activities are not taken into account, could fall anywhere in between. As shown in Table 4.3, experiments indicate that it will typically lie near the middle. This is also intuitively correct, since it is not likely that all of the many random interconnects will be optimal or that all will be worst case.

The optimized, worst-case and random multipliers must be differentiated in the results. In addition, two optimized multipliers, resulted from different input static probabilities, may differ in the structure and therefore must be differentiated. The notation introduced here will be also used in the subsequent chapters. A multiplier will be represented as:

$$\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{Algorithm} \quad Algorithm \rightarrow \begin{cases} OPT \\ RND \\ WC \\ SORT \end{cases}$$

where *Algorithm* is the algorithm used for generating the multiplier and \mathbf{p}_A and \mathbf{p}_B are the static probability vectors of the inputs *A* and *B*, respectively. The *i*:th element of the vector \mathbf{p}_A represents the one-probability of the *i*:th bit of the input *A* (i.e., the one-probability of a_i). The *Algorithm* can be *OPT*, *WC*, *SORT* and *RND*. The multipliers generated by `optimize_multiplier()` and `worst_case_multiplier()` are denoted as \mathfrak{M}^{OPT} and \mathfrak{M}^{WC} respectively. A multiplier denoted as \mathfrak{M}^{SORT} is obtained from sorting the PPs by their transition densities and assigning the bits with low transition densities earlier in the PPRT (similar to the approach in [201]). No other interconnect optimization is performed on \mathfrak{M}^{SORT} . In order to have a fair comparison of the introduced algorithms, random multipliers are generated where the interconnects are chosen randomly. These multipliers are denoted as \mathfrak{M}^{RND} .

The dynamic power consumption of a multiplier is a function of input static probabilities applied to its operands while it is operating. This is the case for all multipliers; whether the multiplier is optimized for the inputs that are applied to it

or not. The energy consumption per operation for a multiplier is denoted as:

$$\mathbf{E} \left(\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{Algorithm}, \mathbf{p}'_A, \mathbf{p}'_B \right)$$

where \mathbf{p}'_A and \mathbf{p}'_B are the static probability vectors of the inputs applied to the multiplier $\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{Algorithm}$.

A uniform random binary bit-vector with no temporal or spatial correlation, where all bits have 0.5 one-probabilities, is the simplest form of the input static probability vector. All elements of this vector are equal to 0.5. This vector is denoted as Ω_n^n where n is the number of bits in the vector. To generalize this representation, a number of bits from LSB are assumed to be inactive; i.e. forced to zero. This situation can happen in systems where the resolution of signals varies dynamically. Ω_i^n represents an input static probability vector where i bits from the MSB side are active and $n - i$ bits from LSB side are forced to zero. If the static probability of a 6-bits vector $A_{0..5}$ is represented by Ω_4^6 , it means that input bits A_0 and A_1 have 0 one-probabilities and input bits A_2, A_3, A_4 and A_5 have 0.5 one-probabilities. That is

$$\Omega_4^6 = \begin{array}{cccccc} & \text{MSB} & & & & \text{LSB} \\ = & [0.5 & 0.5 & 0.5 & 0.5 & 0 & 0] \end{array} \quad (4.4)$$

As an example, $\mathbf{E} \left(\mathfrak{M}_{\Omega_{10}^{12}, \Omega_{10}^{12}}^{OPT}, \Omega_{12}^{12}, \Omega_{12}^{12} \right)$ is the energy per operation for a multiplier that is optimized for Ω_{10}^{12} as input static probability vectors but it is operating at Ω_{12}^{12} as input static probability vectors.

4.2.3 Experiments

The optimization algorithm, the power estimator, and a gate-level VHDL code generator for the optimized structure are implemented in MATLAB. Inputs to the optimization program are the static probabilities of the input bits. The output is a multiplier which is generated according to the *OPT*, *WC*, *SORT* or *RND* algorithms. To evaluate the quality of optimization exact power estimates are performed for the generated multipliers, using switching activity back-annotation; i.e., the transition density of every single node in the designed multiplier is computed by VHDL simulations in ModelSim using large number of input vectors with the desired input static probabilities. These transition densities are fed into Synopsys Power-Compiler and are utilized for power estimation. A $0.35\mu\text{m}$ -3.3V CMOS library is used for VHDL simulation in ModelSim as well as synthesis and

Table 4.3: Estimated energy per operation for different multipliers [pJ]

		$\mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{OPT}$	$\mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{WC}$	$\overline{\mathfrak{M}}^{RND}$	$\mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{SORT}$
Ω_{12}^{12} set as input static probabilities of both operands	Power-Compiler with switching activity back-annotation from ModelSim	98.04	105.14	102.23	100.54
	Power-Compiler with internal switching activity estimation (without back-annotation)	102.40	108.18	105.97	104.31
	The power estimation tool in MATLAB	102.26	108.28	106.06	104.52
Ω_6^{12} set as input static probabilities of both operands	Power-Compiler with switching activity back-annotation from ModelSim	28.26	33.94	31.15	29.56
	Power-Compiler with internal switching activity estimation (without back-annotation)	29.19	34.78	32.70	30.81
	The power estimation tool in MATLAB	29.23	34.90	32.67	30.89

Table 4.4: Estimated energy per operation for optimized multipliers [pJ]

Values of $\mathbf{E} \left(\mathfrak{M}_{\Omega_i^{12}, \Omega_i^{12}, \Omega_j^{12}, \Omega_j^{12}}^{OPT} \right)$ for different i and j values

		j						
		12	11	10	9	8	7	6
i	12	<u>98.04</u>	87.70	74.07	61.24	49.90	37.87	28.26
	11	98.32	<u>87.68</u>	73.90	60.69	49.60	38.04	28.26
	10	98.29	87.97	<u>73.65</u>	59.67	48.68	37.60	28.26
	9	98.96	88.83	74.40	<u>59.14</u>	48.33	37.07	28.14
	8	98.65	88.55	74.55	59.91	<u>48.28</u>	36.74	27.65
	7	98.63	88.59	74.94	60.85	49.35	<u>36.57</u>	27.10
	6	98.66	88.69	75.14	61.73	50.03	37.34	<u>26.90</u>

power estimation in Design-Compiler and Power-Compiler. ModelSim simulations consider the realistic delays provided from the library and therefore, propagation of glitches and spurious transitions are more accurately taken into account.

Energy is a measure of the total number of Joules dissipated by a circuit, whereas power refers to the energy or number of Joules dissipated by a circuit over a certain period of time. Properly speaking, power reduction is a different goal than energy reduction. However, as the timing characteristics of the circuit are not subject to change in this thesis, these two terms, i.e., energy reduction and power reduction, might be used to refer to the same goal. In the reported results, energy per operation is used as the metric that removes the time factor. Energy per operation is the amount of energy needed to complete one multiplication operation.

In the first part of the experiments 12×12 -bit multipliers are considered. The input operands of the multiplier are assumed to be uniform random binary bit-vectors with no temporal or spatial correlation, where all bits have 0.5 one-probabilities. A number of LSB bits are allowed to be inactive. This situation can happen in systems where the resolution of signals varies dynamically. In Table 4.3 the energy per operation numbers are given for optimum multipliers and worst-case multipliers. Three values for energy per operation are provided, from Power-Compiler with switching activity back-annotation provided by ModelSim VHDL simulations with realistic delay-models, from Power-Compiler with internal switching activity estimation (without back-annotation), and from the estimation tool in MATLAB. Table 4.3 and 4.4 provide the energy per operation estimations with accurate transition densities (with switching activity back-annotation) for 12×12 -bit multipliers optimized for different number of active bits. With Ω_6^{12} applied to both operands, the multiplier optimized for this, $\mathfrak{M}_{\Omega_6^{12}, \Omega_6^{12}}^{OPT}$, consumes 4.8% less energy than $\mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{OPT}$. $\mathfrak{M}_{\Omega_6^{12}, \Omega_6^{12}}^{OPT}$ consumes about 16% less energy compared to $\mathfrak{M}_{\Omega_6^{12}, \Omega_6^{12}}^{WC}$ when Ω_6^{12} is applied to inputs. \mathfrak{M}^{WC} is an extreme case that is unlikely to happen. In order to have fair comparison of the optimization algorithm 10 random multipliers are generated where the interconnects are chosen randomly. The averaged energy per operation numbers for these multipliers are reported in Table 4.3 denoted as $\overline{\mathfrak{M}^{RND}}$. $\mathfrak{M}_{\Omega_6^{12}, \Omega_6^{12}}^{OPT}$ consumes about 13.6% less energy compared to $\overline{\mathfrak{M}^{RND}}$ with Ω_6^{12} as inputs. From this it can be concluded that it may be beneficial to optimize the multiplier to the number of bits that are active most of the time. The situations where the signal resolution varies dynamically will be analyzed in Chapter 7.

The static probabilities of the input operands in a multiplier depend strongly

on the nature of input signals. For many natural signals the static probability of the MSB-part is smaller than that of the LSB-part. In these cases, it can be beneficial to optimize the multiplier structure for the given pattern of input static probabilities. In the second part of the experiments 12×12 -bit multipliers with log-normal input distributions are considered.

The log-normal distribution is associated to any random variable whose logarithm is normally distributed, which is very common for natural signals. It can model any variable that can be thought of as the multiplicative product of many small independent identically-distributed factors. Its probability density function is given by:

$$f_X(\mathcal{X}) = \frac{1}{\mathcal{X}\sigma\sqrt{2\pi}} e^{-\frac{(\ln \mathcal{X} - \mu)^2}{2\sigma^2}} \quad \mathcal{X} > 0 \quad (4.5)$$

where μ and σ are the mean and standard deviation of the normal distribution (logarithm of the random variable), respectively. A log-normal random variable can be very large; although the probability of large numbers are very small. Therefore depending on the tolerable error in the system the word-length of such variables are defined. Three values $\ln 2^5$, $\ln 2^6$, and $\ln 2^7$ for μ are chosen in the experiments. σ is chosen to be one for all experiments. The distribution function for a random variable with log-normal distribution is depicted in Figure 4.6. Figure 4.7 illustrates the one-probabilities of input bits assuming these values for μ and σ . Both operands of the multiplier have equal probability functions. In the results Λ_i^n represents an input probability vector where a log-normal variable with $\mu = \ln 2^i$ and $\sigma = 1$ is quantized using n bits; i.e. the input static probability vectors Λ_5^{12} , Λ_6^{12} and Λ_7^{12} , denote the input static probabilities of a 12-bit vector associated with log-normal distributions where $\mu = \ln 2^5$, $\mu = \ln 2^6$ and $\mu = \ln 2^7$ respectively. Table 4.5 summarizes estimated energy per operations for these multipliers with different input static probabilities. When log-normally distributed inputs with Λ_5^{12} are applied to the multipliers, the $\mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{OPT}$ consumes 4.8% more energy than $\mathfrak{M}_{\Lambda_5^{12}, \Lambda_5^{12}}^{OPT}$. $\mathfrak{M}_{\Lambda_5^{12}, \Lambda_5^{12}}^{OPT}$ requires 14.8% less energy than the worst case multiplier $\mathfrak{M}_{\Lambda_5^{12}, \Lambda_5^{12}}^{WC}$ in this case.

From Table 4.5 it can be observed that significant parts of the power reduction in the optimal multiplier appears regardless of input static probabilities. The rest depends on input static probabilities. According to Synopsys Design-Compiler's area estimation, the circuit and routing area of the generated multipliers are the same. This is because the difference between the generated multipliers is only in the interconnects of the adders. The required standard cells are exactly the same for all of the architectures. The speed of the energy-optimized multiplier is

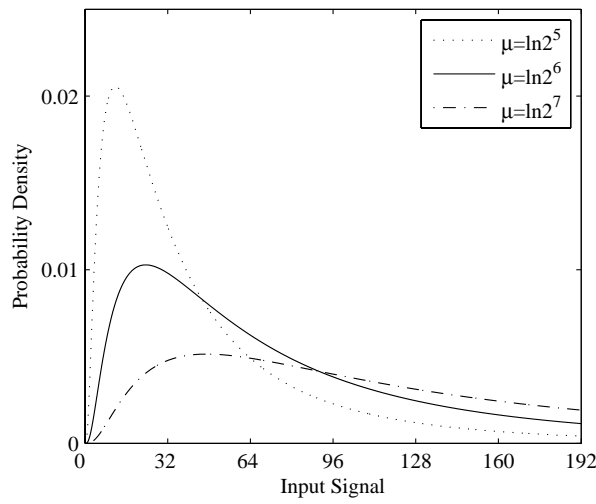


Figure 4.6: Log-normal signal distribution

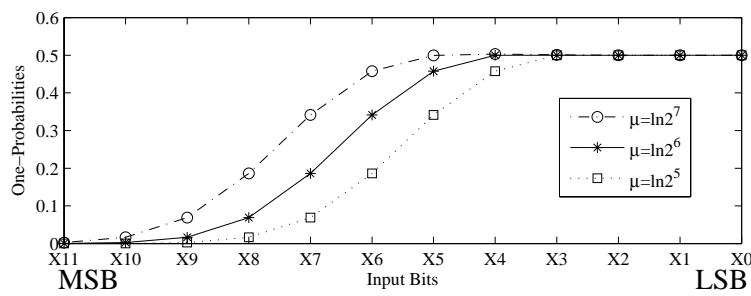


Figure 4.7: One-probabilities of input bits with log-normal distribution
 Input signals for both operands have log-normal distribution
 with $\mu = \ln 2^5$, $\mu = \ln 2^6$ and $\mu = \ln 2^7$ ($\sigma = 1$)

Table 4.5: Estimated energy per operation for multipliers with log-normal inputs

Values of $E(\mathfrak{M}, \mathbf{p}, \mathbf{p})$ for different multipliers (\mathfrak{M}) and different input static probability vectors (\mathbf{p}). Values are in [pJ].

	$\mathbf{p} = \Omega_{12}^{12}$	$\mathbf{p} = \Lambda_7^{12}$	$\mathbf{p} = \Lambda_6^{12}$	$\mathbf{p} = \Lambda_5^{12}$
$\mathfrak{M} = \mathfrak{M}_{\Omega_{12}^{12}, \Omega_{12}^{12}}^{OPT}$	98.04	56.02	44.09	33.82
$\mathfrak{M} = \mathfrak{M}_{\Lambda_7^{12}, \Lambda_7^{12}}^{OPT}$	98.54	54.23	42.95	32.84
$\mathfrak{M} = \mathfrak{M}_{\Lambda_6^{12}, \Lambda_6^{12}}^{OPT}$	98.62	54.70	42.60	32.71
$\mathfrak{M} = \mathfrak{M}_{\Lambda_5^{12}, \Lambda_5^{12}}^{OPT}$	98.79	54.82	42.98	32.27
$\mathfrak{M} = \mathfrak{M}_{\Lambda_5^{12}, \Lambda_5^{12}}^{WC}$	104.52	61.81	48.97	37.86

slightly lower than the average version. Using Design-Compiler, however, performance penalties of more than 1% for any of the generated optimized circuits, compared to average performance of the randomly generated multipliers, have not been observed.

4.3 Method 2: Progressive Design of Reduction Tree

Following method 1 for optimizing the PPRT structure based on the input characteristic information, a second approach is proposed. The objective is to reduce the circuit size that is considered for power estimation. With smaller circuits, it is possible to increase the accuracy of power estimation. As discussed in the previous section, in the first method, power estimation is performed for the complete PPRT, which can be time-consuming. Therefore, large number of iterations cannot be afforded timewise. In the second approach, the power estimation is limited to a small portion of the complete circuit. Therefore larger number of iterations can be afforded, giving better results. The material of this chapter are mostly published in [180, 181]. In this approach, design of the PPRT is combined with the optimization phase. The power estimation and PPRT optimization algorithm is integrated in the algorithm for designing the PPRT shown in Figure 4.1. Therefore, the optimization is merely performed on the part of the PPRT which is constructed. This type of optimization is called progressive design of the PPRT in this thesis.

Figure 4.8 summarizes the progressive multiplier design algorithm. As can be seen, the multiplier generation step is merged with the optimization phase in


```

optimized_multiplier(input static probabilities)
{
  TheMultiplier=0;
  M0=generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  i=0;
  while (||Mi|| > 2)
  {
    [Fi, Hi]=reduction_scheme(Mi);
    Mi+1=apply_FA_HA(Mi, Fi, Hi);
    for current_col=1 to number of columns in Mi do
    {
      current_perm=random permutation of the PPs in current_col of Mi;
      Pmin=+∞;
      α=α0;
      for current_iteration=1 to number_of_iterations do
      {
        old_perm=current_perm;
        swap two random positions in current_perm;
        P=estimate_power(Fi, Hi, current_perm);
        if (P < Pmin)
          Pmin=P;
        else
        {
          r=uniform random (r ∈ [0, 1));
          if (r < e $\frac{P_{min}-P}{\alpha}$ )
            Pmin=P;
          else
            current_perm=old_perm;
        }
        α= $\frac{\alpha}{\tau}$ ;
      }
      reorder current_col of Mi as current_perm;
    }
    TheMultiplier.append(Mi, Fi, Hi);
    i=i+1;
  }
  TheMultiplier.append(final_CPA(Mi));
}

```

Figure 4.8: The multiplier optimization algorithm - Method 2

method 2. The search method is also different from method 1. Method 1 performs the search on a selection of possible permutations, while method 2 performs a search based on simulated annealing. As discussed in Section 4.2, the search in method 1 is limited to a reduced space using four assumptions. Except the independence assumption between columns and stages, the other assumptions are not considered for method 2. Similar to method 1, the search is limited to only one column in one stage at a time.

Simulated annealing (SA) is a method that simulates a thermodynamic process, where a metal is heated to its melting point and then allowed to cool slowly so that its structure is frozen at the lowest energy crystal form. The slow cooling gives the atoms more chances of finding configurations with lower internal energy than the initial one. In this process, the configuration of the atoms is continuously rearranged, moving toward lowest energy. Meanwhile the atoms gradually lose their mobilities as the temperature is reduced during the process. Even though the energy function may have local minima, simulated annealing is able to go uphill occasionally and avoid local minima. Metropolis et al. in [120] developed a heuristics that simulates the cooling of a metal. However, this method is applicable to more general applications other than its original application as simulating metal cooling. Kirkpatrick et al. in [90] discussed various applications of SA. As it will be discussed later, the minimization problem in multiplier PPRT design in method 2 is solved using SA.

In order to use simulated annealing, a cooling strategy must be chosen. The cooling strategy specifies the initial temperature and how to decrease the temperature as a function of time. The cooling strategy depends on the problem being solved. Many successful SA implementations use a geometric cooling strategy with parameters obtained by experimentation. However, an ad-hoc approach may give better results for some applications. In the geometric cooling strategy, temperature is reduced geometrically; i.e. $\alpha^* = \alpha/\tau$, for some fixed τ ($\tau > 1$). α is the current temperature and α^* is the temperature at the next iteration. In addition to the cooling strategy, the probability of accepting a higher energy state must also be chosen in the SA optimizer. Although this probability could be chosen in various ways, it is usually taken as $e^{-\Delta P/\alpha}$ which is proportional to the probability of an energy change of ΔP at temperature α and comes from the original application in [120].

The progressive multiplier design algorithm in Figure 4.8 starts with initialization of the circuit to primary partial products generated by a suitable algorithm discussed in Chapter 2. After the PPs are generated, the multiplier structure is progressively designed. For each column of each stage in the PPRT, the SA op-

timization is performed. A randomly chosen permutations is the start configuration. This configuration is altered in each iteration. A new power estimation is computed for the perturbed configuration. If the move decreases the value of the estimated power, the move is accepted and the new permutation is retained. If SA gets stuck in a local optima then the Boltzmann factor is calculated, and a random number, uniformly distributed in the interval $[0,1)$, is chosen. If the random number is less than the calculated Boltzmann factor, then the new permutation is retained, otherwise, the move is discarded and the configuration before this move is used for the next step. The power estimator in this loop computes the transition densities exclusively for the current full-adder and half-adder stage, i.e. \mathbb{F}_i and \mathbb{H}_i . The temperature α is initialized to α_0 before the SA begins and then is reduced geometrically after each iteration by the cooling factor τ . The parameters α_0 , τ and *number_of_iterations* are chosen by experimentation.

Another difference between two proposed methods of optimization is the power estimation method. The probabilistic gate-level power estimation technique described in Section 3.1.1 is utilized in the progressive PPRT design.

4.3.1 Progressive PPRT Design and the SWS Power Estimator

The SWS power estimator can be easily embedded in the optimization algorithm in Figure 4.8. Figure 4.9 summarizes the progressive PPRT design using the SWS power estimator. The function `computeSWS(TheMultiplier)` computes the SWSs for the nodes of the multiplier circuit for which SWSs are not computed earlier. Note that each time a circuit portion (i.e., a new column of a given stage) is appended to `TheMultiplier`, this function is executed, computing SWSs for the new portion using the SWSs computed earlier. After the PPs are generated, the multiplier structure is progressively designed. The power estimator innermost loop computes the transition densities exclusively for the current full-adder and half-adder stage, i.e., the stage that is being designed, using the SWSs computed earlier. A minimum search mechanism based on simulated annealing is executed for the estimated power consumption. The power estimation (`estimate_power_SWS`) is performed merely for the current column of the current stage, therefore optimization at each stage is performed assuming that it will not affect the other stages. Recall that the same assumption was considered for method 1 in Figure 4.3. The new permutation of PPs and the previous permutation differ only in the position of two PPs. Therefore, the power estimation for the new permutation can be obtained by recomputing the power for the logic gates whose inputs are changed. In fact, `estimate_power_SWS()` can be limited to recompute SWSs for maximum 10 logic

```

optimized_multiplier(input static probabilities)
{
  TheMultiplier=0;
  M0=generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  computeSWS(TheMultiplier);
  i=0;
  while (||Mi|| > 2)
  {
    [Fi, Hi]=reduction_scheme(Mi);
    Mi+1=apply_FA_HA(Mi, Fi, Hi);
    for current_col=1 to number of columns in Mi do
    {
      current_perm=random permutation of the PPs in current_col of Mi;
      Pmin=+∞;
      α=α0;
      for current_iteration=1 to number_of_iterations do
      {
        old_perm=current_perm;
        swap two random positions in current_perm;
        P=estimate_power_SWS(Fi, Hi, current_perm);
        if (P < Pmin)
          Pmin=P;
        else
        {
          r=uniform random (r ∈ [0, 1));
          if (r < e $\frac{P_{min}-P}{\alpha}$ )
            Pmin=P;
          else
            current_perm=old_perm;
        }
      }
      α= $\frac{\alpha}{\tau}$ ;
    }
    reorder current_col of Mi as current_perm;
  }
  TheMultiplier.append(Mi, Fi, Hi);
  computeSWS(TheMultiplier);
  i=i+1;
}
TheMultiplier.append(final_CPA(Mi));
computeSWS(TheMultiplier);
}

```

Figure 4.9: Progressive reduction-tree design using SWS power estimator

gates for each iteration. The worst-case is when the two interchanged PPs are connected to different FAs which have 5 logic gates each (Figure 2.10).

There is a substantial difference between method 1 discussed in Section 4.2 and method 2 discussed in this section. Method 2 uses a more accurate and more complex power estimator (SWS) but only engage the estimator for one stage of full-adders or half-adders at any given time. However, method 1 uses a simpler power estimator and runs the power estimations for the complete multiplier. Compared to method 1, the power estimation for method 2 is faster even though it is more accurate and more complex. The reason is that the estimation is performed on only a small part of the multiplier (limited to maximum ten logic gates) rather than the complete multiplier. This allows running the optimization algorithm for larger number of permutations. Running the power estimator for the complete multiplier as in method 1, even for the parts that are not optimized yet, means that the effects are considered for all successive nodes. But because parts of the multiplier are not optimized yet, the actual effects will be different when those parts are optimized as well. On the other hand, running the power estimation exclusively for the current full-adders/half-adder stage means that the effects are not considered on the PPs that are not the direct outputs of the immediate successive full-adders/half-adders.

The algorithm `optimized_multiplier` shown in Figure 4.9 constructs a multiplier that is optimized for low power consumption. The notations introduced in Section 4.2.2 are used in this part as well. When reporting experimental results in the next section, multipliers generated using `optimized_multiplier` are denoted by index *OPT* (e.g., $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$). An algorithm called `worst_case_multiplier` generates a multiplier that is connected in a worst-case fashion to consume maximum energy. `worst_case_multiplier` is summarized in Figure 4.10. The multipliers generated using `worst_case_multiplier` are indexed by *WC* (e.g., $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$).

4.3.2 Experiments

The proposed method shown in Figure 4.9, the SWS power estimator, and a gate-level VHDL code generator, are all implemented in C++. Inputs to the optimization program are the static probabilities of the primary inputs. It is assumed that the primary input bits are spatially and temporally uncorrelated. After completion of the optimization algorithm, the equivalent VHDL code for the multiplier is generated. To evaluate the quality of the optimization algorithm, this VHDL code is run through a ModelSim simulation. Input stimuli with the same static

```

worst_case_multiplier(input static probabilities)
{
  TheMultiplier= $\emptyset$ ;
   $\mathbb{M}_0$ =generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  computeSWS(TheMultiplier);
   $i=0$ ;
  while ( $\|\mathbb{M}_i\| > 2$ )
  {
    [ $\mathbb{F}_i, \mathbb{H}_i$ ]=reduction_scheme( $\mathbb{M}_i$ );
     $\mathbb{M}_{i+1}$ =apply_FA_HA( $\mathbb{M}_i, \mathbb{F}_i, \mathbb{H}_i$ );
    for  $current\_col=1$  to number of columns in  $\mathbb{M}_i$  do
    {
       $current\_perm$ =random permutation of the PPs in  $current\_col$  of  $\mathbb{M}_i$ ;
       $P_{max}=0$ ;
       $\alpha=\alpha_0$ ;
      for  $current\_iteration=1$  to  $number\_of\_iterations$  do
      {
         $old\_perm=current\_perm$ ;
        swap two random positions in  $current\_perm$ ;
         $P=estimate\_power\_SWS(\mathbb{F}_i, \mathbb{H}_i, current\_perm)$ ;
        if ( $P > P_{max}$ )
           $P_{max}=P$ ;
        else
        {
           $r$ =uniform random ( $r \in [0, 1)$ );
          if ( $r < e^{-\frac{P-P_{max}}{\alpha}}$ )
             $P_{max}=P$ ;
          else
             $current\_perm=old\_perm$ ;
        }
      }
       $\alpha=\frac{\alpha}{\tau}$ ;
    }
    reorder  $current\_col$  of  $\mathbb{M}_i$  as  $current\_perm$ ;
  }
  TheMultiplier.append( $\mathbb{M}_i, \mathbb{F}_i, \mathbb{H}_i$ );
  computeSWS(TheMultiplier);
   $i=i+1$ ;
}
TheMultiplier.append(final_CPA( $\mathbb{M}_i$ ));
computeSWS(TheMultiplier);
}

```

Figure 4.10: The worst-case multiplier generation algorithm

Table 4.6: Average number of transitions for different multipliers
(the input static probabilities Ω_i^{16} are applied to both operands)

	$\mathfrak{M}_{\Omega_i^{16}, \Omega_i^{16}}^{OPT}$	$\mathfrak{M}_{\Omega_i^{16}, \Omega_i^{16}}^{WC}$	$\overline{\mathfrak{M}^{RND}}$
16	1189	1456	1379
15	1037	1286	1216
14	849	1142	1061
13	715	995	901
12	586	848	755
11	477	688	618
10	379	557	487
9	282	433	373

probabilities are used, and the average number of transitions for all nodes is collected. The sum of average node transitions for all nodes in the PPRT is reported. As node load capacitance for the reduction tree does not have large variations, the average number of transitions is strongly related to the power consumption.

Similar to method 1, in the first part of the experiments, multipliers with purely random uncorrelated input bits are considered, where certain numbers of least significant bits are forced to zero. This can for instance appear in systems where the word-length is varying due to changes in the quality of service requirements. The optimized (and worst-case) 16×16 -bit multipliers are designed to be optimized for input probability vectors Ω_i^{16} ($i = 9..16$). In Table 4.6 the average number of transitions is given for $\mathfrak{M}_{\Omega_i^{16}, \Omega_i^{16}}^{OPT}$ and $\mathfrak{M}_{\Omega_i^{16}, \Omega_i^{16}}^{WC}$ when static input probability vector Ω_i^{16} is applied to both operands. The rightmost column in Table 4.6 is dedicated to $\overline{\mathfrak{M}^{RND}}$. The numbers in this column are average numbers of transitions for 10 randomly generated multipliers. From Table 4.6, it can be seen that the average numbers of transitions for optimized multipliers are from 18.3% to 34.9% smaller than the worst-case multipliers. Compared to random multipliers, optimized multipliers have from 13.8% to 24.3% smaller average numbers of transitions. The saving in power consumption is larger when more bits have zero one-probabilities.

As discussed in Section 3.1.1, the estimated power used in the optimization routine is dependent on the delay model used in the power estimator. This affects the generated multipliers considerably. The PPRT that is optimized using a particular delay model, does not exhibit the same amount of saving when it is operating

Table 4.7: Average number of transitions with realistic delay model

(the multipliers are optimized using the fanout delay model but the VHDL simulations are using realistic delay model for a $0.35\mu m$ CMOS library)

	Ω_{16}^{16} applied to input operands	Ω_9^{16} applied to input operands
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	878.2	225.6
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{OPT}$	884.5	201.6
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$	921.4	257.7
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{WC}$	918.7	272.7

Table 4.8: Estimations of energy per operation from Synopsys Power Compiler for $0.35\mu m$ CMOS library

(the multipliers are optimized using the fanout delay model)

	Ω_{16}^{16} applied to input operands	Ω_9^{16} applied to input operands
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	183.2pJ	63.4pJ
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{OPT}$	186.4pJ	59.8pJ
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$	192.1pJ	73.2pJ
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{WC}$	189.4pJ	75.0pJ

Table 4.9: Estimations of energy per operation from Synopsys Power Compiler for a $65nm$ CMOS library

(the multipliers are optimized using the fanout delay model)

	Ω_{16}^{16} applied to input operands	Ω_9^{16} applied to input operands
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	2.0651pJ	705.6fJ
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{OPT}$	2.0763pJ	665.9fJ
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$	2.1328pJ	802.2fJ
$\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{WC}$	2.1232pJ	820.2fJ

with another delay model. A circuit with two different delay models may create completely different patterns of spurious transitions. Therefore the savings due to minimizing the glitches may be annihilated because of the mismatch between the delay models in the optimizer and the real circuit. Table 4.7 reports the average number of transitions for the multipliers that are optimized using the fanout delay model but the VHDL simulations are using the realistic delay model from a $0.35\mu m$ - $3.3V$ CMOS library. Comparing the values in Table 4.7 with the equivalent values in Table 4.6, it can be seen that the reductions in average number of transitions are deteriorated when the delay model in optimizer does not exactly match the real circuit delays.

Table 4.8 and Table 4.9 report the estimated energy per operation for the multipliers that are optimized using the fanout delay model, in $0.35\mu m$ - $3.3V$ CMOS and $65nm$ - $0.9V$ CMOS libraries, respectively. These numbers are obtained from Synopsys Power-Compiler. Since the optimizer uses a different and simpler delay model, the amount of saving is less than the numbers reported in Table 4.6. A more accurate delay model for the optimizer is expected to result in larger savings. The SWS power estimator is not restricted to the fanout delay model as discussed in Section 3.1.1. However, using a realistic delay model results in larger number of possible waveforms because the delays are not integer numbers anymore. Moreover, in a realistic delay model, the delay of a gate is dependent on its input values which increases the number of waveforms even further. This slows down the power estimator because more waveforms need to be processed for each node. According to the results in Table 4.8, the amount of power saving between $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$ and $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$ when Ω_{16}^{16} is applied to both input operands is 4.6%; while it is reported 18.3% in Table 4.6. When Ω_9^{16} is applied to both operands, the amount of power saving between $\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{OPT}$ and $\mathfrak{M}_{\Omega_9^{16}, \Omega_9^{16}}^{WC}$ is 20% (compared to 35% reported in Table 4.6). The savings are deteriorated because of mismatch between the delay models in the optimizer and the real circuit. This is more serious when the input bits have similar input static probabilities.

Table 4.10 compares the average number of transitions for $\mathfrak{M}_{\Omega_i^{16}, \Omega_i^{16}}^{OPT}$ ($i = 9..16$), when different static input probability vectors (Ω_j^{16} , $j = 9..16$) are applied. The average number of transitions for the multiplier \mathfrak{M} , when input static probability vectors \mathbf{p}_A and \mathbf{p}_B are applied to the input operands is denoted by $\mathbf{D}(\mathfrak{M}, \mathbf{p}_A, \mathbf{p}_B)$ in Table 4.10. The multiplier that is optimized for Ω_i^{16} is the one that has the lowest average number of transitions when Ω_i^{16} is applied. However, when a different input static probability vector is applied, the average number of transitions for this multiplier is not minimum any longer. From Table 4.10

Table 4.10: Average number of transitions for different multipliers
 Values of $\mathbf{D}(\mathfrak{M}_{\Omega_i^{16}, \Omega_j^{16}}^{OPT}, \Omega_j^{16}, \Omega_j^{16})$ for different i and j values

		j							
		16	15	14	13	12	11	10	9
i	16	<u>1189</u>	1040	877	751	632	504	403	308
	15	1208	<u>1037</u>	885	757	621	508	397	305
	14	1241	1067	<u>849</u>	725	620	495	394	299
	13	1266	1099	929	<u>714</u>	606	500	393	298
	12	1256	1104	933	771	<u>586</u>	482	385	306
	11	1282	1132	973	820	649	<u>477</u>	383	294
	10	1296	1143	977	836	671	525	<u>380</u>	294
	9	1289	1144	1001	854	712	556	409	<u>282</u>

one may conclude that in a system with dynamically varying data word-length, it might be beneficial to optimize the multiplier to the word-length that is most frequently used. However, as it will be discussed in Chapter 7, finding the best solution for systems with variable word-length is more complex. Using a multiplier that is optimized for the most frequently used word-length does not always provide the best solution.

In the second part of the experiments, similar to method 1, signals with unequal static probabilities are applied to the primary input bits. For many natural signals the switching activities of the MSB-bits are lower than that of LSB-bits. In these cases, the multiplier structure can be optimized for the pattern of static probabilities of the input bits. Similar to method 1, input signals with a log-normal distribution are considered. Eq. 4.5 gives the probability density function for a random variable with log-normal distribution. In this experiment σ is chosen to be one. μ assumes three different values: $\ln 2^8$, $\ln 2^9$ and $\ln 2^{10}$. Figure 4.12 illustrates the one-probabilities of input bits assuming these values for μ and σ . The probability distribution of such random variables are shown in Figure 4.11. The notations Λ_{10}^{16} , Λ_9^{16} and Λ_8^{16} are used to represent 16-bit input static probability vectors with $\mu = \ln 2^{10}$, $\mu = \ln 2^9$ and $\mu = \ln 2^8$, respectively.

Table 4.11 specifies the average number of transitions for multipliers that are optimized for different input static probability vectors. The numbers reported in

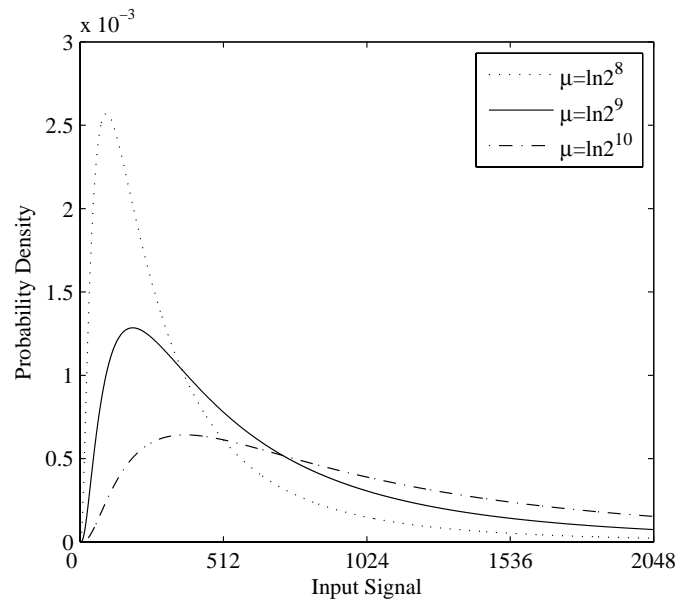


Figure 4.11: Log-normal signal distribution

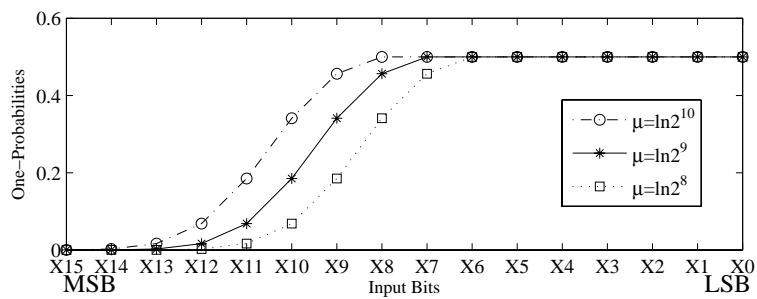


Figure 4.12: One-probabilities of input bits with log-normal distribution
 Input signals for both operands have log-normal distribution
 with $\mu = \ln 2^8$, $\mu = \ln 2^9$ and $\mu = \ln 2^{10}$ ($\sigma = 1$)

Table 4.11: Average number of transitions for different multipliers

Values of $\mathbf{D}(\mathfrak{M}, \mathbf{p}, \mathbf{p})$ for different multipliers (\mathfrak{M}) and different input static probability vectors (\mathbf{p})

	$\mathbf{p} = \Omega_{16}^{16}$	$\mathbf{p} = \Lambda_{10}^{16}$	$\mathbf{p} = \Lambda_9^{16}$	$\mathbf{p} = \Lambda_8^{16}$
$\mathfrak{M} = \mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	<u>1189</u>	570	450	362
$\mathfrak{M} = \mathfrak{M}_{\Lambda_{10}^{16}, \Lambda_{10}^{16}}^{OPT}$	1265	<u>564</u>	458	363
$\mathfrak{M} = \mathfrak{M}_{\Lambda_9^{16}, \Lambda_9^{16}}^{OPT}$	1299	575	<u>446</u>	352
$\mathfrak{M} = \mathfrak{M}_{\Lambda_8^{16}, \Lambda_8^{16}}^{OPT}$	1325	596	460	<u>343</u>
$\mathfrak{M} = \mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$	1456	703	563	438
$\mathfrak{M} = \mathfrak{M}_{\Lambda_{10}^{16}, \Lambda_{10}^{16}}^{WC}$	1425	720	561	442
$\mathfrak{M} = \mathfrak{M}_{\Lambda_9^{16}, \Lambda_9^{16}}^{WC}$	1402	691	578	437
$\mathfrak{M} = \mathfrak{M}_{\Lambda_8^{16}, \Lambda_8^{16}}^{WC}$	1400	686	558	444
$\mathfrak{M} = \overline{\mathfrak{M}^{RND}}$	1379	661	526	407

the last row ($\overline{\mathfrak{M}^{RND}}$), are the mean values for ten randomly generated multipliers. The multiplier generated using the `optimized_multiplier()` algorithm in Figure 4.9 in general creates fewer transitions than a random multiplier, even for cases where the input static probability vector that is applied is not what the multiplier is optimized for. From this it can be concluded that a part of the optimization is independent from the input static probabilities. However, further power savings can be achieved if the optimization is performed for the input static probabilities of multiplier's inputs. For example, if the inputs are variables with log-normal distribution (Λ_8^{16}), the multiplier that is optimized for these static probabilities consumes 5% less power than $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$.

4.3.3 Runtime and Complexity

The power estimate function is located in the core of three loops in the algorithm described in Figure 4.9. The computational complexity of the power estimator is approximately proportional to the product of the average number of waveforms and the average logic depth [179]. In the full-adder based PPRT, as discussed

earlier, the glitch filtering does not happen outside the FA and HA cells. When the glitch filtering is not considered, the number of waveforms in the SWSs grows approximately linearly with the logic depth. The upper-bound for the logic depth is $3 \times \text{current_stage} + 1$. Therefore for an $N \times N$ bits multiplier, the computation complexity can be approximated to $O(\text{number_of_iterations} \times (3 \log_{1.5}^N)^3)$, as the number of stages required to compress the partial products is approximately $\log_{1.5}^N$. `number_of_iterations` is the number of iteration in the SA optimization. In the experiments, constructing a 16×16 bits multiplier using this method requires approximately 3 minutes on a PC with Intel Pentium 1.4GHz processor. `number_of_iterations` is chosen to be 1000. Figure 4.13 and Figure 4.14 show the influence of `number_of_iterations` on the achieved results. When the number of iterations is small, the optimization algorithm generates a close-to-random multiplier. More iterations, which translate into longer runtime, result in more power efficient multipliers compared to a random multiplier. However the improvements saturate after a certain number of iterations and the algorithm is not able to produce better results. It is beneficial to choose smaller values for `number_of_iterations` when a column has small number of PPs. On the other hand, for columns with larger number of PPs, `number_of_iterations` can be larger. The initial temperature (α_0) and the cooling factor (τ) are empirically chosen to be 0.1 and 1.015, respectively. The effect of τ in the achieved results is shown in Figure 4.15. Large values of τ reduce the temperature so rapidly that the SA is incapable of performing uphill moves to avoid local minima. Meanwhile, small values of τ lead to unnecessary uphill moves and require more iterations to reach the minimum energy configuration.

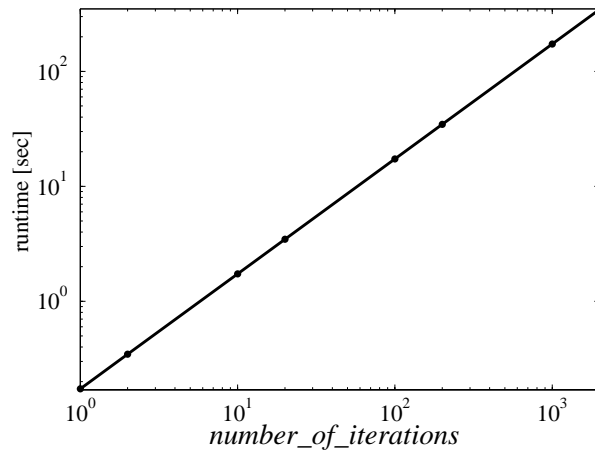


Figure 4.13: Runtime versus number of iteration in SA

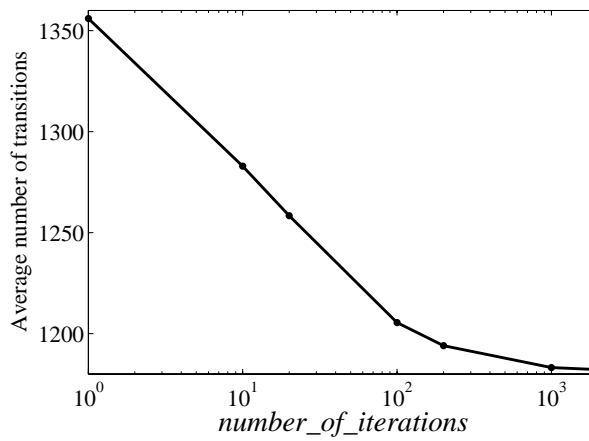


Figure 4.14: Average number of transitions versus number of iterations in SA

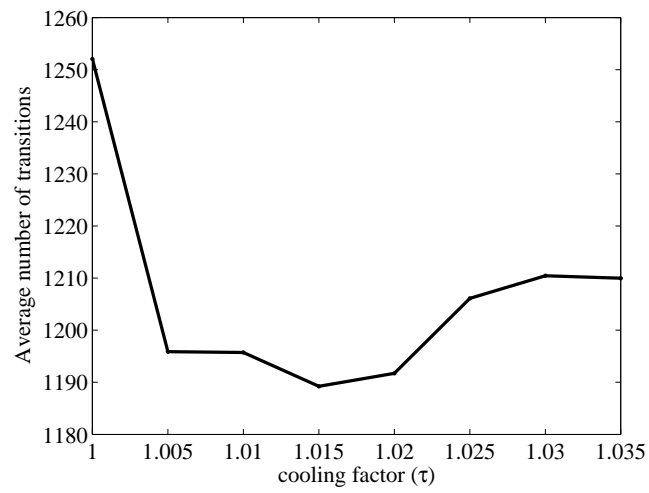


Figure 4.15: Cooling factor

Chapter 5

PPRT Optimization in Presence of Highly Correlated Inputs

Estimating the power consumption is difficult because of the interdependencies between signals in a logic circuit. This becomes even more problematic in presence of highly correlated inputs as the spatial and temporal correlations of the inputs must be considered as well. The data activity of a 16×16 -bit multiplier can vary by as much as one order of magnitude as a function of input correlation [114]. Several methods for estimating power in circuits with highly correlated inputs are proposed [9, 113, 114, 164]. Changing the order of temporally correlated input data changes the temporal correlations significantly. Reordering the input pattern, as a means of switching activity reduction in Multiply-Accumulate (MAC) units, has been investigated in [65, 115, 117].

In Chapter 4, it is assumed that multiplier input signals are independent spatially and temporally; i.e., four random variables $A[n]$, $B[n]$, $A[n-1]$ and $B[n-1]$ are independent for any n . Knowledge about one of them does not change the distribution functions of the others. In this chapter, the dependency of input data and its effect on the power estimation is investigated. The lag-one temporal correlations, e.g. between $A[n]$ and $A[n-1]$, are focused only because correlation of two non-consecutive random variables (e.g. $A[n]$ and $A[n-2]$) will not affect the switching activity, since a transition is defined based on the current value of a node and its previous value only. Therefore, correlations between non-consecutive values do not contribute to the transition activity.

A complete coverage of the input signal spatiotemporal correlations requires knowledge about joint probabilities of all input bits in two consecutive time instance. Let I be the set of all input bits a_i and b_j for $0 \leq i \leq M-1$ and

$0 \leq j \leq N - 1$:

$$I = \{a_0, \dots, a_i, \dots, a_{M-1}, b_0, \dots, b_j, \dots, b_{N-1}\} \quad (5.1)$$

The complete spatiotemporal correlations are described as a joint probability distribution function of all variables in I . Indeed considering thorough correlations is difficult and impractical even for small size multipliers. To make the problem more tractable, the spatiotemporal correlations are approximated. In the power estimator described in Section 3.1.1, the spatial correlations are approximated using pairwise correlation coefficients.

The remainder of this chapter describes a method to integrate the spatiotemporal correlations of inputs in the power estimator and multiplier optimization algorithm in Figure 4.9.

5.1 Spatiotemporal Correlation of Inputs

The inputs of a multiplier can have very complicated dependencies. Let us consider three simple examples at word-level.

1. $|A[n] - A[n - 1]| < \Delta_A$
2. $A[n] \times B[n] > 0$
3. $A[n] + B[n - 1] > 0$

$A[n]$, $A[n - 1]$, $B[n]$ and $B[n - 1]$ are independent if not otherwise stated.

In example 1, A and B are independent but it is known that two consecutive values of input A can not have changes larger than a predefined value Δ_A . In example 2, random variables $A[n]$ and $B[n]$ have the same sign but their sign is independent from that of their previous values. Example 3 illustrates more complex spatiotemporal correlation where the current value of A is correlated to the previous value of B . In example 1, only temporal correlations are present with no spatial correlations. In example 2, spatial correlations are present with no temporal correlations. In example 3, a combination of spatial and temporal correlations are present. Obviously, the spatiotemporal correlations at word-level will translate into spatiotemporal correlations at bit-level. Similar situations can appear at bit-level. Two arbitrary bits from I (Eq. 5.1) can have pure temporal correlations, pure spatial correlations, or a combination of temporal and spatial correlations.

5.1.1 Modeling Spatial Correlations

The spatial correlations in between circuit nodes have two main sources: First is the structural dependency due to reconvergent fanouts (RFO). In the SWS power estimator, this is modeled using correlation coefficients. The pairwise correlation coefficient between two circuit nodes A and B is defined in Eq. 3.10. The correlation coefficients can only model pure spatial correlation as they are independent of time variable n (stationary random process).

The second source of the spatial correlations is due to pattern dependencies at the primary inputs. The correlation coefficients can also be defined for primary input bits. The value of correlation coefficients can be estimated statistically from a large realistic input data stream.

The SWS power estimator described in Chapter 3.1.1 utilizes the correlation coefficients to compute the occurrence probabilities of the waveforms. In Ercolani's method [52] for computing correlation coefficients (Appendix A), the correlation coefficients at the output of a logic gate are estimated using the correlation coefficients of the inputs. Under the independent input assumption, the correlation coefficients of the primary inputs are equal to 1. Now that the inputs are correlated, the estimated correlation coefficients in Eq. 5.4 can simply be used as starting point to compute the correlation coefficients for the rest of the nodes using Ercolani's method. In this way the spatial correlations between the primary inputs will be included in the computations.

The spatial correlations of primary inputs are given to the power estimator as a pairwise correlation coefficient matrix, \mathcal{C} . The value in the i :th row of j :th column in this matrix is the estimated correlation coefficient for the I_i and I_j (i.e., the i :th and j :th element in the set I).

$$I_i = \begin{cases} a_i & \text{if } i \leq M \\ b_{i-M} & \text{if } M < i \leq M + N \end{cases} \quad (5.2)$$

$$\mathcal{C} = [\kappa_{I_i, I_j}]_{(M+N) \times (M+N)} \quad (5.3)$$

where κ_{I_i, I_j} is defined as

$$\kappa_{I_i, I_j} = \frac{p(I_i = 1 \wedge I_j = 1)}{p(I_i = 1)p(I_j = 1)} \quad (5.4)$$

The values of correlation coefficients for primary inputs can be estimated statistically from the large realistic input streams, having a given spatial correlation at

word-level. Using N_S samples of primary input bits, κ_{I_i, I_j} can be approximated as:

$$\kappa_{I_i, I_j} = \lim_{N_S \rightarrow +\infty} \frac{\frac{1}{N_S} \sum_{n=0}^{N_S-1} (I_i[n] \cdot I_j[n])}{\left(\frac{1}{N_S} \sum_{n=0}^{N_S-1} I_i[n] \right) \left(\frac{1}{N_S} \sum_{n=0}^{N_S-1} I_j[n] \right)} \quad (5.5)$$

Choosing $N_S = 100000$ gives a good approximation for most applications.

5.1.2 Modeling Temporal Correlations

The temporal correlations are very important and estimation of power without considering temporal correlations can be extremely inaccurate. Methods for estimating the transition activities at bit-level from word-level input statistics (mean, variance and temporal correlation) are proposed in the past [98, 153]. Landman and Rabaey in [98] proposed a dual bit type (DBT) method to overcome the problem of temporal dependencies. The word-level lag-one temporal correlation is represented by ρ :

$$\rho = \frac{E[X[n]X[n-1]] - E[X[n]]^2}{E[X[n]^2] - E[X[n]]^2} = \frac{R_{XX}(1) - \mu^2}{\sigma^2} \quad (5.6)$$

DBT divides a word with gaussian distribution into three regions depending on the switching activities. These regions are LSB, linear, and MSB, as depicted in Figure 5.1. The bits in the LSB region have random activities with 0.5 switching probability at the input bits. However, the MSB region shows correlated activities. For simplicity, DBT approximates the input data with two types of bits (neglecting the linear region): Uniform white noise (UWN) for LSBs and sign for MSBs. The break points of these regions are empirically defined as [98]:

$$\begin{aligned} BP_0 &= \log_2 \sigma + \log_2 \left(\sqrt{1 - \rho^2} + \frac{|\rho|}{8} \right) \\ BP_1 &= \log_2 (|\mu| + 3\sigma) \\ BP &= \frac{BP_0 + BP_1}{2} \end{aligned} \quad (5.7)$$

In order to include the temporal correlations in the SWS power estimator, a small change is necessary in the initialization of the SWSs for primary inputs. The SWSs for primary input nodes are initialized by four waveforms holding one,

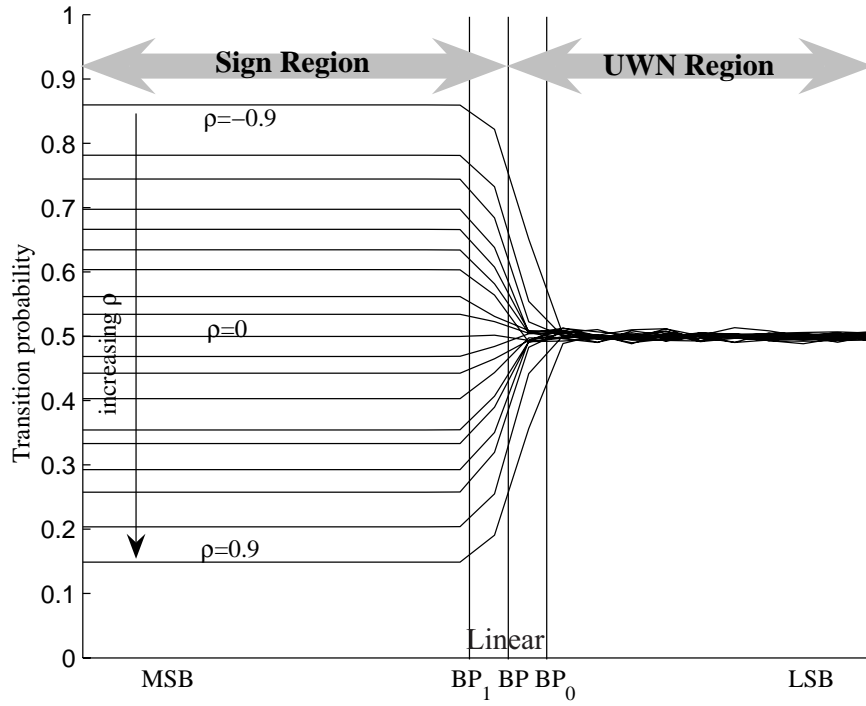


Figure 5.1: Transition activity vs. bit positions with varying ρ

holding zero, zero-one transition and one-zero transition (Chapter 3.1.1.1).

$$\left\{ \begin{array}{l} W_{11} = \Psi(-\infty, +\infty) \\ W_{00} = \Psi(+\infty, -\infty) \\ W_{01} = \Psi(0, +\infty) \\ W_{10} = \Psi(+\infty, 0) \end{array} \right\}$$

Under the temporal independence assumption for input values before and after time 0, occurrence probabilities can be assigned to each waveform based on their static probabilities. Let p_x be the one-probability at an arbitrary input node x . Then $1 - p_x$ will be zero-probability at this node. From Eq. 3.9, the initialization of waveform probabilities for uncorrelated primary inputs are as:

$$\left\{ \begin{array}{l} p(W_{11}) = p_x \times p_x = p_x^2 \\ p(W_{00}) = (1 - p_x) \times (1 - p_x) = 1 - 2p_x + p_x^2 \\ p(W_{01}) = (1 - p_x) \times p_x = p_x - p_x^2 \\ p(W_{10}) = p_x \times (1 - p_x) = p_x - p_x^2 \end{array} \right.$$

When the inputs have temporal correlation, the occurrence probabilities of the waveforms will be different. Using Eq. 5.6, the lag-one temporal correlation ratio for a bit x is defined as:

$$\rho_x = \frac{p \{ (x[n] = 1) \wedge (x[n-1] = 1) \} - p_x^2}{p_x - p_x^2} \quad (5.8)$$

The maximum magnitude of ρ_x is 1. ρ_x is 0 if x does not have temporal dependence. Two methods (exact and approximative) for computing bit-level temporal correlations from word-level signal statistics are presented in [153]. In the experiments presented in Section 5.3 the temporal correlations are approximated by profiling a large number of input samples (100000 samples), which satisfy the properties assumed in the experiment (i.e., the word-level probability distribution and the word-level temporal correlation). This provides a good accuracy for the computed bit-level temporal correlations.

In the presence of temporal correlations, the occurrence probabilities of the initial waveforms in Eq. 5.8 are set to:

$$\left\{ \begin{array}{l} p(W_{11}) = p \{ (x[n-1] = 1) \wedge (x[n] = 1) \} = \rho_x(p_x - p_x^2) + p_x^2 \\ p(W_{00}) = p \{ (x[n-1] = 0) \wedge (x[n] = 0) \} = \rho_x(p_x - p_x^2) + (1 - p_x)^2 \\ p(W_{01}) = p \{ (x[n-1] = 0) \wedge (x[n] = 1) \} = (1 - \rho_x)(p_x - p_x^2) \\ p(W_{10}) = p \{ (x[n-1] = 1) \wedge (x[n] = 0) \} = (1 - \rho_x)(p_x - p_x^2) \end{array} \right. \quad (5.9)$$

Assigning the value of temporal correlation to zero (i.e., temporal independence) will result in the probabilities in Eq. 3.9.

The lag-one temporal correlation ratio, ρ_x , can be approximated statistically using a large input stream:

$$\rho_x = \lim_{N_S \rightarrow +\infty} \frac{\frac{1}{N_S - 1} \sum_{n=1}^{N_S-1} (x[n] \cdot x[n-1]) - \left(\frac{1}{N_S} \sum_{n=0}^{N_S-1} x[n] \right)^2}{\frac{1}{N_S} \sum_{n=0}^{N_S-1} x[n] - \left(\frac{1}{N_S} \sum_{n=0}^{N_S-1} x[n] \right)^2} \quad (5.10)$$

where $p_x = \lim_{N_S \rightarrow +\infty} \frac{1}{N_S} \sum_{n=0}^{N_S-1} x[n]$ is the one-probability at node x (Eq. 1.19).

Using Eq. 5.10, the lag-one temporal correlation ratio vector \mathcal{R} is computed for the primary inputs ($I_i \in I$):

$$\mathcal{R} = [\rho_{I_i}]_{1 \times (M+N)} \quad (5.11)$$

\mathcal{R} is a vector of length $M + N$, where the i :th element in this vector represents the lag-one temporal correlation ratio for $I_i \in I$.

5.2 Design Methodology

Including the information about spatiotemporal correlations between the multiplier's inputs improves the optimization algorithm. This information is captured using lag-one temporal correlation ratios (\mathcal{R}) and pairwise correlation coefficients (\mathcal{C}) for primary inputs. The design methodology is therefore generalized to include such inputs as summarized in Figure 5.2. First, large input streams are generated that satisfy the desired properties (e.g. correlations and static probabilities). For most applications, generating such input streams is easier than computing static-probabilities and classifying the correlations. Indeed such input streams are often needed to estimate static probabilities and dependencies. After generating the input streams, one-probabilities, lag-one temporal correlation ratios, and pairwise correlation coefficients can easily be computed using Eq. 1.19, Eq. 5.10 and Eq. 5.5 respectively. They will be used in the SWS power estimator. Using the computed lag-one temporal correlation ratios and one-probabilities, the SWSs for primary inputs are initialized using the waveforms in Eq. 5.9. At this point the

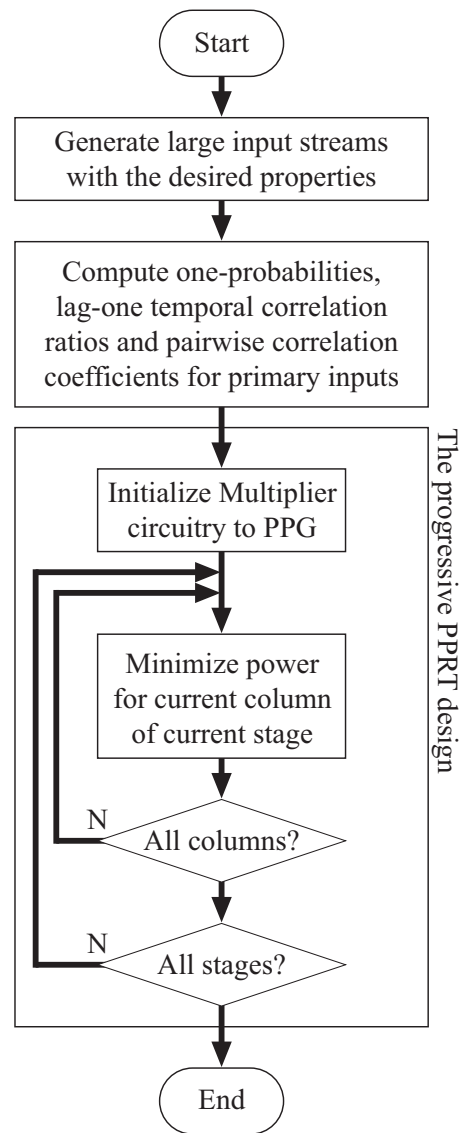


Figure 5.2: The generalized design methodology

progressive PPRT design algorithm in Figure 4.9 can be utilized. The pairwise correlation coefficients will be used for computing the correlation coefficients for intermediate nodes.

5.3 Experiments

This section is divided into three parts. The first part is studying the effect of word-level temporal correlation on multiplier inputs. The second part deals with cases where the inputs are spatially correlated. The third part is optimization of a multiplier used in a multiply-accumulate (MAC) based finite-length impulse response (FIR) filter.

5.3.1 Temporal correlations of input words

In this experiment a signed 16×16 -bit multiplier is considered. The input bit-vectors are assumed to be random variables with normal distributions. The normal distribution also called Laplace-Gaussian distribution, is defined using two parameters, the mean μ and variance σ . The normal distribution arises in many areas. Many natural signals and physical phenomena (like noise) can be approximated by the normal distribution. The probability density function (PDF) of the normal distribution is given by the Gaussian function:

$$f_X(\mathcal{X}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\mathcal{X} - \mu)^2}{2\sigma^2}\right) \quad (5.12)$$

The mean μ and σ are kept constant during this experiment ($\mu = 0$ and $\sigma = 2^8$). The word-level lag-one temporal correlation ratio between the input words, ρ , is altered ($\rho = 0, \pm 0.5, \pm 0.9$ and ± 0.99). For each ρ , after profiling the inputs and computing the vector \mathcal{R} (Eq. 5.11) and the matrix \mathcal{C} (Eq. 5.3), the optimized and worst-case multipliers are generated using algorithms in Figures 4.9 and 4.10. For simplicity, a Gaussian input stream with word-level temporal correlation ratio ρ represented using 16 bits is denoted as Φ_ρ^{16} in this section. Figure 5.3 illustrates the lag-one temporal correlation ratio vector (\mathcal{R}) for $\Phi_{0.99}^{16}$ input pattern. The sign and UWN regions can be seen in this figure. As can be seen in Figure 5.3, the sign regions of input words have strong temporal correlations. Figure 5.4 illustrates the pairwise correlation coefficient matrix (\mathcal{C}) for $\Phi_{0.99}^{16}$ input pattern using different circle radii. As it can be seen in this figure, the input bits (specially significant bits) are not independent spatially and spatial correlations are present

Table 5.1: Average number of transitions for different multipliers with temporally correlated inputs

	Input pattern applied to both operands							
	Φ_0^{16}	$\Phi_{0.5}^{16}$	$\Phi_{0.9}^{16}$	$\Phi_{0.99}^{16}$	$\Phi_{-0.5}^{16}$	$\Phi_{-0.9}^{16}$	$\Phi_{-0.99}^{16}$	Ω_{16}^{16}
$\mathfrak{M}_{\Phi_0^{16}, \Phi_0^{16}}^{OPT}$	<u>1019</u>	970	881	681	1073	1138	1151	1195
$\mathfrak{M}_{\Phi_{0.5}^{16}, \Phi_{0.5}^{16}}^{OPT}$	1024	<u>962</u>	881	675	1079	1145	1156	1205
$\mathfrak{M}_{\Phi_{0.9}^{16}, \Phi_{0.9}^{16}}^{OPT}$	1028	967	<u>875</u>	668	1082	1141	1160	1210
$\mathfrak{M}_{\Phi_{0.99}^{16}, \Phi_{0.99}^{16}}^{OPT}$	1045	985	891	<u>655</u>	1105	1171	1190	1232
$\mathfrak{M}_{\Phi_{-0.5}^{16}, \Phi_{-0.5}^{16}}^{OPT}$	1027	972	889	686	<u>1071</u>	1131	1153	1201
$\mathfrak{M}_{\Phi_{-0.9}^{16}, \Phi_{-0.9}^{16}}^{OPT}$	1040	988	909	690	1088	<u>1126</u>	1144	1205
$\mathfrak{M}_{\Phi_{-0.99}^{16}, \Phi_{-0.99}^{16}}^{OPT}$	1039	988	907	699	1084	1135	<u>1128</u>	1206
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	1034	984	900	731	1108	1166	1193	<u>1189</u>
$\mathfrak{M}_{\Phi_0^{16}, \Phi_0^{16}}^{WC}$	1340	1260	1147	854	1399	1479	1490	1463
$\mathfrak{M}_{\Phi_{0.5}^{16}, \Phi_{0.5}^{16}}^{WC}$	1336	1265	1149	862	1396	1479	1486	1460
$\mathfrak{M}_{\Phi_{0.9}^{16}, \Phi_{0.9}^{16}}^{WC}$	1334	1264	1154	868	1395	1469	1482	1455
$\mathfrak{M}_{\Phi_{0.99}^{16}, \Phi_{0.99}^{16}}^{WC}$	1317	1248	1140	872	1374	1442	1459	1434
$\mathfrak{M}_{\Phi_{-0.5}^{16}, \Phi_{-0.5}^{16}}^{WC}$	1327	1248	1132	846	1405	1470	1492	1463
$\mathfrak{M}_{\Phi_{-0.9}^{16}, \Phi_{-0.9}^{16}}^{WC}$	1323	1238	1121	842	1394	1482	1495	1461
$\mathfrak{M}_{\Phi_{-0.99}^{16}, \Phi_{-0.99}^{16}}^{WC}$	1321	1243	1126	839	1390	1474	1504	1457
\mathfrak{M}^{RND}	1223	1157	1056	798	1283	1355	1370	1378

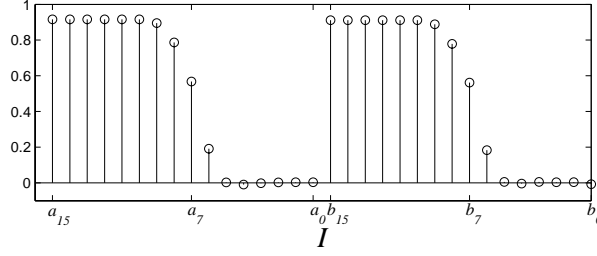


Figure 5.3: Illustration of lag-one temporal correlation ratio vector (\mathcal{R}) for $\Phi_{0.99}^{16}$

for the sign region (Figure 5.4). After computing \mathcal{R} and \mathcal{C} , these values are fed into the optimization algorithm as the primary correlations. The results of the optimization are shown in Table 5.1. $\mathfrak{M}_{\Phi_{\rho}^{16}, \Phi_{\rho}^{16}}^{OPT}$ and $\mathfrak{M}_{\Phi_{\rho}^{16}, \Phi_{\rho}^{16}}^{WC}$ are optimized and worst-case multipliers for the input streams Φ_{ρ}^{16} at both input operands. $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$ is, similar to the experiments in Chapter 4, the optimized multiplier for uncorrelated input stream with 0.5 one-probabilities for all bits. The numbers in the last row ($\overline{\mathfrak{M}^{RND}}$) are the mean values for 10 randomly generated multipliers. The optimized multipliers exhibit about 17% and 24% less transitions compared to the random multipliers and worst-case multipliers, respectively. Note that all input streams (Φ_{ρ}^{16} s and Ω_{16}^{16}) have 0.5 one-probabilities. The difference is in the spatiotemporal correlations of the inputs. Significant reduction in the average number of transitions can be achieved by considering the spatiotemporal correlations. For example, $\mathfrak{M}_{\Phi_{0.99}^{16}, \Phi_{0.99}^{16}}^{OPT}$ has about 10% less transitions compared to $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$, when the input streams $\Phi_{0.99}^{16}$ are applied to the inputs.

5.3.2 Spatial correlations of input words

Four different word-level spatial correlation conditions are assumed in this experiment:

- $A = B$
- $A + B \leq 0$
- $|A - B| < 256$
- $A \times B \geq 1024$

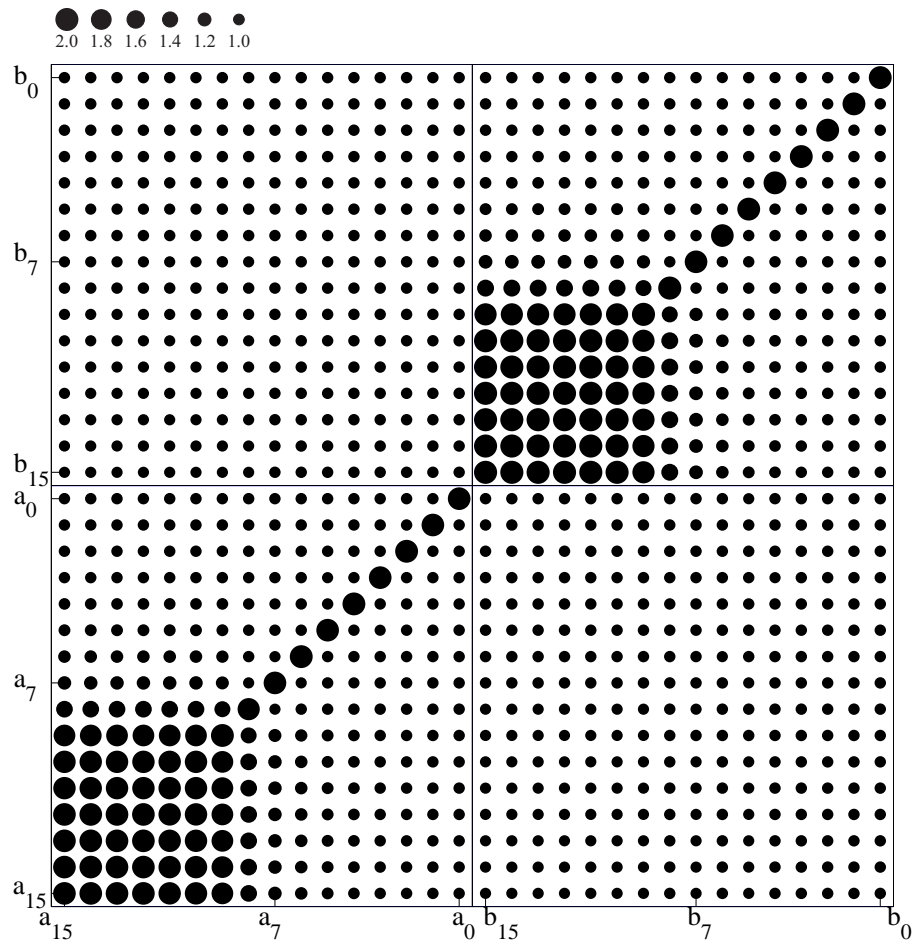


Figure 5.4: Illustration of pairwise correlation coefficient matrix (C)
for $\Phi_{0.99}^{16}$ using circle radii

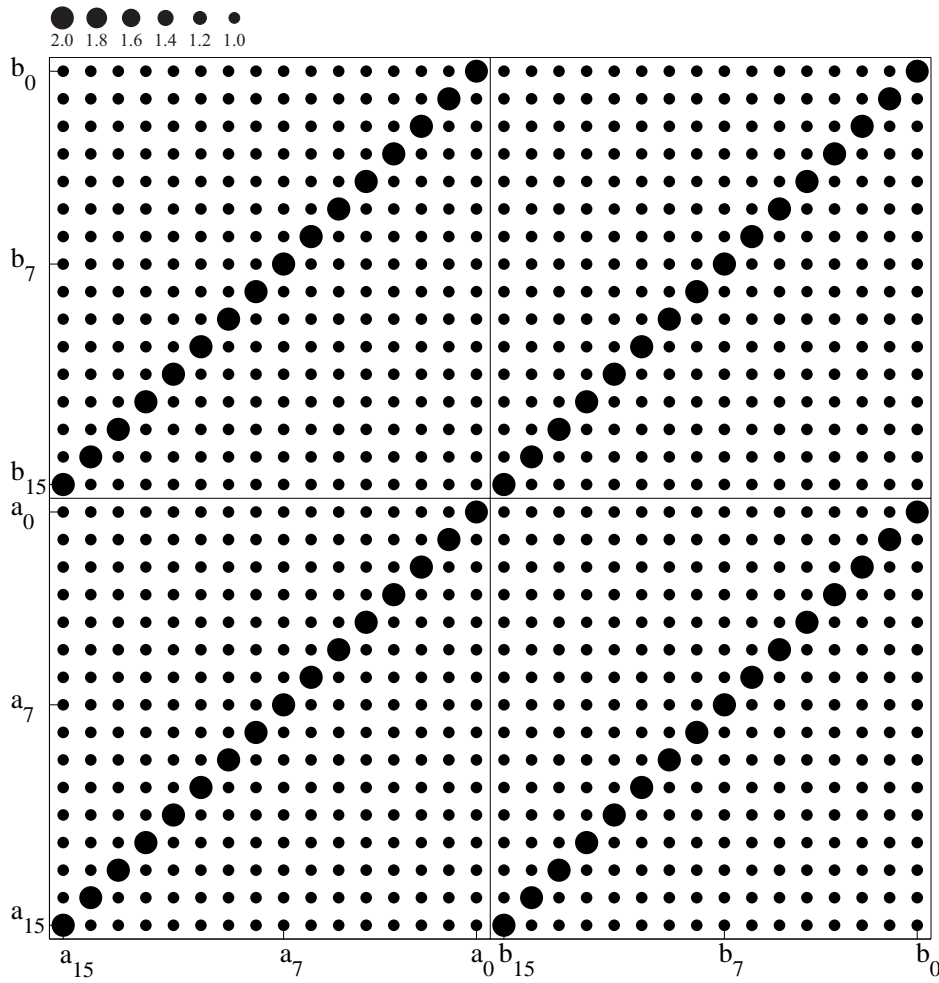


Figure 5.5: Illustration of pairwise correlation coefficient matrix (C) for correlated inputs $(A = B)$ using circle radii

Table 5.2: Average number of transitions for different multipliers with spatially correlated inputs

Correlation Condition	Average number of transitions			
	Optimized	Worst-case	$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	$\overline{\mathfrak{M}^{RND}}$
Uncorrelated	1189	1456	1189	1379
$A = B$	951	1471	1179	1350
$A + B \leq 0$	1102	1408	1143	1335
$ A - B < 256$	1092	1458	1181	1366
$A \times B \geq 1024$	1127	1447	1164	1363

Only one of the correlation conditions are used at a time. The stated condition is the only dependency between the input words. No temporal correlations are involved in this experiment. In order to generate the input streams, first the input word A is selected randomly with a uniform distribution. Then an interval is specified for B such that it satisfies the correlation condition. B is selected randomly from this interval. The pairwise correlation coefficient matrix for primary inputs (C) is visualized in Figure 5.5 for the case with $A = B$ condition. After computing the pairwise correlation coefficients using Eq. 5.5, the optimized and worst-case multipliers are generated. Table 5.2 summarizes the average number of transitions for the optimized, worst-case and random multipliers. Each row specifies the input streams that is applied to the multipliers. For example the first row corresponds to uncorrelated input streams (also denoted as Ω_{16}^{16}). The number in the second column of each row is the average number of transitions for the multiplier that is optimized for the corresponding input stream. Similarly, the third column gives the average number of transitions for the worst-case multipliers. Table 5.2 also compares the optimized multipliers with $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$. The column identified as $\overline{\mathfrak{M}^{RND}}$ represents the mean values for 10 randomly generated multipliers. For some of the correlation conditions including the dependencies in the optimization program results in significantly better designs. For example, the multiplier that is optimized for the condition $A = B$, has 19.3%, 29.6%, and 35.4% less transitions compared to $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$, random multipliers and the worst-case multiplier respectively.

5.3.3 MAC-based FIR filter

The last experiment in this section is optimizing the multiplier used to implement an FIR filter. The output of an N :th-order FIR filter is computed as

$$Y[n] = \sum_{i=0}^N H[i]X[n-i] \quad (5.13)$$

where the filter impulse response, $H[n]$, determines the frequency response of the filter. The transfer function of the FIR filter is

$$\mathbb{H}(z) = \sum_{i=0}^N H[i]z^{-i} \quad (5.14)$$

The two most common filter structures for realizing the transfer function in Eq. 5.14 are the direct form and the transposed direct form structures depicted in Figure 5.6. As it can be seen from Figure 5.6, the basic arithmetic operation is a multiplication followed by an addition. This is usually called a multiply-accumulate (MAC) operation and is commonly supported in programmable DSP processors [99]. In this experiment it is assumed that the FIR filter is realized using the transposed direct form and the input data and coefficients are stored in separate memories as shown in Figure 5.7. Each output is computed by performing $(N + 1)$ MAC operations. The passband and stopband edges are at 0.2π and 0.3π radians, respectively. The multiplier used is signed 24×24 -bits. The filter order is 64. For the given specifications, the filter is designed using the Park-McClellan method [116], which is based on the Remez algorithm [156]. For simplicity the same weights are assigned for the passband and the stopband ripples. The filter coefficients are scaled by a power of two such that the magnitude of the largest coefficients is represented using all available bits, i.e. $0.5 \leq \max(|H[i]|) < 1$.

The input data to the described filter is assumed to be uncorrelated and all input bits have 0.5 one-probabilities. Each output is computed by performing 65 MAC operations. Therefore one of the operands (the data input) is uncorrelated denoted by Ω_{24}^{24} . The other operand (the coefficient input) is connected to the filter coefficients in the natural order (i.e., without the coefficient reordering techniques). Since the filter coefficients appear in the same order in each iteration, therefore, they are correlated. By profiling the coefficients, the lag-one temporal correlation ratio vector, \mathcal{R} (Eq. 5.11), and pairwise correlation coefficient matrix, \mathcal{C} (Eq. 5.3), can be computed and used for the optimization. Figure 5.8 illustrates \mathcal{C} using circle radii. The results of optimizations are shown in Table 5.3. The

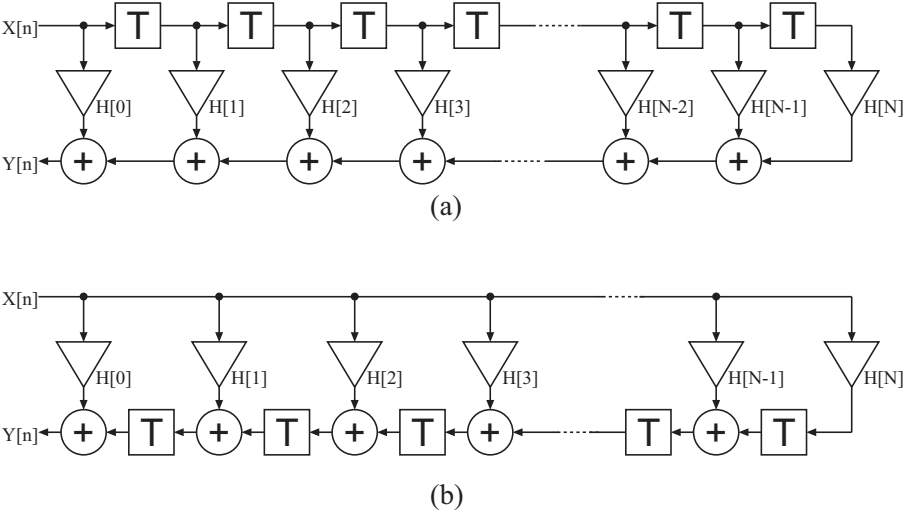


Figure 5.6: FIR filter implementation
(a) direct form and (b) transposed form FIR filter structure

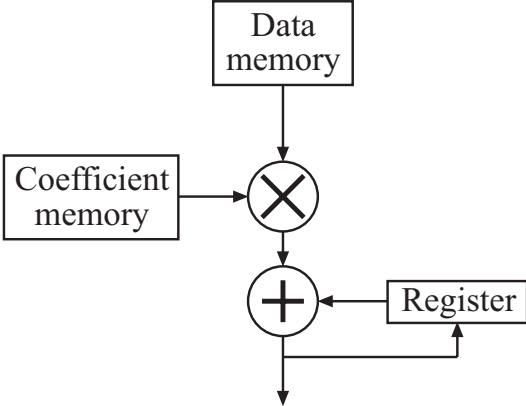


Figure 5.7: Multiply accumulate architecture suitable for realizing direct form FIR filters

Table 5.3: Average number of transitions for MAC-based FIR filter

Multiplier structure	Ω_{24}^{24} is applied to data input Filter coefficients are applied to coefficient input	Ω_{24}^{24} is applied to data and coefficient inputs
$\mathfrak{M}_{\Omega_{24}^{24}, FIR_Coefs}^{OPT}$	<u>3004.9</u>	3105.2
$\mathfrak{M}_{\Omega_{24}^{24}, FIR_Coefs}^{WC}$	3826.4	3807.5
$\mathfrak{M}_{\Omega_{24}^{24}, \Omega_{24}^{24}}^{OPT}$	3097.3	<u>3064.0</u>
\mathfrak{M}^{RND}	3701.5	3678.3

optimization using the dependency information reduces the number of transitions with 3% compared to $\mathfrak{M}_{\Omega_{24}^{24}, \Omega_{24}^{24}}^{OPT}$. The reduction is 18.8% and 21.5% compared to random multipliers and the worst-case multiplier, respectively.

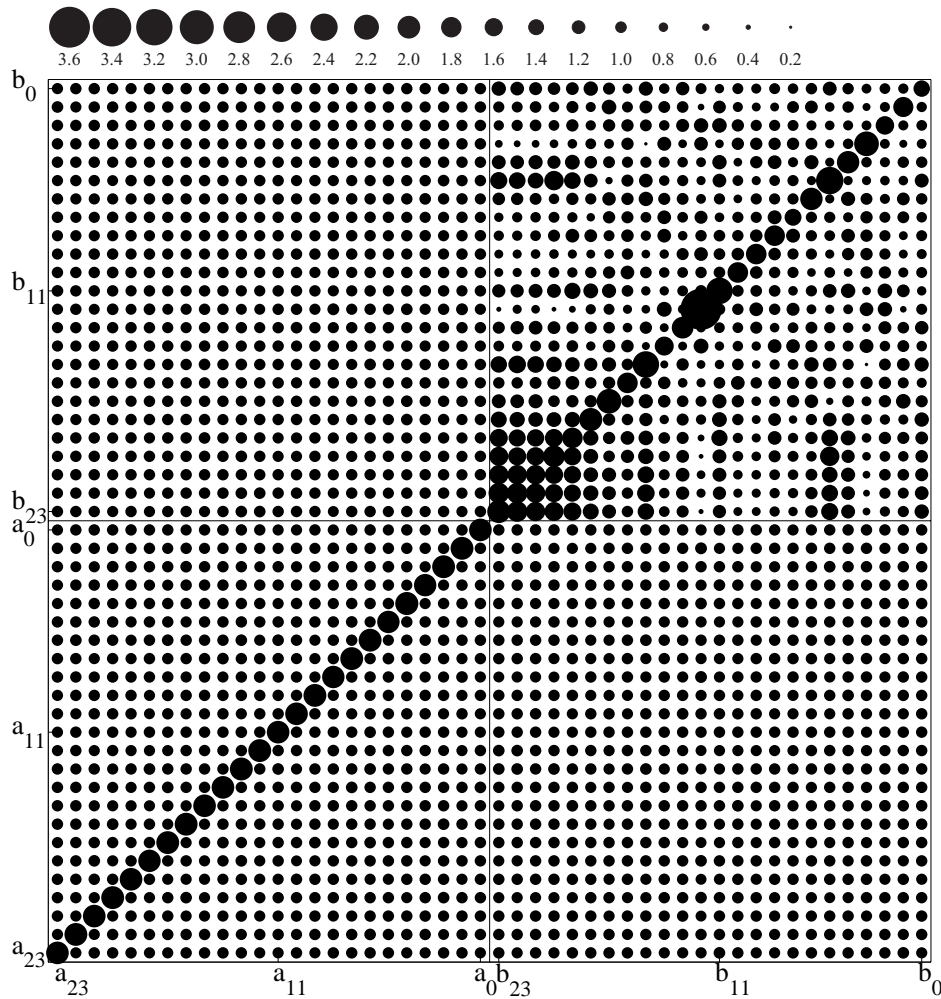


Figure 5.8: Illustration of pairwise correlation coefficient matrix (\mathcal{C}) for the MAC-based FIR filter using circle radii

Chapter 6

Reducing the Static Power for the PPRT

The feature size shrinking has posed a new design challenge as the leakage power dominates the dynamic power on the chip. Traditionally, chip area, speed and dynamic power have been the main criteria in DSP systems design. However, the static power reduction should be given a high priority for a system that is mostly in idle state and operates in short intervals. This is becoming even more important for the CMOS technologies with channel size below 100nm [86]. Indeed, circuit simulations reveal that in the 65nm technology, subthreshold leakage power can be as much as 60% of total chip power in high operation temperatures [150].

In Chapters 4 and 5 optimization methods for dynamic power are introduced. In this chapter the arithmetically equivalent implementations of the PPRT will be utilized for optimization of the power dissipation due to leakage. In the methods for dynamic power optimization (Chapters 4 and 5), an estimator for dynamic power is utilized. Similarly, in the method for static power optimization, a static power estimator is required. The static power estimator described in Chapter 3.2 is utilized in this method.

Different permutations of partial products will result in different static probabilities for the intermediate nodes. Meanwhile, different static probabilities for internal nodes will result in different leakage current values. As discussed in Chapter 3.2, the amount of leakage current in a logic gate strongly depends on its input values [67, 135]. The static probabilities are estimated using the Ercolani's method described in Appendix A. When spatial correlations are present between primary inputs of the multiplier, they can be included in the computations by

feeding in the pairwise correlation coefficient matrix (C) defined in Eq. 5.3. After computing the static probabilities, the static power is estimated using Eq. 3.15.

An important difference must be emphasized between the technique introduced in this chapter for reducing the total leakage current and methods introduced in [1, 3, 67]. In the earlier methods for reducing the static power dissipation, it is assumed that the circuit will enter a stand-by mode when it is idle and in this stand-by mode the leakage currents are reduced by utilizing low leakage transistors or assigning optimum input vectors to the circuit input nodes. However, the proposed technique in this chapter ensures reduction in static power in both operation phase and idle phase. This is becoming even more important as the power dissipation due to leakage becomes comparable with the dynamic power consumption [125].

6.1 Progressive Reduction-Tree Design

By changing the cost function in the progressive reduction-tree design algorithm in Chapters 4 and 5, from dynamic power estimator to static power estimator, a new optimizer can be achieved. This optimizer reduces the overall static power consumption in the reduction-tree. Figure 6.1 summarizes the proposed method for progressive reduction-tree design, minimizing static power consumption. The algorithm starts with initialization of the circuit to primary PPs generated by a suitable algorithm. After the PPs are generated, the multiplier structure is progressively designed similar to the algorithm in Figure 4.9. The function `compute_probabilities()` computes static probabilities for the nodes of the multiplier circuit for which probabilities are not computed earlier. Note that each time a circuit portion is appended to `TheMultiplier`, this function is executed, which computes the static probabilities for the new portion. The static power estimator in this loop (`estimate_static_power`) computes the static power exclusively for the current stage being designed, i.e., the current layer of full- and half-adders.

6.1.1 Experiments

The algorithm shown in Figure 6.1 optimizes the multiplier for minimum static power consumption. An algorithm called `worst_case_multiplier_for_static_power` generates a multiplier that is connected in a worst-case fashion to consume maximum static power. Indeed `optimized_multiplier_for_static_power` is simply converted to `worst_case_multiplier_for_static_power` by altering the `estimate_static_power` function

```

optimized_multiplier_for_static_power(input static probabilities and their correlation coefficients)
{
  TheMultiplier=0;
  M0=generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  compute_probabilities(TheMultiplier);
  i=0;
  while (||Mi|| > 2)
  {
    [Fi, Hi]=reduction_scheme(Mi);
    Mi+1=apply_FA_HA(Mi, Fi, Hi);
    for current_col=1 to number of columns in Mi do
    {
      current_perm=random permutation of the PPs in current_col of Mi;
      Pminstatic=+∞;
      α=α0;
      for current_iteration=1 to number_of_iterations do
      {
        old_perm=current_perm;
        swap two random positions in current_perm;
        Pstatic=estimate_static_power(Fi, Hi, current_perm);
        if (Pstatic < Pminstatic)
          Pminstatic=Pstatic;
        else
        {
          r=uniform random (r ∈ [0, 1));
          if (r < e $\frac{P_{min}^{static} - P^{static}}{\alpha}$ )
            Pminstatic=Pstatic;
          else
            current_perm=old_perm;
        }
      }
      α= $\frac{\alpha}{\tau}$ ;
    }
    reorder current_col of Mi as current_perm;
  }
  TheMultiplier.append(Mi, Fi, Hi);
  compute_probabilities(TheMultiplier);
  i=i+1;
}
TheMultiplier.append(final_CPA(Mi));
compute_probabilities(TheMultiplier);
}

```

Figure 6.1: Progressive reduction-tree design using static power estimator

to return the negated estimated power value rather the power value itself. This ensures that the algorithm minimizes the negated static power or maximizes the actual static power. These algorithms are implemented in C++. Inputs to the optimization program are the static probabilities of the primary inputs and their correlation coefficients (the pairwise correlation coefficient matrix, \mathcal{C}). After completion of the optimization algorithm, the equivalent VHDL code for the multiplier is generated. This VHDL code is then simulated using the ModelSim logic simulator from Mentor Graphics, collecting actual static probabilities for all nodes in the circuit. The simulation is carried out for 10000 samples with the input characteristics that is assumed for optimization. These probabilities, together with the lookup-tables for leakage currents of the logic gates in 65nm CMOS library, are used to calculate the static power consumption for the multipliers. The results are reported in Tables 6.1 and 6.2.

In the first part of the experiments in this chapter, independent primary input bits are connected to the multiplier; i.e., the correlation coefficients are set to 1 for primary inputs. Assuming 0.5 as one-probabilities of the independent input bits will result in a uniform distribution for the input operands in the interval $[0, 2^N)$ where N is the operand word-length. This type of data distribution appears frequently specially in computation units where one multiplier is a shared hardware resource between several simultaneously executed applications. In addition to the uniform input distribution, log-normal input distribution is examined. The log-normal distribution is associated to any random variable whose logarithm is normally distributed. A log-normal distribution results if the variable is the product of a large number of independent, identically-distributed variables. Table 6.1 compares the effects of applying the optimization algorithm on unsigned multipliers with different sizes and different input probability distributions. The second column in Table 6.1 specifies the probability distribution of the input bits. UD stands for the uniform distribution where all input bits have one-probabilities equal to 0.5. LN1 and LN2 stand for log-normally distributed inputs with $\mu = \ln 2^8$ and $\mu = \ln 2^{10}$, respectively, where μ is the mean of the variables logarithm (the standard deviation of the variable's logarithm is assumed to be 1). The mean value of a variable with LN1 (or LN2) distribution is 2^8 (or 2^{10}). The static power consumption for optimized, worst-case and random multipliers are given in the third, fourth, and fifth columns, respectively. The number reported as random multiplier is the average static-power for 10 multipliers where the interconnections are chosen randomly. The reduction in static power consumption ranges from 15.6% to 17.3% compared to the worst-case circuits when the inputs have a uniform distribution (UD). The static power for randomly interconnected multipliers lie between the

Table 6.1: Unsigned multipliers with independent input bits

Operand Size	Input Type	Static Power [nW]		
		Optimized	Worst-Case	Random
12 × 12	UD	518	614	577
12 × 12	LN1	513	595	563
16 × 16	UD	994	1187	1114
16 × 16	LN1	979	1097	1050
16 × 16	LN2	984	1134	1075
24 × 24	UD	2416	2921	2732
24 × 24	LN1	2382	2578	2501
24 × 24	LN2	2387	2621	2531

optimized power and the worst-case power (biased towards worst-case numbers). The reduction ratio is slightly less for log-normally distributed inputs where the most significant bits have one-probabilities close to zero. The computation delay and dynamic power consumption of the three types of multipliers are also investigated. The difference between the optimized circuits and the worst-case and random circuits are negligible. The minor variations in the speed and dynamic power are random in direction; i.e., the reduction in static power consumption seems to be orthogonal with the reduction in dynamic power consumption and/or computation delay. In Section 6.2, it is shown how both dynamic and static power can be optimized at the same time.

In the second part of the experiments, it is assumed that the primary input operands are not independent. The spatial correlations of primary inputs can then be captured using correlation coefficients. In this case, instead of initializing the correlation coefficients to 1, they are initialized to their real values. These values are given to the optimization algorithm in a matrix form (\mathcal{C}). Four different correlation conditions are considered in Table 6.2 for signed 16×16 multipliers. The one-probabilities of the input bits are set equal to 0.5. A correlation is assumed to exist between the multiplier operands A and B , where A and B are two's complement bit-vectors. The pairwise correlation coefficients between the input bits are computed by applying the correlation condition (the first column in Table 6.2) on a large number of random numbers (100000 samples) using Eq. 5.5. The rightmost column in Table 6.2 gives the static power consumption of

Table 6.2: Signed 16×16 multipliers with correlated input bits

Correlation Condition	Static Power [nW]			
	Optimized	Worst-Case	Random	OPT-UC
Uncorrelated	990	1182	1101	990
$A + B \leq 0$	991	1203	1107	1009
$ A - B < 256$	978	1197	1111	1031
$AB \geq 1024$	986	1189	1096	997
$A = B$	954	1173	1107	1022

the multiplier OPT-UC when inputs with each of the specified correlation conditions are applied. OPT-UC is a 16-bit signed multiplier optimized for uncorrelated inputs. As can be seen, applying inputs with spatial correlations to OPT-UC increases the static power consumption compared to having uncorrelated inputs. In comparison, the static power consumption is unchanged, or even reduced, when the multiplier is optimized according to the current correlation condition. This is seen from column Optimized. The reduction in static power consumption using the optimization algorithm is between 10.0 and 14.7% compared to the random multipliers and about 17.1–18.7% compared to the worst-case multipliers. Considering the spatial correlations reduces the static power even further compared to OPT-UC. For example, for the correlation condition $A = B$, the optimized multiplier consumes 6.7% less power compared to OPT-UC.

6.2 Reducing Total Power for the PPRT

The total power dissipation in a CMOS circuit is a combination of static and dynamic power as shown in Eq. 1.1. That is

$$P_{Total} = P_{Static} + P_{Dynamic}$$

The dynamic power was once the dominant power consumption term. However, as the result of technology scaling and threshold voltage decreasing, leakage power will soon account for a large portion of total power consumption. The gradual emerging of the leakage power problem has initiated a number of research activities in the past decade. Nevertheless, the majority of power reduction techniques

proposed in the past focus on either dynamic or static power consumption instead of considering the total power. The progressive PPRT design algorithm with dynamic power as its cost function was discussed in Chapters 4 and 5. In this chapter, by implementing the static power estimator and changing the cost function to static power, the progressive PPRT design algorithm was altered to minimize static power. In this section, by combining the two methods, the progressive PPRT design algorithm is generalized to reduce the total power.

When a circuit is in standby situation, the dynamic power will be zero and the total power will be equal to P_{Static} . Let $t_{active}(T)$ be the amount of time that the circuit is active (not in standby) within the time interval T . Then ω can be defined as the ratio between $t_{active}(T)$ and T , when T is infinitely large; i.e

$$\omega = \lim_{T \rightarrow \infty} \frac{t_{active}(T)}{T} \quad (6.1)$$

ω lies between 0 to 1. The total power for systems with standby periods can be then written as

$$P_{Total} = P_{Static} + \omega P_{Dynamic} \quad (6.2)$$

Figure 6.2 summarizes the progressive PPRT design algorithm for total power reduction for the given value of ω . Low values of ω gives larger weight to the static power, while large values bias the optimization toward dynamic power reduction. The total power is computed as weighted sum of static power and dynamic power. `estimate_power_SWS` estimates the average number of transitions. Therefore the average number of transitions must be converted to power (Eq. 1.8) which is approximated using the factor λ ; i.e. the product of λ and the estimated average number of transitions approximates the switching power. The dynamic power depends on the operation frequency. In the reported results in this section, $\lambda = 3\text{nW}/\text{transition per clock cycle}$ is chosen, which is achieved when the inputs arrive with clock frequency equal to 1.7MHz. Using this value, at the room temperature (25°C), a 16×16 -bit multiplier consumes about 22% of the total power due to leakage and 78% of the total power is due to switching.

6.2.1 Experiments

The joint static and dynamic power optimization algorithm in Figure 6.2 is tested for 16×16 -bit multipliers and the results are summarized in Table 6.3. The inputs are assumed to be independent and random with 0.5 one-probabilities (i.e. Ω_{16}^{16}). The multipliers denoted as $\mathfrak{M}_{\omega=\omega_0}^{OPT_{16}}$ are optimized using the joint static and

```

optimized_multiplier(input static probabilities)
{
  TheMultiplier=0;
  M0=generate_PPs(operand sizes);
  TheMultiplier.append(PPG_circuitry);
  computeSWS(TheMultiplier);
  compute_probabilities(TheMultiplier);
  i=0;
  while (||Mi|| > 2)
  {
    [Fi, Hi]=reduction_scheme(Mi);
    Mi+1=apply_FA_HA(Mi, Fi, Hi);
    for current_col=1 to number of columns in Mi do
    {
      current_perm=random permutation of the PPs in current_col of Mi;
      Pmin=+∞;
      α=α0;
      for current_iteration=1 to number_of_iterations do
      {
        old_perm=current_perm;
        swap two random positions in current_perm;
        Pdynamic=estimate_power_SWS(Fi, Hi, current_perm);
        Pstatic=estimate_static_power(Fi, Hi, current_perm);
        P=Pstatic+ωλPdynamic;
        if (P < Pmin)
          Pmin=P;
        else
        {
          r=uniform random (r ∈ [0, 1));
          if (r < e $\frac{P_{min}-P}{\alpha}$ )
            Pmin=P;
          else
            current_perm=old_perm;
        }
        α= $\frac{\alpha}{\tau}$ ;
      }
      reorder current_col of Mi as current_perm;
    }
    TheMultiplier.append(Mi, Fi, Hi);
    computeSWS(TheMultiplier);
    compute_probabilities(TheMultiplier);
    i=i+1;
  }
  TheMultiplier.append(final_CPA(Mi));
  computeSWS(TheMultiplier);
  compute_probabilities(TheMultiplier);
}

```

Figure 6.2: Progressive PPRT design for total power reduction

Table 6.3: 16×16 multipliers optimized for total power

Multiplier Structure	Total Power [nW]				Static Power [nW]	Average number of Transitions
	$\omega = 0.001$	$\omega = 0.01$	$\omega = 0.1$	$\omega = 1$		
$\mathfrak{M}_{\omega=0.001}^{OPT}$	<u>994.2</u>	1030.9	1398.3	5071.6	990.1	1360.5
$\mathfrak{M}_{\omega=0.01}^{OPT}$	994.8	<u>1030.2</u>	1385.0	4933.1	990.8	1314.1
$\mathfrak{M}_{\omega=0.1}^{OPT}$	1004.8	1038.5	<u>1376.2</u>	4753.1	1001.0	1250.7
$\mathfrak{M}_{\omega=1}^{OPT}$	1027.5	1060.2	1386.5	<u>4649.4</u>	1023.9	1208.5
$\mathfrak{M}_{\omega=0}^{OPT}$	994.2	1031.4	1403.0	5119.3	<u>990.1</u>	1376.4
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$	1103.4	1135.5	1456.6	4667.1	1099.8	<u>1189.1</u>
$\mathfrak{M}_{\omega=0.001}^{WC}$	1186.4	1224.0	1599.5	5354.4	1182.3	1390.7
$\mathfrak{M}_{\omega=0.01}^{WC}$	1185.8	1224.0	1605.9	5424.2	1181.6	1414.2
$\mathfrak{M}_{\omega=0.1}^{WC}$	1175.5	1214.4	1604.0	5499.9	1171.2	1442.9
$\mathfrak{M}_{\omega=1}^{WC}$	1153.7	1193.0	1585.2	5508.1	1149.4	1452.9
$\mathfrak{M}_{\omega=0}^{WC}$	1186.6	1223.3	1590.8	5265.2	1182.5	1360.9
$\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$	1106.4	1145.8	1538.9	5470.7	1102.1	1456.2
\mathfrak{M}^{RND}	1105.5	1142.8	1514.9	5236.6	1101.4	1378.4

dynamic power optimization algorithm in Figure 6.2 using $\omega = \omega_0$. Similarly, the multipliers denoted as $\mathfrak{M}_{\omega=\omega_0}^{WC}$ are the worst-case multipliers using $\omega = \omega_0$. The multipliers $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{OPT}$ and $\mathfrak{M}_{\Omega_{16}^{16}, \Omega_{16}^{16}}^{WC}$ are optimized and worst-case multipliers for dynamic power (without considering static power), obtained from algorithms in Figures 4.9 and 4.10 respectively. The numbers in the last row denoted as $\overline{\mathfrak{M}}^{RND}$ are the averaged values for ten randomly generated multipliers. Looking at the numbers in Table 6.3, it can be seen that compared to random multipliers, 9.2–13.7% power can be saved using this optimization method. The amounts of power saving are 14.2–18.3% compared to worst-case multipliers. If the total power is dominated by the static power (i.e. small values of ω), the optimized multiplier's dynamic power is close to that of random multipliers. On the other hand when the total power is dominated by the dynamic power, the optimized multiplier's static power is close to that of random multipliers. This indicates that optimization for static power and dynamic power are to a large extent orthogonal to each other.

Chapter 7

Optimization of Multipliers Operating with Variable Word-Length

The hardware of a digital signal processing (DSP) system can be optimized for the operation conditions, if these conditions are known in the design phase. In a DSP system operating with a nondeterministic data set, these operation conditions can involve probabilistic measures like the distribution of the input data over time and their correlations. For such systems it can in addition be advantageous to dynamically adapt the computation algorithm based on run-time knowledge about the operating condition and the current states of the inputs. Such adaptation of, e.g., the datapath word-length and other system-level parameters, can significantly improve the computation efficiency, power consumption, robustness, or other merits of the system. Adaptive modulation [61, ch. 9], adaptive power control [123, 134] and tunable word-length [30, 104, 127, 200] are among the adaptation methods that have been proposed in the past.

Adaptive word-length variation for example performs a trade off between performance, power consumption and quality of computations. It is shown in [35] that the problem of word-length optimization in DSP systems belongs to the NP-complete problem classification. There are several approaches to tackle the problem of word-length optimization. Some use heuristic methods, and some use analytical or simulation approaches. These are surveyed in [34]. Word-length variation affects the quantization error introduced when fixed-point operators are utilized instead of operators with an ideally infinite accuracy. It is therefore the acceptable threshold of the quantization error that specifies the minimum number

of bits in a DSP system. In many systems this acceptable threshold can vary over time based on, e.g., user preferences, operating conditions such as communication channel noise, and the application that is currently being executed. It is then advantageous to be able to vary the word-length dynamically. The design-time word-length optimization methods trade off area, speed and signal quality, while dynamic word-length variation enables a trade off between power consumption and signal quality.

Dynamic word-length variation does not necessitate dedicated hardware resources for different word-lengths. The system can keep operating with its hardware resources, but use less number of bits; i.e., fix a number of bits from the least significant bit (LSB) side to zero (or one) and force no transition on these bits. The average number of transitions for a multiplier drops approximately quadratically with the number of inactive bits in the input operands. The amount of saving is less than the case where there is dedicated hardware for variable word-length multiplication. However, the dedicated hardware requires additional circuitry to switch between multipliers.

With evermore important stress on lengthening the battery lifetime in portable devices, more and more focus has been given to the power consumption in such systems. Examples of use of dynamic word-length control can be found in [104, 127, 200] for parts of communication systems. [30] introduces dynamic word-length variation in a 3D graphic texture mapping context. In these systems multipliers are among the main power consuming parts, and have consequently been given great attention in this respect.

In the previous chapters, it is shown that using the progressive reduction-tree design algorithm for uncorrelated input bits in Section 4 and for correlated input bits in Section 5 can reduce the power consumption in multipliers. In this chapter, application of these algorithms for systems with varying word-length is studied. In addition to systems which utilize word-length adaptation techniques for power saving, the proposed method is applicable for general-purpose processors where the word-length is defined by the program that at any given time is being executed. Through profiling of the target programs for a general-purpose processor, the power consumption of build-in multipliers can be reduced using the proposed method. Some of the material in this chapter will be published in [182].

The target application for multipliers with variable word-length are DSP systems and general-purpose processors. For both application areas the multiplier is often placed in the critical path and is therefore speed-limiting. Thus, as discussed in Chapter 2, parallel multipliers are utilized because they offer the best performance compared to other multiplier types.

Similar to previous chapters, for the reported results in Section 7.2, modified Baugh-Wooley [6, 70] is used as the 2's complement multiplier PP generation method and modified Dadda/Wallace reduction scheme [12] is chosen as the reduction scheme. This reduction scheme promises minimal hardware resources and minimal output vector size.

7.1 Variable Word-Length

Multiplication with variable input word-length is addressed, where only one single hardware resource is available; i.e., there is no dedicated multiplier for different word-lengths. The notations introduced in Section 4.2.2 is used in this chapter as well. Let the bit-vectors $A(M \text{ bits})$ and $B(N \text{ bits})$ be the input operands to an $M \times N$ -bit multiplier. The static probabilities of bit-vectors A and B are denoted as vectors \mathbf{p}_A and \mathbf{p}_B respectively. The i :th element of the vector \mathbf{p}_A represents the one-probability of the i :th bit of the input A . A multiplier that is optimized using the algorithm in Figure 4.9, to operate with inputs having \mathbf{p}_A and \mathbf{p}_B as their static probability vectors, is denoted as $\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$. Similarly the worst-case multiplier is referred as $\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{WC}$. Using the algorithm in Figure 4.10, the interconnect orders in the worst-case multiplier are chosen so that the power consumption is maximized. Note that even if a multiplier is optimized at design-time using two specific input static probability vectors, it may at run-time be operating under other input conditions. These conditions can also change over time.

In the context of systems with variable word-length, the A and B inputs have l and k active bits, respectively. The $M - l$ and $N - k$ inactive bits in A and B are assumed to be forced to zero. The number of active bits is variant with time. The active bits are assumed to have 0.5 one-probabilities. This results in uniform distributions for the random variables A and B . It is also possible to assume non-uniform distributions for input words, e.g., a Gaussian distribution. Landman and Rabaey in [98] show that the lower bit positions in normally distributed inputs behave like completely random bits with 0.5 one-probabilities. In the scenario of the word-length variation, the random portion of the input bits is forced to zero. In this paper it is assumed that the input signals have uniform distribution. More specifically, the active input bits are uncorrelated and have 0.5 one-probabilities. For simplicity, the notation Ω_l^M from Section 4.2.2 is used, which is a static probability vector of an M -bit uniform random input with l spatiotemporally uncorrelated active bits. The $M - l$ inactive bits are assigned to be zero; i.e., their one-probabilities are zero.

Let $\mathcal{W} = [\mu_{i,j}]_{M \times N}$ be the word-length probability density matrix ($0 \leq \mu_{i,j} \leq 1$). $\mu_{i,j}$ is the probability that the $M \times N$ multiplier will have i and j active bits in the first and the second operands respectively; i.e. $\mu_{i,j}$ is the probability that Ω_i^M and Ω_j^N are applied to A and B respectively. \mathcal{W} is visualized in Figure 7.2 using radii of circles as the probability. \mathcal{W} satisfies the following condition:

$$\sum_{i=1}^M \sum_{j=1}^N \mu_{i,j} = 1 \quad (7.1)$$

Let i_{\max} and j_{\max} be the row and column indexes of the largest element in the word-length probability density matrix; i.e., $\mu_{i_{\max}, j_{\max}}$ is the largest element of the matrix \mathcal{W} . Intuitively, in systems with variable word-length, the multiplier optimization can be performed for this maximal probability. That is, if $i_{\max} \times j_{\max}$ -bit multiplications are the most frequent multiplications, then the multiplier $\mathfrak{M}_{\Omega_{i_{\max}}^M, \Omega_{j_{\max}}^N}^{OPT}$ is expected to consume the lowest power consumption over time. However, as will be demonstrated in the experimental results later in chapter, this is often not the case.

From the word-length probability density matrix, \mathcal{W} , two vectors $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$ are obtained which are average static probability vectors for the first and second operands over an infinitely large time interval, respectively. The η :th element in $\overline{\mathbf{p}}_A$ is equal to:

$$\overline{\mathbf{p}}_A[\eta] = \frac{1}{2} - \frac{1}{2} \sum_{i=1}^{\eta-1} \sum_{j=1}^N \mu_{i,j} \quad (7.2)$$

For example, for the word-length probability density matrix shown in Figure 7.2, the one-probability for 10th bit is $\overline{\mathbf{p}}_A[10] = 0.5 - 0.5 \times (0.02 + 0.05 + 0.10 + 0.15) = 0.34$. Similarly, the ζ :th element in $\overline{\mathbf{p}}_B$ is equal to:

$$\overline{\mathbf{p}}_B[\zeta] = \frac{1}{2} - \frac{1}{2} \sum_{j=1}^{\zeta-1} \sum_{i=1}^M \mu_{i,j} \quad (7.3)$$

For a multiplier where $M = N$ and the probability density matrix \mathcal{W} is symmetrical, the notation $\overline{\mathbf{p}}$ is used instead of $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$ for simplicity. These vectors are applied as inputs to the optimization algorithm and the optimized multiplier is denoted as $\mathfrak{M}_{\overline{\mathbf{p}}_A, \overline{\mathbf{p}}_B}^{OPT}$. The experimental results shows that this multiplier in general exhibits better performance for the system with variable word-length compared to $\mathfrak{M}_{\Omega_{i_{\max}}^M, \Omega_{j_{\max}}^N}^{OPT}$.

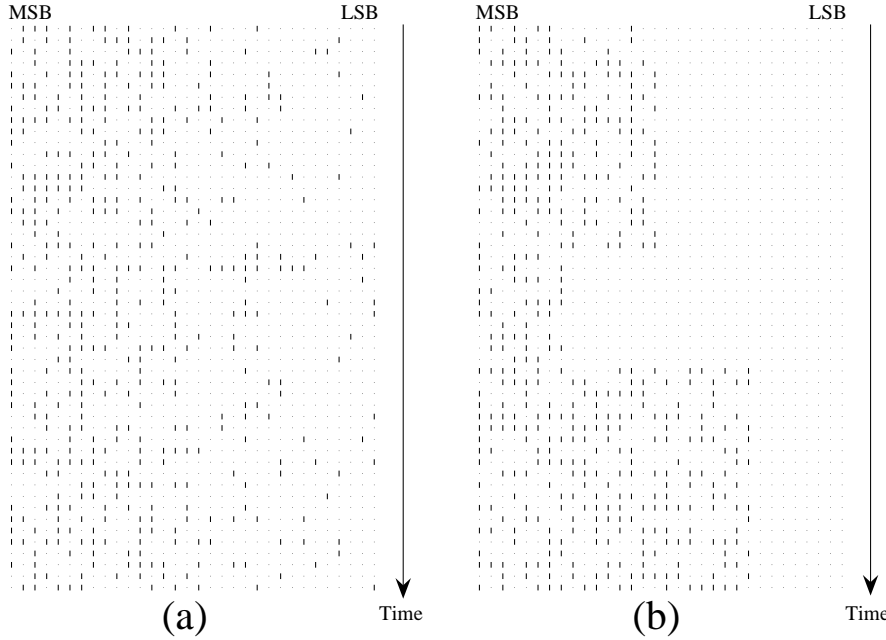


Figure 7.1: Correlated and uncorrelated input pattern

(a) Spatiotemporally uncorrelated input bits pattern (b) Correlated input bits pattern expected in the systems with word-length variation

In the experiments, it is realistically assumed that the changes in word-length occur seldom and therefore, the power consumption due to the actual word-length shift is negligible; i.e., the power consumption can be estimated by estimating the power consumption for various word-lengths separately. The total power consumption P_{tot} is the weighed sum of the estimated power numbers using the word-length probability density matrix \mathcal{W} .

$$\mathbf{P}_{tot} = \sum_{i=1}^M \sum_{j=1}^N \mu_{i,j} \mathbf{P}_{i,j} \quad (7.4)$$

where $\mathbf{P}_{i,j}$ is the estimated power consumption when the $M \times N$ -bit multiplier is operating with $i \times j$ active bits.

The input bits which are forced to zero due to word-length reduction, are all equal and with a high probability will not experience changes in the next clock cycle. This leads to a certain pattern of spatiotemporal correlation between the primary inputs. Figure 7.1(b) depicts such correlated input bits, while Figure 7.1(a)

shows a sequence of uncorrelated inputs. Figures 7.1(b) and 7.1(a) have equal one-probabilities over an infinitely large time interval. The only difference is their spatiotemporal correlations. As will be shown in the experimental results, the power consumption caused by the two input patterns can be quite different. Hence, integrating such correlations in the power estimator and optimization will lead to better solutions compared to using the independence assumption for primary inputs. As discussed in Chapter 5, in order to include the spatiotemporal correlations in the optimization procedure, spatial correlations are captured using a pairwise correlation coefficient matrix between primary inputs, \mathcal{C} (Eq. 5.3). Temporal correlations are captured using a lag-one temporal correlation ratio vector, \mathcal{R} defined in Eq. 5.11. The value of matrix \mathcal{C} and vector \mathcal{R} can be approximated using a large number of inputs with the desired input pattern using Eq. 5.5 and Eq. 5.10. In order to be able to observe the effects of including \mathcal{C} and \mathcal{R} in the optimization, two optimized multipliers are shown in the results; $\mathcal{M}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ is the optimized multiplier without considering correlations between primary inputs, i.e., without including \mathcal{C} and \mathcal{R} . $\widehat{\mathcal{M}}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ is the optimized multiplier with considering correlations between primary inputs, i.e., including \mathcal{C} and \mathcal{R} . Therefore, $\mathcal{M}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ and $\widehat{\mathcal{M}}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ refer to different multipliers, as spatiotemporal correlations between primary inputs are considered for the latter, while it is not considered for the former.

7.2 Experiments

The optimization algorithm, the power estimator and a VHDL generator for the designed multipliers are implemented in C++. This CAD tool generates optimized (and worst-case) multipliers for the given static probabilities, spatial correlation coefficients and temporal correlation ratios. The VHDL code for the generated multiplier structures are simulated using the ModelSim logic simulator from Mentor Graphics and the average number of transitions per clock cycle for all nodes in the multiplier is reported. The average number of transitions per clock cycle is obtained from simulating the circuit for 10000 input samples with the desired pattern and static-probabilities using a fanout-delay model. As the capacitances within the reduction-tree do not have large variations, the average number of transitions gives a good estimate of the dynamic power consumption.

In order to have a fair comparison of the optimization method, the optimized and worst-case multipliers are compared with random multipliers as well. Ten random multipliers are generated for which the permutations of equal-weight PPs are chosen randomly regardless of their transition activities. The average number

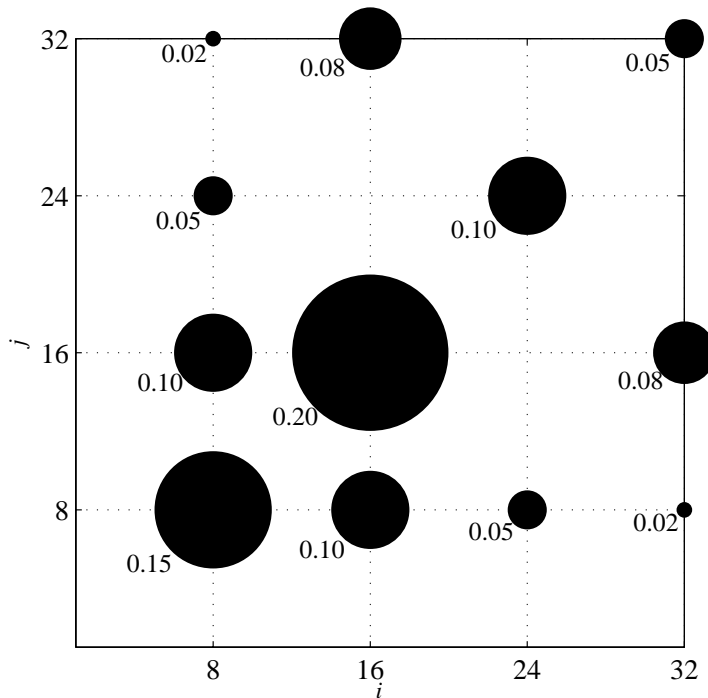


Figure 7.2: Visualization of the word-length probability density matrix (\mathcal{W}) in example 1 using circles' radii

of transitions reported for random multipliers are mean values obtained from these ten random multipliers.

7.2.1 General-Purpose Multiplier

In the first part of the experiments, a parallel multiplier that is operating in a general purpose processor is considered. The programs executed on this processor are controlled by a variety of applications, exploiting the multiplier with different resolutions. By profiling a number of applications, a rough estimate of the word-length probability density matrix can be obtained. As an example, a 32×32 -bit multiplier embedded in a general-purpose processor is considered. The word-length probability density matrix, \mathcal{W} , is visualized in Figure 7.2. The radius of a circle that is centered at location (i, j) is proportional to $\mu_{i,j}$; i.e., the probability of utilizing the multiplier for a $i \times j$ -bit multiplication. The non-zero elements

Table 7.1: Non-zero elements in the word-length probability density matrix (\mathcal{W}) in example 1

$\mu_{8,8}$	0.15	$\mu_{8,16}$	0.10	$\mu_{24,24}$	0.10
$\mu_{16,8}$	0.10	$\mu_{16,16}$	0.20	$\mu_{8,32}$	0.02
$\mu_{24,8}$	0.05	$\mu_{32,16}$	0.08	$\mu_{16,32}$	0.08
$\mu_{32,8}$	0.02	$\mu_{24,8}$	0.05	$\mu_{32,32}$	0.05

Table 7.2: Average number of transitions for different multipliers

Multiplier Structure	Input pattern applied to both operands					
	Ω_{32}^{32}	Ω_{24}^{32}	Ω_{16}^{32}	Ω_8^{32}	\mathbf{P}_{UC}	\mathbf{P}_{VW}
$\mathfrak{M}_{\Omega_{32}, \Omega_{32}}^{OPT}$	<u>6324</u>	3574	1339	228	1728	1737
$\mathfrak{M}_{\Omega_{24}, \Omega_{24}}^{OPT}$	6828	<u>3322</u>	1312	226	1712	1734
$\mathfrak{M}_{\Omega_{16}, \Omega_{16}}^{OPT}$	7109	4053	<u>1250</u>	226	1833	1867
$\mathfrak{M}_{\Omega_8, \Omega_8}^{OPT}$	7182	4084	1512	<u>207</u>	1947	1967
$\mathfrak{M}_{\mathbf{P}, \mathbf{P}}^{OPT}$	6907	3805	1271	217	<u>1693</u>	1781
$\mathfrak{M}_{\mathbf{P}, \mathbf{P}}^{WC}$	7167	4258	1715	307	2249	2132
$\widehat{\mathfrak{M}}_{\mathbf{P}, \mathbf{P}}^{OPT}$	6755	3579	1265	213	1719	<u>1709</u>
$\widehat{\mathfrak{M}}_{\mathbf{P}, \mathbf{P}}^{WC}$	7369	4371	1749	305	2234	2193
$\overline{\mathfrak{M}}^{RND}$	7258	4180	1588	271	2063	2045

of \mathcal{W} are given in Table 7.1. For instance, 20% of the multiplications performed on the multiplier circuit will be 16×16 -bit multiplication, while only 5% of the multiplications will be 32×32 -bit. Using Eq. 7.2 and Eq. 7.3, $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$ can be computed as illustrated in Figure 7.3. The least significant bits have lower average one-probabilities because they are often forced to zero. For the most significant bits, one-probabilities equal to 0.5.

Table 7.2 summarizes the results for this example. Different multiplier structures are placed in different rows of this table. The actual input pattern that is applied to different multiplier structures are shown in different columns. The one-probability vector $\overline{\mathbf{p}}$ is obtained from Eq. 7.2 and Eq. 7.3. The last column denoted as \mathbf{p}_{VW} is the input pattern with a word-length variation similar to that of Figure 7.1(b). The input pattern satisfies the word-length probability distribution shown in Figure 7.2 as well as the spatiotemporal correlations due to minimal word-length variation assumption discussed in Section 7.1. The second rightmost column denoted as \mathbf{p}_{UC} is the uncorrelated input pattern with static probabilities equal to $\overline{\mathbf{p}}$ (Figure 7.1(a)).

The pairwise spatial correlation coefficients and lag-one temporal correlation ratios for primary input bits are obtained from Eq. 5.5 and Eq. 5.10 by generating a large number of random inputs (100000 samples) with the desired pattern, i.e., similar to Figure 7.1(b), for word-length variation. The spatial correlation coefficient matrix for primary input bits (\mathcal{C}) is shown in Figure 7.4 using radii of circles to visualized correlation coefficient values. The lag-one temporal correlation ratio (\mathcal{R}) is illustrated in Figure 7.5. Note that the input patterns \mathbf{p}_{UC} and \mathbf{p}_{VW} are different because the former has uncorrelated random bits (Figure 7.1(a)) while the latter is correlated (Figure 7.1(b)).

The multiplier $\widehat{\mathfrak{M}}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{OPT}$ refers to the multiplier that is optimized for $\overline{\mathbf{p}}$ including the pairwise correlation coefficient matrix (\mathcal{C}) for primary inputs in Figure 7.4 and lag-one temporal correlation ratios (\mathcal{R}) in Figure 7.5. $\mathfrak{M}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{OPT}$ is the optimized multiplier for \mathbf{p}_{UC} . The numbers reported in the last row ($\overline{\mathfrak{M}}^{RND}$) are average number of transitions for ten random multipliers.

From Table 7.2 it can be seen that $\widehat{\mathfrak{M}}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{WC}$ has about 28% more transitions compared to $\widehat{\mathfrak{M}}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{OPT}$ when \mathbf{p}_{VW} is applied to input operands. Compared to randomly interconnected multipliers, the reduction in the average number of transitions is about 16%. Comparing the two multipliers $\widehat{\mathfrak{M}}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{OPT}$ and $\mathfrak{M}_{\overline{\mathbf{p}}, \overline{\mathbf{p}}}^{OPT}$, it can be concluded that considering spatiotemporal correlations for primary inputs, reduces the average number of transitions even further. An important conclusion from Table 7.2 is that if information about the word-length probability density matrix in Figure 7.2

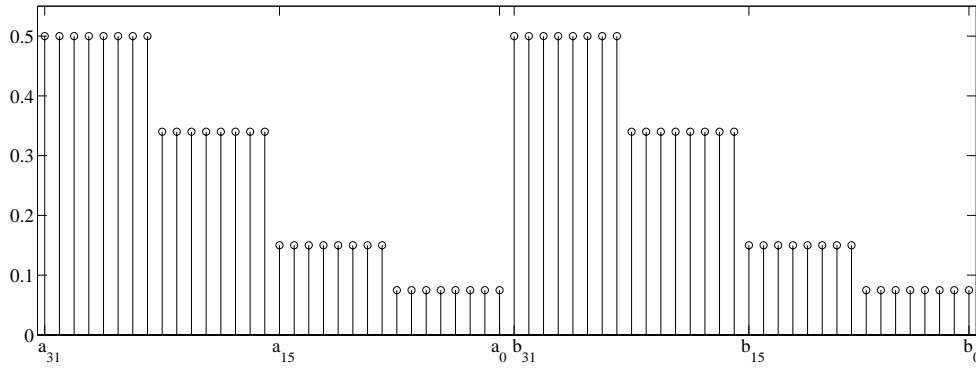


Figure 7.3: One-probabilities for primary input bits in example 1

is not available, even optimizing the multiplier for its full word-length can reduce the power consumption significantly. In this example, $\mathfrak{M}_{\Omega_{32}^{32}, \Omega_{32}^{32}}^{OPT}$ experiences 15% less transitions compared to random multipliers. However, in order to achieve further reduction in power consumption, information about input patterns is necessary including spatiotemporal correlations.

7.2.2 FFT Processor with Variable Word-Length

A relevant case for using adaptive multiplier word-length could be the Fast Fourier Transform (FFT) computation in an Orthogonal Frequency Division Multiplexing (OFDM) receiver. An $M \times N$ -bit multiplier is utilized in the FFT processor. This multiplier computes the product of the data input (input A with M bits) and the twiddle factor (input B with N bits). With a fading channel, the channel noise power experiences large variation in time. Thus, the requirements to quantizing errors in the receiver DSP will also vary with time. For simplicity a constant signal power $\sigma_A^2 = 1$ for each subcarrier at the output of the FFT is assumed. Additionally, a Rayleigh fading channel resulting in a subcarrier noise at the FFT output is assumed. This noise will have a time-varying power σ_C^2 resulting in a subcarrier channel signal to noise ratio (CSNR) γ with the exponential distribution:

$$f_\gamma(\gamma) = \frac{1}{\bar{\gamma}} \exp\left(-\frac{\gamma}{\bar{\gamma}}\right) \quad (7.5)$$

where $\bar{\gamma}$ is the mean value of the random variable γ . Furthermore, the quantization noise due to finite multiplier word-lengths should be negligible in comparison to

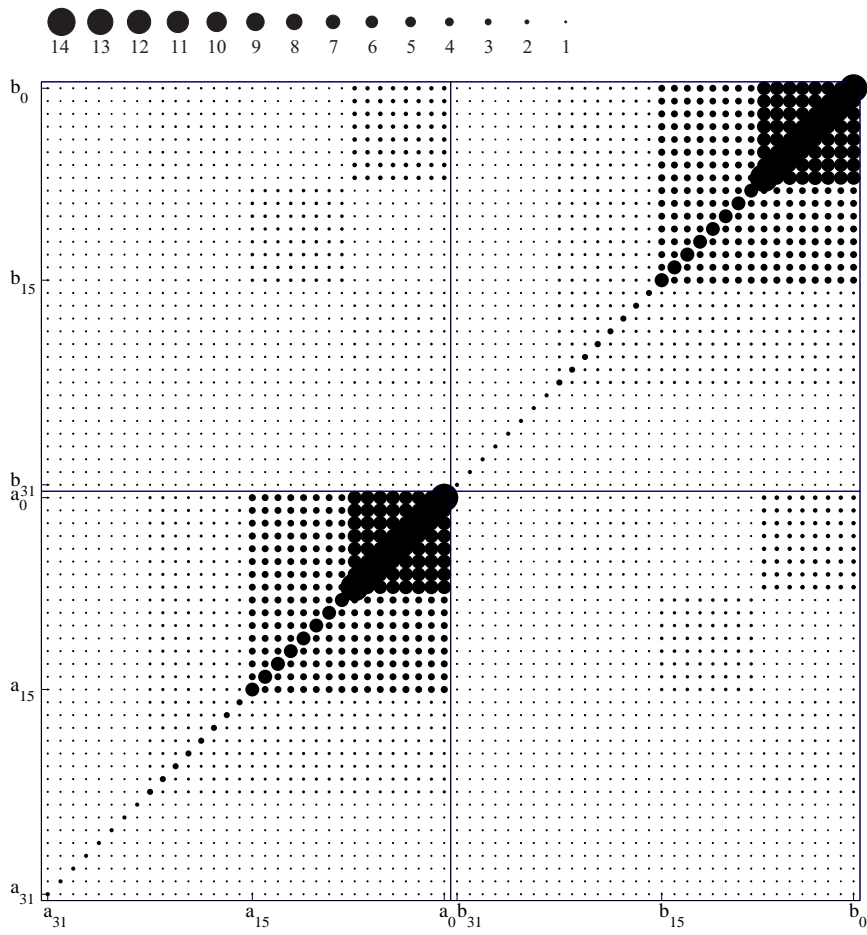


Figure 7.4: Visualization of the correlation coefficient matrix (C) between primary input bits in example 1

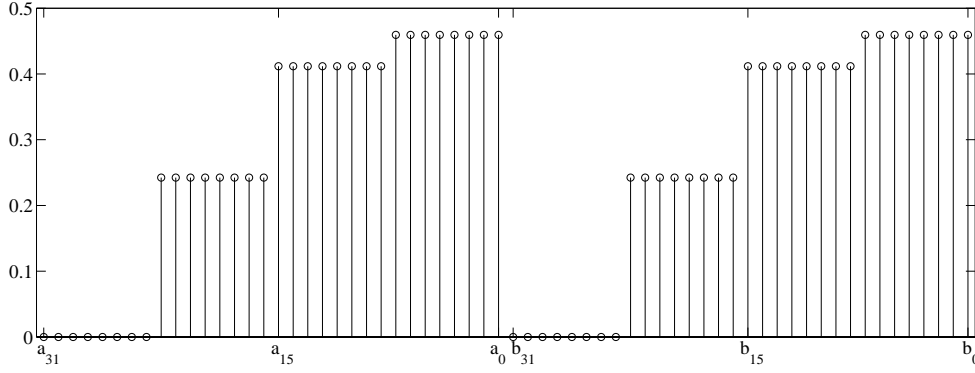


Figure 7.5: Lag-one temporal correlation ratios for primary input bits in example 1

the channel noise. This is ensured by keeping the quantization noise power, σ_Q^2 , 20 dB below the channel noise, i.e.

$$\sigma_Q^2 \leq \frac{1}{100} \sigma_A^2 \quad (7.6)$$

Assuming the model in [149, Section 6.4.2] the quantization power with an M bit data word-length will then be given as

$$\sigma_Q^2 = 2^{1-\nu-2M} \quad (7.7)$$

for an FFT of length 2^ν . The error due to finite twiddle factor word-length is assumed to be negligible compared to σ_A^2 , if $N = M + 2$ bits are used for these values.

Now the target is to find the probability that Eq. 7.6 is fulfilled for the different multiplier sizes. This is indeed the values of the word-length probability density matrix \mathcal{W} . Assuming an FFT of length 2^6 , it is observed that this is equivalent to requiring the CSNR (in dB) to be in a given interval:

$$\mu_{b,(b+2)} = P(41 - 6.02(b+1) < \gamma_{\text{dB}} \leq 41 - 6.02b) \quad (7.8)$$

Assuming a scenario with $\bar{\gamma}_{\text{dB}} = 30\text{dB}$ and using the distribution in Eq. 7.5, the probabilities can be computed as shown in Table 7.3. The elements of matrix \mathcal{W} which are not given in Table 7.3 are zero. The largest multiplier size is chosen to be 15×17 -bit. In order to save power during the operation of the FFT processor, the multiplication size will vary, based on the quality of the channel. For instance,

Table 7.3: Word-length probabilities for an FFT processor with fading channel

$\mu_{5,7}$	0.0001	$\mu_{9,11}$	0.0154	$\mu_{13,15}$	0.2594
$\mu_{6,8}$	0.0002	$\mu_{10,12}$	0.0592	$\mu_{14,16}$	0.0049
$\mu_{7,9}$	0.0010	$\mu_{11,13}$	0.2032	$\mu_{15,17}$	0.0001
$\mu_{8,10}$	0.0039	$\mu_{12,14}$	0.4526		

Table 7.4: Average number of transitions for different multipliers

Multiplier Structure	Applied input pattern								\mathbf{P}_{UC}	\mathbf{P}_{VW}
	$(\Omega_{15}^{15}, \Omega_{17}^{17})$	$(\Omega_{14}^{15}, \Omega_{16}^{17})$	$(\Omega_{13}^{15}, \Omega_{15}^{17})$	$(\Omega_{12}^{15}, \Omega_{14}^{17})$	$(\Omega_{11}^{15}, \Omega_{13}^{17})$	$(\Omega_{10}^{15}, \Omega_{12}^{17})$	$(\Omega_9^{15}, \Omega_{11}^{17})$	$(\Omega_8^{15}, \Omega_{10}^{17})$		
$\mathfrak{M}_{\Omega_{15}^{15}, \Omega_{17}^{17}}^{OPT}$	<u>1158</u>	1052	885	742	610	467	391	304	743	732
$\mathfrak{M}_{\Omega_{14}^{15}, \Omega_{16}^{17}}^{OPT}$	1201	<u>1021</u>	852	740	600	489	391	292	780	719
$\mathfrak{M}_{\Omega_{13}^{15}, \Omega_{15}^{17}}^{OPT}$	1238	1072	<u>844</u>	727	599	489	386	297	804	712
$\mathfrak{M}_{\Omega_{12}^{15}, \Omega_{14}^{17}}^{OPT}$	1262	1112	904	<u>699</u>	601	493	397	308	816	715
$\mathfrak{M}_{\Omega_{11}^{15}, \Omega_{13}^{17}}^{OPT}$	1251	1110	937	758	<u>560</u>	474	377	289	800	741
$\mathfrak{M}_{\Omega_{10}^{15}, \Omega_{12}^{17}}^{OPT}$	1249	1113	959	789	617	<u>443</u>	363	288	811	770
$\mathfrak{M}_{\Omega_9^{15}, \Omega_{11}^{17}}^{OPT}$	1271	1138	985	834	688	520	<u>345</u>	294	834	816
$\mathfrak{M}_{\Omega_8^{15}, \Omega_{10}^{17}}^{OPT}$	1277	1145	975	841	698	568	428	<u>278</u>	832	823
$\widehat{\mathfrak{M}}_{\mathbf{P}_A, \mathbf{P}_B}^{OPT}$	1237	1084	859	725	584	480	385	296	<u>729</u>	711
$\mathfrak{M}_{\mathbf{P}_A, \mathbf{P}_B}^{WC}$	1422	1272	1120	985	828	678	521	406	932	960
$\widehat{\mathfrak{M}}_{\mathbf{P}_A, \mathbf{P}_B}^{OPT}$	1236	1081	848	702	578	478	389	297	780	<u>696</u>
$\widehat{\mathfrak{M}}_{\mathbf{P}_A, \mathbf{P}_B}^{WC}$	1417	1279	1128	988	831	679	542	418	929	966
\mathfrak{M}^{RND}	1359	1218	1046	892	745	610	482	365	887	878

when a 12×14 -bit multiplier is needed three bits from LSB side will be forced to zero on both inputs.

Table 7.4 summarizes the average number of transitions for different multiplier structures. Each row shows a different multiplier structure and each column shows the input pattern that is applied to the multiplier. The rightmost column, \mathbf{p}_{VW} , is the variable word-length input pattern with the probabilities shown in Table 7.3. It is again assumed that word-length transitions do not happen often. The input pattern \mathbf{p}_{UC} shows the uncorrelated random input bits where the input bits have the average one-probabilities equal to $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$ obtained from Eq. 7.2 and Eq. 7.3. $\widehat{\mathfrak{M}}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ is the multiplier that is optimized for $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$, considering the spatiotemporal correlations for primary inputs. $\mathfrak{M}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ is the multiplier that is optimized for $\overline{\mathbf{p}}_A$ and $\overline{\mathbf{p}}_B$, but without considering the spatiotemporal correlations for primary inputs. The lowest number of transitions when \mathbf{p}_{VW} is applied to the multiplier, is found in row $\widehat{\mathfrak{M}}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$ which has 21% and 28% less transitions compared to random multipliers and the worst-case multiplier respectively. The most probable multiplication in this example is 12×14 -bit. Table 7.4 shows that $\mathfrak{M}_{\Omega_{12}^{15}, \Omega_{14}^{17}}^{OPT}$ reduces the number of transitions significantly when \mathbf{p}_{VW} is applied. However, the reduction is less than $\widehat{\mathfrak{M}}_{\mathbf{p}_A, \mathbf{p}_B}^{OPT}$.

Chapter 8

Function Generation using a Weighted Sum of Bit-Products

Multi-operand adder trees are implicit in many application such as multiplication, computation of vector inner products, fused multiply-add operation, recurrences, transformations, and filters. The combinational implementation of multi-operand adders forms a reduction-tree that resembles the PPRT in parallel multipliers. In the previous chapters, the optimization of the reduction-tree structure in parallel multipliers was addressed. The progressive reduction-tree design algorithm introduced in Chapter 4, without major modifications, can be utilized to optimize the reduction-tree of general multi-operand adders. Minor changes might be necessary to generalize the optimization algorithms to arbitrary multi-operand adders. For example, the initial PPs might be clustered in disjoint groups in the general form of multi-operand adders. Figure 8.1 illustrates different types of PP patterns that might appear in a general multi-operand adder. In Figure 8.1(a) the PPs are joint and no intermediate columns are left without PPs. Examples of disjoint PP patterns are illustrated in Figure 8.1(b) and Figure 8.1(c). Figure 8.1(b) is partially disjoint because after a number of FA stages in the reduction-tree the PP pattern will not be disjoint anymore and two clusters of PPs will adhere and form one cluster. Consequently, partially disjoint PP patterns will result in contiguous output vectors. Figure 8.1(c) shows a strictly disjoint PP pattern where even the output vectors are disjoint. The PPRT in parallel multipliers exhibit a joint PP pattern. In order to generalize the optimization algorithms to arbitrary multi-operand adder trees, these algorithms must be able to handle disjoint PP patterns as well. Multi-operand adder trees with strictly disjoint PP patterns may be optimized separately for each cluster.

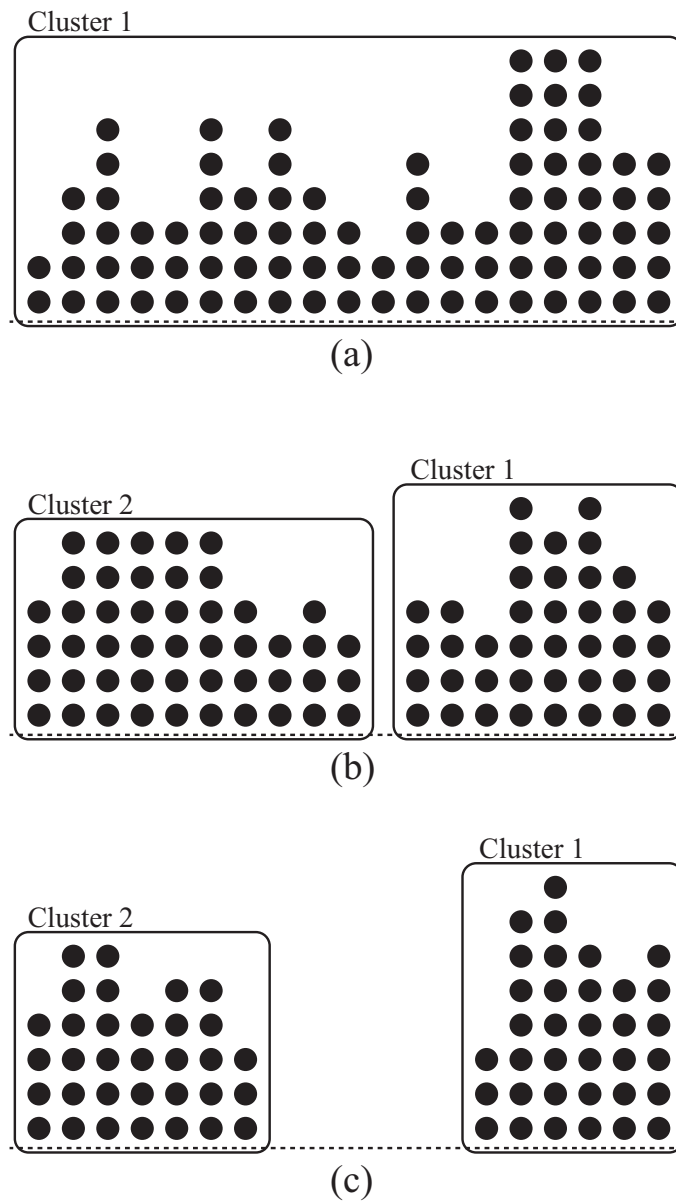


Figure 8.1: Different patterns for partial products in multi-operand adders

- (a) Joint partial product pattern
- (b) Partially disjoint partial product pattern
- (c) Strictly disjoint partial product pattern

In this chapter, application of the progressive reduction-tree design algorithm on a special class of elementary function generators is studied. It is shown that this strategy can result in even larger savings compared to a general multiplier. The results are partially published in [178].

8.1 Elementary Function Generation

This section gives an introduction to the basic techniques for elementary function generation using sum of weighted bit-products. The most commonly used mathematical functions are called elementary functions. Examples of such functions are: \sin , \cos , \tan , \sin^{-1} , \cos^{-1} , \tan^{-1} , \sinh , \cosh , \tanh , \sinh^{-1} , \cosh^{-1} , \tanh^{-1} , exponentials, logarithms, and so on. Working in a finite word-length system necessitates approximation of elementary functions. Approximation of elementary functions finds applications in several areas [126]. For example, in direct digital frequency synthesis (DDFS), logarithmic number systems, transforms such as discrete Fourier transform (DFT), fast Fourier transform (FFT) and discrete cosine transform (DCT), computer graphics, and neural networks. Also it finds applications as seed value generation for Newton-Raphson-based division and square-root computations [50].

Shift-and-add methods are useful approximation techniques which are based on simple elementary steps, addition and shift. The CORDIC algorithm introduced in [192] for example, is a shift-and-add algorithm that can approximate trigonometric and hyperbolic functions. This algorithm was generalized in [194] to compute logarithms, exponentials and square-roots. The method for approximating elementary functions as a weighted sum of bit-products is proposed in [64, 78–80, 196]. The resulting architecture is composed of a number of and-gates and a summation-tree, leading to an architecture that can be easily pipelined to an arbitrary degree. Starting with computation of sine and cosine, an approach based on trigonometric identities was proposed [196]. In [64], an optimization procedure was outlined and the effect of finite word-length was studied. Then, [78] presented a modified architecture that turns off parts of the summation-tree to reduce the power consumption. The proposed method was generalized to arbitrary elementary functions in [79]. In [80], the application of the proposed method to logarithmic number systems was studied. Here, some modifications to produce better results for certain functions were proposed. Although the resulting architecture has similarities with those proposed in [44, 112, 165, 169], a completely different technique is used to derive the approximation. The proposed approxima-

tion method is not limited by the behavior of the approximated function but can be used for all possible functions. However, the complexity depends on the function, for example, the complexity would be high for an irregular function composed by random values. Furthermore, the method in [69] has some similarities, given that the ROMs in [69] are only using one input-bit each.

8.1.1 Approximation using a Weighted Sum of Bit-Products

A function, $f(X)$, where X is an integer composed of N bits, x_i ($0 \leq i < N$), can be represented as

$$f(X) = \sum_{j=0}^{2^N-1} c_j p_j \quad (8.1)$$

where c_j is the weight and p_j is a bit-product [79]. Each bit-product, p_j , is composed of the bits x_i that are one in the binary representation of j , hence

$$p_j = \begin{cases} 1 & j = 0 \\ \prod_{i \in S_j} x_i & j > 0 \end{cases} \quad (8.2)$$

where

$$\sum_{i \in S_j} 2^i = j \quad (8.3)$$

Note that the bit-products can be computed using simple logic AND-operations. For example, for $j = 25$, $S_{25} = \{4, 3, 0\}$ and $p_{25} = x_4 x_3 x_0$, which corresponds to a 3-input AND gate.

The weights, c_j , which in the following also are referred to as coefficients, are computed by vector multiplications according to

$$c_j = q_j F \quad (8.4)$$

where F is a column vector containing all function values. The row vector q_j is the j :th row of the lower triangular matrix Q_N , which is obtained by

$$Q_i = \begin{cases} 1 & i = 0 \\ \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \otimes Q_{i-1} & 1 \leq i \leq N \end{cases} \quad (8.5)$$

where the \otimes operation denotes the Kronecker product. If U is an $m \times n$ matrix and V is a $p \times q$ matrix, then the Kronecker product $U \otimes V$ is the $mp \times nq$ block

Table 8.1: Coefficient values in terms of function values

c_0	$f(0)$
c_1	$-f(0) + f(1)$
c_2	$-f(0) + f(2)$
c_3	$f(0) - f(1) - f(2) + f(3)$
c_4	$-f(0) + f(4)$
c_5	$f(0) - f(1) - f(4) + f(5)$
c_6	$f(0) - f(2) - f(4) + f(6)$
c_7	$-f(0) + f(1) + f(2) - f(3) + f(4) - f(5) - f(6) + f(7)$
c_8	$-f(0) + f(8)$
c_9	$f(0) - f(1) - f(8) + f(9)$
c_{10}	$f(0) - f(2) - f(8) + f(10)$
\vdots	\vdots

matrix:

$$U \otimes V = \begin{bmatrix} u_{11}V & \cdots & u_{1n}V \\ \vdots & \ddots & \vdots \\ u_{m1}V & \cdots & u_{mn}V \end{bmatrix}. \quad (8.6)$$

or more explicitly,

$$U \otimes V = \begin{bmatrix} u_{11}v_{11} & u_{11}v_{12} & \cdots & u_{11}v_{1q} & \cdots & \cdots & u_{1n}v_{11} & u_{1n}v_{12} & \cdots & u_{1n}v_{1q} \\ u_{11}v_{21} & u_{11}v_{22} & \cdots & u_{11}v_{2q} & \cdots & \cdots & u_{1n}v_{21} & u_{1n}v_{22} & \cdots & u_{1n}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ u_{11}v_{p1} & u_{11}v_{p2} & \cdots & u_{11}v_{pq} & \cdots & \cdots & u_{1n}v_{p1} & u_{1n}v_{p2} & \cdots & u_{1n}v_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ u_{m1}v_{11} & u_{m1}v_{12} & \cdots & u_{m1}v_{1q} & \cdots & \cdots & u_{mn}v_{11} & u_{mn}v_{12} & \cdots & u_{mn}v_{1q} \\ u_{m1}v_{21} & u_{m1}v_{22} & \cdots & u_{m1}v_{2q} & \cdots & \cdots & u_{mn}v_{21} & u_{mn}v_{22} & \cdots & u_{mn}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ u_{m1}v_{p1} & u_{m1}v_{p2} & \cdots & u_{m1}v_{pq} & \cdots & \cdots & u_{mn}v_{p1} & u_{mn}v_{p2} & \cdots & u_{mn}v_{pq} \end{bmatrix} \quad (8.7)$$

Using Eq. 8.4, the coefficients are directly obtained from the function values. Some examples are shown in Table 8.1. For functions with certain properties, basically all continuous functions, many of the coefficients, c_j , will be small, i.e., truncated to zero for a limited coefficient word-length. Hence, the function $f(X)$

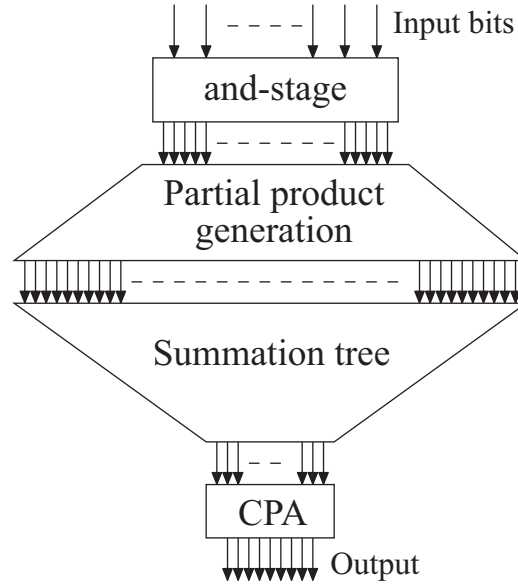


Figure 8.2: Architecture used for approximating elementary functions

can be approximated with $\hat{f}(X)$ which is defined as:

$$\hat{f}(X) = \sum_{j \in G} c_j p_j \quad (8.8)$$

where G is the set of indices for the M coefficients with largest absolute value; i.e. $|G| = M$ and $\min_{j \in G} \{|c_j|\} \geq \max_{j \notin G} \{|c_j|\}$.

8.1.2 Architecture

To implement the expression in Eq. 8.8, an effective architecture, suitable for high-speed implementations, was proposed in [196]. The architecture is illustrated in Figure 8.2, where the and-stage is used to compute the required bit-products, p_j .

The partial product generation stage only shifts and possibly inverts the outputs of the and-stage, according to the corresponding coefficients, c_j . The bit-products are included in the columns corresponding to nonzero digits in the coefficient representation. Hence, by using a minimum signed digit (MSD) representation, the number of partial products to be added is decreased.

In the third stage, a summation-tree accumulates the partial products. The reduction of partial products can be performed in logarithmic time using redun-

dant multi-operand adders with redundant outputs in carry-save form. Several reduction schemes can be chosen for the summation-tree as discussed in Chapter 2. Similar to the multiplier PPRT in the previous chapters, the Modified Wallace/Dadda reduction scheme proposed by Bickerstaff et al. in [12] is utilized for the reported results. This reduction scheme is discussed in detail in Section 2.2.2.4. Finally, the last stage is a carry propagation adder (CPA) used to form a non-redundant representation of the output.

Note that the architecture can easily be pipelined to an arbitrary degree to obtain the desired throughput. In [78], it is shown that the power consumption can be decreased if the architecture is divided into multiple summation trees. For example, all bit-products that include the input bit x_i can be separated into a summation tree that is turned off when x_i is zero. In this chapter pure combinational architecture is considered, but the proposed method can be applied to each summation-tree in the modified architecture. So these two power reduction methods do not contradict each other, but rather complement each other.

8.2 Optimized Summation-Tree Generation

The algorithm for the progressive reduction tree design is summarized in Figure 8.3. The algorithm `generate_summation_tree()` is executed after the generation of the primary partial products pattern. The function generator circuit, denoted as `TheCircuit`, is initialized to contain the AND-terms and the primary partial products generation step as described in Section 8.1.1. The function `compute_SWSs(TheCircuit)` estimates the power consumption using the simple waveform set (SWS) method which is a probabilistic gate-level power estimator described in Section 3.1.1. The power estimator computes a simple waveform set which is a set of possible transition times and their occurrence probabilities for each node of the circuit. The computation of the SWS for a node is based on the SWSs at the predecessor nodes; i.e. the computation starts from primary inputs and traverses towards outputs. Note that each time a circuit portion is appended to `TheCircuit`, this function is executed, computing the power estimation for the new portion using the estimated transition sets of the predecessor nodes. After the PPs are generated, the summation-tree is progressively designed. Similar to the progressive reduction-tree design algorithm in Section 4.3, for each column of each stage in the summation-tree a local search is performed. The power estimator in the innermost loop, `estimate_power_SWS`, computes the transition densities exclusively for the current FA/HA stage, i.e. the stage that is being designed, using

```

generate_summation_tree()
Inputs: static probabilities of input bits and primary PP generation pattern
{
  TheCircuit= $\emptyset$ ;
  [ $M_0$ , PPG_circuitry]=generate_PPs(primary PP generation pattern);
  TheCircuit.append(PPG_circuitry);
  computeSWS(TheCircuit);
   $i=0$ ;
  while ( $\|M_i\| > 2$ )
  {
    [ $F_i$ ,  $H_i$ ]=reduction_scheme( $M_i$ );
     $M_{i+1}$ =apply_FA_HA( $M_i$ ,  $F_i$ ,  $H_i$ );
    for  $current\_col=1$  to number of columns in  $M_i$  do
    {
       $current\_perm$ =random permutation of the PPs in  $current\_col$  of  $M_i$ ;
       $P_{min}=+\infty$ ;
       $\alpha=\alpha_0$ ;
      for  $current\_iteration=1$  to  $number\_of\_iterations$  do
      {
         $old\_perm=current\_perm$ ;
        swap two random positions in  $current\_perm$ ;
         $P$ =estimate_power_SWS( $F_i$ ,  $H_i$ ,  $current\_perm$ );
        if ( $P < P_{min}$ )
           $P_{min}=P$ ;
        else
        {
           $r$ =uniform random ( $r \in [0, 1)$ );
          if ( $r < e^{-\frac{P_{min}-P}{\alpha}}$ )
             $P_{min}=P$ ;
          else
             $current\_perm=old\_perm$ ;
        }
         $\alpha=\frac{\alpha}{\tau}$ ;
      }
      reorder  $current\_col$  of  $M_i$  as  $current\_perm$ ;
    }
    TheCircuit.append( $M_i$ ,  $F_i$ ,  $H_i$ );
    computeSWS(TheCircuit);
     $i=i+1$ ;
  }
  TheCircuit.append(final_CPA( $M_i$ ));
  computeSWS(TheCircuit);
}

```

Figure 8.3: Progressive reduction-tree design algorithm for function generators

the SWSs computed earlier. This power estimation is limited to few logic gates and is hence fast; making it feasible to perform a large number of iterations. As discussed in Chapter 4, the power estimates can be updated in each SA iteration by recalculating the SWSs for maximum ten logic gates, merely the ones that have changed in the current configuration.

In this chapter, it is assumed that primary input bits are uncorrelated. If this assumption is not valid for the primary input bits, the spatiotemporal correlations must be considered using the pairwise correlation coefficient matrix and lag-one temporal correlation ratios as discussed in Chapter 5.

The algorithm `generate_summation_tree()` is implemented in C++. Inputs to the optimization algorithm are the static probabilities of the primary inputs, the partial product generation pattern and eventually the spatiotemporal correlations of the primary inputs. The power estimation in this algorithm uses a fanout delay model for its computations; i.e. the delay of a logic gate, regardless of its function, is equal to its number of fanouts and is an integer number. If needed, a more complex delay model can also be implemented. After completion of the optimization algorithm, the equivalent VHDL code for the circuit is generated. This algorithm constructs an optimum summation-tree such that it consumes less energy while operating. A similar algorithm can be developed by altering `generate_summation_tree()` to maximize the energy consumption and generate a worst-case summation tree. The power consumption of the optimum and worst-case summation trees is compared in Section 8.3. Designing the summation-tree randomly or regardless of the transition activities of partial products results in a power consumption value that lies somewhere between optimum and worst-case values. In fact the experiments show that the power consumption of random summation trees is biased towards the worst-case values.

8.3 Experiments

In this section the proposed method is applied to a number of elementary functions (Table 8.2). The detailed definitions of these functions, except SINE and COSINE, are given in [80]. The number in the different circuit names correspond to the number of coefficients (after optimization some coefficients are truncated to zero so the number of bit-products used might be a few less). The primary PP patterns for the COSINE and SINE functions are illustrated Figure 8.4. After the generation of the partial products, the algorithm `generate_summation_tree()` is executed for each example. The primary inputs of all circuits are assumed to be

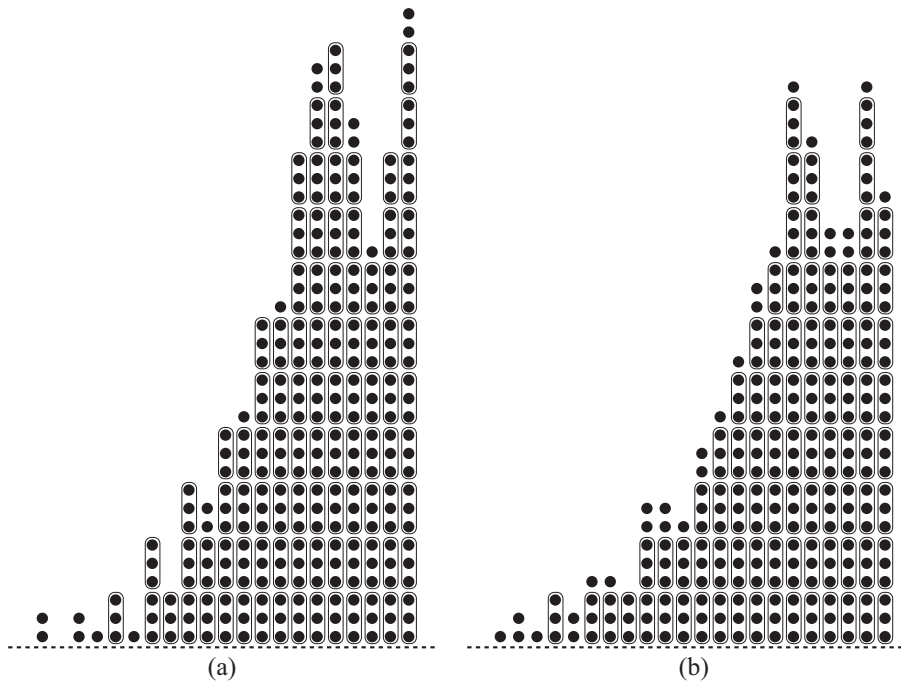


Figure 8.4: The primary PP pattern for the (a) COSINE and (b) SINE functions with 91 and 89 coefficients respectively

Table 8.2: Function definitions

Circuit	$f(X)$	Range of $f(X)$	# inputs	# coefs.	Coef. wordlength	Accuracy [bits]
SINE	$\sin(\pi X/4)$	$0 \leq f(X) \leq 1/\sqrt{2}$	9	89	20	16.00
COSINE	$\cos(\pi X/4)$	$1/\sqrt{2} \leq f(X) \leq 1$	9	91	20	16.00
LOG2-100	$\log_2 X $	$-8 \leq f(X) < 0$	8	99	11	9.07
LOG2-149	$\log_2 X $	$-8 \leq f(X) < 0$	8	149	14	12.00
Φ^- -227	$\log_2 1 - 2^X $	$\log_2 1 - 2^{-2^{-8}} \leq f(X) < 0$	11	227	12	9.15
Φ^- -239	$\log_2 1 - 2^X $	$\log_2 1 - 2^{-2^{-8}} \leq f(X) < 0$	11	229	11	9.00
Φ^- -400	$\log_2 1 - 2^X $	$\log_2 1 - 2^{-2^{-8}} \leq f(X) < 0$	11	394	15	12.15
Φ^+ -70	$\log_2 1 + 2^X $	$0 < f(X) \leq 1$	11	70	13	9.01
Φ^+ -82	$\log_2 1 + 2^X $	$0 < f(X) \leq 1$	11	79	11	9.26
Φ^+ -173	$\log_2 1 + 2^X $	$0 < f(X) \leq 1$	11	169	14	12.12

uncorrelated and random with uniform distribution. The one-probabilities of the primary input bits are set to be 0.5. The VHDL code resulting from the optimization algorithm is compiled for a $0.35\mu\text{m}$ standard cell CMOS library. The power consumption is then estimated using Synopsys NanoSim, which is an accurate circuit simulator featuring common HSPICE models for greater accuracy.

The results of the power estimation are given in Table 8.4. In addition to the accurate power consumption values obtained from NanoSim, the average transition activities of the circuits are reported in Table 8.3. The average transition activities in Table 8.3 are obtained by simulating the resulting VHDL codes in ModelSim using a fanout delay model. Note that the optimization algorithm uses the fanout delay model.

In order to evaluate the quality of the optimized circuits, the results are compared with the worst-case circuits as well as the random circuits (the fourth columns in Tables 8.3 and 8.4). For each circuit, ten versions are generated, where the interconnection between the full-adders and half-adders are chosen randomly without considering the transition activities. Subsequently, the average transition activities for these random circuit versions as well as the mean value of the average transition activities are estimated. In the tables, the results of the circuit version

that has an average transition activity closest to the mean value is presented. The rightmost columns in Tables 8.3 and 8.4 are the reduction in power consumption compared to the random circuits. Comparing the values in Tables 8.3 and 8.4, it can be concluded that the random circuits are closer to worst-case circuits in terms of average transition activities and power consumption.

In some cases the random circuits are estimated to consume more power than the worst-case circuits because of the differences in the power estimator that is embedded in the optimization algorithm and NanoSim. The power estimation that is used in optimization algorithm to generate the optimized and worst-case circuits is at gate-level and uses fanout delay model; while NanoSim is an accurate transistor-level analog power estimator with a real delay computation mechanism based on the actual node capacitances and driving capabilities of the transistors. For example, NanoSim is capable of accounting for glitches that are not full-swing, something that the power estimator in the optimization program is incapable of. From Table 8.4 the power consumption for the optimized circuits is 3–29% lower than the worst-case. The gain is even larger when the average transition activities are compared in Table 8.3 (29–40%). This shows the capacity of the optimization method. Larger improvements in power consumption can be achieved if a more accurate delay model is used in the embedded power estimator.

The differences in improvements between the circuits can be explained by looking at the bit-products. For Φ^+ -82, most bit-products are short, which limits the possibilities to obtain switching activity reductions as most bit-products have similar properties. For Φ^- -400, there are 224 bit-products with more than 4 input bits and 24 with more than 8 input bits, hence, a lot of different interconnect choices can be made. Moreover, having a larger number of bit-products increases the possibilities for improvements. Even more important than studying long bit-products is probably the short ones. The circuits SINE, COSINE, and Φ^+ -82 have almost the same number of bit-products. However, looking at the single bit products, i.e., the bit-products which are only dependent on a single input bit, SINE has all 10 input bits as bit products, COSINE has 8, and Φ^+ -82 has only 5. The single bit products have highest switching activities because they are only dependent on one bit. Hence, largest savings are obtained for SINE and smallest for Φ^+ -82. The improvement in power consumption compared to the random circuits is in the range of 5.4% to 25.6%.

Table 8.3: Average transition activity per clock cycle

Circuit	Optimized	Worst-Case	Random	Improvement
SINE	1334.2	2023.0	1807.5	26.2%
COSINE	1525.5	2204.7	1976.9	22.8%
LOG2-100	1029.4	1660.8	1477.9	30.4%
LOG2-149	1889.1	3145.4	2777.8	32.0%
Φ^- -227	1871.1	2961.8	2691.6	30.5%
Φ^- -239	1487.0	2393.5	2290.1	35.1%
Φ^- -400	3484.5	5739.4	5278.5	34.0%
Φ^+ -70	596.2	889.0	820.2	27.3%
Φ^+ -82	548.1	775.3	700.1	21.7%
Φ^+ -173	1400.4	2036.9	1869.9	25.1%

Table 8.4: Estimated energy per operation from NanoSim [pJ]

Circuit	Optimized	Worst-Case	Random	Improvement
SINE	214.0	232.3	230.9	7.3%
COSINE	181.1	210.0	209.5	13.6%
LOG2-100	149.8	164.1	180.2	16.9%
LOG2-149	247.8	290.8	295.2	16.1%
Φ^- -227	271.8	357.3	364.8	25.6%
Φ^- -239	222.4	265.2	278.4	20.1%
Φ^- -400	497.2	695.5	662.0	25.0%
Φ^+ -70	105.6	130.4	128.1	17.6%
Φ^+ -82	103.7	106.5	109.6	5.4%
Φ^+ -173	231.0	280.2	285.8	19.2%

Chapter 9

Conclusions

9.1 Low-Transition Reduction-Tree Generation

This thesis presents a number of methods for generation of power optimized PPRT in parallel multiplier. Energy saving is achieved without any noticeable area or speed overhead compared to random reduction trees. Chapter 4 presented two methods for optimizing the PPRT. The first method optimizes the complete PPRT; i.e., the power estimates are for the complete PPRT and a complete search is performed among a selection of interesting solutions. The second method is the progressive PPRT design algorithm, which limits the power estimation to one FA/HA stage. This method uses simulated annealing as search algorithm. Within the search loop, the power estimates are updated by recomputing the power estimates for a minimum number of gates for which the inputs are altered. The progressive reduction tree design algorithm employs a probabilistic gate-level power estimation to compute cost functions.

Automatically generated VHDL codes for the resulting multipliers are simulated using ModelSim and the average number of transitions are reported. The average number of transitions is reduced significantly compared to random multipliers. Generally, the improvements are larger if the input pattern space is more limited, for example, due to high correlation between input bits or large number of low activity input bits.

The progressive reduction tree design algorithm is further improved to include the spatiotemporal correlations of the primary inputs in Chapter 5. By profiling a large input data stream with the desired properties, the required parameters can be extracted as discussed in Chapter 5. These parameters are one-probabilities, lag-

one temporal correlation ratios, and pairwise correlation coefficients for primary input bits. Several examples of multipliers with correlated inputs are examined in Chapters 5 and 7. Including the spatiotemporal correlations in the power estimator improves the generated results considerably. From the experiments in Chapter 7, for the multipliers with variable word-length, the average number of transitions is reduced 16–21% in the optimized multipliers, compared to random multipliers.

9.2 Low-Leakage Reduction-Tree Generation

By changing the cost function in the progressive PPRT design algorithm in Chapter 4, this algorithm can be used to generate a low-leakage PPRT as well. In this case, a leakage power estimator is needed to be utilized instead of the dynamic power estimator. The reduction in the total leakage current is achieved by reordering the interconnects between full-adders/half-adders, and hence optimizing the partial product reduction tree with respect to the distribution of static probabilities along the tree. The static probabilities are affected by the input characteristics and their spatial correlations. The experiments in Chapter 6 show that improvements about 18% can be expected in static power dissipation relative to a worst-case reduction-tree multiplier.

An important observation from the experiments in Chapters 4 and 6 is that the optimization for static power does not alter the speed and dynamic power noticeably, compared to random multipliers. In other words, the optimization for static power is orthogonal with the optimization for dynamic power and speed. Based on this observation, both static and dynamic power estimators can be employed in the optimization algorithm for reducing the total power in the PPRTs, as the total power is the sum of static power and dynamic power. The cost function is altered to a weighted sum of static power estimation and dynamic power estimation. The experiments in Chapter 6 shows that the total power can be reduced considerably using this optimization method.

9.3 Generalized Multi-Operand Adders

The progressive PPRT design algorithm can be generalized to work on arbitrary multi-operand adders. As an example application of this algorithm on arbitrary multi-operand adders, which is not in the context of multipliers, a special class of function generators are considered in Chapter 8. This class of function gener-

ators use a weighted sum of bit-products to generate elementary functions. According to the experiments on a number of function generators, by progressively constructing the summation tree, power savings in the range of 5% to 26% are achieved compared to a random interconnect ordering.

The one-probabilities of the primary bit-products and their correlations have larger variations in the function generators, compared to the multiplier PPRT. Therefore, the progressive reduction-tree design algorithm has larger potential to improve the reduction tree for function generators compared to multipliers.

9.4 Probabilistic Gate-Level Power Estimator

As a part of the contributions of this thesis, the simple waveform set power estimator is introduced in Chapter 3. The previous methods of probabilistic power estimation have weak modeling of glitch filtering. The set of simple waveforms technique is capable of accounting for the successor nodes' behavior with respect to introduced glitches. The estimations for a number of benchmark circuits are compared to the estimations obtained from logic simulations. Error-free estimations are obtained for tree structured circuits. Significant improvements in the power estimation of general circuits are achieved compared to earlier techniques.

9.5 Directions for Future Work

9.5.1 Power Estimation with Realistic Delay Models

At this point, the most important task for future work seems to be implementation of a more accurate and realistic delay model in the power estimator. As discussed earlier, the computational complexity may increase a by accommodating real numbers as gate delays. In order to achieve a more accurate power estimator, it may be also necessary to differentiate delay values as a function of input values for each logic gates.

It is important to note that the optimization algorithms introduced in this thesis are to a large extent independent from the power estimator. Therefore any power estimation technique may be engaged in this optimization tool as long as it produces the correct cost functions for the optimizer. However, the power estimator must be able to include spatiotemporal correlations, as it is shown to have great importance in the optimization.

9.5.2 Including Speed and Interconnect Power

As discussed earlier, the arithmetically equivalent implementations of the PPRT exhibit different temporal behavior, dynamic power consumption, leakage currents and overall interconnect wire length. In the optimization methods discussed in this thesis, the weighted sum of leakage power and dynamic power is used as the cost function for the optimization algorithms. As a suggestion for the future work, timing and interconnect power can be included in the cost function as well. This cost function can prioritize different measures based on the application and designer's demand with different weights for speed, dynamic power, static power, and interconnect power. The interconnect power for parallel multipliers is analyzed in [31].

9.5.3 Different PP Generation and Reduction Schemes

In this thesis, relatively simple partial product generation methods are considered. That is, for unsigned multiplications, simple AND operation between input bits and, for signed multiplications, modified Baugh-Wooley method [6,70]. The modified Wallace/Dadda PPRT [12] is applied as the partial product reduction scheme. As a future work, the improvements can be analyzed for other partial product generation and reduction schemes. For full-adder based reduction schemes, the improvements are expected to be similar. However, for other multiplier structure setup, for example reduction schemes based on generalized parallel counters, or PP generation schemes based on Booth recoding or higher radix the improvements may be different.

9.5.4 Other Search Algorithms

The Simulated Annealing is used in the progressive reduction-tree design algorithm. Another search algorithm that might be useful instead of SA is a Genetic Algorithm (GA). GA performs well in presence of local minima, which is desirable for the progressive reduction-tree design algorithm.

9.5.5 Pipelining

In this thesis, pipelining is not considered and the structure of the PPRT is assumed to consist of pure combinational logic. As a future work, pipelining can be considered in the optimization algorithms. Pipelining is an effective method

to reduce the glitches in a parallel multiplier, as the spurious transitions are suppressed in register layers introduced because of pipelining. It might be possible to develop algorithms that optimize each layer of pipelines individually and independent from other layers.

9.5.6 Optimization of Synthesized Designs

A gate-level description of arithmetic units consists of logic gates operating on single bits. Extracting the word-level and arithmetic relations between these bit-level signals is a difficult task [22]. As a suggestion for future, extracting the arithmetic equivalencies for a given gate-level description of arithmetic circuits can be considered. The target will be, for example, to create a list of nodes having outputs that can be interchanged without changing the arithmetic behavior of the circuit. Having such lists, the optimization algorithms in this thesis can be applied to a wider range of circuits and applications.

Appendix A

Computation of static probabilities

Computation of static probabilities of combinational logic networks finds application, among others, in probabilistic power estimation, testability and fault detection and reliability. Therefore computing signal probabilities has attracted much attention. The problem of computing the static probabilities represents a #P-complete problem that is conjectured to be even harder than the NP-complete class [95]. The major difficulty in computing the signal probabilities is dependencies caused by reconvergent fan-out (RFO). Earlier works in computing the static probabilities in a combinational network include [18, 27, 52, 95, 142, 163, 167]. The approximation method in [18, 142] ignores dependencies and approximates the joint probabilities with the product of individual probabilities, which results in an algorithm that can compute static probabilities in linear time. This algorithm is error-free for RFO-free circuits, but it becomes inaccurate in presence of RFOs.

Exact method for computation of static probabilities based on Reduced Ordered Binary Decision Diagrams (ROBDDs) [21] are introduced in [26,94]. However, the worst-case complexity of graph size in this method is exponential with number of circuit inputs and thus it is not practical for many circuits. An ROBDD is a reduced functional representation in which every directed path starts from the root and ends at a ZERO or ONE terminal. The underlying decomposition for ROBDDs is the Shannon decomposition in Eq. A.1:

$$Y = XY|_{X=1} + \bar{X}Y|_{X=0} \quad (\text{A.1})$$

An efficient method for implementing ROBDDs is presented in [174]. If the full ROBDD is known, then exact static probabilities can be computed by counting the paths that go to the terminal ONE [204]. The ROBDDs represent the global function in terms of circuit inputs. Approximation techniques are proposed by

localizing the ROBDDs by expressing the function in terms of intermediate variables other than circuit inputs. An example of local binary decision diagrams are presented in [82].

Bayesian Networks have also been employed as a means for estimating the static probabilities in a logic network [155]. Bayesian networks are graphical model representing a set of variables and their conditional independencies deduced from Bayes's Theorem. Implementing combinational logic using graphical probabilistic models is easy because the conditional independencies are easy to derive for combinational logic networks. All logic gates are described as conditional probability distribution tables which are the interactions between the output node of the gate and its immediate predecessors. Probabilistic networks are capable of computing exact static probabilities. The computation process involves moralization, triangulation and identifying cliques and creating clique tree [37]. Moralization is creating an undirected graph from the directed acyclic graph (DAG) representation of the circuit by removing the directions and adding edges between any two nodes which share a common child node. Triangulated or chordal graph is obtained from moral graph by adding edges to remove all cycles of length ≥ 4 . RFOs in the logic network directly translate to larger clique sizes in the graphical model. The complexity of Bayesian networks is exponential with maximum clique size. Hence Bayesian networks are too impractical for circuits with large number of RFOs. Approximation techniques for Bayesian networks are developed [29, 203]. Yuan and Druzdzel in [203] propose an approximation technique for Bayesian networks using importance sampling and evidence pre-propagation.

For the purpose of dynamic and static power estimation (Chapter 3), several static probability computation methods were tested including BDDs, Bayesian networks. Considering computation complexity, accuracy and ease of manipulation to encapsulate spatial correlations of primary inputs, the method based on pairwise correlation coefficients in [52] suits best for this application. The remainder of this chapter describes pairwise correlation coefficients.

A.1 Pairwise Correlation Coefficients

The pairwise correlation coefficients are used in the SWS power estimator and static power estimator in Chapter 3. The exact computation of the pairwise correlation coefficients can be difficult as discussed earlier. Ercolani et al. in [52] an efficient method for approximating the pairwise correlation coefficients under

zero-delay model. The pairwise correlation coefficients are defined as:

$$\kappa_{A,B}^{a,b} = \frac{p(A = a \wedge B = b)}{p(A = a)p(B = b)} \quad (\text{A.2})$$

where A and B are two logic nodes in the circuit and a and b are two logic values ($a, b \in \{0, 1\}$). For simplicity $\kappa_{A,B}^{1,1}$ is denoted as $\kappa_{A,B}$ and $p(A = 1)$ and $p(B = 1)$ are denote as p_A and p_B respectively. Using Eqs. A.3-A.5, correlation coefficients $\kappa_{A,B}^{0,0}$, $\kappa_{A,B}^{1,0}$ and $\kappa_{A,B}^{0,1}$ can be rewritten in terms of $\kappa_{A,B}$.

$$\kappa_{A,B}^{1,0} = \frac{1 - p_B \kappa_{A,B}}{1 - p_B} \quad (\text{A.3})$$

$$\kappa_{A,B}^{0,1} = \frac{1 - p_A \kappa_{A,B}}{1 - p_A} \quad (\text{A.4})$$

$$\kappa_{A,B}^{0,0} = \frac{1 - p_A - p_B + p_A p_B \kappa_{A,B}}{(1 - p_A)(1 - p_B)} \quad (\text{A.5})$$

From the probability theory, two obvious formulas hold for the correlation coefficient. First, the correlation coefficient between two independent signals is equal to 1. Second, the autocorrelation coefficient for a signal is equal to inverse of the static probability of the signal; i.e. $\kappa_{AA} = \frac{1}{p_A}$.

Throughout the computation of static probabilities from inputs towards outputs using correlation coefficients, higher orders of joint probabilities will be encountered. In [52] the higher order joint probabilities are approximated using the pairwise correlation coefficients. For example, the correlation coefficient among three signals A , B and C is approximated as:

$$\kappa_{A,B,C} = \frac{p(A = 1 \wedge B = 1 \wedge C = 1)}{p_A p_B p_C} \approx \kappa_{A,B} \kappa_{B,C} \kappa_{C,A} \quad (\text{A.6})$$

Although the approximation (A.6) introduces errors in the computed static probabilities, the error is negligible for most circuits [52].

The static probability at the output of a logic gate can be written in terms of input static probabilities and the correlation coefficients. For example, an OR gate produces a one at the output when at least one of the inputs are one. Then the

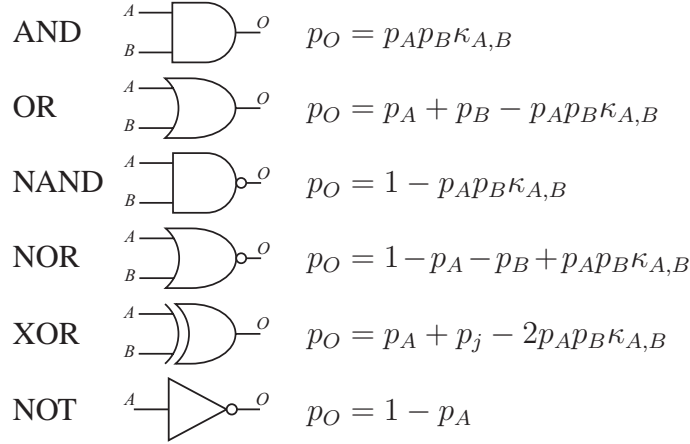


Figure A.1: Calculating output one-probability of the basic logic gates

one-probability of a two-input OR gate, p_O , is equal to:

$$\begin{aligned}
 p_O &= p((A = 1 \wedge B = 1) \vee (A = 0 \wedge B = 1) \vee (A = 1 \wedge B = 0)) \\
 &= p(A = 1 \wedge B = 1) + p(A = 0 \wedge B = 1) + p(A = 1 \wedge B = 0) \\
 &= p_A p_B \kappa_{A,B}^{1,1} + (1 - p_A) p_B \kappa_{A,B}^{0,1} + p_A (1 - p_B) \kappa_{A,B}^{1,0} \\
 &= p_A p_B \kappa_{A,B} + (1 - p_A \kappa_{A,B}) p_B + p_A (1 - p_B \kappa_{A,B}) \\
 &= p_A + p_B - p_A p_B \kappa_{A,B}
 \end{aligned} \tag{A.7}$$

Similarly, the static probabilities of other logic gates can be computed using the input static probabilities and correlation coefficients. Figure A.1 summarizes the expressions for basic logic gates.

The correlation coefficients between the output of a logic gate, O , and another signal X can be also computed in terms of input static probabilities and correlation

coefficients. Let us consider the OR gate again:

$$\begin{aligned}
\kappa_{O,X} &= \frac{p(O = 1 \wedge X = 1)}{p(O = 1)p(X = 1)} \\
&= \frac{p(((A = 1 \wedge B = 1) \vee (A = 0 \wedge B = 1) \vee (A = 1 \wedge B = 0)) \wedge X = 1)}{(p_A + p_B - p_{APB}\kappa_{A,B})p_X} \\
&= \frac{p(A = 1 \wedge X = 1) + p(B = 1 \wedge X = 1) - p(A = 1 \wedge B = 1 \wedge X = 1)}{(p_A + p_B - p_{APB}\kappa_{A,B})p_X} \\
&= \frac{p_{APX}\kappa_{A,X} + p_{BPX}\kappa_{B,X} - p_{APB}p_X\kappa_{A,B,X}}{(p_A + p_B - p_{APB}\kappa_{A,B})p_X} \\
&\approx \frac{p_A\kappa_{A,X} + p_B\kappa_{B,X} - p_{APB}\kappa_{A,B}\kappa_{A,X}\kappa_{B,X}}{p_A + p_B - p_{APB}\kappa_{A,B}} \tag{A.8}
\end{aligned}$$

Figure A.2 summarized the correlation coefficients at outputs of basic logic gates in terms of input static probabilities and correlation coefficients. Because of the approximations, the correlation coefficients may exceed the acceptable interval that will result one-probabilities between 0 and 1. Therefore after computing the correlation coefficients it should be checked that it falls into the acceptable interval:

$$\max\left(0, \frac{p_A + p_B - 1}{p_{APB}}\right) \leq \kappa_{A,B} \leq \min\left(\frac{1}{p_A}, \frac{1}{p_B}\right) \tag{A.9}$$

Computation of static probabilities starts from primary inputs of a combinational logic network and traverses towards the outputs. A list of available nodes is created that contains the nodes that their predecessor nodes are computed. It is initialized to the output nodes of logic gates directly connected to primary inputs. After each computation of static probabilities, the list of available nodes must be updated by adding new nodes that are ready to be computed and removing the computed nodes in the list. The computation process continues until the list of available nodes is empty.

The static probabilities are computed using the expressions in Figure A.1. At this point, input static probabilities are known but the correlation coefficients may be unknown. The computation algorithm backtracks the correlation coefficients using expressions in Figure A.2 until it reaches known correlation coefficients. This can be at worst case the primary inputs where the correlation coefficients are known (for independent inputs the correlation coefficients are set to 1). A list of already computed correlation coefficients is required which can speed up the computation process.

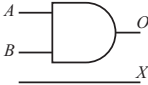




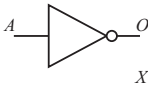
AND		$\kappa_{O,X} = \kappa_{X,A}\kappa_{X,B}$
OR		$\kappa_{O,X} = \frac{p_A\kappa_{A,X} + p_B\kappa_{B,X} - p_A p_B \kappa_{A,B}\kappa_{A,X}\kappa_{B,X}}{p_A + p_B - p_A p_B \kappa_{A,B}}$
NAND		$\kappa_{O,X} = \frac{1 - p_A p_B \kappa_{A,B}\kappa_{A,X}\kappa_{B,X}}{1 - p_A p_B \kappa_{A,B}}$
NOR		$\kappa_{O,X} = \frac{1 - p_A\kappa_{A,X} - p_B\kappa_{B,X} + p_A p_B \kappa_{A,B}\kappa_{A,X}\kappa_{B,X}}{1 - p_A - p_B + p_A p_B \kappa_{A,B}}$
XOR		$\kappa_{O,X} = \frac{p_A\kappa_{A,X} + p_B\kappa_{B,X} - 2p_A p_B \kappa_{A,B}\kappa_{A,X}\kappa_{B,X}}{p_A + p_B - 2p_A p_B \kappa_{A,B}}$
NOT		$\kappa_{O,X} = \frac{1 - p_A\kappa_{A,X}}{1 - p_A}$

Figure A.2: Calculating correlation coefficients for basic logic gates

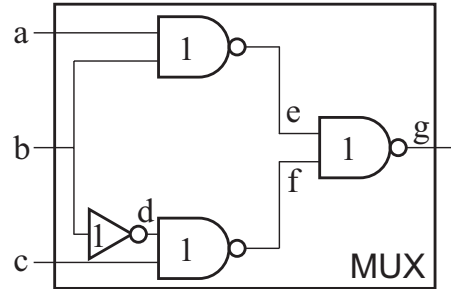


Figure A.3: An example of static probability computation

A.1.1 An Example

As an example of the correlation coefficient method, the static probabilities of a 1-bit multiplexer in Figure A.3 is estimated here. The one-probabilities of the inputs are set to 0.5; i.e.

$$p_A = p_B = p_C = 0.5$$

The mutual correlation coefficients of the inputs are set to 1 and autocorrelation coefficients are set to 2; i.e.

$$\kappa_{A,B} = \kappa_{B,C} = \kappa_{A,C} = 1$$

and

$$\kappa_{A,A} = \kappa_{B,B} = \kappa_{C,C} = 2$$

In this step nodes E and D are computable from the expressions in Figure A.1; i.e.

$$p_D = 1 - p_B = 0.5$$

and

$$p_E = 1 - p_A p_B \kappa_{A,B} = 0.75$$

The next computable node is then F :

$$p_F = 1 - p_C p_D \kappa_{C,D}$$

$\kappa_{C,D}$ is unknown and must be computed using expressions in Figure A.2:

$$\kappa_{C,D} = \frac{1 - p_B \kappa_{B,C}}{1 - p_B} = 1$$

thus, $p_F = 0.75$. Finally, node G is computable:

$$p_G = 1 - p_E p_F \kappa_{E,F}$$

and $\kappa_{E,F}$ must be computed:

$$\kappa_{E,F} = \frac{1 - p_A p_B \kappa_{A,B} \kappa_{A,F} \kappa_{B,F}}{1 - p_A p_B \kappa_{A,B}}$$

$$\kappa_{A,F} = 1$$

$$\kappa_{B,F} = \frac{1 - p_C p_D \kappa_{C,D} \kappa_{B,D} \kappa_{B,C}}{1 - p_C p_D \kappa_{C,D}}$$

$$\kappa_{B,D} = \frac{1 - p_B \kappa_{B,B}}{1 - p_B} = 0$$

Thus, $\kappa_{B,F} = \frac{4}{3}$ and $\kappa_{E,F} = \frac{8}{9}$ resulting $p_G = 0.5$.

Appendix B

Examples of Computing SWSs

In this appendix, the SWS power estimator described in Section 3.1.1 is utilized on two example circuits, a tree-structured circuit and a 2-to-1 MUX circuit. The detailed waveforms are listed for all nodes. The number inside each gate represents the inertial delay of that gate. Primary inputs are assumed to have 0.5 one-probabilities. In addition, primary inputs are assumed to be spatially and temporally independent.

B.2 A 2-to-1 MUX Example

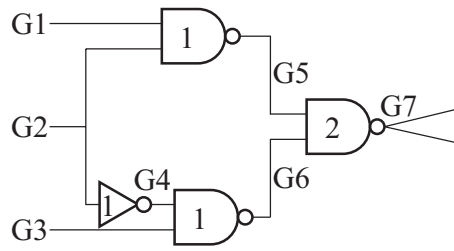


Figure B.2: A 2-to-1 MUX example

Table B.3: Average transition activity for nodes in Figure B.2

Node	Ave.Trans.	Node	Ave.Trans.	Node	Ave.Trans.	Node	Ave.Trans.
G1	0.5	G3	0.5	G5	0.375	G7	0.5
G2	0.5	G4	0.5	G6	0.5		

Waveform	Occurrence Probability	Value		MASK	
		@ $-\infty$	@ $+\infty$	G1	G7
G1					
$\Psi(+\infty, -\infty)$	0.25	0	0	1	1
$\Psi(0, +\infty)$	0.25	0	1	1	1
$\Psi(+\infty, 0)$	0.25	1	0	1	1
$\Psi(-\infty, +\infty)$	0.25	1	1	1	1
G2					
$\Psi(+\infty, -\infty)$	0.25	0	0	1	1
$\Psi(0, +\infty)$	0.25	0	1	1	1
$\Psi(+\infty, 0)$	0.25	1	0	1	1
$\Psi(-\infty, +\infty)$	0.25	1	1	1	1
G3					
$\Psi(+\infty, -\infty)$	0.25	0	0	1	1
$\Psi(0, +\infty)$	0.25	0	1	1	1
$\Psi(+\infty, 0)$	0.25	1	0	1	1
$\Psi(-\infty, +\infty)$	0.25	1	1	1	1
G4					
$\Psi(-\infty, +\infty)$	0.25	1	1	1	1
$\Psi(+\infty, 1)$	0.25	1	0	1	1
$\Psi(1, +\infty)$	0.25	0	1	1	1
$\Psi(+\infty, -\infty)$	0.25	0	0	1	1

G5				
$\Psi(-\infty, +\infty)$	0.5625	1	1	1111111
$\Psi(+\infty, 1)$	0.1875	1	0	1111111
$\Psi(1, +\infty)$	0.1875	0	1	1111111
$\Psi(+\infty, -\infty)$	0.0625	0	0	1111111
G6				
$\Psi(-\infty, +\infty)$	0.5625	1	1	1111111
$\Psi(+\infty, 1)$	0.0625	1	0	1111111
$\Psi(2, +\infty)$	0.0625	1	1	1111110
$\Psi(+\infty, 1)$	0.0625	1	1	1111110
$\Psi(+\infty, -\infty)$	-0.0625	1	1	1111110
$\Psi(-\infty, +\infty)$	-0.0625	1	1	1111110
$\Psi(+\infty, 2)$	0.125	1	0	1111111
$\Psi(1, +\infty)$	0.125	0	1	1111111
$\Psi(+\infty, -\infty)$	0.0625	0	0	1111111
$\Psi(2, +\infty)$	0.0625	0	1	1111111
G7				
$\Psi(+\infty, -\infty)$	0.25	0	0	1111111
$\Psi(3, +\infty)$	0.166667	0	1	1111111
$\Psi(4, +\infty)$	0.0833333	0	1	1111111
$\Psi(+\infty, 3)$	0.208333	1	0	1111111
$\Psi(-\infty, +\infty)$	0.25	1	1	1111111
$\Psi(+\infty, 4)$	0.0416667	1	0	1111111

Table B.4: The simple waveform sets for the 2-to-1 MUX example

Bibliography

- [1] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage current reduction in CMOS VLSI circuits by input vector control," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 2, pp. 140–154, 2004.
- [2] I. Abu-Khater, A. Bellaouar, and M. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers," *IEEE J. Solid -State Circuits*, vol. 31, no. 10, pp. 1535–1546, 1996.
- [3] F. A. Aloul, S. Hassoun, K. A. Sakallah, and D. Blaauw, "Robust SAT-based search algorithm for leakage power reduction," in *Proc. 12th Intr. Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, pp. 167–177, 2002.
- [4] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "The IBM system/360 model 91: Floating-point execution unit," *IBM J. of Research and Development*, vol. 11, no. 1, pp. 34–53, 1967.
- [5] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IRE Trans. Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, 1961.
- [6] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Computers*, vol. C-22, pp. 1045–1047, 1973.
- [7] H. Benaroya and S. M. Han, *Probability Models in Engineering and Science*. Boca Raton, FL, USA: CRC, 1st ed., 2005.
- [8] L. Benini, A. Bogliolo, M. Favalli, and G. D. Micheli, "Regression models for behavioral power estimation," *Integr. Comput.-Aided Eng.*, vol. 5, no. 2, pp. 95–106, 1998.
- [9] S. Bhanja and N. Ranganathan, "Modeling switching activity using cascaded bayesian networks for correlated input streams," in *Proc. IEEE Intr.*

- Conf. on Computer Design: VLSI in Computers and Processors*, pp. 388–390, 2002.
- [10] S. Bhanja and N. Ranganathan, “Switching activity estimation of VLSI circuits using bayesian networks,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 11, no. 4, pp. 558–567, 2003.
- [11] M. Bhardwaj, R. Min, and A. Chandrakasan, “Power-aware systems,” in *Proc. 34th Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1695–1701, Nov. 2000.
- [12] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., “Reduced area multipliers,” in *Proc. Intr. Conf. on Application-Specific Array Processors*, pp. 478–489, 1993.
- [13] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., “Parallel reduced area multipliers,” *J. VLSI Signal Process. Syst.*, vol. 9, no. 3, pp. 181–191, 1995.
- [14] K. C. Bickerstaff, E. E. Swartzlander, Jr., and M. J. Schulte, “Analysis of column compression multipliers,” in *Proc. IEEE Symp. Comp. Arith.*, pp. 33–39, 2001.
- [15] A. D. Booth, “A signed binary multiplication technique,” *Quarterly J. of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.
- [16] B. K. Bose, D. A. Patterson, L. Pei, and G. S. Taylor, “Fast multiply and divide for a VLSI floating-point unit,” in *Proc. 8th IEEE Symp. on Computer Arithmetic*, pp. 87–94, May 1987.
- [17] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” *IEEE Trans. Computers*, vol. 31, no. 3, pp. 260–264, 1982.
- [18] F. Brglez, P. Pownall, and R. Hum, “Applications of testability analysis: From ATPG to critical delay path tracing,” in *Proc. Intr. Test Conf.*, pp. 705–712, 1984.
- [19] D. Brooks and M. Martonosi, “Dynamically exploiting narrow width operands to improve processor power and performance,” in *Proc. 5th Intr. Symp. on High Performance Computer Architecture*, pp. 13–22, 1999.
- [20] D. Brooks and M. Martonosi, “Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance,” *ACM Trans. Comput. Syst.*, vol. 18, no. 2, pp. 89–126, 2000.

- [21] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [22] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Computers*, vol. 40, no. 2, pp. 205–213, 1991.
- [23] R. Burch, F. Najm, P. Yang, and T. Trick, "A monte carlo approach for power estimation," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 1, no. 1, pp. 63–71, 1993.
- [24] T. K. Callaway and E. E. Swartzlander, Jr., "Power-delay characteristics of CMOS multipliers," in *Proc. 13th Symp. on Computer Arithmetic (ARITH '97)*, pp. 26–32, 1997.
- [25] T. Callaway and E. E. Swartzlander, Jr., *Low Power Arithmetic Components*, pp. 161–198. *Low Power Design Methodologies (The International Series in Engineering and Computer Science)*, Rabaey, J.M. and Pedram, M. eds., Norwell, MA, USA: Kluwer Academic Publishers, 1996.
- [26] S. Chakravarty, "On the complexity of using BDDs for the synthesis and analysis of boolean circuits," in *Proc. 27th Annual Allerton Conf. on Communication, Control, and Computing*, pp. 730–739, 1989.
- [27] S. Chakravarty and H. B. Hunt, "On computing signal probability and detection probability of stuck-at faults," *IEEE Trans. Computers*, vol. 39, no. 11, pp. 1369–1377, 1990.
- [28] C.-H. Chang, J. Gu, and M. Zhang, "A review of 0.18- μm full adder performances for tree structured arithmetic circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 6, pp. 686–695, 2005.
- [29] J. Cheng and M. J. Druzdzel, "AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks," *J. Artificial Intelligence Research*, vol. 13, pp. 155–188, 2000.
- [30] J. Chittamuru, W. Burlison, and J. Euh, "Dynamic wordlength variation for low-power 3D graphics texture mapping," in *IEEE Workshop on Signal Processing Systems*, pp. 251–256, 2003.
- [31] G. Choe and E. E. Swartzlander, Jr., "Interconnection effects in fast multipliers," in *Proc. of the 33th Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1224–1227, 1999.

- [32] Y. Choi and E. E. Swartzlander, Jr., "Design of a hybrid prefix adder for nonuniform input arrival times," in *SPIE: Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, vol. 4791, pp. 456–465, 2002.
- [33] L. Ciminiera and P. Montuschi, "Carry-save multiplication schemes without final addition," *IEEE Trans. Computers*, vol. 45, no. 9, pp. 1050–1055, 1996.
- [34] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, *Synthesis And Optimization Of DSP Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.
- [35] G. A. Constantinides and G. J. Woeginger, "The complexity of multiple wordlength assignment," *Appl. Math. Lett.*, vol. 15, no. 2, pp. 137–140, 2002.
- [36] J. T. Coonen, "An implementation guide to a proposed standard for floating point arithmetic," *IEEE Computer*, vol. 13, no. 1, pp. 68–79, 1980.
- [37] R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter, *Probabilistic Networks and Expert Systems*. Secaucus, NJ, USA: Springer-Verlag, 1999.
- [38] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, March 1965.
- [39] L. Dadda, "On parallel digital multipliers," *Alta Frequenza*, vol. 45, pp. 574–580, October 1976.
- [40] E. de Angel, *Low Power Digital Multiplication*. PhD thesis, The University of Texas at Austin, 1996.
- [41] E. de Angel, *Low Power Digital Multipliers*, ch. 4. Application Specific Processors (The International Series in Engineering and Computer Science), E.E. Swartzlander Jr. ed., Norwell, MA, USA: Kluwer Academic Publishers, 1997.
- [42] E. de Angel and E. E. Swartzlander, Jr., "Survey of low power techniques for VLSI design," in *Proc. IEEE Intr. Conf. Innovative Systems in Silicon*, pp. 159–169, Oct 1996.
- [43] E. de Angel and E. E. Swartzlander, Jr., "Switching activity in parallel multipliers," in *Proc. of the 35th Asilomar Conf. on Signals, Systems and Computers*, pp. 857–860, 2001.

- [44] D. De Caro, E. Napoli, and A. Strollo, "Direct digital frequency synthesizers with polynomial hyperfolding technique," *IEEE Trans. Circuit & Syst. II*, vol. 51, no. 7, pp. 337–344, 2004.
- [45] J.-P. Deschamps, G. J. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. John Wiley & Sons, 2006.
- [46] C. Ding, C. Tsui, and M. Pedram, "Gate-level power estimation using tagged probabilistic simulation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 11, pp. 1099–1107, 1998.
- [47] S. Dormido and M. A. Canto, "Synthesis of generalized parallel counters," *IEEE Trans. Computers*, vol. C-30, no. 9, pp. 699–703, 1981.
- [48] S. Dormido and M. A. Canto, "An upper bound for the synthesis of generalized parallel counters," *IEEE Trans. Computers*, vol. C-31, no. 8, pp. 802–805, 1982.
- [49] M. D. Ercegovic and T. Lang, "Fast multiplication without carry-propagate addition," *IEEE Trans. Computers*, vol. 39, no. 11, pp. 1385–1390, 1990.
- [50] M. D. Ercegovic and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.
- [51] M. D. Ercegovic and T. Lang, "On recoding in arithmetic algorithms," *J. VLSI Signal Process. Syst.*, vol. 14, no. 3, pp. 283–294, 1996.
- [52] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco, "Estimate of signal probability in combinational logic networks," in *Proc. 1st European Test Conf.*, pp. 132–138, 1989.
- [53] G. Estrin, B. Gilchrist, and J. H. Pomerene, "A note on high-speed digital multiplication," *IRE Transactions on Electronic Computers*, vol. 5, no. 3, p. 140, 1956.
- [54] M. I. Ferguson and M. D. Ercegovic, "A multiplier with redundant operands," in *Proc. 33rd Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1322–1326, 1999.
- [55] M. J. Flynn and S. S. Oberman, *Advanced Computer Arithmetic Design*. New York, USA: John Wiley & Sons, Inc., rev. ed., 2001.
- [56] C. C. Foster and F. D. Stockton, "Counting responders in an associative memory," *IEEE Trans. Computers*, vol. C-20, no. 12, pp. 1580–1583, 1971.

- [57] D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, ch. 6. New Jersey, USA: Prentice Hall, 1994.
- [58] D. D. Gajski, "Parallel compressors," *IEEE Trans. Computers*, vol. C-29, no. 5, pp. 393–398, 1980.
- [59] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proc. Design Automation Conf.*, pp. 253–259, 1992.
- [60] A. Glaser, *History of Binary and Other Nondecimal Numeration*. US: Tomash Publishers, rev. ed., 1981.
- [61] A. Goldsmith, *Wireless Communications*. New York, USA: Cambridge University Press, 2005.
- [62] J. B. Gosling, "Design of large high-speed floating-point arithmetic units," *IEE Proc.*, vol. 118, pp. 493–498, 1971.
- [63] J. B. Gosling, *Design of Arithmetic Units for Digital Computers*. New York, US: Springer-Verlag, 1980.
- [64] O. Gustafsson, K. Johansson, and L. Wanhammar, "Optimization and quantization effects for sine and cosine computation using a sum of bit-products," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, (Pacific Grove, CA), pp. 1347–1351, 2005.
- [65] O. Gustafsson, S. T. Oskuii, K. Johansson, and P. G. Kjeldsberg, "Switching activity reduction of mac-based fir filters with correlated input data," in *17th Intr. Workshop on Power and Timing Modeling Optimization and Simulation*, pp. 526–535, 2007.
- [66] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Computers*, vol. C-19, no. 2, pp. 153–157, 1970.
- [67] J. Halter and F. Najm, "A gate-level leakage power reduction method for ultra-low-power cmos circuits," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 475–478, 1997.
- [68] C.-Y. Han, H.-J. Park, and L.-S. Kim, "A low-power array multiplier using separated multiplication technique," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 9, pp. 866–871, 2001.
- [69] H. Hassler and N. Takagi, "Function evaluation by table look-up and addition," in *Proc. Symp. Comp. Arith.*, pp. 10–16, 1995.

- [70] M. Hatamian and G. L. Cash, "A 70-MHz 8-bit x 8 bit parallel pipelined multiplier in 2.5- μ m CMOS," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 4, pp. 505–513, 1986.
- [71] S. Hong, S.-S. Chin, S. Kim, and W. Hwang, "Multiplier architecture power consumption characterization for low-power DSP applications," in *Proc. IEEE Intr. Conf. on Electronics, Circuits and Systems (ICECS)*, vol. 2, pp. 741–744, 2002.
- [72] F. Hu and V. D. Agrawal, "Dual-transition glitch filtering in probabilistic waveform power estimation," in *Proc. 15th Great Lakes Symp. on VLSI*, pp. 357–360, 2005.
- [73] F. Hu and V. D. Agrawal, "Enhanced dual-transition probabilistic power estimation with selective supergate analysis," in *Proc. Intr. Conf. on Computer Design*, pp. 366–372, 2005.
- [74] X. Huang, B. W. Y. Wei, H. Chen, and Y. H. Mao, "High-performance VLSI multiplier with a new redundant binary coding," *J. VLSI Signal Process. Syst.*, vol. 3, no. 4, pp. 283–291, 1991.
- [75] Z. Huang, *High-Level Optimization Techniques for Low-Power Multiplier Design*. PhD thesis, University of California Los Angeles, 2003.
- [76] C. M. Huizer, "Power dissipation analysis of CMOS VLSI circuits by means of switch-level simulation," in *Proc. IEEE European Solid State Circuits Conf.*, pp. 61–64, 1990.
- [77] J. Jedwab and C. J. Mitchell, "Minimum weight modified signed-digit representations and fast exponentiation," *Electronics Letters*, vol. 25, no. 17, pp. 1171–1172, 1989.
- [78] K. Johansson, O. Gustafsson, and L. Wanhammar, "Low power architectures for sine and cosine computation using a sum of bit-products," in *Proc. IEEE NorChip Conf.*, (Oulu, Finland), pp. 161–164, 2005.
- [79] K. Johansson, O. Gustafsson, and L. Wanhammar, "Approximation of elementary functions using a weighted sum of bit-products," in *Proc. IEEE Int. Symp. Circuits Syst.*, (Kos Island, Greece), pp. 795–798, 2006.
- [80] K. Johansson, O. Gustafsson, and L. Wanhammar, "Conversion and addition in logarithmic number systems using a sum of bit-products," in *Proc. IEEE Norchip Conf.*, (Linköping, Sweden), pp. 39–42, 2006.

- [81] S. Kang, "Accurate simulation of power dissipation in VLSI circuits," *IEEE journal of solid-state circuits*, vol. SC-21, no. 5, pp. 899–901, 1986.
- [82] B. Kapoor, "Improving the accuracy of circuit activity measurement," in *Proc. Design Automation Conf.*, pp. 734–739, June 1994.
- [83] K.-Y. Khoo, Z. Yu, and A. N. Willson, "Bit-level arithmetic optimization for carry-save additions," in *Proc. IEEE/ACM Intr. Conf. on Computer-aided design*, pp. 14–19, 1999.
- [84] T. Kilburn, D. B. G. Edwards, and D. Aspinall, "Parallel addition in digital computers: A new fast carry circuit," *Proceedings of the IEE*, vol. 106, no. B, pp. 464–466, 1959.
- [85] K. K. Kim, Y.-B. Kim, M. Choi, and N. Park, "Leakage minimization technique for nanoscale CMOS VLSI," *IEEE Design and Test*, vol. 24, no. 4, pp. 322–330, 2007.
- [86] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.
- [87] T. Kim, W. Jao, and S. Tjiang, "Arithmetic optimization using carry-save-adders," in *Proc. Conf. on Design automation*, pp. 433–438, 1998.
- [88] Y.-T. Kim and T. Kim, "An accurate exploration of timing and area trade-offs in arithmetic optimization using carry-save-adders," *Journal of Circuits, Systems, and Computers*, vol. 10, no. 5-6, pp. 279–292, 2000.
- [89] Y.-T. Kim and T. Kim, "Accurate exploration of timing and area trade-offs in arithmetic optimization using carry-save-adders," in *Proc. Conf. on Asia South Pacific design automation*, pp. 622–628, 2001.
- [90] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, 4598, pp. 671–680, May 1983.
- [91] H. Kobayashi and H. Ohara, "A synthesizing method for large parallel counters with a network of smaller ones," *IEEE Trans. Computers*, vol. 27, no. 8, pp. 753–757, 1978.
- [92] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

- [93] I. Koren, *Computer Arithmetic Algorithms*. A. K. Peters, Natick, Massachusetts, 2nd ed., 2002.
- [94] R. Krieger, B. Becker, and R. Sinkovic, "A BDD-based algorithm for computation of exact fault detection probabilities," in *Digest of Papers, Intr. Symp. on Fault-Tolerant Computing*, pp. 186–195, 1993.
- [95] B. Krishnamurthy and I. G. Tollis, "Improved techniques for estimating signal probabilities," *IEEE Trans. Computers*, vol. 38, no. 7, pp. 1041–1045, 1989.
- [96] O. Kwon, E. E. Swartzlander, Jr., and K. Nowka, "A 16-bit x 16-bit MAC design using fast 5:2 compressors," in *Proc. IEEE Intr. Conf. on Application-Specific Systems, Architectures, and Processors*, pp. 235–243, 2000.
- [97] P. Landman and J. Rabaey, "Power estimation for high level synthesis," in *Proc. European Design Automation Conf.*, pp. 361–366, 1993.
- [98] P. E. Landman and J. M. Rabaey, "Architectural power analysis: the dual bit type method," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 2, pp. 173–187, 1995.
- [99] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*. Wiley-IEEE Press, 1996.
- [100] P. Larsson and C. Nicol, "Transition reduction in carry-save adder trees," in *Proc. Intr. Symp. on Low power electronics and design*, pp. 85–88, 1996.
- [101] C. F. Law, S. S. Rofail, and K. S. Yeo, "A low-power 16×16 -b parallel multiplier utilizing pass-transistor logic," *IEEE J. Solid -State Circuits*, vol. SC-34, no. 10, pp. 1395–1399, 1999.
- [102] M.-J. Liao, C.-F. Su, C.-Y. Chang, and A.-H. Wu, "A carry-select-adder optimization technique for high-performance Booth-encoded wallace-tree multipliers," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. I-81–I-84, 2002.
- [103] H. Ling, "High speed binary adder," *IBM J. Research and Development*, vol. 25, no. 2-3, pp. 156–166, 1981.
- [104] W. Ling and Y. Savaria, "Variable-precision multiplier for equalizer with adaptive modulation," in *Proc. 47th Midwest Symp. Circuits and Systems*, pp. I-553–556, 2004.

- [105] M. Lu, *Arithmetic and Logic in Computer Systems*. John Wiley & Sons, 2004.
- [106] G.-K. Ma and F. J. Taylor, "Multiplier policies for digital signal processing," *IEEE ASSP Magazine*, vol. 7, no. 1, pp. 6–19, 1990.
- [107] O. L. MacSorley, "High-speed arithmetic in binary computers," *IRC Proceedings*, vol. 49, no. 1, pp. 67–91, 1961.
- [108] P. E. Madrid, B. Millar, and E. E. Swartzlander, Jr., "Modified Booth algorithm for high radix multiplication," in *IEEE Computer Design: VLSI in Computers and Processors, Intr. Conf. on*, pp. 118–121, October 1992.
- [109] S. S. Mahant-Shetti, P. T. Balsara, and C. Lemonds, "High performance low power array multiplier using temporal tiling," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 7, no. 1, pp. 121–124, 1999.
- [110] J. C. Majithia and R. Kitai, "An iterative array for multiplication of signed binary numbers," *IEEE Trans. Computers*, vol. C-20, no. 2, pp. 214–216, 1971.
- [111] H. Makino, Y. Nakase, and H. Shinohara, "A 8.8-ns 54×54 -bit multiplier using new redundant binary architecture," in *Proc. Intr. Conf. on Computer Design*, pp. 202–205, 1993.
- [112] D. M. Mandelbaum and S. G. Mandelbaum, "A fast, efficient parallel-acting method of generating functions defined by power series, including logarithm, exponential, and sine, cosine," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 1, pp. 33–45, 1996.
- [113] R. Marculescu, D. Marculescu, and M. Pedram, "Switching activity analysis considering spatiotemporal correlations," in *Proc. IEEE/ACM Intr. Conf. on Computer-aided design*, pp. 239–299, 1994.
- [114] R. Marculescu, D. Marculescu, and M. Pedram, "Efficient power estimation for highly correlated input streams," in *Proc. 32nd ACM/IEEE Conf. on Design Automation*, pp. 628–634, 1995.
- [115] K. Masselos, P. Merakos, S. Theoharis, T. Stouraitis, and C. E. Goutis, "Power efficient data path synthesis of sum-of-products computations," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 11, no. 3, pp. 446–450, 2003.

- [116] J. McClellan, T. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IEEE Trans. Audio and Electroacoust.*, vol. AU-21, no. 6, pp. 506–526, 1973.
- [117] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Low-power realization of FIR filters on programmable DSP's," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, no. 4, pp. 546–553, 1998.
- [118] P. C. H. Meier, *Analysis and design of low power digital multipliers*. PhD thesis, Carnegie Mellon University, 1999.
- [119] A. R. Meo, "Arithmetic networks and their minimization using a new line of elementary units," *IEEE Trans. Computers*, vol. 24, no. 3, pp. 258–280, 1975.
- [120] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.
- [121] G. Metze and J. E. Robertson, "Elimination of carry propagation in digital computers," in *IFIP Congress*, pp. 389–395, 1959.
- [122] S. Miller and D. Childers, *Probability and Random Processes: With Applications to Signal Processing and Communications*. New York: Academic Press, 2004.
- [123] J. Monks, V. Bharghavan, and W. Hwu, "A power controlled multiple access protocol for wireless packet networks," in *Proc. IEEE Conf. on Computer Communications (INFOCOM)*, pp. 219–228, 2001.
- [124] R. K. Montoye, E. Hokonek, and S. L. Runyan, "Design of the floating-point execution unit of the IBM RISC system/6000," *IBM J. of Research and Development*, vol. 34, no. 1, pp. 59–70, 1990.
- [125] G. E. Moore, "No exponential is forever: but "forever" can be delayed!," in *Proc. IEEE Int. Solid-State Circuits Conf., Digest of Technical Papers*, pp. 20–23 vol.1, 2003.
- [126] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*. Birkhäuser Boston, 2nd ed., 2006.
- [127] K. T. M. Muroyama, S. Yamaguchi, and H. Yasuura, "A design method for a low power equalization circuit by adaptive bitwidth control," in *IEEE Intr. Symp. Communications and Information Technology*, pp. 704–709, 2004.

- [128] E. Musoll and J. Cortadella, "Low-power array multipliers with transition-retaining barriers," in *5th Intr. Workshop on Power and Timing Modeling Optimization and Simulation*, pp. 227–238, Oct 1995.
- [129] M. Nagamatsu, S. Tanaka, J. Mori, T. Noguchi, and K. Hatanaka, "A 15 ns 32x32-bit CMOS multiplier with an improved parallel structure," in *Proc. of IEEE Custom Integrated Circuits Conf.*, pp. 10.3/1–10.3/4, 1989.
- [130] F. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 310–323, 1992.
- [131] F. Najm, R. Burch, P. Yang, and I. Hajj, "CREST - a current estimator for CMOS circuits," in *Proc. IEEE Intr. Conf. on Computer-aided design*, pp. 204–207, November 1988.
- [132] F. Najm, R. Burch, P. Yang, and I. Hajj, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 4, pp. 439–450, 1990.
- [133] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 2, no. 4, pp. 446–455, 1994.
- [134] S. Narayanaswamy, V. Kawadia, R. Sreenivas, and P. Kumar, "Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the compow protocol," in *European Wireless Conference*, 2002.
- [135] S. G. Narendra and A. Chandrakasan, *Leakage in Nanometer CMOS Technologies (Series on Integrated Circuits and Systems)*. New York, USA: Springer-Verlag, 2005.
- [136] W. Nebel and J. Mermet, eds., *Low power design in deep submicron electronics*. The International Series in Engineering and Computer Science, Norwell, MA, USA: Kluwer Academic Publishers, 1997.
- [137] T. Noll, D. Schmitt, H. Klar, and G. Enders, "A pipelined 330-MHz multiplier," *IEEE journal of solid-state circuits*, vol. SC-21, pp. 411–416, 1986.
- [138] K. Nose and T. Sakurai, "Analysis and future trend of short-circuit power," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1023–1030, 2000.
- [139] V. Oklobdzija, D. Villeger, and S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Trans. Computers*, vol. 45, no. 3, pp. 294–306, 1996.

- [140] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," in *Proc. 29th Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 966–970, 1995.
- [141] B. Parhami, *Computer Arithmetic - Algorithms and Hardware Design*. New York, US: Oxford University Press, 2000.
- [142] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Computers*, vol. 24, no. 6, pp. 668–670, 1975.
- [143] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. Design Automation of Electronic Systems*, vol. 1, no. 1, pp. 3–56, 1996.
- [144] M. Pedram, *Power simulation and estimation in VLSI circuits*, pp. 18–1–18–27. The VLSI Handbook: W-K. Chen ed., CRC Press and IEEE Press, 1999.
- [145] S. D. Pezaris, "A 40 ns 17 bit by 17 bit array multiplier," *IEEE Trans. Computers*, vol. C-20, no. 4, pp. 442–447, 1971.
- [146] M. Potkonjak and J. M. Rabaey, "Optimizing resource utilization using transformations," in *Proc. IEEE/ACM Intr. Conf. on Computer-aided design*, pp. 88–91, 1991.
- [147] S. Powell and P. Chau, "Estimating power dissipation of VLSI signal processing chips: The PFA technique," *IEEE Workshop on VLSI Signal Processing*, IEEE Press, vol. IV, pp. 250–259, 1990.
- [148] K. Prasad and K. K. Parhi, "Low-power 4-2 and 5-2 compressors," in *Proc. of the 35th Asilomar Conf. on Signals, Systems and Computers*, vol. 1, pp. 129–133, 2001.
- [149] J. Proakis and D. G. Manolakis, *Digital Signal Processing, Principles, Algorithms, and Applications*. New Jersey, USA: Prentice Hall, 3rd ed., 1996.
- [150] X. Qi, S. C. Lo, A. Gyure, Y. Luo, M. Shahram, K. Singhal, and D. B. MacMillen, "Efficient subthreshold leakage current optimization - leakage current optimization and layout migration for 90- and 65- nm ASIC libraries," *IEEE Circuits and Devices Magazine*, vol. 22, no. 5, pp. 39–47, 2006.
- [151] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, A Design Perspective*. Prentice Hall, 2nd ed., 2003.

- [152] J. Rabaey and M. Pedram, eds., *Low Power Design Methodologies*. The International Series in Engineering and Computer Science, Norwell, MA, USA: Kluwer Academic Publishers, 1996.
- [153] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "Analytical estimation of transition activity from word-level signal statistics," in *Proc. Design Automation Conf.*, pp. 582–587, 1997.
- [154] G. W. Reitwiesner, "Binary arithmetic," *Advances in Computers*, pp. 231–308, 1960.
- [155] T. Rejimon and S. Bhanja, "An accurate probabilistic model for error detection," in *Proc. Intr. Conf. on VLSI Design held jointly with Intr. Conf. on Embedded Systems Design*, pp. 717–722, 2005.
- [156] E. Y. Remez, "General computational methods of chebychev approximation," *Kiev, USSR: Atomic Energy Translation 4491*, 1957.
- [157] L. Rijnders, Z. Sahraoui, P. Six, and H. D. Man, "Timing optimization by bit-level arithmetic transformations," in *Proc. Conf. on European design automation*, pp. 48–53, 1995.
- [158] J. E. Robertson, "Two's complement multiplication in binary parallel computers," *IEEE Transactions on Electronic Computers*, vol. EC-34, no. 3, pp. 118–119, 1955.
- [159] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," *IEEE Trans. Computers*, vol. 24, no. 10, pp. 1014–1015, 1975.
- [160] T. Sakuta, W. Lee, and P. Balsara, "Delay balanced multipliers for low power/low voltage DSP core," in *Proc. IEEE Symp. Low Power Electronics*, pp. 36–37, Oct 1995.
- [161] H. Sam and A. Gupta, "A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations," *IEEE Trans. Computers*, vol. 39, no. 8, pp. 1006–1015, 1990.
- [162] M. R. Santoro and M. A. Horowitz, "SPIM: a pipelined 64x64-bit iterative multiplier," *IEEE journal of solid-state circuits*, vol. 24, pp. 487–493, 1989.
- [163] J. Savir, G. Ditlow, and P. H. Bardell, "Random pattern testability," in *Proc. IEEE Symp. Fault Tolerant Comput.*, pp. 80–89, 1983.
- [164] P. Schneider and U. Schlichtmann, "Decomposition of boolean functions for low power based on a new power estimation technique," in *Proc. Intr. Low Power Design Workshop*, pp. 123–128, April 1994.

- [165] E. M. Schwarz and M. J. Flynn, "Hardware starting approximation method and its application to the square root operation," *IEEE Trans. Computers*, vol. 45, no. 12, pp. 1356–1369, 1996.
- [166] P.-M. Seidel, L. D. McFearn, and D. W. Matula, "Binary multiplication radix-32 and radix-256," in *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, pp. 23–32, October 2001.
- [167] S. Seth, L. Pan, and V. Agrawal, "Predict – probabilistic estimation of digital circuit testability," in *Proc. Fault Tolerant Computing Symp.*, pp. 220–225, June 1985.
- [168] A. M. Shams, T. K. Darwish, and M. A. Bayoumi, "Performance analysis of low-power 1-bit CMOS full adder cells," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 10, no. 1, pp. 20–29, 2002.
- [169] R. Stefanelli, "A suggestion for a high-speed parallel binary divider," *IEEE Trans. Computers*, vol. C-21, no. 1, pp. 42–55, 1972.
- [170] P. F. Stelling, C. U. Martel, V. G. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Trans. Computers*, vol. 47, no. 3, pp. 273–285, 1998.
- [171] P. F. Stelling and V. Oklobdzija, "Design strategies for the final adder in a parallel multiplier," in *Proc. 29th Asilomar Conf. on Signals, Systems and Computers*, pp. 591–595, 1995.
- [172] P. F. Stelling and V. G. Oklobdzija, "Implementing multiply-accumulate operation in multiplication time," in *Proc. 13th Symposium on Computer Arithmetic*, pp. 99–106, 1997.
- [173] W. J. Stenzel, W. J. Kubitz, and G. H. Garcia, "A compact high-speed parallel multiplication scheme," *IEEE Trans. Computers*, vol. 26, no. 10, pp. 948–957, 1977.
- [174] T. Stornetta and F. Brewer, "Implementation of an efficient parallel BDD package," in *Proc. Design Automation Conf.*, pp. 641–641, June 1996.
- [175] E. E. Swartzlander, Jr. and G. Goto, *Computer Arithmetic*, ch. 9. The Computer Engineering Handbook: Electrical Engineering Handbook, V. G. Oklobdzija ed., Boca Raton, FL, USA: CRC Press, Inc., 2002.
- [176] E. E. Swartzlander, Jr., "Merged arithmetic," *IEEE Trans. Computers*, vol. 29, no. 10, pp. 946–950, 1980.

- [177] Synopsys, “Power management.” Y-2006.06 Synopsys Online Documentation, <http://www.synopsys.com/support/dotw.html>.
- [178] S. Tahmasbi Oskuii, K. Johansson, O. Gustafsson, and P. G. Kjeldsberg, “Power optimization of weighted bit-product summation tree for elementary function generator,” in *to appear in Proc. Intr. Symp. on Circuits and Systems*, (Seattle, USA), pp. –, May 2008.
- [179] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and E. J. Aas, “Probabilistic gate-level power estimation using a novel waveform set method,” in *Proc. 17th Great Lakes Symp. on VLSI*, pp. 37–42, March 2007.
- [180] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and O. Gustafsson, “Power optimized partial product reduction interconnect ordering in parallel multipliers,” in *Proc. 25th IEEE Norchip Conf.*, (Aalborg, Denmark), November 2007.
- [181] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and O. Gustafsson, “Transition-activity aware design of reduction-stages for parallel multipliers,” in *Proc. 17th Great Lakes Symp. on VLSI*, pp. 120–125, March 2007.
- [182] S. Tahmasbi Oskuii, P. G. Kjeldsberg, L. Lundheim, and A. Havashki, “Power optimization of parallel multipliers in systems with variable word-length,” in *submitted to 8th Nordic Signal Processing Symposium (NORSIG’08)*, (Copenhagen, Denmark), pp. –, June 2008.
- [183] N. Takagi, H. Yasuura, and S. Yajima, “High-speed VLSI multiplication algorithm with a redundant binary addition tree,” *IEEE Trans. Computers*, vol. 34, no. 9, pp. 789–796, 1985.
- [184] S. Theoharis, G. Theodoridis, D. Soudris, C. Goutis, and A. Thanailakis, “A fast and accurate delay dependent method for switching estimation of large combinational circuits,” *J. Systems Architecture*, vol. 48, no. 4–5, pp. 113–124, 2002.
- [185] W. J. Townsend, E. E. Swartzlander, Jr., and J. A. Abraham, “Accumulative parallel counters,” in *Proc. SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, vol. 5205, pp. 552–560, 2003.
- [186] W. J. Townsend, E. E. Swartzlander, Jr., and J. A. Abraham, “A comparison of dadda and wallace multiplier delays,” in *Proc. SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, vol. 5205, pp. 552–560, Dec. 2003.

- [187] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Efficient estimation of dynamic power consumption under a real delay model," in *Proc. IEEE/ACM Intr. Conf. on Computer-Aided Design*, pp. 224–228, 1993.
- [188] A. Tyagi, "Hercules: A power analyzer of MOS VLSI circuits," in *Proc. IEEE Intr. Conf. on Computer-aided design*, pp. 530–533, 1987.
- [189] J. Um, T. Kim, and C. L. Liu, "Optimal allocation of carry-save-adders in arithmetic optimization," in *Proc. IEEE/ACM Intr. Conf. on Computer-aided design*, pp. 410–413, 1999.
- [190] H. J. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE J. Solid -State Circuits*, vol. SC-19, pp. 468–473, Aug. 1984.
- [191] S. Vemuru, N. Scheinberg, and E. Smith, "Short-circuit power dissipation formulae for CMOS gates," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 1333–1336, 1993.
- [192] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959.
- [193] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, pp. 14–17, February 1964.
- [194] J. Walther, "A unified algorithm for elementary functions," in *Proc. Joint Computer Conference Proceedings*, vol. 38, pp. 379–385, 1971.
- [195] L. Wanhammar, *DSP Integrated Circuits*. New York: Academic Press, 1999.
- [196] L. Wanhammar, K. Johansson, and O. Gustafsson, "Efficient sine and cosine computation using a weighted sum of bit-products," in *Proc. European Conf. Circuit Theory Design*, vol. 1, (Cork, Ireland), pp. 139–142, 2005.
- [197] A. Weinberger, "4:2 carry-save adder module," *IBM Technical Disclosure Bulletin*, vol. 23, January 1981.
- [198] A. Weinberger and J. L. Smith, "A logic for high-speed addition," *Nat. Bur. Stand. Circ.*, vol. 591, pp. 3–12, 1958.
- [199] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education Inc., Addison-Wesley, 3rd ed., 2004.

- [200] S. Yoshizawa and Y. Miyanaga, "Tunable wordlength architecture for a low power wireless OFDM demodulator," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E89-A, no. 10, pp. 2866–2873, 2006.
- [201] Z. Yu, L. Wasserman, and A. Willson, "A painless way to reduce power dissipation by over 18% in Booth-encoded carry-save array multipliers for DSP," in *Proc. IEEE Workshop Signal Processing Syst.*, pp. 571–580, October 2000.
- [202] Z. Yu, M.-L. Yu, and A. N. Willson, "Signal representation guided synthesis using carry-save adders for synchronous data-path circuits," in *Proc. Design Automation Conf.*, pp. 456–461, 2001.
- [203] C. Yuan and M. Druzdzel, "An importance sampling algorithm based on evidence pre-propagation," in *Proc. Conf. on Uncertainty in Artificial Intelligence*, pp. 624–631, 2003.
- [204] Z. Zeng, Q. Zhang, I. Harris, and M. Ciesielski, "Fast computation of data correlation using BDDs," in *Proc. Conf. on Design, Automation and Test in Europe*, pp. 122–127, 2003.
- [205] R. Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders," in *Intr. Workshop on Logic and Architecture Synthesis*, pp. 123–132, 1996.
- [206] J. H. Zurawski and J. B. Gosling, "Design of a high-speed square root multiply and divide unit," *IEEE Trans. Computers*, vol. 36, no. 1, pp. 13–23, 1987.