University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2003

A reconfigurable digital multiplier architecture.

Pedram. Mokrian
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Mokrian, Pedram., "A reconfigurable digital multiplier architecture." (2003). *Electronic Theses and Dissertations*. 728.
<https://scholar.uwindsor.ca/etd/728>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

A Reconfigurable Digital Multiplier Architecture

by

Pedram Mokrian

A Thesis

Submitted to the Faculty of Graduate Studies and Research through the
Department of Electrical and Computer Engineering in Partial Fulfillment
of the Requirements for the Degree of Master of Applied Science at
the University of Windsor

Windsor, Ontario, Canada

April, 2003

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-82888-3

Our file Notre référence

ISBN: 0-612-82888-3

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

983549

© 2003 Pedram Mokrian

All Rights Reserved. No part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium or by any means without the prior written permission of the author.

Abstract

The recent growth in microprocessor performance has been a direct result of designers exploiting decreasing device feature sizes, while at the same time deepening the depth of pipelines. As transistor sizes continue to shrink, the traditional gains associated with smaller feature sizes will be degraded due to the adverse effects of wire scaling. The consequences of technology scaling on circuit performance have recently become a topic of significant importance, especially in arithmetic circuitry such as digital multipliers, which exhibit highly irregular interconnections.

A digital multiplier architecture will be introduced that alleviates some of the problems associated with interconnect scaling, in addition to allowing for simple variable precision reconfiguration. Regulated by a 2-bit control signal, the multiplier is capable of true double and single precision multiplication, as well as fault tolerant and dual throughput single precision execution. The architecture proposed in this paper is centred on a recursive multiplication algorithm by Danysh and Swartzlander, where a large multiplication is carried out using recursions of simpler base multiplier modules. This multiplication algorithm presents greater regularity in design than standard column compression multipliers, while avoiding the linear latency of array multipliers.

A separate investigation of the recursive multiplication scheme has lead to favourable results for a design methodology recommended for future arithmetic architectures, which makes use of the proposed “locally optimized array” paradigm. Furthermore, a study of column compression techniques will be presented; this includes a novel suggestion for an optimized 4:2 compressor distribution in partial product reduction trees, and an overview of the transistor level configuration of arithmetic cells.

Acknowledgments

I would like to extend my sincere gratitude and appreciation to a number of people who have contributed to the completion of this thesis, including my colleagues, mentors and family.

First and foremost, I would like to thank my supervisor, Dr. Majid Ahmadi, for whom I have the utmost respect and admiration. His confidence in my capabilities, his enthusiasm towards my research and his guidance and assistance in all of my endeavours, have made a tremendous impact on me academically, socially and personally, and for this I am forever in his debt.

I am also grateful to Dr. Graham Jullien for his knowledgeable suggestions and insight on the field of computer arithmetic, and for granting me the opportunity to carry out research, and share my findings with students at the University of Calgary. I would also like to thank the faculty at the University of Windsor, including Dr. W.C. Miller for sharing his wisdom and his words of inspiration for a career in research, and my committee members, Dr. Arunita Jaekel and Dr. Xiang Chen for their patience and support.

Additionally, I would like to credit Mr. Till Kuendiger and Mr. Roberto Muscedere, two of the brightest people I have had the pleasure of working with, for their endless support in the lab environment, and for being instrumental in the completion of this thesis. Finally, I would like to acknowledge my colleagues at the University of Windsor, Mr. Perta, Howard and Soltis, and my parents for their assistance, encouragement and motivation.

Table of Contents

<i>ABSTRACT</i>	<i>iv</i>
<i>ACKNOWLEDGEMENTS</i>	<i>v</i>
<i>LIST OF FIGURES</i>	<i>x</i>
<i>LIST OF ABBREVIATIONS</i>	<i>xiv</i>

Chapter 1 Introduction to Computer Arithmetic 1

1.1	History of Computer Arithmetic	2
1.2	Current Trends in Computer Arithmetic	5
	Low Power Design	6
	High Throughput Systems And Pipelining	6
	Technology Scaling Effects	9
1.3	Thesis Overview	10
	Thesis Highlights	10
	Thesis Organization	11

Chapter 2 Digital Multiplication Overview..... 12

2.1	Basics of Digital Multiplication	13
2.2	Serial Multiplication Schemes.....	15
	Shift-Add Multiplication	15
	High Radix Multipliers	16
2.3	Parallel Multipliers	17
	Column Compression Multipliers	18
	Array Multipliers	21
	Latency Approximations	22
2.4	Number Systems.....	23

	Signed Fixed Point Numbers	24
	Redundant Number Systems	25
	Residue Number Systems	26
	Logarithmic Number Systems	27
	Floating Point Number System	27
2.5	Summary	30

Chapter 3 Partial Product Reduction Techniques32

3.1	Booth Recoding	32
3.2	CSA Reduction Schemes.....	35
	Wallace and Dadda Trees	36
	Minimum Reduction Stage Requirement	38
	Minimum Full Adder Requirement	40
	Variations of CSA Trees	43
3.3	High Order Counters And Compressors	46
	Counters	46
	Compressors	48
3.4	4:2 Compressors	49
	Structure Of 4:2 Compressors	49
	Proposed Optimized Compressor Layout	51
3.5	Low Power	63
	Leakage And Short Circuit Power Dissipation	64
	Dynamic Power Dissipation	64
	Dynamic Power Management	67
3.6	Threshold Logic	70

Chapter 4 Arithmetic Circuitry.....72

4.1	Logic Styles	73
	Static CMOS	74
	Transmission Gate Logic	76
	Dynamic Logic Families	77
	Differential And Dual Rail Logic Families	82
4.2	Pass Transistor Logic.....	83
4.3	Full Adder Circuits	85
4.4	4:2 Compressor Circuits	95
4.5	Overview of Arithmetic Circuitry	101
	Logic Style Selection	101
	Simulation Setup and Environment	107

Chapter 5	<i>Interconnect Effects</i>	114
5.1	Projected Issues With Technology Scaling	116
5.2	Physics of Wire Interconnects	118
5.3	Interconnect Effects On Arithmetic Circuitry	124
5.4	Locally Optimized Arrays	126
Chapter 6	<i>Recursive Multiplication</i>	131
6.1	Overview of the Recursive Multiplication Algorithm	132
	Background Information	132
6.2	6:2 Reduction Circuitry	135
6.3	Analysis Of The Base Multiplier	138
Chapter 7	<i>Reconfigurable Multiplier Architecture</i>	145
7.1	Introduction to DSP Multiplication	146
	Computation Parallelism	147
	Variable Data Width	147
	Fault Tolerance	148
7.2	Reconfigurable Architectures	152
7.3	Proposed Multiplier	153
	Double Precision Mode	155
	Single Precision Mode	155
	Dual Single Precision Mode	157
	Single Precision Fault-tolerant Mode	158
Chapter 8	<i>Modeling and Simulation</i>	159
8.1	HDL Model	159
8.2	Implementation and Layout	162
8.3	Simulation Results	163
8.4	Design Highlights	168

Chapter 9	Conclusions.....	170
9.1	Summary of Contributions	170
	Algorithmic Contributions	170
	Architectural Contributions	171
	Transistor Level Contributions	172
9.2	Conclusions	172
REFERENCES.....		174
Appendix A	Complete 4:2 Compressor Analysis	186
Appendix B	Interconnect Analysis of Various Multiplier Sizes	188
Appendix C	Verilog HDL Code for the Reconfigurable Multiplier	195
Appendix D	Recursive Multiplier Base-Multiplier Analysis.....	214
Appendix E	Component Breakdown of the Reconfigurable Multiplier	215
Appendix F	Simulation Reports and Logs.....	222
Vita Autoris		230

List of Figures

Figure 1.1	(a) Original system taking t time to complete (b) System divided into 5 time equivalent sections (c) Latched pipeline arrangement of subdivided system	7
Figure 1.2	Relative frequency of Intel devices if fabricated on the same process	8
Figure 2.1	Interrelation of arithmetic operations in modern computing devices	13
Figure 2.2	A 4x4-bit multiplication leading to an 8-bit product	14
Figure 2.3	A simple dot diagram of 16-bit partial product array	15
Figure 2.4	Shift-add multiplier implementation.....	16
Figure 2.5	Shift-add implementation of a basic radix-4 multiplier.....	17
Figure 2.6	Schematic representation of the tree multiplier process [20]	18
Figure 2.7	Carry Save Adder (CSA) Array.....	19
Figure 2.8	Carry Save Adder (CSA) used for 3:2 compression.....	19
Figure 2.9	3:2 Compression scheme carried using a CSA array [8]	20
Figure 2.10	One sided CSA tree forming an Array Multiplier	21
Figure 2.11	A 5-bit array multiplier layout (the boxes represent full-adders) [8]	22
Figure 2.12	IEEE floating point standard word widths for (a) single precision (b) double precision.....	28
Figure 2.13	A floating-point Adder / Subtractor scheme [8]	29
Figure 2.14	A floating-point multiplication scheme [8]	30
Figure 3.1	Dot diagram for a Booth-2 16-bit Multiplication [20].....	34
Figure 3.2	The Carry Save Adder - (a) Dot diagram of individual full-adder cell (b) Several full-adders used to form one level of a CSA	37
Figure 3.3	Variations of multiplier partial product reduction trees using CSAs [21] (a) Dadda implementation. (b) Wallace implementation.....	37
Figure 3.4	Actual expansion values and theoretical bounds of minimum full adder requirements in multi-operand addition	39
Figure 3.5	Algebraic representation of minimum full adder requirements	41
Figure 3.6	An 8x8 multiplication partial product matrix	41
Figure 3.7	Array of arrays layout as outlined in [10].....	44

Figure 3.8	Logic diagram of a Full-Adder	45
Figure 3.9	Examples of Dot Representations of Parallel Counters	47
Figure 3.10	Examples of Dot Representations of Multi-column Counters	47
Figure 3.11	Cascaded full adders composing a basic 4:2 compressor	50
Figure 3.12	4:2 compressor layout making use of fast input/output paths	51
Figure 3.13	A minimized gate level representation of a 4:2 compressor	51
Figure 3.14	Definition of a 4:2 compressor row	56
Figure 3.15	Compressor layout for a 16x16 multiplication	58
Figure 3.16	Compressor layout for a 24x24 multiplication	59
Figure 3.17	Total Number of cells in a partial product reduction tree (NFA = Number of (3,2) NCT = Number of total cells for 4:2)	61
Figure 3.18	Total Number of interconnects in a partial product reduction tree (IFA = Interconnects for (3,2) IC = Interconnects for 4:2)	61
Figure 3.19	Total Delay (DFA = (3,2) scheme DC = [4:2] scheme)	62
Figure 3.20	Interchanging rows within an Array Multiplier structure	69
Figure 3.21	Hybrid MSB first Array Multiplier structure	69
Figure 3.22	Threshold Gate implementation of a 4-input AND gate [70]	70
Figure 4.1	Static CMOS logic cell depicting the NFET and PFET networks	75
Figure 4.2	(a) NFET threshold voltage loss (b) PFET threshold voltage gain	75
Figure 4.3	Transmission Gates (a) Transistor level (b) Logic symbol	76
Figure 4.4	Capacitive Nodes (a) Basic circuit (b) Storage Capacitor model	77
Figure 4.5	Voltage drop due to leakage effects in an NFET	79
Figure 4.6	Voltage rise due to leakage effects in a PFET	79
Figure 4.7	Primitive representation of a Precharge-Evaluate Logic block [78]	80
Figure 4.8	A DOMINO logic block [78]	80
Figure 4.9	Single-Phase Logic Circuit Types [78] (a) Single Phase Network setup using latches (b) Single Phase Logic gates cascaded together	81
Figure 4.10	Cascode Voltage Switch Logic (CVSL) (a) Static CVSL (b) Dynamic CVSL	83
Figure 4.11	Basic Pass Transistor Logic Configurations	84
Figure 4.12	AND gate implementations using pass logic	84
Figure 4.13	CPL Implementation of an AND gate	85
Figure 4.14	Gate level (a) and transistor level (b) implementations of 12 transistor full-adder cell proposed in [83]	87
Figure 4.15	A DOMINO logic full adder cell [84]	87
Figure 4.16	Various XOR/XNOR configurations (a) Transmission Gate (b) Transmission Gate with driving outputs (c) Inverter Based (d) Proposed XOR/XNOR Configuration [85]	88
Figure 4.17	Transmission Gate based Full-Adder circuit	89
Figure 4.18	Pass-transmission implementation of a Full Adder	90
Figure 4.19	10 Transistor Full Adders outlined in [89]	90

Figure 4.20	Conventional 28 transistor CMOS full adder implementations (a) standard configuration (b) Mirror cell configuration [84].....	91
Figure 4.21	Low power 16 transistor full adder cell	92
Figure 4.22	Low power 10 transistor full adder cell [91]	92
Figure 4.23	LEAP full adder cell	94
Figure 4.24	Complementary Pass-Transistor Logic (CPL) full adder cell	94
Figure 4.25	1.2 ns 4:2 compressor proposed in [100].....	96
Figure 4.26	Precharged pass logic compressor [56]	97
Figure 4.27	Multiplexer cell based pass-logic 4:2 compressor [57]	97
Figure 4.28	Gate level framework for DPL based compressor cell in [102]	99
Figure 4.29	DPL compressor using Full swing & Non-Full Swing MUX cells [103] ..	99
Figure 4.30	Pass-transmission implementation of the 4:2 compressor [98]	100
Figure 4.31	Threshold voltage floor with decreasing supply voltage [104]	103
Figure 4.32	Threshold voltage tolerance dependence on channel length [104]	103
Figure 4.33	Typical test-bench used for Full-Adder simulations [91].....	109
Figure 4.34	4:2 Compressor distribution in an 8x8 reduction matrix	109
Figure 4.35	Full-adder distribution in an 8x8 reduction matrix.....	110
Figure 4.36	Cadence testbench for 8x8 multipliers.....	110
Figure 4.37	Power measurement circuits proposed in [109] having a current source and parallel RC circuit (a) current controlled source (b) voltage controlled source	112
Figure 5.1	Contribution of interconnect effects to overall delay	117
Figure 5.2	Coupling capacitances associated with metal interconnects.....	122
Figure 5.3	Interconnect RC delay per unit length (ps / mm).....	122
Figure 5.4	Relative delay; binary tree to its no wire implementation for Booth 2, double precision multiplication [12]	125
Figure 5.5	Interconnect effects with respect to multiplier width (a) Total chip interconnect length (b) Average chip interconnect length.....	129
Figure 6.1	Dot diagram of a single level recursive n-bit multiplication	134
Figure 6.2	A schematic of a single level recursive multiplier.....	134
Figure 6.3	Standard 6:2 reduction macrocell composed of 3 stages of full adders...	135
Figure 6.4	6:2 Macrocells capable of receiving a variety of input bits (a) 5 input (b) 4 input (c) 3 input (d) 2 input	137
Figure 6.5	Novel 6:2 Reduction Block.....	138
Figure 6.6	Delay associated with various base-multiplier sizes (a) Array base multipliers (b) Dadda base multipliers.....	141
Figure 6.7	Relationship between recursive multiplier size and base-multiplier size	143
Figure 6.8	Base multiplier comparisons for a 64-bit recursive multiplier (a) total cell count (b) total interconnect segment count	144
Figure 6.9	Percentage increase in cell count and interconnect count for various sizes of Array and Dadda base multipliers	144
Figure 7.1	Time redundant RETWV error correcting multiplier [144]	151

Figure 7.2	Outline of the reconfigurable multiplier	154
Figure 7.3	Default double precision mode	156
Figure 7.4	Single precision mode	156
Figure 7.5	Dual single precision mode	157
Figure 7.6	Single precision - fault tolerant mode	158
Figure 8.1	(a) Top level module of the HDL model (b) Top level module outlining the input output clocked-latching circuitry	160
Figure 8.2	An illustrated representation of the HDL model of the multiplier	161
Figure 8.3	Layout view of the Reconfigurable Multiplier	163
Figure 8.4	Breakdown of the Reconfigurable Multiplier (a) Area (b) Power (c) Delay	164

List of Abbreviations

<i>ALU</i>	<i>Arithmetic Logic Unit</i>
<i>ASIC</i>	<i>Application Specific Integrated Circuit</i>
<i>BiCMOS</i>	<i>Bipolar CMOS</i>
<i>BSDC</i>	<i>Binary Stored Double Carry</i>
<i>CLA</i>	<i>Carry Look-Ahead Adder</i>
<i>CMF</i>	<i>Common Mode Failure</i>
<i>CMOS</i>	<i>Complementary Metal-Oxide Semiconductor</i>
<i>CORDIC</i>	<i>COrdinated Rotation DIgital Computer</i>
<i>CPL</i>	<i>Complementary Pass-Transistor Logic</i>
<i>CPU</i>	<i>Central Processing Unit</i>
<i>CSA</i>	<i>Carry Save Adder</i>
<i>CTL</i>	<i>Capacitive Threshold Logic</i>
<i>CVSL</i>	<i>Cascode Voltage Switch Logic</i>
<i>DA</i>	<i>Distributed Arithmetic</i>
<i>DBNS</i>	<i>Double Base Number System</i>
<i>DC</i>	<i>Direct Current</i>
<i>DCSL</i>	<i>Differential Current Switch Logic</i>
<i>DCT</i>	<i>Discrete Cosine Transform</i>
<i>DIBL</i>	<i>Drain-Induced Barrier Lowering</i>
<i>BIDO</i>	<i>Bi-Directional Operation</i>
<i>DPL</i>	<i>Dual Pass-Transistor Logic</i>
<i>DSP</i>	<i>Digital Signal Processor</i>
<i>DVL</i>	<i>Dual Value Logic</i>
<i>ECDL</i>	<i>Enable-disable Cmos Differential Logic</i>

EDVAC	Electronic Discrete Variable Automatic Computer
ENIAC	<i>Electronic Numerical Integrator and Computer</i>
FOX	<i>Field Oxide</i>
FPGA	<i>Field Programmable Gate Array</i>
GND	<i>Ground Terminal or Voltage</i>
HDL	<i>Hardware Description Language</i>
IC	<i>Integrated Circuit</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
KOA	<i>Karatsuba Ofman Algorithm</i>
LNS	<i>Logarithmic Number System</i>
LSB	<i>Least Significant Bit</i>
MBA	<i>Modified Booth Algorithm</i>
MIPS	<i>Millions of Instructions Per Second</i>
MOS	<i>Metal-Oxide Semiconductor</i>
MOSFET	<i>Metal-Oxide Semiconductor Field Effect Transistor</i>
MSB	<i>Most Significant Bit</i>
MSI	<i>Medium Scale Integration</i>
MUX	<i>Multiplexer Cell (usually assumed as a 2:1 multiplexer)</i>
NFET	<i>N-type Field Effect Transistor</i>
PDP	<i>Power Delay Product</i>
PFET	<i>P-type Field Effect Transistor</i>
PT	<i>Pass-Transistor</i>
RAP	<i>Reconfigurable Arithmetic Processor</i>
REDWC	<i>REcomputing with Duplication With Comparison</i>
REMOD	<i>Reprocessing with Micro Delays</i>
RETWV	<i>REcomputing with Triplication With Voting</i>
RISC	<i>Reduced Instruction Set Computer (Computing)</i>
RMS	<i>Root Mean Square</i>
RNS	<i>Residue Number System</i>
SIA	<i>Silicon Industry Association</i>
SIMD	<i>Single Instruction Multiple Data</i>
SSDL	<i>Sample-Set Differential Logic</i>
SSI	<i>Small Scale Integration</i>
TDM	<i>Three Dimensional Minimization</i>
TG	<i>Transmission Gate</i>
THL	<i>High-to-Low Transition Time</i>

<i>TL</i>	<i>Threshold Logic</i>
<i>TLH</i>	<i>Low-to-High Transition Time</i>
<i>TMR</i>	<i>Triple-Modular Redundancy</i>
<i>TOSHIRO</i>	<i>Fault Tolerance by Shifted and Rotated Operands</i>
<i>TSMC</i>	<i>Taiwan Semiconductor Manufacturing Company Inc.</i>
<i>VDD</i>	<i>Power Supply Terminal or Voltage</i>
<i>VLSI</i>	<i>Very Large Scale Integration</i>
<i>VSS</i>	<i>Lowest Chip Voltage (usually equivalent to GND)</i>
<i>VT</i>	<i>Threshold Voltage</i>

Chapter 1

Introduction to Computer Arithmetic

The computer has permeated every facet of modern society, and has become an integral part of social order. There is a long history, dating back several centuries, of mathematicians and scientists developing computing machines to facilitate the manipulation of numbers. With this intrigue of computing machines was born the field of computer arithmetic. The history of computer arithmetic is closely connected with the development of the digital computer, and was one of the primary areas of research in early electronic computers.

Computer arithmetic is a sub-field of computer organization and architecture, dealing with the implementation of arithmetic algorithms on hardware and/or software, and is a fundamental aspect of computer processor architecture and the arithmetic logic unit (ALU). One of the principle focuses of this field is the development of hardware algorithms and circuitry to enhance the performance of numerical applications. The discussion on computer arithmetic begins with a look at the evolution of the computer, followed by a review of the advancements in arithmetic algorithms over the past century.

1.1 History of Computer Arithmetic

The computer was essentially developed as a means to facilitate numerical calculations, and its origins may be traced to the mechanical adding machine created by Blaise Pascal in 1642, and the various modifications made to the original concept throughout the next several decades. The English mathematics professor, Charles Babbage, drafted a proposal for a steam powered Difference Engine in 1822, capable of solving differential equations. A decade later he prepared plans for his second steam powered machine known as the Analytic Engine, which would be a real parallel decimal computer [1] .

Babbage's inventions, although never fully realized, lay down the conceptual architecture for future computers. One of his intentions was to use punch cards to store the instructions for the machine, similar to those used on a Jacquard loom. Herman Hollerith, of the US census bureau, further explored this concept, where he used punch cards to facilitate the tabulation of census results. Hollerith's tabulator became so successful that he started his own firm to market the device in 1896; his company, through a series of mergers, eventually became International Business Machines (IBM) in 1924 [2] .

Alan Turing introduced the concept of the finite state machine in 1936. In his paper, Turing proposed a machine that would determine an output (or next) state based on its current state and an input value. The nineteenth century English mathematician and logician, George Boole, developed ways of expressing logical processes using algebraic symbols, creating a branch of mathematics known as symbolic logic. Boole's clarification of the binary system of algebra was the basis of the first binary computer, the Z3, developed by the German Engineer Konrad Zuse in 1941. At the same time, spurred on by the strategic importance of computing machines during the world war, the British manufactured their own code-breaking computer known as the Colossus [2] .

In the US, the maturity of the modern computer continued with the works of IBM and Harvard engineer Howard H. Aiken, on the first all-electronic calculator, the Harvard-IBM Automatic Sequence Controlled Calculator, or Mark I for short. Another computer

development spurred by the war was the Electronic Numerical Integrator and Computer (ENIAC), produced by a partnership between the U.S. government and John W. Mauchly and John Presper Eckert at the University of Pennsylvania. ENIAC is generally acknowledged to be the first successful high-speed electronic digital computer (EDC) and was productively used from 1946 to 1955 [3] .

One of the pioneers of modern computing, John von Neumann, initiated concepts in computer design that remained central to computer engineering for the next 40 years. Von Neumann designed the Electronic Discrete Variable Automatic Computer (EDVAC) in 1945, having memory to hold a stored program as well as data, and a central processing unit coordinating all of the computer's processes. This "stored memory" technique, as well as the "conditional control transfer", permitted the computer to be stopped at any point and then resumed, allowing for greater versatility in computer programming.

John von Neumann demonstrated that a computer could have a simple, fixed structure, yet be able to execute any kind of computation given properly programmed control without the need for hardware modification. Von Neumann's contributions resulted in a new understanding of the organization of practical fast computers architectures. As a result of these techniques and several others, computing and programming became faster, more flexible, and more efficient, with the instructions in subroutines performing far more computational work. These ideas, often referred to as the stored-program technique, became the universally adopted rudiments for future generations of high-speed digital computers.

The advent of the transistor in 1948 proved to be the next breakthrough in computer technology. By 1956, the transistor was employed in computers, and coupled with magnetic core memory, they enabled a new generation of faster, smaller and more efficient class of computing machines. These machines were predominantly used for numerical applications, and were used by the government, academia and large businesses.

Jack Kilby, an engineer with Texas Instruments, introduced the concept of the integrated circuit (IC) in 1958, and over the next few years, others were able to integrate more and more components onto a single semiconductor device. In the early seventies, Intel incorporated many facets of the computer onto a single IC, and named their revolutionary design the microprocessor. Thus a new physical archetype for the computer was in place, and although the device dimensions continue to diminish, the model has remained relatively unchanged [3] .

The fundamental nature of the computer is a machine for performing numerical calculations efficiently. With this in mind, the manner in which those calculations are carried out becomes a significant factor in the performance of the machine, thus fostering the need for devoted research in the area of computer arithmetic. Most of the essential methodologies, number systems, and algorithms for the various arithmetic operations were developed in the mid twentieth century. In this era the computer's purpose was restricted to numerical applications, consequently a great deal of emphasis was placed on the advancement of the arithmetic logic of computers.

The historical perspective of computer arithmetic begins with the notions of addition and multiplication set forth by Charles Babbage. Modern hardware algorithms such as the concept of complement representation for subtraction, shift-add multiplication and division were fine tuned in the 1940's. In this time period, the focus of researchers was to prove the feasibility of the computer, and once this was demonstrated, the onus was placed on speed-up techniques. In the 1950's, practically all of the significant modern fast addition algorithms had been published. In addition, concepts of residue arithmetic and CORDIC algorithms were presented.

The 1960's saw the development of redundant, floating point and high radix arithmetic concepts. Significant advancements in digital multiplier algorithms were also carried out in this time period. Specifically, Wallace [4] and Dadda [5] published their work on tree multipliers, and partial product reduction concepts. Word lengths for floating point numbers were also standardized [6] for greater arithmetic functionality across all

platforms . The advent of vector computers and microprocessors in the 1970's, forced the requirement for higher throughput from arithmetic hardware. Shortly thereafter, VLSI technology enabled low cost, high performance embedded arithmetic circuitry. This in turn forced the enhancement of pipelining techniques in order to meet the processing demands of the new breed of computers.

1.2 Current Trends in Computer Arithmetic

The evolution of the microprocessor, and the current employment of deep-submicron technology, has forced the reconsideration of all arithmetic design. Pin and interconnect limitations are now of primary concern, consequently refinement of arithmetic algorithms, and hybrid designs have emerged as the focal points of modern arithmetic development. The shift in technology from SSI (Small Scale Integration) to VLSI (Very Large Scale Integration) has shifted emphasis from algorithm development to hardware level optimization of existing techniques.

As the function, capability and applications of the computer evolves, so too must the attitude towards the design of the modern computer. For the most part, algorithm development has been overshadowed by the significant improvements achievable by adapting existing methodologies to exploit contemporary VLSI technology. Reduction of gates in circuit implementations has taken a back seat to the study of interconnect effects, and the implementation of optimal circuitry on a transistor level.

Furthermore, with the increasing impact of portable devices, low power design has transpired into a critical area of exploration in the field of computer arithmetic, and hardware development. With new technologies and application requirements, the continued emergence of design challenges in the field of computer arithmetic is inevitable. This section focuses on the current trends in computer arithmetic and architecture, and their implication on digital multiplier designs.

1.2.1 Low Power Design

Low power techniques have emerged as the foremost area of interest in modern VLSI design. The recent awareness of this critical field of research is due to concerns over excessive heat dissipation, and the mounting yearn for portable devices. It is intuitive to appreciate the fact that low power devices will inevitably generate less heat, in addition to placing a smaller burden on portable batteries. Area efficiency and gate count have passed the torch of dominant design criteria to low power and low cost strategies.

Of the significant areas of research in low power design policy, those most prevalent to arithmetic design include low power logic circuit families, *Dynamic Power Management*, and reduced switching activity configurations. Asynchronous circuitry eliminates the need for a complex clock tree eradicating a considerable source of power dissipation, estimated at 15 to 45% of total power [7]. Though some may argue that the added overhead for handshaking circuitry may be justified by the savings in power and potentially delay [8], this technology has limited applications in a pure digital multiplier.

Circuit packaging and battery life now pose severe restrictions on many designers, including those engaged in computer architecture. It has been predicted that power dissipation in high performance processors will exceed packaging limits by 25 times in a matter of 15 years [9]. With the increased demand in performance from mobile computing instruments, power conscious strategies must permeate down to processor, and arithmetic unit design.

1.2.2 High Throughput Systems And Pipelining

To speed up arithmetic operations, the input-to-output latency, or simply the time between the application of inputs signals and the arrival of outputs, must be reduced. In order to sustain high throughput (number of operations performed per unit time) and consequently high clock rates in systems, pipelining strategies are employed allowing for concurrent operations. Although this technique has been in existing for a few decades, its impact on computer architecture continues to mount with every new major processor device.

Current designs are incapable of satisfying the clocking demands of most high-speed processors; for this reason concurrency, or one hardware unit performing multiple overlapped processes at once, is employed. Pipelining allows a given piece of hardware to be sub-divided into stages. The stages are separated by clocked (or asynchronous) registers; this all amounts to what is often referred to as the pipeline hardware overhead. By separating the large block into stages, the portions of the circuit that would typically be idle are capable of processing the next set of signals, and so forth. This in effect substantially increases circuit efficiency and throughput. Thus, although an operation will have a higher latency for a single computation due to the added hardware overhead, the overall system will have higher total throughput.

The following is a crude example of the basic principles of pipelined circuitry. Consider a function that takes time t to complete from input to output as in Figure 1.1 (a). Subdividing the function into 5 subsections each of equivalent delay results in Figure 1.1 (b). Placing latches allows the capture of the output of each subsection, thus a new input may be applied at $t/5$ time intervals (Figure 1.1 (c)). This produces an increase in throughput by a factor of 5.

Figure 1.1 (a) Original system taking t time to complete
 (b) System divided into 5 time equivalent sections
 (c) Latched pipeline arrangement of subdivided system

Pipelining techniques are employed in the majority of computing devices, including all of the major processor families. The degree to which a device is pipelined, known as the pipelining depth, is based on the target frequency and the number of gates that may be included per pipeline stage. The deeper the pipeline depth, the more stages per pipeline, the higher the clock speeds. However, this comes at the expense of increased system complexity, and added buffering to accommodate the longer pipelines.

Figure 1.2 depicts a normalized clock frequency graph of Intel's last six processor cores if they were all based on the same technology [10]. With the effects of technology scaling aside, it becomes apparent that as the depth of the pipelining increases (Pentium III and Pentium IV), so too does the relative clock frequency.

Figure 1.2 Relative frequency of Intel devices if fabricated on the same process

For a highly efficient pipeline, wave-pipelining techniques may be explored [8]. Wave pipelining makes use of the inherent delay through a pipelined segment as a temporary storage mechanism, allowing for 'waves' of unlatched data to be transmitted. Typical pipelining techniques employ temporal separation between consecutive signals, as a result achieving spatial separation as well. Wave pipelining, on the other hand, draws on spatial separation of consecutive signals that have no set temporal separation [8]. This method

alleviates the requirement for pipelining overhead, yet demands highly accurate delay predictions within the circuitry and the clock network enabling the inputs and outputs.

In the forthcoming generations of digital signal processors, and microprocessors, the dependence on pipelining methodologies will continue to intensify. Fully distributed micro-pipelines, systolic arrays, and asynchronous pipeline control will have a significant impact on the ability of industry meeting performance demands.

1.2.3 Technology Scaling Effects

Researchers have turned towards advancements in process technology in order to satisfy the ever-increasing demand for high-speed processors, and computational systems. As current devices delve into deep sub-micron technology, not only does the device geometry decrease, but switching times, and operating voltages also scale down. These gains come at the expense of increased layout complexity, and a greater susceptibility to parasitic effects in the interconnections.

We are rapidly approaching the era where interconnect effects due to poor wire scaling are forcing the reconsideration of conventional circuit topologies. According to the International Technology Roadmap for Semiconductors (ITRS 2001), we are beginning to reach the fundamental limits of the materials in the planar CMOS process [9]. As transistor sizes continue to shrink, the traditional gains associated with smaller feature sizes will be degraded due to the adverse effects of wire scaling. There has been recent awareness of the drastic effects of interconnect delay in VLSI implementations [11-16], and several investigations focused on this problem have been linked directly to multiplier structures.

It has been projected that over a span of six technology generations, wires that are scalable will have delays that will become worse by a factor of four relative to gate delays. This problem escalates for non-scalable, global wires, where the wire delay relative to gate delay will double for each generation [15]. For the most part, conventional analysis of

partial product reduction trees has been simplified through the use of compressor or gate delays. These models suffer in accurately modeling the performance of the structure since the associated interconnect delay, and capacitive coupling effects are ignored. Chapter 5 will provide an in-depth survey of interconnect scaling effects.

The implications of technology on digital design may no longer be brushed off as secondary importance to algorithm design. The era of elegant algorithm development by simply observing gate delays and gate counts has passed. Future experts in the field of computer architecture must also have a concrete understanding of the fundamentals of technology effects, and the trends in design. There have been significant advances in micro-architectures in order to cope with evolving technology issues, a comprehensive survey of which is provided in [17]. Such microprocessor level advancements in instruction coding, memory allocation, and system level modifications are beyond the scope of this thesis.

1.3 Thesis Overview

1.3.1 Thesis Highlights

This thesis will present a general investigation of digital multiplication from various levels of abstraction, and will highlight a novel reconfigurable multiplication architecture. The proposed design utilizes a variation of the “divide and conquer” or “recursive” multiplication scheme presented by Swartzlander et al. [18]. The principle advantage of this scheme lies in its multi-mode reconfiguration ability, allowing the user to select between four modes of operation:

- Double Precision (64-bit) Multiplication (default)
- Single Precision (32-bit) Multiplication
- Dual Single Precision Multiplication (double throughput)
- Single Precision Fault Tolerant Multiplication through a Majority Voter

The proposed scheme combines many desirable design characteristics, such as low power dissipation, high throughput capabilities, fault tolerance, and increased regularity reducing

interconnect congestion. Moreover, a 64-bit reconfigurable multiplier, with potential applications in Digital Signal Processor (DSP) devices, has been implemented using the TSMC 0.18 μm technology. This design has been contrasted against a standard high-performance architecture of equivalent size, and has demonstrated promising results, which will be presented in chapter 8.

In addition, other investigations into partial product reduction methodologies using counters and compressors will be addressed. In particular, an optimized 4:2 compressor distribution methodology for partial product reduction arrays will be presented. Furthermore, a design framework using the novel “locally optimized array” formalism will be introduced as a potential solution to the interconnect dilemma in future high performance architectures.

1.3.2 Thesis Organization

The thesis will begin with a general overview of the concept of digital multiplication, and the various multiplication algorithms in chapter 2. The various partial product reduction schemes used in modern multipliers will be the focus of chapter 3. This is a topic of particular importance since it is the partial product reduction scheme that distinguishes the individual multiplication schemes and their inherent characteristics. Chapter 4 will present the arithmetic sub-cells, and the numerous logic styles that are used in makeup of the partial product reduction strategies.

One of the topics of emerging importance in digital design, namely the wire interconnect, will be discussed in considerable detail in chapter 5. The physics behind these thin slices of metal, the trends with diminishing device sizes, and their effects on arithmetic performance will be presented. Chapter 6, 7 and 8 will focus on the introduction of the reconfigurable architecture, beginning with the outline of the recursive multiplication algorithm in chapter 6, followed by implementation and simulation results in chapter 7 and 8 respectively. The thesis will end with some closing remarks in chapter 9.

Chapter 2

Digital Multiplication Overview

In modern digital systems, the component responsible for handling the arithmetic operations is known as the Arithmetic Logic Unit (ALU). Arithmetic units, for the most part, lie in the critical data path of the core data processing elements. These include microprocessors (CPU), digital signal processors (DSP), in addition to application specific (ASIC) and programmable (FPGA) processing and addressing integrated circuits. Naturally the performance of the system, in regards to numerical applications, is directly related to the structure and design of the ALU.

The numerical operations carried out by the arithmetic unit may include, but are not limited to:

- addition/subtraction	- multiplication
- shift/extension	- division
- comparison	- square root extraction
- increment/decrement	- exponential function
- complement	- logarithm function
- trigonometric functions	- hyperbolic functions

As depicted in Figure 2.1 on page 13 [19], one of the critical functions carried out by the ALU is multiplication. Although not the most fundamentally complex operation, digital multiplication is

one of the most extensively used operations in signal processing, and other scientific applications. For this reason, it is one of the most widely studied areas of the field of computer arithmetic.

Figure 2.1 Interrelation of arithmetic operations in modern computing devices

2.1 Basics of Digital Multiplication

Prior to exploring the various multiplication algorithms, and the applications of each, it is imperative to present the essence of digital multiplication, and the standard nomenclature. Just as in the paper and pencil methodology of carrying a multiplication of two values, digital multiplication entails a sequence of additions carried out on partial products. The means by which this partial product array is summed to yield the final product is the key distinguishing factor amongst multiplication schemes.

In general, the partial product array for an $M \times N$ bit multiplication is formed by the bitwise logical AND of the *multiplicand* A and *multiplier* X , where:

$$X = [x_m, x_{m-1}, x_{m-2} \dots x_2, x_1, x_0]$$

$$A = [a_n, a_{n-1}, a_{n-2} \dots a_2, a_1, a_0]$$

The summation of the partial products will yield an $(n+m)$ -bit product, P , where:

$$P = [\sigma_{n+m}, \sigma_{n+m-1}, \sigma_{n+m-2} \dots \sigma_2, \sigma_1, \sigma_0]$$

The partial product array will have $(n \times m)$ bits, arranged in m rows of n -bit values. The array is in essence composed of a sequence of rows that are either shifted versions of the *multiplicand*, A , or zeros, according to the bits of the *multiplier*, X . The multiplication of two 4-bit values is illustrated in Figure 2.2.

				a_3	a_2	a_1	a_0
			X	x_3	x_2	x_1	x_0
				x_0a_3	x_0a_2	x_0a_1	x_0a_0
				x_1a_3	x_1a_2	x_1a_1	x_1a_0
				x_2a_3	x_2a_2	x_2a_1	x_2a_0
				x_3a_3	x_3a_2	x_3a_1	x_3a_0
s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

Figure 2.2 A 4x4-bit multiplication leading to an 8-bit product

To better visualize the partial product reduction process, the concept of dot diagrams shall be introduced [20][21]. A dot diagram is a visual representation of the bits in an algorithm, where in this particular application the dots represent individual partial product bits. The nature of the dot diagram is to depict the bits using the relative position of individual bits, and the manner in which they are manipulated, irrespective of the actual value of each bit.

Figure 2.3 on page 15 shows the partial product array for a 16x16-bit multiplication [20]. The partial products are shifted to account for the differing arithmetic weight of the bits in

the multiplier, where dots of the same arithmetic weight are aligned vertically. The final product, represented by the double length row of dots at the bottom, is obtained via the summation of the dots in each column.

Figure 2.3 A simple dot diagram of 16-bit partial product array

2.2 Serial Multiplication Schemes

2.2.1 Shift-Add Multiplication

In its most basic form, digital multiplication may be carried out through a sequence of shifts and additions of the *multiplicand* to the *partial product* register, governed by the individual bits of the *multiplier* (Figure 2.4 on page 16). This primitive form of multiplication, known as shift-add or iterative multiplication, although very simple in implementation, is very slow. The number of iterations, or cycles of addition, that are required grows linearly with the size of the *multiplier*, with each cycle having a delay of the required fast adder.

Figure 2.4 Shift-add multiplier implementation

2.2.2 High Radix Multipliers

A variation of this rudimentary form of digital multiplication is the high-radix multiplication algorithm. Though fundamentally identical to the shift and add algorithms, these multipliers accept more than one bit of the *multiplier* on each clock cycle. This process reduces the number of clock cycles required to carry out a multiplication, at the added expense of the requirement for the immediate availability of fixed multiples of the *multiplicand*.

Figure 2.5 on page 17 provides an outline for a radix-4 multiplication scheme, where each clock cycle now utilizes two bits of the *multiplier*, effectively doubling the throughput over a conventional radix-2 binary multiplier [8]. Note that in this scheme, a separate register is required to store the previously multiplied value of 3A. The higher the radix of the multiplier, the more stored values that will be required. Through the use of higher radix multipliers (radix-8, radix-16, etc.), the greater the achievable computation

speeds; however, this comes at the expense of increased overhead in terms of shift circuitry, and storage registers for all of the required multiples of the multiplicand.

Figure 2.5 Shift-add implementation of a basic radix-4 multiplier

2.3 Parallel Multipliers

Serial multipliers, and the concept of shift and add algorithms, are a class of primitive multiplication schemes that take advantage of simple implementation techniques. Such methods are employed where hardware overhead is an issue, or if there is a lack of a dedicated hardware multiplier. Modern high performance machines call for more sophisticated algorithms, in order to limit the computation latency.

Parallel multipliers in general may be classified into two distinct categories: linear parallel multipliers, and column compression multipliers. As opposed to the serial multiplier, parallel multipliers generate all of the partial products simultaneously. In addition, parallel multipliers limit the latency associated with carry propagation to one final fast adder.

2.3.1 Column Compression Multipliers

The foundation for the modern column compression multiplier was set forth in the 1960's by the works of C.S. Wallace, Luigi Dadda, and the Russian mathematician Yu Ofman [4][5][22]. The tree multiplier offers the potential for a logarithmic increase in delay relative to operand size. Once formed, the bits in the partial product array are passed onto a reduction network, which performs a column-wise compression of the bits, forming two final partial products. A final stage fast adder is used to sum the two resulting partial products. A schematic representation of this process is depicted in Figure 2.6 [20].

Figure 2.6 Schematic representation of the tree multiplier process [20]

The methodology initially proposed by Wallace [4], makes use of Carry-Save Adder (CSA) arrays in order to carry out the column-wise compression of the partial product bits. The CSA is the most commonly used form of multi-operand adder, and it is simply composed of a series of non-interlinked Full-Adder blocks, as shown in Figure 2.7. The dot diagrams in Figure 2.8 and Figure 2.9 on page 20, depict a simple carry save adder, and the manner in which the CSA is used as a 3:2 compression scheme on a bit array. Through the use of such compression techniques, the carry propagation is postponed until the final stage, where the resulting partial products are summed.

Figure 2.7 Carry Save Adder (CSA) Array

Figure 2.8 Carry Save Adder (CSA) used for 3:2 compression

Luigi Dadda proposed a systematic methodology for laying out the CSA reduction tree such that the minimum number of counters are used [5]. In his investigation, Dadda deduced that by determining the minimum number of required stages required for the partial product reduction process, 3:2 or even higher order counters may be placed in such

a manner as to minimize the hardware requirement. Since its inception, Dadda's minimum circuitry paradigm has been critically analyzed and confirmed [21][23], and further explored for high order and heterogeneous counter arrays [24].

Although fundamentally demonstrated to be the most expeditious multiplication scheme, the column compression process suffers from several drawbacks. The highly irregular architecture leads to arduous an inefficient VLSI layout. In addition, large area requirements, and highly irregular interconnections create the potential for signal skew effects. These topics will be elaborated upon in the forthcoming sections.

Figure 2.9 3:2 Compression scheme carried using a CSA array [8]

2.3.2 Array Multipliers

Linear parallel multipliers, often referred to as array multipliers, obtain their name from the linear relationship between their latency and operand size. The array multiplier may be regarded as a one sided CSA tree (Figure 2.10), where the reduction process occurs in ordered stages.

The highly regular layout of the array structure is depicted in 5-bit multiplier in Figure 2.11 on page 22. The systematic arrangement of the cells makes this design ideal for automated layout techniques, where the bits of the two operands are broadcast across the arrangement of full adder cells. In this scheme, the outputs of the adders trickle horizontally and vertically accordingly until the perimeter of the structure where the product bits are attained. The drawback of this scheme is that the partial products are introduced and reduced one row at a time, not in parallel as in tree multipliers. This leads to higher gate count, and slower performance.

Figure 2.10 One sided CSA tree forming an Array Multiplier

2.3.3 Latency Approximations

One of the key characteristics that differentiates array and tree multipliers is their relative computation delay. Although the forthcoming analysis is by and large accepted as the standard measure of performance amongst multiplication algorithms, it should be noted that it is merely an approximation. In reality, the actual performance calculations encompass several factors that have been neglected for simplicity. This important topic will be discussed in further detail in the subsequent chapters.

Figure 2.11 A 5-bit array multiplier layout (the boxes represent full-adders) [8]

The standard timing performance analysis for multiplication algorithms is the gate, or full adder delay. A k -bit array multiplier is first examined, using the full adder as the standard measure of delay per stage of reduction, since it is the associated delay of one CSA. Referring to Figure 2.10 on page 21, the first stage of the Array multiplier absorbs two rows of partial products with $(k - 2)$ rows remaining. Thus a total of $(k - 2) + 1$, or $(k - 1)$

total reduction stages are required. The delay of the array multiplier has a linear relationship ($O(k)$) with operand size.

The tree multiplier latency calculation is somewhat more involved, since it is necessary to determine the number of stages of reduction required for a given operand size. By considering the reduction process as a column-wise compression of the partial products, where each stage reduces the column height by a factor of $2/3$, we can deduce the relation:

$$n(h) = \left\lceil \frac{3}{2} n(h-1) \right\rceil$$

$$n(0) = 2$$

where $n(h)$ is the column height associated with an h - level tree [5][8][21][25]. Table 2.1 provides an expansion of the series that is generated through this representation, depicting the maximum column height that may be compressed down to 2 for a given number of stages. As always, symmetric multiplication (equal operand sizes) is assumed.

Table 2.1 Max column height per stage of a (3,2) column compression tree

h	0	1	2	3	4	5	6	7	8	9	10
$n(h)$	2	3	4	6	9	13	19	28	42	63	94

The above analysis may be summarized by defining the delay of a column compression multiplier as logarithmic ($O(\log(k))$) with operand size. For this reason, tree multipliers are the design of choice for large, high-speed applications, where the trade-off in irregularity and size are more than compensated for by performance gains.

2.4 Number Systems

One of the underlying considerations that must be made when carrying out digital arithmetic is the number representation. Thus far, the discussion has been centered on the binary number system, and for the most part it is the predominant number representation of choice in most digital systems. However, it is not the only number system used in

computer arithmetic applications. There are several significant classes of unconventional number systems, each having a unique advantage in a given system, such as operation speed-up or increased accuracy.

The selection of a particular number representation may radically alter the fundamental manner in which an operation is carried out. In order to select a particular number system, the relative cost-performance benefit of the various alternatives becomes a critical factor. This section will explore the shortfalls and advantages of several number systems, beginning with fixed-point numbers, followed by real number systems.

2.4.1 Signed Fixed Point Numbers

The natural number system (also known as unsigned integer), although simple to put into practice, limits the potential of the overall design. To realize sophisticated arithmetic computation, the number system of choice must be capable of signed representation. There are several means by which signed representation may be achieved.

The easiest form of signed number representation is known as signed magnitude (or sign-and-magnitude), where a sign bit is included as part of the value. Similar to the manual representation of negative numbers, a number will have a numeric value, and a sign bit in front identifying negation. Thus a k - bit system will have a $(k - 1)$ -bit magnitude description. Although conceptually simple, the signed magnitude format suffers from difficult addition, where oppositely signed values will require supplementary circuitry (such as a magnitude comparator, or a subtractor) for proper addition.

An encoding scheme may also be used to eliminate negative values during computation. Biased representations, for example, will convert everything to positive numbers by adding a fixed bias value. Also referred to as excess-bias encoding, this type of representation appears in the exponential component of floating point numbers. The downfall to this representation is the difficulty in multiplication and division, and the additional computation that is required to unbiased values.

Complement formats are the third major representation of signed numbers. A large complementation constant is added to all negative values, satisfying the condition that there is no overlap in the representation of the positive values. In binary systems, 2's complement representations, obtained by taking the ones complement (bit-wise negation) and adding 1, are used to describe negative values. The ease of negation and computation using complement formats leads to their attractiveness as a signed digit representation.

2.4.2 Redundant Number Systems

A number system with radix- R may be fully described using R distinct digits. For example the binary system (radix-2), can define any value using the 2 digit set [0 , 1]. In general a positional radix- R number system represents a k -digit value as a string of digits:

$$(d_{k-1}, d_{k-2}, \dots d_0)$$

where:

$$\sum_{i=0}^{k-1} d_i R^i$$

A system is referred to as redundant if more than R digits are used to define a radix- R representation. Redundant number systems are primarily used in digital systems for arithmetic speed-up techniques. By over defining a system using redundant values, the cost of computation of certain operations may be appreciably reduced. Addition is one such application, where use of redundant representation allows for constant time addition, since the value of the carry bit may be obtained by examining a fixed number of previous bits [26].

A simple case of redundant number systems that has already been presented is the carry-save representation of a number. By describing a k -bit value using two k -bit numbers in the carry save format, [0, 1, 2], the necessity for carry propagation is alleviated. The description of a value using what are essentially twice as many bits may be clearly justified when considering the array multiplier. By maintaining the partial product

summation in carry save form, the latency of each stage is reduced to only one full adder delay, since carry propagation is postponed until the final stage.

Other redundant number systems have also proven to be favorable alternatives to conventional systems. The signed digit number system, first classified in 1961, has demonstrated 33% savings in adders required for multiplication over standard binary notation [8]. For operand lengths that are halfway between powers of 2 (12, 24, 48, etc...), the binary stored-double-carry (BSDC) number system using the digit set [0, 3], has been proposed as high efficiency representation [27].

The disadvantage of using redundant number representation is need for re-conversion back into conventional notation. A proposed solution is to devise an arrangement of units capable of accepting and forwarding redundant values. In their suggestion for a redundant arithmetic, Ferguson and Ercegovic introduce a multiplier that accepts two redundant operands [28]. By taking advantage of the fact that multiplication may be executed using a redundant *multiplier*, and a converted *multiplicand*, their design achieves gains of 20% over conventional forms. However, this increase in speed comes at the expense of a 30% penalty in both area and power consumption.

2.4.3 Residue Number Systems

In a residue number system representation (RNS), a number x , is represented by the set of its residues with respect to moduli m :

$$x_i = x \bmod m_i = \langle x \rangle_{m_i}$$

Since a value is uniquely represented using smaller residues, the mathematical operations that are carried out will inevitably be fast and simple. Addition, multiplication and subtraction are the primary vantage points in residue number systems. This is due to the fact that these functions may be carried out by directly performing the given operation on the smaller residues. Freking and Parhi [29] present an application of RNS arithmetic in

public-key cryptography schemes, which inherently use modular exponentiation and multiplication. Similar papers propose the use of RNS arithmetic, targeted towards applications that fundamentally use modular mathematics such as cryptography and encryption.

The downfall to such number systems is twofold. First, the representation efficiency of residue number systems is greatly reduced over binary notation. A k -bit representation, yielding 2^k unique values, may only produce half as many in RNS format. Secondly, any gains in performance achieved by implementing addition, subtraction and multiplication may be eclipsed by the severe complexity of other mathematical and logical operations.

2.4.4 Logarithmic Number Systems

In pure mathematics, the multiplication and division of logarithms is quite easily performed by addition and subtraction respectively. Since the hardware implementation of addition/subtraction circuits is substantially more straightforward than that of multipliers and dividers, logarithmic number systems (LNS) may be employed to carry out these operations. As in other non-conventional representations, LNS is only directed towards the enhancement of certain operations, and presents severe restrictions on most other standard numeric tasks, such as addition and subtraction.

Recently the Double Base Number System (DBNS) has been proposed as yet another class of redundant number representation [30]. Simple arithmetic operations are made possible by simple geometric interpretation of the orthogonal bases. DBNS provides logarithmic like computation with reduced look-up table dimensions. This representation provides yet another alternative for application specific computation enhancement.

2.4.5 Floating Point Number System

In order to achieve the levels of precision demanded by modern systems, it becomes imperative to have a number representation capable of describing real numbers. The

limited range and/or precision of fixed-point values are alleviated through the use of the floating-point number system. Unlike fixed-point representations where the location of the decimal point is predefined, floating point values allow extremely large or small numbers to be represented with the same high degree of precision by defining a value using a dynamic range.

As defined in the IEEE standard for binary floating-point arithmetic [6], a floating-point value is defined as:

$$x = \pm f \times b^e$$

where x is the floating point value, f is the fraction of mantissa, b is the base (fixed at 2 for precision) and e is the exponent.

Floating point numbers have two distinct representations according to the standard, depending on word size. Figure 2.12 outlines the difference in the structure of the words of the 32-bit single precision and the 64-bit double precision formats. The sign (s), exponent (e) and fraction or mantissa (f) form the 32 and 64 bit precision formats. The mantissa is normalized to lie within the set $[1, 2)$, such that the MSB is a 1. In this manner the leading 1 is removed and understood; this is referred to as the “hidden one”, saving one bit in the representation. The signed integer exponent is biased accordingly, such that the value will always be a positive number; the exponent is biased for 127 and for single and 1023 for double precision formats. .

**Figure 2.12 IEEE floating point standard word widths for
(a) single precision (b) double precision**

The non-trivial implementations of floating point addition and multiplication are outlined in Figure 2.13 and Figure 2.14 on page 30 respectively [8]. Floating-point numbers are essentially composed of a biased non-negative integer exponent, and a fixed-point fractional representation of the mantissa. Consequently, the mathematical operations that are carried out on floating-point numbers will use fixed-point arithmetic units with additional control and rounding circuitry to accommodate for the dynamic range. For this reason, the focus of design and hardware implementation of mathematical operations has been on fixed point, integer units. Conversion to floating point is made possible through additional circuitry.

Figure 2.13 A floating-point Adder / Subtractor scheme [8]

Figure 2.14 A floating-point multiplication scheme [8]

2.5 Summary

In this chapter the fundamental concepts of digital multiplication have been presented. In addition, the basic framework for the various parallel digital multiplication algorithms have been discussed. It has been demonstrated that column compression techniques outperform array architectures. In general, the trade-off between performance and complexity is the driving force behind variations in multipliers. Designers have exploited the fundamental design characteristics of the individual multipliers for specific applications. For high-speed computation, the tree multiplier has been the design of choice, whereas low power, low area designs will tend to take advantage of the regular layout of the array type multipliers. As always, for the most basic algorithm, shift-add schemes will be drawn on.

A brief introduction to the applications of unconventional number systems in arithmetic systems has also been given. Although the use of such systems presents clear advantages in many unique circumstances, a complete system for generic applications has yet to be formed. Although there has been a proposal for a full DBNS processor for DSP applications [31], non-conventional number systems for the most part are targeted at systems which are capable of exploiting a particular aspect of a representation.

Furthermore, it has been established that although floating-point representations are different in nature to fixed point or integers, the basics of fixed point arithmetic form the framework for floating-point calculations. For the reasons presented, the remainder of the thesis will target integer implementations of arithmetic circuits in conventional binary format.

Chapter 3

Partial Product Reduction Techniques

The essential multiplication algorithms have been presented thus far. Such schemes form the basis for the multiplication schemes that are employed in modern devices. This chapter will focus on the modifications to the standard column compression multiplier that are made possible by the variations of partial product reduction techniques.

The predominant distinguishing factor amongst tree multiplier schemes lies in the manner in which the column wise compression of the partial products occurs. A desired aspect of system behavior, such as speed, area, layout or power, may be optimized by the proper selection of a particular approach. In the subsequent sections, methodologies that concentrate on each of the major design criterion will be presented.

3.1 Booth Recoding

Prior to delving into the understanding of partial product reduction schemes, the means by which the partial product array is obtained should be explored. The most primitive formation of the array is obtained by a bit-wise logical AND of the operands. This has been the method that has been presented in the discussion thus far.

It should be noted that recoding of the multiplier operands leads to significant reductions in the partial product array depth. The recoding formalism that has been used extensively in digital multiplier algorithms is known as Booth recoding [32]. Several variations to this recoding style have been presented [33][34], and their implementation and performance has been explored [20][35]. Some researchers believe that the recoding overhead required for the Booth algorithm counteracts the proposed gains [26][36][37]. However, this methodology has been widely applied in multipliers, and so its principles will be highlighted.

In the high radix multiplication algorithm for serial bit-at-a-time multipliers, the effective clock rate was increased by reducing the number of partial products using multiple *multiplier* bits at a time. However, the multiples of the multiplicand that were to be used in the scheme had to be calculated separately. Recall, in the case of the radix-4 multiplier, the bits of the *multiplier* were grouped into pairs, and the partial products were selected from the set $\{0, A, 2A, 3A\}$, where A is the *multiplicand*. In this case a carry propagate addition is required to generate the $3A$ multiple. Booth's Algorithm reduces the number of partial products without the necessity of any pre-addition to produce the partial products. If the multiples are selected from the set $\{-A, 0, A, 2A, 4A\}$, they are easily obtained by basic shifting and complementing. The methodology suggest the use of $4A - A$, as opposed to $3A$.

Figure 3.1 on page 34 [20] outlines a 16×16 multiply using the 2-bit version of the Modified Booth Algorithm (MBA) known as Booth-2, MBA-2, or Radix-4 Booth. The name is based on the fact that two bits are recoded, thus halving the number of partial products, or simply regarding the *multiplier* as a radix-4 operand. The multiplier is subdivided into overlapping 3-bit wide groups, where each group selects a partial product according to the given selection table. To overcome the selection of $3A$, depending on the group value, either $4A$ is pushed into the next most significant group or $-A$ is pushed into the next least significant group. Negation is achieved by a bit-wise inversion of the value (ones complement), and a single bit (labeled in the bits marked **S**) is added to the least significant bit of the partial product.

Figure 3.1 Dot diagram for a Booth-2 16-bit Multiplication [20]

Although the number of partial products has been reduced from 16 to 9 in the 16-bit example, it is not a true reflection of the overall savings. Further reduction in the number of partial products may be achieved by using larger group sizes for the determination of the partial product; such recoding schemes, known as Booth-3, Booth-4 or higher. It has been demonstrated that due to circuit complexity there is little advantage Booth recoding beyond Booth-3 [20].

As presented earlier, the partial product generation circuitry required for Booth recoding may overwhelm any achievable gains. The vantage point of Booth recoding is the reduction of the number of bits per column to be compressed, especially for multipliers having operands lengths greater than single precision. The SYNOPSIS Foundation Design Libraries, for example, offer implementations of fast multipliers using both Booth-Recoded (wall), and non-Booth-Recoded Wallace Tree (nbw) partial product reduction arrays, the latter being used for smaller designs, while the Booth-Recoded version used for high speed multiplier designs having larger operand widths [38].

The primary disadvantage of the recoding scheme is its call for 2's complement notation for representation of negative partial products that are generated in the recoding process. Floating-point representation uses signed magnitude notation, thus permitting all multiplication to be carried out using positive values, with a simple sign comparison conducted to determine the nature of the product.

With this in mind, Oklobdzija et al. [36][37] claim that a single row of high order compressors, or properly allocated full adders as described in [39], will achieve the same outcome at a higher level of performance. The use of compressors and high order counters achieves at worst the same level of reduction of partial product, in less time [37]. Section 3.3 and Section 3.4 will present the use of high order counters and 4:2 compressors in high-speed multiplication algorithms.

Although Booth recoding does reduce the number of partial product rows, its employment does not affect the general structure of the partial product tree. For this reason, discussions regarding partial product reduction networks for the remainder of this chapter will be limited to non-Booth schemes for clarity; most of the formalisms presented may be extended to have encoding schemes as part of the initial partial product generation strategy.

3.2 CSA Reduction Schemes

In chapter 2, the parallel tree multiplier architecture using carry save adder (CSA) arrays was introduced. This scheme has formed the fundamental framework for the design of high-speed parallel multipliers over the past four decades. In this section, the dissimilarities between the Wallace and Dadda techniques will be presented. In addition, there have been various deviations from the core principles of the CSA tree layouts, each developed to introduce regularity, higher speeds, or greater efficiency; these schemes will also be reviewed.

3.2.1 Wallace and Dadda Trees

The course of action applied in these two pioneering schemes for parallel multiplication is the three phase procedure outlined in Chapter 2, and reiterated below:

- For an $k \times k$ bit multiplication, a partial product matrix is initially formed. Composed of shifted versions of the *multiplicand*, the matrix is k -bits high and $(2k - 1)$ bits wide.
- The matrix is reduced by a set of full adders, also referred to as (3,2) counters, or carry save adder (Figure 3.2 on page 37). The matrix is further reduced according to the number of reduction stages required as per Table 2.1 on page 23, until only two rows of partial products remains.
- A final fast adder, such as a carry look-ahead adder (CLA) is used to sum the two remaining partial products.

The total number of stages required for each scheme is the same, whereas the layout and number of required adder cells is not. The Wallace tree multiplier [4] combines the partial product bits at the earliest opportunity, where rows are put together in groups of three and reduced using (3,2) counters. This design was ameliorated by Dadda in his proposal [5], where he suggested combining partial product bits as late as possible, while keeping the critical path length (number of levels) of the tree minimal. This methodology, as confirmed by Habibi and Wintz [29], utilizes the minimum number of counters, leading to simpler CSA tree structures, but requiring a wider final fast adder.

These two schemes were more recently analyzed by Bickerstaff et al. [21] further confirming earlier results. The dot diagram representation of the two schemes is borrowed from this paper, and provided in Figure 3.3 on page 37 for clarification of the above discussion. It is apparent that the Dadda tree does in fact utilize fewer adders during the reduction process, while the Wallace tree tends to insert adders at the earliest opportunity. Although slightly more irregular, the Dadda scheme presents a more efficient design. An analytic discussion on the minimization of adders and reduction stages will be presented in the upcoming sections.

Figure 3.2 The Carry Save Adder - (a) Dot diagram of individual full-adder cell. (b) Several full-adders used to form one level of a CSA

Figure 3.3 Variations of multiplier partial product reduction trees using CSAs [21] (a) Dadda implementation. (b) Wallace implementation

3.2.2 Minimum Reduction Stage Requirement

As outlined in Chapter 2, considering the reduction process as a column-wise compression of the partial products, where each stage reduces the column height by a factor of $2/3$, a simple relationship between maximum column height and number of required stages for compression may be obtained. This relation is outlined in Table 2.1 on page 23. The series is formed using an expansion involving rounded representations of the preceding values, making an exact algebraic interpretation difficult, if not impossible to formulate. There have been many attempts in providing a simple formula for the determination of the exact number of reduction stages required (h) for any arbitrary operand size (k). Danysh and Swartzlander use the following simplification in their analysis [18]:

$$h = 2 (\log_2 k - 1)$$

This may be rearranged to represent the maximum column height $n(h)$ for a given number of reduction stages (h) as:

$$n(h) = 2^{\left(\frac{h}{2} + 1\right)}$$

This relation holds true only for the most typical operand size, and fails in most other cases. By returning to the formula presented above for the iterative procedure used in calculating the values in Table 2.1, some interesting bounds may be obtained. Parhami [8] has established that by ignoring the floor operation, the theoretical upper and lower bounds of

$$n(h) = \left\lfloor \frac{3}{2} n(h-1) \right\rfloor$$

may be established as:

$$n(h) \leq n_H(h) = 2 \cdot \left(\frac{3}{2}\right)^h$$

$$n(h) > n_L(h) = 2 \cdot \left(\frac{3}{2}\right)^{h-1}$$

The original expansion, denoted $n(h)$, and the two mathematical boundaries are shown graphically in Figure 3.4.

Figure 3.4 Actual expansion values and theoretical bounds of minimum full adder requirements in multi-operand addition

By assuming that the actual value lies midway between the two boundaries, a novel algebraic representation may be developed using the average of the extremities:

$$\begin{aligned}
 n(h) &= \frac{n_L(h) + n_H(h)}{2} \\
 &= \frac{2 \cdot \left(\frac{3}{2}\right)^{h-1} + 2 \cdot \left(\frac{3}{2}\right)^h}{2} \\
 &= \left(\frac{3}{2}\right)^{h-1} + \left(\frac{3}{2}\right)^h \\
 &= \left(1 + \frac{3}{2}\right) \cdot \left(\frac{3}{2}\right)^{h-1} \\
 &= \frac{5}{2} \cdot \left(\frac{3}{2}\right)^{h-1} \\
 \therefore n(h) &= \frac{5}{3} \cdot \left(\frac{3}{2}\right)^h
 \end{aligned}$$

This new representation has been plotted, along with the boundary curves in Figure 3.4. It is apparent that this simplification may be used as an accurate model of the relationship between the number of stages of reduction required for a given operand size, since it almost completely overlaps the original expression derived using the expanded values. As a demonstration of the superiority of this new scheme, a comparison between it, and three other logarithmic simplifications are graphically presented in Figure 3.5. The new averaging approximation in addition to the other methods, labeled as “Log (Danysh)” used in [18] , and “Log (Davio)” as outlined in [40], and “Log (Song)” as described in [41] are defined below:

$$\begin{aligned}
 \text{New Averaging Approximation} \quad n(h) &= \frac{5}{3} \cdot \left(\frac{3}{2}\right)^h \\
 \text{Log (Danysh)} \quad n(h) &= 2^{\left(\frac{h}{2} + 1\right)} \\
 \text{Log (Davio)} \quad n(h) &= 1 + \left(\frac{3}{2}\right)^h \\
 \text{Log (Song)} \quad n(h) &= 2 \cdot \left(\frac{3}{2}\right)^h
 \end{aligned}$$

3.2.3 Minimum Full Adder Requirement

The previous section discussed the development of an algebraic approach to calculating the minimum number of CSA stages with respect to operand size. In this section, the focus will be placed on the calculation of the minimum number of adder cells required in a partial product reduction matrix. The foundation for the following analysis was set forth in [5], with a detailed elaboration presented in [25].

Considering a symmetric $k \times k$ multiplication, a partial product matrix of k rows by $(2k - 1)$ columns is formed. An ascending integer matrix representation is used to identify the columns, beginning with 0 and working to the right until $2k - 2$, as in Figure 3.6.

Figure 3.5 Algebraic representation of minimum full adder requirements

Figure 3.6 An 8x8 multiplication partial product matrix

We now introduce $p(j)$ to denote the number of “partial products” in the j^{th} column. From the symmetry in the partial product matrix outlined in Figure 3.6, the following generality may be made:

$$p(j) = p(2k - 2 - j) = j + 1 \quad \text{for } 0 \leq j \leq k - 1$$

To account for the carry bits that are produced throughout the reduction process, $e(j)$ is introduced. The “expanded column size” for each column is the summation of the partial product bits in the j^{th} column, and all carry bits that flow in from the $(j - 1)^{th}$ column. Let $q(j)$ be the number of adders needed to reduce $e(j)$ to one. A full adder will absorb three bits from a column and generate two bits, one in the same column and one in the proceeding column. In effect, a full adder eliminates 2 bits from each column, disregarding the carry bits. Thus, the following relations must hold true:

$$\begin{aligned} q(j) &= \frac{e(j) - 1}{2} && \text{for odd columns} \\ q(j) &= \frac{e(j)}{2} && \text{for even columns} \\ (1) \quad q(j) &= \left\lfloor \frac{e(j)}{2} \right\rfloor && \text{in general} \end{aligned}$$

Each adder in column $j - 1$, whether a full adder or a half adder, will contribute one carry bit to column j . Using this, the general formula for the expanded column size may be stated as:

$$(2) \quad e(j) = p(j) + q(j - 1)$$

Combining equations (1) and (2), and expanding the recursive relation using the starting condition that the first column does not need an adder ($q(0) = 0$), we obtain:

$$q(j) = q(2k - 1 - j) = j \quad \text{for } 1 \leq j \leq k - 1$$

Thus, the number of required adders is:

$$q_T = 2 \cdot \sum_{j=1}^{k-1} q(j) = 2 \cdot \sum_{j=1}^{k-1} j = n(n - 1)$$

Where the expression was solved using the property:

$$\sum_{j=1}^N j = \frac{1}{2} N(N+1)$$

As mentioned previously, the statements have assumed the reduction to one single row; however, this is not necessary for partial product reduction. Digital multipliers use a final fast adder, due to their superior carry management, for the summation of the final two rows of the reduced partial product matrix. To account for this, the redundant adders will be removed ($2n - 2$ adders to be exact) from the adder count resulting in a total count of:

$$q_T = (n - 2)(n - 1)$$

($n - 1$) of which are half adders.

3.2.4 Variations of CSA Trees

The inevitable evolution of computer arithmetic has brought with it the development of the basic concepts of CSA tree multipliers. Over time, many researchers have contributed to the enhancement of the CSA partial product reduction scheme, targeting area, delay, regularity, and power. Naturally, the advancement of circuit technology created new and more efficient implementations of the CSA on a circuit level, however the focus here will be on algorithmic and architectural modifications.

The most straightforward modification of the Dadda/Wallace tree multiplier, is the incorporation of Booth's algorithm for an overall cutback of the total number of partial products fed into the reduction network. The Booth Encoded Wallace tree multiplier has been the pinnacle high-speed multiplication algorithm used in many digital design suites, including Synopsys. Millar et al. [34] have critically analyzed this technique in terms of area overhead and latency. Although crude estimations using gate count and gate delay were used as the test metrics, their results have shown that potential exists for a 16% decrease in delay, at a cost of 28% increased complexity, for a 32x32 multiplier.

A different approach to the standard tree has been explored by Hekstra and Nouta [10]. In their attempt to increase the regularity of the parallel tree multiplier, they propose an "array of arrays" based structure. By taking advantage of the simplistic layout of array multipliers, they propose the use of sub arrays that are used to sum portions of partial product matrix. The main array then sums the partial product sums formed by the sub arrays to form the product (see Figure 3.7). The choice of the number of stages and sub array height has a significant impact on the performance of the multiplier. It has been demonstrated that although this scheme promotes circuit regularity, it can at most achieve an optimal multiplication time of $O(\sqrt{k})$.

Figure 3.7 Array of arrays layout as outlined in [10]

The "Windsor Multiplier" [25], demonstrates that gains in terms of area and interconnect length may be achieved through the reallocation of adder blocks within a Dadda tree. In this paper, Wang et al. verify that for a k -bit symmetric multiplication (both operands being k bits), the total number of full adders will be $N = (k - 1)(k - 2)$. Furthermore, by referring to Section 2.3.3 on page 22, the number of stages of reduction required for a k -bit multiplication will equal h , so long as:

$$n(h - 1) < k \leq n(h)$$

The distribution of the N adders to the h stages is the focus of attention of this paper. It has been confirmed in [25] that many arrangements may exist which yield the same overall result using the same number of adders and stages. A procedure for allocating adders to a CSA reduction tree has been presented that enables maximum area efficiency in terms of cell distribution across a given region of silicon. In addition, this technique may be modified to pursue minimum inter-stage interconnect lengths. The significance of this work is that it is one of pioneering papers in terms of addressing interconnect configuration as a vital element in high-speed design.

The final amendment to the standard scheme that will be covered in this section deals with the physical interconnection of the adder cells once they have been placed. The notion of the exploitation of fast input and fast outputs in arithmetic sub-cells has been introduced by Oklobdzija et al. [39]. An algorithmic approach to speed optimization in partial product reduction trees has been proposed that draws on the inherent timing characteristics of adder cells.

Figure 3.8 depicts a typical logic level representation of a full adder. With the aid of this diagram, it becomes apparent that the delay from the different inputs to the outputs is not constant. By connecting the fast outputs to the fast inputs a minimal path length will be formed which may be applied to the critical path. The method presented in the paper suggests that through careful modeling of the relationship between input and output delays, followed by global optimization of the interconnections consistent delays may be achieved through each path.

Figure 3.8 Logic diagram of a Full-Adder

The proposed Three Dimensional Minimization (TDM) methodology, though a different approach than Dadda's reduction schemes, maintains the same number of cells and levels as the previous method, while concurrently making use of fast data paths. Though advantageous over a standard Dadda arrangement, use of the TDM leads to more complex interconnections and layout. The analysis in the two papers discussing this approach [39][42] neglect the adverse effects of complicated interconnect and layout plans.

3.3 High Order Counters And Compressors

3.3.1 Counters

Although the work of Dadda has been directly linked to CSA reduction schemes, his manuscript [5] had a much broader focus, encompassing the applications of parallel counters for partial product reduction. The full adder, or carry save adder, is a particular subset of the class of parallel counters. A parallel (N, M) counter is defined as a combinational network having M outputs and $N \leq 2^M$ inputs of equal weight. The M outputs are based on the number of logic 'ones' that appear at the N inputs. Any size counter may be constructed, so long as the M output bits are sufficient to represent all possible sums of the N inputs. Examples of typical counters include (3,2), (7,3), (15,4), and a few examples are depicted in Figure 3.9.

Stenzel et al. [24] expanded the notion of the parallel counter by introducing counters which can receive several successively weighted input columns. Counters of this type are denoted as $(C_{K-1}, C_{K-2}, \dots, C_0, d)$, where K is the number of input columns, C_i is the number of input bits in the column of weight 2^i , and d is the number of bits in the output word. Several examples of such multi-column input counters are provided in Figure 3.10 on page 47.

Counters accepting inputs from adjacent columns in many cases introduce the requirement for carry propagation, thus making them inferior to standard full-adder arrays. By deviating from the normal layout and construction of binary counters, several new

alternatives may be formed. Highly efficient counters employing parallelism in handling large number of input bits have been proposed for high-speed arithmetic circuits [43][44]. These counters employ speed-up techniques used in fast adders, such as carry-select or carry skip. As a means of overcoming the carry propagation dilemma, the use of redundant number representations for counters has also been presented. The gains by these proposals may be questioned, since redundant format will need conversion circuitry to binary format, and sequential counters will necessitate complex clock trees throughout the design.

Figure 3.9 Examples of Dot Representations of Parallel Counters

Figure 3.10 Examples of Dot Representations of Multi-column Counters

Using multi-column and high order counters, a larger portion of the partial product matrix may be reduced in one cell. This comes at the expense of larger input and node capacitances, and longer pull-down paths. This is due to the number of input transistors increasing quadratically with the number of inputs [41]. Furthermore, the inherent irregularity of the manner in which such counters reduce the matrix, forces the need for various counter structures to be used within the same reduction process.

High order counters for partial product reduction have been explored and implemented in numerous proposals [8,23,24,32,41,43-49]. The schemes demonstrating the most promise for general multiplier architectures having arbitrary operand sizes include low order counter classes based on full adders, and 4:2 compressors. An interesting variation, in terms of the fundamental circuitry involved, are high order counters based on the use of *threshold logic (TL)* [50-52]. The use of threshold logic, more specifically *capacitive threshold logic (CTL)*, has been suggested for arithmetic circuitry [53][54], and will be subsequently discussed in Section 3.6.

3.3.2 Compressors

Similar to counter structures, digital compressors are used to reduce a given set of inputs to a vector output. The primary distinction between counter and compressor circuits is that compressors do not necessarily follow the standard pattern of M outputs drawn from 2^M inputs. An $[N : M]$ compressor in essence is a variation of a counter that employs a separate path between compressor units in order to generate M final outputs using $N > 2^M$ input bits. Compressor configurations are generally formed using arrays of horizontally interconnected compressor units. In this manner, a horizontal carry signal may propagate laterally across the row of compressor units in order to account for the excess bits formed in the reduction process.

Higher order classes of compressors may also be used using variations of large counters with horizontal interconnections; however, these circuits suffer the same fate of high capacitance, large circuitry, and problematic matrix positioning as large counters. Song

and DeMichelli [41] have examined the implementation of higher order compressors. Labeled as the 9:2 family of compressors, these structures are formed using 4:2 compressor and (3,2) counters. An analysis of counters against compressors has been carried out by Mehta et al. [45]. In their research, the use of (7,3) counters against (7:3) compressors, amongst many others, has been evaluated. Their findings illustrated no major delay advantage in the use of large compressors over large counters, except for greater interconnect complexity introduced by the inter-cell wiring of the compressors.

The most widely used style of digital compressor, which displays several promising characteristics for multiplication applications, is the 4:2 compressor. This special class of compressors requires a section of discussion on their own.

3.4 4:2 Compressors

Since its inception by Weinberger in 1981 [55], the concept of the 4:2 compressor has soared in popularity in many digital multiplication and multi-operand addition schemes. The application of 4:2 compressors has also been the focus of several studies promoting its use over Booth recoding schemes [26][36][37]. This section provides an in depth look at the various configurations of 4:2 compressors, and a novel layout scheme for optimal placement.

3.4.1 Structure Of 4:2 Compressors

The 4:2 compressor transformed the standard frame of mind of counter based partial product reduction schemes by introducing the notion of horizontal data paths within stages of reduction. Though not technically a counter, since it is impossible to use 2 output bits to represent 4 binary input bits, the 4:2 compressor is based on a 5:3 counter structure. The *vertical*, or cross-stage, data path forwards the outputs on to the next stage, while the *horizontal*, or inter-stage, interconnections allow for the propagation of the generated carry bits. The inter-stage carry transmission is limited to only one stage, due to the offset nature of the internal structure of the circuit.

Figure 3.11 Cascaded full adders composing a basic 4:2 compressor

The most primitive representation of the 4:2 compressor is a pair of cascaded full-adders (Figure 3.11). This configuration does not reduce the overall delay of the column compression process, in fact it may increase the overall delay depending on the column heights [39]. Compressors of this form allow for increased regularity of design.

A step towards the reduction of the delay of the compressor may be achieved by applying the concept of fast input and outputs. Oklobdzija et al. [39], present a modified 4:2 compressor composed of properly stacked full adders with a minimized critical path delay. Figure 3.12 outlines this methodology. The optimal 4:2 compressor structure, in terms of gate delay, arises when the entire structure is regarded as one entity, as opposed to a composition of two full adders. This enables further optimization of the overall scheme, as depicted in Figure 3.13 [36]. In this design, the critical path of the compressor has been reduced from 4 XOR gates, as in the full adder arrangements, to 3.

The circuitry of the 4:2 compressor may be further enhanced if the actual design of the cell is taken down from a gate level description to transistor level. Although a valid estimate of delay, the gate level description of any arithmetic circuitry provides only an estimate of performance. Just as the compressor block was further optimized by decomposing the internal arrangement from two full adders to a series of gates, so too can the gate level design. By dismantling the gates down to the transistor level, further optimization may be carried out, reducing both transistor counts, and path delays. This topic of critical importance to modern arithmetic design will be further explored in chapter 4.

Figure 3.12 4:2 compressor layout making use of fast input/output paths

Figure 3.13 A minimized gate level representation of a 4:2 compressor

3.4.2 Proposed Optimized Compressor Layout

An arbitrary distribution of 4:2 compressors, though effective, may not be entirely efficient. Just as Dadda's scheme takes into account the minimization of the number of counters required per stage, a similar plan is required for 4:2 compressor arrays. Unlike standard counters, the layout of the 4:2 array presents some unique challenges. In a given row of compressors, the middle cells may all be regarded as devices that convert 4 input bits in a column to 2 output bits of differing weight. However the two compressors at the opposite ends of the row must be regarded as (5 , 3) counters. The first compressor takes

in 5 inputs from a given weight, while the last compressor generates 3 output bits, two of which are in the next higher weight.

The process for delegating compressors must follow the same principles that have been established by Dadda [5], and later used by Stenzel et al. in their analysis of variable length counters for partial product reduction [24]. It is necessary to first determine the number of stages required for a given partial product matrix size. Similar to the discussion presented earlier regarding the minimum number of full adders, we now examine the 4:2 compressor distribution. Consider a column within the matrix having height $n(h)$, where:

$$n(h) = \lfloor 2n(h+1) \rfloor$$

$$n(h-1) = \left\lceil \frac{n(h)}{2} \right\rceil$$

This expression is a straightforward extension of the fact that the 4:2 compressor effectively cuts the number of bits in a column by half. By expanding the relationship, we obtain the relationship depicted in Table 3.1 between column height and the number of required stages.

Table 3.1 Max column height per stage of a 4:2 column compression tree

h	0	1	2	3	4	5	6	7
$n(h)$	3	4	8	16	32	64	128	256

Unlike the (3,2) counter case, however, there exist an exact algebraic formula for the association of the column height with the number of required compressor stages, and it is:

$$h = \lceil \log_2 n(h) - 1 \rceil$$

where once again, $n(h)$ represents column height, and h represents the number of stages.

With the development of general relationship between column height and the number of compression stages, an optimization strategy can be formed. As with the previous

methods for counter distribution [5][24], the minimized allocation of 4:2 compressors will be one that reduces the column height sufficiently to meet the maximum column height for the next level.

A novel mathematical analysis has been carried on the minimum 4:2 compressor distribution, borrowing from the principles used in [25] in obtaining the theoretical bounds of full adders in a partial product matrix. Re-introducing the variables used in Section 3.2.3, we denote the number of “partial products” in the j^{th} column as $p(j)$, where due to the symmetry of the matrix,

$$p(j) = p(2k - 2 - j) = j + 1 \quad \text{for } 0 \leq j \leq k - 1$$

The “expanded column size”, $e(j)$, for each column is the summation of the partial product bits in the j^{th} column, and all carry bits that flow in from the $(j - 1)^{th}$ column. Finally, $q(j)$ is the number of compressors necessary to reduce $e(j)$ to two, and their relation may be expressed algebraically as:

$$(1) \quad q(j) = \left\lceil \frac{e(j) - 2}{4} \right\rceil$$

A 4:2 compressor takes in 5 bits from one column (4 partial products and one carry-in), and generates 3 bits, two carry bits to the next column and a sum bit in the same column. As a whole, a 4:2 compressor will eliminate 4 bits from column j , while adding two bits to column $j + 1$. With this in mind, $e(j)$ may be expressed as:

$$(2) \quad e(j) = p(j) + 2 \cdot q(j - 1)$$

Combining (1) and (2), and expanding the recursive equation, we obtain:

$$q(2k - j) = q(j) = \left\lfloor \frac{j}{2} \right\rfloor \quad \text{for } 2 \leq j \leq k - 1$$

$$q(j) = \left\lfloor \frac{k - 1}{2} \right\rfloor \quad \text{for } j = k$$

From these relations, we can calculate the total number of 4:2 compressors as:

$$N_C = 2 \cdot \sum_{j=2}^{k-1} q(j) + q(k)$$

For even operand sizes, the number of compressors is:

$$\begin{aligned} N_C &= 2 \cdot \left(2 \cdot \sum_{j=1}^{\frac{k-2}{2}} j \right) + \left(\frac{k-2}{2} \right) \\ &= 4 \cdot \frac{1}{2} \left(\frac{k-2}{2} \right) \left(\frac{k}{2} \right) + \left(\frac{k-2}{2} \right) \\ &= \left(\frac{k-2}{2} \right) (k+1) \end{aligned}$$

Whereas for odd operand sizes, the number of compressors becomes:

$$\begin{aligned} N_C &= 2 \cdot \left(2 \cdot \sum_{j=1}^{\frac{k-1}{2}} j \right) - \left(\frac{k-1}{2} \right) \\ &= 4 \cdot \frac{1}{2} \left(\frac{k-1}{2} \right) \left(\frac{k+1}{2} \right) - \left(\frac{k-1}{2} \right) \\ &= k \left(\frac{k-1}{2} \right) \end{aligned}$$

The mathematical analysis provides a means by which an estimation on the minimum number of 4:2 compressors required in a reduction matrix for an arbitrary operand size. A further optimized approach has been developed, enabling a more efficient distribution of the compressors in a reduction matrix. Prior to introducing the iterative procedure

developed for minimum 4:2 compressor placement, a set of preliminary definitions and explanations will be presented for clarity:

- a) The number of required stages, h , and the targeted maximum column height, $n(h)$, for the first reduction stage are solved using:

$$h = \lceil \log_2 n(h) - 1 \rceil$$

$$n(h) = 2^{h+1}$$

The maximum column height each subsequent stage is determined according to:

$$n(h-1) = \left\lceil \frac{n(h)}{2} \right\rceil$$

where $n(h)$ is the current maximum column height, and $n(h-1)$ is the maximum column height for the next stage. For any given maximum column height, the maximum column height for the next stage may be calculated according to:

$$n(h-1) = 2^{\lceil \log_2 n(h) - 1 \rceil}$$

- b) The previously defined ascending integer index will be used, assigning 0 to the least significant column and $2k-2$ to the most significant.

- c) A 4:2 compressor row will be defined as a succession of linked 4:2 compressor cells beginning with one half-adder, and ending with one full adder cell. The connection of the cells is outlined in Figure 3.14.

- d) A row of 4:2 compressors will be added for every 2-bits that a given column height must be reduced for the next stage. It should be pointed out that without the half-adder commencing the row, the first compressor in each row would in effect behave as a [5:1] compressor. This becomes apparent when one considers the notion that the compressor cell takes 5 bits from a column, including the *CARRY IN* which is of equivalent

weight, and so may be taken from the same column. Furthermore, the output for this compressor would consist of two bits to the next higher weight, a *CARRY* bit for the next stage, and a *CARRY OUT* signal that propagates horizontally. Only one bit will be passed onto the equivalent weighted column of the next stage, that being the *SUM*. For increased regularity and efficiency the half-adder replaces the initial compressor.

e) The carry out of the final compressor for each row must be accounted for in the allocation of compressors, since the *CARRY OUT* signal is not being absorbed within the same stage. Thus for the proceeding stage the column of the next weight will receive two carry bits as a result of the row of compressors. The row of compressors may need to be extended if the two bits exceed the maximum column height for the next stage. The final full adder ending a given row within a stage will account for the additional bit generated.

Figure 3.14 Definition of a 4:2 compressor row

With the necessary background information in place, the proposed iterative procedure for the compressor layout scheme for symmetric multiplication will be presented:

1) Determine the number of compressor rows (N_R) required for the given stage according to the equation:

$$N_R = \left\lceil \frac{n(h) - 2^{\lceil \log_2 n(h) - 1 \rceil}}{2} \right\rceil$$

where the expression, $2^{\lceil \log_2 n(h) - 1 \rceil}$, refers to the maximum column height for the next stage.

2) N_R rows of 4:2 compressors, as outlined in Figure 3.14 on page 56, are placed in the partial product reduction tree. The first row will begin at column:

$$j_{1F} = 2^{\lceil \log_2 n(h) - 1 \rceil}$$

and ending at column:

$$j_{1L} = 2k - 1 - 2^{\lceil \log_2 n(h) - 1 \rceil}$$

Every subsequent row will begin at column:

$$j_{iF} = 2^{\lceil \log_2 n(h) - 1 \rceil} + 2i$$

$$j_{iF} = j_{(i-1)F} + 2i$$

and end at column:

$$j_{iL} = (2k - 1 - 2^{\lceil \log_2 n(h) - 1 \rceil}) - 2i$$

$$j_{iL} = j_{(i-1)L} - 2i$$

where (i) represents the row number within each stage up to N_R rows.

3) Repeat steps (1) and (2) until only two rows remain within the partial product matrix, at which point a final fast adder will be used.

This process is better explained through the aid of a graphic example. Figure 3.15 and Figure 3.16 on page 59 provide two examples of the minimized 4:2 compressor cell distribution for column compression. The symmetric layout of the compressor rows, in addition to the general configuration of each row (beginning with a half-adder and ending with a full-adder) is now evident.

Using the minimized 4:2 compressor distribution presented, we can obtain the bound on the number of compressor cells required. By observing that the longest compressor row for $k \times k$ multiplication begins at column $j_F = 2$, and ends at column $j_L = 2k - 3$, and so is composed of $2k - 4$ compressor cells. Through a simple observation, which

may be proved algebraically by examining the column height equations, the number of columns (and therefore compressor cells) in each subsequent row will be $2k - 4 - 4i$, where i represents the compressor row number. The total number of rows (N_{RT}) required is:

$$N_{RT} = \left\lceil \frac{k-2}{2} \right\rceil$$

The total number of compressor cells (N_C) will be the summation of the cells within each row by the number of rows required. This expression may be represented algebraically as:

$$N_C = \sum_{i=0}^{\left\lceil \frac{k-2}{2} \right\rceil - 1} 2(k-2-2i)$$

Of this number total, the number of cells which are half or full adders will be:

$$N_A = 2 \cdot \left\lceil \frac{k-2}{2} \right\rceil$$

Figure 3.15 Compressor layout for a 16x16 multiplication

Figure 3.16 Compressor layout for a 24x24 multiplication

Thus the number of 4:2 compressors alone (N_{C^*}) will be reduced to:

$$N_{C^*} = N_C - N_A$$

$$N_{C^*} = \sum_{i=0}^{\left\lceil \frac{k-2}{2} \right\rceil - 1} 2(k-2-2i) - \left(2 \cdot \left\lceil \frac{k-2}{2} \right\rceil \right)$$

This scheme presents an efficient, and very regular, distribution of compressor cells. Through the use of half and full adders, for beginning and terminating each row respectively, redundant reduction is all but eliminated, and the cells, for the most part,

have complete occupancy at their inputs. This new scheme reduces the number of required 4:2 compressors as determined by the theoretical lower bounds calculated previously. There is an average reduction of 2.724% in the total number of cells, and 8.171% average decrease in 4:2 reduction cells if this new scheme is employed as opposed to a standard minimum distribution of 4:2 compressors. Appendix A provides the calculations for the given values.

The previous analysis grants the opportunity of forecasting the inherent characteristics of partial product reduction schemes involving (3,2) and 4:2 compressors. The most significant algebraic calculations include cell count, maximum gate delay and interconnect count. The particular cells are naturally technology and layout dependent, however the number of cells required under each scheme can offer implicit estimations of power and area.

A complete summary of the comparison between the proposed optimized 4:2 distribution and the minimized (3,2) Dadda tree is given in Appendix A. Some of the findings are plotted in Figure 3.17, Figure 3.18 and Figure 3.19. The calculation for the total cell count has been provided in this chapter for both full adder and compressor reduction schemes. As expected the number of cells in the 4:2 scheme is significantly less than that of the full adder reduction scheme, however the individual cells are in fact much larger.

The presented methodology for 4:2 compressor distribution, is by no means the only strategy that may be employed in partial product reduction arrays. It merely provides an algebraic and systematic method for the development of the reduction array having the least number of cells. Other distribution strategies involve the use of the maximum number of compressors as early as possible in order to minimize the final fast adder length [56][57][58]. The strategy employed in these 54-bit multiplier designs is similar to a Wallace tree layout using full adders, whereas the optimized distribution scheme is more closely linked to an optimized Dadda tree layout.

Figure 3.17 Total Number of cells in a partial product reduction tree
(NFA = Number of (3,2) NCT = Number of total cells for 4:2)

Figure 3.18 Total Number of interconnects in a partial product reduction tree
(IFA = Interconnects for (3,2) IC = Interconnects for 4:2)

The cell count, does lead to the calculation of the total number of interconnects within the reduction tree. The calculation for the number of wires is taken as a function of the number of cells, and their outputs. For the (3,2) cells which have 2 outputs, the derivation of the interconnect count, I_{FA} , is simply the sum of the outputs of the cells and the square of the operand size to account for the original partial product matrix formation:

$$I_{FA} = 2 \cdot N_{FA} + k^2$$

The 4:2 compressor interconnect length is calculated in a similar fashion, using a count of 3 for each compressor and 2 for each adder:

$$I_C = 3 \cdot N_{C \text{ no adder}} + 2 \cdot N_{C \text{ adders}} + k^2$$

The interconnect calculations show the advantage of the 4:2 compressor scheme, however there is a hidden benefit in this scheme. Approximately one third of the interconnects in the 4:2 scheme are the local, horizontal intra-stage carry paths. Thus, the advantages are twofold, *fewer* and *shorter* interconnects for the compressor reduction process. The delay for the two schemes is simply a product of the number of reduction stages and the approximate delay per stage (2 XOR for the (3,2) and 3 XOR for the 4:2).

Figure 3.19 Total Delay (DFA = (3,2) scheme DC = [4:2] scheme)

3.5 Low Power

As discussed in chapter 1, low power design techniques have emerged as one of the dominant fields of VLSI research. The expansive applications of mobile electronics, more specifically portable processing units, have created new challenges for digital circuit designers in their attempt to minimize power dissipation. Arithmetic circuitry is by no means immune to these new trends. In fact, low power design methodologies factor significantly in new ALU architectures.

The current dichotomy in digital system design is the need for devices that are both high performance and low power. As with any other design process, this issue is one of compromise, where in this situation it is the trade-off between speed, power and area. In terms of power, there are four predominant concerns related to power dissipation in VLSI designs:

- Portable devices, relying on batteries as an energy source must make use of low power techniques in order to prolong battery life and operating time.
- The energy consumed by devices is for the most part due to heat dissipation. Current techniques for dispersing the heat away from the device packaging will not be able to cope with the increased densities. It has been predicted that in 15 years the amount of heat dissipated will exceed packaging limitations by 25 times [9].
- Overheating as a result of high current densities leads to effects such as electromigration, and other forms of breakdown, resulting in shortened life expectancies.
- High current densities and voltages induce greater coupling effects, leading to an increased susceptibility to erroneous computation and data transfer.

For true 'power conscious' design methodologies, all sources of dissipation, and all opportunities for minimization must be understood and taken into account at every level of the design process [59]. Low power circuitry, including circuit implementations and analysis of low power adders and compressors, will be discussed in chapter 4. This discussion will begin with an overview of sources of power dissipation in modern circuitry, followed by techniques used to alleviate some of the power losses.

3.5.1 Leakage And Short Circuit Power Dissipation

In a standard CMOS process, there are three sources of power dissipation, namely dynamic, short circuit and leakage power [59][60]. Leakage power dissipation refers to any source of power loss that may occur under steady state conditions as a result of small leakage currents in the devices. This type of power consumption is a direct consequence of the chosen logic style, and topology.

Short circuit power dissipation is a form of dynamic power consumption that occurs due to a momentary dc path established between the supply and ground rails during circuit switching. Standard CMOS logic, is prone to such power losses, especially in the case of a digital inverter, where only two transistors exist between the power and ground rails. An accurate estimation of short circuit power dissipation is provided in [61]. The combined effects of short circuit and leakage power may be limited to only 15% of the total power consumption with the remaining caused by dynamic power dissipation [60].

3.5.2 Dynamic Power Dissipation

Dynamic power consumption is the primary source of power dissipation in modern CMOS digital design. Dynamic power loss arises as a result of the charging and discharging of capacitances (some parasitic) due to signal switching, or unintentional transitions as a result of timing effects. The equation for dynamic power dissipation is defined in [7] as:

$$P_{Dynamic} = \sum_i (C_{i \text{ load}} \cdot V_{i \text{ swing}} \cdot \alpha_i) \cdot f_{clk} \cdot V_{DD} + \sum_i (K_i \cdot \alpha_i)(V_{DD} - 2V_T)^3 \cdot f_{clk}$$

where:

$$K_i = \frac{\beta_t}{12}$$

and the summation occurs over all nodes (i) of the circuit.

The individual components in the expression are defined as:

$C_{i \text{ load}}$	load capacitance at node i
$V_{i \text{ swing}}$	voltage swing
α_i	switching activity factor
f_{clk}	system clock frequency
V_{DD}	power supply voltage
V_T	transistor threshold voltage
β	transistor gain factor

The key algebraic representation for power in algorithm or architecture level description may be simplified to:

$$P_{\text{Dynamic}} = \alpha C_L V_{DD}^2 f$$

In this representation, C_L is the physical load capacitance, V_{DD} is the supply voltage, f is the operating frequency and α is the switching activity. This expression is the dynamic power dissipation for one node; naturally the dynamic power of the overall system would be the summation of every node in the circuit. Assuming ideal input conditions are utilized (zero fall and rise times during transitions), then the total power consumption is independent of the transistor characteristics, and the generalized formula may be used [62].

The switching activity is the average rate of switching, or number of edge transitions (0 to 1, or 1 to 0) per clock cycle. The switching activity factor is introduced since the generalized power formula assumes a transition from 0 to V_{DD} occurs during every clock cycle, which is not necessarily the case in real circuits. The power dissipated in a transition is in fact the dissipation of heat through a PFET as the output node charges to V_{DD} , and the dissipation of heat through the NFET as the output node discharges the stored charge. It is said that one half of the power drawn from the supply is dissipated by

the PFET [62], and only half is used to store charge on the output node. No energy is drawn from the supply during the discharge phase.

By examining the given expression, it is clear that the reduction in any of the given parameters will lead to a reduction in power. The operating frequency is one means of reducing the power dissipation, however, it is in direct conflict with performance requirements, and so is usually considered as a last resort. Having a quadratic relationship with power, the operating voltage is the most critical factor in reducing power. There are a number of methods for reducing voltages in circuitry, such as dynamic voltage scaling [63], multiple on chip voltage levels, and simply reducing the supply voltage due to process advancements. Although theoretically effective, the first two techniques are still in their infancy, and the latter technique is process and technology dependent, and is not influenced by the design.

Reduction in parasitic capacitances would result in the charging and discharging of smaller capacitive nodes. This has the benefit of not only decreasing the power dissipation, but the switching time also. The push towards deep sub-micron technology has lead to smaller device dimensions, requiring shorter interconnects; both critical in limiting the parasitic capacitance. The only issue that comes into play on an architectural level is the effects of parasitic and coupling capacitance that is aggravated by technology scaling. This is a topic of special interest in this thesis, and will be further explored in chapter 5.

Finally, one of the most significant parameters of dynamic power dissipation is the switching activity, or the average rate of switching on a circuit's nodes. Limiting the switching activity, regardless of technology and implementation details, may drastically reduce dynamic power consumption in arithmetic circuitry. By analyzing the switching characteristics of arithmetic algorithms, optimal circuit configurations may be developed for low power applications.

3.5.3 Dynamic Power Management

As a whole, the most obvious means of reducing power is by shutting down a circuit completely. *dynamic power management* refers to the selective shut down or slow-down of a system's components that may be idle or under utilized [64]. Computational circuits are on the whole event-driven, for this reason there will be periods of time that the system will be inoperative. By removing power from these dormant regions, dynamic power dissipation within these regions will be eliminated.

One of the most efficient means of reducing power dissipation in digital circuits is clock gating. As described in [65 - 68], clock gating is used to cut-off the clock signal from portions of the clock tree in synchronous systems. In this manner, the affected sections of the system will be forced into an idle state, preventing wasteful switching of the internal nodes.

For purely combinational systems, such as most partial product reduction trees, a technique known as *guarded evaluation* may be employed to shut down circuits during periods of inactivity. In this approach the entry of inputs to the main circuit, or portions of the system, is obstructed. Without a change in inputs, the circuit will remain at a halt, drastically reducing power dissipation.

Precomputation networks, fall in line with the concept of guarded evaluation, since they are used to block signals from entering and manipulating all or part of a system. The precomputation network is used to perform a preliminary calculation to determine the nature and size of the input vector entering the main circuitry. By taking advantage of a simple circuit up front, considerable savings in power may be realized in the overall system. For example, such a circuit may be put in place to determine the correlation between consecutive inputs to a multiplier. Based on its findings, a multiplication process may be carried out, else the previous output may simply be reused, and the new inputs discarded.

Algorithm inherent activity is the switching activity that has to occur in any realization of the design, independent of the implementation style chosen [59]. This particular topic is of great interest in architectural design, since it is technology and process independent. A designer has the ability of modifying an algorithm in order to prevent unnecessary transitions in logic states through a system.

The work of Muhammad et al. [69] provides an excellent example of limiting algorithm inherent activity with specific implications on digital multipliers. In this study, various configurations of array multipliers were evaluated against 4:2 tree multipliers for DSP applications. The research examined the effects of signal strength of the DSP input signals, and consequently the location of zeros in the operands of the multipliers, on switching activity. This problem may be more clearly defined by way of an example.

Assuming an array multiplication structure is used having a **multiplier** A , and **multiplend** B . Any input having $a_i = 0$ will pass a row of zeros, and those where $a_i = 1$ will pass on the bits of B . In Figure 3.20, the authors depict a scenario where bits a_3 and a_7 of the multiplier A are interchanged in the array structure, to demonstrate the independence of the row position to the overall structure. Such structures are possible since array architectures are concerned with adding partial products from cells occupying the same column; the order in which the partial products appear is insignificant.

In the case where the least significant bits (LSB) of A are 1's and the most significant bits (MSB) are 0's, then it is clear that the output will have to propagate down through the remainder of the bits until the final bit of A is applied, generating undue switching activity. If for the same example, a most-significant-bit first strategy is employed, the switching activity will be limited to the final portion of the array structure, and the top portion will remain idle due to the rows of zeros (Figure 3.21). Furthermore, the authors present hybrid array structures, where based on a priori signal probabilities and statistics, a certain partial product order is to be selected that minimizes the switching activity.

Figure 3.20 Interchanging rows within an Array Multiplier structure

Figure 3.21 Hybrid MSB first Array Multiplier structure

In the domain of general application circuitry, the research in [69] found that due to the balanced nature of tree multipliers, strong correlation in signals reduces switching activity automatically. These results are promising for general DSP architectures employing tree multipliers, since data signals are for the most part correlated, and rapid changes are seldom processed. The tree multiplication scheme lends itself to general applications, without the need for customized inner structures to meet predetermined statistical values.

3.6 Threshold Logic

Threshold logic offers a unique alternative to the traditional Boolean methodology of switching logic, restricted to only two possible states within a system, namely *on* (1) or *off* (0). Introduced in the early 1960's, threshold logic gates, otherwise known as majority gates, can compute any linearly separable Boolean function, where the result is based on the weighted sum of its inputs relative to a specified threshold value [70]. Threshold logic is closely related to artificial neural networks, in that each gate is formed by a large fan-in of input signals having an associated weight, and the gate (or node) must determine the output state based on the total input voltage level.

A primitive example of a threshold logic gate is depicted in Figure 3.22 on page 70. In this example the four inputs each have an associated weight of 1, and the gate has a threshold value of 4. Thus the output is active if the combined weight of the active inputs is greater than or equal to four. This only occurs in the case where all input signals are high, and so a four-input AND gate is formed.

Figure 3.22 Threshold Gate implementation of a 4-input AND gate [70]

Threshold logic gates form the most theoretically efficient counter structures, since a group of nodes, one for each output, are all that is required in the formation of a high order counter. This offers a huge advantage over the gate level implementation of the high order counter structures. The challenge with such arithmetic implementations lies in the creation of the gate structure itself; the inherently analog threshold gate presents a great hindrance to the digital designer.

The formation of threshold gates based on the use of capacitors to hold the charge and trigger the output of the gate has been one solution [70][71][72]. This CTL methodology suffers from noise sensitivity, and obvious technology scaling issues, since each CTL gate will have to be completely custom designed for each fabrication process, and for each technology generation of a fabrication process. Variations of the threshold logic gate involving implementations using MOSFET circuits have been proposed [73], along with several other derivatives focusing on noise immune [74] low power [75], and latch implementations [76].

Without delving too deeply into the precise silicon layout of such devices, it is intuitive to conclude that the implementation of arithmetic counters using such unconventional design methods will have to be based on custom and technology specific applications. The analog nature of the threshold logic gate limits its employment in general digital architectures, and although they have demonstrated superior performance results in many cases, their restricted physical use inhibits their advancement.

Chapter 4

Arithmetic Circuitry

Until now, the bulk of the discussion has been devoted to the analysis of algorithms and architectural techniques employed in computer arithmetic. Another facet of this broad topic that must also be considered is the low level implementation of arithmetic techniques. This chapter deals with the details and the techniques involved with the physical design and layout of the aforementioned high level algorithms.

The traditional philosophy towards computer arithmetic has been the development of algorithms, and system level architectures that may be employed for the enhancement of numerical operations. Algorithms would be developed by computer scientists or mathematicians, and would then be implemented by a hardware engineer. It becomes apparent, however, that this approach will lead to sub-optimal solutions. Modern researchers, for the most part, have an outstanding comprehension of both VLSI design methodologies, as well as numerical techniques. This is only natural, since the thrust of present-day research deals with the adaptation of existing techniques to suit the latest integrated circuit technologies.

This chapter will begin with an overview of logic styles for arithmetic circuitry, followed by an in depth analysis of several newly proposed logic styles. The use of novel "pass-logic" design techniques for arithmetic sub-cells, with focus on full adders and 4:2 compressor layouts, will be presented next. Finally, an overall analysis of the arithmetic circuitry presented throughout the chapter will be presented, along with some comments and considerations for the development and simulation of novel circuitry.

4.1 Logic Styles

Digital design encompasses a wide variety of logic implementations, which arise, for all intents and purposes, as a result of the transistor configurations composing the individual logic elements. The synthesis of the particular digital system (or sub-system) will dictate the nature of the particular logic family chosen. In a survey of logic styles, Zimmermann and Fichtner [77], outline the various characteristics of the final digital system that are dictated by the initial selection of the logic style chosen for implementation. The factors include:

- **Circuit delay:** a function of the number of inversion levels, the number of transistors in series, transistor sizes (i.e., channel widths), and intra- and inter-cell wiring capacitances.
- **Circuit size:** depends on the number of transistors and their sizes and on the wiring complexity.
- **Power dissipation:** determined by the switching activity and the node capacitances (made up of gate, diffusion, and wire capacitances), the latter of which in turn is a function of the same parameters that also control circuit size.
- **Wiring complexity:** the number of connections and their lengths in addition to the choice of single-rail or dual-rail logic
- **Generality:** ease-of-use of logic gates in standard cell design techniques and logic synthesis
- **Robustness:** determined by the resilience to voltage and transistor scaling as well as varying process and working conditions
- **Compatibility:** ability to seamlessly integrate with the surrounding circuitries

All of these characteristics may vary considerably from one logic style to another and thus make the proper choice of logic style crucial for circuit performance. Several logic styles, and logic families will be presented in each section, along with their potential vantage points and applications in arithmetic circuitry.

4.1.1 Static CMOS

Static CMOS logic, otherwise known as standard CMOS logic, is the logic style of choice for most implementations, and is most often used in the development of standard cell libraries for automated digital synthesis. The principle behind a static logic cell is that it exhibits a well-defined output once the inputs are stabilized and the switching transients have decayed away. The cell is composed of complementary NFET and PFET networks, where the input voltages control the conductance of the networks. The switching network is designed such that only one network is a closed switch for any input combination, thus determining whether V_{DD} or GND is connected to the output. Figure 4.1 on page 75 outlines a typical static CMOS cell configuration.

The reason behind the use of two separate networks is due to the physical nature of the MOSFET structures. The notion of threshold voltage loss inhibits the arbitrary transmission of signals through any MOS device. An NFET is designed such that a strong logic '0' is passed, while a logic '1' is passed with a threshold voltage loss according to:

$$V_{out} = V_{DD} - V_{TN}$$

Conversely, a PFET is designed to pass a strong '1', and a logic '0' with a threshold voltage gain according to:

$$V_{out} = |V_{TP}|$$

Figure 4.2 on page 75 provides graphs of this phenomenon in the two types of devices.

As mentioned previously, the individual networks of a logic cell are composed of an interacting group of transistors. The layout of the transistors ensures the proper functioning of the circuit, whereas the sizing of the transistors governs the DC switching voltages and transient switching times. The charge/discharge times of the circuit (T_{LH} ,

T_{HL}) are determined by the transistor aspect ratios. The aspect ratio, the ratio between channel width and length, is very important in the device geometry as it determines the current flow, and is the easiest parameter to control since it is defined by the device layout.

Figure 4.1 Static CMOS logic cell depicting the NFET and PFET networks

Figure 4.2 (a) NFET threshold voltage loss (b) PFET threshold voltage gain

The advantage of such logic families is in the simplicity of developing a circuit that will perform a given function, however complex, while providing robust performance measures. Static logic, for the most part, demonstrates excellent noise immunity, and is less susceptible to process variation since the sizing of individual transistors do not vastly alter the circuits functionality. One disadvantage of this type of logic is the use of a large number of PFET devices, being both slow and large in comparison to NFETs. Furthermore, the longest length chain within each network will determine the worst-case scenario for charge/discharge delay, forcing more complex systems to carry out potentially sluggish execution.

4.1.2 Transmission Gate Logic

The CMOS transmission gate (TG) is designed to act as a very efficient voltage-controlled switch, and was one of the fundamental building blocks in SSI and MSI technologies. It is formed by a parallel combination of one NFET and one PFET device, as depicted in Figure 4.3, set-up in such a manner as to allow a full-voltage swing output based on the control signal. The use transmission gates to form logic cells simplifies the design of many involved circuits, by allowing signals to determine the conduction path of other signals. The formation of multiplexor cells using TG logic is one straightforward application of this type of logic.

Figure 4.3 Transmission Gates (a) Transistor level (b) Logic symbol

The downfall of TG logic lies in its requirement of a control signal and its complement, thus increasing interconnect and signal requirements. In addition, the output node does not

receive voltage support since there is no pure path to either the supply voltage or to ground. For this reason the input signal must be able to drive the output capacitance, leading to potential difficulties in high-fanout applications.

4.1.3 Dynamic Logic Families

Standard static CMOS logic maintains a valid output voltage, so long as the inputs are well defined and continuous. Dynamic CMOS logic on the other hand makes use of capacitive nodes to store electrical charge, and so are capable of sustaining a valid output only for a short period of time. The advantage of such logic families is in their ability to quickly transfer charge, and in turn have a tremendous performance advantage over static CMOS, and are common in high speed applications. Dynamic circuits differ from static circuits in that instead of fighting the constant limits due to parasitic RC elements, capacitances are used as integral components of the circuits.

Though there are several distinct logic families that fall under the dynamic CMOS classification, there is a common underlying principle behind their operation. In general, charge is supplied by the supply voltage to a few select capacitive nodes during pre-specified clock times. The stored charge is then used to control the movement of other charges. A basic capacitive node is formed when a transistor is in the cutoff region, the isolated node at its gate may be modeled as a *storage capacitor* (C_S). Figure 4.4 [78] provides a circuit diagram of this simple model.

Figure 4.4 Capacitive Nodes (a) Basic circuit (b) Storage Capacitor model

The name 'dynamic logic' is in direct reference to the operating characteristics of such circuits. Unlike static designs that maintain their output values indefinitely, the outputs of dynamic circuits are constantly changing, and have only a short window in time within which they are valid. By using capacitive nodes to hold a charge, the circuit is prone to leakage current effects given by:

$$I_{leak} = -\frac{dQ_S}{dt} = -C_S \frac{dV_S}{dt}$$

Maintaining I_{leak} and C_S constant and integrating we obtain:

$$V_S(t) \cong V_{max} - \frac{I_{leak}}{C_S} t$$

In the case of an NFET driving the capacitive node, the transistor will slowly remove the charge from the node. Figure 4.5 on page 79 graphically illustrates the voltage drop at a capacitive node due to leakage effects. From the previous expression we can define the maximum logic 1 *hold time* as:

$$t_H \cong \frac{C_S}{I_{leak}} (V_{max} - V_1)$$

When a PFET drives the capacitive node (Figure 4.6), it in actuality adds charge to the node, thus forcing a maximum logic 0 hold time:

$$t_H \cong \frac{C_S}{I_{leak}} (V_0 - V_{min})$$

Dynamic logic design must be regarded as a system level design methodology, since it is almost impossible, and utterly imprudent to incorporate dynamic and static cells within a digital module. In fact, this notion may be extended further as to say that the combination of differing dynamic styles within a module should be avoided. The nature of the control and distribution of the signals through charged nodes is for the most part incompatible between the logic styles. To further exemplify this, a few of the prominent dynamic logic families will be presented.

Figure 4.5 Voltage drop due to leakage effects in an NFET

Figure 4.6 Voltage rise due to leakage effects in a PFET

The most basic type of dynamic logic is known as precharge-evaluate logic. This is a two state logic style where a clock signal controls a pair of complementary FETs managing the operation of the logic gate (Figure 4.7 on page 80). The two stages of operation are known as the precharge and evaluate stages. During precharge, the output node is charged via the precharge PFET, this is known as "pre-conditioning" the node. while the evaluate NFET is cut off. During this phase of the clock, the output and all of the inputs are invalid. During the evaluate stage, the evaluation NFET conducts, while the precharge PFET is cutoff. The inputs to the logic array are now valid, and if the logic array produces a value of '0', there will be a conduction path for the output charge to ground, else the charge will be maintained at the output and a result of logic '1'. The charge on the output node may only be held for a limited duration before being corrupted by charge leakage, and so timing is critical.

Figure 4.7 Primitive representation of a Precharge-Evaluate Logic block [78]

A dynamic system may be formed through the simple cascading of the individual cells. This leads to the formation of a variation of the precharge-evaluate logic known as *DOMINO* logic (Figure 4.8). The cascading of NFET logic arrays for the standard precharge-evaluate logic poses a potential glitch problem, which is overcome in *DOMINO* logic through the inversion of the output signal between cascaded cells. The name *DOMINO* logic arises from the rippling effect of the charge passing through successive stages of the cascade, where it is necessary for one stage to discharge prior to the proper functioning of the next stage.

Figure 4.8 A *DOMINO* logic block [78]

DOMINO logic is of particular interest in high-speed arithmetic system design since it is employed in the design of the double-pumped arithmetic logic unit found in the Intel Pentium IV architecture [79]. Though this ALU does not feature a built in multiplier module, it does carry out the pertinent delay critical arithmetic operations required by the microprocessor.

One final class of dynamic logic that will be introduced in this section is single-phase logic. Up to this point, single-clock, dual phase circuits have been discussed, where one clock, ϕ , is used but both ϕ and $\bar{\phi}$ are used for timing. Single-phase logic uses one clock and one phase only, thus simplifying clock distribution and generation. The clock is applied to either a single NFET or PFET clock transistor as in Figure 4.9, where the NFET is active when $\phi = 1$, and the PFET is active when $\phi = 0$; thus, a complementary clock signal is not required.

Figure 4.9 Single-Phase Logic Circuit Types [78] (a) Single Phase Network setup using latches (b) Single Phase Logic gates cascaded together

In general, dynamic logic gates exhibit superior timing performance over static logic, at the expense of several critical issues. The use of a clock signal for synchronization leads to increased interconnect requirements, a more complex clock tree, increased switching activity resulting in high power dissipation. In addition, the use of isolated charge storage nodes raises the concern of charge leakage and charge sharing issues which must now be dealt with. For these reasons, it is the author's opinion that the use of dynamic logic techniques for digital multiplication schemes presents an unwarranted increase in architectural complexity and synchronization, while a combinational logic system implemented using static techniques will suffice for most applications.

4.1.4 Differential And Dual Rail Logic Families

The logic families covered in this section provide shortcuts to the development of switching arrays through the use of both the input signals, and their respective complements. Single rail logic makes use of a set of input signals to generate one or more valid outputs. Dual rail logic, on the other hand, requires an input pair for each signal, and in turn generates one or more output pairs. Differential logic refers to the use of the voltage difference between a signal and its complement for the generation of the output. The advantage in this arrangement is the doubling of the slew rate (the rate of change of the output), at the expense of an effective doubling of the interconnection requirement.

One of the more notable classes of dual rail logic families is known as Cascode Voltage Switch Logic (CVSL). The standard configuration of a CVSL gate, depicting the two major sections of the gate, is provided in Figure 4.10 (a). The cross-coupled PFET pair forms a simple latch for the outputs, while the NFET network, formed by two complementary switching blocks, dictates the logic function. A clock signal applied to the cross-coupled transistor gates, and the addition of an evaluate NFET as in Figure 4.10 (b), will transpose this circuit into a dynamic logic gate, unsurprisingly named *DYNAMIC CVSL*. There are several other variations of CVSL, such as *Sample-Set Differential Logic* (SSDL), *Enable-disable Cmos Differential Logic* (ECDL), *Differential Current Switch Logic* (DCSL); the elaborate details of such logic types is beyond the scope of this thesis.

Figure 4.10 Cascode Voltage Switch Logic (CVSL)
(a) Static CVSL (b) Dynamic CVSL

4.2 Pass Transistor Logic

A family of dual rail logic that has been regarded as having significant promise in the creation of arithmetic sub-cells has been pass transistor (PT) logic. Simultaneously developed by Hitachi, Complementary Pass-Transistor Logic (CPL) and Dual Pass-Transistor Logic (DPL) are CMOS logic families targeted at low power and high performance architectures [80]. Pass transistor logic is the most prominent of alternate logic families amongst arithmetic circuit designers. This section will provide a brief overview of this variety of CMOS logic, prior to delving into the details of transistor level arithmetic circuitry in Section 4.3.

A pass transistor is simply a MOSFET with the input signal fed to the source and the output taken from the drain, with a control signal connected to the gate governing the output. A pass network is an interconnection of a number of PT's to achieve a function. Figure 4.11 provides the basic Pass Transistor configurations using NMOS and PMOS transistors. In both cases, X is the control variable, and Y is the pass variable, and the notation is $X(Y)$ and is read as "X passing Y".

Figure 4.11 Basic Pass Transistor Logic Configurations

One form of pass transistor logic, referred to as *Complementary Pass-Transistor Logic* (CPL), is based on the use of multiplexers to construct logic functions. The exclusive use of NFETs in the data path simplifies circuit layout. Figure 4.12 (a) shows a single NFET used to yield the AND function. It should be evident that this minimal layout does not account for all possible input combinations; to overcome this dilemma, a second transistor is required (Figure 4.12 (b)).

Figure 4.12 AND gate implementations using pass logic

There are downfalls to using strictly NFET arrays for the logic. The transfer of a logic one through an NFET is both slow and exhibits threshold voltage losses. To overcome the non-full swing output, an inverter is used at the output nodes. This "restoring" inverter is used to buffer the output by accommodating for the threshold losses, as well as increasing the drive strength of the output signal. Finally, since CPL uses both the original input signal and its complement, the output of a CPL gate must also provide the complement of the signal for the proceeding CPL cell. Consequently the CPL AND gate that was initially presented must be augmented to resemble the circuit in Figure 4.13 on page 85.

generally recognized and so there has been considerable focus on the development of full adder circuits [77,80-97]. This section will outline several recently proposed architectures.

The logic families discussed in Section 4.1 serve as the basis for the analysis of the various formations of the full adder cell. Simply stated, for every logic family there exists one or more CMOS implementations of the full adder circuit, each offering a particular vantage point. The logic styles of primary significance include:

- Standard CMOS
- Transmission Gate Logic
- Pass Transistor Logic
- Dynamic Logic

According to the previous analysis of the logic families, we may neglect the dynamic logic families from any further analysis due to the clocking and interconnect overheads that would be required in the highly irregular topology of multiplier circuits. The only dynamic logic family that has been employed on a large scale on a commercially available microprocessor has been *DOMINO* logic [79]; in spite of this, it has been determined by Alioto and Palumbo [84] that this type of logic consumes up to 700% more power and 150% more area than an equivalent standard CMOS cell. A *DOMINO* logic full adder cell is provided in Figure 4.15 on page 87 for reference, however the analysis for viable alternatives for CMOS full adders will focus strictly on the first three methods.

Pass transistor logic has been generally accepted as a low power, high performance logic style, since it allows for reduced signal transitions [81], reduced input capacitance, and smaller transistors [82]. One of the more significant incentives for the use of pass logic in full adder designs is the ease to which XOR/XNOR gates may be constructed using the multiplexer nature of pass-logic [77]. Figure 4.14 on page 87 depicts a transistor level and logic level depiction of the full adder cell, and the role of the XOR cell within its composition. By examining the expansion of the logic equations for the *SUM* and *CARRY* outputs of a full adder, the significance of the XOR gate becomes quite apparent:

$$Sum = (X \oplus Y) \oplus C_{in}$$

$$Carry = (X \oplus Y) \cdot C_{in} + XY$$

$$= (X \oplus Y) \cdot C_{in} + (X \oplus Y) \cdot X$$

Figure 4.14 Gate level (a) and transistor level (b) implementations of 12 transistor full-adder cell proposed in [83]

Figure 4.15 A *DOMINO* logic full adder cell [84]

Non-monotonic gates, such as XOR and multiplexers, are regarded as the most difficult logic gates to layout in standard CMOS logic. Since such gates form the basis of full adders, and have inefficient standard CMOS layouts, pass logic techniques have emerged as the leading alternate logic style. One of the most cited papers on this topic was written by Wang et. al in the early 90's [85]. In their analysis, the authors develop a novel approach to the pass logic XOR gate, and at the same time provide an overview of the existing reduced transistor XOR gates.

Figure 4.16 Various XOR/XNOR configurations

(a) Transmission Gate (b) Transmission Gate with driving outputs
(c) Inverter Based (d) Proposed XOR/XNOR Configuration [85]

Figure 4.16 on page 88 provides several of the reduced transistor count XOR/XNOR cells provided in [85]. A variation of the transmission gate XOR cell has been employed by Zhuang and Wu [86] in the development of transmission gate based full adder circuits (Figure 4.17). The principle advantage in such designs is the strong transfer of signals without threshold loss; however this comes at the expense of weakly driven outputs, and the necessity for the XOR/XNOR signal pair for the transmission gates.

Figure 4.17 Transmission Gate based Full-Adder circuit

One application of transmission gates, is through the use of pass-transmission circuits, where the overall flavour of the design is pass transistor logic with transmission gates applied only where needed for full swing capabilities. Damu Radhakrishnan is a strong advocate of the use of pass logic in arithmetic circuitry, and has proposed several circuits based on the hybrid pass-transmission gate logic style for full adders, and 4:2 compressors [81][87][98]. He has proposed the use of a combination XOR/XNOR cell along with transmission gates to form a full swing full adder (Figure 4.18 [87]). The vantage point in his design lies in the formation of the XOR signal and its complement simultaneously for controlling the operation of the output transmission gates.

There has been a considerable amount of literature surveying and providing novel circuitry for full-adders based on pass logic principles. These circuits vary in size from 20 transistors down to 10 as the current minimum, with every other combination of transistor counts present in between. The driving force behind the development of such circuits lies

in the realm of low power electronics. The notion of *POWERLESS*, and *GROUNDLESS* logic gates has been presented in [89], where the authors eliminate all connections to the power supply and all paths to ground respectively. This in effect limits the short circuit power dissipation, which relies on conduction paths between the power supply and ground.

Figure 4.18 Pass-transmission implementation of a Full Adder

Figure 4.19 10 Transistor Full Adders outlined in [89]

The authors in [89] present an analysis of various configurations of 10 transistor full adders, a sampling of which is provided in Figure 4.19. The conclusions drawn from this investigation point to ADDER9A as the lowest in power consumption, and ADDER13A as the best performing. Furthermore, the conclusions provide favourable results for the use of such configurations over a standard CMOS adder, as shown in Figure 4.20, under both power and performance analysis.

(a)

(b)

Figure 4.20 Conventional 28 transistor CMOS full adder implementations
 (a) standard configuration (b) Mirror cell configuration [84]

Shams et al. [90] have carried out a similar analysis of full adder cells for power and performance measurements. Their approach involves the decomposition of the full adder into three sub-modules, namely the *XOR/XNOR*, the *SUM* block and the *CARRY* block. Through the various configurations of the possible combinations of such modules, a comprehensive collection of full adder cells is developed. The circuit having the least power dissipation is depicted in Figure 4.21. In a separate investigation [91], a 10-transistor full adder (Figure 4.22) is developed having approximately one half of the power dissipation of comparable circuits.

Figure 4.21 Low power 16 transistor full adder cell

Figure 4.22 Low power 10 transistor full adder cell [91]

In one of the most complete manuscripts on the topic of full adder designs, Alioto and Palumbo [84] provide an in depth analysis of several of the most promising full adder designs, including:

- Standard CMOS (conventional and mirror cell implementations as in Figure 4.20)
- LEAP [92] (as in Figure 4.23)
- Complementary Pass-Transistor Logic (CPL as in Figure 4.24)
- Optimum circuit as defined in [93] (Figure 4.21)
- Transmission gate full adder (Figure 4.17)
- DOMINO logic full adder (Figure 4.15)

In this investigation, the authors provide simulation results of the cells, implemented in 0.35 micron technology, using power consumption, *power delay product* (PDP), voltage scaling and a novel metric used for calculating delay. Their results naturally pointed in favour of *DOMINO* logic for high performance implementations, and CPL as a high performance lower power alternative. The lowest power dissipation belonged to the cells that had no driving capability, namely the transmission gate and pass-transmission configuration. The limitation of these cells lies in their inability to provide signal drive strength, and so were recommended only for applications where high fanout, or cell cascading are not present.

The analysis did fail to consider most 10 transistor full adder implementations, however this oversight was justified by the authors' claims of signal degradation issues connected to such designs. The overall conclusions of the study revealed the advantages of using the robust conventional CMOS implementations, and the closely related 28 transistor mirrored-cell design. In the simulations involving decreasing supply voltage levels, the conventional CMOS logic circuits maintained a relatively consistent performance and power consumption measurements, which did not degrade to the extent as the other logic styles.

Figure 4.23 LEAP full adder cell

Figure 4.24 Complementary Pass-Transistor Logic (CPL) full adder cell

4.4 4:2 Compressor Circuits

In Chapter 3 the notion of the 4:2 compressor was introduced. Though the most primitive structure of the 4:2 compressor remains a simple two level cascade of two full adders, it was demonstrated that further optimization could be achieved through the decomposition of the cells. To this point, two gate level representations have been provided that present a reduction in the overall cell latency (Figure 3.12 on page 51 and Figure 4.13 on page 85). In this section, the 4:2 compressor is further broken to exploit the benefits of transistor level optimization.

Though there have been claims that the particular number system chosen, whether redundant, radix-2, or signed digit representations using various digit sets, effects the overall characteristics of the partial production reduction network. However, a recent paper by Peter Kornerup [99] elegantly disproves such notions and shows that the same 4:2 structure, with minor modifications to the external interface, may be used for all implementations. In his conclusions, he notes that no fundamentally different encoding system will allow for performance gains. Consequently, any further analysis of 4:2 compressors may be examined on a circuit level, without the need to take number systems or digit encoding into consideration.

Although not as popular a research area as the full adder, there have been quite a large volume of papers focused on the transistor layout of the 4:2 compressor [56-58, 98-103]. Several of the more interesting, and promising alternatives to the stacked full adder configuration will be presented in this section.

The first design, introduced in 1991 by Mori et al. [100], was incorporated into a 54-bit multiplier. This 58 transistor design is shown in Figure 4.25 on page 96, and in this schematic the longest delay path (1.2 ns) is identified. The downfall of this design rests in the number of inverters that are used throughout the compressor, leading to high short circuit power consumption, and in many cases unnecessary delay.

In a similar 54x54 multiplier [57], the authors present a multiplexer based approach to the design of the 4:2 compressor. The cell, implemented in 0.25 micron CMOS technology and having a latency of 460ps, features the use of complementary pass transistor logic multiplexer cells. An overly simplified rendition of the compressor is presented in Figure 4.27. The significant omission from this schematic is either the necessity for dual rail signals, or the placement of an inverter for compliment formation prior to each MUX cell.

Figure 4.25 1.2 ns 4:2 compressor proposed in [100]

Figure 4.26 Precharged pass logic compressor [56]

Figure 4.27 Multiplexer cell based pass-logic 4:2 compressor [57]

In order to increase the performance of the compressor cell, Hanawa et al. [56] propose the use of precharged pass transistor. In their design, *DOMINO* logic is used in order to boost the charge/discharge times of the pass logic circuitry at the outputs of the cell (Figure 4.26 on page 97), thus forming a 417ps latency compressor cell using 0.3 micron CMOS technology. As typical with all dynamic logic devices, this circuit suffers from high dynamic power consumption.

Margala and Durdle [101] explored the use of not only different logic styles, but also a particular technology in their development of new 4:2 compressor cells. In their research paper, new 4-2 compressors, based on new BiNMOS, a reduced-swing DPLBiNMOS and a BiDPL logic, are presented, where the circuits are designed and fabricated in 0.8 micron BiCMOS technology. Though the use of the BiCMOS process does present several unique advantages, unrealistic loading conditions were used in the power and delay analysis, thus not providing realistic loading conditions for the new circuits. Nevertheless, the standard CMOS implementation demonstrated relatively low average power dissipation in their investigation, second only to the BiDPL.

In a separate study [102], DPL is once again employed in the construction of the gate level circuit depicted in Figure 4.28 on page 99, primarily for the reduced internal load capacitances in the critical path. A 4:2 compressor is proposed in [58] that also uses dual rail DPL, for high performance applications. The use of inverters within the signal path is eliminated in this design, thus reducing short circuit power, and inversion delays.

In order to minimize the power dissipation while avoiding the drawbacks of non-full swing outputs and signal deterioration, the authors in [103] devise a 4:2 compressor layout that incorporates a series of full swing (FS) and non-full swing (NFS) multiplexers as shown in Figure 4.29 on page 99. In this manner the signal will never propagate through a succession of non-full swing cells, and so signal integrity is maintained. The advantage of the dual rail styles stems from the simultaneous availability of both the original signal and its complement. This benefit comes at the obvious expense of twice the wire interconnect overhead for every internal signal.

Figure 4.28 Gate level framework for DPL based compressor cell in [102]

Figure 4.29 DPL compressor using FS and NFS MUX cells [103]

For high speed and low power, pass transistor logic is once again claimed to be the logic style of choice for 4:2 compressors according to Damu Radhakrishnan [98]. Structured using the same cells as in their full adder design [87] (Figure 4.18 on page 90), the authors present a compressor cell which is derived from the relationship:

$$\begin{aligned}
 S &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus C_{in} \\
 C &= (X_1 \oplus X_2 \oplus X_3 \oplus X_4) C_{in} \\
 &\quad + (\overline{X_1 \oplus X_2 \oplus X_3 \oplus X_4}) X_4 \\
 C_{out} &= (X_1 \oplus X_2) X_3 + (\overline{X_1 \oplus X_2}) X_1
 \end{aligned}$$

Reducing the above representation, we obtain:

$$\begin{aligned}
 H_3 &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \\
 S &= (H_3) \overline{C_{in}} + (\overline{H_3}) C_{in} \\
 C &= (H_3) C_{in} + (\overline{H_3}) X_4
 \end{aligned}$$

A 4:2 compressor circuit is developed using the above relationships (Figure 4.30).

Figure 4.30 Pass-transmission implementation of the 4:2 compressor [98]

4.5 Overview of Arithmetic Circuitry

This chapter has provided a brief introduction to the variations of logic styles, and design techniques that may be employed in the creation of arithmetic circuitry. With the necessary background in logic styles, coupled with a wide sampling of the most recent arithmetic circuitry from across the globe, the foundation for a systematic approach to arithmetic design is now in place.

In general, there is no consummate collection of arithmetic cells presented in any one paper. Each of the published manuscripts presenting either a survey or novel approach to the composition of arithmetic sub-cells have inherent biases towards either a certain logic family, or a particular test/simulation paradigm. This section will attempt to summarize and present a design formalism for the simulation and synthesis of arithmetic macrocells.

4.5.1 Logic Style Selection

As mentioned earlier, there exist a plethora of basic logic styles for a circuit designer to choose from. The decision must be centred around the application and the objectives of the overall design. Naturally it would be absurd to incorporate a dynamic logic style, such as *DOMINO* logic, in a power conscious system; however, there are several concepts which must be taken into consideration which may not be as intuitive or straightforward, and are often overlooked.

The use of pass transistor logic for the most part should be approached with great discretion. Although this logic style has a great following in arithmetic circuitry for its unsophisticated approach to non-monotonic gates, there are countless more who strongly oppose its use. The standard CPL and DPL logic families, if deployed as they are intended, offer no overall gains in any one category over conventional CMOS [77][84]. In fact, CPL cells suffer from static power consumption at the inverters of the output nodes, due to the low voltage swings at the inputs of the nodes [96]. The inevitable drawback to such logic styles is their use of dual rails; a problem which will only compound as feature sizes continue to shrink, and interconnect dimensions begin to dominate. Furthermore,

having a signal and its complement present will lead to guaranteed doubling of switching activity for any signal transition.

The logic styles exhibiting the most applicable potential are in the hybrid transmission gate/ pass transistor logic, which has been coined “pass-transmission logic”. The majority of the low transistor count full adder cells fall under this general classification, and present some unique advantages. Naturally, the size of such circuits is smaller, and there are fewer internal nodes, thus decreasing both capacitance, and internal switching activity; two key elements in low power design. Finally, the circuits are all single rail logic.

There are downfalls with the use of such non-conventional logic styles. The case of the 10 transistor adders, and all of the variations thereof, exhibit threshold voltage loss (gain) issues at one point or another. Through the elimination of the transmission gate at the output nodes, the reduced transistor count full adders will inevitably have non-full swing outputs. Though some claim that reduced voltage swing is a means of reducing power consumption, it has a larger affect on signal quality, degradation and interpretation than any minimal power savings. For this reason, the most recent analysis of full adder cells [84] completely dismisses the 10 transistor class of adders as a viable solution.

Moreover, the topic of threshold voltage loss becomes an even greater challenge for deep sub-micron technologies having reduced supply voltages. The scaling of supply voltage with respect to device threshold voltage is not proportionate. The general trends have indicated that the process-nominal V_T has been scaling in proportion to V_{DD} , however the worst-case low V_T has remained constant. This V_T floor is set by various chip parameters that become increasingly significant as the overall supply voltage is decreased. Figure 4.33 on page 109 represents an empirical fit to the nominal-process V_T trend [104]. It can be deduced that the threshold voltage will represent a larger relative percentage of the supply voltage with technology scaling. Consequently any threshold voltage loss (or gain) will have a greater effect on signal integrity, leading to invalid logic level interpretations. The threshold voltage has an increased dependence on channel length as the length is

decreased (Figure 4.32). Threshold voltage tolerance is also worse with short channel devices. Thus any process variation at these short channels will have a significant impact on the threshold voltage.

Figure 4.31 Threshold voltage floor with decreasing supply voltage [104]

Figure 4.32 Threshold voltage tolerance dependence on channel length [104]

Furthermore, *Narrow Channel Effects* on threshold voltage levels in deep submicron technologies may further ostracize the development of pass-logic circuitry. A narrow channel is defined as a MOS transistor having a channel width on the same order of magnitude as the maximum depletion region thickness (x_{dm}). The predominant narrow channel effect is the increase in threshold voltage. This is due to the thicker field oxide

(FOX) that exists on the edges of the channel. This creates a shallow depletion region beneath the FOX overlap area. The gate voltage must then be capable of supporting the additional depletion area before a conduction path is established. In wider devices, the charge contribution of this fringe area is negligible in comparison to the overall channel depletion charge, however since it must be accounted for in the narrow channel device model, the immediate effect is an increase in the threshold voltage [78].

A secondary concern with pass-logic devices in deep submicron implementations in the concept of *Subthreshold Current Flow*. The current flow in the channel of a typical MOS device is dependent on the creation of an inversion layer through the application of a gate bias voltage. In the standard model, if this bias voltage is insufficient ($V_{GS} < V_T$), a potential barrier blocks the flow of current. However with diminishing device geometry, the potential for subthreshold current flow exists for conditions where typically no current would be flowing. This is a result of the dependency of the potential barrier on both the gate to source voltage (V_{GS}) in addition to the drain to source voltage (V_{DS}). If a voltage is applied at the drain, a phenomenon known as *Drain-Induced Barrier Lowering* (DIBL) decreases the potential barrier, thus allowing for electron flow in the channel [78]. Some logic styles, pass-logic in particular, are more prone to DIBL effects.

The suggestion of *POWERLESS* and *GROUNDLESS* logic gates [89], is a clear indication that the signals must be used to drive the outputs; this is in fact the underlying principle behind pass-logic. The congenital flaw is that the signals are not decoupled, that is to say that a signal may be transmitted directly through the cell. Under ideal situations, and inadequate simulation environments using unsophisticated models, the line and transistor losses are neglected; however this is not the case. The passing of a signal from the source to the drain of a transistor will undergo signal degradation to a certain extent, and will eventually reach a state where the logic state may be too weak to be properly identified.

Considering that signals must be passed *through* transistors, sizing of transistors becomes a critical factor in the cell's operation. Having sizing dependant, otherwise known as *ratioed*, logic makes the formation of standard cell libraries with such devices more difficult.

With each signal seeing a variety of input capacitances at the input of the subsequent cell, the drive strength of the preceding cell must be adequate to match the output loads. This leads to the inability of scaling the cells both within a given technology, and most certainly across technologies. This is a critical dilemma when dealing with a logic style's robustness and ease-of use.

Besides the pass-logic families, the standard transmission gate logic adders face their own challenges. Though they offer the benefit of full voltage swing outputs, the outputs are still driven by other signals, and not clean paths to logic '1' or '0' (V_{DD} and V_{SS} respectively). For situations where the cells must be cascaded, which is in most applications of arithmetic macrocells, the driving power of the cell is imperative. To do accommodate for this, at least one of the following methods must be exercised:

- Proper sizing of the transistors in the critical path; usually involves increasing the transistor's size in order to meet current demands
- Buffer insertion following every cell
- Using cells with pre-buffered (inverted) outputs

The first alternative is usually avoided, from a power consumption point of view, where the addition of extra transistors leads to a more efficient design, than having abnormally large transistors [90]. On a layout level, having similarly sized transistors facilitates the implementation of a circuit. By increasing the aspect ratio of a transistor, the drain, source and gate dimensions are increased, subsequently raising the overall and parasitic capacitances. Thus the notion of regularity may be passed down from system regularity to device regularity as well.

The analysis of 10 transistor adders as presented by Bui et al. [89] clearly states that the circuits developed suffer from threshold loss and drive-strength problems, yet the authors claim that "the adders are useful in larger circuits such as multipliers despite the threshold-loss problem." This statement is completely unjustified and misleading. Even a regular array multiplier topology will suffer from signal degradation through the cascade of reduction blocks; this problem is further aggravated in the irregular tree multiplier layout.

Buffering is most certainly required, and this is one of the most commonly overlooked issues in circuit performance calculations.

One of the most widely studied aspects of dynamic power reduction in modern CMOS devices is the switching activity. A great deal of effort has been put forth towards the reduction of switching activity, and consequently the number of times a node capacitance is charged and discharged. Besides algorithmic, or desirable, switching activity, there are spurious transitions that may occur due to unequal signal propagation paths between blocks, known as glitches. Glitches add unnecessary transitions to a system due to critical races or dynamic hazards. One aspect of switching activity that is often overlooked in power analysis is glitching that may occur within the internal nodes of a block.

This is a topic of interest since it investigates the effects of avoidable charging and discharging of the capacitive nodes that form the logic block. Investigations on glitching for the most part have been focused on the superfluous transitions that may occur between logic gates, and not on the unbalanced paths that exist within the blocks themselves as the signal transitions propagate through the transistors. By reviewing several of the proposed unconventional designs, it becomes apparent that there are uneven paths throughout these configurations. More importantly, in several designs (Figure 4.30 on page 100 for example), an input signal is used to generate an intermediary signal, which is then used to pass the original input signal once again. This is a clear example of unbalanced signal paths built into the design itself.

In terms of the more obscure logic styles using completely different number systems or technology process (BiCMOS for example) are not considered as part of any rational analysis. It would be imprudent to postulate that the use of unconventional number systems does not present any gains in performance, power, noise margins, scaling potential, or any other pertinent metric. However, the interfacing circuitry overhead will, for the most part, negate any gains that an individual arithmetic component may have in an overall system. For this reason, diverse number systems, or technology processes must be

regarded as system level designs, and not be compared with individual cells implemented using contemporary processes.

4.5.2 Simulation Setup and Environment

In order to conduct a valid comparison amongst various designs and logic styles, a fair and undistorted simulation environment is required. Unfortunately this appears to be the one of the most critical oversights in most publications. Although consistency in an investigation will, for the most part, produce a legitimate comparison under the given conditions, it does not necessarily obstruct experimental biases. The most cogent studies are those carried out under realistic conditions, with as sophisticated a model as possible.

In a recent study [84], the authors point out the importance of proper assessment of parasitic capacitances that are present in any physical implementation of a logic block. To account for this in their analysis layout level models with extracted parasitics were used. This is a requirement for the proper simulation of pass-transistor based designs, since the internal parasitics have a considerable affects on the transmitted signal that must propagate through the transistor array. Less complex models, such as PSPICE, will not properly account for this, and so will yield overly optimistic results.

Furthermore, this method of modeling allows for the proper assessment of a cell's physical dimensions as opposed to merely providing vague transistor counts. As an example the 22 transistor LEAP and 26 transistor TG designs were determined to be 26% and 14% larger, respectively, than the 28 transistor CMOS design. The authors provide sound evidence refuting the comparison of macrocells through transistor counts as a valid measure of size. This size difference is due to the high circuit regularity of the conventional CMOS topology as opposed to the irregular, and often large transistor requirements of the other logic styles [77].

A separate issue that must be taken into consideration for pass-logic and pass-transmission circuitry is the fact that many signals are connected to a gate of one or more transistor(s)

and the source of other transistor(s) at the same time. When a current is drawn from an input, switching of the signal connected to the corresponding transistor source slows down. If that signal is connected to another gate simultaneously, the switching of that transistor, and consequently another signal, will also slow down. This phenomenon, known as the *source-gate effect* [77], must be taken into account for proper simulation results, and circuit latency calculations.

Uneven signal path considerations should also be taken into account. Delay of the input signals through a range of values should be analyzed, and their effect on circuit performance. This will properly model the variations that are possible in the manufacturing process of a die. The CMOS fabrication process is highly complex, consisting of numerous stages and procedures, thus it is inaccurate to assume that all circuits will behave similarly once produced. The variations in manufacturing processes and operating conditions will inevitably alter the circuit parameters from the target or nominal values. The circuit simulations should incorporate a nominal value for a given parameter (such as W , L , temperature, V_T , t_{ox} , etc.), in addition to a statistically varying component of that parameter to accurately reflect the range or tolerance that a device has to deviations from the norm.

In terms of the simulation test bench used, an environment which closely mimics the cell's functional application is advised. If a cell is proposed for high order counters, or partial product reduction arrays, then it should be modeled and tested within those settings. It is insufficient to merely replace the cascading effects of such irregular topologies with idealistic capacitance models and take measurements using only a limited test structure as proposed in [96]. The test circuit shown in Figure 4.33 on page 109 [91] is acceptable only for the case where the cells are not cascaded. Through isolating the device under test, the dynamic loading effects present in pass-logic designs are eliminated. Consequently, the simulation results provide an idealistic environment for such cells where the signal is decoupled, and so is less susceptible to degradation. Moreover, the necessity for inter-stage buffering has been addressed in many papers [77][84][90][91], yet the buffers are often not included as part of the cell's latency or power consumption.

Figure 4.33 Typical test-bench used for Full-Adder simulations [91]

For the most accurate representation of the cascaded environment for a full adder, or for a 4:2 compressor, a simple test bench based on an 8x8 partial product reduction matrix has been developed. Figure 4.34 and Figure 4.35 show the internal configuration of the multiplier reduction array used for the compressor and full adder respectively, while Figure 4.36 outlines the testbench setup used in the Cadence Virtuoso environment. The buffered inputs provide realistic source inputs, as opposed to a direct connection to an ideal voltage source. The individual blocks within the reduction trees may be interchanged such that a wide sampling of cells may be tested using the same configuration.

In this manner, a more dynamic simulation environment is provided which accounts for the irregular interconnections, and loads of the individual cells. Thus the sizing of the transistors, where necessary, may be done by examining all possible applications and locations of a reduction subcell. The testbench is further enhanced if the extracted layouts of the individual cells are used, where the parasitics are accurately modeled.

Figure 4.34 4:2 Compressor distribution in an 8x8 reduction matrix

Figure 4.35 Full-adder distribution in an 8x8 reduction matrix

Figure 4.36 cadence testbench for 8x8 multipliers

The proper simulation environment is incomplete without a suitable set of input stimuli. The need for a robust input transition pattern has been examined by Shams and Bayoumi [88], and in their study they have uncovered some very interesting results. By selecting a set of input signals that do not cover the entire range of possible transitions, they managed to reveal invalid average power consumption and latency values for a set of full adders. Their conclusions stated the need for input signals that not only cover every incoming combination of logic values, but also a valid permutation of all possible input *transitions*.

The input signal transitions force the various charge and discharge paths of the internal nodes of the device under test. The charge differences between subsequent states substantially effects the power consumption and latency calculations. For a full adder, having three inputs, there are N_{3T} input combinations, where:

$$N_{3T} = 8 \times 7 = 56$$

Similarly, the number of combinations for a 5 input 4:2 compressors is:

$$N_{5T} = 32 \times 31 = 992$$

In general there is a need for:

$$N_{kT} = 2^k \times (2^k - 1)$$

input combinations for a system having k inputs, where the entire range of 2^k input combinations must be used against all $2^k - 1$ combinations of the proceeding transition.

In terms of the actual delay calculation, a new metric has been developed in [84] that incorporates the propagation delay (τ_{PD}) information, in addition to the rise/fall time (t_R). The propagation delay alone will not account for the output signal shape, whereas the rise and fall times have no bearing on the input signals and only describe the output dynamics. The novel figure of merit, denoted as t_{SPEED} , represents the time required to reach the steady state output value starting from the time in which the input signal crosses the logic threshold:

$$t_{SPEED} = \tau_{PD} + \frac{t_R}{2}$$

The calculation of power dissipation is much more complex situation. In conventional CMOS circuits, the overall power dissipation may be calculated as the total power drawn from the supply. When considering pass-logic and transmission gate based circuits, this calculation becomes much more complex since now the power drawn from the input signals must also be considered. In discussions with Dr. Graham Jullien [105], Dr. Jim Haslett [106], Dr. Wael Badawy [107] and Dr. Farid Najm [108], no single resolution to this dilemma has been formulated. However the common stance on the need to take into consideration the power drawn from the input source has been established.

The decomposition of such cells into their individual constituents and capacitive nodes is far too complex an approach, and an overly involved solution. The general approach should be to consider the entire circuit as one entity [105 - 108], and determine the power drawn by the entire cell by examining the current flow from the supply using one of the two power measurement subcircuits [109] outlined in Figure 4.37.

Figure 4.37 Power measurement circuits proposed in [109] having a controlled current source and parallel RC circuit (a) current controlled (b) voltage controlled

To account for the power drawn from the inputs, the dissipation through the input buffers may be analyzed by first measuring the unloaded buffer consumption, and subtracting this value from the loaded buffer consumption. This will provide the difference in power consumption as a result of the device under test's loading conditions on the input. A more accurate measurement would be to analyze a cascaded structure's overall dissipation, and use this as a comparison amongst subcell configurations.

In general the power dissipated through the buffers should be considered as part of the overall cell consumption. This is justified when one considers the fact the buffers are an integral component of the low transistor count cells, without which they would not be able to function properly. Furthermore, an unfair advantage is given to such circuits over conventional CMOS since the standard CMOS circuit does not require refreshing at its output nodes. An argument may be made that in most practical applications of large scale multipliers, the stages are pipelined, thus the cascading effect is not as severe, and refreshing may not be required. To support this argument then, the cells must be loaded and driven by the pipelining latches, and the effects of poor drive capacity, and non-full swing outputs must be taken into consideration for the latches involved.

Conventional CMOS logic has, to date, withstood the test of time, proving to be the most resilient logic style in terms of technology scaling, robustness, variations in operating conditions, ease of use in standard cell applications with synthesis tools and interfacing with its environment. Consequently it should be the targeted logic style for the majority of applications, unless in the extreme case where ultra low power, or high-speed are of the essence, and the price in the trade-off of the other factors is not a consideration.

Chapter 5

Interconnect Effects

“Conventional interconnect scaling will no longer satisfy performance requirements. Defining and finding solutions beyond copper and low k will require material innovation combined with accelerated design, packaging, and unconventional interconnect.” - SIA 2001 Technology Roadmap [9]

To date, the semiconductor industry has been able to satisfy Moore's Law through advancements in process technology alone. Moore's Law is named after Intel executive Gordon Moore who observed that the general trend over the past 30 years is that functionality per chip (transistor count) in addition to microprocessor performance (clock frequency x instructions per cycle = millions of instructions per second MIPS) doubles every 1.5 to 2 years.

However, according to the latest International Technology Roadmap for Semiconductors (ITRS 2001), published by the Semiconductor Industry Association in 2001 [9], breakthroughs in conventional technologies are required in order to meet the more aggressive scaling requirements through decreased feature sizes that are required in order to maintain the historical trends. Feature size reduction refers to the minimum channel width (poly width

available) or minimum interconnect width available for a technology. Suggestions for optical or wireless interconnects have been presented in order to alleviate the restrictions imposed by current metal layers [110]. Until those technologies appear as a feasible alternative, focus must be placed on the development of strategies for overcoming the obstacles using current processes.

The progression of fabrication technologies opens the door for more complex applications and developments targeting the integrated circuit. The monolithic integration of a large number of functions on a single chip provides:

- Less area
- Less power consumption
- Less system level test requirements
- Higher reliability (on chip interconnects)
- Higher Speed (on chip interconnects)
- Cost savings

Current designs focus on the concerted performance of interconnected transistor systems, and not on the functioning of individual transistors. With the foreknown gains of amalgamating multiple functions and systems onto a single piece of silicon, the complexity of the integrated circuit and the interaction of various sub-components is bound to increase. For this reason it is imperative to take into consideration the manner in which the transistors and subcells are interconnected, and not merely singular transistors.

The predominant motivation behind the study of interconnect effects has been the result of technology scaling effects. When physical devices are scaled from one technology to another, the transistor dimensions are reduced, as are the lengths of the local interconnects joining the transistors. Although transistor delay scales appropriately with device size, interconnect delay does not; in fact, as feature sizes shrink, wires contribute a larger portion of the total delay [12]. This is clearly outlined in the graph developed in [111] and presented in Figure 5.1 on page 117.

5.1 Projected Issues With Technology Scaling

The SIA produces a general survey of semiconductor technology approximately every 4 years outlining the predicted technology over the next several years, while outlining potential bottlenecks. The most recent issue was released in 2001, with an update in 2002 [9]. In this report, several areas have been designated as "*Red Brick Walls*", referring to predicted objectives that have no current solution and will not be met unless innovative solutions are developed. The relevant sections of the 200 page report pertaining to interconnections will be outlined in this section.

With the predicted microprocessor speeds exceeding 28GHz, and feature sizes below 0.009 micron (9 nm) by 2016, it is reasonable to agree with the claims that the existing CMOS technology as we know today will not be able to maintain these requirements. Breakthrough technologies are required to not only carry us to, but also beyond, these objectives. Currently, the interconnection density, measured as the length in metres of metal interconnects per square centimetre of active chip area, is 4843 m/cm², and it is predicted to reach 11169 m/cm² by the year 2007. Furthermore, the delays per millimetre of local metal interconnections are now 121 ps and are expected to increase by 200% by 2007, and 1600% by 2016 (depicted graphically in Figure 5.3 on page 122). Table 2.1 on page 117 summarizes a few of the values related to interconnect effects taken from the SIA roadmap.

One often overlooked topic in the analysis of interconnects in VLSI circuitry are thermal issues. The flow of current through metal wires causes wire-self-heating, which has a twofold effect on chip design and reliability. As depicted in Table 2.1 on page 117, the maximum allowable current density is of great significance in future technologies, and wire-self-heating will add to this problem by further limiting the maximum allowable current density. Secondly, wire reliability, which is governed by electromigration (EM), has an exponential dependence on the inverse of metal temperature [112].

Figure 5.1 Contribution of interconnect effects to overall delay

Table 2.1 2001 SIA Roadmap Summary

Year of Production	2001	2002	2003	2004	2005	2006	2007
Interconnect RC delay 1 mm line (ps)	86	121	176	198	256	303	342
Local Wiring Pitch (nm)	350	295	245	210	185	170	150
Total interconnect Length (m/cm ²)	4086	4843	5788	6879	9068	10022	11169
Local wire aspect ratio for copper	1.6	1.6	1.6	1.7	1.7	1.7	1.7
Jmax (A/cm ²)—wire (at 105°C)	9.60E+05	1.10E+06	1.30E+06	1.50E+06	1.70E+06		
Interconnect RC delay 1 mm line (ps)	53	75	101	127	155	191	198
Intermediate wiring pitch (nm)	450	380	320	275	240	215	195
Intermediate wiring dual Damascene A/R (Cu wire)	4.6	1.6	1.7	1.7	1.7	1.7	1.8
MPU Printed Gate Length (nm)	90	75	65	53	45	40	35
Chip Frequency (MHz)	1684	2317	3088	3990	5173	5631	6739
MPU High-Performance Total Chip Area (mm ²)	310	310	310	310	310	310	310
MPU High-Performance Active Transistor Area (mm ²)	28.7	28.2	27.7	27.2	26.8	26.8	26.8
Equivalent Oxide Thickness - Tox (Electrical) (nm)	2.3	2.1	2	2			
Nominal power supply voltage (Vdd) (V)	1.1	1	1	1	0.9	0.9	0.7

Year of Production	2010	2013	2016
Interconnect RC delay 1 mm line (ps)	565	970	2008
Local Wiring Pitch (nm)	105	75	50
Total interconnect Length (m/cm ²)	10053	22695	33508
Local wire aspect ratio for copper	1.8	1.9	2
Jmax (A/cm ²)—wire (at 105°C)	2.70E+06	3.30E+06	5.50E+06
Interconnect RC delay 1 mm line (ps)	348	614	1203
Intermediate wiring pitch (nm)	135	95	65
Intermediate wiring dual Damascene A/R (Cu wire)	1.8	1.9	2
MPU Printed Gate Length (nm)	18	13	9
Chip Frequency (MHz)	11511	19348	28751
MPU High-Performance Total Chip Area (mm ²)	310	310	310
MPU High-Performance Active Transistor Area (mm ²)	26.8	26.8	26.8
Equivalent Oxide Thickness - Tox (Electrical) (nm)			
Nominal power supply voltage (Vdd) (V)	0.6	0.5	0.4

White—Manufacturable Solutions Exist, and Are Being Optimized	
Yellow—Manufacturable Solutions are Known	
Red—Manufacturable Solutions are NOT Known	

The dilemma associated with thermal issues is exacerbated with technology scaling. Increased current densities will add to the thermal issues, since the RMS value of current density is responsible for heat generation. The use of low- k dielectrics (materials having a lower thermal conductivity) and the increased number of metal layers, will contribute to the thermal issues by trapping more heat in the wires. Finally, wire congestion will increase not only the capacitive coupling between adjacent wires, but also thermal coupling.

Though the use of unconventional interconnect methodologies and materials have been suggested in the SIA Roadmap, the elimination of metal wires does not appear to be viable option in the near future. A summary of several of the proposed alternatives are provided in [119]. The use of low-loss off-chip thin film wiring, and optical interconnects appear to be the most promising alternatives to the conventional metal layer medium of present day processes. These solutions, if at all attainable, will be aimed at lengthy cross-chip lines and buses, and will see limited service in the local interconnect applications.

5.2 Physics of Wire Interconnects

Due to the large bandwidth and ultra fast switching speeds of modern devices, transmission line theory is required in order to properly assess signal propagation in on-chip interconnects. By investigating the transmission line effects of wire interconnects, a threefold concern of wires on circuit performance arises: added capacitive loads to driving gates, signal delay, and coupling noise. These arise due to the electrical characteristics of wires, namely resistance, capacitance and inductance, and the wire's geometry and relative position. Such effects will have implications on circuit performance, power consumption, reliability and cost. Ho et al. [15] have studied individual effect of each of the characteristics and their summarized results will be subsequently discussed.

All types of wire interconnects exhibit resistance as charge flows through them. Wire resistance (measured per unit length) is calculated as the material resistivity divided by cross sectional area. The resistance grows quadratically as the cross sectional area is

decreased. To counter this, wires are not scaled by an ideal scaling factor:

$$S = \frac{1}{\text{feature size}}$$

but by a quasi-ideal scaling factor:

$$S = \sqrt{\frac{1}{\text{feature size}}}$$

This creates a taller wire while maintaining a constant cross sectional area [12]. With scaling effects, the wire resistance will inevitably increase as the height and the width of the wires scale down with the technology.

With an ideal scaling factor, having the overall wire dimensions decrease by a scaling factor S , the wire resistivity per unit length will increase by a factor of S^2 [110]. This value is reduced to $S^{1.5}$ with the quasi ideal scaling methods. It may be argued that the overall length of the wire interconnect will also be reduced by an equivalent scaling factor, since the individual points will now be located within closer proximity, and based on this notion, the total wire resistance will remain constant. It must be pointed out that the general trend, as outlined in the SIA roadmap, is for an increase in chip functionality and device counts as the technology scales. Hence, the assumption that as feature sizes decrease, so too will chip sizes does not necessarily hold true.

A system designer will take advantage of the enhanced circuit density to increase the on-chip system functionality, and with it increase the chip complexity [110]. For this reason, it is safe to expect wire resistances to increase. One means of overcoming this dilemma is through the use of *locally optimized arrays*, which will be subsequently discussed in further detail.

Wire capacitance is a result of required charge that must be added or removed from a wire in order to change its electric potential. For the most part, basic capacitance models simply take into account the parallel plate capacitor effects of wires over a plane. A generally accepted capacitance model has been introduced in [113], and is used for the per unit

length capacitance estimation with edge fringing effects taken into consideration. The model has two representations, each dependent on the wire aspect ratio.

For $w \geq \frac{d}{2}$

$$C = \epsilon_x \left(\frac{w - \frac{d}{2}}{t_{ox}} \right) + \epsilon_x \left(\frac{2\pi}{\ln \left(1 + \frac{2t_{ox}}{d} + \sqrt{\frac{2t_{ox}}{d} \left(\frac{2t_{ox}}{d} + 2 \right)} \right)} \right)$$

For $w < \frac{d}{2}$

$$C = \epsilon_x \left(\frac{w}{t_{ox}} \right) + \epsilon_x \left(\frac{\pi \left(1 - \frac{0.0543d}{2t_{ox}} \right)}{\ln \left(1 + \frac{2t_{ox}}{d} + \sqrt{\frac{2t_{ox}}{d} \left(\frac{2t_{ox}}{d} + 2 \right)} \right)} \right)$$

In these expressions, w and d are the wire width and depth, t_{ox} and ϵ_x are the oxide thickness and dielectric constant, and C is the capacitance of the wire per unit length. According to the SIA roadmap [9], the aspect ratio of local interconnects will not exceed 2 in the foreseeable future, and so the first expression will be the one of prime importance.

As discussed previously, as the aspect ratio of wires increases, the side profiles of the wires will have a greater contribution to the overall capacitive effect. Thus, the wire is better modeled by four parallel plate capacitors, plus a constant fringing term [15]. Although the parallel plate capacitance decreases with wire dimensions, the benefits are offset by the increased coupling capacitance to neighbouring wires as a result of increased wire congestion per layer [14]. This increased trend in capacitance is in direct accord with the predicted rise in interconnect RC delay per unit length as graphed in Figure 5.3 [9].

Figure 5.2 on page 122 outlines the coupling capacitances associated with metal interconnects within an integrated circuit, with particular focus on the second metal layer

(M2). The complex interaction between the various wires may easily be visualized. In addition, the relative sizes and positions of the various metal layers may be observed, and the manner in which the wire dimensions increase with the metal layers. The layers appearing closest to the substrate are reserved for short local interconnects, and as the layers progress further away from the substrate they are used for intermediary and global lines respectively. The global lines have the largest dimensions simply for conductance purposes since the resistance of the larger lines will be much less than the finer pitch wires used for the local connections. The coupling capacitance between two wires, i and j , may be represented as:

$$C(i, j) = \frac{f_{ij} l_{ij}}{d_{ij}} \cdot \frac{1}{1 - \frac{w_i + w_j}{2d_{ij}}}$$

where, w is the wire size, f is the unit length fringing capacitance, between the wires, l is the overlap length of the wires, and d is the distance between the wire centres [114].

Apart from the obvious downfalls associated with a quadratic increase in interconnect delay with wire length [15], circuit robustness especially in terms of coupling noise is another serious hindrance. CMOS devices are voltage controlled, thus the voltage noise on the wires relative to voltage margins at the receiving gates is crucial. Capacitive coupling is the principle contributor to this type of cross-coupling noise, especially on weakly driven, or excessively long wires.

Cross-talk induced noise is a result of undesired voltage brought about through parasitic coupling capacitances from one net onto another. It has a significant and detrimental effect in deep submicron technologies, as it leads to decreased signal integrity and increased delays. The two side capacitances of adjacent lines will experience data dependent capacitances, upwards of 70% of the total capacitance, and in turn are the most common source of noise injection [15]. More recently, the effects of crosstalk and process variation have been combined in a new design parameter referred to as ***crosstalk sensitivity*** [114]. Crosstalk sensitivity is a measure of the effects of process variation on crosstalk. Based on the analysis in [114], it has been shown that the lower bound on crosstalk sensitivity

increases quadratically with technology scaling, while crosstalk increases linearly. Thus the significance of process variation on crosstalk effects are more profound in deep submicron technologies.

Figure 5.2 Coupling capacitances associated with metal interconnects

Figure 5.3 Interconnect RC delay per unit length (ps / mm)

Although there has been substantial research done as of late on accurate prediction and reallocation of problematic routing from a design level [114][115][116], this problem may only be moderated, not eliminated. VLSI CAD tools, in general, carry out routing in two phases. The *global routing* stage plans a coarse outline of the interconnections based on a

grid, which provides a tight boundary as to where the final wires may be placed. Maze search or Steiner embedding techniques govern the path congestion and delay requirements of the global routing process. Detailed Routing is the last step of the interconnect layout procedure, where the wires are placed within the predefined confinements according to the technology design rules. Neither routing stage is optimal for tackling the crosstalk effects, since global routing is too early for accurate prediction, whereas detailed routing does not have the liberty of restructuring the overall interconnection scheme to deal with conflicting aggressor and victim nets [116].

Since coupling effects are highly dependent on local geometries of interconnections, one solution has been to provide structural solutions in problematic areas, such as shielding wires, or repowering buffers. The use of repeaters has been shown to reduce crosstalk by up to 50% in VLSI implementations. In addition, the use of larger driver circuits have been shown to reduce the amount of crosstalk noise due to the reduced effective resistance; however, larger drivers lead to increased capacitive loads which in turn has an unfavorable effect on signal delay [117]. In Intel's 64-bit Itanium processor, 85% of the full-chip level routes require at least one repeater. There are approximately 13000 nets at the top level of the Itanium, 11000 of which require some type of a repeater [118].

In order to maintain the capacitance of interconnects relatively constant, the use of low- k dielectrics has been explored. The dielectric constant is directly proportional to the capacitance, which is also proportional to the leakage current. While low- k dielectrics reduce the parasitic capacitance between the metal lines, low- k materials are less mechanically, chemically, thermally, and electrically stable than the historical material of choice, deposited SiO_2 [119]. The use of new materials, such as Copper metal and low- k dielectrics, in fabrication processes help reduce the resistance and parasitic capacitance of the metal interconnect lines. This process, known as "Dual Damascene", is used to create the multi-level, high-density metal interconnections needed for advanced, high performance ICs. [120].

For the most part, inductance effects are a point of concern for global wires. This is due in part to the lower resistance per unit length of these lines which results in the reactive component of the interconnect impedance becoming comparable to the resistive component [121]. In addition, the longer current return paths in global wires added to the fact that they are the furthest from the substrate, creates substantial mutual inductive coupling [122]. Furthermore, it has been shown that a properly designed inductive line can reduce the total power dissipated [123]. However, these inductive effects may be ignored for local interconnections, since they are pertinent only to longer global wires and clock networks, and are dominated by the local RC effect [124].

5.3 Interconnect Effects On Arithmetic Circuitry

Historically, the measure of circuit delay has been a function of the gate delay. With this frame of mind, the reduction of the gate count in the critical path was determined to be an accurate reflection of performance gains. This rationale has been the driving force behind arithmetic circuitry development over the past several decades. Recently the physical restrictions imposed by interconnections of highly irregular topologies have surfaced as a credible limiting element in high performance design.

Recent studies [11][12][13] have examined the consequences of technology scaling on arithmetic circuitry. Multiplier circuits present several unique challenges, such as sizable areas, large volumes of short but irregular interconnections, and significant amounts of switching activity within their cascading chains of sub-cells. For these reasons, multiplier architectures are highly susceptible to interconnect effects, and are of particular importance in this area of research. As depicted in Figure 5.4, the delay of a double precision multiplier is greatly affected when the presence of interconnections is taken into account.

Choe et al. [11] have predicted that with uniform scaling into deep sub-micron technology, the larger the multiplier the greater the effects of interconnect delays. Their analytic evaluation of interconnect delays demonstrated that as the operand size of multipliers

increases, so too does the influence of interconnections on circuit performance. Huang et al. [13] have obtained similar results in their study of wire effects on prefix adders. Their simulations have shown that in many cases wire delay exceeds logic delay, and has a significant impact on the critical path delay. For a given technology, the contribution of wires on circuit operation, increases as the datapath, and operand width, becomes wider.

In the investigation of technology scaling effects on multipliers [12], several recoding schemes have been examined leading to some conclusions regarding the influence of algorithm level design. It has been observed that the recoding schemes generating the fewest number of partial products, while having the largest volume of logic cells in the critical path, are the least vulnerable to interconnect effects. This is only natural since limiting the number of partial products in the array will lead to fewer interconnects, while the placement of logic cells will in effect act as buffers along the wires, thus reducing the interconnect lengths.

Figure 5.4 Relative delay; binary tree to its no wire implementation for Booth 2, double precision multiplication [12]

5.4 Locally Optimized Arrays

The analysis and understanding of the ramifications of interconnections within arithmetic circuitry will present opportunities for the development of the next generation of arithmetic circuitry. It has been observed that the schemes least affected by interconnect delay are those which have minimum length and methodical wire placement. In an invited paper by Luigi Dadda [125], the author stresses that a successful VLSI architecture will satisfy the criteria that only a few types of simple cells must be laid out using local and regular connections, avoiding long, irregular data paths. In this section, the notion of forming complex arithmetic circuits through the application of locally optimized arrays will be presented.

The notion of system partitioning has been in existence for quite some time, and it has been suggested that future microprocessors must be further partitioned into independent physical regions in order to cope with technology scaling effects [14]. The decomposition of a large system into smaller sub-blocks has been explored on a large scale with suggestion for solutions to scaling effects in system on chip applications [126]. On a lower level, a partitioning scheme has been presented for the optimization of interconnect power. In [60], the authors advance the notion of distributed computing through their approach of exploiting locality through the subdivision of an algorithm into spatially local clusters.

The concept of “systolic arrays” closely resembles the proposed optimized array formalism. Systolic arrays are arrays of processors, or identical processing elements, which are connected in a mesh-like topology to a small number of nearest neighbours. Generally the operations will be the same in each processor, with each processor performing an operation (or small number of operations) on a data item and then passing it on to its neighbour. The lock-step data movement through the array of cells resembles the rhythmic pumping of blood in the veins. One of the primary requirements in systolic array architectures is that no unlatched signal is allowed to propagate across the cells [8]. This is the distinguishing factor between this and the locally optimized array design paradigm, which excludes this restriction.

The Pentium IV architecture represents one of the most familiar general purpose architectures, and employs the partitioning scheme for performance optimization. The double pumped (operating at twice the system clock) high-speed ALU core of the processor is kept as small as possible to minimize the metal length and loading. Only the essential hardware necessary to perform the frequent operations is included in this high-speed ALU execution loop. Functions that are not used very frequently, for most integer programs, such as the multiplier, shifts, flag logic, and branch processing, are isolated from the key low-latency ALU loop and are implemented independently [79].

The concept of partitioning may also be applied on an even lower level abstraction, where the fundamental segment of an algorithm's composition, namely the arithmetic blocks, may be partitioned. Once again, the Pentium ALU presents an example of this through the "scattered add" formation of the fast adder which is a cluster of 16-bit adder slices [79]. Through the use of locally optimized arrays, several of the drawbacks of scaling of larger circuits may be accommodated. This holds particularly true in the case of digital multipliers that are particularly prone to scaling effects. Though irregular topology of large tree multipliers has in the past offered significant performance gains, the use of such schemes in deep-submicron implementations has been shown to be inefficient [21][11]. To counter this, the decomposition of large multiplications into smaller multipliers is suggested, with the details of the algorithm presented in Chapter 6.

According to the literature supporting the relationship between operand size and interconnect related issues, the larger the arithmetic element, the greater the effects of interconnects. Moreover, it has been shown that the doubling of operand sizes in a multiplier increases the average power dissipation by more than a factor of four, and increases the power to area ratio [21]. The use of smaller multipliers leads to the generation of fewer partial products, leading to shorter interconnect requirements. Furthermore, the reduction in the number of partial products will in turn affect the average net length due to the fewer number of stages required in the reduction array. The signals will on average need to by-pass fewer reduction sub-blocks, and the distribution of the blocks will be more dense due to the scaled down partial product array.

The decomposition of a large tree multiplier into smaller multipliers will increase the overall design regularity, while maintaining the performance advantages of tree multipliers over the linear array multipliers. The use of smaller multipliers will increase the number of logic blocks in the critical path, and will cut down on the length of the longest nets, mimicking the insertion of small repeaters on such wires. The use of deep pipelining will achieve the same effect, but will not increase the circuit's regularity as significantly.

To support this theory, an analysis has been carried using various multiplier sizes implemented in 0.18 micron CMOS technology. Table 2.2 summarizes a sampling of the simulation results for the booth-recoded multipliers of various length. A complete summary of the analysis results, in addition to the simulation logs are provided in Appendix B .

Table 2.2 Summary of Interconnect Analysis for Various Multiplier Sizes

	64	54	32	16
<i>Number of Components</i>	12916	9391	3687	1069
<i>Number of Pins</i>	64481	56505	17559	5036
<i>Number of Nets</i>	14409	10388	3915	1140
<i>Average Number of Pins per Net</i>	4.48	4.48	4.49	4.42
<i>Total segments in regular wiring</i>	128059	82116	27057	6834
<i>Total segments in special wiring</i>	401	338	224	140
<i>Total wirelength in regular wiring (um)</i>	1512966.26	861731.66	230623.16	49271
<i>Total wirelength in special wiring (um)</i>	105351.24	78250.08	36160.6	15203.28
<i>Total wiring (um)</i>	1618317.5	939981.74	266783.76	64474.28
<i>Average regular wiring net length (um)</i>	11.8146	10.4941	8.5236	7.2097
<i>Average special wiring net length (um)</i>	262.7213	231.5091	161.4313	108.5949
<i>Average net length (um)</i>	112.3130	90.4873	68.1440	56.5564
<i>METAL 1</i>	146231.92	102118.66	44620.96	17036.44
<i>METAL 2</i>	273265.32	185962.3	69145.3	17575.84
<i>METAL 3</i>	475396.2	294621.48	96337.8	25701.42
<i>METAL 4</i>	388446.9	211710	40386.22	3398.28
<i>METAL 5</i>	194880.84	111835.58	16193	762.3
<i>METAL 6</i>	140096.32	33433.72	100.8	0
<i>Max crosstalk induced timing delta (ns)</i>	1.66	1.66	0.467	0.497

The results above strongly support the presumption that with increasing operand widths, the complexity and volume of the interconnections increases. The average net lengths, and

individual wire segment lengths increase dramatically with increasing multiplier sizes. Although it may be argued that the number of components does not increase by a factor of four as the operand width is doubled, it is clear that the advantage of partitioning a large operation lies in the increased regularity and size of the wire segments. Figure 5.5 outlines the relationship between multiplier size and interconnect length. The non-linear relationship between the net length and count with respect to the multiplier size is an accurate reflection of the decrease in device regularity with size.

The increase by a factor of six in the total net length of a 64-bit multiplier over a 32-bit multiplier, corresponds to the 3.5 fold increase in capacitive coupling (crosstalk) induced timing delay. As summarized in Table 2.2, the larger multipliers experience greater coupling delay than the smaller, more regular designs. This study is merely a reflection of the current 0.18 micron CMOS technology. The coupling effects are bound to increase as feature sizes continue to shrink.

Figure 5.5 Interconnect effects with respect to multiplier width
(a) Total chip interconnect length (b) Average chip interconnect length

The distribution of the interconnections within the metal layers is also of significant importance. It is clear that the smaller multiplier sizes, make use of the lowest metal layers, leaving the upper layers open for global routing lines. By relieving the

interconnect congestion within these layers, the more compact designs allow for greater routability of the overall design. In addition, the dimensions of the upper metal layers, by their fundamental design, are aimed at global distribution signals, and do not present any advantages (if not presenting detrimental inductance effects) to local signal wiring.

On a separate note, it has been demonstrated that the use of 4:2 compressors will reduce the total cell and interconnect counts in partial product reduction arrays (Figure 3.17 and Figure 3.18 on page 61 respectively). As mentioned previously in chapter 3, the 4:2 compressors provide a further advantage in interconnections since approximately one third of the overall wire count is devoted to the short interstage horizontal carry paths. Coupled with their high distribution regularity, the 4:2 compressors makes an ideal candidate for use in future multiplier algorithms, and a reduction subcell which promotes the locally optimized methodology for arithmetic circuit composition.

Chapter 6

Recursive Multiplication

The notion of carrying out multiplication by breaking up the operands into smaller sections has been in existence for several decades. Such schemes offer several advantages over performing standard multiplication. By breaking a large multiplication into recursions of smaller multiplications, the regularity of the design is increased, since smaller multipliers are inherently less complex. In addition, fewer, shorter interconnects are required to carry out the multiplication, with a limited number of global lines used to collect the final outputs of each recursion.

The name “Recursive Multiplier” may at first appear misleading, since in the implementation of this algorithm, there are no recursions, or repeated iterations of the same procedure. The process is simply broken down into smaller sub-processes which are carried out in parallel. However, for the sake of consistency with the authors [18] the same nomenclature is adopted.

This chapter will examine the essence of recursive multiplication, beginning with an overview and proof of this concept. Simulation results will then be presented to support its potential vantage points over conventional schemes.

6.1 Overview of the Recursive Multiplication Algorithm

6.1.1 Background Information

One of the pioneering schemes for "divide and conquer", or recursive, multiplication was proposed by Karatsuba and Ofman in 1962, and translated from Russian into English in 1963 [22]. The Karatsuba-Ofman Algorithm (KOA) gets the multiplication of two long integers by executing multiplications and additions on their divided parts. The KOA as described by Christof Paar [127] allows for a low complexity multiplier in Galois Fields. A field is an algebraic structure in which the operations of addition, subtraction, multiplication, and division (except by zero) can be performed while satisfying the standard rules. A Galois field is a finite field with p^n elements generated as the set of polynomials with coefficients in a modulo of an irreducible polynomial of degree n , and p is a prime integer [128].

The discussion of fields and the Karatsuba-Ofman Algorithm are beyond the scope of this thesis; however the fundamental principles of the KOA are used in the recursive algorithm presented by Danysh and Swartzlander [18]. Mathematically, the recursive algorithm may be proven by first considering two unsigned n -bit operands, the *multiplier* X and *multiplicand* A :

$$X = \sum_{k=0}^{n-1} x_k \cdot 2^k \quad A = \sum_{k=0}^{n-1} a_k \cdot 2^k$$

By dividing each of the two operands into 2 m -bit values, where $m = n/2$, we obtain:

$$X = \sum_{k=0}^{m-1} x_k \cdot 2^k + \sum_{k=m}^{2m-1} x_k \cdot 2^k$$

$$A = \sum_{k=0}^{m-1} a_k \cdot 2^k + \sum_{k=m}^{2m-1} a_k \cdot 2^k$$

X and A may now be redefined as:

$$X = X_L + X_H \quad A = A_L + A_H$$

where:

$$\begin{aligned} X_L &= \sum_{k=0}^{m-1} x_k \cdot 2^k & X_H &= \sum_{k=m}^{2m-1} x_k \cdot 2^k \\ A_L &= \sum_{k=0}^{m-1} a_k \cdot 2^k & A_H &= \sum_{k=m}^{2m-1} a_k \cdot 2^k \end{aligned}$$

The overall multiplication of A and X is given by:

$$\begin{aligned} P &= A \cdot X \\ &= (A_L + A_H) \cdot (X_L + X_H) \\ &= A_L \cdot X_L + A_L \cdot X_H + A_H \cdot X_L + A_H \cdot X_H \\ &= P_0 + P_1 + P_2 + P_3 \end{aligned}$$

Therefore, the overall multiplication may be reduced to four smaller multiplications, and this process may be repeated using even smaller multipliers for the base multipliers. In order to minimize the delay introduced by subdividing the process, the result of the sub-multipliers, or the intermediary products, will be kept in carry save form; hence only one final fast adder will be required to yield the final product.

Each of the 4 *n-bit* intermediary products in carry save format will occupy a given series of bit positions. By examining the results of the expanded multiplication outlined above, the following relationship for the positions may be deduced:

$$\begin{aligned} P_0 &\Rightarrow [0 : n - 1] \\ P_1 &\Rightarrow \left[\frac{n}{2} - 1 : \frac{3n}{2} - 1 \right] \\ P_2 &\Rightarrow \left[\frac{n}{2} - 1 : \frac{3n}{2} - 1 \right] \\ P_3 &\Rightarrow [n : 2n - 1] \end{aligned}$$

A dot diagram representation of the multiplication is outlined in Figure 6.1, and a schematic representation is provided in Figure 6.2. It becomes apparent that there will be

3 intermediary products that will overlap from bit $(n/2 - 1)$ to $(3n/2 - 1)$. Consequently, that leads to 6 bits that must be reduced to 2 to provide one final product in carry save form. A 6:2 reduction scheme has been proposed for the recursive multiplier [18][129], which introduces at most an equivalent delay of three full adders. The reduction circuit is formed by an interconnection of variations of the reduction sub-block depicted in Figure 6.3 .

Figure 6.1 Dot diagram of a single level recursive n -bit multiplication

Figure 6.2 A schematic of a single level recursive multiplier

6.2 6:2 Reduction Circuitry

The main function of the reduction circuit is to reduce the four results generated by the intermediary multipliers down to one value, in carry save format. The 6:2 reduction block is composed of a chain of full adders generating a two bit output value, along with inter-block carry signals that propagate laterally along the reduction sub-block array. Each reduction sub-blocks will take anywhere from two to six input bits, and generate a two bit output value. In addition, there are various inter-block carry signals that propagate laterally along the reduction sub-block array. Figure 6.3 outlines a typical 6:2 reduction sub-block as proposed in [18]. Similar to the 4:2 compressor, the offset nature of the carry signals negates carry propagation across the reduction macrocells, ensuring a maximum delay of 3 full adders for the complete process.

Figure 6.3 A standard 6:2 reduction macrocell composed of 3 stages of full adders

Kim and Swartzlander [129] introduced a set of enhanced reduction sub-blocks to be used where the reduction process takes in 2, 3, 4 or 5 bit inputs. The circuits as defined in the manuscript are presented in Figure 6.4. It should be noted that although the circuits are

not entirely efficient in their objective, they provide regularity in the reduction chain, and are capable of receiving and transmitting the carry signals without disrupting the chain.

In the architecture, two n -bit operands are bisected, resulting in 4 $n/2$ -bit sub-multiplications. The overall input to the reduction circuit arrives in a set of four n -bit values in carry save format as the output of the four intermediary multipliers. This may be more clearly defined if the dot diagram representing the overall process in Figure 6.1 on page 134 is re-analyzed. It may be intuitively observed that the first $n/2$ bits of the reduction circuit output may be obtained directly from the output of the multipliers. So the reduction circuitry will be required to accommodate $3n/2$ bits of the product.

The reduction pattern leads to the simple expression for the allocation of the reduction sub-blocks for an $n \times n$ bit multiplier as:

- Bits 0 through $n/2-1$ are obtained directly as a result of the inputs.
- Bits $n/2$ through $3n/2-1$ are obtained via 6-input reduction blocks
- Bits $3n/2$ through $2n-1$ are obtained using 2-input reduction blocks

A superior implementation of the reduction circuit than that proposed by the original authors has been developed. Observing the lateral carry propagation in the 2-input reduction block in Figure 6.4 on page 137, a carry signal has been grounded, and there is extensive use of Half Adder blocks. With some minor alteration, a simplified version of this block may be formed to accept the lateral carry signals from the 6-input block preceding it, and an omission of a carry signal for the subsequent block. With the exclusion of the lateral carry signal, the remainder of the 2-input reduction blocks may be replaced by an array of half-adders, without any carry propagation delay.

A modified version of the presented reduction scheme is outlined in Figure 6.5 on page 138. The reduction sub blocks having 6 inputs exist from bit position $(n/2 - 1)$ to $(3n/2 - 1)$. The transition to the 2-input sub-blocks is composed of a two input reduction cell, a full adder, and a series of half adders for the remaining bits.

Figure 6.4 6:2 Macrocells capable of receiving a variety of input bits
 (a) 5 input (b) 4 input (c) 3 input (d) 2 input

Figure 6.5 Novel 6:2 Reduction Block

The need for the Half-Adder blocks, may not be obvious at first glance, however if the Carry-out of the Full-adder immediately preceding the cells is taken into consideration, then there will be three bits in position $3n/2+1$. Consequently, the use of the Half Adder cells will shift a carry bit laterally down the chain, allowing for the final result to have at most two bits in each position. Furthermore, it should be noted that the final carry out signal of the last Half Adder is omitted since it would be mathematically impossible to obtain a bit in the 129th position of a 64-bit multiplication. This new 6:2 reduction configuration will be used for any further analysis of the recursive multiplication algorithm.

6.3 Analysis Of The Base Multiplier

To examine the optimal size of the base multiplier required, and consequently, the number of recursions, it is imperative to examine the relationship between base multiplier size and the associated delay. This analysis will begin with the delay calculation provided in [18]. Assuming that each Full-Adder has a gate delay of 3 (2 gate delays for an XOR and 1 for

other gates), 1 gate delay is required for the initial partial product generation. Assuming that $2[\log_2(b) - 1]$ stages are required for a Dadda Multiplier, then we will expect a delay of:

$$\begin{aligned} D_{Dadda} &= 3 \times 2[\log_2(b) - 1] + 1 \\ &= 6\log_2(b) - 5 \end{aligned}$$

If n is the number of bits of the overall multiplier, and b the bits of the base sub-multiplier, then the number of recursions required is:

$$\log_2(n/b) = \log_2(n) - \log_2(b)$$

Since each recursion will require a 6:2 reduction stage, corresponding to 9 gate delays, the delay relationship for the overall scheme using a b -bit base Dadda multiplier will be:

$$\begin{aligned} D_{recursive} &= 9\log_2(n/b) + 6\log_2(b) - 5 \\ &= 9\log_2(n) - 3\log_2(b) - 5 \end{aligned}$$

Similarly the overall delay may be calculated for a b -bit Array base multiplier, by defining the delay for an array multiplier being:

$$\begin{aligned} D_{Array} &= 3 \times (b - 1) + 1 \\ &= 3b - 2 \\ D_{recursive} &= 9\log_2(n/b) + 3b - 2 \end{aligned}$$

As per the analysis carried out in Section 3.2.2 on page 38, the simplification for the number of required reduction stages as defined in [18] is not the most accurate. Consequently, the above calculations will be re-examined using the more accurate representations developed in chapter 3.

To begin, the delay analysis will be considered as a function of the delay associated with one (3,2) counter. In this manner, any approximation associated with the standard gate delay is avoided, and as outlined in the analysis of full adders in chapter 4, this would introduce further inaccuracies. Thus, the simplified analysis for the delay of the recursive

multiplier using a Dadda base multiplier is reduced to a function of the number of reduction stages required in the b -bit base-multiplier:

$$D_{Dadda} = \left\lceil \log_{1.5} \left(\frac{3b}{5} \right) \right\rceil$$

and the number of recursions:

$$\log_2(n/b) = \log_2(n) - \log_2(b)$$

where each reduction stage has a delay of one full adder, and each recursion will have a delay of 3 full adders. Thus the overall relation for an n -bit multiplication is:

$$\begin{aligned} D_{\text{Recursive_D}} &= D_{Dadda} + 3 \cdot \log_2 \left(\frac{n}{b} \right) \\ \therefore D_{\text{Recursive_D}} &= \left\lceil \log_{1.5} \left(\frac{3b}{5} \right) \right\rceil + 3 \cdot \log_2 \left(\frac{n}{b} \right) \end{aligned}$$

The delay associated with a recursive multiplier having an Array base multiplier is somewhat less complicated, and is given as:

$$\begin{aligned} D_{\text{Recursive_A}} &= D_{\text{Array}} + 3 \cdot \log_2 \left(\frac{n}{b} \right) \\ \therefore D_{\text{Recursive_A}} &= (b - 1) + 3 \cdot \log_2 \left(\frac{n}{b} \right) \end{aligned}$$

These simplified representations of the various delays allow an approximation of the relative delay that may be expected from the various configurations of the recursive multiplier architecture. In order to identify the optimal solution, the various delays of this architecture have been analyzed for the combinations of base multiplier size, and base multiplier type. Figure 6.6 (a) depicts the relationship between delay and overall operand size for various sizes of Array base multipliers. Likewise, Figure 6.6 (b) illustrates the relationship between delay and overall operand size for various sizes of Dadda base multipliers. Equivalent Array and Dadda multipliers not using the recursive architecture have been provided for benchmark comparisons in each graph.

Figure 6.6 Delay associated with various base-multiplier sizes
(a) Array base multipliers (b) Dadda base multipliers

Figure 6.7 provides a three dimensional curve portraying the effects of base multiplier size on overall delay for various overall multiplier sizes using a Dadda base multiplier. From this figure, it becomes evident that the most efficient recursive multiplier corresponds to only one level of recursion; thus adding at most one full adder delay to that of a typical non-recursive Dadda tree.

Furthermore, from the analysis it is intuitive to conclude that the recursive architecture using column compression (or Dadda type) base multipliers demonstrate considerable performance gains over array base multipliers (Figure 6.6). It is apparent that the array base multipliers will exhibit similar delays only in the case where several recursion stages are employed, at which point the benefits of regularity of the array multipliers will be completely overshadowed by the 6:2 reduction stages.

Table 2.1 on page 143 summarizes an analytic comparison of the two base-multiplier styles (Dadda and Array) for a 64-bit recursive multiplier. the complete analysis chart is provided for further reference in Appendix D . The total cell count and interconnect for the two sub-multiplier schemes are presented graphically in Figure 6.8 on page 144. In addition, Figure 6.9 on page 144 depicts the percentage increase of the various base multiplier sizes for each scheme in both cell and interconnect segment count, over a typical non-recursive 64-bit multiplier.

From these results, it can be concluded that there are obvious limitations to the extent of the application of the concept of locally optimized arrays. There is a definitive trade-off in both size and performance once the recursions become excessively small. The increase in the number of interconnect segments also poses a secondary limitation to the utility of the recursive scheme once the base-multiplier becomes too small. As stated in chapter 5, the interconnects are in fact shorter as the multiplier size decreases, however the total number of nets may pose a problem. Since each level of recursion increases the number of required of multipliers required by a factor of four, the area, speed, and power trade-offs in using a significant number of small base-multipliers may in fact be disadvantageous to the overall design.

3D plot of n-bit multipliers vs. b-bit base Dadda multipliers

Table 2.1. Analytic comparison of “Dadda” vs. “Array” base multipliers of different widths for a 64-bit recursive multiplier

Base Multiplier Size		64	32	16	8	4
Number of Multipliers		1	4	16	64	256
Dadda	Total Cell Count	3906	3945	4037	4277	4981
	% Increase - Cell Count	0.00%	1.00%	3.35%	9.50%	27.52%
	Total Interconnections	7812	7825	7877	8085	8917
	% Increase - Interconnections	0.00%	0.17%	0.83%	3.49%	14.14%
	Overall Delay (FAs)	10	11	12	13	14
	% Increase - Delay	0.00%	10.00%	20.00%	30.00%	40.00%
Array	Total Cell Count	3969	4069	4277	4725	5749
	% Increase - Cell Count	0.00%	2.52%	7.76%	19.05%	44.85%
	Total Interconnections	7938	8073	8357	8981	10453
	% Increase - Interconnections	0.00%	1.70%	5.28%	13.14%	31.68%
	Overall Delay (FAs)	63	34	21	16	15
	% Increase - Delay	0.00%	-46.03%	-66.67%	-74.60%	-76.19%

Figure 6.8 Base multiplier comparisons for a 64-bit recursive multiplier
 (a) total cell count (b) total interconnect segment count

Figure 6.9 Percentage increase in cell count and interconnect count for various sizes of Array and Dadda base multipliers

Chapter 7

Reconfigurable Multiplier Architecture

The successful design of high-speed computational systems is often predicated on the realization of advanced arithmetic circuits in digital hardware. The notions of reconfigurable architectures have been regarded as a means of adapting the hardware to achieve optimal performance under various conditions. This implies a level of intelligence built into the device for physical modification in order to meet operating requirements. The principle advantage of such systems rests in the fact that hardware realizations of computing algorithms outperform their software alternatives.

The intent of a reconfigurable architecture is to provide a means by which the performance of arithmetic hardware may be enhanced according to the desired function. For example, many modern DSP chips offer variable precision [130][131], or fault tolerant arithmetic implemented using software [132]. The operation of these devices may be ameliorated if such functions were executed directly on hardware. Since multiplication is considered to be the dominant computation in most digital signal processing (DSP) algorithms [19], a reconfigurable multiplier architecture may prove to be a desirable augmentation to existing ALUs in the quest for maximizing performance.

7.1 Introduction to DSP Multiplication

Digital signal processing has become such an ingrained element of modern society, that its existence in our daily activities often goes unnoticed. It is present in one shape or form in almost all of aspects of contemporary life. The continued development of this field is an area of significant research. There are typically two facets to modern processors: the software and instruction set that manipulates the signals, and the dedicated hardware that physically carries out the given commands. The hardware dictates the limitations of the instruction set and how those instructions may be carried out, thus predicting the general characteristics and performance of the device.

As outlined in Figure 2.1 on page 13, multiplication is one of the most important operations in advanced signal processing, and the framework of the dedicated hardware multiplier has a severe impact on the overall system. In some specific signal processing applications, linear transforms with predefined, fixed coefficients are used (such as the DCT). In this situation, multiplier-less processing may be carried out using a combination of shift-add algorithms, look-up-tables, and/or distributed arithmetic may be employed [133] (Distributed arithmetic (DA) is a bit-serial operation that performs an inner (dot) product of a pair of vectors). Additionally, unconventional arithmetic techniques and number systems may be employed to take advantage of a particular attribute of the data being processed in order to improve the overall system performance; the double-base number system (DBNS) is an excellent example of this [30].

Special purpose processors offer limited programmability (such as the redefinition of local coefficients) allowing for increased processing speeds; thus presenting a trade-off between generality and lower power, area and/or speed. General purpose microprocessors, microcontrollers, and DSP chips do not often have the luxury of a priori knowledge of the incoming signals, and so must be flexible in their architecture to accommodate a variety of instructions and data patterns. For this reason, conventional arithmetic circuitry must be present. The increased fabrication densities in the past few decades have opened the door for the amalgamation of special purpose and general purpose architectures, such as the

TMS320C8x generation that contains several DSP cores, a RISC microcontroller and a floating point ALU [134]. Hitachi offers yet another example of a combined architecture with their DSP-enhanced microcontroller, the SH-DSP, which adds a 16-bit data path to the original microcontroller for increased functionality [132]. The two distinct data paths in this device rely on one processor core, and so concurrent functionality is not possible.

7.1.1 Computation Parallelism

Concurrent, or parallel, processing is an obvious solution to increased productivity within a device. By augmenting a data path with extra execution units, such as the use of two multipliers instead of just one, the amount of work performed per instruction cycle is significantly increased. Superscalar architectures issue and execute multiple instructions simultaneously, and are employed in high-performance processor designs, such as the Pentium and the PowerPC. The immense gains achievable through parallelism and the founding principles of superscalar and massively parallel architectures is thoroughly presented in [135].

Single Instruction Multiple Data (SIMD) techniques are one of the most common examples of parallel processing employed in general purpose processors, such as the MMX and AltiVec extensions on the Pentium and PowerPC processors. SIMD is useful in applications with high levels of parallelism where numeric operations may be improved on by dissecting a large data word into shorter ones, and operating on them in parallel using one single instruction. SIMD capabilities are also becoming common in high performance DSP chips [132], since these techniques can greatly increase the rate of computation for vector operations, image processing and solving partial differential equations [125].

7.1.2 Variable Data Width

An excellent example of the presence of variable data widths within modern architectures is the SH-DSP chip. This hybrid controller/DSP chip utilizes an extended data bus width to accommodate both operations. Since the hybrid architecture requires 16-bits for the microcontroller and 32-bits for the DSP data buses, the primary processing core must be

able to accommodate both operand widths. A more common example is the Pentium IV processor, where the instructions are SIMD operations that operate on 8, 16, 32 or 64-bit operands [79]. In addition, many modern DSP chips [130][131] offer variable precision arithmetic execution depending on the processor mode. As mentioned earlier SIMD operations may split words into smaller chunks for parallel operations. Some SIMD processors support multiple data widths (16-bit, 8-bit), examples of which include the Lucent DSP16xxx, ADI ADSP-2116x, ADI TigerSHARC [136].

The IEEE Floating Point Standard [6], outlines the requirement for the two standard word widths, namely single and double precision. It is necessary for most DSP floating point cores to be able to support both of the precision lengths (Figure 2.12 on page 28). The sign (*s*), exponent (*e*) and fraction or mantissa (*f*) bits total 32 and 64-bits for single and double precision respectively. The particulars of precision conversion, rounding schemes and other details of this standard are beyond the scope of the current discussion.

As is the case in the Pentium IV [137], the majority of processors achieve double and double extended format floating point multiplication through the iterated use of singleprecision multipliers. Though effective, the formulation of double-precision multiplication using iterative single precision operations is an inefficient compromise to variable precision arithmetic using dedicated hardware. The floating point co-processor outlined in [130], for example, uses a 59-bit double precision multiplier, and its partial product CSA tree is partially decoupled for single precision applications.

7.1.3 Fault Tolerance

No single event has provoked as much interest and debate in fault tolerant arithmetic than the Pentium Division bug [138]. The work of Thomas Nicely, in 1994, led to the discovery that one in every eight billion inputs to the division circuitry yielded inaccurate results. Though this "bug" appears as a design flaw of minor significance, due to its infrequent occurrence, nevertheless Intel was forced to re-issue the processor with the problem corrected. It is imperative for a scientist carrying out an analysis or

mathematically intensive simulation to be able to have a high level of confidence in the end result. The same requirement may be extended to any computation device, especially those carrying out critical processes.

Checking hardware functionality is currently a more practical approach to fault tolerance. Irrespective of high performance demands, the increasing complexity of physical devices, make embedded fault detection and correction units a commercially wise decision. Fault tolerant hardware may be employed to detect a fault, and either correct the situation if a transient fault has occurred, or send a critical error in the case of a permanent physical breakdown. The essence of fault tolerance lies in redundancy [139], whether time redundancy, where an operation is repeated in time, or hardware redundancy, where duplicate hardware carries out parallel versions of an operation. Several schemes will now be presented, including those targeted for fault tolerant multiplication [139-145].

A simple multiplication check, which may be used for basic error detection, is through the comparison of residues [140]. If, for example $A \times B = C$, then $(A \bmod r) \times (B \bmod r) = (AB \bmod r)$, for a random, relatively small value r . Thus if $AB \bmod r$ is equivalent to $C \bmod r$, the multiplication has been successful. Another unique approach to error detection limited solely array multipliers is presented in [141]. The authors present the bi-directional operation (BIDO) implementation of an array multiplier where the idle cells are re-used for the repeated calculation of the product. The normal and repeated operations are performed simultaneously by taking advantage of the time and space redundancy in the architecture with little hardware overhead and performance degradation.

One proposal for the correction of physical faults is the through the employment of a fault tolerant technique referred to as Reprocessing with Micro Delays (REMOD) [142]. This fault detection is based on the principle of node covering, in which the circuit is decomposed into an array of identical cells, each checked by a covering cell. This scheme allows for the detection of faulty cells, and the reconfiguration of the overall circuit to disengage that cell from the data path for future computation. This scheme offers

single fault detection and correction for a 32-bit Wallace tree multiplier with a 43% and 48% increase in time and area respectively.

Time redundancy for error correction simply refers to the repetition of an operation one or more times to detect and restore potential faults. One of the most frequently employed forms of redundant fault tolerance is known as Triple-Modular Redundancy (TMR). This highly reliable scheme is often used for critical applications and has been proposed as a viable choice for general deep submicron systems [143]. The TMR framework involves three copies of an original logic or arithmetic unit, and a majority voting circuitry, used to select the majority output amongst the three units.

REcomputing with Duplication With Comparison (REDWC) may be used to detect errors, or extended to REcomputing with Triplication With Voting (RETWV) for error correction. Duplication, though hardware intensive, offers one of the more infallible means of fault tolerance. A basic RETWV multiplier is outlined in Figure 7.1 on page 151 [144], where an $n \times n$ multiplication is carried out through three iterations of $n/3 \times n$ sub-multiplications. The three sub-multipliers featured carry out three identical versions of the multiplication with their results passed on to a majority voter circuit. By exploiting time redundancy through the iterative procedure, hardware overhead has been limited in this TMR application.

The concept of RETWV is further extended in [145] with the proposal of a time-shared triple modular redundancy with alternating logic scheme. The dilemma concerning the wrong generated output from the majority voter due to two or more of the redundant modules having stuck-at faults in the same position, may be overcome through this methodology. By monitoring cases where the majority voter does not receive identical inputs, and inverting the signals with potential stuck-at faults, 100% permanent fault detection is achievable. This comes at the expense of a 37% delay increase over a typical TMR scheme.

A separate issue from stuck at faults are common mode failures (CMF), which arise as a result of a flaw in the original design. Naturally three copies of the identical circuit having a CMF at the same position will yield results that will go undetected in a TMR scheme. To overcome this problem, the Fault Tolerance by Shifted and Rotated Operands (TOSHIRO) in TMR architectures has been proposed in [143]. The scheme proposes a standard execution in two of the units, and a shifted and rotated version of the same operand in the third unit. The result is then unrotated and shifted back prior to the majority voter.

Figure 7.1 Time redundant RETWV error correcting multiplier [144]

7.2 Reconfigurable Architectures

The principles of reconfigurable hardware was substantiated with the introduction of the first class of the Field Programmable Gate Array (FPGA). The ability to modify hardware via software control gave the end-user an alternative to using cumbersome software subroutines for executing operations on predetermined hardware resources. Programmable or reconfigurable architectures offer the benefits of:

- shorter design time, and faster time to market
- performance gains over software alternatives
- flexibility to make changes, catering to evolving standards
- field upgradability, and modification for prolonged time to obsolescence
- re-use of the same platform for a variety of applications, and variations of the same product
- conformation to targeted application for optimal performance, power and/or functionality

Although variable precision and reconfigurable architectures have been suggested in the past [126, 146-152], the focus has predominantly been on FPGA implementations [146-149]. Such techniques do indeed offer a considerable performance edge without the need for additional software overhead; however, dedicated application specific hardware offers the potential for increased savings in resources, power and latency. The design in [146] is flexible enough to accommodate most basic arithmetic operations especially targeted at multimedia applications. A reconfigurable multiplier is outlined in [147] that uses the Karatsuba Multiplication algorithm, which is similar in style to the recursive multiplier, but is employed in finite field mathematics.

The Reconfigurable Arithmetic Processor (RAP), intended for DSP applications, is presented in [150]. This is one of the first proposals for dedicated hardware reconfigurability, where the functionality of the hardware is dictated by the control criteria. This scheme combines the simplicity of transmitting and executing serial arithmetic, with the performance advantages of functional parallelism. By arranging a sequence of serial arithmetic cells, the RAP is capable of interconnecting the cells in

accordance with a given function, such that entire arithmetic formulas may be calculated without the intermediate results going off chip or to local memory. The RAP is improved upon in [151] by identifying several of the most dominant arithmetic operations for DSP applications, and devising a scheme to carry out these functions using efficient reconfiguration. A shared hardware architecture composed of several interacting modules, configured via control signals, is once again employed.

A reconfigurable fixed hardware parallel inner processor has been suggested [152]. The complexity associated with the multiple levels recommended, as a result of the small base multiplier sizes, is in direct contrast to the derivation of optimum delay for recursive structures presented in previous chapters. However the concepts presented in [147] and [152] correspond to the reconfigurable multiplier architecture that will be presented in the upcoming section.

7.3 Proposed Multiplier

Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects. Potential solutions are adaptive and self-correcting/self-repairing circuits, and the use of on-chip reconfigurability - SIA 2001 Technology Roadmap [9]

The proposed multiplication architecture envelopes the concepts of fault tolerant computing, low power design, and high throughput arithmetic into one design. The scheme utilizes a 2-bit control signal to select one of four modes of operation:

- Double Precision (64-bit) Multiplication (default)
- Single Precision (32-bit) Multiplication
- Dual Single Precision Multiplication (double throughput)
- Single Precision Fault Tolerant Multiplication through a Majority Voter

The recursive multiplier architecture with one level of recursion will be used as the foundation for the reconfigurable architecture. The advantage offered by the recursive multiplication scheme is the use of smaller multipliers to implement a larger operation, which is in direct compliance with the presented results in the previous chapters. This structure indirectly promotes the notion of locally optimized arrays through shorter local interconnects, and a more regular integration of the sub-components on a larger scale.

In addition to the basic multipliers, a series of 2:1 multiplexers, will be used to guide the signal flow through the device. Since all of the necessary components for each mode of operation are present in the design, there will be no reconfiguration time required. The device will be capable of switching between modes of operation in real time without the necessity to completely reconfigure the internal layout of a programmable device, as is the case with FPGA devices. A schematic representation of the reconfigurable multiplier is provided in Figure 7.2, in which the 4 sub-multipliers, the reduction circuitry, the voter and the final fast adder are clearly defined.

Figure 7.2 Outline of the reconfigurable multiplier

This architecture lends itself to four modes of operation, and thus requires a 2-bit control signal for selection. The signal and the corresponding modes of operation are summarized in Table 2.1 .

Table 2.1. Modes of Operation of the Reconfigurable Multiplier

Control Signal	Mode of Operation
0 0	<i>Default - Double precision</i>
0 1	<i>Single Precision</i>
1 0	<i>Single Precision with Fault Tolerance</i>
1 1	<i>Dual Single Precision</i>

7.3.1 Double Precision Mode

The default double precision mode is simply a recursive multiplier with one level of recursion. This mode of operation reaps the benefits of the recursive multiplier architecture, while bearing no delay penalties, and minimal hardware overhead. The majority voter circuitry is disengaged through the multiplexor array (Figure 7.3). The reconfigurable architecture may be of any size, with the restriction that the single precision mode must be exactly one half of the double precision mode. To satisfy the IEEE floating point guidelines, a double precision multiplier having 54-bit operands is suggested, with each of the base multipliers being 27-bits wide.

7.3.2 Single Precision Mode

Single precision mode uses gating techniques to shut down three of the base multipliers, effectively cancelling over 75% of the circuit, in addition to the reduction circuitry and the majority voter (Figure 7.4). The final fast adder is also partitioned in the reconfigurable architecture, such that the upper portion of the adder may be shut down in order to avoid spurious transitions, which consume unnecessary power. The effect of this mode of operation is similar to that of clock gating techniques employed in low power design [65-68], used to blackout idle portions of a circuit. Moreover, the overall latency now becomes that of the base multiplier, allowing faster operation in single precision mode than would be possible if the entire circuit was active.

The advantage of this scheme is that the single precision multiplication is carried out using a full single precision multiplier as opposed to shutting portions of a larger partial product reduction tree, as is proposed in other variable precision schemes [130]. In this manner, both single and double precision operations are carried out at maximum efficiency in terms of area, performance and power.

Figure 7.3 Default double precision mode

Figure 7.4 Single precision mode

7.3.3 Dual Single Precision Mode

If the input bus is configured to allow for two sets of operands occupying the low order and high order bits of the bus, then two operations may be carried out on these operands concurrently. This scenario is employed in existing architectures [130][132], especially those with high levels of parallelism as outlined earlier in this chapter. The reconfigurable architecture is ideal for such applications, where two of the base multipliers may operate in parallel on two different sets of operands, while the remaining two multipliers are inactive (Figure 7.5). This effectively doubles the system throughput, with a latency of a single precision multiplier.

Once again, with the fast adder partitioned into two identical halves, linked via a multiplexed carry signal, two single precision fast additions may be carried out in parallel. This configuration comes at little to no delay overhead in most carry look ahead (CLA) and carry skip addition schemes. The gating of signals into the idle multipliers, in addition to the 6:2 reduction and majority voter circuitries allows for power savings. The idle circuits are not entirely disconnected from the power supply in order to allow for rapid and accurate engagement into any other mode of operation.

Figure 7.5 Dual single precision mode

7.3.4 Single Precision Fault-tolerant Mode

Although there are numerous methods of implementing fault tolerance in digital systems, one of the most basic methods is through majority voting between three duplicate values, which is also referred to as RETWV. Since this scheme is composed of four identical base multipliers, three of those may be used in conjunction with an array of 64 XOR gates and 2:1 MUX cells, to form a simple single precision fault tolerant multiplier (Figure 7.6). This architecture lends itself to any variation of TMR fault detection and correction scheme. A RETWV method is employed due to its simplicity and limited hardware overhead, however the majority voter may in fact incorporate any of the other more sophisticated schemes outlined earlier in this chapter.

Figure 7.6 Single precision - fault tolerant mode

With the theoretical framework for the reconfigurable multiplier architecture in place, the next chapter will focus on the implementation and simulation details.

Chapter 8

Modeling and Simulation

For the proper assessment of the performance characteristics of the proposed reconfigurable architecture, a valid model must be created, and compared against a benchmark model representing the state-of-the-art. For this reason a 64-bit reconfigurable multiplier has been designed using TSMC 0.18 μm , 6 metal layer technology. Additionally, a standard 64-bit Booth-recoded Wallace tree multiplier, similar to that employed in many of today's high performance processors, such as the Pentium IV [137], has been developed as a benchmark for comparison purposes.

This chapter will begin with a description of the hierarchical design of the multiplier using the Verilog Hardware Description Language (HDL). The implementation, simulation and comparison of the of the design against the benchmark will be the subsequent topics of discussion

8.1 HDL Model

Verilog describes a digital design as a set of modules, which are the basic building blocks forming the complete system. This hierarchical design methodology is a fundamental concept in Verilog digital designs. The reconfigurable multiplier design

features four 32-bit Booth-Recoded Wallace-Tree base multipliers, a 48-bit 6:2 reduction block and two 64-bit carry-look-ahead adders. The overhead from the additional features are four arrays of 32 2:1 MUX cells, two arrays of 64 2:1 MUX cells, and a series of 64 XOR gates and 2:1 MUX cells for the majority voter. All of these individual modules are enveloped by the top level module which acts as a general input/output (I/O) interface for the multiplier, as outlined in Figure 8.1 (a). The top level module contains the clocked latching circuitry required for design synthesis (Figure 8.1 (b)), and does not effect the internal configuration of the multiplier itself.

(a)

(b)

Figure 8.1 (a) Top level module of the HDL model
(b) Top level module outlining the input output clocked-latching circuitry

The multiplier itself is partitioned into the major sections as outlined previously, and depicted graphically in Figure 8.2. The coded model is marginally more complicated than the initial model of the multiplier presented in the previous chapter. This is due to the placement of the multiplexer blocks required to direct the signal paths. Built in Synopsys module definitions for Booth-recoded Wallace tree multipliers and carry-look-ahead adders have been used to model and synthesize portions of the code, ensuring that the most efficient synthesized netlist is obtained.

The Verilog code that defines the various modules, and their interaction is provided in Appendix C . The code in itself is over 1000 lines long, and consists of 17 different modules. A complete breakdown of the hierarchical expansion of the overall reconfigurable architecture, compiled by the Synopsys Design Analyzer, confirms the proper framework of the design. The standard cell components and the gate level configuration of each element may also be referenced from this file.

Figure 8.2 An illustrated representation of the HDL model of the multiplier

8.2 Implementation and Layout

The multiplier has been implemented using the TSMC 0.18 micron CMOS process using standard cell libraries provided by the Canadian Microelectronics Corporation. Semi-custom design makes use of standard cell libraries for the fabrication of custom integrated circuits. Since the designer is limited to the library cells at his disposal, having fixed layout and orientation, the design may not be fully optimized. However, through the use of powerful CAD packages, capable of quickly and effectively compiling a digital design using standard cell libraries, the development time of such chips is far less. This design methodology is by and large the most economic and practical means by which a custom ASIC design can reach the marketplace within a "technology window".

Full custom VLSI design allows for an increased opportunity for performance improvement of a chip, since the placement, geometry, and virtually every aspect of a design on a transistor level is adjustable and amendable by the architect. The end result will usually demonstrate superior characteristics such as low power dissipation, high operating frequencies and/or reduced silicon area. Unfortunately, the development timeline is generally far too long for such design methodologies to produce an economically viable product.

The Cadence Design Suite, including the AreaPdp and Silicon Ensemble tools have been used for the layout placement and routing of the cells. The design has been developed to meet fabrication requirements, however due to the large size, and I/O bound nature of the end result, fabrication was not feasible. An I/O bound design refers to a design having substantially more input and output pad overhead than the actual silicon requirements for the integrated circuit core itself. This leads to inefficient fabrication due to the large masses of silicon that will be squandered. Figure 8.3 on page 163 provides a view of the placed and routed reconfigurable multiplier architecture. Due to the I/O bound nature of the design, the core of the multiplier, without input buffers or output drivers, is the only component that is implemented. Since the intent of this design is to be incorporated into a larger processor, the isolated IC implementation was not explored.

Figure 8.3 Layout view of the Reconfigurable Multiplier

8.3 Simulation Results

Table 8.1 summarizes the breakdown of the area, power and delay for the reconfigurable multiplier. The percentage contributions for each of the components of this architecture are presented in Figure 8.4 on page 164. The four base multipliers account for the majority of the area and power consumption. The final carry-look-ahead fast adders (two 64-bit adder halves forming the 128-bit adder), though minimal in terms of area, are responsible for a considerable portion of the overall latency. The complete simulation report and log files for the reconfigurable multiplier are provided in appendix E.

Table 8.1 Breakdown of the parameters for the Reconfigurable Multiplier

	Area (sq um)	Power (mW)	Delay (ns)
Overall	443322.75	753.01	9.00
RecMult	381895.59	613.38	8.74
Fast Adder	10944.59	19.78	3.75
Mux Reduction	9025.44	11.52	0.11
Reduction	28670.47	51.76	1.27
Mux Majority	6728.42	8.57	0.00
Majority	5724.29	10.98	0.00
Multipliers	320802.38	510.77	3.49
I/O Registers	61427.16	139.64	0.26

Figure 8.4 Breakdown of the Reconfigurable Multiplier
(a) Area (b) Power (c) Delay

In order to obtain a valid measure of the characteristics and the performance of the proposed architecture, it is compared to a 64-bit conventional multiplier. A Booth-recoded Wallace-tree implementation of a multiplier having a carry-look-ahead fast adder is compiled in Synopsys, and fully laid out, in a similar fashion to the reconfigurable multiplier. Table 8.2 provides the comparison statistics of the two architectures. The first implementation of the benchmark multiplier, named *Mult64d*, produced results that significantly favoured the reconfigurable multiplier. The new proposal outperformed the conventional design in practically every statistical category from overall latency, coupling capacitance, average net length, area, and silicon area utilization.

A second conventional multiplier (*Mult64e*) has been recompiled using more stringent design constraints for reaffirmation of the results. In this case, the conventional multiplier drastically reduced the overall performance margins, beating the proposed architecture in area and total wiring requirements, thus reducing the capacitive coupling effects. Although the reconfigurable multiplier was not compiled with such high effort settings, it managed to top both conventional implementations in area utilization, power, and delay. Furthermore the new design provides the lowest average net length.

The delay calculations were performed using the Cadence Pearl timing Analyzer tool. This tool assures confidence in the accuracy of the latency calculations, which incorporate the interconnect and layout effects of the two architectures. Pearl's analysis is path-oriented and reports longest paths and timing violations. To ensure accuracy, Pearl also takes the gate input slew rate and interconnect resistance and capacitance (which can be back-annotated) into account during delay calculations. Because Pearl is a static timing analyzer, it traces and analyzes all paths in the circuit. This thorough path analysis guarantees the reliability of the timing verification. A dynamic timing analyzer can only check paths exercised by the simulation vectors; the quality of its results therefore depend on the quality of the supplied simulation vectors. Appendix F provides the complete simulation reports and logs of the three test multipliers.

Table 8.2 Summary of the 64-bit multiplier implementations

	Reconfigurable Multiplier	Standard Mult64d	Standard Mult64e
Summary			
<i>Number of Components</i>	17158.00	13851.00	9882.00
<i>Number of Pins</i>	87986.00	70539.00	54738.00
<i>Number of Nets</i>	18908.00	15589.00	11630.00
<i>Average Number of Pins per Net</i>	4.65	4.52	4.71
<i>Area of Chip (square DBU)</i>	7.1333E+11	7.6971E+11	5.9581E+11
<i>Area Required by all Cells</i>	4.4332E+11	3.7372E+11	3.6037E+11
<i>Area Utilization (%)</i>	62.15	48.55	60.48
Wiring			
<i>Total segments in regular wiring</i>	147400.00	120407.00	110441.00
<i>Total segments in special wiring</i>	419.00	44.00	1140.00
<i>Total wirelength in regular wiring (um)</i>	1308570.68	1535832.92	1226989.02
<i>Total wirelength in special wiring (um)</i>	115260.48	16556.08	96835.80
<i>Average regular wiring net length (um)</i>	8.88	12.76	11.11
<i>Average special wiring net length (um)</i>	275.08	376.27	84.94
<i>METAL 1</i>	162348.96	70292.58	127593.76
<i>METAL 2</i>	278562.04	318176.90	232618.42
<i>METAL 3</i>	441000.74	510327.86	432153.96
<i>METAL 4</i>	334782.50	412856.72	314007.86
<i>METAL 5</i>	171144.40	181818.78	162043.82
<i>METAL 6</i>	35992.52	58916.16	55407.00
<i>TOTAL WIRING (um)</i>	1423831.16	1552389.00	1323824.82
<i>Average net length (um)</i>	75.30	99.58	113.83
<i>Max crosstalk induced timing delta (ns)</i>	1.34	2.22	1.19
Power			
<i>Cell Internal Power (mW)</i>	322.65	410.48	417.97
<i>Net Switching Power (mW)</i>	290.73	337.17	309.52
<i>Total Dynamic Power (mW)</i>	613.38	747.65	727.49
<i>Cell Leakage Power (uW)</i>	20.46	16.94	13.61
Timing			
<i>Data Arrival Time (ns)</i>	9.00	11.18	9.92
<i>I/O Delay (ns)</i>	0.26	1.10	2.23
<i>Multiplier (ns)</i>	4.99	4.88	4.37
<i>Fast Adder (ns)</i>	3.75	5.20	3.32

Table 8.3. Comparative summary of the 64-bit multipliers

	Reconfigurable Multiplier	Standard Average	Percentage Change
Summary			
<i>Number of Components</i>	17158.00	11866.50	30.84%
<i>Number of Pins</i>	87986.00	62638.50	28.81%
<i>Number of Nets</i>	18908.00	13609.50	28.02%
<i>Average Number of Pins per Net</i>	4.65	4.62	0.75%
<i>Area of Chip (square DBU)</i>	7.1333E+11	6.8276E+11	4.29%
<i>Area Required by all Cells</i>	4.4332E+11	3.6704E+11	17.21%
<i>Area Utilization (%)</i>	62.15	54.52	12.28%
Wiring			
<i>Total segments in regular wiring</i>	147400.00	115424.00	21.69%
<i>Total segments in special wiring</i>	419.00	592.00	-41.29%
<i>Total wirelength in regular wiring (um)</i>	1308570.68	1381410.97	-5.57%
<i>Total wirelength in special wiring (um)</i>	115260.48	56695.94	50.81%
<i>Average regular wiring net length (um)</i>	8.88	11.93	-34.41%
<i>Average special wiring net length (um)</i>	275.08	230.61	16.17%
<i>METAL 1</i>	162348.96	98943.17	39.06%
<i>METAL 2</i>	278562.04	275397.66	1.14%
<i>METAL 3</i>	441000.74	471240.91	-6.86%
<i>METAL 4</i>	334782.50	363432.29	-8.56%
<i>METAL 5</i>	171144.40	171931.30	-0.46%
<i>METAL 6</i>	35992.52	57161.58	-58.82%
<i>TOTAL WIRING (um)</i>	1423831.16	1438106.91	-1.00%
<i>Average net length (um)</i>	75.30	106.71	-41.70%
<i>Max crosstalk induced timing delta (ns)</i>	1.34	1.71	-27.24%
Power			
<i>Cell Internal Power (mW)</i>	322.65	414.23	-28.38%
<i>Net Switching Power (mW)</i>	290.73	323.35	-11.22%
<i>Total Dynamic Power (mW)</i>	613.38	737.57	-20.25%
<i>Cell Leakage Power (uW)</i>	20.46	15.27	25.33%
Timing			
<i>Data Arrival Time (ns)</i>	9.00	10.55	-17.22%

As previously mentioned, the implementation of the final fast adder makes a significant impact on the overall design. For this reason, the proposed reconfigurable implementation may be further improved upon by compiling and placing the final fast adder using the highest effort settings available on the CAD tools. This was the exact procedure followed in obtaining an improved benchmark multiplier for the previous comparative study.

The most effective comparison of the architecture would be through the analysis of two fully optimized custom layouts. Semi-custom standard cell implementations have an immense interdependence on the CAD tools used to compile, place and route the designs. The designer is thus at the mercy of the software algorithms used to implement the architecture. As was demonstrated in the study, a minor modification of the design constraints and the specification of a higher degree of effort in the initial compiling of the netlist can drastically alter the end-result. Although the most accurate reflection of the proposed architectures performance should be gauged against the initial layout (*Mult64d*), the overall results will use the average of the two Wallace multipliers as a benchmark. The results of the comparison have been summarized in Table 8.3 on page 167.

8.4 Design Highlights

The primary considerations which were taken into account while conducting the research and developing the proposed architecture have been the independence of :

- platform
- function
- application
- technology

Many of the “breakthrough” designs in the field of computer arithmetic, especially in digital multiplication designs, have come at the expense of heavy reliance on one of the listed factors above. A design which targets a particular function or application may take advantage of unconventional methodologies for the architecture. This is the case in filter designs. Number systems may be exploited for their various advantages in designs that target a particular platform without the need for conversion. Finally, analog based designs,

such as threshold logic, multivalued logic or neural network based arithmetic all suffer from technology dependence. A layout in one technology may not necessarily work in another.

The obvious desired attributes for the end-product would include one or more of the following characteristics:

- high-speed
- low-power
- fault tolerance
- reduced layout complexity (regularity)

The paradoxical desire to incorporate all of these desires into one design has been a significant challenge for circuit designers. The reconfigurable architecture attempts to embody all of these characteristics, while granting the end-user the ability to determine the criteria of importance. This concept places the final decision regarding the hardware configuration in the hands of the user, without the performance and area overheads associated with FPGAs or software routines.

Chapter 9

Conclusions

9.1 Summary of Contributions

The aim of this thesis has been to provide a thorough analysis of the state-of the-art in the area of digital multiplication. The investigations have focused on all aspects of the digital multiplier, on all levels of design abstraction. A summary of the contributions to the field of computer arithmetic will be provided in this section.

9.1.1 Algorithmic Contributions

The majority of the elegant algorithms used today in the field of computer arithmetic were developed many decades ago. Over the years, only marginal enhancements and suggestions have been provided. This thesis contributes to the algorithmic analysis, development and modeling of partial product reduction arrays.

A novel relationship between minimum full adder requirements according to column height has been introduced in Chapter 3. This scheme has been shown to more accurately reflect the number of reduction stages required according to the partial product matrix height, while maintaining algebraic simplicity. The new model substantially outperforms existing logarithmic estimates.

A second algorithmic assessment of partial product reduction arrays has been carried out using 4:2 compressors. A relationship between the partial product matrix size and compressor count and stage count has been developed. This scheme is employed in a new optimized compressor distribution scheme that effectively minimizes the number of compressor cells required in a reduction matrix, while maintaining the overall regularity of the matrix.

Finally, the various configurations of the recursive multiplier have been extensively analyzed using algebraic models of the base-multipliers. The use of tree multipliers over array multipliers has been assessed, and the various sizes of the base multipliers have been examined. It has been demonstrated that the most efficient recursive structure is composed of only one level of recursion using tree (or Dadda) base multipliers.

9.1.2 Architectural Contributions

In Chapter 5 the topic of wire interconnects was examined in some detail, along with the scaling issues associated with both wires and devices in general. A design methodology employing the “locally optimized array” framework is outlined for digital multipliers. The simulation results of various multiplier widths supports this framework, as well as the recursive multiplier architecture in general. It has been demonstrated that in order to overcome the proven repercussions of interconnects with continued device scaling, high-performance multipliers of the future will need to employ an alternate architecture, such as the one proposed.

The recursive multiplier architecture has been further explored, and a new reduction scheme is proposed which is smaller and more logically efficient than those proposed in the past. The single level recursive multiplier has then been enhanced with some gating multiplexers and a majority voter forming a novel reconfigurable multiplier architecture. This proposal has been implemented and simulated on the latest available CMOS process. Its performance and overall characteristics have been compared against a standard Booth-recoded Wallace multiplier of the same size.

9.1.3 Transistor Level Contributions

Although no new transistor level designs have been suggested, a framework for the selection, design and accurate simulation of digital circuits has been established. The various logic families, and transistor level arrangement have been critically analyzed for both full adders and 4:2 compressors. The various shortcomings of pass-transistor logic styles have been presented and the use of such design methodologies has been cautioned. Furthermore a suggestion has been made for the accurate simulation setup and analysis of arithmetic cells in terms of power consumption, area and delay.

9.2 Conclusions

By targeting many of the contemporary requirements for digital systems such as power, performance, regularity, and fault tolerance, a multiplication architecture that embodies these characteristics has been developed. This scheme takes full advantage of the benefits of reconfigurable architectures, while offering performance, power and area characteristics that are normally associated with custom digital designs.

This design has been implemented using the TSMC 0.18 CMOS standard cell libraries, and its performance has been compared against a typical multiplier architecture, namely a Booth-recoded Wallace Tree multiplier. The simulation results have demonstrated that, on average, the reconfigurable multiplier is extremely efficient in terms of interconnect layout. The new scheme offers over a 5% savings in regular interconnect requirements, resulting in a reduction in excess of 40% in average net lengths. This conservation in wire resources creates the opportunity for a 27% average reduction in coupling capacitance delay. Furthermore, this scheme promotes a refined approach to intercell wiring, promoting the locally optimized array paradigm. This is evident in the 39% increase in low-level, local wiring while eliminating over 58% of the global interconnects from the top-most metal layer.

The reconfigurable multiplier contains an average of 30% more components than the Wallace scheme, yet the overall dimensions are only increased by slightly over 4%. This is a result of the greater area utilization (12%) offered by this design. The small sacrifice in area overhead is more than compensated for by the 20% reduction in dynamic power consumption, and a 17% average decrease in overall latency.

The locally optimized array paradigm introduced in chapter 5, has demonstrated gains in multiplier architectures, offering a potential solution to the interconnect challenges faced by digital designs in deep-submicron technologies. Moreover, a variation of this concept has been implemented and analyzed against current high performance multipliers. The proposed architecture exhibits exceptional interconnect regularity, and promising overall characteristics while offering far greater versatility than any other multiplier architecture currently available.

REFERENCES

- [1] Charles Babbage Institute - Center for the History of Information Technology, <http://www.cbi.umn.edu>, 2000
- [2] Jones Telecommunications and Multimedia Encyclopedia, "Computers: History and Development", http://www.digitalcentury.com/encyclo/update/comp_hd.html, 1999
- [3] Michelle A. Hoyle, "The History of Computing Science", <http://www.eingang.org/Lecture/index.html>, 2002
- [4] C.S. Wallace, "A suggestion for a fast multiplier", IEEE Transactions on Electronic Computers, vol. 13, pp 14-17, 1964.
- [5] Luigi Dadda, "Some schemes for parallel multipliers", Alta Frequenza, vol. 45, pp. 574-580, 1966
- [6] IEEE Standard for Binary Floating-Point, IEEE Std. 754-1985
- [7] H.J.M. Veendrick, "Short-Circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits", IEEE Journal of Solid State Circuits, Volume: 19, pp. 468-473, August 1984
- [8] Behrooz Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press, New York, 2000
- [9] "International Technology Roadmap for Semiconductors", Semiconductor Industry Association, 2001
- [10] G.J Hekstra and R. Nouta, "A fast parallel multiplier architecture", IEEE International Symposium on Circuits and Systems, ISCAS '92, vol. 5 , pp. 2128 - 2131, 1992
- [11] Gwangwoo Choe and E.E. Swartzlander Jr., "Interconnection effects in fast multipliers", Asilomar Conference on Signals, Systems, and Computers, vol. 2 , pp. 1224-1227, 1999
- [12] H.A. Al-Twaijry, M.J. Flynn, "Technology scaling effects on multipliers", IEEE Transactions on Computers, Volume: 47 Issue: 11, pp 1201 -1215, Nov. 1998

-
- [13] Zhijun Huang and Milos Ercegovac, "Effect of wire delay on the design of prefix adders in deep-submicron technology", Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1713 -1717, 2000
 - [14] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, D. Burger, "Clock rate versus IPC: the end of the road for conventional microarchitectures", Proceedings of the 27th International Symposium on Computer Architecture, pp 248 -259, 2000
 - [15] R. Ho, K.W. Mai, M.A. Horowitz, "The future of wires", Proceedings of the IEEE , Volume: 89 Issue: 4 , pp 490 -504, April 2001
 - [16] "Current Technical Trends: Dual Damascene & Low-k Dielectrics", Jerry Healey, President - Threshold Systems, 2002.
 - [17] A. Moshovos, G.S. Sohi, "Microarchitectural innovations: boosting microprocessor performance beyond semiconductor technology scaling", Proceedings of the IEEE, Volume: 89 Issue: 11, pp. 1560-1575, Nov. 2001
 - [18] A.N. Danysh, and E.E. Swartzlander Jr., "A recursive fast multiplier", Asilomar Conference on Signals, Systems & Computers, vol. 1, pp. 197 -201, 1998
 - [19] Reto Zimmermann, "Computer Arithmetic: Principles, Architectures, and VLSI Design", Lecture notes for the Integrated Systems Laboratory, Swiss Federal Institute of Technology (ETH), 1999
 - [20] Gary. W. Bewick, "Fast Multiplication: Algorithms and Implementation", Dissertation for the Degree of Doctor in Philosophy, Stanford University, 1994
 - [21] Bickerstaff, Schulte and E.E. Swartzlander Jr., "Analysis of column compression multipliers", IEEE Symposium on Computer Arithmetic, pp. 33 -39, 2001
 - [22] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata", Sov. Phys.-Dokl. (English Translation), vol. 7, pp. 595-596, 1963
 - [23] A. Habibi and P.A. Wintz, "Fast Multipliers", IEEE Transactions on Computers, vol. C-19, pp. 153-157, 1970
 - [24] W. Stenzel, W. Kubitz, G. Garcia, "A Compact High-Speed Parallel Multiplication Scheme", IEEE Transactions on Computers, vol. 26, pp. 948-957, 1977
 - [25] Wang, Jullien and Miller, "A New Design Technique for Column Compression Multipliers", IEEE Transactions on Computers, vol. 44, pp. 962-970, August 1995
 - [26] R.V.K. Pillai, D. Al-Khalili and A.J. Al-Khalili, "Energy delay analysis of partial product reduction methods for parallel multiplier implementation", International Symposium on Low Power Electronics and Design, pp. 201 -204, 1996
-

-
- [27] B. Parhami, "Variations on multioperand addition for faster logarithmic-time tree multipliers", Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers, vol. 2, pp 899-903, 1997
 - [28] Ferguson and M.D. Ercegovac, "A multiplier with redundant operands", Asilomar Conference on Signals, Systems, and Computers, vol. 2, pp. 1322 -1326, 1999
 - [29] Freking and Parhi, "Montgomery modular multiplication and exponentiation in the residue number system", Asilomar Conference on Signals, Systems, and Computers, vol. 2, pp. 1312 -1316, 1999
 - [30] V.S. Dimitrov, G.A. Jullien, W.C. Miller, "Theory And Applications of the Double-Base Number System", IEEE Transactions on Computers, , vol. 48 pp. 1098 -1106, 1999
 - [31] Jullien, Dimitrov, Li, Miller, Lee, and M. Ahmadi, "A hybrid DBNS processor for DSP computation", IEEE International Symposium on Circuits and Systems, ISCAS '99. vol. 1 , pp. 5-8, 1999
 - [32] A. D. Booth, "A Signed Binary Multiplication Technique", Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, pp 236-240, June 1951
 - [33] O. L. MacSorley, "High-Speed Arithmetic in Binary Computers", Proceedings of the IRE, pp. 67-91, 1961.
 - [34] Millar, Madrid and E.E. Swartzlander Jr., "A fast hybrid multiplier combining Booth and Wallace/Dadda algorithms", Midwest Symposium on Circuits and Systems, vol. 1, pp. 158 -165,1992
 - [35] Daumas and David Matula, "A Booth multiplier accepting both a redundant or a non redundant input with no additional delay", IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 205-214, 2000
 - [36] D. Villeger and V.G. Oklobdzija, "Analysis of Booth encoding efficiency in parallel multipliers using compressors for reduction of partial products", Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 781 -784, 1993
 - [37] Pascal Bonatto and Vojin Oklobdzija, "Evaluation of Booth's algorithm for implementation in parallel multipliers", Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 608 -610, 1996
 - [38] "Synopsys Online Documentation", v2001.08, Synopsys Inc. 2001
 - [39] Oklobdzija, Villeger and Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", IEEE Transactions on Computers, vol. 45, pp. 294-306, March 1996
-

-
- [40] M. Davio, J.P. Deschamps, A. Thayse, "Digital Systems with Algorithm Implementation", John Wiley and Sons, 1983.
 - [41] P.J. Song, G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication" IEEE Journal of Solid-State Circuits, vol 26, pp. 1184-1198, 1991
 - [42] Paul F. Stelling, Charles U. Martel, Vojin G. Oklobdzija, Fellow and R. Ravi, "Optimal Circuits for Parallel Multipliers", IEEE Transactions on Computers, vol. 47, pp.273-285, March 1998
 - [43] Chi-Hsiang Yeh and B. Parhami, "Efficient designs for multi-input counters", Asilomar Conference on Signals, Systems, and Computers, vol.2, pp. 1340-1344, 1999
 - [44] Chi-Hsiang Yeh and B. Parhami, Y. Wang, "Designs of counters with near minimal counting/sampling period and hardware complexity", Asilomar Conference on Signals, Systems and Computers, vol. 2, pp.894-898 , 2000
 - [45] M. Mehta, V. Parmar, E. Swartzlander, "High-speed multiplier design using multi-input counter and compressor circuits", IEEE Symposium on Computer Arithmetic, pp. 43-50, 1991
 - [46] Kwon, Nowka and E.E. Swartzlander Jr., "A 16-bit x 16-bit MAC Design using fast 5:2 Compressors", IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 235 -243, 2000
 - [47] Rong Lin, "Non-binary (7, 2)-(9, 2) parallel counters for multiplier designs", International Conference on ASIC, pp. 543-546, 2001
 - [48] Rong Lin, "A regularly structured parallel multiplier with non-binary-logic counter circuits", Proceedings of the Second IEEE Asia Pacific Conference on ASICs, pp. 359-362, 2000
 - [49] Rong Lin, "Fast multiplier schemes using large parallel counters and shift switches", Fourth International Conference on High Performance Computing, pp. 302-308, Dec 1997
 - [50] Y. Leblebici, H. Ozdemir, A. Kepke, U. Cilingiroglu, "A compact high-speed (31,5) parallel counter circuit based on capacitive threshold-logic gates", IEEE Journal of Solid-State Circuits, Volume: 31 Issue: 8, pp. 1177-1183, Aug. 1996
 - [51] S. Vassiliadis, S. Cotofana, "7/2 counters and multiplication with threshold logic", Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers, Volume: 1, pp. 192-196, Nov. 1996

-
- [52] P. Celinski, J.F. Lopez, S. Al-Sarawi, D. Abbott, "Compact parallel (m,n) counters based on self-timed threshold logic", *Electronics Letters*, Volume: 38 Issue: 13, pp. 633-635, June 2002
- [53] C. Burwick, M. Thomas, J. Dienstuhl, K.F. Goser, "Threshold-gates in arithmetic circuits", *The 8th IEEE International Conference on Electronics, Circuits and Systems*, Volume: 2, pp. 909-912, 2001
- [54] P. Celinski, S. Al-Sarawi, D. Abbott, J.F. Lopez, "Low depth carry lookahead addition using charge recycling threshold logic", *IEEE International Symposium on Circuits and Systems*, Volume: 1, pp. 469-472, 2002
- [55] A. Weinberger, "4:2 Carry-Save Adder Module", *IBM Technical Disclosure Bulletin*, vol. 23, Jan. 1981
- [56] M. Hanawa, K. Kaneko, T. Kawashimo, H. Maruyama, "A 4.3 ns 0.3 μ m CMOS 54x54 b multiplier using precharged pass-transistor logic", *IEEE International Solid-State Circuits Conference*, pp. 364-365, Feb 1996
- [57] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakagome, "A 4.4 ns CMOS 54x54-b multiplier using pass-transistor multiplexer", *IEEE Journal of Solid-State Circuits*, Volume: 30 Issue: 3, pp. 251-257, Mar 1995
- [58] Hagihara, Inui, Yoshikawa, Nakazato, Iriki, Ikeda, Shibue, Inaba, Kagamihara, Yamashina, NEC Corp. "A 2.7ns 0.25 μ m CMOS 54x54b Multiplier", *IEEE Solid State Circuits Conference*, pp. 296-297, 1998
- [59] Jan Rabaey, "Exploring the Power Dimension", *IEEE Custom Integrated Circuits Conference*, pp. 512-220, 1996
- [60] Mehra, L. Guerra and Jan Rabaey, "A Partitioning Scheme for Optimizing Interconnect Power", *IEEE Journal of Solid State Circuits*, vol. 32, pp. 433-443, March 1997
- [61] Vemuru and Schienberg, "Short-Circuit Power Dissipation Estimation for CMOS Logic Gates", *IEEE Transactions on Circuits and Systems*, vol. 41, pp. 762-765, November 1994
- [62] Sung-Mo Kang and Yusuf Leblebici, "CMOS Digital Integrated Circuits Analysis and Design" Second Edition, McGraw-Hill Higher Education, 1998
- [63] S. Huang, C. Chen, M. Ahmadi, P. Mokrian, "An Optimal Variable Voltage Scheduling", *2002 Midwest Symposium on Circuits and Systems*, August 2002
- [64] Massoud Pedram, "Power Optimization and Management in Embedded Systems", *Asia and South Pacific Design Automation Conference*, pp. 239 -244, 2001
-

-
- [65] Q. Wu, X. Wu and M. Pedram, "Clock-Gating and its Application to Low-Power Design of Sequential Circuits", IEEE Transactions on Circuits and Systems, vol. 47, pp. 415-420, March 2000
- [66] Oh and M. Pedram, "Gated Clock Routing for Low-Power Microprocessor Design", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 20, pp. 715-722, June 2001
- [67] X. Wu and M. Pedram, "Low-Power CMOS Circuits with Clocked Power", IEEE Asia-Pacific Conference on Circuits and Systems, APCCAS 2000, pp. 513 -516, 2000
- [68] X. Wu and M. Pedram, "Low Power Sequential Circuit Design by Using Priority Encoding and Clock Gating", International Symposium on Low Power Electronics and Design, ISLPED '00, pp. 143 -148, 2000
- [69] Muhammad, Somasekhar and Roy, "Switching Characteristics of generalized Array Multiplier Architectures and their Applications to Low-Power Design", IEEE International Conference on Computer Design, (ICCD '99), pp. 230-235, 1999
- [70] Alexander Stokman, "Implementation of Threshold Logic", Master's Thesis, Delft University of Technology, Netherlands, Jan. 1998
- [71] A. Stokman, S.D. Cotofana, S. Vasiliadis, "A versatile threshold logic gate", International Semiconductor Conference, Volume: 1, pp. 163-166, Oct. 1998
- [72] H. Ozdemir, A. Kepke, B. Pamir, Y. Leblebici, U. Cilingiroglu, "A capacitive threshold-logic gate", IEEE Journal of Solid-State Circuits, Volume: 31 Issue: 8, pp. 1141-1150, Aug. 1996
- [73] A. Luck, S. Jung, R. Brederlow, R. Thewes, K. Goser, W. Weber, "On the design robustness of threshold logic gates using multi-input floating gate MOS transistors", IEEE Transactions on Electron Devices, Volume: 47 Issue: 6, pp. 1231-1240, June 2000
- [74] V. Beiu, "Ultra-fast noise immune CMOS threshold logic gates", Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, Volume: 3, pp. 1310-1313, 2000
- [75] M.J. Avedillo, J.M. Quintana, A. Rueda, E. Jimenez, "Low-power CMOS threshold-logic gate", Electronics Letters, Volume: 31 Issue: 25, pp. 2157-2159, Dec. 1995
- [76] M. Padure, S. Cotofana, C. Dan, M. Bodea, S. Vassiliadis, "A new latch-based threshold logic family", International Semiconductor Conference, Volume: 2, pp. 531-534, Oct. 2001
-

-
- [77] Zimmermann and Fichtner, "Low-Power Logic Styles: Cmos Versus Pass-Transistor Logic", IEEE Journal of Solid-State Circuits, vol. 32, pp.1079-1090, July 1997
 - [78] J.P. Uyemura, "CMOS Logic Circuit Design", Kluwer Academic Publishers, 1999
 - [79] Hinton et al., "A 0.18 μ m CMOS IA-32 Processor With a 4-GHz Integer Execution Unit", IEEE Journal of Solid State Circuits, vol. 36, pp. 1617-1627, November 2001
 - [80] V.G. Oklobdzija, M. Soderstrand, B. Duchene, "Development and synthesis method for pass-transistor logic family for high-speed and low power CMOS", Proceedings of the 38th Midwest Symposium on Circuits and Systems, Volume: 1, pp. 298-301, 1995
 - [81] Damu Radhakrishnan, "Low Power Arithmetic Units using Pass Logic", Proceedings from Conference/Exhibition on High Performance Computing, vol.1, pp. 332 -336, 2000
 - [82] Quintana, Avedillo, Jimenez, Rodriguez-Villegas, "Low-Power Logic Styles for Full Adder Circuits", IEEE International Conference on Electronics, Circuits and Systems, ICECS 2001, vol.3, pp. 1417 -1420, 2001
 - [83] B. Alhalabi, A. Al-Sheraidah, "A novel low power multiplexer-based full adder cell", IEEE International Conference on Electronics, Circuits and Systems, ICECS 2001, Volume: 3, pp. 1433-1436, 2001
 - [84] M. Alioto, G. Palumbo, "Analysis and comparison on full adder block in submicron technology", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 10 Issue: 6, pp. 806-823, December 2002
 - [85] Wang, Fang and Wu-Shiung Feng, "New Efficient Designs for XOR and XNOR Functions on the Transistor Level", IEEE Journal of Solid State Circuits, vol.29, pp.780-786, July 1994
 - [86] Nan Zhuang, Haomin Wu, "A new design of the CMOS full adder", IEEE Journal of Solid-State Circuits, Volume: 27, pp. 840-844, May 1992
 - [87] Radhakrishnan, "Low voltage CMOS full adder cells", Electronics Letters, vol. 35, pp.1792-1794, October 1999
 - [88] A.M. Shams, M.A. Bayoumi, "A framework for fair performance evaluation of 1-bit full adder cells", 42nd Midwest Symposium on Circuits and Systems, Volume: 1, pp. 6-9, 1999
 - [89] Hung Tien Bui, Yuke Wang, Yingtao Jiang, "Design and analysis of low-power 10-transistor full adders using novel XOR-XNOR gates", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Volume: 49 Issue: 1, pp. 25-30, 2002
-

-
- [90] A.M. Shams, T.K. Darwish, M.A. Bayoumi, "Performance analysis of low-power 1-bit CMOS full adder cells", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume: 10, pp. 20-29, Feb 2002
- [91] A.A. Fayed, M.A. Bayoumi, "A low power 10-transistor full adder cell for embedded architectures", *IEEE International Symposium Circuits and Systems*, Volume: 4, pp. 226-229, May 2001
- [92] K. Yano, Y. Sasaki, K. Rikino, K. Seki, "Top-down pass-transistor logic design", *IEEE Journal of Solid State Circuits*, Volume: 31, pp. 792-803, June 1996
- [93] Shams and Bayoumi, "A structured approach for designing low power adders" *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems & Computers*, vol. 1, pp. 757 -761, 1997
- [94] Fahim and Elmasry, "Low-Power High Performance Arithmetic Circuits and Architectures", *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 90-94, January 2002
- [95] Hung Tien Bui, A. Al-Sheraidah, Yuke Wang, "Design and analysis of 10-transistor full adders using novel XOR-XNOR gates", *International Conference on Signal Processing Proceedings, WCCC-ICSP 2000*, Volume: 1, pp. 619-622, 2000
- [96] I.S. Abu-Khater, A. Bellaouar, M.I. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers", *IEEE Journal of Solid-State Circuits*, Volume: 31 Issue: 10, pp. 1535-1546, Oct. 1996
- [97] Lu Junming, Shu Yan, Lin Zhenghui, Wang Ling, "A novel 10-transistor low-power high-speed full adder cell", *International Conference on Solid-State and Integrated-Circuit Technology*, Volume: 2, pp.1155-1158, 2001
- [98] Radhakrishnan and Preethy, "Low-Power CMOS Pass-Logic 4-2 Compressor for High-Speed Multiplication, *IEEE Midwest Symposium on Circuits and Systems*, pp. 1296-1298, August 2000
- [99] P. Kornerup, "Reviewing 4-to-2 adders for multi-operand addition", *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 218-229, 2002
- [100] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashimoto, H. Hayashida, K. Maeguchi, "A 10 ns 54x54 b parallel structured full array multiplier with 0.5 μm CMOS technology", *IEEE Journal of Solid-State Circuits*, Volume: 26 Issue: 4, pp. 600-606, Apr. 1991
- [101] Margala and Durdle, "Low-Power Low-Voltage 4-2 Compressors for VLSI Applications", *Proceedings of the Workshop on Low Power Design*, 1999
-

-
- [102] Shen-Fu Hsiao, Ming-Roun Jiang and Jia-Sien Yeh, "Design of high-speed low-power 3-2 counter and 4-2 compressor for fast multipliers", *Electronics Letters*, vol. 34 Issue: 4, pp. 341-343, February 1998
- [103] Law, Rofail and Yeo, "Low-Power circuit implementation for partial product addition using pass-transistor logic", *IEE Proceedings Circuits Devices Systems*, vol. 146, pp. 124-129 June 1999
- [104] K. Bernstein, K. Carrig, C. Durham, P. Hansen, D. Hogenmiller, E. Nowak, N. Rohrer, *IBM Microelectronics*, "High Speed CMOS Design Styles", Kluwer Academic Publishers, 1998
- [105] Dr. Graham Jullien, personal communication, University of Windsor, November 2002
- [106] Dr. James Haslett, personal communication, University of Calgary, December 2002
- [107] Dr. Wael Badawy, personal communication, University of Calgary, December 2002
- [108] Dr. Farid Najm, personal communication, University of Toronto, January 2003
- [109] Sung Mo Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits", *IEEE Journal of Solid State Circuits*, Volume: 21, pp. 889-891, October 1986
- [110] A.C. Cangellaris, "The interconnect bottleneck in multi-GHz processors; new opportunities for hybrid electrical/optical solutions", *Fifth International Conference on Massively Parallel Processing*, pp. 96-103, 1998
- [111] "International Technology Roadmap for Semiconductors", Semiconductor Industry Association, 1997
- [112] K. Banerjee, M. Pedram, A. Ajami, "Analysis and Optimization of Thermal Issues in High-Performance VLSI", *International Symposium on Physical Design*, pp. 230-237, 2001
- [113] C.P. Yuan and T.N. Trick, "A Simple Formula for the Estimation of the Capacitance of Two-Dimensional Interconnects in VLSI Circuits", *IEEE Electron Device Letters*, vol. 3, pp. 391-393, 1982
- [114] I. Jiang, S. Pan, Y. Chang, J. Jou, "Optimal Reliable Crosstalk-Driven Interconnect Optimization", *International Symposium on Physical Design*, pp. 128-133, 2000
- [115] A. Vittal, M. Marek-Sadowska, "Crosstalk reduction for VLSI", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Volume: 16 Issue: 3, pp. 290-298, March 1997
-

-
- [116] R. Kay, R. Rutenbar, "Wire Packing: A Strong Formulation of Crosstalk-Aware Chip-Level Track/Layer Assignment with an Efficient Integer Programming Solution", International Symposium on Physical Design, pp. 61-68, 2000
- [117] D. Sylvester, C. Hu, O.S. Nakagawa, S. Oh, "Interconnect scaling: signal integrity and performance in future high-speed CMOS designs", Symposium on VLSI Technology, pp. 42-43, 1998
- [118] R. McInerney, et al., "Methodology for Repeater Insertion Management in the RTL, Layout, Floorplan and Fullchip Timing Databases of the Itanium Microprocessor", International Symposium on Physical Design, pp. 99-104, 2000
- [119] "Low-K Dielectrics", International Sematech Annual Report, 2001
- [120] "Current Technical Trends: Dual Damascene & Low-k Dielectrics", Jerry Healey, President - Threshold Systems, 2002.
- [121] K. Banerjee, A. Mehrotra, "Inductance aware interconnect scaling", Proceedings. International Symposium on Quality Electronic Design, pp 43-47, 2002
- [122] K. Banerjee, A. Mehrotra, "Analysis of on-chip inductance effects for distributed RLC interconnects", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 21 Issue: 8, pp 904 -915, Aug. 2002
- [123] Magdy A. El-Moursy and Eby G. Friedman, " Optimizing Inductive Interconnect for Low Power", Proceedings of the The International Workshop on System on Chip for Real-time Applications, pp 206-216, July 2002
- [124] A. Deutsch et al., "When are transmission-line effects important for on-chip interconnections?", IEEE Transactions on Microwave Theory and Techniques, vol. 45, pp. 1836 -1846, 1997
- [125] Luigi Dadda, "The evolution of computer architectures", CompEuro '91, European Computer Conference on Advanced Computer Technology, Reliable Systems and Applications. pp. 9-16, 1991
- [126] K. Mai, T. Paaske, N. Jayasena, R. Ho, W.J. Dally, M. Horowitz, "Smart Memories: a modular reconfigurable architecture", Proceedings of the 27th International Symposium on Computer Architecture, pp. 161-171, 2000
- [127] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields", IEEE Transactions on Computers, vol. 45, pp. 856-861, 1996
- [128] Peter Cameron, "Encyclopaedia of Design Theory", <http://www.maths.qmul.ac.uk/~pjc/design/encyc/topics/>, 2002
-

-
- [129] J. Kim and E.E. Swartzlander Jr., "Improving the recursive multiplier", Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1320 -1324, 2000
- [130] R. Rogenmoser, L. O'Donnell, S. Nishimoto, "A dual-issue floating-point coprocessor with SIMD architecture and fast 3D functions", IEEE International Solid-State Circuits Conference, Volume: 1m pp. 414 -415, 2002
- [131] R.K. Kolagotla, J. Fridman, B.C. Aldrich, M.M. Hoffman, W.C. Anderson, M.S. Allen, D.B. Witt, R.R. Dunton, L.A. Booth Jr., "High performance dual-MAC DSP architecture", IEEE Signal Processing Magazine , Volume: 19 Issue: 4 , pp. 42 -53, July 2002
- [132] J. Eyre and J. Bier, "The evolution of DSP processors", IEEE Signal Processing Magazine, vol. 17 Issue: 2 , pp. 43-51, March 2000
- [133] M. Mehendale, S.D. Sherlekar, "VLSI Synthesis of DSP Kernels", Kluwer Academic Publishers, Boston, 2001
- [134] G.A. Jullien, "VLSI Digital Signal Processing: Some Arithmetic Issues", Keynote at SPIE Conference on Advanced Signal Processing, August 1996
- [135] J.E. Smith, G.S. Sohi, "The microarchitecture of superscalar processors", Proceedings of the IEEE , Volume: 83 Issue: 12, pp. 1609-1624, Dec 1995
- [136] "Understanding the New DSP Processor Architectures", Workshop #111, Berkeley Design Technology, Inc. , 2000
- [137] R. Kaivola, N. Narasimhan, "Formal Verification of the Pentium 4 Floating-Point Multiplier", Design, Automation and Test in Europe Conference and Exhibition, pp. 20-27, 2002
- [138] Blum and Wasserman, "Reflections on the Pentium Division Bug", IEEE Transactions on Computers, vol. 45, pp. 382-393, April 1996
- [139] T. H. Chen and L.G. Chen, "Fault-tolerant serial-parallel multiplier", IEE Proceedings on Computers and Digital Techniques, vol. 138 Issue: 4, pp. 276 -280, July 1991
- [140] D. Radhakrishnan and Preethy, "A Novel 36-Bit Single Fault Tolerant Multiplier Using 5-Bit Moduli", IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control, vol. 1, pp. 128 -130, 1998
- [141] Chen, Lee and Chen, "Concurrent Error Detection in Array Multipliers by BIDO", IEE Proceedings on Computers and Digital Techniques, vol. 142, pp. 425 -430, November 1995
-

-
- [142] Dutt and Hanchek, "REMOD: A New Methodology for Designing Fault-Tolerant Arithmetic Circuits", IEEE Transactions on VLSI Systems, vol. 5, pp. 34-56, March 1997
- [143] Alderighi, D'Angelo, Metra and Sechi, "Achieving fault-tolerance by shifted and rotated operands in TMR non-diverse ALUs", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 155-163, 2000
- [144] Hsu and E.E. Swartzlander Jr., "Time redundant error correcting adders and multipliers", IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 247-256, 1992
- [145] Ngai, He and E.E. Swartzlander Jr., "Enhanced Concurrent Error Correcting Arithmetic Unit Design Using Alternating Logic", IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 78-83, 2001
- [146] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, Brad Hutchings, "A reconfigurable arithmetic array for multimedia applications", Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays, pp. 135-143, 1999
- [147] Michael Jung, Felix Madlener, Markus Ernst and Sorin A. Huss, "A Reconfigurable Coprocessor for Finite Field Multiplication in $GF(2^n)$ ", Proceedings of the IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip, April 2002
- [148] Tencu and M.D. Ercegovac, "A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures", IEEE Symposium on FPGAs for Custom Computing Machines, pp. 216-225, 1998
- [149] N.L. Miller, S.F. Quigley, "A reconfigurable integrated circuit for high performance computer arithmetic", Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, Volume: 2, pp. 161-164, June 1998
- [150] S. Fiske, W.J. Dally, "The reconfigurable arithmetic processor", IEEE International Symposium on Computer Architecture, pp. 30-36, 1988
- [151] A. Rajagopal, B. Kuttanna, B. Janakiraman, R. Mukherjee, J. Shetler, "A Reconfigurable Arithmetic Processor", The Sixth International Conference on VLSI Design, pp. 172-175, Jan 1993
- [152] Rong Lin, "Reconfigurable parallel inner product processor architectures", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 9 Issue: 2, pp. 261-272, April 2001
-

Appendix A

Complete 4:2 Compressor Analysis

k	Operand Size (bits)	Theoretical Bound on [4:2] Cells				Optimized [4:2] Strategy			Improvement over Theoretical Bound		Analysis of Interconnects		Stage Count and Delay			
		Number of (3:2) cells required $NFA = (k-1)(k-2)$	Even Bound on [4:2] cells $QE = ((k-2)/2)(k+1)$	Odd Bound on [4:2] cells $QO = k*((k-1)/2)$	Number of Compressor Cells $QC = QE(\text{if even}) + QO(\text{if odd})$	Total cell count in Optimized Strategy	[4:2] cell count in Optimized Strategy	adder cell count in Optimized Strategy	Percentage Decrease in total cell count	Percentage Decrease in [4:2] cell count	Interconnect count for (3:2) cells $IFA = 2*NFA + k^2$	Interconnect count for [4:2] cells $IC = 3*NC + NCA + k^2$	Number of Stages Required for (3:2) Trees	Number of Stages Required for [4:2] Trees	Total Delay for (3:2) Reduction where $1FA = 2\text{-XOR delay}$	Total Delay for [4:2] Reduction where $1C = 3\text{-XOR delay}$
	NFA	QE	QO	QC	NCT	NC	NCA	DT	DC		IFA	IC	SFA	SC	DFA	DC
3	2	2	2	3	2	0	2	33.33%	100.00%		13	13	1	1	2	3
4	6	5	6	5	4	2	2	20.00%	60.00%		28	26	2	1	4	3
5	12	9	10	10	8	4	4	20.00%	60.00%		49	45	3	2	6	6
6	20	14	15	14	12	8	4	14.29%	42.86%		76	68	3	2	6	6
7	30	20	21	21	18	12	6	14.29%	42.86%		109	97	4	2	8	6
8	42	27	28	27	24	18	6	11.11%	33.33%		148	130	4	2	8	6
9	56	35	36	36	32	24	8	11.11%	33.33%		193	169	4	3	8	9
10	72	44	45	44	40	32	8	9.09%	27.27%		244	212	5	3	10	9
11	90	54	55	55	50	40	10	9.09%	27.27%		301	261	5	3	10	9
12	110	65	66	65	60	50	10	7.69%	23.08%		364	314	5	3	10	9
13	132	77	78	78	72	60	12	7.69%	23.08%		433	373	5	3	10	9
14	156	90	91	90	84	72	12	6.67%	20.00%		508	436	6	3	12	9
15	182	104	105	105	98	84	14	6.67%	20.00%		589	505	6	3	12	9
16	210	119	120	119	112	98	14	5.88%	17.65%		676	578	6	3	12	9
17	240	135	136	136	128	112	16	5.88%	17.65%		769	657	6	4	12	12
18	272	152	153	152	144	128	16	5.26%	15.79%		868	740	6	4	12	12
19	306	170	171	171	162	144	18	5.26%	15.79%		973	829	6	4	12	12
20	342	189	190	189	180	162	18	4.76%	14.29%		1084	922	7	4	14	12
21	380	209	210	210	200	180	20	4.76%	14.29%		1201	1021	7	4	14	12
22	420	230	231	230	220	200	20	4.35%	13.04%		1324	1124	7	4	14	12
23	462	252	253	253	242	220	22	4.35%	13.04%		1453	1233	7	4	14	12
24	506	275	276	275	264	242	22	4.00%	12.00%		1588	1346	7	4	14	12
25	552	299	300	300	288	264	24	4.00%	12.00%		1729	1465	7	4	14	12
26	600	324	325	324	312	288	24	3.70%	11.11%		1876	1588	7	4	14	12
27	650	350	351	351	338	312	26	3.70%	11.11%		2029	1717	7	4	14	12
28	702	377	378	377	364	338	26	3.45%	10.34%		2188	1850	7	4	14	12
29	756	405	406	406	392	364	28	3.45%	10.34%		2353	1989	8	4	16	12
30	812	434	435	434	420	392	28	3.23%	9.68%		2524	2132	8	4	16	12
31	870	464	465	465	450	420	30	3.23%	9.68%		2701	2281	8	4	16	12
32	930	495	496	495	480	450	30	3.03%	9.09%		2884	2434	8	4	16	12
33	992	527	528	528	512	480	32	3.03%	9.09%		3073	2593	8	5	16	15
34	1056	560	561	560	544	512	32	2.86%	8.57%		3268	2756	8	5	16	15
35	1122	594	595	595	578	544	34	2.86%	8.57%		3469	2925	8	5	16	15
36	1190	629	630	629	612	578	34	2.70%	8.11%		3676	3098	8	5	16	15
37	1260	665	666	666	648	612	36	2.70%	8.11%		3889	3277	8	5	16	15
38	1332	702	703	702	684	648	36	2.56%	7.69%		4108	3460	8	5	16	15
39	1406	740	741	741	722	684	38	2.56%	7.69%		4333	3649	8	5	16	15
40	1482	779	780	779	762	722	38	2.44%	7.32%		4564	3842	8	5	16	15
41	1560	819	820	820	800	760	40	2.44%	7.32%		4801	4041	8	5	16	15
42	1640	860	861	860	840	800	40	2.33%	6.98%		5044	4244	8	5	16	15
43	1722	902	903	903	882	840	42	2.33%	6.98%		5293	4453	9	5	18	15
44	1806	945	946	945	924	882	42	2.22%	6.67%		5548	4666	9	5	18	15
45	1892	989	990	990	968	924	44	2.22%	6.67%		5809	4885	9	5	18	15

		Theoretical Bound on [4:2] Cells				Optimized [4:2] Strategy			Improvement over Theoretical Bound		Analysis of Interconnects		Stage Count and Delay			
k	Operand Size (bits)	Number of (3:2) cells required $NFA = (k-1)(k-2)$	Even Bound on [4:2] cells $QE = ((k-2)/2)(k+1)$	Odd Bound on [4:2] cells $QO = k \lceil (k-1)/2 \rceil$	Number of Compressor Cells $QC = QE$ (if even) + QO (if odd)	Total cell count in Optimized Strategy	[4:2] cell count in Optimized Strategy	adder cell count in Optimized Strategy	Percentage Decrease in total cell count	Percentage Decrease in [4:2] cell count	Interconnect count for (3:2) cells $IFA = 2 \cdot NFA + k^2$	Interconnect count for [4:2] cells $IC = 3 \cdot NC + NCA + k^2$	Number of Stages Required for (3:2) Trees	Number of Stages Required for [4:2] Trees	Total Delay for (3:2) Reduction where 1 FA = 2-XOR delay	Total Delay for [4:2] Reduction where 1 C = 3-XOR delay
	NFA	QE	QO	QC	NCT	NC	NCA	DT	DC	IFA	IC	SFA	SC	DFA	DC	
46	1980	1034	1035	1034	1012	968	44	2.13%	6.38%	6076	5108	9	5	18	15	
47	2070	1080	1081	1081	1058	1012	46	2.13%	6.38%	6349	5337	9	5	18	15	
48	2162	1127	1128	1127	1104	1058	46	2.04%	6.12%	6628	5570	9	5	18	15	
49	2256	1175	1176	1176	1152	1104	48	2.04%	6.12%	6913	5809	9	5	18	15	
50	2352	1224	1225	1224	1200	1152	48	1.96%	5.88%	7204	6052	9	5	18	15	
51	2450	1274	1275	1275	1250	1200	50	1.96%	5.88%	7501	6301	9	5	18	15	
52	2550	1325	1326	1325	1300	1250	50	1.89%	5.66%	7804	6554	9	5	18	15	
53	2652	1377	1378	1378	1352	1300	52	1.89%	5.66%	8113	6813	9	5	18	15	
54	2756	1430	1431	1430	1404	1352	52	1.82%	5.45%	8428	7076	9	5	18	15	
55	2862	1484	1485	1485	1458	1404	54	1.82%	5.45%	8749	7345	9	5	18	15	
56	2970	1539	1540	1539	1512	1458	54	1.75%	5.26%	9076	7618	9	5	18	15	
57	3080	1595	1596	1596	1568	1512	56	1.75%	5.26%	9409	7897	9	5	18	15	
58	3192	1652	1653	1652	1624	1568	58	1.69%	5.08%	9748	8180	9	5	18	15	
59	3306	1710	1711	1711	1682	1624	58	1.69%	5.08%	10093	8469	9	5	18	15	
60	3422	1769	1770	1769	1740	1682	58	1.64%	4.92%	10444	8762	9	5	18	15	
61	3540	1829	1830	1830	1800	1740	60	1.64%	4.92%	10801	9061	9	5	18	15	
62	3660	1890	1891	1890	1860	1800	60	1.59%	4.76%	11164	9364	9	5	18	15	
63	3782	1952	1953	1953	1922	1860	62	1.59%	4.76%	11533	9673	9	5	18	15	
64	3906	2015	2016	2015	1984	1922	62	1.54%	4.62%	11908	9986	10	5	20	15	
65	4032	2079	2080	2080	2048	1984	64	1.54%	4.62%	12289	10305	10	6	20	18	
66	4160	2144	2145	2144	2112	2048	64	1.49%	4.48%	12676	10628	10	6	20	18	
67	4290	2210	2211	2211	2178	2112	66	1.49%	4.48%	13069	10957	10	6	20	18	
68	4422	2277	2278	2277	2244	2178	66	1.45%	4.35%	13468	11290	10	6	20	18	
69	4556	2345	2346	2346	2312	2244	68	1.45%	4.35%	13873	11629	10	6	20	18	
70	4692	2414	2415	2414	2380	2312	68	1.41%	4.23%	14284	11972	10	6	20	18	
71	4830	2484	2485	2485	2450	2380	70	1.41%	4.23%	14701	12321	10	6	20	18	
72	4970	2555	2556	2555	2520	2450	70	1.37%	4.11%	15124	12674	10	6	20	18	
73	5112	2627	2628	2628	2592	2520	72	1.37%	4.11%	15553	13033	10	6	20	18	
74	5256	2700	2701	2700	2664	2592	72	1.33%	4.00%	15988	13396	10	6	20	18	
75	5402	2774	2775	2775	2738	2664	74	1.33%	4.00%	16429	13765	10	6	20	18	
76	5550	2849	2850	2849	2812	2738	74	1.30%	3.90%	16876	14138	10	6	20	18	
77	5700	2925	2926	2926	2888	2812	76	1.30%	3.90%	17329	14517	10	6	20	18	
78	5852	3002	3003	3002	2964	2888	76	1.27%	3.80%	17788	14900	10	6	20	18	
79	6006	3080	3081	3081	3042	2964	78	1.27%	3.80%	18253	15289	10	6	20	18	
80	6162	3159	3160	3159	3120	3042	78	1.23%	3.70%	18724	15682	10	6	20	18	
81	6320	3239	3240	3240	3200	3120	80	1.23%	3.70%	19201	16081	10	6	20	18	
82	6480	3320	3321	3320	3280	3200	80	1.20%	3.61%	19684	16484	10	6	20	18	
83	6642	3402	3403	3403	3362	3280	82	1.20%	3.61%	20173	16893	10	6	20	18	
84	6806	3485	3486	3485	3444	3362	82	1.18%	3.53%	20668	17306	10	6	20	18	
85	6972	3569	3570	3570	3528	3444	84	1.18%	3.53%	21169	17725	10	6	20	18	
86	7140	3654	3655	3654	3612	3528	84	1.15%	3.45%	21676	18148	10	6	20	18	
87	7310	3740	3741	3741	3698	3612	86	1.15%	3.45%	22189	18577	10	6	20	18	
88	7482	3827	3828	3827	3784	3698	86	1.12%	3.37%	22708	19010	10	6	20	18	
89	7656	3915	3916	3916	3872	3784	88	1.12%	3.37%	23233	19449	10	6	20	18	
90	7832	4004	4005	4004	3960	3872	88	1.10%	3.30%	23764	19892	10	6	20	18	
91	8010	4094	4095	4095	4050	3960	90	1.10%	3.30%	24301	20341	10	6	20	18	
92	8190	4185	4186	4185	4140	4050	90	1.08%	3.23%	24844	20794	10	6	20	18	
93	8372	4277	4278	4278	4232	4140	92	1.08%	3.23%	25393	21253	10	6	20	18	
94	8556	4370	4371	4370	4324	4232	92	1.05%	3.16%	25948	21716	10	6	20	18	
95	8742	4464	4465	4465	4418	4324	94	1.05%	3.16%	26509	22185	11	6	22	18	
96	8930	4559	4560	4559	4512	4418	94	1.03%	3.09%	27076	22658	11	6	22	18	
97	9120	4655	4656	4656	4608	4512	96	1.03%	3.09%	27649	23137	11	6	22	18	
98	9312	4752	4753	4752	4704	4608	96	1.01%	3.03%	28228	23620	11	6	22	18	
99	9506	4850	4851	4851	4802	4704	98	1.01%	3.03%	28813	24109	11	6	22	18	
100	9702	4949	4950	4949	4900	4802	98	0.99%	2.97%	29404	24602	11	6	22	18	

Appendix B

Interconnect Analysis of Various Multiplier Sizes

	64	54	32	16
Summary				
Number of Components	12916	9391	3687	1069
Number of Pins	64481	56505	17559	5036
Number of Nets	14409	10388	3915	1140
Average Number of Pins per Net	4.48	4.48	4.49	4.42
Area of Chip (square DBU)	6.52266E+11	5.06873E+11	2.51071E+11	1.16752E+11
Area Required by all Cells	3.99982E+11	2.9885E+11	1.28184E+11	46465742400
Area Utilization (%)	61.32	58.96	51.05	39.8
Wiring				
Total segments in regular wiring	128059	82116	27057	6834
Total segments in special wiring	401	338	224	140
Total wirelength in regular wiring (um)	1512966.26	861731.66	230623.16	49271
Total wirelength in special wiring (um)	105351.24	78250.08	36160.6	15203.28
Average regular wiring net length (um)	11.81460311	10.4940774	8.523604243	7.20968686
Average special wiring net length (um)	262.7212968	231.5091124	161.43125	108.5948571
METAL 1	146231.92	102118.66	44620.96	17036.44
METAL 2	273265.32	185962.3	69145.3	17575.84
METAL 3	475396.2	294621.48	96337.8	25701.42
METAL 4	388446.9	211710	40386.22	3398.28
METAL 5	194880.84	111835.58	16193	762.3
METAL 6	140096.32	33433.72	100.8	0
TOTAL WIRING (um)	1618317.5	939981.74	266783.76	64474.28
Average net length (um)	112.3129641	90.487268	68.144	56.55638596
Max crosstalk induced timing delta (ns)	1.66	1.66	0.467	0.497
Synopsys				
Power				
Cell Internal Power (mW)	41.6141	32.1372	15.3479	6.1922
Net Switching Power (mW)	312.7236	259.548	149.541	70.0727
Total Dynamic Power (mW)	354.3378	291.6852	164.8889	76.265
Cell Leakage Power (uW)	26.5031	20.1685	9.298	3.7045
Timing				
Data Arrival Time (ns)	7.37	7.34	6.44	5
SE				
Total Power (20% toggle rate) (mW)	435.617	359.265	206.188	101.215

64-Bit Multiplier:

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 0:12:55, 3 April 2003

Design name: SPIE64

Report file name: SPIE64_apr.summary

Number of macros: 402

Number of components: 12916

Number of pins: 64481

Number of regular pins: 39776

Number of special pins: 24352

Number of unused pins: 353

Number of nets: 14409

Average number of pins per net: 4.48

Number of subnets: 737

Number of regular pins for subnets: 385

Number of special pins for subnets: 0

Number of virtual pins for subnets: 1089

Average number of pins per subnet: 2.00

Number of routing tracks available: 2666

Number of GCELLS per layer: 17835

**** UTILIZATION OF ALL ROW TYPES**

Type	Number	Length	Area	%_Row_Space
core Rows	114	80582040	496385366400	
core Cells	12176	64932120	399981859200	80.58

Area of chip: 652266014400 (square DBU)

Area required for all cells: 399981859200 (square DBU)

Area utilization of all cells: 61.32%

*******SILICON_ENSEMBLE WIRING REPORT*******

Time: 0:12:59, 3 April 2003

Design name: SPIE64

Report file name: SPIE64_apr.wires

Total vias in regular wiring: 120310

Total segments in regular wiring: 128059

Total vias in special wiring: 1190

Total segments in special wiring: 401

LAYER name: metall

Total wire length: 146231.92 microns

Length of regular wires: 56000.68 microns

Length of special wires: 90231.24 microns

LAYER name: metal2

Total wire length: 273265.32 microns

Length of regular wires: 258145.32 microns

Length of special wires: 15120.00 microns

LAYER name: metal3

Total wire length: 475396.20 microns

Length of regular wires: 475396.20 microns

Length of special wires: .00 microns

LAYER name: metal4

Total wire length: 388446.90 microns

Length of regular wires: 388446.90 microns

Length of special wires: .00 microns

LAYER name: metal5

Total wire length: 194880.84 microns

Length of regular wires: 194880.84 microns

Length of special wires: .00 microns

LAYER name: metal6

Total wire length: 140096.32 microns

Length of regular wires: 140096.32 microns

Length of special wires: .00 microns

Total wirelength in regular wiring: 1512966.26 microns

Total wirelength in special wiring: 105351.24 microns

Total wirelength in regular+special wiring: 1618317.50 microns

CROSSTALK:

0 nets claimed N coupling caps, but had different no.

28 nets had more coupling than total capacitance

0 pin lists with fewer pins than they said they had.

0 nets with bizarre cap/unit length

28 nets had implausible coupling ratios.

2 wires had implausibly small Rs.

131 rising drive and 131 falling drive implausibly small.

14409 nets processed, total wire 1520786.4

0 unknown nets encountered (maybe with repetitions)

130 constant nets, length 16591.2, (1.1%)

0 nets had errors, length 0.0, (0.0%)

Max crosstalk induced timing delta is 1.66n, (0 > 1us)
 Sum of all error voltages is 0.000 volts

SYNOPSIS SPIE64:

Number of ports: 385
 Number of nets: 1025
 Number of cells: 641
 Number of references: 3

Combinational area: 376566.437500
 Noncombinational area: 23417.855469
 Net Interconnect area: undefined (Wire load has zero net area)
 Total cell area: 399984.281250
 Total area: undefined

Cell Internal Power = 41.6141 mW (12%)
 Net Switching Power = 312.7236 mW (88%)

 Total Dynamic Power = 354.3378 mW (100%)
 Cell Leakage Power = 26.5031 uW

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
multiplier_1/out_low_71_ (INTERMULT64)	0.00	7.37 f
OUT1_reg_71_/D (DFFPQ1)	0.00	7.37 f
data arrival time		7.37
clock clk (rise edge)	10.00	10.00
clock network delay (propagated)	0.00	10.00
clock uncertainty	-0.50	9.50
OUT1_reg_71_/CK (DFFPQ1)	0.00	9.50 r
library setup time	-0.17	9.33
data required time		9.33
data required time		9.33
data arrival time		-7.37
slack (MET)		1.96

54-Bit Multiplier:

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 12:34:56, 26 March 2003

Design name: SPIE54

Report file name: SPIE54.summary

Number of macros: 342
 Number of components: 9391
 Number of pins: 46505
 Number of regular pins: 28671
 Number of special pins: 17540
 Number of unused pins: 294
 Number of nets: 10388
 Average number of pins per net: 4.48
 Number of subnets: 618
 Number of regular pins for subnets: 325
 Number of special pins for subnets: 0
 Number of virtual pins for subnets: 911
 Average number of pins per subnet: 2.00
 Number of routing tracks available: 2350
 Number of GCELLS per layer: 13824

** UTILIZATION OF ALL ROW TYPES

Type	Number	Length	Area	%_Row_Space
core Rows	99	60570180	373112308800	
core Cells	8770	48514620	298850059200	80.10

Area of chip: 506872766400 (square DBU)
 Area required for all cells: 298850059200 (square DBU)
 Area utilization of all cells: 58.96%

*****SILICON_ENSEMBLE WIRING REPORT*****

Time: 2:48:06, 31 March 2003

Design name: SPIE54

Report file name: SPIE54.wires

Total vias in regular wiring: 77755
 Total segments in regular wiring: 82116
 Total vias in special wiring: 728

Total segments in special wiring: 338

LAYER name: metall

Total wire length: 102118.66 microns

Length of regular wires: 33135.46 microns

Length of special wires: 68983.20 microns

LAYER name: metal2

Total wire length: 185962.30 microns

Length of regular wires: 176695.42 microns

Length of special wires: 9266.88 microns

LAYER name: metal3

Total wire length: 294621.48 microns

Length of regular wires: 294621.48 microns

Length of special wires: .00 microns

LAYER name: metal4

Total wire length: 211710.00 microns

Length of regular wires: 211710.00 microns

Length of special wires: .00 microns

LAYER name: metal5

Total wire length: 111835.58 microns

Length of regular wires: 111835.58 microns

Length of special wires: .00 microns

LAYER name: metal6

Total wire length: 33733.72 microns

Length of regular wires: 33733.72 microns

Length of special wires: .00 microns

Total wirelength in regular wiring: 861731.66 microns

Total wirelength in special wiring: 78250.08 microns

Total wirelength in regular+special wiring: 939981.74 microns

CROSSTALK:

0 nets claimed N coupling caps, but had different no.

14 nets had more coupling than total capacitance

0 pin lists with fewer pins than they said they had.

0 nets with bizarre cap/unit length

14 nets had implausible coupling ratios.

2 wires had implausibly small Rs.

111 rising drive and 111 falling drive implausibly small.

10388 nets processed, total wire 866884.9

0 unknown nets encountered (maybe with repetitions)

110 constant nets, length 10532.4, (1.2%)

0 nets had errors, length 0.0, (0.0%)

Max crosstalk induced timing delta is 1.66n, (0 > 1us)

SYNOPSIS SPIE54:

Number of ports: 325

Number of nets: 865

Number of cells: 541

Number of references: 3

Combinational area: 279092.906250

Noncombinational area: 19758.816406

Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 298851.718750

Total area: undefined

Cell Internal Power = 32.1372 mW (11%)

Net Switching Power = 259.5480 mW (89%)

Total Dynamic Power = 291.6852 mW (100%)

Cell Leakage Power = 20.1685 uW

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
multiplier_1/out_low_70_ (INTERMULT54)	0.00	7.34 r
OUT1_reg_70_/D (DFFPQ1)	0.00	7.34 r
data arrival time		7.34
clock clk (rise edge)	10.00	10.00
clock network delay (propagated)	0.00	10.00
clock uncertainty	-0.50	9.50
OUT1_reg_70_/CK (DFFPQ1)	0.00	9.50 r
library setup time	-0.08	9.42
data required time		9.42
data required time		9.42
data arrival time		-7.34
slack (MET)		2.09

32-Bit Multiplier:

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 11:56:57, 26 March 2003

Design name: SPIE32

Report file name: SPIE32.summary

Number of macros: 210
 Number of components: 3687
 Number of pins: 17559
 Number of regular pins: 10746
 Number of special pins: 6642
 Number of unused pins: 171
 Number of nets: 3915
 Average number of pins per net: 4.49
 Number of subnets: 363
 Number of regular pins for subnets: 193
 Number of special pins for subnets: 0
 Number of virtual pins for subnets: 533
 Average number of pins per subnet: 2.00
 Number of routing tracks available: 1654
 Number of GCELLS per layer: 6840

**** UTILIZATION OF ALL ROW TYPES**

Type	Number	Length	Area	%_Row_Space
core Rows	65	26040300	160408248000	
core Cells	3321	20809140	128184302400	79.91
Area of chip: 251071128000 (square DBU)				
Area required for all cells: 128184302400 (square DBU)				
Area utilization of all cells: 51.05%				

*****SILICON_ENSEMBLE WIRING REPORT*****

Time: 11:48:59, 26 March 2003

Design name: SPIE32

Report file name: SPIE32.wire

Total vias in regular wiring: 25931
 Total segments in regular wiring: 27057
 Total vias in special wiring: 350
 Total segments in special wiring: 224

LAYER name: metall

Total wire length: 44620.96 microns
 Length of regular wires: 12973.96 microns
 Length of special wires: 31647.00 microns

LAYER name: metal2

Total wire length: 69145.30 microns
 Length of regular wires: 64631.70 microns
 Length of special wires: 4513.60 microns

LAYER name: metal3

Total wire length: 96337.48 microns
 Length of regular wires: 96337.48 microns
 Length of special wires: .00 microns

LAYER name: metal4

Total wire length: 40386.22 microns
 Length of regular wires: 40386.22 microns
 Length of special wires: .00 microns

LAYER name: metal5

Total wire length: 16193.00 microns
 Length of regular wires: 16193.00 microns
 Length of special wires: .00 microns

LAYER name: metal6

Total wire length: 100.80 microns
 Length of regular wires: 100.80 microns
 Length of special wires: .00 microns

Total wirelength in regular wiring: 230623.16 microns

Total wirelength in special wiring: 36160.60 microns

Total wirelength in regular+special wiring: 266783.76 microns

CROSSTALK:

0 nets claimed N coupling caps, but had different no.
 10 nets had more coupling than total capacitance
 0 pin lists with fewer pins than they said they had.
 0 nets with bizarre cap/unit length
 10 nets had implausible coupling ratios.
 2 wires had implausibly small Rs.
 67 rising drive and 67 falling drive implausibly small.
 3915 nets processed, total wire 232417.7
 0 unknown nets encountered (maybe with repetitions)
 66 constant nets, length 5419.3, (2.3%)
 0 nets had errors, length 0.0, (0.0%)
 Max crosstalk induced timing delta is 0.467n, (0 > 1us)

```

SYNOPSIS  SPIE32:
Number of ports:      193
Number of nets:       513
Number of cells:      321
Number of references: 3

Combinational area: 116475.921875
Noncombinational area: 11708.927734
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area: 128184.851562
Total area: undefined

Cell Internal Power = 15.3479 mW (9%)
Net Switching Power = 149.5410 mW (91%)

Total Dynamic Power = 164.8889 mW (100%)
Cell Leakage Power = 9.2980 uW

```

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
multiplier_1/out_low_44_ (INTERMULT32)	0.00	6.44 r
OUT1_reg_44_/D (DFFPQ1)	0.00	6.44 r
data arrival time		6.44
clock clk (rise edge)	10.00	10.00
clock network delay (propagated)	0.00	10.00
clock uncertainty	-0.50	9.50
OUT1_reg_44_/CK (DFFPQ1)	0.00	9.50 r
library setup time	-0.08	9.42
data required time		9.42
data required time		9.42
data arrival time		-6.44
slack (MET)		2.98

16-bit Multiplier:

```

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****
Time: 21:34:48, 30 March 2003
Design name: SPIE16
Report file name: SPIE16.summary

```

```

Number of macros: 113
Number of components: 1069
Number of pins: 5036
  Number of regular pins: 3096
  Number of special pins: 1940
  Number of unused pins: 0
Number of nets: 1140
Average number of pins per net: 4.42
Number of subnets: 0
Number of routing tracks available: 1128
Number of GCELLS per layer: 3224

```

```

** UTILIZATION OF ALL ROW TYPES
Type      Number      Length      Area %_Row_Space
core Rows      39      9395100      57873816000
core Cells     970      7543140      46465742400      80.29

```

```

Area of chip: 116751835200 (square DBU)
Area required for all cells: 46465742400 (square DBU)
Area utilization of all cells: 39.80%

```

```

*****SILICON_ENSEMBLE WIRING REPORT*****
Time: 21:34:56, 30 March 2003
Design name: SPIE16
Report file name: SPIE16.wires

```

```

Total vias in regular wiring: 6674
Total segments in regular wiring: 6834
Total vias in special wiring: 176
Total segments in special wiring: 140

```

```

LAYER name: metall
Total wire length: 17036.44 microns
Length of regular wires: 4171.72 microns
Length of special wires: 12864.72 microns

```

```

LAYER name: metal2
  Total wire length: 17575.84 microns
  Length of regular wires: 15237.28 microns
  Length of special wires: 2338.56 microns
LAYER name: metal3
  Total wire length: 25701.42 microns
  Length of regular wires: 25701.42 microns
  Length of special wires: .00 microns
LAYER name: metal4
  Total wire length: 3398.28 microns
  Length of regular wires: 3398.28 microns
  Length of special wires: .00 microns
LAYER name: metal5
  Total wire length: 762.30 microns
  Length of regular wires: 762.30 microns
  Length of special wires: .00 microns

Total wirelength in regular wiring: 49271.00 microns
Total wirelength in special wiring: 15203.28 microns
Total wirelength in regular+special wiring: 64474.28 microns

```

```

CROSSTALK:
  0 nets claimed N coupling caps, but had different no.
  2 nets had more coupling than total capacitance
  0 pin lists with fewer pins than they said they had.
  0 nets with bizarre cap/unit length
  2 nets had implausible coupling ratios.
  2 wires had implausibly small Rs.
  35 rising drive and 35 falling drive implausibly small.
  1140 nets processed, total wire      49744.3
  0 unknown nets encountered (maybe with repetitions)
  34 constant nets, length      2044.3, ( 4.1%)
  0 nets had errors, length      0.0, ( 0.0%)
Max crosstalk induced timing delta is 0.497n, (0 > 1us)

```

SYNOPSIS SPIE16:

```

Number of ports:      97
Number of nets:       257
Number of cells:      161
Number of references:  3

Combinational area:   40611.406250
Noncombinational area: 5854.463867
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area:      46465.871094
Total area:           undefined

Cell Internal Power = 6.1922 mW (8%)
Net Switching Power = 70.0727 mW (92%)
-----
Total Dynamic Power = 76.2650 mW (100%)
Cell Leakage Power  = 3.7045 uW

```

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
multiplier_1/out_low[21] (INTERMULT16)	0.00	5.00 r
OUT1_reg_21_/D (DFFPQ1)	0.00	5.00 r
data arrival time		5.00
clock clk (rise edge)	10.00	10.00
clock network delay (propagated)	0.00	10.00
clock uncertainty	-0.50	9.50
OUT1_reg_21_/CK (DFFPQ1)	0.00	9.50 r
library setup time	-0.07	9.43
data required time		9.43
data arrival time		-5.00
slack (MET)		4.43

Appendix C

Verilog HDL Code for the Reconfigurable Multiplier

```

/*****
Author:      Pedram Mokrian
Title:      RecursiveMultiplierV4.v
Purpose:     Main file for Thesis Multiplier

Comments:    File revised Feb 5, 2003
              Latest file to be used for the Reconfigurable Multiplier

Updates:

March 3, 2002
Change the output size of the fast adder to overcome the previous discrepancies.
Reformatted the text and positioning of the code for enhanced legibility

*****/

module RecursiveMultiplierV4 (AH,AL,XH,XL,OUT_HIGH,OUT_LOW,CONTROL,clk,reset);

// High and Low order bits of input A (MULTIPLICAND)
input [32:1]  AH, AL;

// High and Low order bits of input X (MULTIPLIER)
input [32:1]  XH, XL;

// Clock and Control Signals
input  clk,reset;
input [2:1]   CONTROL;

// High and Low Order Output Bits
output [64:1] OUT_HIGH, OUT_LOW;

/*****
NOMENCLATURE
no suffix -> top module ports
_GATED    -> clocked signals, that are wires to RecMult3 module
*****/
reg [64:1]  OUT_LOW, OUT_HIGH;

wire [64:1] OUT_LOW_GATED, OUT_HIGH_GATED;
reg [32:1]  AH_GATED, AL_GATED, XH_GATED, XL_GATED;

// The system operates with a high reset signal, and is set to zero if reset=0
always @ (posedge clk or negedge reset)
begin

```



```

        if (!reset) begin
            AH_GATED <= 32'b0;
            AL_GATED <= 32'b0;
            XH_GATED <= 32'b0;
            XL_GATED <= 32'b0;

            OUT_HIGH <= 32'b0;
            OUT_LOW <= 32'b0;

        end // if (!reset)

        else begin
            AH_GATED <= AH;
            AL_GATED <= AL;
            XH_GATED <= XH;
            XL_GATED <= XL;

            OUT_HIGH <= OUT_HIGH_GATED;
            OUT_LOW <= OUT_LOW_GATED;
        end // else: !if(!reset)
    end // always @ (posedge clk or negedge reset)

    RecMult recmult
    (AH_GATED,AL_GATED,XH_GATED,XL_GATED,OUT_HIGH_GATED,OUT_LOW_GATED,CONTROL);

endmodule

/*****
Author:      Pedram Mokrian
Title:       RecMult.v
Purpose:     Main file for Thesis Multiplier

Comments:    This code is for the design of a variable precision
              multiplier with a control signal governing the 4 modes of
              operation
*****/

module RecMult (AH,AL,XH,XL,OUT_HIGH,OUT_LOW,CONTROL);

    // High and Low order bits of input A (MULTIPLICAND)
    input [32:1]  AH;
    input [32:1]  AL;

    // High and Low order bits of input X (MULTIPLIER)
    input [32:1]  XH;
    input [32:1]  XL;

    // Control Signal
    input [2:1]   CONTROL;

    // High and Low Order Output Bits
    output [64:1] OUT_HIGH;
    output [64:1] OUT_LOW;

```

```

/*****
CONTROL DEFINITIONS:
    CONTROL = 00    => Double precision (64-bit) mode
    CONTROL = 01    => Dual Single Precision mode
    CONTROL = 10    => Single Precision Fault Tolerant mode
    CONTROL = 11    => Dual Single Precision mode

MULTIPLIER DEFINITIONS:
    MULT_1  => AH * XH
    MULT_2  => AL * XH    or AL * XL
    MULT_3  => AH * XL    or AL * XL
    MULT_4  => AL * XL
*****/

/*****
The Intermediary Multipliers
*****/

// defining the inputs to the intermediary multipliers in carry save format
wire [32:1] MULT_1_IN_LOW;
wire [32:1] MULT_1_IN_HIGH;
wire [32:1] MULT_2_IN_LOW;
wire [32:1] MULT_2_IN_HIGH;
wire [32:1] MULT_3_IN_LOW;
wire [32:1] MULT_3_IN_HIGH;
wire [32:1] MULT_4_IN_LOW;
wire [32:1] MULT_4_IN_HIGH;

// defining the outputs to the intermediary multipliers in carry save format
wire [64:1] MULT_1_OUT_LOW;
wire [64:1] MULT_1_OUT_HIGH;
wire [64:1] MULT_2_OUT_LOW;
wire [64:1] MULT_2_OUT_HIGH;
wire [64:1] MULT_3_OUT_LOW;
wire [64:1] MULT_3_OUT_HIGH;
wire [64:1] MULT_4_OUT_LOW;
wire [64:1] MULT_4_OUT_HIGH;

// First stage of multiplexers governing the input to the Multipliers.

// The inputs to MULT_1 are the outputs of MUX1
// MUX1 (input1, input2, output1, output2, control)
MUX1 mux1(AH, XH, MULT_1_IN_LOW, MULT_1_IN_HIGH, CONTROL);

// The inputs to MULT_2 are the outputs of MUX2
// MUX2 (input_solo, input2, input3, output1, output2, control)
MUX2 mux2 (AL, XH, XL, MULT_2_IN_LOW, MULT_2_IN_HIGH, CONTROL);

// The inputs to MULT_3 are the outputs of MUX3
// MUX3 (input1, input2, input_solo, output1, output2, control)
MUX3 mux3 (AH, AL, XL, MULT_3_IN_LOW, MULT_3_IN_HIGH, CONTROL);

// The inputs for MULT_4 are always the same
assign MULT_4_IN_LOW = AL;
assign MULT_4_IN_HIGH = XL;

// Intermediary multiplier instantiations

```

```

        INTERMULT multiplier_1
(MULT_1_IN_LOW,MULT_1_IN_HIGH,MULT_1_OUT_LOW,MULT_1_OUT_HIGH);
        INTERMULT multiplier_2
(MULT_2_IN_LOW,MULT_2_IN_HIGH,MULT_2_OUT_LOW,MULT_2_OUT_HIGH);
        INTERMULT multiplier_3
(MULT_3_IN_LOW,MULT_3_IN_HIGH,MULT_3_OUT_LOW,MULT_3_OUT_HIGH);
        INTERMULT multiplier_4
(MULT_4_IN_LOW,MULT_4_IN_HIGH,MULT_4_OUT_LOW,MULT_4_OUT_HIGH);

/*****
The output of the Multipliers and the REDUCTION AND MAJORITY stage
*****/

        wire [64:1] RED_1_OUT_LOW;
        wire [64:1] RED_1_OUT_HIGH;
        wire [64:1] RED_2_OUT_LOW;
        wire [64:1] RED_2_OUT_HIGH;
        wire [64:1] RED_3_OUT_LOW;
        wire [64:1] RED_3_OUT_HIGH;
        wire [64:1] RED_4_OUT_LOW;
        wire [64:1] RED_4_OUT_HIGH;

        wire [64:1] MAJ_2_OUT_LOW;
        wire [64:1] MAJ_2_OUT_HIGH;
        wire [64:1] MAJ_3_OUT_LOW;
        wire [64:1] MAJ_3_OUT_HIGH;
        wire [64:1] MAJ_4_OUT_LOW;
        wire [64:1] MAJ_4_OUT_HIGH;

        wire [64:1] MAJ_LOW1;
        wire [64:1] MAJ_LOW2;
        wire [64:1] RED_LOW1;
        wire [64:1] RED_LOW2;
        wire [64:1] RED_HIGH1;
        wire [64:1] RED_HIGH2;

/*****
The outputs of the multipliers are guided by MUX_Reduction and MUX_Majority
multiplexers
        MUX_Reduction (input1, input2, input3, input4, input5, input6, input7, input8,
                        output1, output2, output3, output4, output5, output6, output7,
                        output8, control)
*****/

// This MUX provides inputs to the Reduction block if control = 00

MUX_Reduction redmux (MULT_1_OUT_LOW, MULT_1_OUT_HIGH, MULT_2_OUT_LOW,
MULT_2_OUT_HIGH, MULT_3_OUT_LOW, MULT_3_OUT_HIGH, MULT_4_OUT_LOW,
MULT_4_OUT_HIGH, RED_1_OUT_LOW, RED_1_OUT_HIGH, RED_2_OUT_LOW, RED_2_OUT_HIGH,
RED_3_OUT_LOW, RED_3_OUT_HIGH, RED_4_OUT_LOW, RED_4_OUT_HIGH, CONTROL);

// This MUX provides inputs to the Majority block if control = 10

MUX_Majority majmux ( MULT_2_OUT_LOW, MULT_2_OUT_HIGH, MULT_3_OUT_LOW,
MULT_3_OUT_HIGH, MULT_4_OUT_LOW, MULT_4_OUT_HIGH, MAJ_2_OUT_LOW, MAJ_2_OUT_HIGH,
MAJ_3_OUT_LOW, MAJ_3_OUT_HIGH, MAJ_4_OUT_LOW, MAJ_4_OUT_HIGH, CONTROL);

```

```

MAJORITY Majority_Voter(MAJ_2_OUT_LOW, MAJ_2_OUT_HIGH, MAJ_3_OUT_LOW,
MAJ_3_OUT_HIGH, MAJ_4_OUT_LOW, MAJ_4_OUT_HIGH, MAJ_LOW1, MAJ_LOW2);

REDUCTION Reduction_6_to_2 (RED_1_OUT_LOW, RED_1_OUT_HIGH, RED_2_OUT_LOW,
RED_2_OUT_HIGH, RED_3_OUT_LOW, RED_3_OUT_HIGH, RED_4_OUT_LOW, RED_4_OUT_HIGH,
RED_LOW1, RED_HIGH1, RED_LOW2, RED_HIGH2);

// Register definition for the inputs to the two FAST-ADDERS
reg [64:1] FASTADDER_LOW1;
reg [64:1] FASTADDER_LOW2;
reg [64:1] FASTADDER_HIGH1;
reg [64:1] FASTADDER_HIGH2;
// carry over signal used between the two adder sections
wire CARRY_OVER, C_OUT;
wire DUMMY;
assign DUMMY = 1'b0;

wire [64:1] OUT_HIGH;
wire [64:1] OUT_LOW;

// The inputs to the Final Fast Adders are governed by the always@(control) block
always@ (CONTROL)
begin
    if (CONTROL == 2'b01) // Dual Single Precision Mode
    begin
        FASTADDER_LOW1 = MULT_4_OUT_LOW;
        FASTADDER_LOW2 = MULT_4_OUT_HIGH;
        FASTADDER_HIGH1 = MULT_1_OUT_LOW;
        FASTADDER_HIGH2 = MULT_1_OUT_HIGH;
    end

    else if (CONTROL == 2'b10) // Single Precision Fault Tolerant
    begin
        FASTADDER_LOW1 = MAJ_LOW1;
        FASTADDER_LOW2 = MAJ_LOW2;
        FASTADDER_HIGH1 = 64'b0;
        FASTADDER_HIGH2 = 64'b0;
    end // 10

    else if (CONTROL == 2'b11) // Single Precision
    begin
        FASTADDER_LOW1 = MULT_4_OUT_LOW;
        FASTADDER_LOW2 = MULT_4_OUT_HIGH;
        FASTADDER_HIGH1 = 64'b0;
        FASTADDER_HIGH2 = 64'b0;
    end // 11

    else // Double Precision Mode (DEFAULT MODE)
    begin
        FASTADDER_LOW1 = RED_LOW1;
        FASTADDER_LOW2 = RED_LOW2;
        FASTADDER_HIGH1 = RED_HIGH1;
        FASTADDER_HIGH2 = RED_HIGH2;
    end // 00
end // always@ (CONTROL)

```

```
// instantiation of the two fast adder modules
    FASTADDER
FinalLow(DUMMY, FASTADDER_LOW1, FASTADDER_LOW2, OUT_LOW, CARRY_OVER);
    FASTADDER
FinalHigh(CARRY_OVER, FASTADDER_HIGH1, FASTADDER_HIGH2, OUT_HIGH, COUT);

endmodule // RecMult3
```

```

/*****
The MUX Modules
*****/

module MUX3 (input1, input2, input_solo, output1, output2, control);
    input [1:0] control;
    input [31:0] input_solo, input1, input2;
    output [31:0] output1, output2;

    reg [31:0] output1, output2;
    wire [31:0] ZERO;
    assign ZERO=32'b0;

    always@(input_solo or input1 or input2 or control)
    begin
        if (control == 2'b00)
            begin
                output1 = input1;
                output2 = input_solo;
            end
        else if (control == 2'b10)
            begin
                output1 = input2;
                output2 = input_solo;
            end
        else
            begin
                output1 = ZERO;
                output2 = ZERO;
            end
        end // always@ (input_solo or input2 or input3 or control)
    endmodule // MUX3
```

```

module MUX2 (input_solo, input2, input3, output1, output2, control);
    input [1:0] control;
    input [31:0] input_solo, input2, input3;
    output [31:0] output1, output2;

    reg [31:0] output1, output2;
    wire [31:0] ZERO;
    assign ZERO=32'b0;

    always@(input_solo or input2 or input3 or control)
    begin
        if (control == 2'b00)
            begin
```

```

        output1 = input_solo;
        output2 = input2;
    end
    else if (control == 2'b10)
    begin
        output1 = input_solo;
        output2 = input3;
    end
    else
    begin
        output1 = ZERO;
        output2 = ZERO;
    end
    end // always@ (input_solo or input2 or input3 or control)
endmodule // MUX2

```

```

module MUX1 (input1, input2, output1, output2, control);
    input [1:0] control;
    input [31:0] input1, input2;
    output [31:0] output1, output2;

    reg [31:0] output1, output2;
    wire [31:0] ZERO;
    assign ZERO=32'b0;

    always@(input1 or input2 or control)
    begin
        if (control == 2'b00)
        begin
            output1 = input1;
            output2 = input2;
        end
        else if (control == 2'b01)
        begin
            output1 = input1;
            output2 = input2;
        end
        else
        begin
            output1 = ZERO;
            output2 = ZERO;
        end
    end // always@ (input1 or input2 or control)
endmodule // MUX1

```

```

/*****
The OUTPUTMUX Modules
These mux cells are used to guide the ouputs of the 4 multipliers
*****/

```

```

module MUX_Reduction (input1, input2, input3, input4, input5, input6, input7,
input8, output1, output2, output3, output4, output5, output6, output7, output8,
control);

    input [1:0] control;
    input [63:0] input1, input2, input3, input4, input5, input6, input7, input8;
    output [63:0] output1,output2,output3,output4,output5,output6,output7,output8;

```

```

reg [63:0] output1,output2,output3,output4,output5,output6,output7,output8;
wire [63:0] ZERO;
assign      ZERO=64'b0;

always@(input1 or input2 or input3 or input4 or input5 or input6 or input7 or
input8 or control)
begin
if (control == 2'b00)
begin
output1 = input1;
output2 = input2;
output3 = input3;
output4 = input4;
output5 = input5;
output6 = input6;
output7 = input7;
output8 = input8;
end
else
begin
output1 = ZERO;
output2 = ZERO;
output3 = ZERO;
output4 = ZERO;
output5 = ZERO;
output6 = ZERO;
output7 = ZERO;
output8 = ZERO;
end
end // always@
endmodule // MUX_Reduction

```

```

module MUX_Majority (input1, input2, input3, input4, input5, input6,
output1, output2, output3, output4, output5, output6, control);

input [1:0] control;
input [63:0] input1, input2, input3, input4, input5, input6;
output [63:0] output1, output2, output3, output4, output5, output6;

reg [63:0] output1, output2, output3, output4, output5, output6;
wire [63:0] ZERO;
assign ZERO=64'b0;

always@(input1 or input2 or input3 or input4 or input5 or input6 or control)
begin
if (control == 2'b10)
begin
output1 = input1;
output2 = input2;
output3 = input3;
output4 = input4;
output5 = input5;
output6 = input6;
end
else
begin
output1 = ZERO;

```

```

        output2 = ZERO;
        output3 = ZERO;
        output4 = ZERO;
        output5 = ZERO;
        output6 = ZERO;
    end // else: !if(control == 2'b10)
    end // always@ (input1 or input2 or input3 or input4 or input5 or input6 or
control)
endmodule // MUX_Majority

```

```

/*****
The INTER_MULT module instantiates the DW02_multp_instance module
and truncates the output of the 2 MSB since they are sign bits'
and the tc signal should be set low.
*****/

```

```

module INTERMULT (in_low, in_high, out_low, out_high);
    input [31:0] in_low, in_high;
    output [63:0] out_low, out_high;

    wire [65:0] multp_low, multp_high;
    wire DUMMY;
    assign DUMMY=1'b0;

    DW02_multp_instance inter_mult(in_low,in_high,DUMMY,multp_low,multp_high);

    assign out_low = multp_low [63:0];
    assign out_high = multp_high [64:1];
endmodule // INTERMULT

```

```

/*****
This module is used to instantiate the partial product multiplier
supplied in the Synopsys Foundation Design Libraries.
*****/

```

```

module DW02_multp_instance (ina,inb,intc,prod0,prod1);

// parameter definition for a 32 bit multiplier
parameter a_width = 32;
parameter b_width = 32;
parameter out_width = 66;

    input [a_width-1 : 0] ina;
    input [b_width-1 : 0] inb;
    input intc;
    output [out_width-1:0] prod0,prod1;

// Instantiation of DW02_multp.v
DW02_multp #(a_width, b_width, out_width) U1
    (.a(ina), .b(inb), .tc(intc), .out0(prod0), .out1(prod1));

endmodule // DW02_multp_instance

```



```

/*****
Final Fast Adder Module
*****/

module FASTADDER (C_IN, IN1, IN2, OUT, C_OUT);
    input [63:0] IN1, IN2;
    input C_IN;
    output [63:0] OUT;
    output C_OUT;

    reg [64:0] SUMOUT;
    reg C_OUT;
    reg [63:0] OUT;

    always@(IN1 or IN2 or C_IN)
    begin
        SUMOUT = IN1 + IN2 + C_IN;
        C_OUT = SUMOUT[64];
        OUT = SUMOUT[63:0];
    end
endmodule // FASTADDER

/*****
The majority module is designed for error correction, where the products from the
intermediary multipliers are voted on to determine a bitwise 2 out of 3 decision
by the circuitry. This module is only active in single precision, fault tolerant
operation.
*****/

module MAJORITY (low1, high1, low2, high2, low3, high3, output_low, output_high);
    input [64:1] low1, low2, low3;
    input [64:1] high1, high2, high3;

    output [64:1] output_low, output_high;
    reg [64:1] output_low, output_high;

    always@(low1 or low2 or low3 or high1 or high2 or high3)
    begin
        //The majority check performs XNOR on inputs 1 and 2, if true input 1 is passed/
        //else input 3 is passed as the final result.

        output_low[1] = (low1[1] ^ low2[1]) ? low1[1] : low3[1];
        output_low[2] = (low1[2] ^ low2[2]) ? low1[2] : low3[2];
        output_low[3] = (low1[3] ^ low2[3]) ? low1[3] : low3[3];
        output_low[4] = (low1[4] ^ low2[4]) ? low1[4] : low3[4];
        output_low[5] = (low1[5] ^ low2[5]) ? low1[5] : low3[5];
        output_low[6] = (low1[6] ^ low2[6]) ? low1[6] : low3[6];
        output_low[7] = (low1[7] ^ low2[7]) ? low1[7] : low3[7];
        output_low[8] = (low1[8] ^ low2[8]) ? low1[8] : low3[8];
        output_low[9] = (low1[9] ^ low2[9]) ? low1[9] : low3[9];
        output_low[10] = (low1[10] ^ low2[10]) ? low1[10] : low3[10];
        output_low[11] = (low1[11] ^ low2[11]) ? low1[11] : low3[11];
        output_low[12] = (low1[12] ^ low2[12]) ? low1[12] : low3[12];
        output_low[13] = (low1[13] ^ low2[13]) ? low1[13] : low3[13];
        output_low[14] = (low1[14] ^ low2[14]) ? low1[14] : low3[14];
    end
endmodule

```

```

output_low[15] = (low1[15]~^low2[15]) ? low1[15] : low3[15];
output_low[16] = (low1[16]~^low2[16]) ? low1[16] : low3[16];
output_low[17] = (low1[17]~^low2[17]) ? low1[17] : low3[17];
output_low[18] = (low1[18]~^low2[18]) ? low1[18] : low3[18];
output_low[19] = (low1[19]~^low2[19]) ? low1[19] : low3[19];
output_low[20] = (low1[20]~^low2[20]) ? low1[20] : low3[20];
output_low[21] = (low1[21]~^low2[21]) ? low1[21] : low3[21];
output_low[22] = (low1[22]~^low2[22]) ? low1[22] : low3[22];
output_low[23] = (low1[23]~^low2[23]) ? low1[23] : low3[23];
output_low[24] = (low1[24]~^low2[24]) ? low1[24] : low3[24];
output_low[25] = (low1[25]~^low2[25]) ? low1[25] : low3[25];
output_low[26] = (low1[26]~^low2[26]) ? low1[26] : low3[26];
output_low[27] = (low1[27]~^low2[27]) ? low1[27] : low3[27];
output_low[28] = (low1[28]~^low2[28]) ? low1[28] : low3[28];
output_low[29] = (low1[29]~^low2[29]) ? low1[29] : low3[29];
output_low[30] = (low1[30]~^low2[30]) ? low1[30] : low3[30];
output_low[31] = (low1[31]~^low2[31]) ? low1[31] : low3[31];
output_low[32] = (low1[32]~^low2[32]) ? low1[32] : low3[32];
output_low[33] = (low1[33]~^low2[33]) ? low1[33] : low3[33];
output_low[34] = (low1[34]~^low2[34]) ? low1[34] : low3[34];
output_low[35] = (low1[35]~^low2[35]) ? low1[35] : low3[35];
output_low[36] = (low1[36]~^low2[36]) ? low1[36] : low3[36];
output_low[37] = (low1[37]~^low2[37]) ? low1[37] : low3[37];
output_low[38] = (low1[38]~^low2[38]) ? low1[38] : low3[38];
output_low[39] = (low1[39]~^low2[39]) ? low1[39] : low3[39];
output_low[40] = (low1[40]~^low2[40]) ? low1[40] : low3[40];
output_low[41] = (low1[41]~^low2[41]) ? low1[41] : low3[41];
output_low[42] = (low1[42]~^low2[42]) ? low1[42] : low3[42];
output_low[43] = (low1[43]~^low2[43]) ? low1[43] : low3[43];
output_low[44] = (low1[44]~^low2[44]) ? low1[44] : low3[44];
output_low[45] = (low1[45]~^low2[45]) ? low1[45] : low3[45];
output_low[46] = (low1[46]~^low2[46]) ? low1[46] : low3[46];
output_low[47] = (low1[47]~^low2[47]) ? low1[47] : low3[47];
output_low[48] = (low1[48]~^low2[48]) ? low1[48] : low3[48];
output_low[49] = (low1[49]~^low2[49]) ? low1[49] : low3[49];
output_low[50] = (low1[50]~^low2[50]) ? low1[50] : low3[50];
output_low[51] = (low1[51]~^low2[51]) ? low1[51] : low3[51];
output_low[52] = (low1[52]~^low2[52]) ? low1[52] : low3[52];
output_low[53] = (low1[53]~^low2[53]) ? low1[53] : low3[53];
output_low[54] = (low1[54]~^low2[54]) ? low1[54] : low3[54];
output_low[55] = (low1[55]~^low2[55]) ? low1[55] : low3[55];
output_low[56] = (low1[56]~^low2[56]) ? low1[56] : low3[56];
output_low[57] = (low1[57]~^low2[57]) ? low1[57] : low3[57];
output_low[58] = (low1[58]~^low2[58]) ? low1[58] : low3[58];
output_low[59] = (low1[59]~^low2[59]) ? low1[59] : low3[59];
output_low[60] = (low1[60]~^low2[60]) ? low1[60] : low3[60];
output_low[61] = (low1[61]~^low2[61]) ? low1[61] : low3[61];
output_low[62] = (low1[62]~^low2[62]) ? low1[62] : low3[62];
output_low[63] = (low1[63]~^low2[63]) ? low1[63] : low3[63];
output_low[64] = (low1[64]~^low2[64]) ? low1[64] : low3[64];

output_high[1] = (high1[1]~^high2[1]) ? high1[1] : high3[1];
output_high[2] = (high1[2]~^high2[2]) ? high1[2] : high3[2];
output_high[3] = (high1[3]~^high2[3]) ? high1[3] : high3[3];
output_high[4] = (high1[4]~^high2[4]) ? high1[4] : high3[4];
output_high[5] = (high1[5]~^high2[5]) ? high1[5] : high3[5];
output_high[6] = (high1[6]~^high2[6]) ? high1[6] : high3[6];
output_high[7] = (high1[7]~^high2[7]) ? high1[7] : high3[7];
output_high[8] = (high1[8]~^high2[8]) ? high1[8] : high3[8];

```

```

output_high[9] = (high1[9]~^high2[9]) ? high1[9] : high3[9];
output_high[10] = (high1[10]~^high2[10]) ? high1[10] : high3[10];
output_high[11] = (high1[11]~^high2[11]) ? high1[11] : high3[11];
output_high[12] = (high1[12]~^high2[12]) ? high1[12] : high3[12];
output_high[13] = (high1[13]~^high2[13]) ? high1[13] : high3[13];
output_high[14] = (high1[14]~^high2[14]) ? high1[14] : high3[14];
output_high[15] = (high1[15]~^high2[15]) ? high1[15] : high3[15];
output_high[16] = (high1[16]~^high2[16]) ? high1[16] : high3[16];
output_high[17] = (high1[17]~^high2[17]) ? high1[17] : high3[17];
output_high[18] = (high1[18]~^high2[18]) ? high1[18] : high3[18];
output_high[19] = (high1[19]~^high2[19]) ? high1[19] : high3[19];
output_high[20] = (high1[20]~^high2[20]) ? high1[20] : high3[20];
output_high[21] = (high1[21]~^high2[21]) ? high1[21] : high3[21];
output_high[22] = (high1[22]~^high2[22]) ? high1[22] : high3[22];
output_high[23] = (high1[23]~^high2[23]) ? high1[23] : high3[23];
output_high[24] = (high1[24]~^high2[24]) ? high1[24] : high3[24];
output_high[25] = (high1[25]~^high2[25]) ? high1[25] : high3[25];
output_high[26] = (high1[26]~^high2[26]) ? high1[26] : high3[26];
output_high[27] = (high1[27]~^high2[27]) ? high1[27] : high3[27];
output_high[28] = (high1[28]~^high2[28]) ? high1[28] : high3[28];
output_high[29] = (high1[29]~^high2[29]) ? high1[29] : high3[29];
output_high[30] = (high1[30]~^high2[30]) ? high1[30] : high3[30];
output_high[31] = (high1[31]~^high2[31]) ? high1[31] : high3[31];
output_high[32] = (high1[32]~^high2[32]) ? high1[32] : high3[32];
output_high[33] = (high1[33]~^high2[33]) ? high1[33] : high3[33];
output_high[34] = (high1[34]~^high2[34]) ? high1[34] : high3[34];
output_high[35] = (high1[35]~^high2[35]) ? high1[35] : high3[35];
output_high[36] = (high1[36]~^high2[36]) ? high1[36] : high3[36];
output_high[37] = (high1[37]~^high2[37]) ? high1[37] : high3[37];
output_high[38] = (high1[38]~^high2[38]) ? high1[38] : high3[38];
output_high[39] = (high1[39]~^high2[39]) ? high1[39] : high3[39];
output_high[40] = (high1[40]~^high2[40]) ? high1[40] : high3[40];
output_high[41] = (high1[41]~^high2[41]) ? high1[41] : high3[41];
output_high[42] = (high1[42]~^high2[42]) ? high1[42] : high3[42];
output_high[43] = (high1[43]~^high2[43]) ? high1[43] : high3[43];
output_high[44] = (high1[44]~^high2[44]) ? high1[44] : high3[44];
output_high[45] = (high1[45]~^high2[45]) ? high1[45] : high3[45];
output_high[46] = (high1[46]~^high2[46]) ? high1[46] : high3[46];
output_high[47] = (high1[47]~^high2[47]) ? high1[47] : high3[47];
output_high[48] = (high1[48]~^high2[48]) ? high1[48] : high3[48];
output_high[49] = (high1[49]~^high2[49]) ? high1[49] : high3[49];
output_high[50] = (high1[50]~^high2[50]) ? high1[50] : high3[50];
output_high[51] = (high1[51]~^high2[51]) ? high1[51] : high3[51];
output_high[52] = (high1[52]~^high2[52]) ? high1[52] : high3[52];
output_high[53] = (high1[53]~^high2[53]) ? high1[53] : high3[53];
output_high[54] = (high1[54]~^high2[54]) ? high1[54] : high3[54];
output_high[55] = (high1[55]~^high2[55]) ? high1[55] : high3[55];
output_high[56] = (high1[56]~^high2[56]) ? high1[56] : high3[56];
output_high[57] = (high1[57]~^high2[57]) ? high1[57] : high3[57];
output_high[58] = (high1[58]~^high2[58]) ? high1[58] : high3[58];
output_high[59] = (high1[59]~^high2[59]) ? high1[59] : high3[59];
output_high[60] = (high1[60]~^high2[60]) ? high1[60] : high3[60];
output_high[61] = (high1[61]~^high2[61]) ? high1[61] : high3[61];
output_high[62] = (high1[62]~^high2[62]) ? high1[62] : high3[62];
output_high[63] = (high1[63]~^high2[63]) ? high1[63] : high3[63];
output_high[64] = (high1[64]~^high2[64]) ? high1[64] : high3[64];

end // always@ (low1 or low2 or low3 or high1 or high2 or high3)
endmodule // MAJORITY

```

```

/*****
The reduction module is designed to reduce the 4 64-bit values in carry save
format from the intermediary multipliers, down to two 128 bit values. The output
is split into 4 sections, where each 128-bit value is divided into two halves.
This module is only active in double precision operation.
*****/

```

```

/*****
The RED_BLOCK1 module is the first reduction module used in the REDUCTION module
*****/
module RED_BLOCK1 (C2,C1,C0,S2,S1,S0,COUT1,COUT2,COUT3,CP,SP);
    input C2,C1,C0,S2,S1,S0;
    output COUT1,COUT2,COUT3,CP,SP;

    wire FA1_SUM;
    wire FA2_SUM;

    FULLADDER fa1(S2,S1,S0,FA1_SUM,COUT1);
    FULLADDER fa2(C2,C1,C0,FA2_SUM,COUT2);
    HALFADDER fa3(FA1_SUM,FA2_SUM,SP,COUT3);
    assign CP = 1'b0;
endmodule // RED_BLOCK1

```

```

/*****
The RED_BLOCK6 module is used as part of the REDUCTION module
*****/
module RED_BLOCK6 (C2,C1,C0,S2,S1,S0,CIN1,CIN2,CIN3,COUT1,COUT2,COUT3,CP,SP);
    input C2,C1,C0,S2,S1,S0,CIN1,CIN2,CIN3;
    output COUT1,COUT2,COUT3,CP,SP;

    wire FA1_SUM;
    wire FA2_SUM;
    wire FA3_SUM;

    FULLADDER fa1(S2,S1,S0,FA1_SUM,COUT1);
    FULLADDER fa2(C2,C1,C0,FA2_SUM,COUT2);
    FULLADDER fa3(CIN1,FA1_SUM,FA2_SUM,FA3_SUM,COUT3);
    FULLADDER fa4(CIN2,CIN3,FA3_SUM,SP,CP);
endmodule // RED_BLOCK6

```

```

/*****
The RED_BLOCK2 module is used as part of the REDUCTION module
*****/
module RED_BLOCK2 (S1,S0,CIN1,CIN2,CIN3,COUT,CP,SP);
    input S1,S0,CIN1,CIN2,CIN3;
    output COUT,CP,SP;

    wire FA1_SUM;

    FULLADDER fa1(S1,S0,CIN1,FA1_SUM,COUT);
    FULLADDER fa2(CIN2,CIN3,FA1_SUM,SP,CP);
endmodule // RED_BLOCK2

```

```

/*****
The FULLADDER module is used as part of the REDUCTION module
*****/
module FULLADDER (a,b,c,SUM,CARRY);
    input    a;
    input    b;
    input    c;
    output    SUM;
    output    CARRY;

    reg      SUM,CARRY;

    always @(a or b or c)
        begin
            CARRY = (a&&b) || (a&&c) || (b&&c);
            SUM = (a^b^c);

            end // initial begin
endmodule // FULLADDER


/*****
The HALFADDER module is used as part of the REDUCTION module
*****/
module HALFADDER (a,b,SUM,CARRY);
    input    a;
    input    b;
    output    SUM;
    output    CARRY;

    reg      SUM,CARRY;

    always @(a or b)
        begin
            CARRY = (a&&b);
            SUM = (a^b);

            end // initial begin
endmodule // HALFADDER


/*****
The REDUCTION module definition
*****/
module REDUCTION (in1_L,in1_H,in2_L,in2_H,in3_L,in3_H,in4_L,in4_H,
out1_L,out1_H,out2_L,out2_H);

    input [64:1] in1_L;
    input [64:1] in1_H;
    input [64:1] in2_L;
    input [64:1] in2_H;
    input [64:1] in3_L;
    input [64:1] in3_H;
    input [128:65] in4_L;
    input [128:65] in4_H;

    output [64:1] out1_L;

```

```

    output [64:1] out1_H;
    output [64:1] out2_L;
    output [64:1] out2_H;

// in1 is in the lowest bit position (from 1 to 64)
// in2 and in3 are from bit positions 33 to 96
// in4 is in the highest bit position (from 65 to 128)

// these two registers are used to hold the final output values of the reduction
// circuit and then to pass them directly onto the output pins of the module

    wire [128:1] redL;
    wire [128:0] redH;
    wire [96:33] C1,C2,C3;
    wire          COUT;

// calling the 6 input reduction block module
// module RED_BLOCK6 (C2,C1,C0,S2,S1,S0,CIN1,CIN2,CIN3,COUT1,COUT2,COUT3,CP,SP);

    RED_BLOCK1
    block1(in1_L[33],in1_H[33],in2_L[1],in2_H[1],in3_L[1],in3_H[1],C1[33],C2[33],C3[
33],redH[33],redL[33]);

    RED_BLOCK6
    block2(in1_L[34],in1_H[34],in2_L[2],in2_H[2],in3_L[2],in3_H[2],C1[33],C2[33],C3[
33],C1[34],C2[34],C3[34],redH[34],redL[34]);
    RED_BLOCK6
    block3(in1_L[35],in1_H[35],in2_L[3],in2_H[3],in3_L[3],in3_H[3],C1[34],C2[34],C3[
34],C1[35],C2[35],C3[35],redH[35],redL[35]);
    RED_BLOCK6
    block4(in1_L[36],in1_H[36],in2_L[4],in2_H[4],in3_L[4],in3_H[4],C1[35],C2[35],C3[
35],C1[36],C2[36],C3[36],redH[36],redL[36]);
    RED_BLOCK6
    block5(in1_L[37],in1_H[37],in2_L[5],in2_H[5],in3_L[5],in3_H[5],C1[36],C2[36],C3[
36],C1[37],C2[37],C3[37],redH[37],redL[37]);
    RED_BLOCK6
    block6(in1_L[38],in1_H[38],in2_L[6],in2_H[6],in3_L[6],in3_H[6],C1[37],C2[37],C3[
37],C1[38],C2[38],C3[38],redH[38],redL[38]);
    RED_BLOCK6
    block7(in1_L[39],in1_H[39],in2_L[7],in2_H[7],in3_L[7],in3_H[7],C1[38],C2[38],C3[
38],C1[39],C2[39],C3[39],redH[39],redL[39]);
    RED_BLOCK6
    block8(in1_L[40],in1_H[40],in2_L[8],in2_H[8],in3_L[8],in3_H[8],C1[39],C2[39],C3[
39],C1[40],C2[40],C3[40],redH[40],redL[40]);
    RED_BLOCK6
    block9(in1_L[41],in1_H[41],in2_L[9],in2_H[9],in3_L[9],in3_H[9],C1[40],C2[40],C3[
40],C1[41],C2[41],C3[41],redH[41],redL[41]);
    RED_BLOCK6
    block10(in1_L[42],in1_H[42],in2_L[10],in2_H[10],in3_L[10],in3_H[10],C1[41],C2[41],
C3[41],C1[42],C2[42],C3[42],redH[42],redL[42]);
    RED_BLOCK6
    block11(in1_L[43],in1_H[43],in2_L[11],in2_H[11],in3_L[11],in3_H[11],C1[42],C2[42],
C3[42],C1[43],C2[43],C3[43],redH[43],redL[43]);
    RED_BLOCK6
    block12(in1_L[44],in1_H[44],in2_L[12],in2_H[12],in3_L[12],in3_H[12],C1[43],C2[43],
C3[43],C1[44],C2[44],C3[44],redH[44],redL[44]);
    RED_BLOCK6
    block13(in1_L[45],in1_H[45],in2_L[13],in2_H[13],in3_L[13],in3_H[13],C1[44],C2[44],
C3[44],C1[45],C2[45],C3[45],redH[45],redL[45]);

```

```

    RED_BLOCK6
    block14(in1_L[46], in1_H[46], in2_L[14], in2_H[14], in3_L[14], in3_H[14], C1[45], C2[45],
    C3[45], C1[46], C2[46], C3[46], redH[46], redL[46]);

    RED_BLOCK6
    block15(in1_L[47], in1_H[47], in2_L[15], in2_H[15], in3_L[15], in3_H[15], C1[46], C2[46],
    C3[46], C1[47], C2[47], C3[47], redH[47], redL[47]);

    RED_BLOCK6
    block16(in1_L[48], in1_H[48], in2_L[16], in2_H[16], in3_L[16], in3_H[16], C1[47], C2[47],
    C3[47], C1[48], C2[48], C3[48], redH[48], redL[48]);

    RED_BLOCK6
    block17(in1_L[49], in1_H[49], in2_L[17], in2_H[17], in3_L[17], in3_H[17], C1[48], C2[48],
    C3[48], C1[49], C2[49], C3[49], redH[49], redL[49]);

    RED_BLOCK6
    block18(in1_L[50], in1_H[50], in2_L[18], in2_H[18], in3_L[18], in3_H[18], C1[49], C2[49],
    C3[49], C1[50], C2[50], C3[50], redH[50], redL[50]);

    RED_BLOCK6
    block19(in1_L[51], in1_H[51], in2_L[19], in2_H[19], in3_L[19], in3_H[19], C1[50], C2[50],
    C3[50], C1[51], C2[51], C3[51], redH[51], redL[51]);

    RED_BLOCK6
    block20(in1_L[52], in1_H[52], in2_L[20], in2_H[20], in3_L[20], in3_H[20], C1[51], C2[51],
    C3[51], C1[52], C2[52], C3[52], redH[52], redL[52]);

    RED_BLOCK6
    block21(in1_L[53], in1_H[53], in2_L[21], in2_H[21], in3_L[21], in3_H[21], C1[52], C2[52],
    C3[52], C1[53], C2[53], C3[53], redH[53], redL[53]);

    RED_BLOCK6
    block22(in1_L[54], in1_H[54], in2_L[22], in2_H[22], in3_L[22], in3_H[22], C1[53], C2[53],
    C3[53], C1[54], C2[54], C3[54], redH[54], redL[54]);

    RED_BLOCK6
    block23(in1_L[55], in1_H[55], in2_L[23], in2_H[23], in3_L[23], in3_H[23], C1[54], C2[54],
    C3[54], C1[55], C2[55], C3[55], redH[55], redL[55]);

    RED_BLOCK6
    block24(in1_L[56], in1_H[56], in2_L[24], in2_H[24], in3_L[24], in3_H[24], C1[55], C2[55],
    C3[55], C1[56], C2[56], C3[56], redH[56], redL[56]);

    RED_BLOCK6
    block25(in1_L[57], in1_H[57], in2_L[25], in2_H[25], in3_L[25], in3_H[25], C1[56], C2[56],
    C3[56], C1[57], C2[57], C3[57], redH[57], redL[57]);

    RED_BLOCK6
    block26(in1_L[58], in1_H[58], in2_L[26], in2_H[26], in3_L[26], in3_H[26], C1[57], C2[57],
    C3[57], C1[58], C2[58], C3[58], redH[58], redL[58]);

    RED_BLOCK6
    block27(in1_L[59], in1_H[59], in2_L[27], in2_H[27], in3_L[27], in3_H[27], C1[58], C2[58],
    C3[58], C1[59], C2[59], C3[59], redH[59], redL[59]);

    RED_BLOCK6
    block28(in1_L[60], in1_H[60], in2_L[28], in2_H[28], in3_L[28], in3_H[28], C1[59], C2[59],
    C3[59], C1[60], C2[60], C3[60], redH[60], redL[60]);

    RED_BLOCK6
    block29(in1_L[61], in1_H[61], in2_L[29], in2_H[29], in3_L[29], in3_H[29], C1[60], C2[60],
    C3[60], C1[61], C2[61], C3[61], redH[61], redL[61]);

    RED_BLOCK6
    block30(in1_L[62], in1_H[62], in2_L[30], in2_H[30], in3_L[30], in3_H[30], C1[61], C2[61],
    C3[61], C1[62], C2[62], C3[62], redH[62], redL[62]);

    RED_BLOCK6
    block31(in1_L[63], in1_H[63], in2_L[31], in2_H[31], in3_L[31], in3_H[31], C1[62], C2[62],
    C3[62], C1[63], C2[63], C3[63], redH[63], redL[63]);

    RED_BLOCK6
    block32(in1_L[64], in1_H[64], in2_L[32], in2_H[32], in3_L[32], in3_H[32], C1[63], C2[63],
    C3[63], C1[64], C2[64], C3[64], redH[64], redL[64]);

```

```

    RED_BLOCK6
    block33(in4_L[65],in4_H[65],in2_L[33],in2_H[33],in3_L[33],in3_H[33],C1[64],C2[64],C3[64],C1[65],C2[65],C3[65],redH[65],redL[65]);
    RED_BLOCK6
    block34(in4_L[66],in4_H[66],in2_L[34],in2_H[34],in3_L[34],in3_H[34],C1[65],C2[65],C3[65],C1[66],C2[66],C3[66],redH[66],redL[66]);
    RED_BLOCK6
    block35(in4_L[67],in4_H[67],in2_L[35],in2_H[35],in3_L[35],in3_H[35],C1[66],C2[66],C3[66],C1[67],C2[67],C3[67],redH[67],redL[67]);
    RED_BLOCK6
    block36(in4_L[68],in4_H[68],in2_L[36],in2_H[36],in3_L[36],in3_H[36],C1[67],C2[67],C3[67],C1[68],C2[68],C3[68],redH[68],redL[68]);
    RED_BLOCK6
    block37(in4_L[69],in4_H[69],in2_L[37],in2_H[37],in3_L[37],in3_H[37],C1[68],C2[68],C3[68],C1[69],C2[69],C3[69],redH[69],redL[69]);
    RED_BLOCK6
    block38(in4_L[70],in4_H[70],in2_L[38],in2_H[38],in3_L[38],in3_H[38],C1[69],C2[69],C3[69],C1[70],C2[70],C3[70],redH[70],redL[70]);
    RED_BLOCK6
    block39(in4_L[71],in4_H[71],in2_L[39],in2_H[39],in3_L[39],in3_H[39],C1[70],C2[70],C3[70],C1[71],C2[71],C3[71],redH[71],redL[71]);
    RED_BLOCK6
    block40(in4_L[72],in4_H[72],in2_L[40],in2_H[40],in3_L[40],in3_H[40],C1[71],C2[71],C3[71],C1[72],C2[72],C3[72],redH[72],redL[72]);
    RED_BLOCK6
    block41(in4_L[73],in4_H[73],in2_L[41],in2_H[41],in3_L[41],in3_H[41],C1[72],C2[72],C3[72],C1[73],C2[73],C3[73],redH[73],redL[73]);
    RED_BLOCK6
    block42(in4_L[74],in4_H[74],in2_L[42],in2_H[42],in3_L[42],in3_H[42],C1[73],C2[73],C3[73],C1[74],C2[74],C3[74],redH[74],redL[74]);
    RED_BLOCK6
    block43(in4_L[75],in4_H[75],in2_L[43],in2_H[43],in3_L[43],in3_H[43],C1[74],C2[74],C3[74],C1[75],C2[75],C3[75],redH[75],redL[75]);
    RED_BLOCK6
    block44(in4_L[76],in4_H[76],in2_L[44],in2_H[44],in3_L[44],in3_H[44],C1[75],C2[75],C3[75],C1[76],C2[76],C3[76],redH[76],redL[76]);
    RED_BLOCK6
    block45(in4_L[77],in4_H[77],in2_L[45],in2_H[45],in3_L[45],in3_H[45],C1[76],C2[76],C3[76],C1[77],C2[77],C3[77],redH[77],redL[77]);
    RED_BLOCK6
    block46(in4_L[78],in4_H[78],in2_L[46],in2_H[46],in3_L[46],in3_H[46],C1[77],C2[77],C3[77],C1[78],C2[78],C3[78],redH[78],redL[78]);
    RED_BLOCK6
    block47(in4_L[79],in4_H[79],in2_L[47],in2_H[47],in3_L[47],in3_H[47],C1[78],C2[78],C3[78],C1[79],C2[79],C3[79],redH[79],redL[79]);
    RED_BLOCK6
    block48(in4_L[80],in4_H[80],in2_L[48],in2_H[48],in3_L[48],in3_H[48],C1[79],C2[79],C3[79],C1[80],C2[80],C3[80],redH[80],redL[80]);
    RED_BLOCK6
    block49(in4_L[81],in4_H[81],in2_L[49],in2_H[49],in3_L[49],in3_H[49],C1[80],C2[80],C3[80],C1[81],C2[81],C3[81],redH[81],redL[81]);
    RED_BLOCK6
    block50(in4_L[82],in4_H[82],in2_L[50],in2_H[50],in3_L[50],in3_H[50],C1[81],C2[81],C3[81],C1[82],C2[82],C3[82],redH[82],redL[82]);
    RED_BLOCK6
    block51(in4_L[83],in4_H[83],in2_L[51],in2_H[51],in3_L[51],in3_H[51],C1[82],C2[82],C3[82],C1[83],C2[83],C3[83],redH[83],redL[83]);

```



```

    RED_BLOCK6
    block52(in4_L[84],in4_H[84],in2_L[52],in2_H[52],in3_L[52],in3_H[52],C1[83],C2[83],C3[83],C1[84],C2[84],C3[84],redH[84],redL[84]);

    RED_BLOCK6
    block53(in4_L[85],in4_H[85],in2_L[53],in2_H[53],in3_L[53],in3_H[53],C1[84],C2[84],C3[84],C1[85],C2[85],C3[85],redH[85],redL[85]);

    RED_BLOCK6
    block54(in4_L[86],in4_H[86],in2_L[54],in2_H[54],in3_L[54],in3_H[54],C1[85],C2[85],C3[85],C1[86],C2[86],C3[86],redH[86],redL[86]);

    RED_BLOCK6
    block55(in4_L[87],in4_H[87],in2_L[55],in2_H[55],in3_L[55],in3_H[55],C1[86],C2[86],C3[86],C1[87],C2[87],C3[87],redH[87],redL[87]);

    RED_BLOCK6
    block56(in4_L[88],in4_H[88],in2_L[56],in2_H[56],in3_L[56],in3_H[56],C1[87],C2[87],C3[87],C1[88],C2[88],C3[88],redH[88],redL[88]);

    RED_BLOCK6
    block57(in4_L[89],in4_H[89],in2_L[57],in2_H[57],in3_L[57],in3_H[57],C1[88],C2[88],C3[88],C1[89],C2[89],C3[89],redH[89],redL[89]);

    RED_BLOCK6
    block58(in4_L[90],in4_H[90],in2_L[58],in2_H[58],in3_L[58],in3_H[58],C1[89],C2[89],C3[89],C1[90],C2[90],C3[90],redH[90],redL[90]);

    RED_BLOCK6
    block59(in4_L[91],in4_H[91],in2_L[59],in2_H[59],in3_L[59],in3_H[59],C1[90],C2[90],C3[90],C1[91],C2[91],C3[91],redH[91],redL[91]);

    RED_BLOCK6
    block60(in4_L[92],in4_H[92],in2_L[60],in2_H[60],in3_L[60],in3_H[60],C1[91],C2[91],C3[91],C1[92],C2[92],C3[92],redH[92],redL[92]);

    RED_BLOCK6
    block61(in4_L[93],in4_H[93],in2_L[61],in2_H[61],in3_L[61],in3_H[61],C1[92],C2[92],C3[92],C1[93],C2[93],C3[93],redH[93],redL[93]);

    RED_BLOCK6
    block62(in4_L[94],in4_H[94],in2_L[62],in2_H[62],in3_L[62],in3_H[62],C1[93],C2[93],C3[93],C1[94],C2[94],C3[94],redH[94],redL[94]);

    RED_BLOCK6
    block63(in4_L[95],in4_H[95],in2_L[63],in2_H[63],in3_L[63],in3_H[63],C1[94],C2[94],C3[94],C1[95],C2[95],C3[95],redH[95],redL[95]);

    RED_BLOCK6
    block64(in4_L[96],in4_H[96],in2_L[64],in2_H[64],in3_L[64],in3_H[64],C1[95],C2[95],C3[95],C1[96],C2[96],C3[96],redH[96],redL[96]);

    // module RED_BLOCK2 (S1,S0,CIN1,CIN2,CIN3,COUT,CP,SP);
    RED_BLOCK2
    block65(in4_L[97],in4_H[97],C1[96],C2[96],C3[96],COUT,redH[97],redL[97]);

    // module FULLADDER (a,b,c,SUM,CARRY);
    FULLADDER block66(in4_L[98],in4_H[98],COUT,redL[98],redH[98]);

    // module HALFADDER (a,b,SUM,CARRY);
    HALFADDER block67(in4_L[99],in4_H[99],redL[99],redH[99]);
    HALFADDER block68(in4_L[100],in4_H[100],redL[100],redH[100]);
    HALFADDER block69(in4_L[101],in4_H[101],redL[101],redH[101]);
    HALFADDER block70(in4_L[102],in4_H[102],redL[102],redH[102]);
    HALFADDER block71(in4_L[103],in4_H[103],redL[103],redH[103]);
    HALFADDER block72(in4_L[104],in4_H[104],redL[104],redH[104]);
    HALFADDER block73(in4_L[105],in4_H[105],redL[105],redH[105]);
    HALFADDER block74(in4_L[106],in4_H[106],redL[106],redH[106]);
    HALFADDER block75(in4_L[107],in4_H[107],redL[107],redH[107]);
    HALFADDER block76(in4_L[108],in4_H[108],redL[108],redH[108]);

```

```

HALFADDER block77(in4_L[109],in4_H[109],redL[109],redH[109]);
HALFADDER block78(in4_L[110],in4_H[110],redL[110],redH[110]);
HALFADDER block79(in4_L[111],in4_H[111],redL[111],redH[111]);
HALFADDER block80(in4_L[112],in4_H[112],redL[112],redH[112]);
HALFADDER block81(in4_L[113],in4_H[113],redL[113],redH[113]);
HALFADDER block82(in4_L[114],in4_H[114],redL[114],redH[114]);
HALFADDER block83(in4_L[115],in4_H[115],redL[115],redH[115]);
HALFADDER block84(in4_L[116],in4_H[116],redL[116],redH[116]);
HALFADDER block85(in4_L[117],in4_H[117],redL[117],redH[117]);
HALFADDER block86(in4_L[118],in4_H[118],redL[118],redH[118]);
HALFADDER block87(in4_L[119],in4_H[119],redL[119],redH[119]);
HALFADDER block88(in4_L[120],in4_H[120],redL[120],redH[120]);
HALFADDER block89(in4_L[121],in4_H[121],redL[121],redH[121]);
HALFADDER block90(in4_L[122],in4_H[122],redL[122],redH[122]);
HALFADDER block91(in4_L[123],in4_H[123],redL[123],redH[123]);
HALFADDER block92(in4_L[124],in4_H[124],redL[124],redH[124]);
HALFADDER block93(in4_L[125],in4_H[125],redL[125],redH[125]);
HALFADDER block94(in4_L[126],in4_H[126],redL[126],redH[126]);
HALFADDER block95(in4_L[127],in4_H[127],redL[127],redH[127]);
HALFADDER block96(in4_L[128],in4_H[128],redL[128],redH[128]);

// The first 32 bits are taken directly from the inputs
assign out1_L [64:1] = {redL [64:33] , in1_L [32:1]};
assign out1_H [64:1] = redL [128:65];

// There is no carry signal in the 32nd position thus, the signal is grounded.
assign out2_L [64:1] = {redH [63:33] , 1'b0 , in1_H [32:1]};
assign out2_H [64:1] = redH [127:64];

endmodule // REDUCTION

```

Appendix D

Recursive Multiplier Base-Multiplier Analysis

k	Cell Count		Interconnects		Stage Count		6.2 Reduction Circuit	
	NAFA	NDFA	IAFA	IDFA	SAFA	SDFA	NRC	NRI
4	5	6	18	12	3	2	29	49
5	16	12	32	24	4	3	36	61
6	25	20	50	40	5	3	43	73
7	36	30	72	60	6	4	50	85
8	49	42	98	84	7	4	57	97
9	64	56	128	112	8	4	64	109
10	81	72	162	144	9	5	71	121
11	100	90	200	180	10	5	78	133
12	121	110	242	220	11	5	85	145
13	144	132	288	264	12	5	92	157
14	169	156	338	312	13	6	99	169
15	196	182	392	364	14	6	106	181
16	225	210	450	420	15	6	113	193
17	256	240	512	480	16	6	120	205
18	289	272	578	544	17	6	127	217
19	324	306	648	612	18	6	134	229
20	361	342	722	684	19	7	141	241
21	400	380	800	760	20	7	148	253
22	441	420	882	840	21	7	155	265
23	484	462	968	924	22	7	162	277
24	529	506	1058	1012	23	7	169	289
25	576	552	1152	1104	24	7	176	301
26	625	600	1250	1200	25	7	183	313
27	676	650	1352	1300	26	7	190	325
28	729	702	1458	1404	27	7	197	337
29	784	756	1568	1512	28	8	204	349
30	841	812	1682	1624	29	8	211	361
31	900	870	1800	1740	30	8	218	373
32	961	930	1922	1860	31	8	225	385
33	1024	992	2048	1984	32	8	232	397
34	1089	1056	2178	2112	33	8	239	409
35	1156	1122	2312	2244	34	8	246	421
36	1225	1190	2450	2380	35	8	253	433
37	1296	1260	2592	2520	36	8	260	445
38	1369	1332	2738	2664	37	8	267	457
39	1444	1406	2888	2812	38	8	274	469
40	1521	1482	3042	2964	39	8	281	481
41	1600	1560	3200	3120	40	8	288	493
42	1681	1640	3362	3280	41	8	295	505
43	1764	1722	3528	3444	42	9	302	517
44	1849	1806	3698	3612	43	9	309	529
45	1936	1892	3872	3784	44	9	316	541
46	2025	1980	4050	3960	45	9	323	553
47	2116	2070	4232	4140	46	9	330	565
48	2209	2162	4418	4324	47	9	337	577
49	2304	2256	4608	4512	48	9	344	589
50	2401	2352	4802	4704	49	9	351	601
51	2500	2450	5000	4900	50	9	358	613
52	2601	2550	5202	5100	51	9	365	625
53	2704	2652	5408	5304	52	9	372	637
54	2809	2756	5616	5512	53	9	379	649
55	2916	2862	5832	5724	54	9	386	661
56	3025	2970	6050	5940	55	9	393	673
57	3136	3080	6272	6160	56	9	400	685
58	3249	3192	6498	6384	57	9	407	697
59	3364	3306	6728	6612	58	9	414	709
60	3481	3422	6962	6844	59	9	421	721
61	3600	3540	7200	7080	60	9	428	733
62	3721	3660	7442	7320	61	9	435	745
63	3844	3782	7688	7564	62	9	442	757
64	3969	3906	7938	7812	63	10	449	769

Appendix E

Component Breakdown of the Reconfigurable Multiplier

Reconfigurable Multiplier Breakdown from Synopsys:

Top I/O interface Module

Report : area
Design : RecursiveMultiplierV4
Version: 2001.08-SP2
Date : Tue Apr 1 12:47:38 2003

Number of ports: 260
Number of nets: 644
Number of cells: 385
Number of references: 4

Combinational area: 424568.093750
Noncombinational area: 18754.662109
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area: 443322.750000
Total area: undefined

Cell Internal Power = 78.0747 mW (56%)
Net Switching Power = 61.5631 mW (44%)

Total Dynamic Power = 139.6377 mW (100%)
Cell Leakage Power = 25.8288 uW

Report : timing
-path full
-delay max
-max_paths 1
Design : RecursiveMultiplierV4
Version: 2001.08-SP2
Date : Tue Apr 1 12:47:44 2003

Point	Incr	Path
OUT_HIGH_reg_64_/CK (DFFRPQ1)	0.00	0.00 r
OUT_HIGH_reg_64_/Q (DFFRPQ1)	0.37	0.37 f
U649/Z (BUFBD32)	0.35	0.72 f
OUT_HIGH_64_ (out)	0.00	0.72 f
data arrival time		0.72

(Path is unconstrained)

RecMult Module (actual multiplier module)

Report : area
Design : RecMult
Version: 2001.08-SP2
Date : Wed Apr 2 13:58:29 2003

Number of ports: 258
Number of nets: 2544
Number of cells: 315
Number of references: 20

Combinational area: 381895.593750

Noncombinational area: 0.000000
 Net Interconnect area: undefined (Wire load has zero net area)
 Total cell area: 381895.593750
 Total area: undefined

Cell Internal Power = 322.6490 mW (53%)
 Net Switching Power = 290.7263 mW (47%)

 Total Dynamic Power = 613.3752 mW (100%)
 Cell Leakage Power = 20.4450 uW

Report : timing

-path full
 -delay max
 -max_paths 1

Design : RecMult

Version: 2001.08-SP2

Date : Wed Apr 2 10:43:41 2003

Point	Incr	Path
input external delay	0.00	0.00 f
CONTROL_2_ (in)	0.00	0.00 f
mux1/control_1_ (MUX1)	0.00	0.00 f
mux1/U23/Z (INVD1)	0.45	0.45 r
mux1/U39/Z (AND2D1)	0.29	0.73 r
mux1/output1_31_ (MUX1)	0.00	0.73 r
multiplier_1/in_low_31_ (INTERMUL_T_0)	0.00	0.73 r
multiplier_1/inter_mult/ina_31_ (DW02_multp_instance_0)	0.00	0.73 r
multiplier_1/inter_mult/U1/a_31_ (DW02_multp_instance_0_DW02_multp_32_32_66_0)	0.00	0.73 r
multiplier_1/inter_mult/U1/U2_BC/A_31_ (DW02_multp_instance_0_DW02_booth_32_1_0)	0.00	0.73 r
multiplier_1/inter_mult/U1/U2_BC/ENC_15/c (DW02_multp_instance_0_DW_bthenc_16)	0.00	0.73 r
multiplier_1/inter_mult/U1/U2_BC/ENC_15/U4/Z (EXNOR2D1)	0.48	1.21 f
multiplier_1/inter_mult/U1/U2_BC/ENC_15/U10/Z (NOR2D1)	0.22	1.44 r
multiplier_1/inter_mult/U1/U2_BC/ENC_15/shift1 (DW02_multp_instance_0_DW_bthenc_16)	0.00	1.44 r
multiplier_1/inter_mult/U1/U2_BC/A_coded_46_ (DW02_multp_instance_0_DW02_booth_32_1_0)	0.00	1.44 r
multiplier_1/inter_mult/U1/U231/Z (BUFD1)	0.79	2.23 r
multiplier_1/inter_mult/U1/U256/Z (NAN2D1)	0.14	2.37 f
multiplier_1/inter_mult/U1/U1036/Z (NAN2D1)	0.16	2.53 r
multiplier_1/inter_mult/U1/U1035/Z (EXOR2D1)	0.43	2.96 f
multiplier_1/inter_mult/U1/U2_WT/pp_array_515_ (DW02_multp_instance_0_DW_mtree_32_32_0)	0.00	2.96 f
multiplier_1/inter_mult/U1/U2_WT/U351/CO (ADFULD1)	0.29	3.26 f
multiplier_1/inter_mult/U1/U2_WT/U38/Z (MUXB2D0)	0.64	3.90 r
multiplier_1/inter_mult/U1/U2_WT/U506/S (ADFULD1)	0.63	4.53 r
multiplier_1/inter_mult/U1/U2_WT/U165/S (ADFULD1)	0.44	4.97 f
multiplier_1/inter_mult/U1/U2_WT/U322/S (ADHALFD1)	0.41	5.39 r
multiplier_1/inter_mult/U1/U2_WT/U144/S (ADFULD1)	0.52	5.90 f
multiplier_1/inter_mult/U1/U2_WT/out0_36_ (DW02_multp_instance_0_DW_mtree_32_32_0)	0.00	5.90 f
multiplier_1/inter_mult/U1/out0_36_ (DW02_multp_instance_0_DW02_multp_32_32_66_0)	0.00	5.90 f
multiplier_1/inter_mult/prod0_36_ (DW02_multp_instance_0)	0.00	5.90 f
multiplier_1/out_low_36_ (INTERMUL_T_0)	0.00	5.90 f
redmux/input1_36_ (MUX_Reduction)	0.00	5.90 f
redmux/U546/Z (AND2D2)	0.18	6.09 f
redmux/output1_36_ (MUX_Reduction)	0.00	6.09 f
Reduction_6_to_2/in1_L_37_ (REDUCTION)	0.00	6.09 f
Reduction_6_to_2/block5/C2 (RED_BLOCK6_24)	0.00	6.09 f
Reduction_6_to_2/block5/fa2/a (FULLADDER_98)	0.00	6.09 f
Reduction_6_to_2/block5/fa2/U8/Z (EXOR3D1)	0.51	6.60 f
Reduction_6_to_2/block5/fa2/SUM (FULLADDER_98)	0.00	6.60 f
Reduction_6_to_2/block5/fa3/c (FULLADDER_97)	0.00	6.60 f
Reduction_6_to_2/block5/fa3/U7/Z (OR2D1)	0.23	6.82 f
Reduction_6_to_2/block5/fa3/U9/Z (OAI22M22D1)	0.31	7.13 f
Reduction_6_to_2/block5/fa3/CARRY (FULLADDER_97)	0.00	7.13 f
Reduction_6_to_2/block5/COUT3 (RED_BLOCK6_24)	0.00	7.13 f
Reduction_6_to_2/block6/CIN3 (RED_BLOCK6_57)	0.00	7.13 f

Reduction_6_to_2/block6/fa4/b (FULLADDER_230)	0.00	7.13 f
Reduction_6_to_2/block6/fa4/U9/Z (EXOR3D2)	0.45	7.58 r
Reduction_6_to_2/block6/fa4/SUM (FULLADDER_230)	0.00	7.58 r
Reduction_6_to_2/block6/SP (RED_BLOCK6_57)	0.00	7.58 r
Reduction_6_to_2/out1_L_38_ (REDUCTION)	0.00	7.58 r
U190/Z (AO222D2)	0.20	7.78 r
FinalLow/IN1_37_ (FASTADDER_1)	0.00	7.78 r
FinalLow/add_519/B_37_ (FASTADDER_1_DW01_add_65_1)	0.00	7.78 r
FinalLow/add_519/U185/Z (INVD1)	0.08	7.86 f
FinalLow/add_519/U222/Z (NAN2D2)	0.13	7.99 r
FinalLow/add_519/U510/Z (INVD1)	0.09	8.08 f
FinalLow/add_519/U20/Z (NOR2D1)	0.18	8.26 r
FinalLow/add_519/U19/Z (NOR2D2)	0.06	8.33 f
FinalLow/add_519/U43/Z (NOR2D1)	0.20	8.52 r
FinalLow/add_519/U42/Z (NOR2D2)	0.07	8.59 f
FinalLow/add_519/U274/Z (NOR2D2)	0.14	8.73 r
FinalLow/add_519/U273/Z (NOR2D2)	0.08	8.81 f
FinalLow/add_519/add_519_cell_19613_syn154152/Z (INVD2)	0.10	8.91 r
FinalLow/add_519/add_519_cell_19613_syn151559/Z (NOR2D4)	0.07	8.98 f
FinalLow/add_519/add_519_cell_19613_syn151561/Z (NOR2D4)	0.08	9.06 r
FinalLow/add_519/U46/Z (NOR2D2)	0.06	9.12 f
FinalLow/add_519/U45/Z (NOR2D2)	0.12	9.24 r
FinalLow/add_519/add_519_cell_19613_syn155716/Z (NAN3M1D2)	0.19	9.43 r
FinalLow/add_519/add_519_cell_19613_syn155718/Z (NAN3M1D2)	0.08	9.51 f
FinalLow/add_519/U571/Z (NAN2D1)	0.15	9.67 r
FinalLow/add_519/add_519_cell_19613_syn155721/Z (INVD2)	0.09	9.76 f
FinalLow/add_519/add_519_cell_19613_syn150776/Z (NOR3D4)	0.19	9.95 r
FinalLow/add_519/add_519_cell_19613_syn151586/Z (NOR2D4)	0.07	10.02 f
FinalLow/add_519/add_519_cell_19613_syn151592/Z (NOR2D4)	0.13	10.15 r
FinalLow/add_519/add_519_cell_19613_syn151597/Z (NOR2D4)	0.06	10.21 f
FinalLow/add_519/add_519_cell_19613_syn151604/Z (NOR2D4)	0.08	10.29 r
FinalLow/add_519/U86/Z (NOR2D2)	0.06	10.35 f
FinalLow/add_519/U85/Z (NOR2D2)	0.14	10.49 r
FinalLow/add_519/add_519_cell_19613_syn155738/Z (INVD2)	0.06	10.54 f
FinalLow/add_519/add_519_cell_19090_TEMP_G_2_0_3_/Z (NAN2M1D2)	0.16	10.70 r
FinalLow/add_519/SUM_64_ (FASTADDER_1_DW01_add_65_1)	0.00	10.70 r
FinalLow/C_OUT (FASTADDER_1)	0.00	10.70 r
FinalHigh/C_IN (FASTADDER_0)	0.00	10.70 r
FinalHigh/add_519/CI (FASTADDER_0_DW01_add_65_1)	0.00	10.70 r
FinalHigh/add_519/U112/Z (INVD2)	0.10	10.80 f
FinalHigh/add_519/_cell_20115_syn158921/Z (NOR3D4)	0.17	10.97 r
FinalHigh/add_519/_cell_20115_syn160706/Z (NOR2D4)	0.08	11.05 f
FinalHigh/add_519/_cell_20115_syn162381/Z (NAN2D4)	0.16	11.20 r
FinalHigh/add_519/_cell_20115_syn162347/Z (NAN3M1D2)	0.12	11.33 f
FinalHigh/add_519/_cell_20115_syn162349/Z (NAN2D4)	0.19	11.52 r
FinalHigh/add_519/_cell_20115_syn162316/Z (NAN3M1D2)	0.10	11.62 f
FinalHigh/add_519/_cell_20115_syn162318/Z (NAN3M1D2)	0.13	11.75 r
FinalHigh/add_519/_cell_20115_syn161684/Z (NAN2M1D2)	0.09	11.84 f
FinalHigh/add_519/_cell_20115_syn161686/Z (NAN2D2)	0.10	11.94 r
FinalHigh/add_519/_cell_20115_syn161687/Z (INVD2)	0.08	12.02 f
FinalHigh/add_519/_cell_20115_syn157192/Z (NOR2D4)	0.08	12.10 r
FinalHigh/add_519/U587/Z (INVD2)	0.05	12.15 f
FinalHigh/add_519/U586/Z (NAN2D2)	0.11	12.27 r
FinalHigh/add_519/_cell_20115_syn161703/Z (NAN2D2)	0.07	12.34 f
FinalHigh/add_519/_cell_20115_syn161705/Z (NAN2D2)	0.12	12.46 r
FinalHigh/add_519/_cell_20115_syn161711/Z (NAN2D2)	0.07	12.54 f
FinalHigh/add_519/_cell_20115_syn161713/Z (NAN2D2)	0.10	12.64 r
FinalHigh/add_519/_cell_20115_syn161714/Z (INVD2)	0.08	12.72 f
FinalHigh/add_519/_cell_20115_syn157212/Z (NOR2D4)	0.10	12.82 r
FinalHigh/add_519/_cell_20115_syn161718/Z (INVD4)	0.04	12.86 f
FinalHigh/add_519/U305/Z (NAN2D2)	0.09	12.95 r
FinalHigh/add_519/_cell_20115_syn161723/Z (INVD2)	0.08	13.03 f
FinalHigh/add_519/_cell_20115_syn157223/Z (NOR2D4)	0.08	13.11 r
FinalHigh/add_519/U355/Z (INVD2)	0.05	13.17 f

FinalHigh/add_519/U354/Z (NAN2D2)	0.09	13.26 r
FinalHigh/add_519/_cell_20115_syn161734/Z (INVD2)	0.05	13.31 f
FinalHigh/add_519/U353/Z (NOR2D1)	0.14	13.45 r
FinalHigh/add_519/U612/Z (NOR2D1)	0.09	13.54 f
FinalHigh/add_519/U750/Z (EXNOR2D1)	0.38	13.91 f
FinalHigh/add_519/SUM_63_ (FASTADDER_0_DW01_add_65_1)	0.00	13.91 f
FinalHigh/OUT_63_ (FASTADDER_0)	0.00	13.91 f
OUT_HIGH_64_ (out)	0.00	13.91 f
data arrival time		13.91

Fast Adder Module:

Report : area

Design : FASTADDER_1

Version: 2001.08-SP2

Date : Tue Apr 1 17:55:05 2003

Number of ports: 194

Number of nets: 194

Number of cells: 1

Number of references: 1

Combinational area: 10944.590820

Noncombinational area: 0.000000

Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 10944.590820

Total area: undefined

Cell Internal Power = 9.4226 mW (48%)

Net Switching Power = 10.3596 mW (52%)

Total Dynamic Power = 19.7821 mW (100%)

Cell Leakage Power = 502.1571 nW

Report : timing

-path full

-delay max

-max_paths 1

Design : FASTADDER_1

Version: 2001.08-SP2

Date : Tue Apr 1 17:55:12 2003

Point	Incr	Path
input external delay	0.00	0.00 f
IN1_0_ (in)	0.00	0.00 f
add_519/B_0_ (FASTADDER_1_DW01_add_65_1)	0.00	0.00 f
add_519/U494/Z (NAN2D1)	0.12	0.12 r
add_519/U495/Z (INVD1)	0.09	0.21 f
add_519/U203/Z (NAN2D1)	0.09	0.30 r
add_519/U617/Z (AND2D0)	0.23	0.53 r
add_519/U111/Z (NOR2D1)	0.08	0.61 f
add_519/U440/Z (NOR2D1)	0.20	0.81 r
add_519/U615/Z (OR2D0)	0.21	1.02 r
add_519/U413/Z (NAN3D2)	0.14	1.16 f
add_519/add_519__cell_19613_syn155045/Z (NAN3M1D2)	0.10	1.26 r
add_519/U63/Z (NAN2D1)	0.08	1.34 f
add_519/U62/Z (NAN3M1D1)	0.14	1.48 r
add_519/U619/Z (AND2D0)	0.32	1.80 r
add_519/add_519__cell_19613_syn151301/Z (NOR3D2)	0.08	1.88 f
add_519/U175/Z (NOR2D1)	0.18	2.05 r
add_519/U174/Z (NOR3D2)	0.07	2.12 f
add_519/U475/Z (INVD1)	0.09	2.21 r
add_519/U474/Z (NAN2D1)	0.08	2.29 f
add_519/U473/Z (INVD1)	0.17	2.46 r
add_519/add_519__cell_19613_syn151324/Z (NOR3D2)	0.10	2.55 f
add_519/add_519__cell_19613_syn151326/Z (NOR2D4)	0.09	2.64 r
add_519/U121/Z (NOR2D2)	0.05	2.69 f
add_519/U446/Z (INVD1)	0.07	2.77 r
add_519/U631/Z (AND2D0)	0.35	3.12 r
add_519/add_519__cell_19613_syn151420/Z (NOR3D2)	0.11	3.23 f
add_519/add_519__cell_19613_syn151422/Z (NOR3D4)	0.21	3.44 r
add_519/add_519__cell_19613_syn151442/Z (NOR3D4)	0.07	3.51 f
add_519/add_519__cell_19613_syn151444/Z (NOR2D4)	0.10	3.62 r
add_519/add_519__cell_19613_syn151445/Z (NOR2D4)	0.06	3.68 f

add_519/U107/Z (INVD1)	0.08	3.76 r
add_519/U132/Z (NAN2D1)	0.12	3.88 f
add_519/add_519_cell_19613_syn155714/Z (NAN2D2)	0.10	3.98 r
add_519/add_519_cell_19613_syn155716/Z (NAN3M1D2)	0.08	4.06 f
add_519/add_519_cell_19613_syn155718/Z (NAN3M1D2)	0.09	4.15 r
add_519/U571/Z (NAN2D1)	0.10	4.25 f
add_519/add_519_cell_19613_syn155721/Z (INVD2)	0.12	4.37 r
add_519/add_519_cell_19613_syn150776/Z (NOR3D4)	0.07	4.44 f
add_519/U583/Z (INVD1)	0.09	4.53 r
add_519/U635/Z (AND2D0)	0.28	4.80 r
add_519/U362/Z (NOR2D1)	0.09	4.89 f
add_519/U601/Z (NOR2D1)	0.20	5.08 r
add_519/U636/Z (OR2D0)	0.18	5.26 r
add_519/U576/Z (NAN2D1)	0.11	5.37 f
add_519/U261/Z (NAN2D1)	0.11	5.48 r
add_519/U541/Z (NAN2D1)	0.11	5.59 f
add_519/U280/Z (NAN2D1)	0.10	5.69 r
add_519/U637/Z (AND2D0)	0.23	5.91 r
add_519/U378/Z (NOR2D1)	0.09	6.01 f
add_519/U613/Z (NOR2D1)	0.12	6.12 r
add_519/U726/Z (EXNOR2D1)	0.38	6.50 f
add_519/SUM_63_ (FASTADDER_1_DW01_add_65_1)	0.00	6.50 f
OUT_63_ (out)	0.00	6.50 f
data arrival time		6.50

Majority Voter Module:

 Report : area
 Design : MAJORITY
 Version: 2001.08-SP2
 Date : Tue Apr 1 17:52:00 2003

Number of ports: 512
 Number of nets: 640
 Number of cells: 256
 Number of references: 2

Combinational area: 5724.288086
 Noncombinational area: 0.000000
 Net Interconnect area: undefined (Wire load has zero net area)
 Total cell area: 5724.288086
 Total area: undefined

Cell Internal Power = 5.7532 mW (52%)
 Net Switching Power = 5.2250 mW (48%)

 Total Dynamic Power = 10.9782 mW (100%)
 Cell Leakage Power = 298.5331 nW

 Report : timing
 -path full
 -delay max
 -max_paths 1
 Design : MAJORITY
 Version: 2001.08-SP2
 Date : Tue Apr 1 17:52:08 2003

Point	Incr	Path
input external delay	0.00	0.00 f
low1_64_ (in)	0.00	0.00 f
U1157/Z (OR2D1)	0.19	0.19 f
U1195/Z (OAI22M22D1)	0.24	0.43 f
output_low_64_ (out)	0.00	0.43 f
data arrival time		0.43

6:2 Reduction Module:

 Report : area
 Design : REDUCTION
 Version: 2001.08-SP2
 Date : Tue Apr 1 17:56:13 2003

Number of ports: 768
 Number of nets: 895


```

Number of cells:          96
Number of references:     96

Combinational area:      28670.464844
Noncombinational area:   0.000000
Net Interconnect area:   undefined (Wire load has zero net area)
Total cell area:         28670.464844
Total area:              undefined

```

```

Cell Internal Power = 31.7210 mW (61%)
Net Switching Power = 20.0382 mW (39%)
-----
Total Dynamic Power = 51.7592 mW (100%)
Cell Leakage Power  = 1.7344 uW

```

```

Report : timing
-path full
-delay max
-max_paths 1
Design : REDUCTION
Version: 2001.08-SP2
Date : Tue Apr 1 17:56:27 2003
*****

```

Point	Incr	Path
input external delay	0.00	0.00 f
in1_L_37_ (in)	0.00	0.00 f
block5/C2 (RED_BLOCK6_24)	0.00	0.00 f
block5/fa2/a (FULLADDER_98)	0.00	0.00 f
block5/fa2/U8/Z (EXOR3D1)	0.48	0.48 f
block5/fa2/SUM (FULLADDER_98)	0.00	0.48 f
block5/fa3/c (FULLADDER_97)	0.00	0.48 f
block5/fa3/U7/Z (OR2D1)	0.22	0.69 f
block5/fa3/U9/Z (OAI22M22D1)	0.29	0.99 f
block5/fa3/CARRY (FULLADDER_97)	0.00	0.99 f
block5/COUT3 (RED_BLOCK6_24)	0.00	0.99 f
block6/CIN3 (RED_BLOCK6_57)	0.00	0.99 f
block6/fa4/b (FULLADDER_230)	0.00	0.99 f
block6/fa4/U9/Z (EXOR3D2)	0.48	1.46 f
block6/fa4/SUM (FULLADDER_230)	0.00	1.46 f
block6/SP (RED_BLOCK6_57)	0.00	1.46 f
out1_L_38_ (out)	0.00	1.46 f
data arrival time		1.46

6:2 Reduction Cell MUX Module:

```

*****
Report : area
Design : MUX_Reduction
Version: 2001.08-SP2
Date : Wed Apr 2 12:53:25 2003
*****

```

```

Number of ports:          1026
Number of nets:           1075
Number of cells:          561
Number of references:     5

Combinational area:      9025.444336
Noncombinational area:   0.000000
Net Interconnect area:   undefined (Wire load has zero net area)
Total cell area:         9025.444336
Total area:              undefined

```

```

Cell Internal Power = 5.1077 mW (44%)
Net Switching Power = 6.4080 mW (56%)
-----
Total Dynamic Power = 11.5157 mW (100%)
Cell Leakage Power  = 514.8147 nW

```

```

Report : timing
-path full
-delay max
-max_paths 1
Design : MUX_Reduction
Version: 2001.08-SP2
Date : Wed Apr 2 12:53:46 2003

```

Point	Incr	Path
input external delay	0.00	0.00 f
control_0_ (in)	0.00	0.00 f
U518/Z (OR2D1)	0.21	0.21 f
U538/Z (INVD1)	0.17	0.38 r
U542/Z (BUFD1)	0.22	0.60 r
U403/Z (BUFD1)	0.59	1.19 r
U402/Z (AND2D1)	0.19	1.39 r
output1_31_ (out)	0.00	1.39 r
data arrival time		1.39

Majority Voter MUX module:

 Report : area
 Design : MUX_Majority
 Version: 2001.08-SP2
 Date : Wed Apr 2 12:54:02 2003

Number of ports: 770
 Number of nets: 807
 Number of cells: 421
 Number of references: 5

Combinational area: 6728.423828
 Noncombinational area: 0.000000
 Net Interconnect area: undefined (Wire load has zero net area)
 Total cell area: 6728.423828
 Total area: undefined

Cell Internal Power = 3.7859 mW (44%)
 Net Switching Power = 4.7839 mW (56%)

 Total Dynamic Power = 8.5698 mW (100%)
 Cell Leakage Power = 378.6657 nW

 Report : timing
 -path full
 -delay max
 -max_paths 1
 Design : MUX_Majority
 Version: 2001.08-SP2
 Date : Wed Apr 2 12:54:16 2003

Point	Incr	Path
input external delay	0.00	0.00 f
control_0_ (in)	0.00	0.00 f
U187/Z (NAN2M1D1)	0.16	0.16 f
U414/Z (INVD1)	0.22	0.38 r
U386/Z (BUFD1)	0.21	0.59 r
U214/Z (BUFD1)	0.58	1.16 r
U84/Z (AND2D1)	0.19	1.36 r
output5_23_ (out)	0.00	1.36 r
data arrival time		1.36

(Path is unconstrained)

Appendix F

Simulation Reports and Logs

RECONFIGURABLE MULTIPLIER:

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 12:45:12, 1 April 2003

Design name: RecursiveMultiplierV4

Report file name: RecursiveMultiplierV4.summary

Number of macros: 308

Number of components: 17158

Number of pins: 87986

Number of regular pins: 54424

Number of special pins: 33332

Number of unused pins: 230

Number of nets: 18908

Average number of pins per net: 4.65

Number of subnets: 486

Number of regular pins for subnets: 257

Number of special pins for subnets: 0

Number of virtual pins for subnets: 715

Average number of pins per subnet: 2.00

Number of routing tracks available: 2788

Number of GCELLS per layer: 19328

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 12:45:13, 1 April 2003

Design name: RecursiveMultiplierV4

Report file name: RecursiveMultiplierV4.summary

** NET STATISTICS OF PIN COUNTS

Number of 2-pin nets: 14841

Number of 3-pin nets: 1878

Number of 4-pin nets: 980

Number of 5-pin nets: 66

Number of 6-pin nets: 22

Number of 7-pin nets: 26

Number of 8-pin nets: 22

Number of 9-pin nets: 42

Number of 10-pin nets: 148

Number of 11-pin nets: 137

Number of 12-pin nets: 139

Number of 13-pin nets: 214

Number of 14-pin nets: 124

Number of 15-pin nets: 67

Number of 16-pin nets: 197

Number of 144-pin nets: 1

Number of 257-pin nets: 2

Number of 16667-pin nets: 2

Time: 12:45:13, 1 April 2003

Design name: RecursiveMultiplierV4

Report file name: RecursiveMultiplierV4.summary

** UTILIZATION OF ALL ROW TYPES

Type	Number	Length	Area	%_Row_Space
core Rows	120	89258400	549831744000	
core Cells	16666	71967720	443321155200	80.63

Area of chip: 713332065600 (square DBU)

Area required for all cells: 443321155200 (square DBU)

Area utilization of all cells: 62.15%

*****SILICON_ENSEMBLE WIRING REPORT*****

Time: 12:45:59, 1 April 2003

Design name: RecursiveMultiplierV4
 Report file name: RecursiveMultiplierV4.wires

page 1

** (only DETAILED wiring are reported for REGULAR nets)

Total vias in regular wiring: 141193
 Total segments in regular wiring: 147400
 Total vias in special wiring: 1250
 Total segments in special wiring: 419

LAYER name: metal1
 Total wire length: 162348.96 microns
 Length of regular wires: 62947.68 microns
 Length of special wires: 99401.28 microns
 LAYER name: metal2
 Total wire length: 278562.04 microns
 Length of regular wires: 262702.84 microns
 Length of special wires: 15859.20 microns
 LAYER name: metal3
 Total wire length: 441000.74 microns
 Length of regular wires: 441000.74 microns
 Length of special wires: .00 microns
 LAYER name: metal4
 Total wire length: 334782.50 microns
 Length of regular wires: 334782.50 microns
 Length of special wires: .00 microns
 LAYER name: metal5
 Total wire length: 171144.40 microns
 Length of regular wires: 171144.40 microns
 Length of special wires: .00 microns
 LAYER name: metal6
 Total wire length: 35992.52 microns
 Length of regular wires: 35992.52 microns
 Length of special wires: .00 microns

Total wirelength in regular wiring: 1308570.68 microns
 Total wirelength in special wiring: 115260.48 microns
 Total wirelength in regular+special wiring: 1423831.16 microns

CROSSTALK:

0 nets claimed N coupling caps, but had different no.
 55 nets had more coupling than total capacitance
 0 pin lists with fewer pins than they said they had.
 0 nets with bizarre cap/unit length
 56 nets had implausible coupling ratios.
 2 wires had implausibly small Rs.
 134 rising drive and 134 falling drive implausibly small.
 18908 nets processed, total wire 1318432.0
 0 unknown nets encountered (maybe with repetitions)
 294 constant nets, length 83677.8, (6.3%)
 1 nets had errors, length 244.8, (0.0%)
 Max crosstalk induced timing delta is 1.34n, (0 > 1us)
 Sum of all error voltages is 0.083 volts

Mult64d:

Standard Multiplier Testbench (Booth Recoded Wallace Tree)

 Report : area
 Design : Mult64
 Version: 2000.11-SP1
 Date : Sat Nov 16 20:22:10 2002

Number of ports: 257
 Number of nets: 641
 Number of cells: 385
 Number of references: 3

Combinational area: 358108.718750
 Noncombinational area: 15611.904297
 Net Interconnect area: undefined (Wire load has zero net area)
 Total cell area: 373720.625000
 Total area: undefined

Top Module:

Cell Internal Power = 22.9755 mW (16%)
 Net Switching Power = 122.0531 mW (84%)

Total Dynamic Power = 145.0287 mW (100%)
 Cell Leakage Power = 22.5103 uW

Actual Multiplier:

Cell Internal Power = 410.4787 mW (55%)
 Net Switching Power = 337.1711 mW (45%)

Total Dynamic Power = 747.6498 mW (100%)

Cell Leakage Power = 16.9419 uW

Report : timing

-path full
 -delay max
 -max_paths 1

Design : Mult64

Version: 2000.11-SP1

Date : Sat Nov 16 20:24:23 2002

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (propagated)	0.00	0.00
A_bottom_reg_60_/CK (DFFPQ1)	0.00	0.00 r
A_bottom_reg_60_/Q (DFFPQ1)	0.45	0.45 r
mult/in_low_60_ (Multiplier64)	0.00	0.45 r
mult/mul_42/A_59_ (Multiplier64_DW02_mult_64_64_0)	0.00	0.45 r
mult/mul_42/U2/MULT1/a_59_ (Multiplier64_DW02_multp_64_64_130_0)	0.00	0.45 r
mult/mul_42/U2/MULT1/U1/BC/A_59_ (Multiplier64_DW02_booth_64_1_0)	0.00	0.45 r
mult/mul_42/U2/MULT1/U1/BC/ENC_30/a (Multiplier64_DW_bthenc_8)	0.00	0.45 r
mult/mul_42/U2/MULT1/U1/BC/ENC_30/U9/Z (EXOR2D1)	0.64	1.09 f
mult/mul_42/U2/MULT1/U1/BC/ENC_30/U10/Z (NOR2D1)	0.38	1.47 r
mult/mul_42/U2/MULT1/U1/BC/ENC_30/shift1 (Multiplier64_DW_bthenc_8)	0.00	1.47 r
mult/mul_42/U2/MULT1/U1/BC/A_coded_91_ (Multiplier64_DW02_booth_64_1_0)	0.00	1.47 r
mult/mul_42/U2/MULT1/U5065/Z (BUFD1)	0.79	2.26 r
mult/mul_42/U2/MULT1/U58/Z (NAN2D1)	0.15	2.41 f
mult/mul_42/U2/MULT1/U2258/Z (NAN2D1)	0.16	2.57 r
mult/mul_42/U2/MULT1/U2257/Z (EXOR2D1)	0.49	3.06 f
mult/mul_42/U2/MULT1/U1/WT/pp_array_1990_ (Multiplier64_DW_mtree_64_64_0)	0.00	3.06 f
mult/mul_42/U2/MULT1/U1/WT/U2_1_70_9/S (ADFULD1)	0.53	3.59 f
mult/mul_42/U2/MULT1/U1/WT/U77/Z (MUXB2D0)	0.71	4.30 r
mult/mul_42/U2/MULT1/U1/WT/U4_1_70_2_3/S (ADFULD1)	0.66	4.95 r
mult/mul_42/U2/MULT1/U1/WT/U4_1_70_3_2/S (ADFULD1)	0.52	5.47 f
mult/mul_42/U2/MULT1/U1/WT/U4_1_70_4_1/S (ADFULD1)	0.49	5.97 f
mult/mul_42/U2/MULT1/U1/WT/U4_1_70_5_1/S (ADFULD1)	0.54	6.50 f
mult/mul_42/U2/MULT1/U1/WT/U4_1_70_6_0/S (ADFULD1)	0.52	7.02 f
mult/mul_42/U2/MULT1/U1/WT/U4_0A_1_70_7/S (ADFULD1)	0.52	7.54 r
mult/mul_42/U2/MULT1/U1/WT/out0_70_ (Multiplier64_DW_mtree_64_64_0)	0.00	7.54 r
mult/mul_42/U2/MULT1/out0_70_ (Multiplier64_DW02_multp_64_64_130_0)	0.00	7.54 r
mult/mul_42/U2/ADD1/A_69_ (Multiplier64_DW01_add_127_0)	0.00	7.54 r
mult/mul_42/U2/ADD1/U0_2_69/Z (NOR2D1)	0.16	7.70 f
mult/mul_42/U2/ADD1/U1_5_0_70/Z (NOR2D1)	0.20	7.90 r
mult/mul_42/U2/ADD1/U1_5_1_70/Z (NAN2D1)	0.12	8.01 f
mult/mul_42/U2/ADD1/U1_4_2_70/Z (OAI21D1)	0.29	8.30 r
mult/mul_42/U2/ADD1/U1_4_3_78/Z (AOI21D1)	0.28	8.58 f
mult/mul_42/U2/ADD1/U114/Z (OAI21D1)	0.85	9.43 r
mult/mul_42/U2/ADD1/U1_4_5_95/Z (AOI21D1)	0.18	9.61 f
mult/mul_42/U2/ADD1/U1_4_6_95/Z (OAI21D1)	0.17	9.78 r
mult/mul_42/U2/ADD1/U0_5_96/Z (EXOR2D1)	0.56	10.34 f
mult/mul_42/U2/ADD1/SUM_96_ (Multiplier64_DW01_add_127_0)	0.00	10.34 f
mult/mul_42/PRODUCT_97_ (Multiplier64_DW02_mult_64_64_0)	0.00	10.34 f
mult/out_98_ (Multiplier64)	0.00	10.34 f
OUT1_reg_98_/D (DFFPQ1)	0.00	10.34 f
data arrival time		10.34
clock clk (rise edge)	15.00	15.00
clock network delay (propagated)	0.00	15.00
clock uncertainty	-0.50	14.50

OUT1_reg_98_/CK (DFFPQ1)	0.00	14.50 r
library setup time	-0.17	14.33
data required time		14.33

data required time		14.33
data arrival time		-10.34

slack (MET)		3.99

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 15:24:10, 31 March 2003
 Design name: Mult64
 Report file name: Mult64d_routed.summary

Number of macros: 281
 Number of components: 13851
 Number of pins: 70539
 Number of regular pins: 43590
 Number of special pins: 26714
 Number of unused pins: 235
 Number of nets: 15589
 Average number of pins per net: 4.52
 Number of subnets: 491
 Number of regular pins for subnets: 257
 Number of special pins for subnets: 0
 Number of virtual pins for subnets: 725
 Average number of pins per subnet: 2.00
 Number of routing tracks available: 2896
 Number of GCELLS per layer: 20881

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 15:24:10, 31 March 2003
 Design name: Mult64
 Report file name: Mult64d_routed.summary

Number of macros: 281
 Number of components: 13851
 Number of pins: 70539
 Number of regular pins: 43590
 Number of special pins: 26714
 Number of unused pins: 235
 Number of nets: 15589
 Average number of pins per net: 4.52
 Number of subnets: 491
 Number of regular pins for subnets: 257
 Number of special pins for subnets: 0
 Number of virtual pins for subnets: 725
 Average number of pins per subnet: 2.00
 Number of routing tracks available: 2896
 Number of GCELLS per layer: 20881

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 15:24:10, 31 March 2003
 Design name: Mult64
 Report file name: Mult64d_routed.summary

** UTILIZATION OF ALL ROW TYPES

Type	Number	Length	Area	%_Row_Space
core Rows	126	97879320	602936611200	
core Cells	13357	60668520	373718083200	61.98

Area of chip: 769707892800 (square DBU)
 Area required for all cells: 373718083200 (square DBU)
 Area utilization of all cells: 48.55%

*****SILICON_ENSEMBLE WIRING REPORT*****

Time: 15:24:15, 31 March 2003
 Design name: Mult64
 Report file name: Mult64d_routed.wires

Total vias in regular wiring: 114122
 Total segments in regular wiring: 120407
 Total vias in special wiring: 32
 Total segments in special wiring: 44

LAYER name: metall
 Total wire length: 70292.58 microns
 Length of regular wires: 66979.38 microns
 Length of special wires: 3313.20 microns
 LAYER name: metal2

```

Total wire length: 318176.90 microns
Length of regular wires: 304934.02 microns
Length of special wires: 13242.88 microns
LAYER name: metal3
Total wire length: 510327.86 microns
Length of regular wires: 510327.86 microns
Length of special wires: .00 microns
LAYER name: metal4
Total wire length: 412856.72 microns
Length of regular wires: 412856.72 microns
Length of special wires: .00 microns
LAYER name: metal5
Total wire length: 181818.78 microns
Length of regular wires: 181818.78 microns
Length of special wires: .00 microns
LAYER name: metal6
Total wire length: 58916.16 microns
Length of regular wires: 58916.16 microns
Length of special wires: .00 microns

Total wirelength in regular wiring: 1535832.92 microns
Total wirelength in special wiring: 16556.08 microns
Total wirelength in regular+special wiring: 1552389.00 microns

```

CROSSTALK:

```

130 nets had no input C - consult log file for details
19 inconsistencies in HyperExtract output
    0 nets claimed N coupling caps, but had different no.
    19 nets had more coupling than total capacitance
    0 pin lists with fewer pins than they said they had.
    0 nets with bizarre cap/unit length
20 nets had implausible coupling ratios.
2 wires had implausibly small Rs.
131 rising drive and 131 falling drive implausibly small.
15589 nets processed, total wire 1543183.4
0 unknown nets encountered (maybe with repetitions)
130 constant nets, length 11552.3, ( 0.7%)
0 nets had errors, length 0.0, ( 0.0%)
Max crosstalk induced timing delta is 2.22n, (0 > 1us)
Sum of all error voltages is 0.000 volts

```

Mult64e:**Standard Multiplier Testbench (Booth Recoded Wallace Tree)**

```

*****
Report : area
Design : Mult64
Version: 2001.08-SP2
Date   : Wed Apr 2 13:53:12 2003
*****

```

```

Number of ports:      257
Number of nets:       641
Number of cells:      385
Number of references: 4

```

```

Combinational area: 344678.187500
Noncombinational area: 15689.158203
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area: 360367.343750
Total area: undefined

```

```

Mult64 top level:
  Cell Internal Power = 34.8979 mW (16%)
  Net Switching Power = 180.9021 mW (84%)
-----
Total Dynamic Power = 215.8000 mW (100%)
Cell Leakage Power = 19.1866 uW

```

```

Multiplier64 (part of Mult64):
  Cell Internal Power = 417.9735 mW (57%)
  Net Switching Power = 309.5211 mW (43%)
-----
Total Dynamic Power = 727.4946 mW (100%)
Cell Leakage Power = 13.6060 uW

```

```

*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : Mult64
Version: 2001.08-SP2
Date   : Wed Apr 2 13:57:02 2003
*****

```

Point	Incr	Path

clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
A_bottom_reg_18_/CK (DFFPQ1)	0.00	0.00 r
A_bottom_reg_18_/Q (DFFPQ1)	0.49	0.49 r
mult/in_low[18] (Multiplier64)	0.00	0.49 r
mult/mul_42/A[17] (Multiplier64_DW02_mult_64_64_0)	0.00	0.49 r
mult/mul_42/U2_MULT1/a[17] (Multiplier64_DW02_multp_64_64_130_0)	0.00	0.49 r
mult/mul_42/U2_MULT1/U1_BC/A[17] (Multiplier64_DW02_booth_64_1_0)	0.00	0.49 r
mult/mul_42/U2_MULT1/U1_BC/ENC_8/c (Multiplier64_DW_bthenc_1)	0.00	0.49 r
mult/mul_42/U2_MULT1/U1_BC/ENC_8/U14/Z (INVD1)	0.08	0.57 f
mult/mul_42/U2_MULT1/U1_BC/ENC_8/U12/Z (AND2D1)	0.17	0.74 f
mult/mul_42/U2_MULT1/U1_BC/ENC_8/U19/Z (MUX2D1)	0.39	1.13 f
mult/mul_42/U2_MULT1/U1_BC/ENC_8/shift1 (Multiplier64_DW_bthenc_1)	0.00	1.13 f
mult/mul_42/U2_MULT1/U1_BC/A_coded[25] (Multiplier64_DW02_booth_64_1_0)	0.00	1.13 f
mult/mul_42/U2_MULT1/U10679/Z (INVD1)	0.29	1.43 r
mult/mul_42/U2_MULT1/U11773/Z (INVD2)	0.11	1.53 f
mult/mul_42/U2_MULT1/U12350/Z (OAI22M22D1)	0.31	1.84 f
mult/mul_42/U2_MULT1/U9853/Z (EXOR2D1)	0.48	2.33 f
mult/mul_42/U2_MULT1/U1_WT/pp_array[578] (Multiplier64_DW_mtree_64_64_0)	0.00	2.33 f
mult/mul_42/U2_MULT1/U1_WT/U2_1_66_2/S (ADFULD1)	0.54	2.87 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_1_6/S (ADFULD1)	0.53	3.41 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_2_3/S (ADFULD1)	0.52	3.93 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_3_2/S (ADFULD1)	0.52	4.44 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_4_1/S (ADFULD1)	0.49	4.94 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_5_1/S (ADFULD1)	0.54	5.47 f
mult/mul_42/U2_MULT1/U1_WT/U4_1_66_6_0/S (ADFULD1)	0.52	5.99 f
mult/mul_42/U2_MULT1/U1_WT/U4_0A_1_66_7/S (ADFULD1)	0.54	6.53 f
mult/mul_42/U2_MULT1/U1_WT/out0[66] (Multiplier64_DW_mtree_64_64_0)	0.00	6.53 f
mult/mul_42/U2_MULT1/out0[66] (Multiplier64_DW02_multp_64_64_130_0)	0.00	6.53 f
mult/mul_42/U2_ADD1/A[65] (Multiplier64_DW01_add_127_0)	0.00	6.53 f
mult/mul_42/U2_ADD1/U0_2_65/Z (NOR2D1)	0.30	6.83 r
mult/mul_42/U2_ADD1/U1_5_0_66/Z (NOR2D1)	0.12	6.96 f
mult/mul_42/U2_ADD1/U1_4_1_66/Z (AOI21D1)	0.42	7.38 r
mult/mul_42/U2_ADD1/U1_4_2_70/Z (OAI21D1)	0.20	7.58 f
mult/mul_42/U2_ADD1/U1_4_3_78/Z (AOI21D2)	0.38	7.96 r
mult/mul_42/U2_ADD1/U1_2_4_78/Z (INVD1)	0.08	8.04 f
mult/mul_42/U2_ADD1/U171/Z (OAI21D1)	0.41	8.44 f
mult/mul_42/U2_ADD1/U1_4_5_95/Z (AOI21D1)	0.26	8.70 r
mult/mul_42/U2_ADD1/U1_4_6_95/Z (OAI21D2)	0.25	8.94 f
mult/mul_42/U2_ADD1/U0_5_96/Z (EXOR2D2)	0.40	9.35 f
mult/mul_42/U2_ADD1/SUM[96] (Multiplier64_DW01_add_127_0)	0.00	9.35 f
mult/mul_42/PRODUCT[97] (Multiplier64_DW02_mult_64_64_0)	0.00	9.35 f
mult/out[98] (Multiplier64)	0.00	9.35 f
OUT1_reg_98_/D (DFFPQ1)	0.00	9.35 f
data arrival time		9.35

clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
clock uncertainty	-0.50	9.50
OUT1_reg_98_/CK (DFFPQ1)	0.00	9.50 r
library setup time	-0.15	9.35
data required time		9.35

data required time		9.35
data arrival time		-9.35

slack (MET)		0.00

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 23:36:38, 2 April 2003
 Design name: Mult64
 Report file name: Mult64e.summary

Number of macros: 297
 Number of components: 9882
 Number of pins: 54738
 Number of regular pins: 35717
 Number of special pins: 18796
 Number of unused pins: 225
 Number of nets: 11630
 Average number of pins per net: 4.71
 Number of subnets: 481
 Number of regular pins for subnets: 257
 Number of special pins for subnets: 0
 Number of virtual pins for subnets: 705
 Average number of pins per subnet: 2.00
 Number of routing tracks available: 2548
 Number of GCELLS per layer: 16146

** NET STATISTICS OF PIN COUNTS

Number of 2-pin nets: 9567
 Number of 3-pin nets: 703
 Number of 4-pin nets: 243
 Number of 5-pin nets: 100
 Number of 6-pin nets: 186
 Number of 7-pin nets: 79
 Number of 8-pin nets: 21
 Number of 9-pin nets: 28
 Number of 10-pin nets: 9
 Number of 11-pin nets: 20
 Number of 12-pin nets: 22
 Number of 13-pin nets: 39
 Number of 14-pin nets: 53
 Number of 15-pin nets: 81
 Number of 16-pin nets: 476
 Number of 257-pin nets: 1
 Number of 9399-pin nets: 2

*****SILICON_ENSEMBLE DESIGN SUMMARY REPORT*****

Time: 23:36:38, 2 April 2003
 Design name: Mult64
 Report file name: Mult64e.summary

** UTILIZATION OF ALL ROW TYPES

Type	Number	Length	Area	%_Row_Space
core Rows	109	73162980	450683956800	
core Cells	9398	58501080	360366652800	79.96

Area of chip: 595814049600 (square DBU)
 Area required for all cells: 360366652800 (square DBU)
 Area utilization of all cells: 60.48%

*****SILICON_ENSEMBLE WIRING REPORT*****

Time: 23:36:45, 2 April 2003
 Design name: Mult64
 Report file name: Mult64e.wires

Total vias in regular wiring: 105125
 Total segments in regular wiring: 110441
 Total vias in special wiring: 1140
 Total segments in special wiring: 386

LAYER name: metal1

Total wire length: 127593.76 microns
 Length of regular wires: 45205.96 microns
 Length of special wires: 82387.80 microns

LAYER name: metal2

Total wire length: 232618.42 microns
 Length of regular wires: 218170.42 microns
 Length of special wires: 14448.00 microns

LAYER name: metal3

Total wire length: 432153.96 microns
 Length of regular wires: 432153.96 microns
 Length of special wires: .00 microns

LAYER name: metal4

Total wire length: 314007.86 microns
 Length of regular wires: 314007.86 microns

```

      Length of special wires: .00 microns
LAYER name: metal5
      Total wire length: 162043.82 microns
      Length of regular wires: 162043.82 microns
      Length of special wires: .00 microns
LAYER name: metal6
      Total wire length: 55407.00 microns
      Length of regular wires: 55407.00 microns
      Length of special wires: .00 microns

Total wirelength in regular wiring: 1226989.02 microns
Total wirelength in special wiring: 96835.80 microns
Total wirelength in regular+special wiring: 1323824.82 microns

CROSSTALK:
      0 nets claimed N coupling caps, but had different no.
      4 nets had more coupling than total capacitance
      0 pin lists with fewer pins than they said they had.
      0 nets with bizarre cap/unit length
      4 nets had implausible coupling ratios.
      2 wires had implausibly small Rs.
      131 rising drive and 131 falling drive implausibly small.
      11630 nets processed, total wire      1233651.7
      0 unknown nets encountered (maybe with repetitions)
      130 constant nets, length      10179.0, ( 0.8%)
      0 nets had errors, length      0.0, ( 0.0%)
Max crosstalk induced timing delta is 1.19n, (0 > 1us)
Sum of all error voltages is      0.000 volts

Mult64e power: 231.069mW

```

DELAY ANALYSIS: From Pearl

```

RecursiveMultiplierV4
input register0.26
MUX10.12
Multiplier3.49
MUX20.11
Reduction1.27
FastAdder_low1.75
FastAdder_high2.00

Mult64d
input register1.10
multiplier4.88
adder5.20

Mult64e
input register2.23
multiplier4.37
adder3.32

```

Vita Auctoris

Pedram Mokrian received the B.A.Sc. degree in Electrical Engineering from the University of Windsor, graduating with first class honours (11.92/13) in 2001. A recognized member of the Dean's list and the President's Roll over his entire academic career at Windsor, Pedram has also gained invaluable industrial experience through his enrollment in the co-op program. His professional employment includes positions at Ford Motor Company in Windsor, and research and development placements in Ottawa at JDS Uniphase and at Nortel Networks as a hardware design engineer.

In the Fall of 2001, Pedram pursued a M.A.Sc. degree in Electrical Engineering under the supervision of Dr. Majid Ahmadi at the University of Windsor. During this period, he has received a University of Windsor Tuition Scholarship, an Ontario Graduate Scholarship, an Ontario Graduate Scholarship in Science and Technology, and most recently a National Research Council of Canada Post-Graduate Fellowship. His area of specialization has been the VLSI implementation of arithmetic algorithms, with focus on the impact of technology scaling on digital multiplication architectures.

Pedram intends on commencing his studies towards a Ph.D. degree in the Fall of 2003.