University of California

Los Angeles

High-Level Optimization Techniques for Low-Power Multiplier Design

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Zhijun Huang

2003

The dissertation of Zhijun Huang is approved.

Jason Cong

Babak Daneshrad

Majid Sarrafzadeh

Miloš D. Ercegovac, Committee Chair

University of California, Los Angeles

2003

To my parents, Guangzhong Huang and Lanying Kan.

Table of Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Power Optimization 2

1.3 Low-Power Multiplier Design 6

1.4 Research Approach 9

1.5 Organization 10

2 Optimization of Multiplier Recoding for Low Power **13**

2.1 Introduction 13

2.2 Radix-4 Recoding 16

2.2.1 Some Existing Recoding Schemes 19

2.2.2 New Recoding Schemes 31

2.2.3 Experimental Evaluation 36

2.3 Comparison of Radix-4 and Radix-2 Multipliers 41

2.4 Summary 45

3 Optimization of Multiplier Operand Representations for Low Power **47**

3.1 Introduction 47

3.2 Representation Conversion for Radix-2 2C Multipliers 50

3.2.1 2C-SM-2C: A Straightforward Conversion 50

3.2.2 2C-P1-2C: Conversion to SM with Postponed "+1" 55

3.2.3 2C-P1-CS: Conversion to 2C Using Carry-Save Addition . 57

3.3 Representation Conversion for Radix-4 2C Multipliers 59

3.4 Experimental Evaluation 61

3.5 Summary 64

4 Optimization of Reduction Structure for Array Multipliers . . 65

4.1 Introduction 65

4.2 Left-to-Right Array Multipliers 68

4.3 Structure Optimization 71

4.3.1 Signal Flow Optimization in [3:2]-CSA Based Array 73

4.3.2 CRAs for PP Reduction 79

4.3.3 [4:2]-CSAs for PP Reduction 81

4.3.4 Split Array: Even/odd and Upper/lower 83

4.4 Experimental Evaluation 85

4.4.1 Results for Radix-2 LR Multipliers 87

4.4.2 Results for Radix-4 LR Multipliers 90

4.5 Summary 94

5 Design of High-Performance Low-Power Multipliers 96

5.1 Introduction 96

5.2 Partial Product Generation 98

5.3 Partial Product Reduction by Split Array Structure 99

5.3.1 Split Array LRLF PP reduction 100

5.3.2 Optimization of [3:2]-CSAs 104

5.3.3 Optimization of [4:2]-CSAs 106

5.4 Partial Product Reduction by Tree Structure 107

5.5 Final Addition Optimized for Arbitrary Input Arrival Time 109

5.6 Experimental Evaluation 114

5.6.1 Delay Comparison at Logic Level 114

5.6.2 Power/Delay/Area Comparison at Physical Layout Level . 116

5.7 Summary 120

6 Signal Gating in Linear Array Multipliers 122

6.1 Introduction 122

6.2 Signal Gating Schemes 126

6.2.1 General Description 127

6.2.2 Static Gating versus Dynamic Gating 128

6.2.3 One-Dimensional Signal Gating 129

6.2.4 Two-Dimensional Signal Gating 132

6.3 Bit-level Implementation 134

6.3.1 Sign-Extension Detection Logic 135

6.3.2 Signal Gating Logic 137

6.3.3 Modification in Array Multiplier Core 139

6.4 Experimental Evaluation 151

6.5 Summary 152

7 Signal Gating in High-Performance Multipliers 154

7.1 Introduction 154

7.2 Signal Gating in ULLRLF Array Multiplier 155

7.2.1 Modification of PP Bit Matrix 155

7.2.2 Resource Sharing 158

7.3 Signal Gating in Tree Multiplier 161

7.4 Experimental Evaluation 166

7.5 Summary 168

8 Conclusion and Future Work 170

8.1 Research Contributions 170

8.2 Future Work 175

A Abbreviations 177

B Design and Experimental Methodology 179

B.1 Logic Style 179

B.2 Artisan Standard Cell Library 180

B.3 Power/Delay/Area Estimation 182

B.4 Test Data Sets 185

References 188

List of Figures

2.1 Parallel radix-4 recoding. 16

2.2 Serial radix-4 recoding. 17

2.3 Transformation and pre-computation rules. 18

2.4 Radix-4 PP bit array in 8×10-bit two's-complement multiplication. 18

2.5 PR3a recoder and PP generator. 20

2.6 PR3b recoder and PP generator [7]. 23

2.7 PR3b PP generator using pass transistors. 23

2.8 PR3c recoder and PP generator. 24

2.9 PR4a recoder and PP generator. 25

2.10 PR4b recoder. 26

2.11 PR4b PP generators. 27

2.12 PR5 recoder and PP generator. 28

2.13 SR3 recoder and PP generator. 30

2.14 NPR3a recoder and PP generator. 33

2.15 NPR3b recoder and PP generator. 33

2.16 NSR3 serial recoder. 34

2.17 NSR4 recoder and PP generator with $P2/P1/N1/zero$. 36

2.18 NSR4 recoder and PP generator with hidden *zero*. 37

3.1 2C-SM-2C conversion scheme. 51

3.2 Representation conversion logic for 2C-SM-2C. 52

3.3 Linear structure of simplified 2C-SM conversion. 53

3.4 Tree structure of 2C-SM conversion ($m = 9$). 55

3.5 2C-P1-2C structure. 56

3.6 PP bit array in 2C-P1-2C ($m = 8, n = 6$). 57

4.1 Radix-2 PP bit matrix (n=8): (a) RL; (b) LR. 69

4.2 Radix-4 PP bit matrix (n=8): (a) RL; (b) LR. 69

4.3 Radix-2 RL carry-save array multiplier (n=8). 70

4.4 Radix-2 LR carry-save array multiplier (n=8). 71

4.5 Radix-4 LR carry-save array multiplier (n=12). 72

4.6 Two designs of full adder. 74

4.7 Scenarios of glitch generation in FA-MUX. 75

4.8 α-τ signal flow optimization in [3:2]-CSA Linear PPR. 76

4.9 Radix-2 LR [3:2]-CSA array multiplier with CSSC flow (n=8). . . 77

4.10 Radix-4 LR [3:2]-CSA array multiplier with CSSC flow (n=12). . 78

4.11 Radix-2 LR CRA based PPR algorithm. 80

4.12 Radix-2 LR CRA based array multiplier (n=8). 80

4.13 1-bit [4:2]-CSA. 82

4.14 Radix-4 LR array multiplication based on [4:2]-CSAs (n=16). . . 82

4.15 High-level radix-4 LRLF algorithm. 85

4.16 Radix-4 LRLF array multiplier (n=12). 86

4.17 Radix-4 LR upper/lower array multiplier (n=16). 87

5.1 MSB-first radix-4 PP bit array (n=12). 99

5.2 High-level EOLRLF algorithm. 101

5.3 EOLRLF array multiplier (n=24). 101

5.4 Portion of a LRLF structure for upper half PPs (n=24). 102

5.5 ULLRLF array multiplier (n=24). 102

5.6 High-level ULLRLF algorithm. 103

5.7 PPGR delay profiles (n=32). 104

5.8 NAND3-based FA design (FA-ND3). 105

5.9 Two simplified M42 designs. 107

5.10 Tree PPR with $9T_{XOR2}$ delay (n=32). 108

5.11 A special [9:4]-CSA with $3T_{XOR2}$ delay. 109

5.12 Earlist-first PA carry generation algorithm (adapted from [132]). . 111

5.13 A PA example. 112

5.14 PA delay profile in EOLRLF (n=32). 113

5.15 Floorplan of tree9to4 (n=32). 119

5.16 Floorplan of ULLRLF (n=32). 120

6.1 The proposed 4-point ensemble multiplier system in [13]. 125

6.2 The proposed low-power multiplier in [50]. 126

6.3 General behavior of signal gating schemes. 128

6.4 1-D gating diagrams. 130

6.5 High-level algorithm of G-X. 130

6.6 High-level algorithm of G-Y. 131

6.7 High-level algorithm of G-P ($m > NG_P > n$). 132

6.8 2-D gating diagrams. 133

6.9 High-level algorithm of G-J. 133

6.10 High-level algorithm of G-G. 134

6.11 Sign-extension detection logic. 136

6.12 Timing diagram of SED with glitch elimination. 137

6.13 Tree structure of SED. 137

6.14 Various implementations of gating logic. 138

6.15 A baseline radix-2 two's-complement array multiplier. 140

6.16 The 1-D G-X scheme for two's-complement array multiplier. . . . 141

6.17 The 1-D G-Y scheme for two's-complement array multiplier. . . . 142

6.18 The 1-D G-P scheme for two's-complement array multiplier. . . . 144

6.19 The 2-D G-J scheme for two's-complement array multiplier. . . . 145

6.20 The 2-D G-G scheme for two's-complement array multiplier. . . . 147

6.21 An example of 2-D G-G. 149

6.22 Special cells in 2-D G-G. 150

7.1 High-level diagram of 2-D G-G for ULLRLF. 156

7.2 Bit-level diagram of 2-D G-G in ULLRLF. 156

7.3 The shaded region in Figure 7.2. 157

7.4 LRLF for upper PPs with only [3:2]-CSAs. 159

7.5 Cell FAP1 for $(A + B + C + 1)$. 159

7.6 Signal gating in LRLF for upper PPs. 160

7.7 2-D G-G in a tree multiplier. 161

7.8 Bit-level diagram of 2-D G-G in a tree multiplier. 162

7.9 The shade region in Figure 7.8. 163

8.1 Normalized experimental results of optimization techniques in (a) array multipliers, (b) tree multipliers. 173

8.2 Another view of normalized experimental results in (a) array multipliers, (b) tree multipliers. 174

B.1 Probability of being extension bit in *djpeg* data. 186

B.2 Static probability and toggle rate of each bit in *djpeg* data. 187

List of Tables

2.1 Names and features of recoding schemes studied 19
2.2 PR3a recoding 20
2.3 PR3b recoding 22
2.4 PR3c recoding 24
2.5 PR4a recoding 25
2.6 PR4b recoding 26
2.7 PR5 recoding 28
2.8 SR3 recoding 29
2.9 High-level comparison of existing recoding schemes 31
2.10 NPR3a recoding 32
2.11 NSR3 recoding 34
2.12 NSR4 recoding 36
2.13 High-level comparison of proposed recoding schemes 37
2.14 Comparison of multipliers with different parallel recoding schemes 39
2.15 Comparison of multipliers with different serial recoding schemes . 40
2.16 Comparison of radix-2 and radix-4 32 × 32-bit array multipliers . 43
2.17 Comparison of radix-2 and radix-4 32 × 32-bit tree multipliers . . 43

3.1 Power distribution in 2C linear array multiplier under *djpeg* data . 48
3.2 Power comparison of representation conversion array schemes . . . 62
3.3 Power comparison of representation conversion tree schemes . . . 62
3.4 Area/delay comparison of representation conversion schemes . . . 63

4.1 Power distribution in 2C linear array multiplier under *random* data 66

4.2 Delay of the n-th column in radix-4 LR with CSSC 79

4.3 Power in radix-2 LR multipliers under *random* data 88

4.4 Power in radix-2 LR multipliers under *djpeg* data 88

4.5 Cell area in radix-2 LR multipliers 89

4.6 Delay in radix-2 LR multipliers 89

4.7 Power in radix-4 LR multipliers under *random* data 91

4.8 Power in radix-4 LR multipliers under *djpeg* data 91

4.9 Area in radix-4 LR multipliers 92

4.10 Delay in radix-4 LR multipliers 92

5.1 Effects of FA type and SFO on logic delay in LRLF/SALRLF . . 115

5.2 Logic delay comparison of tree multipliers and LRLF/SALRLF . . 116

5.3 Power/area/delay after automatic layout (24-bit) 118

5.4 Power/area/delay after automatic layout (32-bit) 118

5.5 Power/area/delay after layout with guided floorplanning (32-bit) . 120

6.1 Power comparison of 2-D signal gating schemes 151

6.2 Delay/area comparison of 2-D signal gating schemes 152

7.1 Power comparison of 2-D G-G high-performance multipliers . . . 167

7.2 Delay/area comparison of 2-D G-G high-performance multipliers . 167

B.1 Area/delay/power characteristics of some Artisan library cells . . 181

Acknowledgments

I am deeply indebted to my advisor, Prof. Miloš D. Ercegovac, for all of his technical, financial, and personal support during my years at UCLA. He has been patiently guiding me and enthusiastically encouraging me throughout these four years. He developed my skills of identifying research problems and finding solutions precisely and clearly. What I have learned from him will benefit me well beyond my graduation in my future research career and personal life.

I am grateful to Profs. Jason Cong, Majid Sarrafzadeh, and Babak Daneshrad for serving on my committee. They have not only offered invaluable technical advice, but also offered CAD tool support in my dissertation work. Moreover, I treasure my joint-work experience with Prof. Cong and I highly appreciate his helps in my career planning. I would also like to thank Dr. Farzan Fallah from Fujitsu Laboratories of America, Inc., for his interest and suggestions on my work. Thanks also go to Prof. Lei He and Prof. Sheldon Tan (UCR) for their continuous encouragement and advice since the time I knew them. I especially thank Ms. Verra Morgan for her efforts in making my student life joyful.

I would like to thank these wonderful friends in the Computer Science department and Electrical Engineering department. To name a few, they are Deming Chen, Ian Ferguson, Soheil Ghiasi, Roozbeh Jafari, Ashok Jagannathan, Fei Li, Robert McIlhenny, John Pipan, Ankur Srivastava, Zhiwei Xu, Guichang Zhong. They have helped me in many ways during my study at UCLA.

Most of all, I am grateful to my wife Min Zhang for her devotion, understanding, and support. She is the most important source of inspiration, encouragement, and happiness.

Vita

1996	B.S. in Electronics Engineering Fudan University, China
1999	M.S. in Electronics Engineering Fudan University, China
2000	M.S. in Computer Science University of California, Los Angeles

Publications

Huang Zhijun and Tong Jiarong, "An efficient FPGA logic block for word-oriented datapath," in *Proc. 1998 Int. Conf. ASIC*, Beijing, China, Oct. 1998.

Zhou Feng, Huang Zhijun, Tong Jiarong, and Tang Pushan, "An analytical delay model for SRAM-based FPGA interconnections," in *Proc. ASP-DAC'99: Asia and South Pacific Design Automation Conference*, vol.1, pp.101-104, Jan. 1999.

Zhang Peng, Huang Zhijun, and Tong Jiarong, "A novel fault-tolerant method of a FPGA for datapath," in *Proc. 1999 Int. Conf. CAD/Graphics*, Shanghai, China, Dec. 1999.

Z. Huang and M.D. Ercegovac, "Effect of Wire Delay on the Design of Prefix

Adders in Deep-Submicron Technology," in *Proc. 34th Asilomar Conf. Signals, Systems and Computers*, pp.1713-1717, Nov. 2000.

R. McIlhenny, Z. Huang, K. Wong, A. Schneider, and M.D. Ercegovac, "BigSky - a tool for mapping numerically intensive computations onto reconfigurable hardware," in *Proc. 34th Asilomar Conf. Signals, Systems and Computers*, Nov. 2000.

D. Chen, J. Cong, M.D. Ercegovac, and Z. Huang, "Performance-driven mapping for CPLD structures," in *Proc. FPGA'01: ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Feb. 2001.

Z. Huang and M.D. Ercegovac, "FPGA implementation of pipelined on-line scheme for 3-D vector normalization," in *Proc. FCCM'01: Ninth Annual IEEE Symp. Field-Programmable Custom Computing Machines*, Apr. 2001.

Z. Huang and M.D. Ercegovac, "On signal-gating schemes for low-power adders," in *Proc. 35th Asilomar Conf. Signals, Systems and Computers*, pp.867-871, Nov. 2001.

Z. Huang and M.D. Ercegovac, "Two-dimensional signal gating for low-power array multiplier design," in *Proc. 2002 IEEE Int. Symp. Circuits and Systems*, vol.1, pp.489-492, May 2002.

Z. Huang and M.D. Ercegovac, "Number representation optimization for low-power multiplier design," in *Proc. SPIE 2002 Advanced Signal Processing Algo-*

rithms, Architectures, and Implementations XII, July 2002.

Z. Huang and M.D. Ercegovac, "Low power array multiplier design by topology optimization," in *Proc. SPIE 2002 Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, July 2002.

D. Chen, J. Cong, M.D. Ercegovac, and Z. Huang, "Performance-driven mapping for CPLD structures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (to appear).

Z. Huang and M.D. Ercegovac, "High-performance left-to-right array multiplier design," in *16th IEEE Symp. Computer Arithmetic*, June 2003.

Z. Huang and M.D. Ercegovac, "Two-dimensional signal gating for low power in high-performance multipliers," in *SPIE 2003 Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, Aug. 2003.

Abstract of the Dissertation

High-Level Optimization Techniques for Low-Power Multiplier Design

by

Zhijun Huang

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2003

Professor Miloš D. Ercegovac, Chair

While performance and area remain to be two major design goals, power consumption has become a critical concern in today's VLSI system design. Multiplication is a fundamental operation in most arithmetic computing systems. Multipliers have large area, long latency and consume considerable power. Previous work on low-power multipliers focuses on low-level optimizations and has not considered well the arithmetic computation features and application-specific data characteristics.

At the algorithm and architecture level, this dissertation addresses low-power multiplier design systematically from two aspects: internal efforts considering multiplier architectures and external efforts considering input data characteristics. For internal efforts, we consider recoding optimization for partial product generation, operand representation optimization, and structure optimization of partial product reduction. For external efforts, we consider signal gating to deactivate portions of a full-precision multiplier. Several multiplier types are studied: linear array multipliers, leapfrog array multipliers, left-to-right linear array multipliers, split array multipliers, and tree multipliers. Experiments show that recoding optimization and structure optimization have achieved steady power reduction with reduced delay for both random test data and large-dynamic-range data. Operand representation optimization and signal gating have demonstrated significant power saving for large-dynamic-range data with relatively small overhead.

CHAPTER 1

Introduction

1.1 Motivation

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip [1][25][31]. These signal processing applications not only demand great computation capacity but also consume considerable amounts of energy. While performance and area remain to be two major design goals, power consumption has become a critical concern in today's VLSI system design [64]. The need for low-power VLSI systems arises from two main forces. First, with the steady growth of operating frequency and processing capacity per chip, large current has to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices.

Multiplication is a fundamental operation in most signal processing algorithms [101]. Multipliers have large area, long latency and consume considerable power. Therefore, low-power multiplier design has been an important part in low-power VLSI system design. There has been extensive work on low-power multipliers at technology, physical, circuit and logic levels. These low-level techniques are not unique to multiplier modules and they are generally applicable to other types of modules. The characteristics of arithmetic computation in multipliers are not considered well. Moreover, power consumption is directly related to data switching patterns. However, it is difficult to consider application-specific data characteristics in low-level power optimization.

This dissertation addresses high-level optimization techniques for low power multipliers. High-level techniques refer to algorithm and architecture level techniques that consider multiplication's arithmetic features and input data characteristics. The main research hypothesis of this work is that high-level optimization of multiplier designs produces more power-efficient solutions than optimization only at low levels. Specifically, we consider how to optimize the internal algorithm and architecture of multipliers and how to control active multiplier resource to match external data characteristics. The primary objective is power reduction with small area and delay overhead. By using new algorithms or architectures, it is even possible to achieve both power reduction and area/delay reduction, which is another strength of high-level optimization. The tradeoff between power, area and delay is also considered in some cases.

1.2 Power Optimization

Power refers to the number of Joules dissipated over a certain amount of time whereas energy is a measure of the total number of Joules dissipated by a circuit. Strictly speaking, low-power design is a different goal from low-energy design although they are related [130][87]. Power is a problem primarily when cooling is a concern. The maximum power at any time, *peak power*, is often used for power and ground wiring design, signal noise margin and reliability analysis. Energy per operation or task is a better metric of the energy efficiency of a system, especially in the domain of maximizing battery lifetime.

In digital CMOS design, the well-known *power-delay product* is commonly used to assess the merits of designs. In a sense, this is a misnomer as $power \times delay = (energy/delay) \times delay = energy$, which implies delay is irrelevant [87]. Instead, the term energy-delay product should be used since it involves two independent measures of circuit behaviors. Therefore, when *power-delay products* are used as a comparison metric, different schemes should be measured at the same frequency to ensure that it is equivalent to *energy-delay product* comparison.

There are two major sources of power dissipation in digital CMOS circuits: dynamic power and static power [127][11][104][130][106]. Dynamic power is related to circuit switching activities or the changing events of logic states, including power dissipation due to capacitance charging and discharging, and dissipation due to short-circuit current (SCC). In CMOS logic, unintended leakage current, either reverse biased PN-junction current or subthreshold channel conduction current, is the only source of static current. However, occasional deviations from the strict CMOS style logic, such as pseudo NMOS logic, can cause intended static current. The total power consumption is summarized in the following equations [104]:

$$P_{total} = P_{dynamic} + P_{static} = P_{cap} + P_{scc} + P_{static} \tag{1.1}$$

$$P_{cap} = \alpha_{0\to1} f_{clk} \cdot \int_0^T i_{V_{DD}}(t) V_{DD} dt = \alpha_{0\to1} f_{clk} \cdot C_L V_{DD}^2 \tag{1.2}$$

$$P_{scc} = \alpha_{0\to1} f_{clk} \cdot I_{peak}(\frac{t_r + t_f}{2}) V_{DD} \tag{1.3}$$

$$P_{static} = I_{static} V_{DD} \tag{1.4}$$

P_{cap} in Equation 1.2 represents the dynamic power due to capacitance charging and discharging of a circuit node, where C_L is the loading capacitance, f_{clk} is the clock frequency, and $\alpha_{0\to1}$ is the $0 \to 1$ transition probability in one clock period. In most cases, the voltage swing V_{swing} is the same as the supply voltage V_{DD}; otherwise, V_{swing} should replace V_{DD} in this equation. P_{scc} is a first-order average power consumption due to short-circuit current. The peak current, I_{peak}, is determined by the saturation current of the devices and is hence directly proportional to the sizes of the transistors. t_r and t_f are rising time and falling time of short-circuit current, respectively. The static power P_{static} is primarily determined by fabrication technology considerations, which is usually several orders of magnitude smaller than the dynamic power. The leakage power problem mainly appears in very low frequency circuits or ones with "sleep modes" where dynamic activities are suppressed [130]. The dominant term in a "well-designed" circuit during its active state is the dynamic term due to switching activity on loading capacitance, and thus low-power design often becomes the task of minimizing $\alpha_{0\rightarrow 1}$, C_L, V_{DD} and f_{clk}, while retaining the required functionality [26]. In the future, static power will become increasingly important as the supply voltage keeps scaling. To avoid performance degrading, the threshold voltage V_t is lowered accordingly and subthreshold leakage current increases exponentially [106]. Leakage power reduction heavily depends on circuit and technology techniques such as dual V_t partitioning and multi-threshold CMOS [66]. In this work, we will not consider leakage power reduction.

Power optimization of digital systems has been studied at different abstract levels, from the lowest technology level, to the highest system level [26][27][11][130]. At the technology level, power consumption is reduced by the improvement in fabrication process such as small feature size, very low voltages, copper interconnects, and insulators with low dielectric constants [26][51]. With the fabrication support of multiple supply voltages, lower voltages can be applied on non-critical system blocks. At the layout level, placement and routing are adjusted to reduce wire capacitance and signal delay imbalances [8][86]. At the circuit level, power reduction is achieved by transistor sizing, transistor network restructuring and reorganization, and different circuit logic styles. At the gate level, a lot of techniques have been proposed. In commercial tool Power Compiler [120], the gate-level techniques include cell sizing, cell composition, equivalent pin swapping, and buffer insertion, which produces 11% power reduction on average with 9% area increase [28]. Some other proposed techniques are gate-level signal gating [76][122][116], delay balancing [107], input synchronization [47], signal polarity optimization [87], etc. At register-transfer level (RTL), clock gating has been applied extensively to disable combinational or sequential blocks not used during a particular period [57][128][129]. In Power Compiler [120], clock gating is applied to disable a whole register bank and operand isolation is applied to disable a datapath unit if their outputs are not used. Precomputation is a RTL signal gating technique which identifies output-invariant logical conditions at some inputs of a combinational block and then disables inputs under these conditions [4]. Retiming re-positions registers in sequential circuits so that the propagation of spurious transitions is stopped [88]. At the architecture and system level, there is a great amount of freedom in power optimization. Parallelism and pipeline are two main techniques to first achieve higher-than-necessary performance and then trade operation frequency for supply voltage reduction [27]. Control-data-flow graph (CDFG) transformation is another efficient technique to design low power architecture [27][91]. Asynchronous systems are investigated to avoid a global clock signal and reduce useless computations [81][70][103]. In many event-driven systems, components are disabled or shutdown when they are in idle states, which is generally called power management strategy [53][57].

Although optimization techniques at all levels have achieved power reduction, the techniques at the lowest technology level and the highest architecture/system level are generally more efficient than techniques at middle levels. Technology-level optimization affects three important factors C_L, V_{DD}, and f_{clk} in power consumption. Algorithm/architecture-level optimization affects all four factors

$\alpha_{0\to1}$, C_L, f_{clk}, and V_{DD} (by identifying candidates for V_{DD} lowering). In contrast, middle-level optimization usually affects one or two factors in a limited way.

1.3 Low-Power Multiplier Design

Multiplication consists of three steps: generation of partial products or PPs (PPG), reduction of partial products (PPR), and final carry-propagate addition (CPA) [74][102]. In general, there are sequential and combinational multiplier implementations. We only consider combinational multipliers in this work because the scale of integration now is large enough to accept parallel multiplier implementation in digital VLSI systems. Different multiplication algorithms vary in the approaches of PPG, PPR, and CPA. For PPG, radix-2 digit-vector multiplication is the simplest form because the digit-vector multiplication is produced by a set of AND gates. To reduce the number of PPs and consequently reduce the area/delay of PP reduction, one operand is usually recoded into high-radix digit sets. The most popular one is the radix-4 digit set $\{-2, -1, 0, 1, 2\}$. For PPR, two alternatives exist [45]: reduction by rows [125], performed by an array of adders, and reduction by columns [37], performed by an array of counters. In reduction by rows, there are two extreme classes: linear array and tree array. Linear array has the delay of $O(n)$ while both tree array and column reduction have the delay of $O(\log n)$, where n is the number of PPs. The final CPA requires a fast adder scheme because it is on the critical path. In some cases, final CPA is postponed if it is advantageous to keep redundant results from PPG for further arithmetic operations.

Many power optimization techniques introduced in Section 1.2, especially low-level techniques, are applicable to low-power multiplier design. These low-level techniques that has been studied for multipliers include using voltage scaling [8], layout optimization [8], transistor reordering and sizing [8], using pass-transistor logic [2][77] and swing limited logic [48], signal polarity optimization [87], delay balancing [107][87][72][16] and input synchronization [92][47]. However, these techniques have only achieved moderate improvement on power consumption in multipliers with much design effort or considerable area/delay overhead. The difficulty of low-power multiplier design lies in three aspects. First, the multiplier area is quadratically related to the operand precision. Second, parallel multipliers have many logic levels that introduce spurious transitions or glitches. Third, the structure of parallel multipliers could be very complex in order to achieve high speed, which deteriorates the efficiency of layout and circuit level optimization. As a fundamental arithmetic operation, multiplication has many algorithm-level and bit-level computation features in which it differs from random logic. These features have not been considered well in low-level power optimization. It is also difficult to consider input data characteristics at low levels. Therefore, it is desirable to develop algorithm and architecture level power optimization techniques that consider multiplication's arithmetic features and operands' characteristics.

There has been some work on low-power multipliers at the algorithm and architecture level. In [12], Bewick presented the power consumptions of different fast multiplication designs based on ECL (Emitter Coupled Logic). Bewick's results cannot be extended into CMOS logic because power consumption in ECL is dominated by static power while power in CMOS is dominated by dynamic power. In [3], Al-Twaijry studied area and performance optimized CMOS multipliers. As smaller area usually leads to less switching capacitance, the results in [3] could provide a rough estimation of relative power consumptions in different multiplication schemes. In [22], Callaway studied the power/delay/area characteristics of four classical multipliers. In [8], Angel proposed low-power sign extension schemes and self-timed design with bypassing logic for zero PPs in radix-4 multipliers. In [30], Cherkauer and Friedman proposed a hybrid radix-4/radix-8 low power signed multiplier architecture. For multiplication data with large dynamic range, several approaches have been proposed. Architecture-level signal gating techniques have been studied [17][18][13][70][50]. In [136], a mixed number representation for radix-4 two's-complement multiplication is proposed. In [97], radix-4 recoding is applied to the constant input instead of the dynamic input in low-power multiplication for FIR filters. In [58], multiplication is separated into higher and lower parts and the results of the higher part are stored in a cache in order to reduce redundant computation. In [135], two techniques are proposed for data with large dynamic range: most-significant-digit-first carry-save array for PP reduction [133] and dynamically-generated reduced two's-complement representation [134]. In [111], the precisions of two input data are compared at runtime and two operands are then exchanged if necessary so that radix-4 recoding is applied on the operand with smaller precisions in order to generate more zero PPs.

At the algorithm and architecture level, this dissertation addresses low-power multiplier design systematically from two aspects: internal efforts considering multiplier architectures and external efforts considering input data characteristics. For internal efforts, we consider the optimization techniques of PPG recoding, operand representation, and PPR structure. For external efforts, we consider signal gating to deactivate portions of a full-precision multiplier. Several multiplier types are studied: linear array multipliers, leapfrog array multipliers, left-to-right linear array multipliers, split array multipliers, and tree multipliers. Column reduction multipliers are not considered because column reduction achieves similar delay and area as tree array while making the regularity much worse [14]. As two's-complement representation is convenient for additions of signed integers and thus very popular in digital arithmetic systems, we assume the multiplicand X and the multiplier Y are initially in radix-2 two's-complement representation:

$$X = -x_{m-1}2^{m-1} + \sum_{j=0}^{m-2} x_j 2^j \tag{1.5}$$

$$Y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i \tag{1.6}$$

where n is assumed even for simplicity. The result P is an $(m+n)$-bit number also in two's-complement representation:

$$P = -p_{m+n-1}2^{m+n-1} + \sum_{k=0}^{m+n-2} p_k 2^k \tag{1.7}$$

1.4 Research Approach

The research approach followed in this dissertation is briefly described next. Appendix B gives a detailed description of the design and experimental methodologies used in our research.

First, we identify important factors that affect power consumption at the algorithm and architecture level. These factors include internal architectures of multipliers and external data characteristics.

Second, we develop high-level power optimization techniques including internal efforts and external efforts. Internal efforts are optimizations of multiplier recoding, operand representation, and reduction structure. External efforts are several types of signal gating. High-level first-order estimation and theoretical analysis are conducted to evaluate the viability of each technique.

Third, we implement the proposed schemes and related previous schemes in technology-independent structural VHDL descriptions. When necessary, VHDL generators rather than static VHDL codes are written to facilitate design optimization and provide maintenance flexibility. These VHDL designs are first mapped into Artisan TSMC 0.18μm 1.8-Volt standard-cell library and evaluated using wire-load models in Synopsys design environment. Automatic layouts are then conducted using Cadence Silicon Ensemble. For high-performance multipliers, layouts with guided floorplanning are conducted to study the effect of floorplanning. Interconnect parameters are extracted and back-annotated into Synopsys for more precise delay and power calculation. Power estimation is obtained through full-timing gate-level simulation. Experimental results with comparisons of different schemes are finally presented and analyzed.

1.5 Organization

This dissertation is organized as follows. Internal efforts are addressed in Chapter 2 (optimization of multiplier recoding), Chapter 3 (optimization of operand representation), Chapter 4 and 5 (optimization of reduction structure), respectively. External efforts are addressed in Chapter 6 and Chapter 7 (signal gating).

Chapter 2 — Present optimization techniques of multiplier recoding.

Several new radix-4 recoding designs with lower power consumption are proposed, implemented, and compared with existing radix-4 recoding designs. With improved recoding designs, the effects of radix-4 recoding versus non-recoding in multipliers are re-investigated by experiments.

Chapter 3 — Propose optimization techniques of operand representation.

Operand representation conversion schemes for two's-complement multipliers are proposed to utilize the low-power feature of sign-and-magnitude representation. Techniques to reduce the cost of conversion are proposed with power/delay tradeoff. For signed data with large dynamic range, experiments show that the conversion schemes achieve large power reductions with small area and delay overhead.

Chapter 4 — Present optimization techniques of reduction structure for array multipliers.

Structure optimization techniques for the PPR step in L-R array multipliers are proposed. These techniques include: signal flow optimization in [3:2] carry-save adder ([3:2]-CSA) based linear PPR, carry-ripple adder (CRA) for PPR, [4:2] carry-save adder ([4:2]-CSA) for PPR, and split structures. Experiments show that both power and delay are improved considerably with most of these techniques.

Chapter 5 — Propose high-performance low-power left-to-right array multipliers.

To narrow the speed gap between linear array multipliers and tree multipliers, reduction structure optimization techniques are combined to develop high-performance lower-power array multipliers. Experiments indicate that the proposed multipliers have slightly less area and power than optimized tree multipliers while keeping the same delay for $n \leq 32$. The proposed multipliers are also more regular than tree multipliers, which results in simpler design description and less custom layout efforts.

Chapter 6 — Present signal gating schemes for linear array multipliers.

To further reduce power in multipliers with large-dynamic-range input data, two-dimensional signal gating techniques are proposed and a class of signal gating schemes is generalized for traditional array multipliers. Signal gating does not change the basic structure of multipliers. Instead, multipliers are partitioned and ancillary logic is inserted along the gating boundaries. For input data with a large dynamic range, significant power reduction is achieved in the experiments.

Chapter 7 — Propose signal gating schemes for high-performance multipliers.

Two-dimensional signal gating techniques are developed for high-performance multipliers including tree multipliers and newly proposed left-to-right array multipliers. Structures are tuned to simplify signal gating design. Because high-performance multipliers have more complex structures, the gating overhead is also larger. However, the net power reduction is still considerable for input data with a large dynamic range in the experiments.

Chapter 8 — Give conclusion and future directions.

The contributions are summarized and an overview of all experiment results is given. Future research directions are discussed.

Appendix A — List abbreviations.

Appendix B — Explain design and experimental methodology.

CHAPTER 2

Optimization of Multiplier Recoding for Low Power

2.1 Introduction

The multiplier operand Y is often recoded into a radix higher than 2 in order to reduce the number of partial products. The most common recoding is radix-4 recoding with the digit set $\{-2, -1, 0, 1, 2\}$. For a series of consecutive 1's, the recoding algorithm converts them into 0's surrounded by a 1 and a (-1), which has the potential of reducing switching activity. At the binary level, there are many design possibilities. The traditional design objectives are small delay and small area. The power issues of different designs have not been addressed well. In this chapter, we focus on the effects of radix-4 recoding schemes in multipliers and optimize their designs for low power. First, we give an overview and analysis of several known recoding schemes and their designs. Then, we propose new recoding schemes and evaluate the effects of different recoding schemes in 32×32-bit radix-4 linear array multipliers. Finally, the best recoding schemes are chosen and applied into both linear array and tree multipliers to compare the effects of radix-4 recoding versus non-recoding.

Intuitively, radix-4 multipliers could consume less power than their radix-2 counterparts as recoding reduces the number of PPs to half. However, the extra recoding logic and the more complex PP generation logic may present significant overheads. In addition, recoding introduces extra unbalanced signal propagation paths because of the additional delay on the paths from operand Y to the product output. Previous work [23] showed that radix-2 [3:2]-CSA tree (Wallace tree [125]) multipliers consumed less power than Booth-recoded radix-4 multipliers although the radix-2 scheme had twice as many PPs as the radix-4 scheme. This leads us to believe that the design of recoders and PP generators plays an important role in the overall power consumption in multipliers.

There has been extensive work on radix-4 recoding with the objectives of high speed and small area [112][127][56][33]. Recently some work on low-power recoding schemes has appeared. In [6][7], it was pointed out that the method of handling "-0" in the recoding design had an impact on the power consumption. In some recoders, "-0" was implemented as "$111\cdots11+1$", which caused unnecessary switching activities. Instead, "-0" should be treated in the same way as "$+0$". Furthermore, excessive switching in the PP generation was blocked when the recoded digit is zero. In [68], the probability of zero digits after recoding was improved from 0.25 to 0.313 by changing the pattern "$1\bar{2}$" and "$\bar{1}2$" to "02" and "$0\bar{2}$", respectively. With this recoding modification, the power consumption was reduced by about 4%. As mentioned earlier, [23] showed that Booth-recoded radix-4 Wallace tree structures consumed more power than radix-2 tree structures under random test data. This is a surprising result considering that recoding reduces the number of PPs by half with a significant area reduction [98]. In [22], the gate counts of multiplier designs in [23] are presented: for 32×32-bit multiplication, a radix-4 Wallace tree had 51% more gates than a radix-2 tree; for 16×16-bit, radix-4 had 63% more gates than radix-2. Judged from this area information, the results in [23] were explainable because larger area would have larger power consumption by a high-level estimation. However, these designs of radix-4 multipliers are inefficient in area. To fairly compare radix-2 with radix-4 multipliers, efficient radix-4 multiplier designs are needed. In [52], a race-free Booth recoding design was proposed to eliminate glitches due to recoding. The principle is to balance signal propagation paths by developing the fastest possible equal paths. Such a design is not really glitch-free if buffer delay, signal fanouts and different cell pin characteristics are considered. However, this technique indicates a good direction to reduce the power and delay. This design was improved at the transistor level by dividing the complex gate into small cascade stages [35]. The complex PP generator in [52] was slightly simplified by logic transformation in [131].

To further reduce the number of PPs, operand Y is recoded into a radix higher than 4, such as 8, 16, 32. In radix higher-than-4 recoding, the bottleneck is the generation of *hard* PPs such as $3X$ which may involve a carry-propagate step [12]. In [108], Sam and Gupta presented a generalized multi-bit recoding algorithm in the two's-complement system. In [124], Vassiliadis *et. al.* proposed a signal-encoding algorithm for multibit overlapped scanning multiplication. In [12], Bewick proposed radix-8 recoding with redundant PP representations for fast multiplication. In [30], Cherkauer and Friedman proposed a hybrid radix-4/radix-8 signed multiplier architecture as a compromise between the high speed of a radix-4 multiplier architecture and the low power dissipation of a radix-8 multiplier architecture. In [3], Al-Twaijry showed that radix-4 multiplier were superior in terms of delay, while radix-8 multipliers had the smallest area-delay product. In [29], Chen *et. al.* explored the programmable FIR architectures using different radices for hardware complexity and throughput rate tradeoff. According to their analysis, the radix-4 approach would be a good choice to achieve a low hardware complexity while radix 16 or higher should be considered for a high throughput rate. In [110], Seidel *et. al.* proposed recoding schemes for flexible radix-32 and radix-256 multiplier designs that could be implemented in shorter pipeline stages. In this work, we restrict our study to radix-4 recoding as it is the most widely used scheme in multipliers.

2.2 Radix-4 Recoding

For radix-4 recoding, the popular algorithm is parallel recoding or Modified Booth recoding [83]. In parallel radix-4 recoding, Y in Equation (1.6) becomes:

$$Y = \sum_{i=0}^{n/2-1} v_i 4^i = \sum_{i=0}^{n/2-1} (-2y_{2i+1} + y_{2i} + y_{2i-1}) 4^i \qquad (2.1)$$

where $y_{-1} = 0$ and $v_i \in \{\bar{2}, \bar{1}, 0, 1, 2\}$. As all bits are recoded in parallel without data dependency, the recoded digits are available simultaneously. This process is illustrated in Figure 2.1.

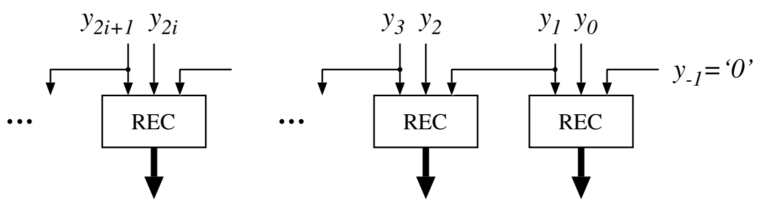


Figure 2.1: Parallel radix-4 recoding.

In linear array radix-4 multipliers, recoded digits are used serially along the array. In this case, serial recoding can be used [45]:

$$Y = \sum_{i=0}^{n/2-1} v_i 4^i \qquad (2.2)$$

$$v_i = \begin{cases} 2y_{2i+1} + y_{2i} + c_i - 4c_{i+1} & \in \{\bar{1}, 0, 1, 2\} \quad i \neq n/2-1 \\ -2y_{2i+1} + y_{2i} + c_i & \in \{\bar{2}, \bar{1}, 0, 1, 2\} \quad i = n/2-1 \end{cases} \qquad (2.3)$$

where $c_0 = 0$ and $c_i \in \{0, 1\}$. Serial recoding involves a carry-propagate process, as shown in Figure 2.2. For the last digit, a special recoding similar to parallel recoding is necessary because the most-significant bit (MSB) of Y has a negative weight. As recoded digits are generated serially and also used serially along the linear array, there are fewer glitches due to unbalanced arrival times of PP bits and reduction signals. In addition, most digits have only four possible values instead of five. However, the carry propagation in the recoder itself may introduce glitches as in the case of carry-ripple adders [80]. The asymmetry of the digit set might also complicate the logic design.

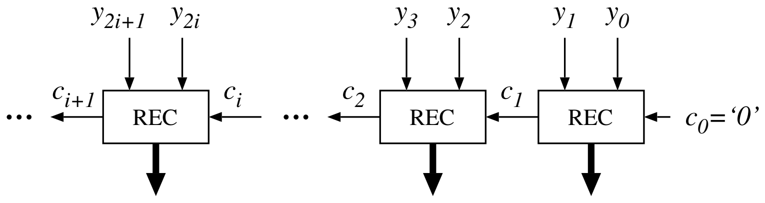


Figure 2.2: Serial radix-4 recoding.

At the gate level, these bit-level algorithms are implemented by two components: the recoder which changes Y from radix-2 to radix-4 digits and the PP generator which multiplies recoded digits with X. Digit multiplication by 2 is achieved by shifting one bit left. Negation is achieved by bit-complementing and "+1" to the least significant bit (LSB). This '+1' is implemented as a correction bit *cor* in the PP bit matrix. Because PPs are shifted signed integers in two's-complement representation, sign-extensions are required to align MSB parts of PPs. These extra bits complicate the addition step and consume extra power. They are simplified by transformation and pre-computation with the rules shown in Figure 2.3 [45]. The transformation rule in Figure 2.3a changes a sequence of sign bits into a sequence with many constant bits for pre-computation. The pre-computation rule in 2.3b computes the addition of constant bits. Shaded bits out of the range are discarded without affecting the correctness of the result. An example of the simplified radix-4 PP bit matrix for 8×10-bit two's-complement multiplication is shown in Figure 2.4.

		s=0	s=1
Original	sss . . . sss	000 . . . 000	111 . . . 111
Modified	111 . . . 11s' 000 . . . 001	▨000 . . . 000	111 . . . 111

(a)

Original	111 . . . 111 1
Reduced	▨000 . . . 000

(b)

Figure 2.3: Transformation and pre-computation rules.

$$\begin{array}{cccccccccccccccccc}
 & & & & & & & s_e' & s_e & s_e & e & e & e & e & e & e & e & e \\
 & & & & & & 1 & s_f' & f & f & f & f & f & f & f & f & & cor_e \\
 & & & & 1 & s_g' & g & g & g & g & g & g & g & g & & cor_f & & \\
 & & 1 & s_k' & h & h & h & h & h & h & h & h & & cor_g & & & & \\
1 & s_k' & k & k & k & k & k & k & k & k & & cor_h & & & & & & \\
 & & & & & & & & & cor_k & & & & & & & &
\end{array}$$

Figure 2.4: Radix-4 PP bit array in 8 × 10-bit two's-complement multiplication.

The bit matrix has the same structure independent of recoding. However, there are many design possibilities of recoders and PP generators depending on how to represent radix-4 digits and generate control signals. Multipliers using different recoding schemes have different power/area/delay characteristics. For completeness, we first analyze existing recoder and PP generator designs [5][127][6][7][56][33][45] from the view point of low power. Then we propose new low-power schemes for both parallel recoding and serial recoding. Finally we evaluate the effects of different recoding schemes in 32 × 32-bit linear array multipliers.

2.2.1 Some Existing Recoding Schemes

In parallel recoding, at least three signals are needed to represent the digit set $\{\bar{2}, \bar{1}, 0, 1, 2\}$. To achieve area/delay tradeoff, additional signals are often used. We classify existing schemes of radix-4 recoding design by the number of control signals. There are three classes: three-signal schemes, four-signal schemes, and five-signal schemes. Table 2.1 lists the names and features of all schemes studied in this Chapter, including existing schemes and newly proposed schemes. They are different in one or more aspects of parallel/serial recoding, control signals, zero handling, and logic organization.

Table 2.1: Names and features of recoding schemes studied

Name	Feature
PR3a[5]	parallel, $neg/two/one$
PR3b[127][7]	parallel, $neg/two/zero$
PR3c[56]	parallel, $neg/pos/two$
PR4a[33]	parallel, $neg/two/one/zero$
PR4b[52][131]	parallel, $neg/tz/one/z$
PR5[127]	parallel, $P2/N2/P1/N1/zero$
NPR3a	new, parallel, $neg/two/one$, unique 0
NPR3b	new, parallel, $neg/two/one$, unique 0, neg-first
SR3[45]	serial, $neg/one/zero$
NSR3a	new, serial, $neg/two/one$, unique 0
NSR3b	new, serial, $neg/two/one$, unique 0, neg-first
NSR4	new, serial, $P2/P1/N1$, hidden $zero$

Three-signal schemes

In three-signal schemes, one standard approach [5] is the *neg/two/one* based PR3a scheme in Table 2.2. The signal *cor* is the correction bit for negative numbers. The following switching expressions are deduced:

$$neg_i = y_{2i+1} \tag{2.4}$$

$$two_i = y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1} \tag{2.5}$$

$$one_i = y_{2i} \oplus y_{2i-1} \tag{2.6}$$

$$cor_i = y_{2i+1} \tag{2.7}$$

The corresponding recoder and PP generator are shown in Figure 2.5.

Table 2.2: PR3a recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	two_i	one_i	cor_i
0	0	0	$+0$	0	0	0	0
0	0	1	$+X$	0	0	1	0
0	1	0	$+X$	0	0	1	0
0	1	1	$+2X$	0	1	0	0
1	0	0	$-2X$	1	1	0	1
1	0	1	$-X$	1	0	1	1
1	1	0	$-X$	1	0	1	1
1	1	1	-0	1	0	0	1

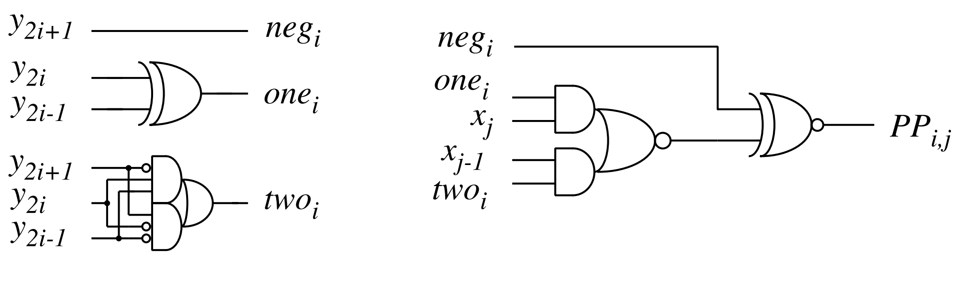


Figure 2.5: PR3a recoder and PP generator.

The advantage of this scheme is its simplicity. But the "-0" operation is implemented as $111\cdots 11+1$ rather than $000\cdots 00$. For random data, the impact of this "-0" operation on power consumption is small because the occurrence probability $P\{y_{2i+1}y_{2i}y_{2i-1} = 111\} = 0.125$ and separated "$111\cdots 11+1$" operations have limited effects on power. For input data with a large number of short-precision negative data, leading sign bits of negative numbers are recoded into a series of "-0" and cascaded "$111\cdots 11+1$" operations would increase the power consumption significantly (refer to our experimental result in Table 2.14). In [54], an equivalent *pos/two/one* recoding is used. PP generation logic is optimized at the circuit level using pass transistors. For "-0", the logic forces the output to be zero and thus no correction is needed. Instead of aggressive circuit-level optimization, explicit *zero* can be used to ensure a unique handling of "± 0", which leads to the *neg/two/zero* based PR3b scheme. One possible design [127] is shown in Table 2.3. From this table, the switching expressions are:

$$neg_i = y_{2i+1}(y_{2i}y_{2i-1})' \quad (2.8)$$

$$two_i = y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1} \quad (2.9)$$

$$zero_i = y_{2i+1}y_{2i}y_{2i-1} + y'_{2i+1}y'_{2i}y'_{2i-1} \quad (2.10)$$

$$cor_i = neg_i \quad (2.11)$$

Note that the neg_i expression in Equation 2.8 can be simplified to be $neg_i = y_{2i+1}$ because *zero* could set PP to zero regardless of *neg*. For "-0", $neg_i = y_{2i+1}$ introduces useless switching activities.

In [6][7], the switching expressions are slightly different:

$$neg_i = y_{2i+1} \quad (2.12)$$

$$zero_i = y_{2i+1}y_{2i}y_{2i-1} + y'_{2i+1}y'_{2i}y'_{2i-1} \quad (2.13)$$

$$two_i = (y_{2i} \oplus y_{2i-1})'zero'_i \quad (2.14)$$

Table 2.3: PR3b recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	two_i	$zero_i$	cor_i
0	0	0	$+0$	0	0	1	0
0	0	1	$+X$	0	0	0	0
0	1	0	$+X$	0	0	0	0
0	1	1	$+2X$	0	1	0	0
1	0	0	$-2X$	1	1	0	1
1	0	1	$-X$	1	0	0	1
1	1	0	$-X$	1	0	0	1
1	1	1	-0	0	0	1	0

$$cor_i = y_{2i+1} zero_i' \tag{2.15}$$

In these expressions, *zero* is used to block transitions in signal *two* and *cor* in PP generation logic. The recoder and PP generator are illustrated in Figure 2.6. At the circuit level, pass transistor based multiplexers are often used to implement PP generator with reduced delay, as shown in Figure 2.7. The voltage V_{cc} representing '1' would lose a threshold voltage V_{Tn} when passing NMOS transistors. A feedback PMOS transistor is used to restore the voltage level from $V_{cc} - V_{Tn}$ to V_{cc} so that the DC current in the inverter due to the reduced voltage level is eliminated, which permits fast and low-power operations [11]. To reduce the number of inverters, both control signals and their complement forms can be provided for all multiplexers.

Another three-signal recoding scheme is the *neg*/*pos*/*two* based PR3c scheme [56], as shown in Table 2.4. The switching expressions are:

$$neg_i = y_{2i+1}(y_{2i} y_{2i-1})' \tag{2.16}$$

$$pos_i = y_{2i+1}'(y_{2i} + y_{2i-1}) \tag{2.17}$$

$$two_i = (y_{2i} \oplus y_{2i-1})' \tag{2.18}$$

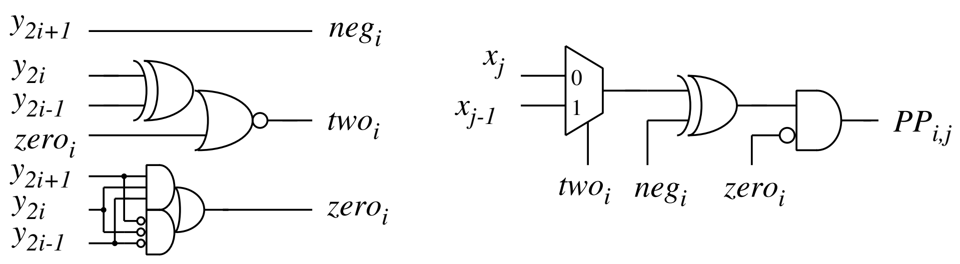


Figure 2.6: PR3b recoder and PP generator [7].

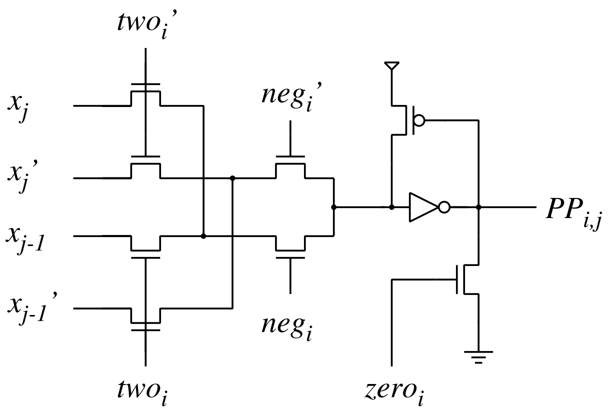


Figure 2.7: PR3b PP generator using pass transistors.

$$cor_i \quad = \quad neg_i \tag{2.19}$$

The recoder and PP generator are shown in Figure 2.8. Compared to the PR3b design, both recoder logic and PP generation logic are simpler. In PR3c, gate sharing between adjacent bits is used to reduce logic. "*zero*" operations are implicitly implemented in PP generation multiplexers.

Four-signal schemes

The second class of recoding schemes is four-signal schemes. In [33], PR4a based on *neg/two/one/zero* is used. The truth table is shown in Table 2.5 and the logic expressions are:

$$neg_i \quad = \quad y_{2i+1} \tag{2.20}$$

Table 2.4: PR3c recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	pos_i	two_i	cor_i
0	0	0	$+0$	0	0	1	0
0	0	1	$+X$	0	1	0	0
0	1	0	$+X$	0	1	0	0
0	1	1	$+2X$	0	1	1	0
1	0	0	$-2X$	1	0	1	1
1	0	1	$-X$	1	0	0	1
1	1	0	$-X$	1	0	0	1
1	1	1	-0	0	0	1	0

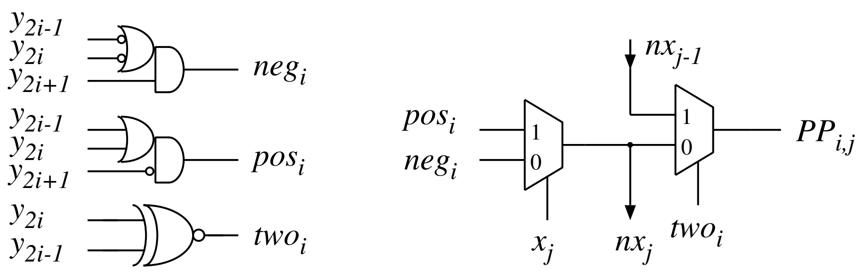


Figure 2.8: PR3c recoder and PP generator.

$$tmp1_i = y_{2i+1} \oplus y_{2i-1} \tag{2.21}$$

$$tmp2_i = y_{2i+1} \oplus y_{2i} \tag{2.22}$$

$$one_i = y_{2i} \oplus y_{2i-1} \tag{2.23}$$

$$two_i = tmp1_i \cdot tmp2_i \tag{2.24}$$

$$zero_i = (tmp1_i + tmp2_i)' \tag{2.25}$$

$$cor_i = y_{2i+1} zero_i' \tag{2.26}$$

where XOR gates are shared. For PP generation logic, two levels of multiplexers are still needed. The recoder and PP generator are illustrated in Figure 2.9. The same gate sharing technique as in PR3c is applied. Compared to PR3c, this design is more complex in both recoders and PP generators. Therefore, we will not consider this design in our work.

Table 2.5: PR4a recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	two_i	one_i	$zero_i$	cor_i
0	0	0	$+0$	0	0	0	1	0
0	0	1	$+X$	0	0	1	0	0
0	1	0	$+X$	0	0	1	0	0
0	1	1	$+2X$	0	1	0	0	0
1	0	0	$-2X$	1	1	0	0	1
1	0	1	$-X$	1	0	1	0	1
1	1	0	$-X$	1	0	1	0	1
1	1	1	-0	1	0	0	1	0

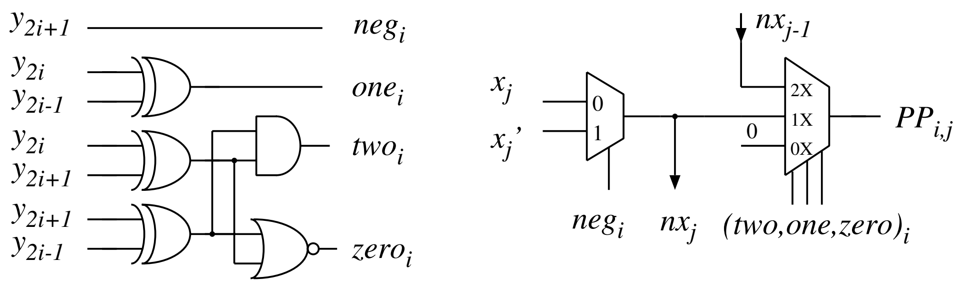


Figure 2.9: PR4a recoder and PP generator.

In [52] a glitch-free *neg/tz/one/z* based PR4b is proposed. The recoding scheme is presented in Table 2.6. *tz* covers both *two* and *zero* conditions. Extra signal *z* together with *tz* is used to set PP to zero. To achieve equal signal propagation paths, some logic is duplicated instead of being shared. The corresponding recoder and PP generators are shown in Figure 2.10 and 2.11, respectively. The PP generator in Figure 2.10b is an improved version [131] of the generator in Figure 2.10a. It is shown that the paths from both X and Y to PPs consist of only two XOR gates. These designs are not glitch-free when buffer delay, signal fanouts and different cell pin characteristics are considered. Moreover, the PP generators are more complicated than the conventional generator in Figure 2.5. However, PR4b provides good insights into low-power recoders.

Table 2.6: PR4b recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	tz_i	one_i	z_i	cor_i
0	0	0	$+0$	0	1	0	1	0
0	0	1	$+X$	0	0	1	1	0
0	1	0	$+X$	0	0	1	0	0
0	1	1	$+2X$	0	1	0	0	0
1	0	0	$-2X$	1	1	0	0	1
1	0	1	$-X$	1	0	1	0	1
1	1	0	$-X$	1	0	1	1	1
1	1	1	-0	1	1	0	1	0

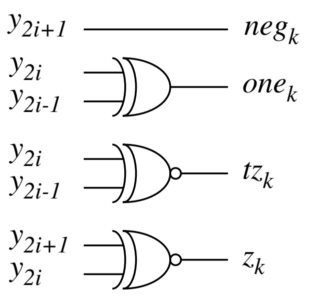
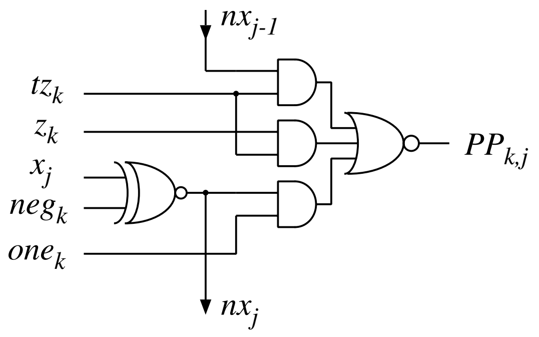


Figure 2.10: PR4b recoder.

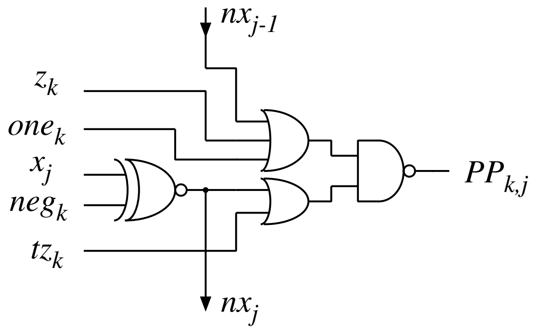
Five-signal scheme

The third class of recoding schemes is to generate five separate control signals, $P2$, $N2$, $P1$, $N1$, and $zero$, corresponding to recoding digits +2, -2, +1, -1, and 0 [127][61]. The recoding table is shown in Table 2.7 and the logic expressions are:

$$N2_i = y_{2i+1}y'_{2i}y'_{2i-1} \tag{2.27}$$



(a)



(b)

Figure 2.11: PR4b PP generators.

$$P2_i = y'_{2i+1}y_{2i}y_{2i-1} \tag{2.28}$$

$$tmp_i = y_{2i} \oplus y_{2i-1} \tag{2.29}$$

$$N1_i = y_{2i+1} \cdot tmp_i \tag{2.30}$$

$$P1_i = y'_{2i+1} \cdot tmp_i \tag{2.31}$$

$$zero_i = y_{2i+1}y_{2i}y_{2i-1} + y'_{2i+1}y'_{2i}y'_{2i-1} \tag{2.32}$$

$$cor_i = y_{2i+1}(y_{2i}y_{2i-1})' \tag{2.33}$$

For PP generation, pass transistor based 5-1 multiplexers are often used. The recoder and PP generator are shown in Figure 2.12. If standard CMOS AND/OR gates are used, signal *zero* is not necessary because it is implied when the other four signals are all 0. Note that at most 2 out of 5 control signals would change each time, which could somewhat offset the power inefficiency due to more control signals than in other designs.

Table 2.7: PR5 recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	$P2_i$	$N2_i$	$P1_i$	$N1_i$	$zero_i$	cor_i
0	0	0	$+0$	0	0	0	0	1	0
0	0	1	$+X$	0	0	1	0	0	0
0	1	0	$+X$	0	0	1	0	0	0
0	1	1	$+2X$	1	0	0	0	0	0
1	0	0	$-2X$	0	1	0	0	0	1
1	0	1	$-X$	0	0	0	1	0	1
1	1	0	$-X$	0	0	0	1	0	1
1	1	1	-0	0	0	0	0	1	0

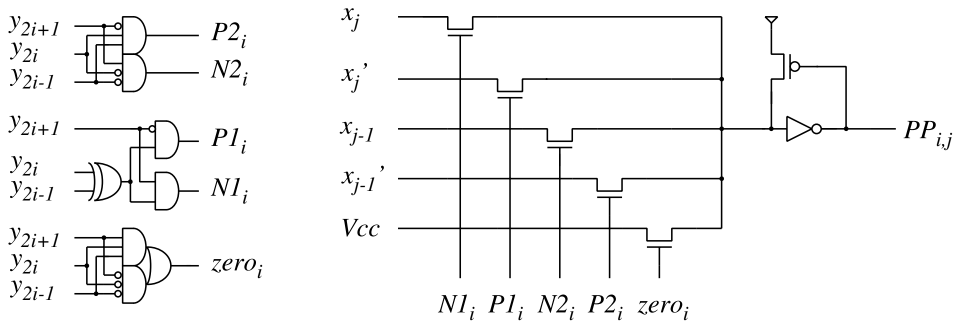


Figure 2.12: PR5 recoder and PP generator.

Serial recoding scheme

For serial radix-4 recoding, there is not much previous work because its application is limited to linear array multipliers although it has the potential of power saving. Glitches due to unbalanced signal arrival times are reduced because recoded digits are generated and used serially. The power associated with negation operation decreases as the probability of negative digits is decreased. However, the use of four possible values instead of five does not necessarily simplify the design because the digit set is not symmetric. In [45], *neg*/*one*/*zero* based SR3 corresponding to Table 2.8 was used:

$$neg_i = y_{2i+1}(y_{2i} + c_i) \tag{2.34}$$

$$one_i = y_{2i} \oplus c_i \tag{2.35}$$

$$zero_i = y_{2i+1}y_{2i}c_i + y'_{2i+1}y'_{2i}c'_i \tag{2.36}$$

$$cor_i = y_{2i+1} \cdot one_i \tag{2.37}$$

$$c_{i+1} = neg_i \tag{2.38}$$

The carry propagation in the recoders may introduce spurious transitions, which is a negative factor for power saving. The recoder and PP generator are illustrated in Figure 2.13. In this scheme, the use of four values does not result in a simpler design. As mentioned earlier, a special recoding similar to the above parallel recoding has to be used for the last digit.

Table 2.8: SR3 recoding

y_{2i+1}	y_{2i}	c_i	OP	neg_i	one_i	$zero_i$	cor_i	c_{i+1}
0	0	0	$+0$	0	0	1	0	0
0	0	1	$+X$	0	1	0	0	0
0	1	0	$+X$	0	1	0	0	0
0	1	1	$+2X$	0	0	0	0	0
1	0	0	$+2X$	0	0	0	0	0
1	0	1	$-X$	1	1	0	1	1
1	1	0	$-X$	1	1	0	1	1
1	1	1	-0	1	0	1	0	1

High-level comparison

A high-level comparison of power-related parameters in different existing recoding schemes is given in Table 2.9. The cell areas of 1-bit recoder (REC) and

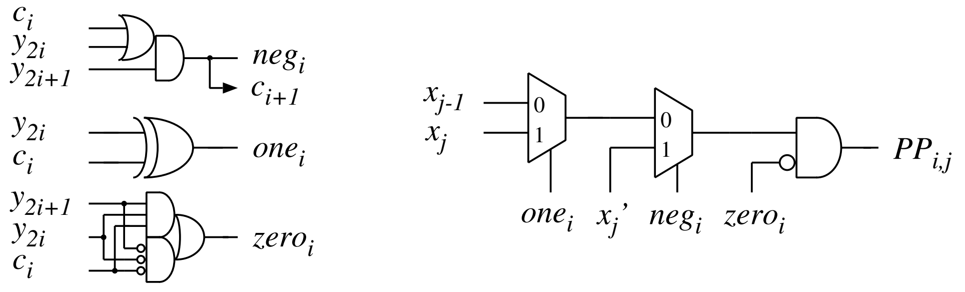


Figure 2.13: SR3 recoder and PP generator.

PP generator (PPG) are measured in μm^2 based on the Artisan standard cell library [10] described in Appendix B. The area of inverting logic is not included as we assume logic polarities are optimized in the actual implementations. The delay of REC and PPG is estimated roughly as equivalent XOR2 gate delay (T_{XOR2}) without considering fanout and signal transition directions. Simple gates such as AND2 have the delay of $0.5T_{XOR2}$. Two-level AND/OR gates and MUX21 have the same delay as XOR2. The number of control lines, N_{ctrl}, is also listed because it reflects the buffer and interconnect requirements. $u0$ indicates if there is a unique zero representation in recoding. Among existing recoding schemes, PR3a has both the simplest recoder and the simplest PP generator. The number of control lines in PR3a is the minimum 3. But PR3a does not have unique zero handling. As to delay and signal balance, PR4b is the best. Although PR4b does not have unique zero representation and some useless switching may happen internally, z and tz would jointly set the PP output to zero for "-0", which is different from PR3a. SR3 is very similar to PR3b and does not show the advantage of serial recoding.

Table 2.9: High-level comparison of existing recoding schemes

Schemes	A_{REC1b}	A_{PPG1b}	Delay	N_{ctrl}	$u0$
PR3a	53	43	3	3	N
PR3b	63	63	4	3	Y
PR3c	59.5	53	2.75	3	Y
PR4b	79.5	49.5	2	4	N
PR5	99	66	2.5	5	Y
SR3	69.5	63	3.5	3	N

2.2.2 New Recoding Schemes

After examining existing recoding schemes, we now propose new recoding schemes for low power. The guidelines are as follows. In an $m \times n$-bit radix-4 multiplier, the total area of recoders is $\frac{n}{2}A_{REC1b}$ while the total area of PP generators is $\frac{n}{2}(m+1)A_{PPG1b}$. In a 32×32-bit linear array multiplier, the recoders only occupy less than 2% of the total cell area while the PP generators occupy over 30% of the total area. Therefore, it is more important to simplify PP generators than to simplify recoders for power saving. In addition, unique zero handling is desired in order to reduce the number of unnecessary switching activities. Moreover, delay-balanced logic designs are helpful in order to reduce glitches. This is important because recoders and PP generators are the first two steps on the long logic paths in multipliers and glitches have a snow-balling effect along the paths.

New parallel recoding schemes

For parallel recoding, we propose to keep the simplicity of $neg/two/one$ PP generator and increase the complexity of $neg/two/one$ recoder to add the power-efficient unique zero handling. The new scheme is NPR3a shown in Table 2.10.

The two changes on neg_i and cor_i are bold-faced. The new switching expressions are:

$$\begin{aligned} neg_i &= y_{2i+1}(y_{2i}y_{2i-1})' && (2.39)\\ two_i &= y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1} && (2.40)\\ one_i &= y_{2i} \oplus y_{2i-1} && (2.41)\\ cor_i &= neg_i && (2.42) \end{aligned}$$

The NPR3a recoder and PP generator are shown in 2.14. The PP generator is the one in Figure 2.5. The only difference is neg_i logic in the recoder.

Table 2.10: NPR3a recoding

y_{2i+1}	y_{2i}	y_{2i-1}	OP	neg_i	two_i	one_i	cor_i
0	0	0	$+0$	0	0	0	0
0	0	1	$+X$	0	0	1	0
0	1	0	$+X$	0	0	1	0
0	1	1	$+2X$	0	1	0	0
1	0	0	$-2X$	1	1	0	1
1	0	1	$-X$	1	0	1	1
1	1	0	$-X$	1	0	1	1
1	1	1	-0	**0**	0	0	**0**

In NPR3a, the signal paths in the generator are not balanced well as *one* and *two* arrive later than X. To improve the balance and reduce glitches in subsequent PPR logic, the XNOR negation logic is exchanged with AOI22 selection logic. This exchange utilizes the arithmetic computation feature of $-(C\cdot X) = C\cdot(-X)$, which cannot be discovered by gate-level optimizations. A gate-level approach to improve the balance is to insert some delay buffers on the fast signals, which represents an external effort with area overhead and will not be considered here. The resulting scheme NPR3b are illustrated in Figure 2.15. In NPR3b, the neg_i

expression in Equation 2.39 can be simplified to be $neg_i = y_{2i+1}$ because two and one could set PP to zero regardless of neg. For "-0", however, $neg_i = y_{2i+1}$ introduces more switching activities because it triggers switching in XNOR gate while the final PP output remains at 0. Compared to the PR4b PP generator in Figure 2.11, this new PP generator has simpler logic and one less control signal.

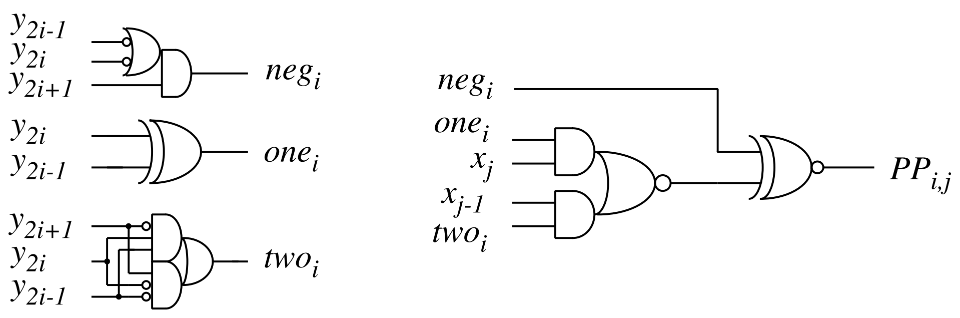


Figure 2.14: NPR3a recoder and PP generator.

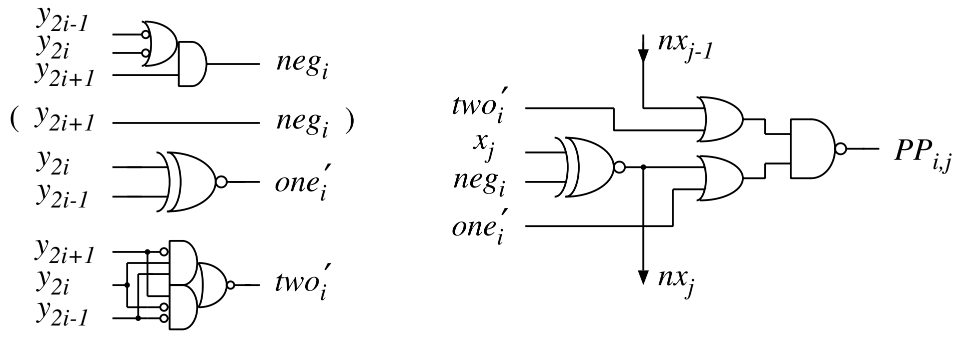


Figure 2.15: NPR3b recoder and PP generator.

New serial recoding schemes

For serial recoding, $neg/two/one$ with unique zero handling can also be applied. The recoding relationship is shown in Table 2.11 and the logic expressions are:

$$two_i \quad = \quad y'_{2i+1}y_{2i}c_i + y_{2i+1}y'_{2i}c'_i \qquad (2.43)$$

$$one_i = y_{2i} \oplus c_i \tag{2.44}$$

$$neg_i = y_{2i+1} \cdot one_i \tag{2.45}$$

$$cor_i = neg_i \tag{2.46}$$

$$c_{i+1} = y_{2i+1}(y_{2i} + c_i) \tag{2.47}$$

This serial recoder is illustrated in 2.16. The PP generators are the same as in NPR3a (Figure 2.14) or NPR3b (Figure 2.15). Correspondingly, the two new serial recoding schemes are named NSR3a and NSR3b.

Table 2.11: NSR3 recoding

y_{2i+1}	y_{2i}	c_i	OP	neg_i	two_i	one_i	cor_i	c_{i+1}
0	0	0	$+0$	0	0	0	0	0
0	0	1	$+X$	0	0	1	0	0
0	1	0	$+X$	0	0	1	0	0
0	1	1	$+2X$	0	1	0	0	0
1	0	0	$+2X$	0	1	0	0	0
1	0	1	$-X$	1	0	1	1	1
1	1	0	$-X$	1	0	1	1	1
1	1	1	-0	0	0	0	0	1

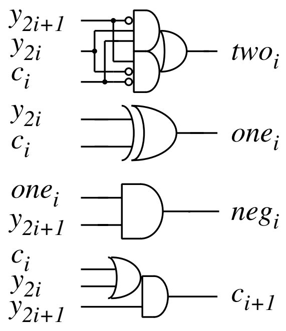


Figure 2.16: NSR3 serial recoder.

Since there are only four possible values, it is attractive to use 4-1 multiplexers. Similar to PR5 with $P2/N2/P1/N1/zero$, NSR4 with $P2/P1/N1/zero$ is developed for serial recoding. The recoding table is shown in Table 2.12. The logic expressions are:

$$P2_i = y'_{2i+1}y_{2i}c_i + y_{2i+1}y'_{2i}c'_i \quad (2.48)$$

$$tmp_i = y_{2i} \oplus c_i \quad (2.49)$$

$$N1_i = y_{2i+1} \cdot tmp_i \quad (2.50)$$

$$P1_i = y'_{2i+1} \cdot tmp_i \quad (2.51)$$

$$zero_i = y_{2i+1}y_{2i}c_i + y'_{2i+1}y'_{2i}c'_i \quad (2.52)$$

$$cor_i = N1_i \quad (2.53)$$

$$c_{i+1} = y_{2i+1}(y_{2i} + c_i) \quad (2.54)$$

The recoder and pass transistor based PP generator are shown in Figure 2.17. If AND/OR gates are used, signal *zero* is redundant as it is implied when $P2$, $P1$, and $N1$ are all 0. In this case, the recoder and PP generator are simplified to become NSR4 with $P2/P1/N1$ in Figure 2.18. Now the PP generator is just an AO222 gate, the simplest among all schemes. In Artisan library, the area of AO222 (AOI222 + INV) is $33\mu m^2$. Because of the simplified PP generator, NSR4 is expected to have the least power consumption among serial schemes. NSR4 demonstrates the advantage of using only four values in serial recoding.

High-level comparison

A high-level comparison of power-related parameters in the proposed new recoding schemes is given in Table 2.13. NSR4 is the scheme with hidden *zero*. All proposed schemes here have three control signals, unique zero handling, and small A_{PPG1b}. Although serial recoding only has four possible values, NSR3a and NSR3b are not superior to NPR3a and NPR3b because the asymmetry of the digit set and the carry bit complicate the logic. With hidden *zero*, NSR4 takes the advantage of the small digit set and thus has the smallest A_{PPG1b}.

Table 2.12: NSR4 recoding

y_{2i+1}	y_{2i}	c_i	OP	$P2_i$	$P1_i$	$N1_i$	$zero_i$	cor_i	c_{i+1}
0	0	0	$+0$	0	0	0	1	0	0
0	0	1	$+X$	0	1	0	0	0	0
0	1	0	$+X$	0	1	0	0	0	0
0	1	1	$+2X$	1	0	0	0	0	0
1	0	0	$+2X$	1	0	0	0	0	0
1	0	1	$-X$	0	0	1	0	1	1
1	1	0	$-X$	0	0	1	0	1	1
1	1	1	-0	0	0	0	1	0	1

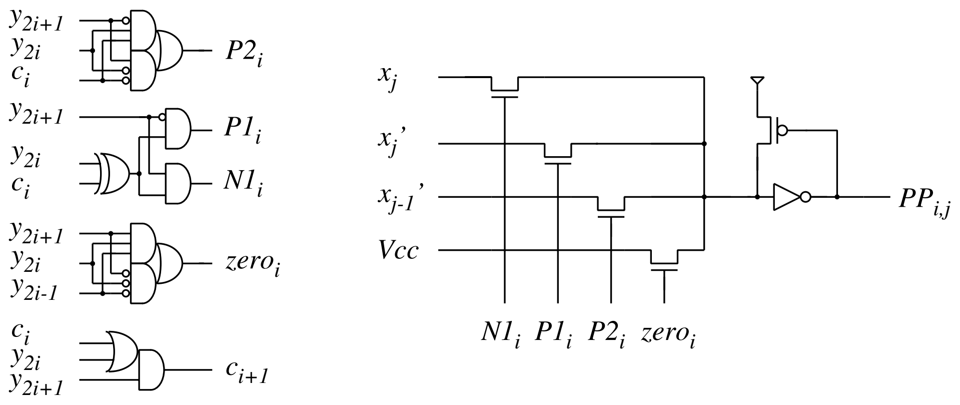


Figure 2.17: NSR4 recoder and PP generator with $P2/P1/N1/zero$.

2.2.3 Experimental Evaluation

The recoder itself is small and does not consume much power. In a radix-4 32×32-bit linear array multiplier, the recoder consumes less than 1% of the total power. Even the PP generator only consumes about 10% of the total power. However, the design of recoding schemes affects the power dissipation in subsequent PPR and final CPA significantly because recoding is the the first step on the long logic paths in multipliers. Therefore, comparisons of individual recoders and PP generators are not enough. For this reason, we have implemented

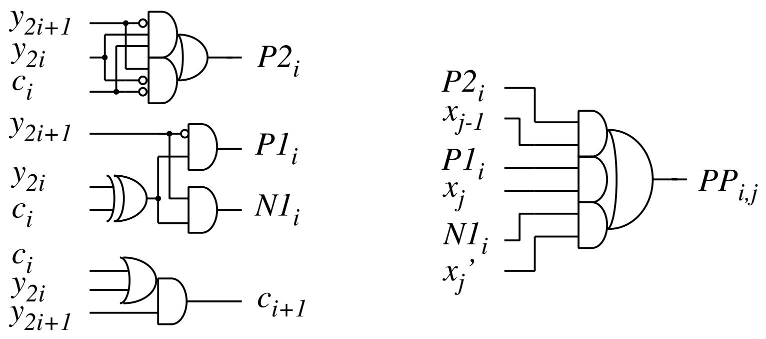


Figure 2.18: NSR4 recoder and PP generator with hidden *zero*.

Table 2.13: High-level comparison of proposed recoding schemes

Schemes	A_{REC1b}	A_{PPG1b}	Delay	N_{ctrl}	$u0$
NPR3a	66	43	3	3	Y
NPR3b	66	46.5	2.75	3	Y
NSR3a	79.5	43	3	3	Y
NSR3b	79.5	46.5	3.5	3	Y
NSR4	89.5	33	2.5	3	Y

and simulated six existing recoding schemes and five new recoding schemes in a 32×32-bit linear array multiplier framework. The final CPA is a two-level carry-lookahead adder (CLA). All multipliers are described in structural VHDL using technology-independent switching expressions. Multipliers differ only in the recoding schemes. These VHDL designs are optimized and mapped into Artisan $0.18\mu m$ standard cell library using Synopsys Design Compiler. Automatic layouts are then conducted using Cadence Silicon Ensemble. Interconnect parameters are extracted and back-annotated into Synopsys tools for delay and power calculation. Two test data sets are used: *random* and *djpeg* with a large dynamic range. The complete design and experimental methodologies are described in

Appendix B.

2.2.3.1 Effects of Parallel Recoding in Linear Array Multipliers

The power/area/delay comparison results of 32×32-bit linear array multipliers with different parallel recoding schemes are listed in Table 2.14. The values in parentheses are normalized values. The smallest value of each characteristic is highlighted in boldface. As multipliers in this experiment are only different in recoding schemes, we use the names of recoding schemes to distinguish different multipliers. The power consumption is measured at $50MHz$. The static power is less than $1\mu W$ in all cases. As static power is irrelevant to the clock frequency and is less than 0.1% of the total power, only dynamic power is used for comparison. *CArea* is the total cell area. The actual chip area is the cell area divided by the row utilization rates. The row utilization rates for layouts are initially set at a high value (85%), and decrease 5% each time if the routing cannot be completed. The rate with the densest successful layout is named as a *routable rate*. In this experiment, the routable rate for the NPR3b multiplier is 80%, the rate for NPR3a, PR5, and PR4b multipliers is 75%, and the rate for others is 70%. In this experiment, we find no obvious relation between the areas and the routable rates. The chip area is not used directly in comparison because the varying routable rates affect the accuracy of the information that can be derived from the chip area for power analysis.

The experimental results show that small PPG area and unique zero handling play important roles in power reduction. The relationship of $CArea$s roughly matches the relationship of A_{PPG1b}s in Table 2.9 and 2.13. The smaller the A_{PPG1b}, the smaller the total area. This verifies that the area of PPG is more important than that of REC. The delay relationship does not match well because

Table 2.14: Comparison of multipliers with different parallel recoding schemes

Schemes	Power(mW) ($50MHz$)		CArea(μm^2)	Delay(ns)
	Random	Djpeg		
PR3a	29.72 (1.00)	18.66 (1.00)	**71182 (1.00)**	11.00 (1.00)
PR3b	30.80 (1.04)	15.54 (0.83)	80219 (1.13)	11.15 (1.01)
PR3c	27.88 (0.94)	12.69 (0.68)	75306 (1.06)	10.73 (0.98)
PR4b	29.82 (1.00)	14.34 (0.77)	73783 (1.04)	10.93 (0.99)
PR5	26.83 (0.90)	12.59 (0.67)	79757 (1.12)	10.74 (0.98)
NPR3a	27.32 (0.92)	**12.57 (0.67)**	71358 (1.00)	10.31 (0.94)
NPR3b	**26.13 (0.88)**	12.64 (0.68)	71907 (1.01)	**10.19 (0.93)**

the difference of logic delay is at most $2T_{XOR2}$ while the mapping and layout steps have many factors that may affect the overall delay. Unique zero handling is very efficient for power reduction under *djpeg* test data. PR3a consumes up to 48% power than other schemes under *djpeg*. From PR3a to NPR3a, 33% power reduction is achieved due to unique zero handling. Small area is also helpful in power saving. From PR3b to PR3c, the $10 \sim 20\%$ power reduction mainly comes from the 7% area reduction as both are three-signal schemes with unique zero handling. From PR3b/PR3c to NPR3a/NPR3b, the area reduction is $5 \sim 10\%$ and the power reduction is up to 20%. PR5 is an exception: PR5 has 11% larger area than NPR3a but consumes similar power as NPR3a. This is because control signal *zero* is eliminated in actual implementation due to AND/OR expressions and at most 2 signals would change for each new computation. Finally, smaller delay and more balanced paths also help reduce the power consumption. From NPR3a to NPR3b, 4% power reduction under *random* data is due to more balanced signal paths in NPR3b.

Overall, NPR3b is a good choice for parallel recoding in terms of power, delay, and area. Note that this conclusion is valid when a similar standard cell library is used. In full-custom design with aggressive MUX and XOR implementations, PR5 and PR3c might be competitive choices besides NPR3b.

2.2.3.2 Effects of Serial Recoding in Linear Array Multipliers

The comparison results of 32×32-bit linear array multipliers with different serial recoding schemes are listed in Table 2.15. The names of recoding schemes are used to distinguish different multipliers. The values in parentheses are normalized values. NSR4 multiplier uses NSR4 recoding with hidden $zero$. The routable rate for SR3 is 75% and the rate for NSR3a is 70%. The rate for NSR3b and NSR4 is 80%.

Table 2.15: Comparison of multipliers with different serial recoding schemes

Schemes	Power(mW) ($50MHz$)		CArea(μm^2)	Delay(ns)
	Random	Djpeg		
SR3	31.43 (1.00)	13.48 (1.00)	71431 (1.00)	10.97 (1.00)
NSR3a	28.18 (0.90)	14.35 (1.06)	71657 (1.00)	11.30 (1.03)
NSR3b	30.07 (0.96)	14.14 (1.05)	72233 (1.01)	11.21 (1.02)
NSR4	**25.59 (0.81)**	**11.41 (0.85)**	**71019 (0.99)**	**10.60 (0.97)**

The NSR4 multiplier has the least power, area, and delay. The other three multipliers, SR3, NSR3a, and NSR3b consume $10 \sim 26\%$ more power than NSR4. NSR4 is the only scheme that takes advantage of the four digit values in serial recoding. The other three schemes use the same PP generators as their counterparts in parallel recoding while the recoder designs are more complex. Compared to NPR3a and NPR3b, NSR3a and NSR3b multipliers consume up to 15% more power. As recoder itself does not consume much power, the extra power consumption comes from glitches due to carry propagation in serial recoding. In NSR4, the recoder is simplified to produce three control signals and the PP generator is very simple. Despite the carry propagation in the recoder, the overall power consumption in NSR4 is still smaller than that in parallel recoding schemes. Therefore, NSR4 is a good choice for low-power linear array multipliers. When there is not enough time for serial recoding, parallel recoding schemes such as NPR3b are good candidates.

2.3 Comparison of Radix-4 and Radix-2 Multipliers

Radix-4 recoding reduces the number of PPs from n to $n/2$, at the cost of one $(n/2)$-bit recoder and $n/2$ $(m+1)$-bit PP generation logic. A rough estimation of the difference in cell area between radix-4 and radix-2 multipliers is:

$$\Delta A = \frac{n}{2}A_{REC1b} + \frac{n}{2}(m+1)A_{PPG1b} - nmA_{AND2} - \frac{n}{2}mA_{[3:2]CSA1b} \tag{2.55}$$

If MUX based full adders (FAs) are used for [3:2]-CSAs, $A_{FA} = A_{[3:2]CSA1b} = 79.5\mu m^2$ based on Artisan standard cell library [10]. For parallel recoding NPR3b, $A_{REC1b} = 66\mu m^2$ and $A_{PPG1b} = 46.5\mu m^2$. $A_{AND2} = 13\mu m^2$. If $m = n = 32$, $\Delta A = -28,408\mu m^2$. The areas of PPG and PPR modules in an $n \times n$-bit radix-2 two's-complement linear array multiplier are estimated as:

$$A_{PPG} = (n^2 - 2n + 2)A_{AND2} + (2n-2)A_{NAND2} \tag{2.56}$$

$$A_{PPR} = (n^2 - 3n + 2)A_{FA} + (n-1)A_{HA} \tag{2.57}$$

Buffers are not included. $A_{NAND2} = 10\mu m^2$. $A_{HA} = 39.5\mu m^2$. For $n = 32$, the total area of PPG and PPR, $A_{PPGR} = 88,286\mu m^2$. It can be verified that the areas of tree multipliers are similar to those of linear array multipliers. By using radix-4 recoding, the PPGR area is reduced by $\Delta A/A_{PPGR} = 32\%$ for 32×32-bit multipliers. Although recoding introduces unbalanced signal propagation paths due to the recoding on operand Y, we believe the overall power consumption will still be reduced because of the area advantage in radix-4 multipliers. This conclusion is opposite to the previous work in [23].

In our comparison of radix-2 versus radix-4, three widely used structures for PP reduction are considered: [3:2]-CSA linear array, [3:2]-CSA tree and [4:2]-CSA tree. Tree multipliers have the smallest logic delay proportional to $log(n)$. However, they have irregular layout with complicated interconnects. On the other hand, linear array multipliers have larger delay but offer regular layout and simpler interconnects. Between [3:2]-CSA tree and [4:2]-CSA tree, [4:2]-CSA tree is more popular due to its regularity and smaller delay [126][93][89][55][98][33]. A [4:2]-CSA has the same gate complexity as two [3:2]-CSAs. But a [4:2]-CSA is faster than two cascaded [3:2]-CSAs because an efficient design could have 3 XOR-gate delay while each single [3:2]-CSA has 2 XOR-gate delay. For odd numbers of PPs, the use of [3:2]-CSA tree is sometimes better than the use of [4:2]-CSA tree because [4:2] reduction may waste resources for odd numbers. Parallel recoding NPR3b is used for all 32×32-bit multipliers studied here. NSR4 is not used because not all schemes have enough time for serial recoding. The final CPA in all multipliers is a two-level CLA. The optimization of final CPAs will be addressed in Chapter 5. The power consumption is tested under both *djpeg* and *random*. For linear array multipliers, the power consumption is measured at 50MHz, which is enough for the slowest schemes to finish computing in the worst case. For fast tree multipliers, the power is measured at 100MHz. The comparison results for linear array multipliers and tree multipliers are listed in Table 2.16 and 2.17, respectively. The values in parentheses are normalized values. The routable utilization rates are also listed for comparison.

Table 2.16: Comparison of radix-2 and radix-4 32 × 32-bit array multipliers

Schemes	Power(mW) ($50MHz$)		CArea(μm^2)	Delay(ns)	Rate
	Random	Djpeg			
r2-array	56.95 (1.00)	21.07 (1.00)	93721 (1.00)	15.92 (1.00)	85%
r4-array	**26.13 (0.46)**	**12.64 (0.60)**	**71907 (0.77)**	**10.19 (0.64)**	80%

Table 2.17: Comparison of radix-2 and radix-4 32 × 32-bit tree multipliers

Schemes	Power(mW) ($100MHz$)		CArea(μm^2)	Delay(ns)	Rate
	Random	Djpeg			
r2-tree3to2	62.15 (1.00)	24.41 (1.00)	97390 (1.00)	9.02 (1.00)	65%
r4-tree3to2	**39.18 (0.63)**	**18.88 (0.77)**	**73537 (0.76)**	**7.74 (0.86)**	70%
r2-tree4to2	63.43 (1.00)	24.14 (1.00)	100101 (1.00)	**8.33 (1.00)**	70%
r4-tree4to2	**42.57 (0.67)**	**19.81 (0.82)**	**75130 (0.75)**	8.65 (1.04)	70%

After radix-4 recoding, the overall cell area is reduced about 25% in both array multipliers and tree multipliers, which matches our our high-level estimation above if final CPA and buffer area are included. In linear array multipliers, the delay is also reduced 36% from radix-2 to radix-4 because the number of PPs along the reduction path is reduced to half. However, the delay does not change much in tree multipliers. From radix-2 to radix-4 [3:2]-CSA tree multipliers, AND gates for PPG and two levels of [3:2]-CSAs for PPR are replaced by radix-4 recoders and PP generators, which leads to about $1.5T_{XOR2}$ gate delay reduction. The delay reduction in the experimental result is larger (14%) because it includes the layout effects. In radix-4 [4:2]-CSA tree multipliers, AND gates for PPG and one level of [4:2]-CSAs are replaced by radix-4 recoders and PP generators. The delay is unchanged because the delay of radix-4 recoding is similar to that of a [4:2]-CSA ($3T_{XOR2}$). In our experiment, [4:2]-CSA tree multipliers are slightly worse than [3:2]-CSA tree multipliers because one extra level of [3:2]-CSAs is used to add the single correction bit on the last row (Figure 2.4) in [4:2]-CSA tree multipliers. To handle this bit, a [9:4] adder with $3T_{XOR2}$ can be developed, which will be addressed in Chapter 5. In [3:2]-CSA tree multipliers, this correction bit is added earlier without introducing an extra level. In this experiment, the routable utilization rates roughly reflect the complexity of distinct architectures. In general, tree multipliers are irregular and more complex than linear array multipliers. Thus, the routable rates for tree multipliers are $65 \sim 70\%$ while the rates for linear array multipliers are $80 \sim 85\%$. From radix-2 to radix-4, the change of routable rates depends on the relative influences of area reduction and complicated structures.

Regarding power in linear array multipliers, radix-4 consumes 54% less power than radix-2 under *random* and 40% less power under *djpeg* because of the 23% area reduction and 36% delay reduction. Regarding power in [3:2]-CSA tree multipliers, radix-4 consumes 37% less power under *random* and 23% under *djpeg* because of the 24% area reduction and 14% delay reduction. Regarding power in [4:2]-CSA tree multipliers, radix-4 consumes 33% less power under *random* and 18% under *djpeg* because of the 25% area reduction. The effects of radix-4 recoding on power in tree multipliers are smaller than the effects in array multipliers because radix-4 recoding does not reduce the delay in tree multipliers. Note that we haven't considered any circuit-level delay balancing techniques here. We have just shown the algorithmic advantage of radix-4 schemes over radix-2 schemes. When the circuit-level balancing techniques [107][87][72][16] are used, the power consumption in radix-4 schemes can be further reduced at the cost of extra chip area and circuit design effort. Our conclusion here conflicts with the previous work in [23] where random test data were used. As indicated in Section 2.1, the area of radix-4 Wallace tree multipliers in [23] is much larger than the area of radix-2 multipliers, which is generally not acceptable.

The experiments also show that tree multipliers are better than linear array multipliers using the same radix in terms of delay and power. For radix-2, tree multipliers consume 43 $\sim$ 48% less delay and 42 $\sim$ 45% less power (using the same frequency) than array multipliers. For radix-4, tree multipliers consume 15 $\sim$ 18% less delay and 19 $\sim$ 25% less power than array multipliers. The power saving in tree multipliers comes from the delay reduction as tree multipliers even has larger area than linear array multipliers. The cell areas of tree multipliers are only 2 $\sim$ 7% more than those of linear array multipliers. But the complex tree structure decreases the routability of tree multipliers by 10 $\sim$ 20%. Thus, the total chip areas of tree multipliers are 20 $\sim$ 36% more than those of array multipliers.

2.4 Summary

In this chapter, we have studied the power characteristics of existing radix-4 recoding schemes and proposed several new recoding schemes. The techniques we have considered for low-power recoding are: simple PP generation, unique-zero handling, and balanced logic design. Besides the traditional parallel recoding schemes, serial recoding schemes are studied for linear array multipliers. By experimental evaluation of the effects of different recoding schemes in linear array multipliers, we find that NPR3b is a good choice for parallel recoding in terms of power, delay, and area if a similar standard cell library is used. For linear array multipliers, serial recoding NSR4 with hidden *zero* is a good choice for low power because of the simplified digit set.

With optimized recoding designs, the effects of radix-4 recoding versus non-recoding in multipliers are then studied. Both linear array multipliers and tree multipliers are considered. Due to the reduced number of PPs, the area is reduced 25% by radix-4 recoding in 32×32-bit multipliers. In linear array multipliers, the delay is also reduced 36%. The power consumption in linear array multipliers is reduced $40 \sim 54\%$ under two test data sets. The power consumption in tree multipliers is reduced $18 \sim 36\%$ under two test data sets. Overall, radix-4 recoding is a good choice for low-power multiplier design.

CHAPTER 3

Optimization of Multiplier Operand Representations for Low Power

3.1 Introduction

Number representation of multiplication operands affects multiplier design and its power consumption. The use of two's-complement (2C) representation for multiplication inputs and outputs is popular in digital arithmetic systems because 2C is convenient for additions of signed numbers that happen before or after multiplication. As to internal data representation within a multiplier, however, representations other than 2C are possible. Sign-and-magnitude (SM) representation and signed digit (SD) representation are two other widely considered candidates. When the radix is 2, the SD representation is also called redundant binary (RB) representation with digit set $\{-1, 0, 1\}$. Because of the redundancy in SD representation, SD usually consume more area and power than SM and 2C representations [67][9]. Therefore, we focus on 2C and SM representations. For positive numbers, there is no difference between SM and 2C representations. For negative numbers, however, all bits higher than the actual precision are extended 1's in 2C representation, while these bits remains at 0 in SM representation. These 0's in SM reduce the switching activities in multipliers considerably. In addition, zero is a natural gating signal in AND/NAND gates which are used as

PP generation logic in radix-2 multipliers. Overall, numbers in SM representation have lower switching frequency than numbers in 2C representation, especially for multimedia data [105]. Table 3.1 gives the power distribution in a 32×32-bit radix-2 2C linear array multiplier (without final CPA) under *djpeg* test data. Each cell(i, j) on the grid consists of an AND/NAND gate and a full adder. Only powers in cells with $(i + 1)$ and $(j + 1)$ being multiples of 4 are shown for clarity. The (3,31)-(31,3) diagonal cells are highlighted as they consume the largest power in each column (except cells in the last row). The precisions of most 32-bit *djpeg* data are less than 16 bits and about 30% data are negative numbers (see Appendix B). Because of the 2C representation, the left and bottom portions of the 2C multiplier waste a large amount of power. If SM representation is used and the multiplier core is an unsigned multiplier, the power consumptions in the left and bottom portions would be much smaller because the sign extension bits remain at 0. The abnormally high power consumptions in last-row cell$(31, i)$ are due to inverted PP bits on this row that have more probabilities of being '1' than being '0'.

Table 3.1: Power distribution in 2C linear array multiplier under *djpeg* data

(i, j)	31	27	23	19	15	11	7	3
3	**0.52**	5.72	5.73	5.72	5.72	5.72	5.81	6.42
7	0.46	**9.74**	8.92	8.92	8.91	8.91	9.28	12.00
11	0.27	8.91	**12.20**	10.50	11.00	10.60	11.10	13.90
15	0.33	7.90	11.50	**14.20**	11.90	11.90	11.90	16.20
19	0.33	7.21	9.43	13.00	**14.60**	12.00	12.40	14.60
23	0.33	7.21	8.27	10.90	12.80	**14.40**	12.20	15.20
27	0.33	7.21	8.27	9.67	11.00	12.70	**14.40**	15.20
31	0.53	10.30	12.20	14.00	14.60	15.50	18.90	**24.20**

In this work, we consider how to utilize the low-power feature of SM representation in multipliers with 2C interface. A natural idea is to convert 2C data to SM data (2C-SM), use unsigned multipliers as the computation core, and then convert the result back to 2C representation (SM-TC). Despite of the simplicity of this idea, it is not clear what are good conversion schemes and if there is a need to change the multiplication core. Moreover, how to apply the conversion efficiently in radix-4 multipliers is a problem because the recoding produces negative partial products. This chapter addresses these problems.

There has been some previous work on the power features of SM, 2C, and RB representations. Estimating from word-level statistics, Ramprasad *et. al.* [105] showed that the SM representation of multimedia signals has lower switching frequency than 2C. They focused upon the problem of high-level power estimation. No actual multiplier architectures are considered. Zheng and Albicki [136] proposed mixed number representations for radix-4 2C multiplication. In this scheme, the multiplicand is converted from 2C to SM while the multiplier is directly recoded into radix-4 $\{-2, -1, 0, 1, 2\}$. The SM PPs are then converted to signed digit $\{-1, 0, 1\}$ and redundant binary adder tree is used for PP reduction. The authors claimed significant improvement over previous tree multipliers by analytical estimation. However, there is a problem in their work, which will be addressed in Section 3.3 in detail. In [67], Keane *et. al.* compared several multiplication schemes using different operand representations. They found that multipliers using signed binary number representation is the worst in both area and power. Recently, Angel and Swartzlander [9] compared carry save representation with redundant binary representation in Booth-recoded radix-4 linear array multipliers. Their experiment showed that the redundant binary scheme in [60] consumes 32% more power than the carry save scheme but the power-delay product is better. This comparison of power-delay products is not fair because the redundant binary scheme is a [2:1] reduction while the carry save scheme is a [3:2] reduction. A better comparison is to compare with a scheme using [4:2] reduction. Here we have skipped Angel's result about the EBCA scheme because EBCA is not correct [46].

3.2 Representation Conversion for Radix-2 2C Multipliers

When two operands are converted from 2C representation to SM representation, the computation core is actually an unsigned multiplier with two sign bits treated separately. For short-precision data, the unsigned multiplier has a lower switching activities as leading bits of both operands often remain at 0 instead of switching between 0 and 1.

3.2.1 2C-SM-2C: A Straightforward Conversion

A straightforward approach is to convert both X and Y from 2C representation in Equation (1.5)-(1.6) to SM:

$$H = (-1)^{s_h} \sum_{j=0}^{m-1} h_j 2^j \tag{3.1}$$

$$K = (-1)^{s_k} \sum_{i=0}^{n-1} k_i 2^i \tag{3.2}$$

where sign bit $s_h = s_x = x_{m-1}$ and $s_k = s_y = y_{n-1}$. Note that one extra bit is needed in SM representation because (-2^{n-1}) is not representable in n-bit SM. Then, SM multiplication $Z = H \times K$ is performed:

$$\begin{aligned} s_z &= s_p = (-1)^{s_h + s_k} && (3.3) \\ |Z| &= |H||K| = \sum_{j=0}^{m-1} h_j 2^j \times \sum_{i=0}^{n-1} k_i 2^i && (3.4) \end{aligned}$$

Finally, Z in SM representation is converted back to P in 2C. This process is illustrated in Figure 3.1. The sign bit is computed separately as

$$s_z = s_h \oplus s_k \tag{3.5}$$

We assume that two registers store input data. The conversion logic is inserted in the previous stage before input registers. Three considerations account for such a partition. First, the multiplier core often has the largest combinational delay and thus determines the clock frequency. By placing the 2C-SM logic into the previous stage which probably has less combinational delay, the clock frequency would not be affected very much. Second, unbalanced input arrivals to the multiplication core are avoided through registering. Third, the lower switching frequency of SM signals helps reduce power consumption in registers.

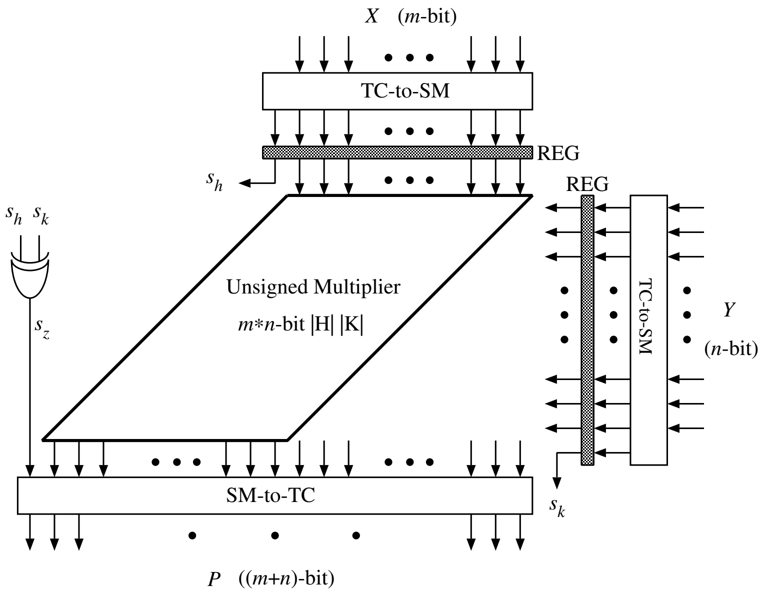


Figure 3.1: 2C-SM-2C conversion scheme.

From 2C to SM, each operand is first extended with one sign bit. If the number is negative, every bit except the most significant bit is complemented and ‘1’ is added to the least significant bit:

$$H = \begin{cases} (x_{m-1}, x_{m-1}, x_{m-2}, x_{n-3}, \cdots, x_0) & \text{if } \ x_{m-1} = 0 \\ (x_{m-1}, x'_{m-1}, x'_{m-2}, x'_{n-3}, \cdots, x'_0) + 1 & \text{if } \ x_{m-1} = 1 \end{cases} \tag{3.6}$$

From SM to 2C, the operation is the same except that no bit extension is necessary. An intuitive design of the conversion logic is shown in Figure 3.2. The signals in 2C-SM are expressed as

$$C_0 = x_{m-1} \tag{3.7}$$

$$C_{i+1} = C_i(x_i \oplus C_0) \qquad \text{for } \ i = 1, 2, \cdots, m-2 \tag{3.8}$$

$$h_i = C_i \oplus (x_i \oplus C_0) \qquad \text{for } \ i = 0, 1, \cdots, m-2 \tag{3.9}$$

$$h_{m-1} = C_{m-1} \tag{3.10}$$

$$s_h = x_{m-1} \tag{3.11}$$

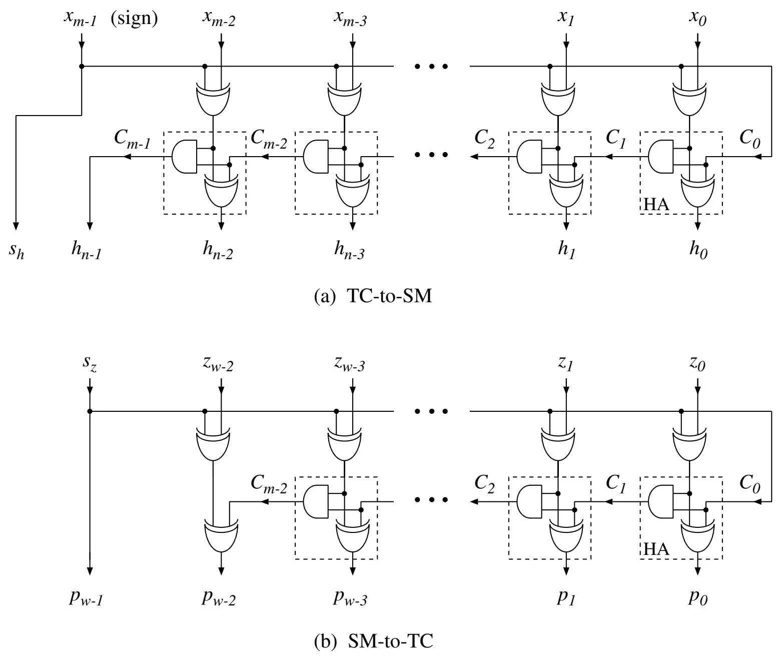


Figure 3.2: Representation conversion logic for 2C-SM-2C: (a) 2C-to-SM; (b) SM-to-2C

This intuitive design is further simplified by logic optimization of the carry signals. The non-recursive form of carry signal in Equation (3.8) is

$$C_{i+1} = (x_i \oplus C_0)(x_{i-1} \oplus C_0) \cdots (x_0 \oplus C_0)C_0$$

which is simplified to

$$C_{i+1} = x_i' x_{i-1}' \cdots x_0' C_0$$

Then, h_i is rewritten as

$$h_i = x_i \oplus ((x_{i-1} + x_{i-2} + \cdots + x_0)C_0)$$

resulting in the new recursive expressions

$$\begin{aligned}
R_1 &= h_0 = x_0 && (3.12)\\
R_{i+1} &= R_i + x_i \qquad \text{for } \ i = 1, \cdots, m-2 && (3.13)\\
h_i &= x_i \oplus (R_i \cdot x_{m-1}) \quad \text{for } \ i = 1, \cdots, m-2 && (3.14)\\
h_{m-1} &= x_{m-1} \cdot R_{m-1}' && (3.15)\\
s_h &= x_{m-1} && (3.16)
\end{aligned}$$

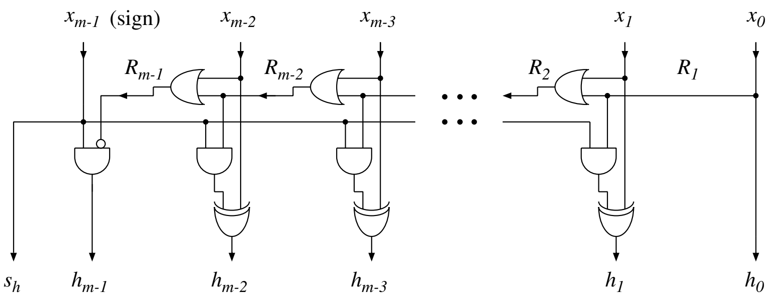


Figure 3.3: Linear structure of simplified 2C-SM conversion.

A linear structure for Equation (3.12)-(3.16) is illustrated in Figure 3.3. The delay of this structure is

$$t_{linear} = (m-3)t_{OR2} + t_{AND2} + t_{XOR2}$$

The cell area is

$$A_{linear} = (m-2)A_{OR2} + (m-1)A_{AND2} + (m-2)A_{XOR2}$$

For $m = 32$, $A_{linear} = 1588\mu m^2$ based on Artisan standard cell library. The linear delay is often too large to be acceptable. To speed up the computation, a tree structure with the delay of $O(\log m)$ is developed with increased area. The tree structure is similar to the carry-lookahead and prefix structures in fast adder design. A 9-bit example of the tree structure is shown in Figure 3.4. For this tree structure, the delay is reduced to

$$t_{tree} = \log(m-1)t_{OR2} + t_{AND2} + t_{XOR2}$$

while the area is increased to

$$A_{tree} = \frac{m-1}{2}\log(m-1)A_{OR2} + (m-1)A_{AND2} + (m-2)A_{XOR2}$$

For $m = 32$, $A_{tree} = 2206\mu m^2$, 39% larger than A_{linear}. Another disadvantage of this tree structure is that the fanout of some signals (except the sign) is as large as $m/2$. The fanout can be restricted using the same techniques as in prefix-adder design [71]. Note that the AND/OR gates in these structures are replaced with NAND/NORs in actual implementations.

The computation core of 2C-SM-2C is an $m \times n$-bit unsigned multiplier, which has the same area complexity as an $m \times n$-bit 2C multiplier. However, the power consumption of this $m \times n$-bit unsigned multiplier in 2C-SM-2C is similar to a regular $(m-1) \times (n-1)$-bit unsigned multiplier. Except the number 2^{m-1} converting from $X = (-2^{m-1})$, all other unsigned data are representable in $(m-1)$ bits. This is also true for operand Y. Therefore, the $m \times n$-bit unsigned multiplier in 2C-SM-2C acts an $(m-1) \times (n-1)$-bit unsigned multiplier during most of the time. Only when $X = -2^{m-1}$ or $Y = -2^{n-1}$, the MSB logic switches affecting dynamic power.

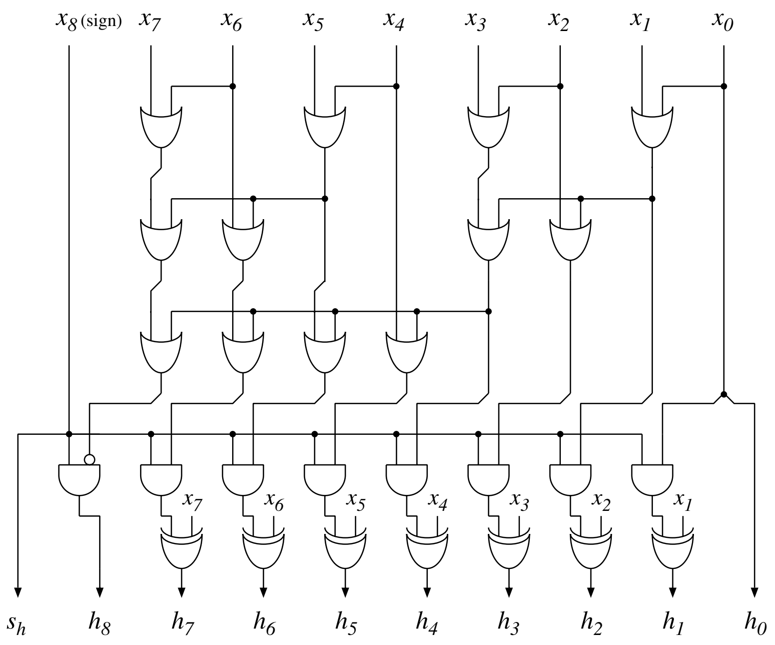


Figure 3.4: Tree structure of 2C-SM conversion ($m = 9$).

3.2.2 2C-P1-2C: Conversion to SM with Postponed "+1"

The delay of the conversion can be further reduced to be roughly a constant number. The conversion from 2C to SM involves a time-consuming carry-propagation step for "+1" when the number is negative. To avoid carry propagation, this "+1" step in 2C-SM conversion can be postponed and integrated into the PP reduction process. This scheme is named 2C-P1-2C. The SM-2C conversion logic is less delay sensitive because the conversion and the CPA have the same computation direction.

The 2C-P1-2C scheme is depicted in Figure 3.5. The 2C-SM conversions are replaced by two XOR arrays, which implement

$$h_j = x_j \oplus x_{m-1} \qquad \text{for } j = 0, \cdots, m-2 \tag{3.17}$$

$$k_i = y_i \oplus y_{n-1} \qquad \text{for } i = 0, \cdots, n-2 \tag{3.18}$$

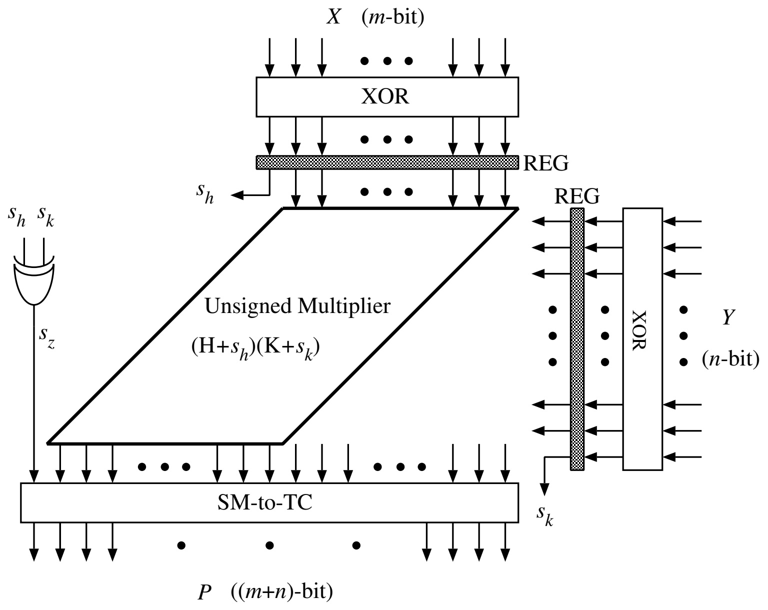


Figure 3.5: 2C-P1-2C structure.

The step “$+s_h$” and “$+s_k$” are postponed to the unsigned multiplication part as

$$H = \sum_{j=0}^{m-2} h_j 2^j \tag{3.19}$$

$$K = \sum_{i=0}^{n-2} k_i 2^i \tag{3.20}$$

$$|Z| = (H + s_h) \cdot (K + s_k) \tag{3.21}$$

This unsigned multiplication core has n $(m-1)$-bit PPs, one $(n-1)$-bit PP, and 1-bit $s_h \cdot s_k$. Thus, the total number of bits in the PP array is mn, the same as in a regular $m \times n$-bit 2C multiplication core (constant 1’s are not counted). But there is one more PP row. An example of the PP bit array for an 8×6-bit 2C multiplier is shown in Figure 3.6.

For the multiplication core in 2C-P1-2C, the power consumption would be similar to an $m \times n$-bit unsigned multiplier because they have the same number of PP bits. In Section 3.2.1, we analyzed that the power consumption of the $m \times n$-bit unsigned multiplier in 2C-SM-2C is similar to a regular $(m-1) \times (n-1)$-bit

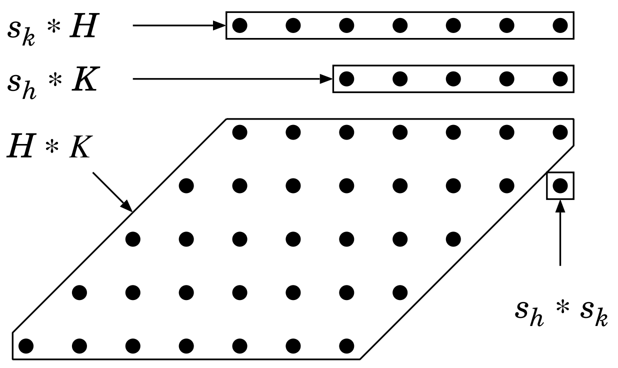


Figure 3.6: PP bit array in 2C-P1-2C ($m = 8, n = 6$).

unsigned multiplier. Therefore, the power consumption of the unsigned multiplier in 2C-P1-2C is larger than that of the unsigned multiplier in 2C-SM-2C.

3.2.3 2C-P1-CS: Conversion to 2C Using Carry-Save Addition

In above 2C-SM-2C and 2C-P1-2C schemes, two unsigned carry save vectors from PPR are added by CPA and the result in SM representation is finally converted to 2C representation. When a slow CPA is used for final addition in the multiplier, the effect of final SM-2C conversion on the overall delay is not obvious because the carries in the CPA and the SM-2C conversion go in the same direction. When a fast CPA is used and outputs are available simultaneously, however, the delay of SM-2C conversion becomes evident. Assume the number of latest signals arriving simultaneously from CPA is W_{max} and these signals are converted using a tree structure similar to Figure 3.4. The delay of final conversion is

$$T_{fconv} = T_{CPA} + log(W_{max} - 1)t_{OR2} + t_{AND2} + t_{XOR2} \tag{3.22}$$

To speed up the final conversion, we first combine the sign bit and two unsigned vectors to get two SM numbers. These two SM numbers are converted to two 2C numbers using carry-save addition and then a CPA is used to add the two 2C numbers. This scheme is named 2C-P1-CS.

After PPR, suppose the two vectors (ZS, ZC) in SM are

$$\begin{array}{cccccc} s_z & ZS_{w-2} & ZS_{w-3} & \cdots & ZS_2 & ZS_1 & ZS_0 \\ s_z & ZC_{w-2} & ZC_{w-3} & \cdots & ZC_2 & ZC_1 & ZC_0 \end{array}$$

where w is the vector length and $w = m + n$. For negative numbers, the rule of bit-complement and '$+s_z$' is applied. Consequently, the conversion of two SM vectors to two 2C vectors is to add the following vectors

$$\begin{array}{cccccc} ZS_{w-2} \oplus s_z & ZS_{w-3} \oplus s_z & \cdots & ZS_2 \oplus s_z & ZS_1 \oplus s_z & ZS_0 \oplus s_z \\ ZC_{w-2} \oplus s_z & ZC_{w-3} \oplus s_z & \cdots & ZC_2 \oplus s_z & ZC_1 \oplus s_z & ZC_0 \oplus s_z \\ 0 & 0 & \cdots & 0 & s_z & 0 \end{array}$$

The MSB sign bit s_z is known in advance and omitted. The $(s_z + s_z)$ on LSB is pre-computed to be $2s_z$. To avoid carry propagation delay, carry-save adders are used to produces 2C vectors (PS, PC). The shared signal s_z in each column is helpful in simplifying logic. For each carry-save adder on bit i $(i \neq 1)$, the following logic equations are deduced

$$PS_i = ZS_i \oplus ZC_i \tag{3.23}$$

$$PC_{i+1} = ZS_i ZC_i s_z' + ZS_i' ZC_i' s_z \tag{3.24}$$

For $i = 1$, the equations are

$$PS_1 = ZS_1 \oplus ZC_1 \oplus s_z \tag{3.25}$$

$$PC_2 = (ZS_1 ZC_1) \oplus s_z \tag{3.26}$$

When ZC_i is zero in some cases such as the right portion of ZC in radix-2 linear array multipliers, these equations are further simplified.

The delay of final conversion is reduced from Equation 3.22 to

$$T_{fconv} = T_{CPA} + MAX(t_{AND3} + t_{OR2}, t_{XOR2}) \tag{3.27}$$

Signal s_z is not on the critical path since it is precomputed and becomes available in advance even after buffer delays. The delay overhead of final conversion is the delay of one complex gate. However, the delay reduction comes at the cost of power increase. Because the two inputs of final CPA are now in 2C representation, the inherent power disadvantage of 2C for short-precision negative data appears in the CPA.

3.3 Representation Conversion for Radix-4 2C Multipliers

For radix-2 2C multipliers, representation conversion schemes are efficient in power saving for short-precision data because leading bits of both operands in the unsigned multiplier core often remains at 0's and thus have very low switching frequencies. For radix-4 multipliers, recoding itself has some good power-saving features. The area is smaller because the number of PPs is reduced. In addition, recoding changes a series of 1's in Y into 0's, which lead to zero PPs. However, recoding also generates negative PPs even if Y is an unsigned number. If 2C is used to represent PPs, leading bits would become 1's for negative numbers and switch frequently when negative numbers are mixed with positive numbers.

In [136], a mixed number representation was proposed for low-power radix-4 2C multiplication. In this scheme, the multiplicand is converted from 2C to SM while the multiplier is directly recoded into radix-4 $\{-2, -1, 0, 1, 2\}$ using Booth recoding. Instead of using 2C representation, PPs are now generated in SM representation. The motivation is that the PP generation in SM is much simpler. The negation of a SM number is simply the negation of one sign bit and, thus, it consumes much less power than the negation of a 2C number. Because SM numbers are not suitable for addition, the SM PPs are converted to RB $\{-1, 0, 1\}$ and a tree of redundant binary adders (RBAs) is then used for PP reduction. The conversion from SM to RB representation is done by grouping sign bits with each magnitude bit, which only require some wiring. The authors claimed significant improvement over previous tree multipliers by analytical estimation. However, there is a problem in their work. Although the power consumption in the PP generation circuit is reduced, the number of bit-vectors (not digit-vectors) for PP reduction goes back to n while this number is $n/2$ in the traditional radix-4 $m \times n$-bit multipliers. Consequently, the PP reduction tree would use one extra level of RBAs and the total number of RBAs is $(n/2 - 1)$, in contrast to $(n/4 - 1)$ in the regular RBA tree multipliers [60][61][85]. RBA tree multipliers have the similar complexity with [4:2]-CSA tree multipliers [93][89][55][98][45]. In a 32×32-bit radix-4 tree multiplier based on [4:2]-CSAs, our experiment shows that the PP generation logic accounts for about 10% power consumption while the four CSAs on the first level of PP reduction consumes 30% of total power. Hence, the power consumption in the extra level of RBAs would definitely exceed the power saving in the PP generation due to SM representation. This mixed number representation could not save power.

When the representation conversion schemes proposed here are applied to radix-4 2C multipliers, the power saving due to representation conversion of operand Y is minimal because the recoding produces a similar effect for negative signal bits. The main power saving comes from the conversion of operand X if X also has a large dynamic range of signed integers.

3.4 Experimental Evaluation

To evaluate representation conversion schemes, we have implemented 2C-SM-2C, 2C-P1-2C and 2C-P1-CS for both radix-2 [3:2]-CSA linear array multipliers and radix-2 [4:2]-CSA tree multipliers. The design and experimental methodologies are described in Appendix B. To reduce the delay of 2C-SM conversion, logic tree structures are used in 2C-SM-2C schemes. We assume that input data are stored in two registers and conversion logic is inserted before input registers. Both *djpeg* test data and *random* data are used. The power comparison results are shown in Table 3.2 and Table 3.3. The values in parentheses are normalized values. The smallest value of each characteristic among three conversion schemes is highlighted in boldface. The power in linear array multipliers is measured at $50MHz$ and the power in fast tree multipliers is measured at $100MHz$. The area/delay comparison results are shown in Table 3.4. *CArea* is the total cell area. The routable rate for array multipliers is 85% and the rate for tree multipliers is 70%. Note that power, delay, and area of input registers are included in these results. The power saving under short-precision data *djpeg* is quite impressive. For radix-2 array multipliers, the representation conversion schemes consume only $12 \sim 17\%$ of the baseline r2-array-reg. For tree multipliers, the conversion schemes consume $23 \sim 33\%$ of r2-tree4to2. A good feature of these conversion schemes is that they consume at most 14% more power than non-conversion schemes under *random* data. For 2C-SM-2C, the power consumption is even slightly less than that in the non-conversion schemes. This is because SM representation could also save power for random data although the saving is not significant. Compared to 2C-SM-2C, 2C-P1-2C schemes consume $18 \sim 28\%$ more power under *djpeg* and $9 \sim 13\%$ more power under *random*, which is consistent with our analysis in Section 3.2.2. Compared to 2C-P1-2C, 2C-P1-CS schemes consume 15 $\sim$ 26% more power under *djpeg* and 3 $\sim$ 4% more power under *random*. The power overhead in 2C-P1-CS comes from the CPA with 2C inputs, as analyzed in Section 3.2.3.

Table 3.2: Power comparison of representation conversion array schemes

Schemes	Power(mW) (50MHz)	
	Random	Djpeg
r2-array-reg	58.07 (1.00)	21.56 (1.00)
2C-SM-2C-array	**56.82 (0.98)**	**2.49 (0.12)**
2C-P1-2C-array	64.05 (1.10)	3.19 (0.15)
2C-P1-CS-array	66.02 (1.14)	3.66 (0.17)

Table 3.3: Power comparison of representation conversion tree schemes

Schemes	Power(mW) (100MHz)	
	Random	Djpeg
r2-tree4to2-reg	76.81 (1.00)	25.87 (1.00)
2C-SM-2C-tree	**76.52 (1.00)**	**5.99 (0.23)**
2C-P1-2C-tree	82.62 (1.08)	7.22 (0.28)
2C-P1-CS-tree	84.01 (1.09)	8.62 (0.33)

The representation conversion schemes have 5 $\sim$ 9% more area than non-conversion radix-2 schemes. 2C-P1-2C schemes have slight less area than 2C-SM-2C because two tree structures for 2C-SM conversion is replaced by two rows of XOR gates. As for delay, we consider two parts: T_{conv}, the delay of 2C-SM or 2C-P1 conversion before input registers and T_{core}, the delay of the computation core with registers. T_{conv} is reduced from 2.47 $\sim$ 2.49ns to 1.18 $\sim$ 1.37ns when

Table 3.4: Area/delay comparison of representation conversion schemes

Schemes	CArea(μm^2)	Delay(ns)		
		T_{conv}	T_{core}	Total
r2-array-reg	98325 (1.00)	0	16.49	16.49 (1.00)
2C-SM-2C-array	104792 (1.07)	2.47	17.65	20.12 (1.22)
2C-P1-2C-array	**104655 (1.06)**	1.18	18.08	19.26 (1.17)
2C-P1-CS-array	106698 (1.09)	1.28	17.40	**18.68 (1.13)**
r2-tree4to2-reg	104705 (1.00)	0	9.15	9.15 (1.00)
2C-SM-2C-tree	110949 (1.06)	2.49	10.15	12.64 (1.38)
2C-P1-2C-tree	**110097 (1.05)**	1.37	10.45	11.82 (1.29)
2C-P1-CS-tree	111717 (1.07)	1.29	9.94	**11.23 (1.23)**

the gate tree is changed to a row of parallel XORs. These parallel XORs have a common input – the sign signal, which contributes much delay due to driving buffers. For the same conversion technique, T_{conv} is the same at the logic level. The variation in measured T_{conv}'s comes from the variation in automatic layouts. T_{core} in 2C-P1-2C is slightly larger than T_{core} in 2C-SM-2C because 2C-P1-2C has one more PP for reduction. The total delay, $(T_{conv} + T_{core})$, of 2C-SM-2C-array is 22% worse than that of a normal radix-2 array multiplier. For tree multipliers, it is 38% worse. When 2C-P1-2C is applied, T_{core} is reduced by $4 \sim 7\%$ from 2C-SM-2C. When 2SM-2C conversion is applied, the total delay is further reduced by $0.6ns$, or 3% in the array multiplier and 10% in the tree multiplier. Compared to the baseline multipliers, the delay in 2C-P1-CS-array is 13% worse and the delay in 2C-P1-CS-tree is 23% worse. Because tree multipliers are much faster than array multipliers, the delay overhead in tree multiplier is more evident. In terms of power-delay and power-area products, all conversion schemes are still much better for short-precision test data.

3.5 Summary

In this chapter, operand representation conversion schemes for two's-complement multipliers are proposed to utilize the low-power feature of sign-magnitude representation. 2C-SM-2C is a straightforward conversion scheme which converts SM numbers directly from 2C numbers and convert the SM result back to 2C using the costly "+1" operation. 2C-P1-2C reduces the 2C-SM delay by postponing this "+1" step and integrating it into the PP reduction process. 2C-P1-CS further reduces the SM-2C delay by converting two SM vectors from PPR directly to two 2C vectors before the final CPA. For a typical data set from application *djpeg*, all conversion schemes reduce power consumption by more than 60% compared to the baseline two's-complement schemes. The overheads are $5 \sim 9\%$ more area and $13 \sim 44\%$ more delay.

CHAPTER 4

Optimization of Reduction Structure for Array Multipliers

4.1 Introduction

To reduce PP bit arrays into two carry-save vectors, several arithmetic structures can be used. The linear array structure is one of the most popular structures due to its simple interconnect and regular layout. As the interconnect effects become dominant in deep sub-micron designs, architectures with regular and local interconnects are more favorable [64]. However, array multipliers are generally not as fast as tree multipliers. Moreover, array multipliers have an architectural disadvantage in terms of power consumption [80][107]. A large number of glitches or spurious transitions are generated because PP bits arrive at the same time but are added serially and the input-to-output paths in adder cells have different delays. Glitches cause a snow-balling effect as signals propagate through the array. Table 4.1 presents the power distribution in a 32×32-bit radix-2 two's-complement linear array multiplier (without final CPA) under *random* test data. Each cell(i, j) on the grid consists of an AND/NAND gate and a full adder. Only powers in cells with $(i + 1)$ and $(j + 1)$ being multiples of 4 are shown for clarity. Cell power consumption increases along the highlighted (3,31)-(31,3) diagonal and the contours also move along this line. On each column j, the power consumption in cell(i, j) increases linearly with i before i reaches the diagonal position but does not change much after i crosses the diagonal. The abnormally high power consumptions in cell$(31, i)$ are due to inverted PP bits on this row that have more probabilities of being '1' than being '0'.

Table 4.1: Power distribution in 2C linear array multiplier under *random* data

(i, j)	31	27	23	19	15	11	7	3
3	**1.17**	12.00	11.30	11.20	11.50	12.30	11.80	12.00
7	1.07	**21.60**	26.70	26.30	26.40	26.90	26.40	26.30
11	1.05	22.30	**36.90**	40.40	40.50	41.60	41.50	40.90
15	0.93	22.90	37.80	**52.10**	56.80	55.20	56.00	55.80
19	1.11	23.00	36.90	52.50	**67.80**	71.10	69.40	70.30
23	1.02	22.40	37.20	52.90	68.70	**84.00**	86.10	84.20
27	1.07	22.10	37.20	53.10	68.30	83.60	**95.80**	101.00
31	1.43	26.90	43.60	61.30	81.70	99.80	118.00	**137.00**

Various approaches have been proposed to reduce the power consumption of array multipliers. At the circuit level, logic styles other than conventional CMOS logic are studied, such as complementary pass-transistor logic [2], self-timed double pass-gate logic [72], and clocked CMOS [16]. At the architecture level, the imbalance of signal delays is reduced by inserting ancillary logic such as latches and buffers [80][107][47]. More aggressive architecture-level optimization is to change the internal structure of array multipliers, especially the PPR structure. In [84], a *leapfrog* structure is proposed to take advantage of the existing delay imbalances in full adders, which is an improved structure of the *even/odd* split array scheme [65][100]. In a *leapfrog* scheme, carry signals still go directly to the next row in the PPR array. Only sum signals from the even (odd) row jump one row and feed into the next even (odd) row.

Recently, Left-to-Right (LR) or most-significant-digit first array multipliers have been developed and implemented. LR linear array multiplication provides an interesting alternative to the conventional right-to-left (RL) array multiplication as LR computation has the potential of saving power and delay. There are two types of LR schemes: LR with final carry-propagate addition (CPA) and LR without final CPA. By using on-the-fly conversion (OTFC) [40], the LR scheme [41] was proposed to eliminate the final CPA step in conventional RL linear array multipliers. It was further discovered that glitches in LR computation are fewer in the conventional RL computation, especially for data with a large dynamic range. In [90], LR array multipliers and hybrid structures with combined LR and RL types are studied in a generalized cellular template for different data characteristics. The switching activities are evaluated and compared with activities in a tree multiplier by assuming an ideal situation: the multipliers are well balanced in delay and zero-delay model could be used. In [133], the power consumption in the LR PPR array for Booth recoded multiplication is studied in detail for DSP applications. In [54], a low-power LR array multiplier without final CPA is designed using strategically placed (3,2), (5,3) and (7,4) counters and the modified on-the-fly converter [36]. In [103], an asynchronous array multiplier with split RL upper array and LR lower array is proposed to make the computation time faster with relatively lower power consumption.

This chapter considers how to further optimize the structure for low power LR array multipliers. Our goal is to reduce the power consumption without significant increase in the complexities of modules and interconnects. The following structure optimization techniques are considered: signal flow optimization in [3:2]-CSA linear PPR, carry-ripple adders (CRAs) for PPR, [4:2]-CSAs for PPR, even/odd split structure, and upper/lower split structure. When exploring these PPR optimization techniques, we only consider LR array multipliers with the same final CPA. We will consider OTFC and delay optimized final CPA in next

Chapter on high-performance low-power multipliers. For simplicity, we consider $n \times n$-bit TC multipliers and n is an even number.

4.2 Left-to-Right Array Multipliers

In conventional RL linear array multipliers, the PPs are added in series starting from y_0X (z_0X in radix-4), as shown in Figure 4.1a (4.2a). The reduction is usually performed using [3:2]-CSAs in which carries pass to the next row. A radix-2 8×8 RL multiplier is depicted in Figure 4.3. The black dots correspond to the bit matrix in Figure 4.1a, obtained with NAND2 or AND2 gates. Each '+' symbol is a full adder (FA) if there are three inputs, or a half adder (HA) if there are only two inputs. The '1' in the first row in Figure 4.1a is added as a carry-in of the final CPA. The numbers associated with wires are signal arrival times assuming a unit delay model as explained later. The final addition is a fast n-bit CPA. The total cell area of a radix-2 RL multiplier is estimated as

$$A_{RL-r2} = A_{PPG} + A_{PPR} + A_{CPA(n)} \tag{4.1}$$

$$A_{PPG} = (n^2 - 2n + 2)A_{AND2} + (2n-2)A_{NAND2} \tag{4.2}$$

$$A_{PPR} = (n^2 - 3n + 2)A_{FA} + (n-1)A_{HA} \tag{4.3}$$

Buffers are not included. Using Artisan standard cell library [10], $A_{NAND2} = 10\mu m^2$, $A_{AND2} = 13\mu m^2$, $A_{HA} = 39.5\mu m^2$, $A_{FA} = 79.5\mu m^2$. For $n = 32$, $A_{PPG} = 13126$ and $A_{PPR} = 75160$. A_{PPG} remains unchanged for all schemes using the same radix.

In LR linear array multipliers, the PPs are added serially from $y_{n-1}X$ or $z_{n/2-1}X$, as shown in Figure 4.1b and 4.2b. PP reduction using CSAs is still a popular choice here. For radix-2, a 12×12 LR multiplier architecture using CSAs is illustrated in Figure 4.4. The final $(n-1)$-bit CPA generates the most-

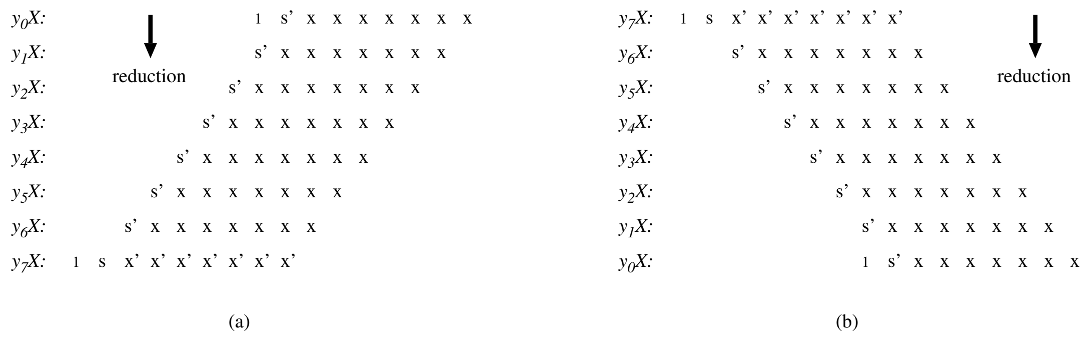


Figure 4.1: Radix-2 PP bit matrix (n=8): (a) RL; (b) LR.

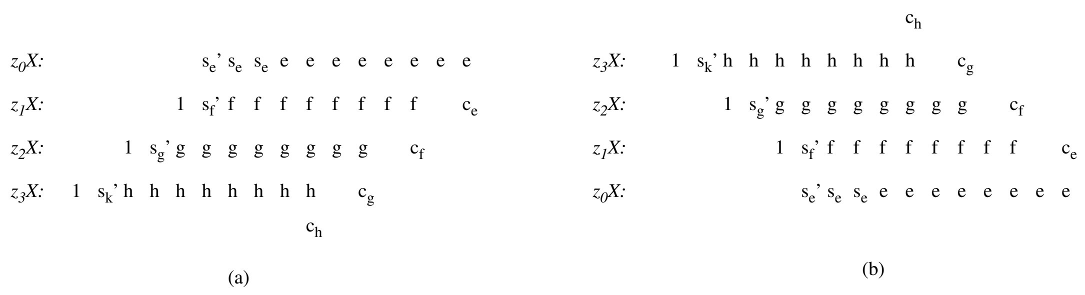


Figure 4.2: Radix-4 PP bit matrix (n=8): (a) RL; (b) LR.

significant half of the product. The shaded cells in the last row comprise a CRA which generates the least-significant half of the product and a carry-in of the final CPA. As the arrival times of these carry/sum bits match the computation direction and speed of the CRA, there is little delay penalty due to the use of CRA. The shaded cells on the left are used to add three bits each column from the reduction array into two bits. These shaded cells are extra hardware unique to LR multipliers. However, these extra cells do not increase the overall area because the number of cells in the PPR core is reduced. The total cell area is estimated as

$$A_{LR-r2} = A_{PPG} + A_{PPR} + A_{extra} + A_{CPA(n-1)} \tag{4.4}$$

$$A_{PPG} = (n^2 - 2n + 2)A_{AND2} + (2n - 2)A_{NAND2} \tag{4.5}$$

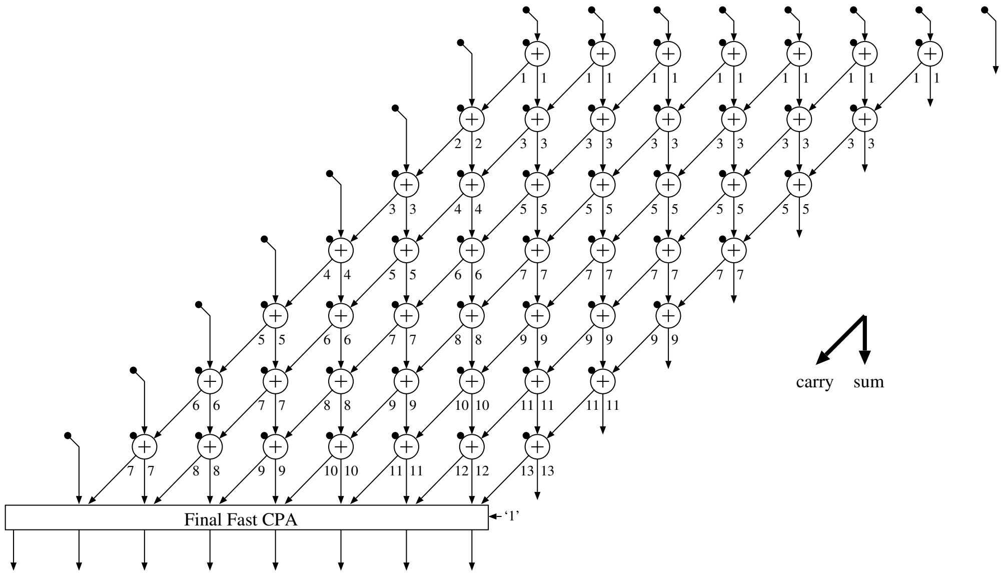


Figure 4.3: Radix-2 RL carry-save array multiplier (n=8).

$$A_{PPR} = (n^2 - 5n + 7)A_{FA} + (n-2)A_{HA} \tag{4.6}$$

$$A_{extra} = (2n-5)A_{FA} + A_{HA} \tag{4.7}$$

A_{PPG} is the same as in Equation 4.2. $A_{PPR} + A_{extra} = (n^2 - 3n + 2)A_{FA} + (n - 1)A_{HA}$ is the same as in Equation 4.3. Thus, A_{LR-r2} is very close to A_{RL-r2}. The carry signals propagate fewer stages in RL schemes than in RL schemes, which may reduce the power consumption in the left region. For data with a large dynamic range, PPs corresponding to sign extension bits are located in the upper region of an LR array. If sign extension bits switch less frequently than other bits as in many multimedia data [75], glitches can be reduced because the upper portion is not polluted by frequent switches in the lower portion in LR arrangement.

In radix-4 LR multipliers, the CRA is no longer suitable to add the right half carry/sum vectors from the reduction array because the vector bits arrive faster than the CRA computation. To avoid becoming the critical path, CRA should

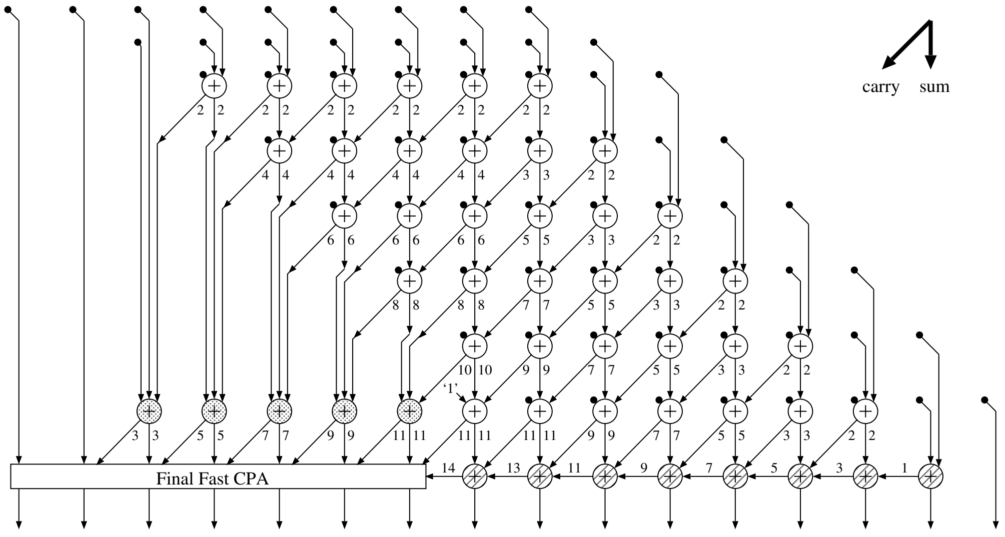


Figure 4.4: Radix-2 LR carry-save array multiplier (n=8).

be replaced by a fast CPA. For vectors from the left part of the reduction array, CSAs are still needed because about half columns have three bits. The total area of a radix-4 LR multiplier is also very close to that of a radix-4 RL multiplier. A 12×12 multiplication example is shown in Figure 4.5.

4.3 Structure Optimization

The use of CSAs for PP reduction in LR array multiplication follows the tradition in RL multiplication. It is unknown if it is still a good choice for LR computation or if there are other better candidates from the perspective of low power. Detailed studies are desirable in order to explore the potential advantages of LR multiplication. In this section, we present several structure optimization techniques for low power LR array multipliers. Some of these techniques have been also used in RL array multipliers and they are investigated here to see how they perform in LR multipliers. For theoretical analysis, the delay of a 2-input XOR2 gate, T_{XOR2}, is used as the base unit delay. MUX21 has the same complexity as

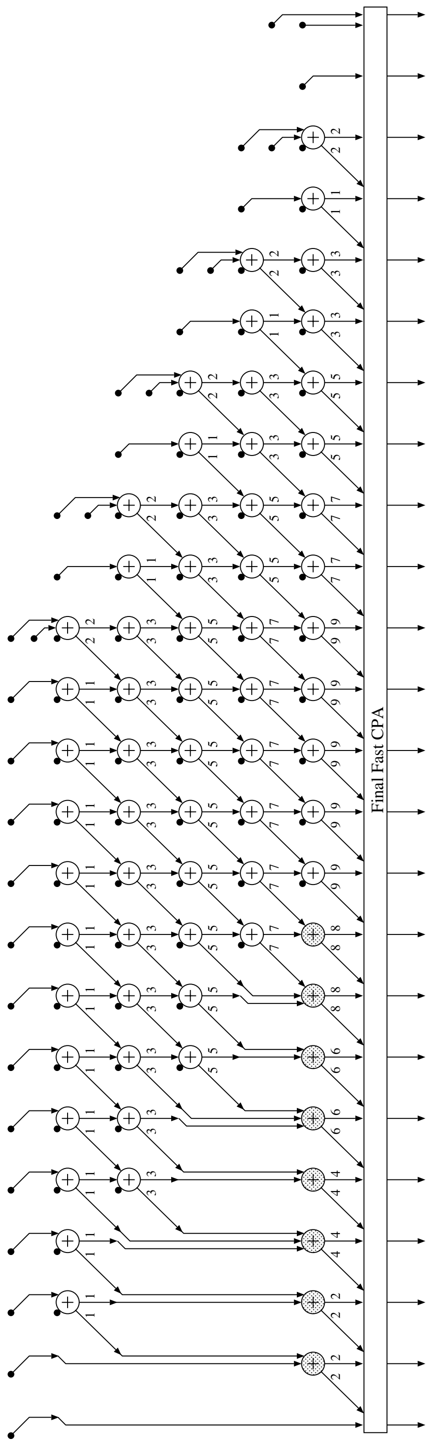


Figure 4.5: Radix-4 LR carry-save array multiplier (n=12).

XOR2. We assume that positive logic such as AND/OR could be optimized to be negative logic NAND/NOR in actual implementation and hence no difference is made between positive logic and negative logic. The delay of one simple gate such as NOR2 is $0.5T_{XOR2}$. The delay of two-level complex gate such as AOI22 (AND2-NOR2) is equivalent to T_{XOR2}.

4.3.1 Signal Flow Optimization in [3:2]-CSA Based Array

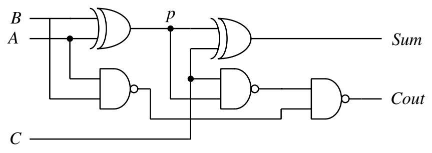
We first focus on the popular [3:2]-CSA based linear PPR structure. One reason of large power consumption in linear array multiplication is unbalanced signal arrivals in the adders. In order to analyze the imbalances, the behavior of the fundamental FA in multipliers is studied here. Figure 4.6 shows two popular FA structures. Figure 4.6a is a NAND2-based structure, FA-NAND2. Figure 4.6b is a MUX-based structure, FA-MUX. The worst-case delays from inputs to outputs are similar in two structure. For area, FA-MUX is superior because a MUX is smaller and simpler than three separate NAND2s in full-custom designs and many standard-cell based designs. We will focus on FA-MUX here. The worst-case delay from A and B to $Sum/Cout$ is always $2T_{XOR2}$. The delay from C to $Sum/Cout$ is T_{XOR2}. The availability of $Sum/Cout$ depends on the relative arrival times of three inputs. We use τ_i to represent the arrival time of signal i. Denote $\tau_{A,B}$ as $max(\tau_A, \tau_B)$. τ_{Sum} and τ_{Cout} in FA-MUX are expressed as:

$$\tau_{Sum} = \tau_{Cout} = max(\tau_{A,B} + T_{XOR2}, \tau_C) + T_{XOR2} \tag{4.8}$$

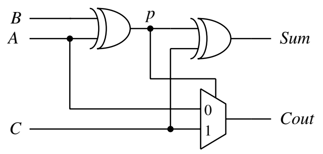
Specifically,

$$\tau_{Sum} = \tau_{Cout} = \begin{cases} \tau_{A,B} + 2T_{XOR2} & \text{if } \tau_C \leq \tau_{A,B} \\ \tau_C + T_{XOR2} & \text{if } \tau_C \geq \tau_{A,B} + T_{XOR2} \end{cases} \tag{4.9}$$

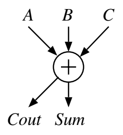
Different τ scenarios introduce different switching activities in FAs (the switching probability is denoted by α in Equation 1.2). To simplify analysis, we assume



(a) FA-NAND2



(b) FA-MUX



(c) Symbol

Figure 4.6: Two designs of full adder.

there is no change in signal A. This assumption is reasonable if A is connected to a PP bit that only switches at the very beginning. In terms of power consumption, the best scenario of signal arrivals is

$$\tau_C = \tau_B + T_{XOR2} \tag{4.10}$$

where there is no glitches on Sum or $Cout$, as shown in Figure 4.7a. Partial glitches due to inertial delay [130] ($T_{inertial}$) are generated if

$$\tau_B + T_{XOR2} - T_{inertial} \leq \tau_C \leq \tau_B + T_{XOR2} + T_{inertial} \tag{4.11}$$

and

$$\tau_C \neq \tau_B + T_{XOR2} \tag{4.12}$$

This situation is shown in Figure 4.7b. The shaded region is the region affected by delay inertia. The voltage swing in partial glitches is not rail-to-rail and cannot propagate through the next gate [96][104]. Full rail-to-rail glitches in Figure 4.7c are generated if

$$\tau_C < \tau_B + T_{XOR2} - T_{inertial} \tag{4.13}$$

or

$$\tau_C > \tau_B + T_{XOR2} + T_{inertial} \tag{4.14}$$

Full glitches consume more power than partial glitches because they are rail-to-rail transitions that could propagate through many gate stages if they are wide enough to propagate [82]. Therefore, it is still beneficial to make B and C arrive as close as possible and have a narrow pulse in the case of full glitches.

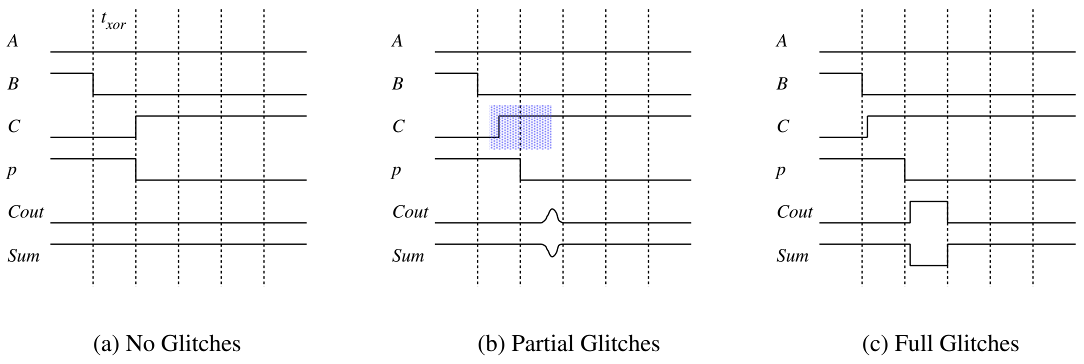


Figure 4.7: Scenarios of glitch generation in FA-MUX.

From the viewpoint of arithmetic computation, three input pins of an FA are logically equivalent and exchangeable. In the PPR part of linear array multipliers, each FA has three incoming signals Sin, Cin, and PP_{ij}. There are six possibilities of connecting those signals to three input pins. A usual flow is to connect Cin to C, Sin to either A or B. As A or B can be viewed as a sum pin and C as a carry pin, this signal flow is named as *SSCC*. The schemes in Figure 4.4, 4.3, and 4.5 use the classical SSCC flow and τ's have been marked as numbers associated with each FA output. To reduce power consumption, we propose to optimize the signal flow based on signal switching probabilities (α's) and arrival times (τ's). This optimization algorithm is named α-τ and shown in Figure 4.8. First, PP_{ij} is always connected to pin A instead of B. PP bits arrive at the earliest point which we define as time 0 and only switch once. Compared to pin A and B, the delay from pin C to $Sum/Cout$ is only one T_{XOR2} delay and the switching of pin C affects only two gates (XOR and MUX). Therefore, it is better not to use C for PP_{ij} so that C is connected to signals with more switching activities. Between A and B, pin A is chosen for PP_{ij} because B has less load capacitance and is also reserved for high switching signals. Second, Sin/Cin are connected to B/C according to signal τ's. If $\tau_{Sin} = \tau_{Cin}$, the one with higher switching activity is connected to pin C because pin C affects one less gate than B. Gate-level power estimation techniques [94][95] are used to calculate the transition probabilities of FA input signals in the multiplier. In linear array multipliers, we find that most Sin signals have higher switching activities than Cin signals. Therefore, most Sin signals are connected C when $\tau_{Sin} = \tau_{Cin}$. The resulting structure with Cin-to-B and Sin-to-C flows is named as *CSSC*.

```
Input: $\alpha$ and $\tau$ of $Sin$, $Cin$, $PP_{ij}$;
Output: optimized flow into input $A$, $B$, $C$ of FA;

Connect $PP_{ij}$ to pin $A$;
**if** $\tau_{Sin} < \tau_{Cin}$ **then**
    Connect $Sin$ to pin $B$ and $Cin$ to $C$;
**else if** $\tau_{Sin} > \tau_{Cin}$ **then**
    Connect $Sin$ to pin $C$ and $Cin$ to $B$;
**else**
    **if** $\alpha_{Sin} < \alpha_{Cin}$ **then**
        Connect $Sin$ to pin $B$ and $Cin$ to $C$;
    **else**
        Connect $Sin$ to pin $C$ and $Cin$ to $B$;
    **end if**
**end if**
```

Figure 4.8: α-τ signal flow optimization in [3:2]-CSA Linear PPR.

This module-level signal flow optimization is different from gate-level pin swapping [28] in two ways. First, gate-level pin swapping only considers equivalent pins of simple gates such as NAND3. It is not easy to detect the arithmetic equivalence of module input pins when multipliers are treated in the same way as random logic. Second, gate-level pin swapping only considers signal switching and pin capacitance. The signal arrival times and the delay property of FAs are not used. Figure 4.9 and Figure 4.10 are CSSC flow optimized structures for radix-2 and radix-4 LR array multipliers, respectively. In radix-2 structure, about 40% FAs in PPR now have glitch-free signal arrival scenarios described before. In radix-4 structure, 15% of FAs in PPR have better signal arrival scenarios. The effect of CSSC flow optimization in the radix-4 architecture is not as significant as in the radix-2 architecture because radix-4 PPs are less regular and each PP row is shifted two-bit positions. Most FAs with improved flows happen in the right portion of the array. In the left portion, there is no much change except those signals affected by right-portion signals.

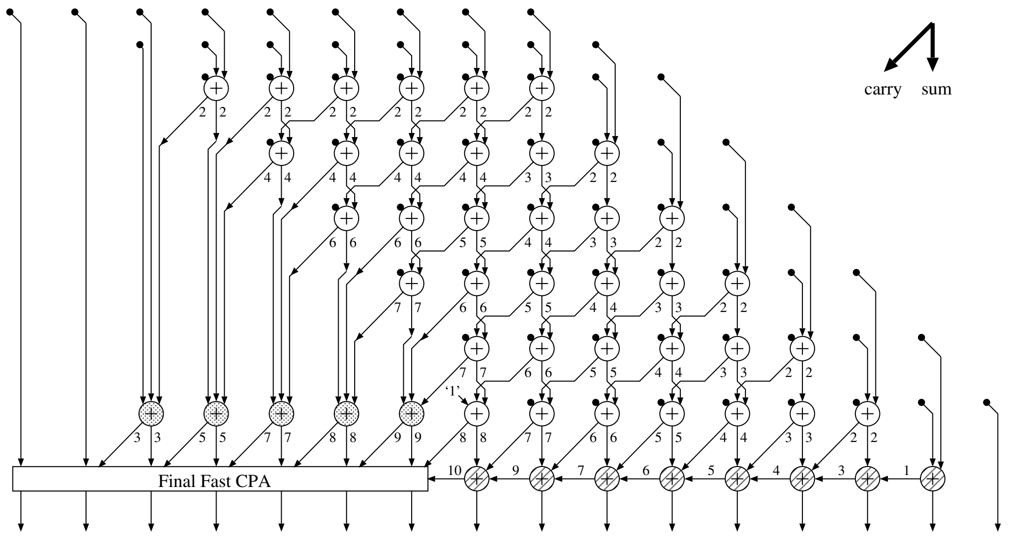


Figure 4.9: Radix-2 LR [3:2]-CSA array multiplier with CSSC flow (n=8).

With CSSC flow optimization, the delay in the PP reduction step is also reduced considerably, which is advantageous to power saving. Similar flow optimization has been applied in tree multipliers for high speed [99]. Along the n-th column from the right in radix-2 LR, the delay is reduced from $(2n-5)T_{XOR}$ to $(n-1)T_{XOR}$. For $n = 32$, the reduction is 47%. Along the n-th column in radix-4 LR, the delay is reduced from $(n-3)T_{XOR}$ to $\lfloor 2n/3 - 1 \rfloor T_{XOR}$, as shown

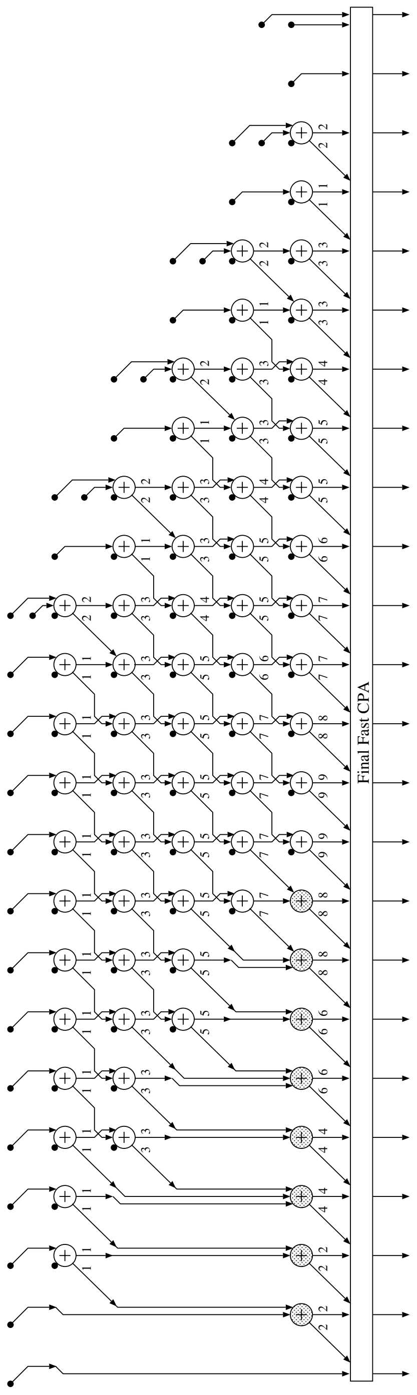


Figure 4.10: Radix-4 LR [3:2]-CSA array multiplier with CSSC flow (n=12).

in Table 4.2. For $n = 32$, the reduction is 30%.

Table 4.2: Delay of the n-th column in radix-4 LR with CSSC

n	12	14	16	18	20	22	24	26	28	30	32	$\cdots$
t	7	8	9	11	12	13	15	16	17	19	20	$\cdots$

4.3.2 CRAs for PP Reduction

In conventional RL array multiplication, CRAs are not used for PP reduction because the delay of the reduction would be increased from $(n-1)t_{FA}$ to $(2n-2)t_{FA}$ in radix-2 array. t_{FA} is the delay of a FA. As for power, the carry propagation in CRA introduces a large number of glitches when carry and sum inputs arrive at quite different time [80], which happens in RL linear array multipliers. In LR array multiplication, however, the use of CRA has little effect on the delay as the CRA computation direction matches the shape of PP bit matrix. Moreover, the number of glitches is reduced because the carry and sum inputs of most FAs arrive at similar periods in radix-2 LR array. A high-level recursive description of using CRAs for radix-2 PPR is

$$PS_i = CRA(PS_{i+1}, PP_i) \qquad i = n-2, \cdots, 0 \tag{4.15}$$

with $PS_{n-1} = PP_{n-1}$ and $P = PS_0$. A direct implementation of this recursion requires no final CPA. However, the length of CRAs increases step by step and excessive adders are used. A better way is to keep the length of CRAs unchanged and use a final CPA, as described in Figure 4.11. One example of using CRAs for LR PPR is shown in Figure 4.12. Different from CSA-based LR array multiplier in Figure 4.4, no extra CRA or CSA is need to add carry/sum vectors.

```
PP: Partial product bit matrix;
(FS, FC): input vectors of final CPA;
PS: sum vectors from CRAs;

$PS_{n-1}[n, \cdots, 0] = PP_{n-1}[n, \cdots, 0]$;
$FS[2n-1] = 0$;
for $i$ from $n-2$ downto 0 do
    $FC[n+i+1] = PS_{i+1}[n]$;
    $FS[n+i] = PS_{i+1}[n-1]$;
    $PS_i[n, \cdots, 0] = \mathbf{CRA}(2PS_{i+1}[n-2, \cdots, 0], PP_i[n-1, \cdots, 0])$;
end for
$FC[n] = PS_0[n]$;
$P[2n-1, \cdots, n] = \mathbf{CPA}(FS[2n-1, \cdots, n], FC[2n-1, \cdots, n], 1)$;
$P[n-1, \cdots, 0] = PS_0[n-1, \cdots, 0]$;
```

Figure 4.11: Radix-2 LR CRA based PPR algorithm.

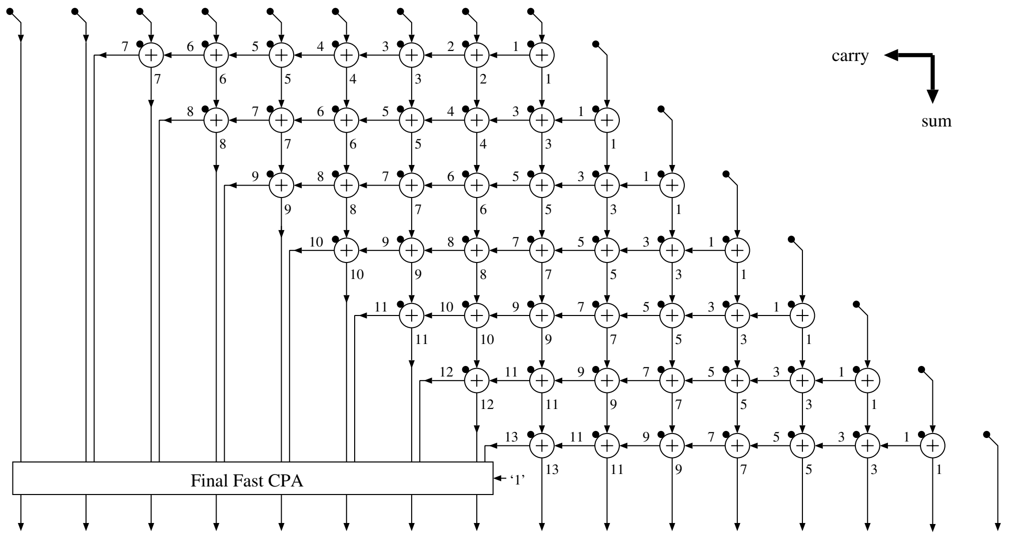


Figure 4.12: Radix-2 LR CRA based array multiplier (n=8).

The total cell area of a radix-2 CRA multiplier is:

$$A_{RL-r2-CRA} = A_{PPG} + A_{PPR} + A_{CPA(n)} \tag{4.16}$$

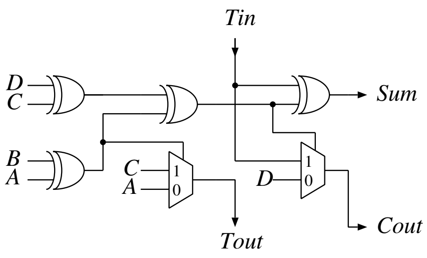
$$A_{PPG} = (n^2 - 2n + 2)A_{AND2} + (2n-2)A_{NAND2} \tag{4.17}$$

$$A_{PPR} = (n^2 - 3n + 2)A_{FA} + (n-1)A_{HA} \tag{4.18}$$

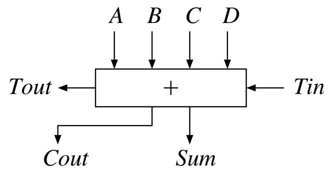
which is exactly the same as the area of a radix-2 RL multiplier described in Equation (4.1)-(4.3). Common in all linear array multipliers, PPs introduce initial glitches as they are available well ahead of other signals. For most FAs in LR CRA based PPR, however, the carry and sum inputs arrive at similar periods and do not introduce many glitches, as indicated by τ values in Figure 4.12. In the left array portion, the signal τ scenarios of most FAs are actually the best glitch-free scenarios. In the right portion, the τ differences between sum and carry inputs of FAs are smaller than those in the CSA-based multiplier in Figure 4.4. Therefore, we expect that CRA-based LR array multipliers consume less power than CSA-based multipliers.

4.3.3 [4:2]-CSAs for PP Reduction

Another departure from using [3:2]-CSAs is to use [4:2]-CSAs for PP reduction in array multipliers. [4:2]-CSAs have been widely used in tree multipliers. However, there is little previous work on applying [4:2]-CSAs in array multipliers. A [4:2]-CSA has the same gate complexity as two [3:2]-CSAs. But a [4:2]-CSA is faster than two cascaded [3:2]-CSAs because an efficient design could have $3T_{XOR2}$ delay while each single [3:2]-CSA has $2T_{XOR2}$ delay. One gate-level design of such a [4:2]-CSA is shown in Figure 4.13. When [4:2]-CSAs are applied to array multipliers, the first [4:2]-CSA row accepts four PPs and generates two carry-save vectors. Each subsequent [4:2]-CSA row accepts two previous carry-save vectors and two new PPs and generates two current carry-save vectors. A 16×16-bit LR array multiplier using [4:2]-CSAs is illustrated in Figure 4.14. dark dots are PP bits. the correct bits for radix-4 recoding are shown as small circles for distinction. Gray dots are carry/sum vectors from CSAs. An extra [3:2]-CSA is used because of the extra PP bits due to radix-4 recoding.

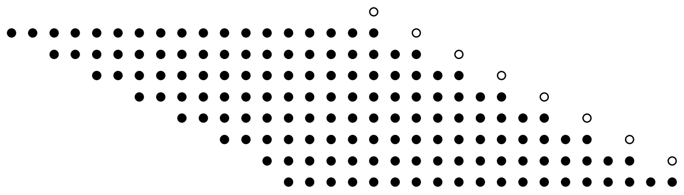


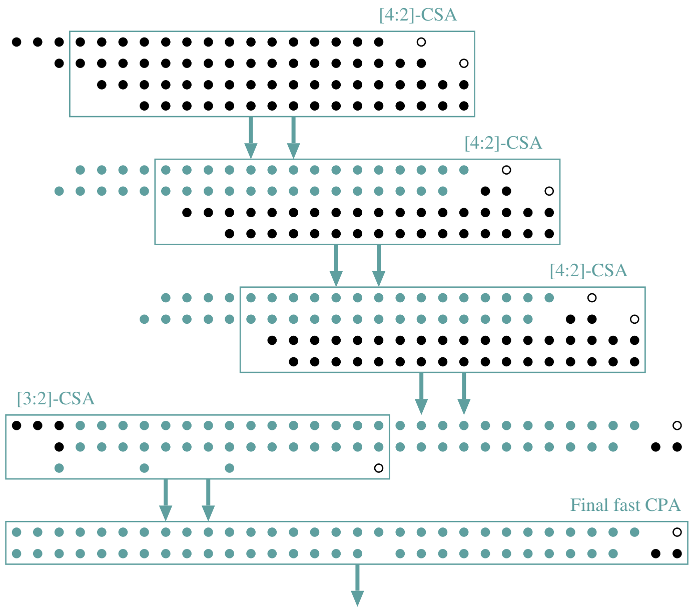
(a) MUX-based Design



(b) Symbol

Figure 4.13: 1-bit [4:2]-CSA.


(a) PP Bit Matrix



(b) Linear Array Reduction using [4:2]-CSAs

Figure 4.14: Radix-4 LR array multiplication based on [4:2]-CSAs (n=16).

For h n-bit PPs in general, the area and delay of PPR using [4:2]-CSA linear array are roughly estimated as

$$A_{PPR} = (\frac{h}{2} - 1)A_{[4:2]CSA(n)} = (h-2)A_{[3:2]CSA(n)} \tag{4.19}$$

and

$$t_{PPR} = (\frac{h}{2} - 1)t_{[4:2]CSA} = 3(\frac{h}{2} - 1)T_{XOR2} \tag{4.20}$$

In contrast, the area and delay of PPR using [3:2]-CSA linear array are

$$A_{PPR} = (h-2)A_{[3:2]CSA(n)} \tag{4.21}$$

and

$$t_{PPR} = (h-2)t_{[3:2]CSA} = 2(h-2)T_{XOR2} \tag{4.22}$$

By using [4:2]-CSAs, the PPR delay is reduced about 25% while the area has no change. The delay reduction is positive for power as less switching activities are generated when signals propagate fewer stages. Compared to two stacked [3:2]-CSAs, a [4:2]-CSA has more balanced structure: four inputs are not only equivalent in logic but also equal in the worst-case delay. The balanced structure is suitable in tree multipliers because the reduction tree itself is balanced. However, the balanced structure doesn't match the PPR linear array structure, which is a negative factor for power consumption. The CSSC flow optimization technique cannot be applied here. Considering the 25% PPR delay reduction, however, we believe that the overall power consumption in [4:2]-CSA linear array multipliers will still be reduced.

4.3.4 Split Array: Even/odd and Upper/lower

As shown in Table 4.1, glitches cause a snow-balling effect as signals propagate through an array multiplier. Therefore, the lower rows consume much more power than the upper rows in the PPR array. If the long path of PPR could be broken into parallel short paths, there would be a saving in power. One approach to reduce the PPR length is to split the array into two parts and each part only have a half number of rows. If each part is split further, a multi-level tree structure will be finally generated. In order to keep simple interconnects and structural regularity, we avoid further splitting.

To split the array, an *even/odd* scheme with even rows in one part and odd rows in another part has been proposed [65][59]. Each part is added separately in parallel. The final even/odd vectors from two parts are reduced to two vectors using a [4:2]-CSA. By interleaved placement and routing, the layout regularity of linear array multipliers is kept. The *even/odd* scheme was originally proposed for speed improvement. But it is obvious that even/odd is also efficient in power. Recently, an improved even/odd structure named *leapfrog* was proposed for RL array multipliers [84]. The existing delay imbalances in FAs are utilized by connecting sum signals from one even (odd) row to the next even (odd) row while connecting carry signals always to the next row. In addition, the final CSA to add even/odd vectors are simplified to be a [3:2]-CSA. Because all the carry signals propagate through the entire array in RL multipliers, the right portion of PPR final vectors arrive at the same time. This eliminates the possibility of final addition optimization existing in tree multipliers and LR array multipliers. Therefore, we combine LR computation and the leapfrog structure to develop LR leapfrog (LRLF) array multipliers. A high-level description of radix-4 LRLF scheme is given in Figure 4.15. The basic component is still a standard [3:2]-CSA. However, sum signals PS_i skip the next row while carries PC_i go directly to the next row.

A straightforward implementation of LRLF would result in excessive cells in

```
$h = n/2$: the number of PPs;
$(PS, PC)_i$: sum and carry vectors from [3:2]-CSAs;

$(PS, PC)_0 = \mathbf{CSA}(PP_h, PP_{h-1}, PP_{h-2})$;
$(PS, PC)_1 = \mathbf{CSA}(PP_{h-3}, PP_{h-4}, PC_0)$;
for $i$ from 2 to $h-3$ do
  $(PS, PC)_i = \mathbf{CSA}(PP_{h-3-i}, PS_{i-2}, PC_{i-1})$;
end for
$(PS, PC)_{h-2} = \mathbf{CSA}(PS_{h-3}, PC_{h-3}, PS_{h-4})$;
```

Figure 4.15: High-level radix-4 LRLF algorithm.

the bottom-left triangle portion of the array. To avoid excessive computation and get better signal arrival profile from PPR, PS/PC signals entering this portion are processed together by a $(n-3)$-bit [4:2]-CSA, as illustrated by the example in Figure 4.16. The small dots are PP bits. The dashed lines are carries and solid lines are sum signals. Different from RL *leapfrog*, an $(n-3)$-bit [4:2]-CSA is required to add the left-side vectors in LRLF.

Besides *even/odd* splitting, another possibility is to simply divide the array into upper and lower parts. The PPR array could still be kept regular by interleaving. Each part is a smaller LR PPR array with a left-side [3:2]-CSA for vector merging. The final upper/lower vectors are added by a [4:2]-CSA. For each part, CSSC flow optimization is applied. This scheme is called *LR-CSSC-U/L*. A high-level diagram of this scheme is shown in Figure 4.17.

4.4 Experimental Evaluation

We have implemented three radix-2 LR schemes and six radix-4 LR schemes with different structure optimization techniques in a 32×32-bit linear array multiplier framework. The design and simulation methodologies are described in

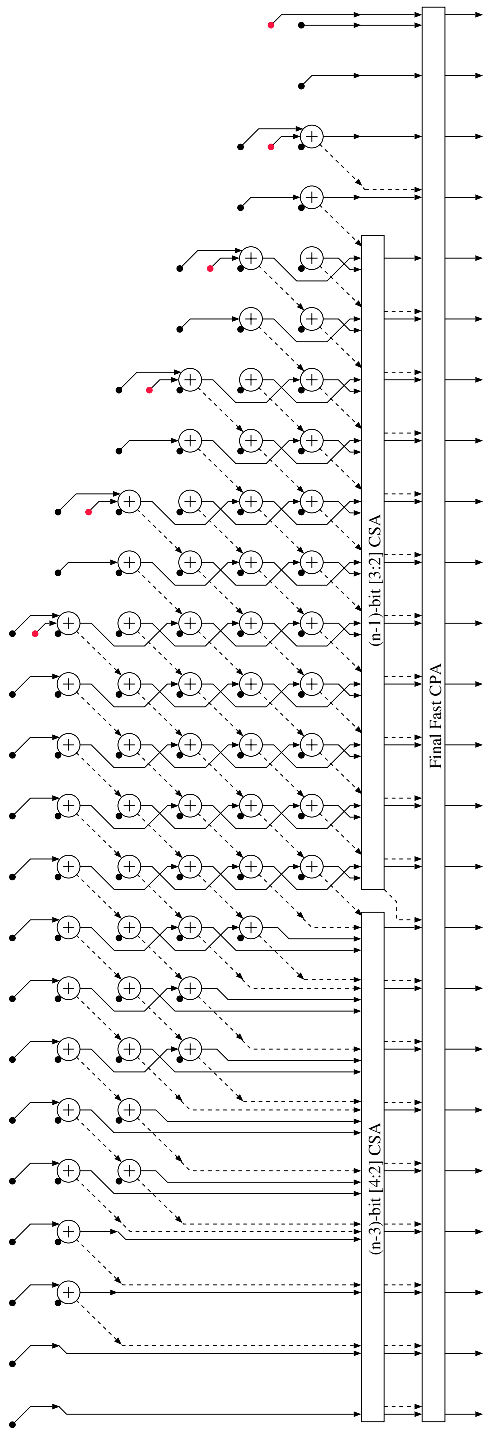


Figure 4.16: Radix-4 LRLF array multiplier (n=12).

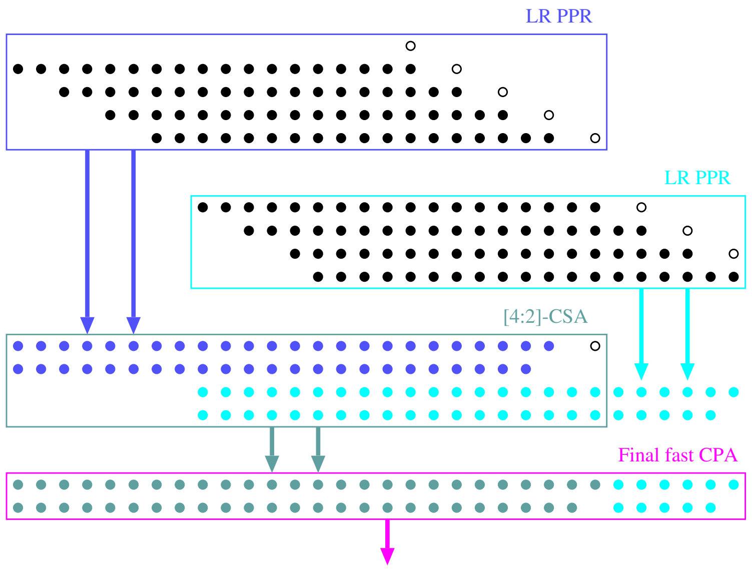


Figure 4.17: Radix-4 LR upper/lower array multiplier (n=16).

Appendix B. As our major focus is on the PP reduction step, the final CPAs in our designs are not optimized for different input arrival scenarios [99][131]. A two-level carry-lookahead adder structure is used in all designs. Fortunately, CPAs are the final modules in all multipliers and the results on PPG and PPR modules are not affected. Two test data sets, *random* and *djpeg*, are used in order to capture power features in different application environments.

4.4.1 Results for Radix-2 LR Multipliers

For radix-2 LR linear array multipliers, three schemes are implemented: LR using the default [3:2]-CSAs, LR using CRAs, and LR with CSSC flow optimization. Other structure optimization techniques are implemented only for radix-4 LR multipliers because radix-4 schemes are more widely used. The final CPA in all schemes is a two-level CLA. The comparison results of power consumption are shown in Table 4.3 and Table 4.4. The power consumption is measured at $50MHz$. The area results are shown in Table 4.5 and the delay results are in Table 4.6. The baseline structure is a RL [3:2]-CSA based array multiplier. The values in parentheses are normalized values. The smallest value of each characteristic is highlighted in boldface. *CArea* is the total cell area. The routable rate for basic LR, LR-CRA, and LR-CSSC is 80%, 5% less than the rate for the baseline RL array multiplier. Besides the total values, power/delay/area in each sub-module are also listed for comparison. These modules are: PPG and PPR (PPGR), extra CSA (eCSA) and extra CRA (eCRA) in some schemes, and final CPA in all schemes.

Table 4.3: Power in radix-2 LR multipliers under *random* data

Schemes	Power (mW) $(50MHz)$				
	PPGR	eCSA	eCRA	CPA	total
RL-baseline	52.29	0	0	4.66	56.95 (1.00)
LR	50.79	2.92	2.64	4.58	60.93 (1.07)
LR-CRA	37.94	0	0	3.97	41.91 (0.74)
LR-CSSC	31.74	1.92	1.63	3.33	**38.62 (0.68)**

Table 4.4: Power in radix-2 LR multipliers under *djpeg* data

Schemes	Power (mW) $(50MHz)$				
	PPGR	eCSA	eCRA	CPA	total
RL-baseline	19.33	0	0	1.74	21.07 (1.00)
LR	15.93	0.68	1.03	2.23	19.87 (0.94)
LR-CRA	12.23	0	0	1.53	13.76 (0.65)
LR-CSSC	9.44	0.43	0.65	1.54	**12.06 (0.57)**

Table 4.5: Cell area in radix-2 LR multipliers

Schemes	CArea (μm^2)				
	PPGR	eCSA	eCRA	CPA	total
RL-baseline	90215	0	0	3506	93721 (1.00)
LR	85100	2315	2475	3469	93359 (1.00)
LR-CRA	89181	0	0	3506	**92687 (0.99)**
LR-CSSC	85692	2315	2475	3469	93951 (1.00)

Table 4.6: Delay in radix-2 LR multipliers

Schemes	Delay(ns)				
	PPGR	eCSA	eCRA	CPA	total
RL-baseline	13.27	0	0	2.65	15.92 (1.00)
LR	12.94	0	0.72	1.89	15.55 (0.98)
LR-CRA	13.08	0	0	2.99	16.07 (1.01)
LR-CSSC	8.05	0.42	0	2.52	**10.99 (0.69)**

For the basic LR scheme, the LR computation reduces the power consumption in PPGR module by 3% under *random* and 18% under *djpeg*. Because of the extra power in eCSA and eCRA, the overall power in basic LR is increased by 7% under *random*. Under *djpeg*, the overall power is reduced by 6% despite of the extra power in eCSA and eCRA. This demonstrates that LR computation is superior in power for input data with a large dynamic range. The area and delay of LR are very close to those of RL, as we expected before. The LR-CRA scheme improves the power consumption by 26 $\sim$ 35% without any delay or area overhead. The most interesting results are from LR-CSSC which reduces the power by 32 $\sim$ 43% and the delay by 31% with no area increase. If input buffer and PPG delay (1.75ns) is excluded from PPGR delay, the net PPR delay is reduced by 45% (from 11.52ns in LR to 6.3ns in LR-CSSC), which well matches our theoretical analysis in Section 4.3.1.

The experiment results also show that the differences of power consumptions in the final CPAs are less than 5% of the total power consumptions. Therefore, these un-optimized final CPAs will not affect the relative behaviors of different schemes although they may increase some errors in the absolute power consumption values.

4.4.2 Results for Radix-4 LR Multipliers

For radix-4 LR linear array multipliers, six schemes are implemented: LR using the default [3:2]-CSAs, LR using CRAs, LR with CSSC flow optimization, LR using [4:2]-CSAs, LR *leapfrog* split structure, and LR upper/lower split structure with CSSC flow optimization. The radix-4 recoding design is parallel recoding NPR3b proposed in Chapter 2 because of its overall performance. Serial recoding NSR4 is not used because not all LR schemes here have enough PPR delay for serial recoding operations. The final CPA in all schemes is a two-level CLA. The power consumption are shown in Table 4.7 and Table 4.8. The area results are shown in Table 4.9 and the delay results are in Table 4.10. The baseline structure is a radix-4 RL [3:2]-CSA based array multiplier. The values in parentheses are normalized values. The smallest value of each characteristic is highlighted in boldface. The routable rate for basic LR, LR-CRA, and LR-CSSC is 80%. The routable rate for more complex LR-[4:2]CSA, LR-leapfrog, LR-CSSC-U/L is 75%. Power/delay/area in each module are also listed for comparison. There are no extra CRA modules as there is no enough time slack for CRA computation in radix-4 schemes and they are implemented as a part of the final fast CPAs.

Table 4.7: Power in radix-4 LR multipliers under *random* data

Schemes	Power (mW) $(50MHz)$			
	PPGR	eCSA	CPA	total
RL-baseline	22.23	0	3.90	26.13 (1.00)
LR	20.28	1.06	4.71	26.05 (1.00)
LR-CRA	23.50	0	1.78	25.28 (0.93)
LR-CSSC	17.06	0.90	3.77	21.73 (0.83)
LR-[4:2]CSA	19.75	0.91	3.93	24.59 (0.94)
LR-leapfrog	15.46	3.12	3.57	22.15 (0.85)
LR-CSSC-U/L	13.26	3.55	3.84	**20.65 (0.79)**

Table 4.8: Power in radix-4 LR multipliers under *djpeg* data

Schemes	Power (mW) $(50MHz)$			
	PPGR	eCSA	CPA	total
RL-baseline	10.15	0	2.49	12.64 (1.00)
LR	7.47	0.19	3.07	10.73 (0.85)
LR-CRA	8.8	0	1.27	10.07 (0.80)
LR-CSSC	6.44	0.15	2.52	**9.11 (0.72)**
LR-[4:2]CSA	5.33	0.20	2.62	10.26 (0.81)
LR-leapfrog	4.00	1.01	2.40	9.87 (0.78)
LR-CSSC-U/L	4.12	0.79	2.35	9.32 (0.74)

For the basic radix-4 LR scheme, the power consumption in PPGR and eCSA modules is reduced by 4% under *random* and 25% under *djpeg*. As for the overall power consumption, there is no reduction under *random* and only 15% reduction under *djpeg* because the power in the final CPA is increased. The power increase

Table 4.9: Area in radix-4 LR multipliers

Schemes	Area (μm^2)			
	PPGR	eCSA	CPA	total
RL-baseline	65816	0	6091	71907 (1.00)
LR	63315	1677	6995	71987 (1.00)
LR-CRA	68121	0	2249	**70370 (0.98)**
LR-CSSC	63394	1677	6995	72066 (1.00)
LR-[4:2]CSA	64921	1547	6936	73404 (1.02)
LR-leapfrog	59250	6410	7025	72695 (1.01)
LR-CSSC-U/L	60001	5868	6899	72768 (1.01)

Table 4.10: Delay in radix-4 LR multipliers

Schemes	Delay(ns)			
	PPGR	eCSA	CPA	total
RL-baseline	7.54	0	2.65	10.19 (1.00)
LR	7.32	0	2.98	10.30 (1.01)
LR-CRA	8.19	0	2.47	10.66 (1.05)
LR-CSSC	6.03	0	3.16	9.19 (0.90)
LR-[4:2]CSA	5.38	0.41	3.20	8.99 (0.88)
LR-leapfrog	5.18	0.47	2.54	**8.19 (0.80)**
LR-CSSC-U/L	4.25	0.73	3.61	8.59 (0.84)

in the final CPA of LR is due to different signal τ scenarios. The left half inputs of the CPA arrive in a stair-case profile with MSB being the earliest, which introduce more glitches than the situation in RL. In other LR schemes, the scenarios are improved because of less signal τ differences. Note that the CPA delay in the basic LR scheme can be almost removed by using on-the-fly conversion to take advantage of the stair-case profile [41][36].

Because each PP shifts two bits and there are extra correction bits, LR-CRA is less efficient in radix-4 than in radix-2 LR multipliers. Compared to the basic LR, LR-CRA reduces the power consumption by 7% under *random* and by 6% under *djpeg* with 4% delay increase. The delay is increased because the carry propagation path in each CRA row is longer than the carry save path in radix-4 LR.

LR-CSSC is still very effective. LR-CSSC achieves 15% power reduction under *djpeg* and 17% reduction under *random* from the basic LR. The delay in LR-CSSC is also 10% less. If input buffer and PPG delay ($2.38ns$) is excluded from PPGR delay, the net PPR delay is reduced by 26% (from $4.94ns$ in LR to $3.65ns$ in LR-CSSC), which also matches our theoretical analysis in Section 4.3.1. Considering that LR-CSSC has the same cell area and interconnect complexity as the basic LR, this should be a primary choice for LR array multiplier design.

With 12% delay reduction from LR, LR-[4:2]CSA only achieves 6% power reduction under *random* and 4% power reduction under *djpeg*. The delay reduction in PPR and eCSA is about 26%, which matches the analysis in Section 4.3.3. The power consumption is reduced because the signal propagation paths are shorter. However, the power reduction is limited by the balanced structure of [4:2]-CSA design. As we indicated before, the balanced [4:2]-CSA structure in Figure 4.13 does not match the linear reduction structure in array multipliers.

As to split structures, both schemes are efficient in power and delay reduction. Compared to basic LR, LR-leapfrog reduces 15% power under *random* and 8% power under *djpeg*. LR-leapfrog reduces 21% power under *random* and 13% power under *djpeg*. The delay in LR-leapfrog is reduced by 20% and the delay in

LR-CSSC-U/L is reduced by 16%. Among all schemes, LR-CSSC-U/L has the least power consumption under *random* and LR-CSSC has the least power consumption under *djpeg*. This indicates that CSSC flow optimization is a very useful power-saving technique in LR array multipliers. As to delay, LR-leapfrog is the best followed by LR-CSSC-U/L. In terms of power-delay product, LR-leapfrog and LR-CSSC-U/L are the best because array splitting reduces the PPR delay significantly. As splitting structures are more complex, however, LR-leapfrog and LR-CSSC-U/L are the best candidates only when small power-delay product is the main goal.

4.5 Summary

LR linear array multiplication provides an interesting alternative to the conventional RL array multiplication as LR computation has the potential of saving power and delay. In this chapter, we have presented reduction structure optimization techniques for radix-2 and radix-4 LR linear array multipliers. These techniques include: CSSC flow optimization, CRAs for PP reduction, [4:2]-CSAs for PP reduction, even/odd (leapfrog) split structure and upper/lower split structure. Detailed experimental results are given to compare the power/area/delay characteristics of each 32×32-bit multiplication scheme. Because the optimizations are at the architecture and algorithm level, both power reduction and delay reduction have been achieved, which demonstrates one strength of high-level optimization. Among different optimization techniques for LR array multipliers, LR-CSSC is a primary choice if power is the critical concern. LR-CSSC achieves the least power consumption in most cases with relatively small delay. When small power-delay product is the main goal, the more complex LR-leapfrog and LR-CSSC-U/L split array structures are better candidates. These more complex

PP reduction structures decreases the routable row utilization rate by 5% in our experiment.

CHAPTER 5

Design of High-Performance Low-Power Multipliers

5.1 Introduction

The structure optimizations for LR array multipliers discussed in Chapter 4 have shown that power reduction could come from delay reduction at the architecture and algorithm level. Following this direction, we combine these structure optimization techniques to design high-performance low-power array multipliers. We aim at low-power array multipliers with similar delay as tree multipliers while maintaining the regularity of array structures in the precision range $n \leq 32$.

Tree multipliers have the smallest logic delay proportional to $log(n)$. However, they have irregular layout with complicated interconnects. On the other hand, array multipliers have larger delay but offer regular layout and simpler interconnects. As interconnects become dominant in deep sub-micron design [64], architectures with regular layout and simple interconnects are preferable. Irregular layouts with complicated interconnects not only demand more physical design effort but also introduce significant interconnect capacitances that affect delay, power, and signal integrity [14, 64].

Modern tree multipliers use [4:2] adders [93] to reduce the PPR logic delay and regularize the layout. To develop regular and compact layout, regularly structured tree (RST) with recurring blocks [55] and rectangular-styled tree by folding [63] were proposed. These schemes achieve regular layout at the expense of more complicated interconnects. In [99][114], three dimensional minimization (TDM) algorithm was proposed to design column counters of the maximal possible size with optimized signal connections, which further shortened the PPR path by $1 \sim 2$ XOR2 gates. However, the resulting structure has more complex wiring and layout than a [4:2]-adder based tree. In [38][69], multiplication was divided recursively into smaller multiplications to increase layout regularity and scalability, which essentially resulted in a hierarchical tree structure.

In conventional RL linear array multiplier design, the even/odd split structure [65, 100] was proposed to reduce both delay and power. In [84], an improved even/odd structure *leapfrog* was proposed to take advantage of the delay imbalances in adders. In [41], a LR carry-free (LRCF) array multiplier was proposed where the final CPA step to produce the MS bits of the product was avoided by on-the-fly conversion in parallel with the linear reduction. In [36], this LRCF approach was extended to produce a $2n$-bit product.

Except in LRCF, final CPAs are required and contribute significant delay in multipliers. When PPR delay is reduced by adder trees, the delay of CPA becomes more evident. The optimization of the final CPA was considered in [113] where the adder was partitioned into several conditional-sum adder (CSUMA) blocks according to the arrival profile of the inputs. In [138], a non-heuristic algorithm was proposed to synthesize the family of prefix adders comprising carry-ripple, carry-increment, carry-lookahead adders (CLAs), and many more. Starting from a serial-prefix graph, the first step of this algorithm performs all possible depth-decreasing transformations to produce the fastest prefix structure. The second step performs size-decreasing transformations at the cost of an increased depth without violating the depth constraints. This algorithm results in size-optimal parallel-prefix adder structures under arbitrary depth constraints including non-uniform input and output signal arrival times. In [131], multi-level CSUMAs (MLCSUMAs) were used to optimize the final adder bit by bit from LSB to MSB, which was improved to become a generalized earliest-first (GEF) algorithm covering both CSUMA and CLA in [132]. In [73, 121], the on-the-fly converter for LRCF array multipliers was replaced by a multi-level carry-select adder (CSELA) or CSUMA.

In this chapter, we combine structure optimization techniques in Chapter 4 and propose split array LRLF multipliers (SALRLF) with signal flow optimization. From the basic LRLF structure, two types of further splitting are considered: even/odd and upper/lower. Because LRLF well maintains the regularity of array multipliers, this new splitting does not result in a hierarchical tree structure. Instead, SALRLF has a simpler structure than a tree multiplier and requires less design effort. For tree multipliers, we optimize the reduction structures by developing a special [9:4] adder with only $3T_{XOR2}$ delay. Besides the optimizations in PPR, PPG and final CPA are also optimized for delay. SALRLF is compared with structure-optimized array multipliers in Chapter 4 as well as tree multipliers. Because of the distinct structures of array and tree, physical layouts with guided floorplanning are conducted for comparison.

5.2 Partial Product Generation

We consider radix-4 recoding only. From Chapter 2, we select parallel recoding NPR3b for high-performance multipliers. Due to shifting, each PP has a 0 between $PP_{i+1,0}$ and correction bit cor_i. To have a more regular LSB part of each PP, $PP_{i,0}$ is added with cor_i in advance [131]. The $PP_{i,0}^{(new)}$ and $cor_i^{(new)}$ are described as

$$PP_{i,0}^{(new)} = x_0 \cdot (y_{2i} \oplus y_{2i-1}) \tag{5.1}$$

$$cor_i^{(new)} = y_{2i+1}y'_{2i}y'_{2i-1} + y_{2i+1}x'_0(y_{2i} \oplus y_{2i-1}) \tag{5.2}$$

Both $cor_i^{(new)}$ and $P_{i,0}^{(new)}$ are obtained no later than other PP bits.

For high-performance array multiplier design, the generated PP bit-array is arranged in an LR manner as shown in Figure 5.1. The grey circles are $PP_{i,0}^{(new)}$ and the white circles are $cor_i^{(new)}$. The sign-extension constants are used in the first row for reduced area and delay. If they are placed on the left of each PP row, the array reduction of $PP_{n/2}$, $PP_{n/2-1}$, and $PP_{n/2-2}$ still requires one full adder (FA) and $(n-3)$ half adders (HAs) although $PP_{n/2}$ only contains one bit $cor_{n/2-1}^{(new)}$. For each 1 on the left, one extra FA has to be used during reduction and $n/2$ extra FAs are needed in total. When these 1's are placed in the first row, however, no extra FA is needed. These 1's are assimilated using existing HAs by modifying them to be: $Sum = \overline{a \oplus b}$ and $Carry = a + b$.

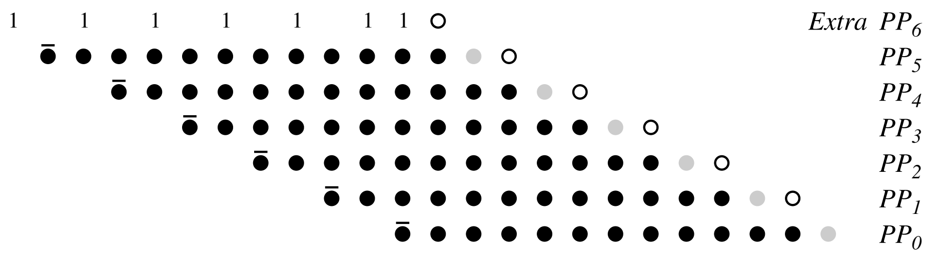


Figure 5.1: MSB-first radix-4 PP bit array (n=12).

5.3 Partial Product Reduction by Split Array Structure

The delay gap between tree multipliers and array multipliers is due to the linear PPR structure in array multipliers. To improve the speed of array multipliers, parallelism is introduced in PPR. In addition, different adder types and the signal flow between adders also have impact on delay, area, and power.

5.3.1 Split Array LRLF PP reduction

The PPR critical path of an LRLF array multiplier in Chapter 4 is about $\lceil \frac{n}{2} \rceil$ XOR2 gates while that of an $n \times n$-bit radix-4 tree multiplier is $3(\lceil \log_2(\frac{n}{4}) \rceil)$ XOR2 gates. The delay of LRLF is not comparable with the delay of tree multipliers when $n \geq 16$. To reduce PPR delay, another level of parallelism is necessary. LRLF is used as the basic structure to maintain the regularity of array multipliers.

One approach is to split the PP bit array into even PPs and odd PPs. In each split part, PPs are shifted four bits each row and reduced into two vectors using a LRLF structure. The final vectors from even and odd parts are added by a $(2n-3)$-bit [4:2] adder. This algorithm is named as even/odd LRLF (EOLRLF) and a high-level description is shown in Figure 5.2. A structure example is illustrated in Figure 5.3.

Another approach is to split the PP bit array into upper PPs and lower PPs. In each part, PPs are shifted two bits each row and reduced into two vectors by a LRLF structure. The final vectors from upper and lower parts are also added by a [4:2] adder. To reduce the size of this adder, the highest carry bit from the right-side [3:2]-CSA of LRLF in Figure 4.16 is fed into the left-side [4:2]-CSA as Tin_0 instead of being a bit of the final vector. For upper half PPs, a portion of this modified LRLF is shown in Figure 5.4. The right-side [3:2]-CSA is still $(n-3)$-bit while the left-side [4:2]-CSA is $(n/2-1)$-bit. For low half PPs, a similar structure is applied. In this way, only a $(n+2)$-bit [4:2] adder is required because of an empty position on the $(\frac{3}{2}n+1)$-th column, as shown in Figure 5.5. This algorithm is named as upper/lower LRLF (ULLRLF) and a high-level description

```
$h = n/2$: the number of PPs;
$q = h/2$: half of $h$ and assume to be even;
$(ES, EC)_i$: sum and carry vectors in even part;
$(OS, OC)_i$: sum and carry vectors in odd part;

/* odd part using LRLF */
$(OS, OC)_0 = \textbf{CSA}(PP_h, PP_{h-1}, PP_{h-3})$;
$(OS, OC)_1 = \textbf{CSA}(PP_{h-5}, PP_{h-7}, OC_0)$;
for $i$ from 2 to $q-3$ do
  $(OS, OC)_i = \textbf{CSA}(PP_{h-5-2i}, OS_{i-2}, OC_{i-1})$;
end for
$(OS, OC)_{q-2} = \textbf{CSA}(OS_{q-3}, OC_{q-3}, OS_{q-4})$;

/* even part using LRLF */
$(ES, EC)_0 = \textbf{CSA}(PP_{h-2}, PP_{h-4}, PP_{h-6})$;
$(ES, EC)_1 = \textbf{CSA}(PP_{h-8}, PP_{h-10}, EC_0)$;
for $i$ from 2 to $q-4$ do
  $(ES, EC)_i = \textbf{CSA}(PP_{h-8-2i}, ES_{i-2}, EC_{i-1})$;
end for
$(ES, EC)_{q-3} = \textbf{CSA}(ES_{q-4}, EC_{q-4}, ES_{q-5})$;

/* add even and odd vectors using [4:2]-CSA */
$(PS, PC) = \textbf{CSA42}((OS, OC)_{q-2}, (ES, EC)_{q-3})$;
```

Figure 5.2: High-level EOLRLF algorithm.

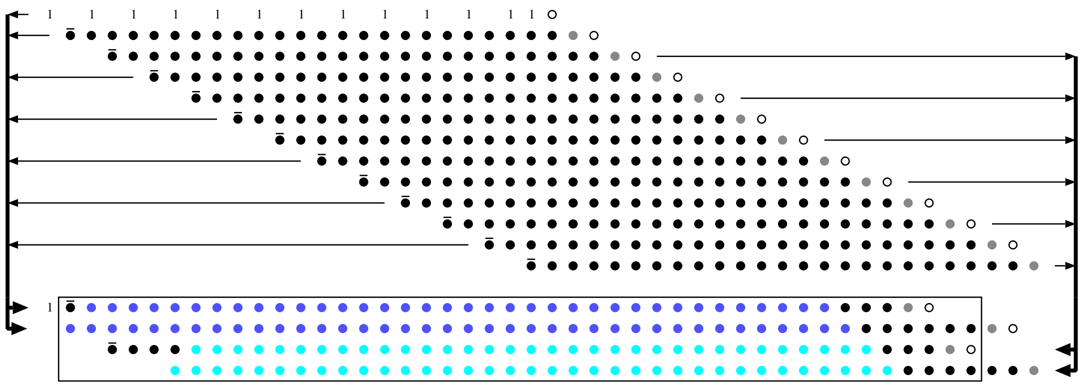


Figure 5.3: EOLRLF array multiplier (n=24).

is shown in Figure 5.6.

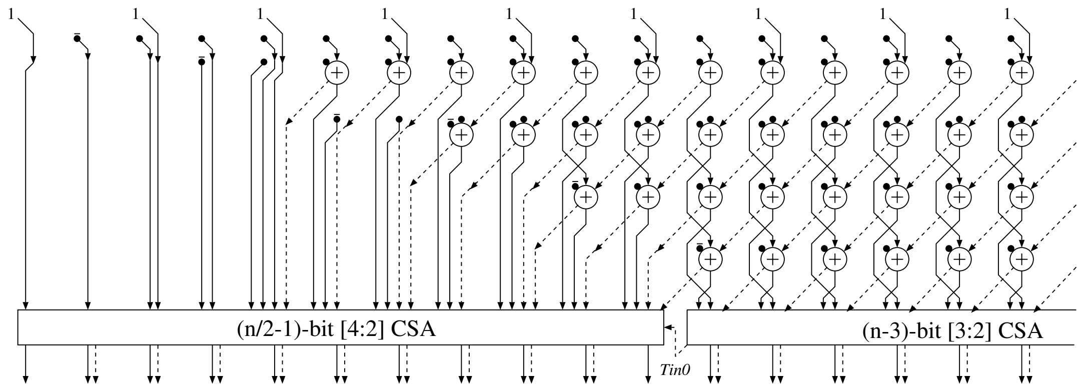


Figure 5.4: Portion of a LRLF structure for upper half PPs (n=24).

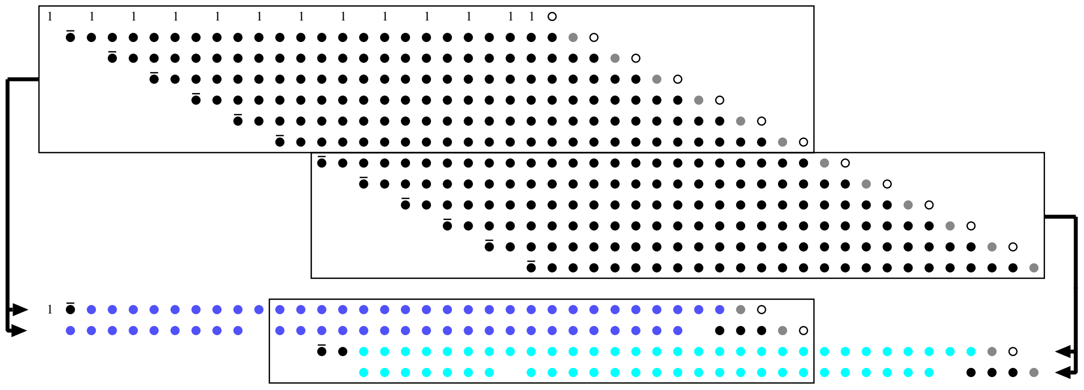


Figure 5.5: ULLRLF array multiplier (n=24).

The PPR delay of SALRLF is about $(\lceil \frac{n}{4}+3\rceil \sim \lceil \frac{n}{4}+4\rceil)T_{XOR2}$, depending on the type of adders used. For $n \leq 32$, the delay is $< 11 \sim 12$ while the best result of a tree multiplier is ≤ 9. Further splitting of the PP array reduces the layout regularity and will not be considered. Instead, optimization of CSAs and the final CPA will be used to narrow the remaining gap. In EOLRLF, the arrival profile of PPR final vectors has fewer latest-arriving bits than that in tree multipliers. Figure 5.7 shows the PPGR delay profiles in a TDM multiplier, an EOLRLF, and a ULLRLF. The number of latest-arriving bits in EOLRLF is 5 while this

```
$h = n/2$: the number of PPs;
$q = h/2$: half of $h$ and assume to be even;
$(US, UC)_i$: sum and carry vectors in upper part;
$(LS, LC)_i$: sum and carry vectors in lower part;

/* upper part using LRLF */
$(US, UC)_0 = \mathbf{CSA}(PP_h, PP_{h-1}, PP_{h-2})$;
$(US, UC)_1 = \mathbf{CSA}(PP_{h-3}, PP_{h-4}, UC_0)$;
for $i$ from 2 to $q-3$ do
  $(US, UC)_i = \mathbf{CSA}(PP_{h-3-i}, US_{i-2}, UC_{i-1})$;
end for
$(US, UC)_{q-2} = \mathbf{CSA}(US_{q-3}, UC_{q-3}, US_{q-4})$;

/* lower part using LRLF */
$(LS, LC)_0 = \mathbf{CSA}(PP_{q-1}, PP_{q-2}, PP_{q-3})$;
$(LS, LC)_1 = \mathbf{CSA}(PP_{q-4}, PP_{q-5}, LC_0)$;
for $i$ from 2 to $q-4$ do
  $(LS, LC)_i = \mathbf{CSA}(PP_{q-4-i}, LS_{i-2}, LC_{i-1})$;
end for
$(LS, LC)_{q-3} = \mathbf{CSA}(LS_{q-4}, LC_{q-4}, LS_{q-5})$;

/* add upper and lower vectors using [4:2]-CSA */
$(PS, PC) = \mathbf{CSA42}((US, UC)_{q-2}, (LS, LC)_{q-3})$;
```

Figure 5.6: High-level ULLRLF algorithm.

number is 8 in TDM. The bit delay distribution in EOLRLF is also more regular. Most bit groups in EOLRLF have 4-5 bits. But the group size varies a lot in TDM. The final adder design could exploit these better-shaped arrival profiles in EOLRLF to reduce delay.

Compared with EOLRLF, ULLRLF has two main advantages. First, the shifting distance between PPs in each upper/lower part is 2 bits instead of 4, which leads to simpler interconnects. Second, the final [4:2] adder in ULLRLF is only $(n+2)$-bit in contrast to $(2n-3)$-bit in EOLRLF. On the other hand,

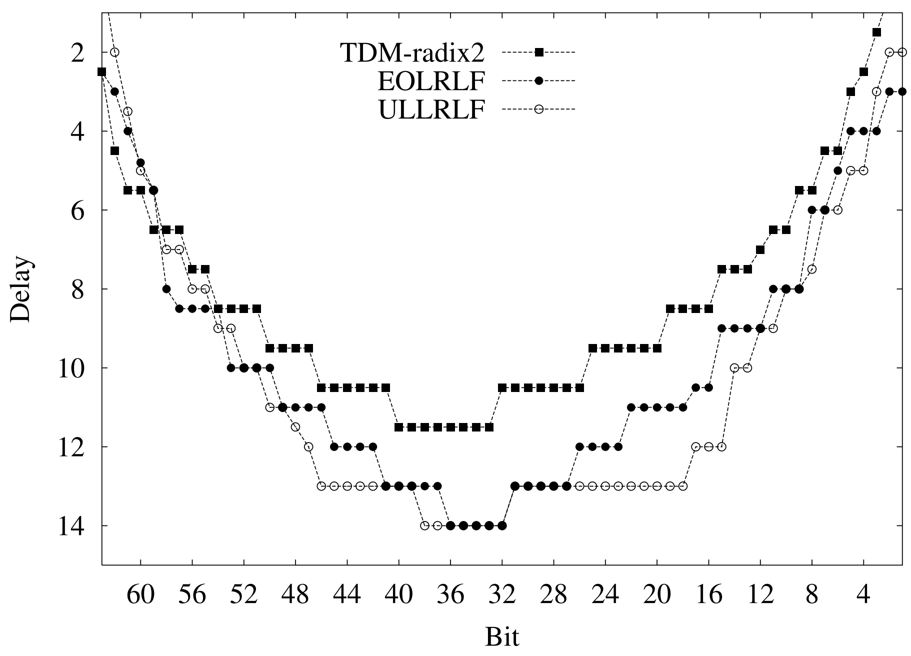


Figure 5.7: PPGR delay profiles (n=32).

ULLRLF has a worse arrival profile than EOLRLF. However, such a profile only leads to just one T_{AO21} delay, which will be explained in Section 5.5. The overall performance depends on the relative effects of these factors.

5.3.2 Optimization of [3:2]-CSAs

LRLF is based on [3:2]-CSAs and the basic components are FAs. Besides FA-MUX and FA-NAND2 discussed in Chapter 4 (Figure 4.6), FA-ND3 in Figure 5.8 is another choice. Denote $\tau_{A,B}$ as $max(\tau_A, \tau_B)$ and $\tau_{A,B,C}$ as $max(\tau_A, \tau_B, \tau_C)$. τ_{Sum} and τ_{Cout} in FA-ND3 are expressed as:

$$\tau_{Sum} = max(\tau_{A,B} + T_{XOR2}, \tau_C) + T_{XOR2} \tag{5.3}$$

$$\tau_{Cout} = \tau_{A,B,C} + T_{AO222} \tag{5.4}$$

Compared to FA-MUX and FA-NAND2, FA-ND3 is better in logic delay because the delay from all inputs to $Cout$ is T_{AO222} ($T_{AO222} \approx T_{XOR2}$). As to area, however, FA-MUX has smaller area than FA-ND3 even if pass transistors are not used to implement FA-MUX. Since FA is the most used element in array multipliers, smaller FA leads to smaller overall area, which is also helpful in the reduction of power consumption and interconnect delay.

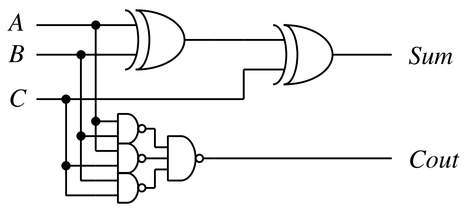


Figure 5.8: NAND3-based FA design (FA-ND3).

In Chapter 4, we have proposed signal flow optimization for FAs with one input being PP_{ij} in linear array multipliers. Here we extend the signal flow optimization to general FAs in SALRLF with the primary objective of delay reduction. Besides FAs with one PP_{ij} input, SALRLF has FAs with no PP_{ij} input in the extra CSA part. Signal flow optimization for delay has been applied in TDM tree multipliers [99]. In addition to delay, signal flow optimization affects power as studied in Chapter 4. Assume the three input signals to FA are Ain, Sin, Cin. These input signals are sorted according to their arrival times. We assume that the τ relationship is $\tau_{Ain} \le \tau_{Cin} \le \tau_{Sin}$. The order is arbitrary since the inputs are functionally equivalent. In FA-ND3, the latest Sin is connected to pin C because C is a fast input. There is no restriction on the connections between $Ain(Bin)$ and pin $A(B)$ unless transistor-level difference between A and B is considered. In FA-MUX, Sin is also connected to pin C. Between Ain and Bin, the signal with less switching activity is connected to pin A for power saving because pin B has less load capacitance and is used for the other one with more switching activity.

5.3.3 Optimization of [4:2]-CSAs

Besides [3:2]-CSAs, there is one extra [4:2]-CSA row in LRLF. These [4:2]-CSAs are also optimized according to input τ scenarios. The basic [4:2]-CSA module, M42, is shown in Chapter 4 (Figure 4.13). The delay from any input A, B, C, and D to output Sum or $Cout$ is $3T_{XOR2}$. Each M42 actually has five inputs because there is one intermediate signal Tin. In the $(n/2-1)$-bit [4:2]-CSA for LRLF in Figure 5.4, more than half 5-input M42s have one or more zero inputs, which can be simplified to have smaller delay. For M42s with one zero input, the simplification is as follows. Assume the four non-zero inputs of a simplified M42 are A, B, C, and D with τ relation $\tau_A \leq \tau_B \leq \tau_C \leq \tau_D$. The order is arbitrary as all inputs are equivalent in functionality. According to input arrival profiles, two designs with different Sum logic are developed: M42L (linear-Sum) in Figure 5.9a and M42T (tree-Sum) in 5.9b. The arrival times of $Tout$ and $Cout$ are

$$\tau_{Tout} = \tau_B + T_{AND2} \tag{5.5}$$

$$\tau_{Cout} = max(\tau_B + T_{XOR2}, \tau_D) + T_{AO222} \tag{5.6}$$

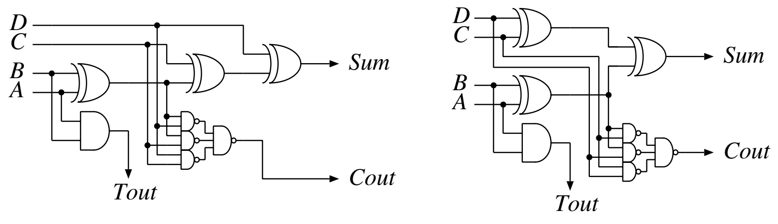
which are smaller than those in M42. In M42L, Sum arrives at

$$\tau_{linear-Sum} = max(\tau_{BCX}, \tau_D) + T_{XOR2} \tag{5.7}$$

where $\tau_{BCX} = max(\tau_B + 2T_{XOR2}, \tau_C + T_{XOR2})$. In M42T,

$$\tau_{tree-Sum} = \tau_D + 2T_{XOR2} \tag{5.8}$$

The [4:2] adder in LRLF is designed from LSB to MSB as follows. For each bit, sort five inputs A, B, C, D, and E (one of them being Tin) according to



(a) M42L: linear-*Sum* (b) M42T: tree-*Sum*

Figure 5.9: Two simplified M42 designs.

arrival time and assume that $\tau_E \leq \tau_A \leq \tau_B \leq \tau_C \leq \tau_D$. If no input is 0, M42 of Figure 4.13 is used. To minimize delay, five inputs E, A, B, C, and D are connected to pin A, B, C, D, and Tin in that order. If two or more inputs are 0s, the adder is reduced to a FA or HA. If only one input is 0, it must be E because a constant is available at the earliest time. A simplified 4-input M42 is used in this case. Four inputs A, B, C, and D are connected to pin A, B, C, and D in that order. M42L is used for faster Sum if $\tau_{linear-Sum} < \tau_{tree-Sum}$. Otherwise, M42T is used. The output $Tout$ is fed into the next bit position and the process is repeated. With this optimization, more than half [4:2]-CSAs are simplified and many outputs of the [4:2] adder become available one T_{XOR2} earlier.

5.4 Partial Product Reduction by Tree Structure

Both tree reduction and column reduction have the smallest logic delay proportional to $log(n)$ and have irregular layout with complicated interconnects. However, the regularity of tree reduction is better between them. To fairly compare SALRLF with the fastest multipliers, we also design radix-4 tree multipliers based on [4:2] and [3:2] CSAs with optimized signal flow. These optimized tree multipliers eliminate the delay due to the extra PP in radix-4 TC tree multipliers based on [4:2]-adders.

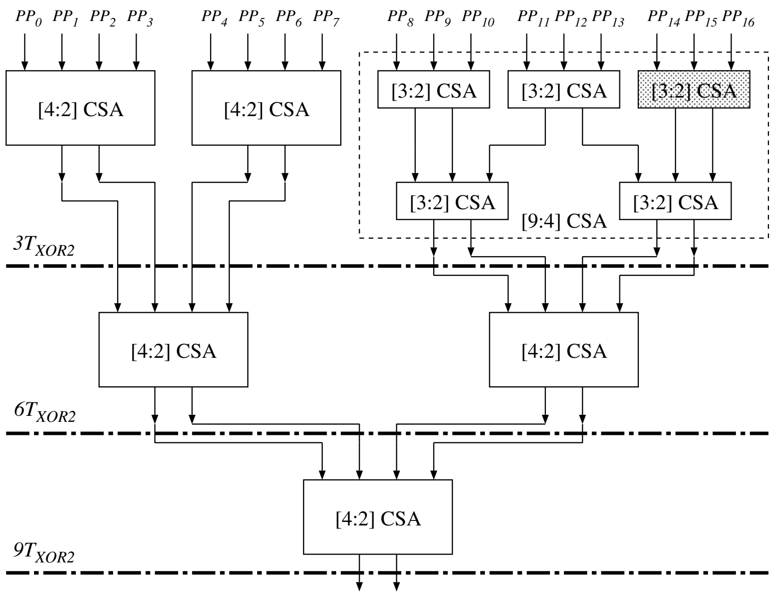

PP_0 PP_1 PP_2 PP_3
PP_4 PP_5 PP_6 PP_7
PP_8 PP_9 PP_{10}
PP_{11} PP_{12} PP_{13}
PP_{14} PP_{15} PP_{16}
[4:2] CSA
[3:2] CSA
[9:4] CSA
$3T_{XOR2}$
$6T_{XOR2}$
$9T_{XOR2}$

Figure 5.10: Tree PPR with $9T_{XOR2}$ delay (n=32).

For $n = 32$, the optimized tree multiplier is shown in Fig. 5.10. To avoid the delay due to extra $PP[n/2]$, the reduction of 9 PPs from $PP[n/2-8]$ to $PP[n/2]$ is based on a [9:4]-CSA with only $3T_{XOR2}$ delay, as illustrated in Fig. 5.11. The $3T_{XOR2}$ delay is achieved as follows. By using FA-ND3, carries are computed faster than sums. FAs with one constant inputs in the shaded CSA in Figure 5.10 are simplified to be half adders with half carry/sum delay. PS_{k-1} and PC_{k-1} are then switched and input into two second-level [3:2]-CSAs. In the dot graph for second-level [3:2]-CSAs, the arrival times of signals are illustrated by the gray levels of dots: the lighter the earlier. Those signals coming directly from the first level are marked as circles. It is shown that all FAs in the second-level [3:2]-CSAs have one input arriving at least T_{XOR2} later than the other two inputs. This late input is connected pin C of FAs to ensure one T_{XOR2} delay in second-level CSAs. The only HA with two late inputs also has one T_{XOR2} delay because it is a HA. The bit of PC_k on the dashed-line column must be empty (zero) in order to guarantee a HA with one T_{XOR2}. To distinguish from other tree multipliers, the radix-4 tree multiplier using this special [9:4]-CSA is named *tree9to4*.

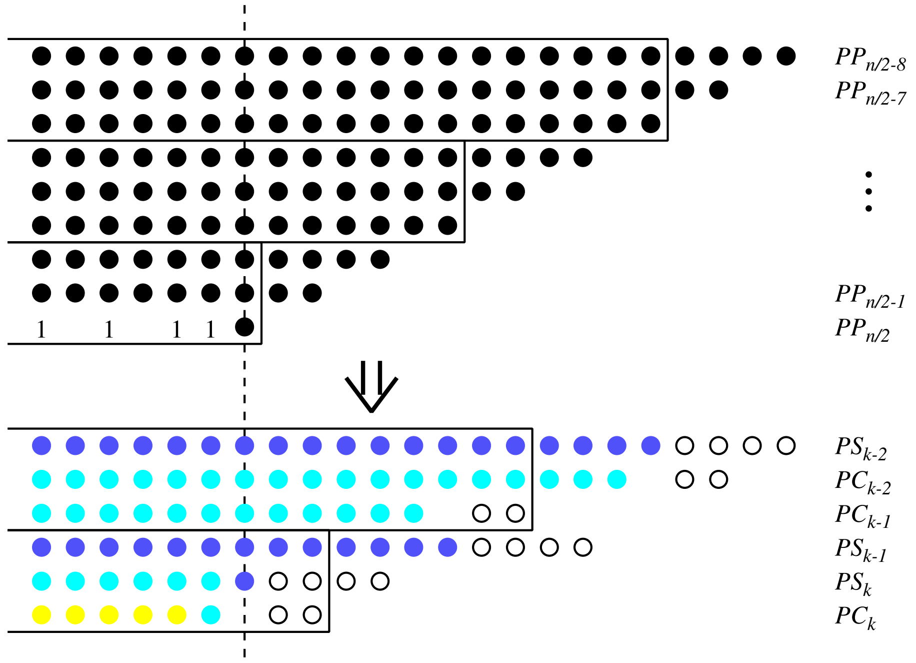


Figure 5.11: A special [9:4]-CSA with $3T_{XOR2}$ delay.

5.5 Final Addition Optimized for Arbitrary Input Arrival Time

Final adders are optimized to match the non-uniform input arrival profiles. The optimal final adder for tree multipliers is CSUMA-based design [113]. Efficient design of the on-the-fly converter for LR array multipliers also corresponds to a multi-level CSELA or CSUMA optimized for staircase signal arrivals [73]. The CSUMA/CSELA for on-the-fly converter operates in parallel with the reduction and introduce little CPA delay, which is different from a traditional CSUMA/CSELA in multipliers. In [138], the fastest prefix adder (PA) structures under arbitrary depth constraints are synthesized by depth-decreasing transformations (prefix graph compression) starting from a serial-prefix graph. The family of PAs comprises carry-ripple, carry-increment, carry-lookahead adders (CLAs), and many more. In [132], the similarity between CSUMA and CLA is formulated and a generalized earliest-first (GEF) algorithm was proposed to design CSUMA/CLA for arbitrary input arrival profile. The difference between prefix graph compression and GEF is that graph compression is a recursive optimization approach while GEF is a straightforward constructive approach. Both the prefix graph transformation algorithm and the GEF algorithm can generate adder-based on-the-fly converters given proper LR staircase signal arrivals. Thus, we do not distinguish on-the-fly converters from CPAs because these synthesis algorithms handle arbitrary signal arrivals.

We follow the earliest-first idea of GEF algorithm and develop an improved version for PA design. We choose PA for final addition because the PA operators, AO21 and AND2, are simpler than the basic CSUMA operators – a pair of MUX21. The algorithm is outlined in Figure 5.12. The basic idea is to sort the carry generate and alive signal pair $(G, A)^{L[i]}_{i,R[i]}$ (denoted as GA_i) according to arrival times and combine adjacent GAs in an earliest-first manner. Two lists, $Plist$ and $Tlist$, are maintained in the process. All (G, A) signal pairs are initially put into $Plist$ and sorted according to arrival times. The earliest pairs are then moved to $Tlist$. Adjacent signal pairs in $Tlist$ are retrieved and combined by operator '$\bullet$' from left to right. The combined pairs are put back into $Plist$. The iteration continues until the generation of the MSB carry bit. Other carry bits are generated using existing (G, A) bits. A PA example for a hill-shaped arrival profile is shown in Fig. 5.13. Black nodes in PA are computation cells and white nodes have no logic or only buffers.

```
$W$: width of input data;
$GA_i = (G, A)^{L[i]}_{i,R[i]}$: level-$L[i]$ (G,A) in bit column $i$;
$R[i]$: the right-most bit covered by $(G, A)^{L[i]}_{i,R[i]}$;

– step 1: level-0 (G,A) –
Plist = null; Tlist = null;
for each bit $i$ do
  $L[i] = 1$; $R[i] = i$;
  generate $(G, A)^{L[i]}_{i,R[i]}$ and put into Plist;
end for

– step 2: generate MSB-carry –
while length(Plist) + length(Tlist) > 1 do
  sort Plist in delay-ascending order;
  TG0 = Delay of $G$ signal of Plist[0];
  move items with TG=TG0 from Plist to Tlist;
  sort Tlist in bit-descending order (MSB-first);
  while two items are adjacent in Tlist do
      remove them ($GA_i$ and $GA_j$) from Tlist;
      $(G, A)^{L[i]+1}_{i,R[j]} = GA_i \bullet GA_j$;
      $R[i] = R[j]$; $L[i] = L[i] + 1$;
      insert new $GA_i$ into Plist;
  end while
end while

– step 3: generate other carries –
for each bit $i$ from $W - 2$ downto 1 do
  while $R[i] \neq 0$ do
    $j = R[i] - 1$; /* the bit to be combined */
    $(G, A)^{L[i]+1}_{i,R[j]} = GA_i \bullet GA_j$;
    $R[i] = R[j]$; $L[i] = L[i] + 1$;
  end while
end for
```

Figure 5.12: Earlist-first PA carry generation algorithm (adapted from [132]).

Let W_{max} be the largest number of adjacent signals that arrive at the same time. If these W_{max} signals are also the latest arriving signals in a hill-shaped

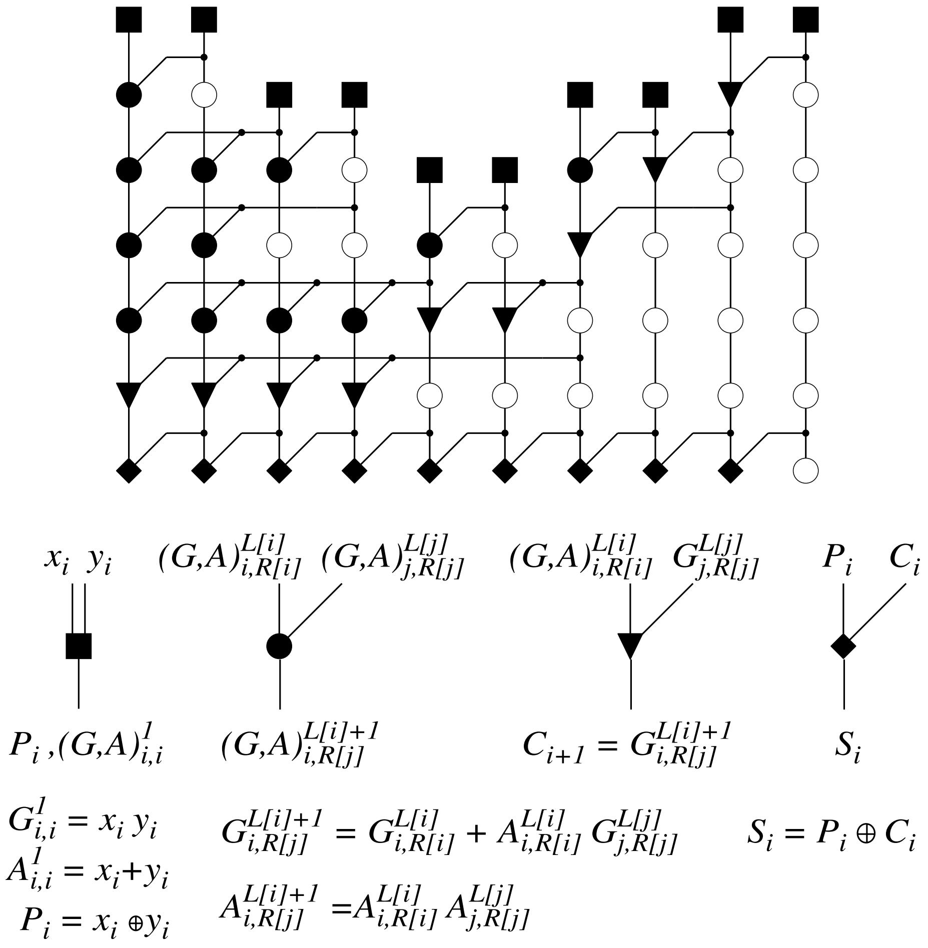

x_i y_i
$(G,A)^{L[i]}_{i,R[i]}$ $(G,A)^{L[j]}_{j,R[j]}$
$(G,A)^{L[i]}_{i,R[i]}$ $G^{L[j]}_{j,R[j]}$
P_i C_i
$P_i, (G,A)^1_{i,i}$
$(G,A)^{L[i]+1}_{i,R[j]}$
$C_{i+1} = G^{L[i]+1}_{i,R[j]}$
S_i
$G^1_{i,i} = x_i y_i$
$A^1_{i,i} = x_i + y_i$
$P_i = x_i \oplus y_i$
$G^{L[i]+1}_{i,R[j]} = G^{L[i]}_{i,R[i]} + A^{L[i]}_{i,R[i]} G^{L[j]}_{j,R[j]}$
$A^{L[i]+1}_{i,R[j]} = A^{L[i]}_{i,R[i]} A^{L[j]}_{j,R[j]}$
$S_i = P_i \oplus C_i$

Figure 5.13: A PA example.

arrival profile, the delay of PA for such a profile is estimated as

$$T_{PA} = (\log_2(W_{max}) + 2)T_{AO21} + T_{XOR2} \tag{5.9}$$

which is not directly related to the adder width $2n$. A small W_{max} would lead to a small T_{PA}. However, the difference in T_{PA} is just one T_{AO21} for most schemes in our study because of the logarithmic relationship. One T_{AO21} delay could be further eliminated from T_{PA} if carry-select adders are used for the final stages of the left part in hill-shaped arrival profiles [34].

There are two modifications to the original GEF. First, the adjacent signal pairs in *Tlist* are combined from MSB to LSB rather than from LSB to MSB. Because the path from $G_{j,R[j]}^{L[j]}$ to $G_{i,R[j]}^{L[i]+1}$ has one more AND gate than the path from $G_{i,R[i]}^{L[i]}$ to $G_{i,R[j]}^{L[i]+1}$ in operator '$\bullet$', this MSB-to-LSB combining leads to an improvement of one AND gate delay when there are odd numbers of adjacent signals. For the example in Figure 5.14, the critical-path delay is improved by $0.5T_{XOR2}$. Second, the generation of non-MSB carry signals does not require explicit carry decomposition or the use of GEF again to schedule the order of operation. Instead, it is a direct merging process from MSB to LSB. As outlined in Figure 5.12, a new GA_i is recursively generated until it covers all input bits ($R[i] = 0$). In the generation, existing GA_j ($j < i$) is used automatically.

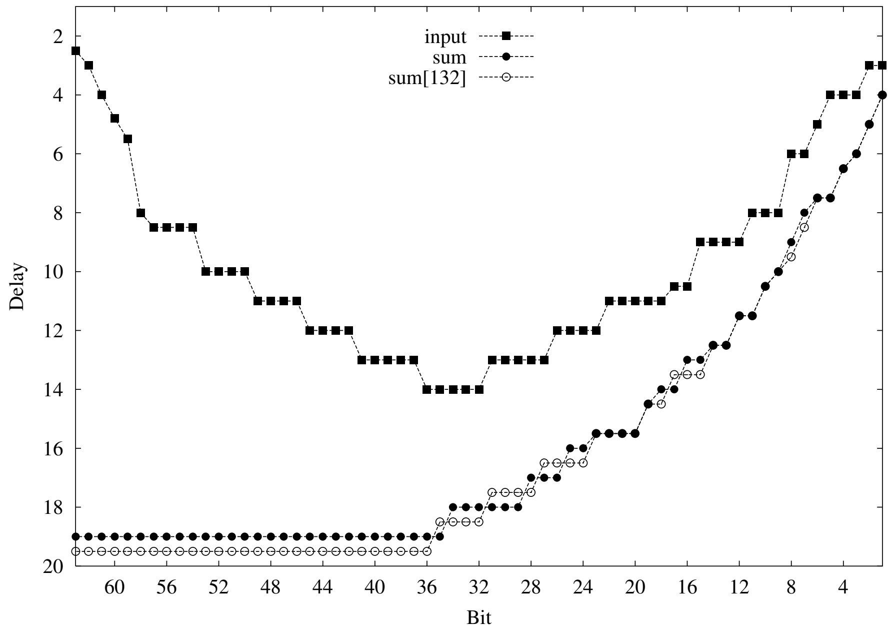


Figure 5.14: PA delay profile in EOLRLF (n=32).

5.6 Experimental Evaluation

To compare the proposed SALRLFs with LRLFs in Chapter 4 and tree multipliers, logic-level delay analysis is first conducted. Actual VHDL implementation and physical layout with guided floorplanning are then performed for more realistic power/delay/area comparison.

5.6.1 Delay Comparison at Logic Level

VHDL generation programs for LRLF and SALRLF algorithms have been written with the flexibility of FA selection and signal flow optimization. The comparison results at logic level without layout effects are normalized to T_{XOR2} and listed in Table 5.1. Each scheme is tested with the selection of FA-MUX or FA-ND3 and the optional signal flow optimization (SFO). T_{GR} is the delay of PPG and PPR. T_A is the delay of the final adder. The total delay $T = T_{GR} + T_A$. The smallest delay of each multiplier type is highlighted in boldface. For LRLF, the use of FA-ND3 rather than FA-MUX reduces PPR delay and the overall delay by 1 T_{XOR2}. SFO reduces the delay by 1 T_{XOR2} except for 48-bit LRLF-ND3 where the reduction is 2. For EOLRLF, FA-ND3 reduces 1 T_{XOR2} in PPR but the worse CPA input profile cancels the gain. SFO in EOLRLF-MUX reduces 1 T_{XOR2} but again the worse CPA input profile cancels the gain. SFO in EOLRLF-ND3 helps reduce the overall delay by $0.5 \sim 1$ T_{XOR2}. For ULLRLF, FA-ND3 reduces one T_{XOR2} in PPR, but not the overall delay. SFO only reduces PPR delay in ULLRLF-MUX by 0.5 and also has no effect on the overall delay. Varying FAs and applying SFO change the input arrival profiles of the final adder and affect T_A by up to 1 T_{XOR2}. Even if there is little delay advantage, however, it is still useful to apply SFO for power reduction, as demonstrated in Chapter 4. As to the overall logic delay, EOLRLF is the best because of interleaved splitting.

Finally, it is worthwhile to note that T_A does have little relation with the adder width as explained in Equation 5.9.

Table 5.1: Effects of FA type and SFO on logic delay in LRLF/SALRLF

Schemes	$n = 24$			$n = 32$			$n = 48$		
	T_{GR}	T_A	T	T_{GR}	T_A	T	T_{GR}	T_A	T
LRLF-MUX	15	5	20	19	5	24	27	5	32
LRLF-MUX-SFO	14	5	19	18	5	23	26	5	31
LRLF-ND3	14	5	19	18	5	23	26	5	31
LRLF-ND3-SFO	13	5	**18**	17	5	**22**	24.5	4.5	**29**
EOLRLF-MUX	12	5	17	14	5	19	18	5	23
EOLRLF-MUX-SFO	11	6	17	13	6	19	17	6	23
EOLRLF-ND3	11	6	17	13	6	19	17	6	23
EOLRLF-ND3-SFO	11	5.5	**16.5**	12.5	6	**18.5**	16.5	5.5	**22**
ULLRLF-MUX	12	5	17	14	6	20	18	6	24
ULLRLF-MUX-SFO	11.5	5.5	17	13.5	6.5	20	17.5	6.5	24
ULLRLF-ND3	11	6	17	13	6	**19**	17	7	24
ULLRLF-ND3-SFO	11	6	17	13	6	**19**	17	7	24

T_{GR}: the delay of PPG and PPR; T_A: the delay of final adder; $T = T_{GR} + T_A$

Using the best results from Table 5.1, we now compare the delays of LRLF and SALRLF with tree multipliers. Besides our proposed tree9to4, radix-2 and radix-4 TDM schemes based on column reduction [99][131] are chosen because they are the fastest multipliers at the logic level to our knowledge. The delay comparison results are given in Table 5.2. The blank boxes with '–' are because T_{GR}s or delay profiles from PPR are not available from literature. The original TDM-radix4 data in [131] are normalized to our measurement base. It is shown that the radix-4 tree multipliers based on our [9:4] adder design have almost the same T_{PPGR} as TDM schemes. For $n \leq 32$, the delay of EOLRLFs is $0.5T_{XOR2}$ less than ULLRLFs and is very close to that of tree multipliers including TDM. For larger $n = 48$, EOLRLF shows 13% more gate delay and ULLRLF shows 23% more gate delay. If another level of parallelism is allowed, the delay difference could be reduced to be within 10% while the regularity is degraded towards a tree. We don't consider this further parallelism here.

Table 5.2: Logic delay comparison of tree multipliers and LRLF/SALRLF

Schemes	$n = 24$			$n = 32$			$n = 48$		
	T_{GR}	T_A	T	T_{GR}	T_A	T	T_{GR}	T_A	T
TDM-radix2	10.5	–	–	11.5	6	**17.5**	13.5	6	**19.5**
TDM-radix4	11	5.5	16.5	12	6	18	–	–	–
Tree9to4	10	6	**16**	11	7	18	13	7	20
LRLF	13	5	18	17	5	22	24.5	4.5	29
EOLRLF	11	5.5	16.5	12.5	6	18.5	16.5	5.5	22
ULLRLF	11	6	17	13	6	19	17	7	24

5.6.2 Power/Delay/Area Comparison at Physical Layout Level

Three radix-4 schemes for 24×24-bit and 32×32-bit multiplications are compared: tree9to4, EOLRLF-MUX-SFO, and ULLRLF-MUX-SFO. Because VHDL descriptions of TDM multipliers are not available, we use our own tree multiplier tree9to4 instead. Optimized with a special [9:4] adder, tree9to4 has the similar delay as TDM but is more regular. MUX-SFO based designs are chosen because they have smaller area than ND3-SFO based designs while the overall logic delays are very close. SFO is also applied to tree9to4 for power.

For layout experiments, we first conduct automatic layout and the results are shown in Table 5.3 and 5.4. For 24×24-bit multipliers, power is measured at $100MHz$ using only *random* test data as we do not have 24-bit test data with a large dynamic range. For 32×32-bit multipliers, power is measured at $100MHz$ using both *random* and *djpeg* test data. For routability, the row utilization rates are set to 80% in 24×24-bit multipliers and 70% in 32×32-bit multipliers. We find that it is more difficult to route a larger multiplier even for the same type. *CArea* is the total area of all cells. The actual die area (excluding IO area) is *CArea* divided by the row utilization rate.

The experimental results show that EOLRLFs are worse than ULLRLFs in all measured parameters. As analyzed in section 5.3.1, EOLRLF has 2-bit more shifting distance and an extra $(n-5)$-bit [4:2] adder compared to ULLRLF, which complicates the layout and eliminates the one T_{XOR2} delay advantage. For 24-bit, ULLRLF consumes 6% less power than tree9to4 and is slightly better in area and delay. For 32-bit, ULLRLF and tree9to4 have similar area and delay while ULLRLF consumes $8 \sim 10\%$ less power. Because tree9to4 is more regular than column reduction based TDM, we believe that TDM would have worse results than tree9to4 if physical layout is conducted. Note that the radix-4 tree multipliers we have studied in Chapter 2 use un-optimized two-level CLAs for final CPA. These two-level CLAs are smaller and slower than optimized PAs in this Chapter. Because of the optimized [9:4]-CSA and final CPA, tree9to4 is 17% faster than the radix-4 [4:2]-CSA tree multiplier in Chapter 2 while consuming 6% more power. Compared with the fastest LR-leapfrog in Chapter 4, ULLRLF improves the delay by 12% and reduces the power consumption by 8% if power is measured at the same frequency using *random* test data.

For 32×32-bit tree9to4 and ULLRLF, we further conduct layout with guided

Table 5.3: Power/area/delay after automatic layout (24-bit)

Schemes	Power(mW)	CArea(μm^2)	Delay(ns)
tree9to4	26.27 (1.00)	44680 (1.00)	6.01 (1.00)
EOLRLF	25.06 (0.95)	44680 (1.00)	5.91 (0.98)
ULLRLF	**24.59 (0.94)**	**43217 (0.97)**	**5.88 (0.98)**

Table 5.4: Power/area/delay after automatic layout (32-bit)

Schemes	Power(mW) (100MHz)		CArea(μm^2)	Delay(ns)
	Random	Djpeg		
tree9to4	45.12 (1.00)	21.42 (1.00)	76434 (1.00)	**7.18 (1.00)**
EOLRLF	43.48 (0.96)	22.15 (1.03)	77711 (1.02)	7.36 (1.02)
ULLRLF	**40.65 (0.90)**	**19.65 (0.92)**	**74598 (0.98)**	7.25 (1.01)

floorplanning by specifying placement regions. The floorplan of tree9to4 is shown in Figure 5.15, which is based on H-tree for symmetry and regularity [123]. Because of region constraints, the row utilization rate is relaxed to 63% from 70% in automatic layout for routability. In addition, all blocks have to be assigned to specific regions for delay reduction. We have tried coarser region specifications and different floorplans and got worse results. The floorplan of ULLRLF is shown in Figure 5.16, which is simpler and more regular than the floorplan of tree9to4. The left-side [4:2] CSA in each part is distributed into PPR rows. Despite of region constraints, the routable rate remains at 70%. Regions are assigned for two big upper/lower blocks, final [4:2] adder, and CPA. The results are shown in Table 5.5. The delay is improved by 4% from automatic layout. For ULLRLF, there is no area cost for this delay improvement because of the regularity of ULLRLF. For tree9to4, the die area increases 10%. After layout with guided floorplanning, ULLRLF and tree9to4 still have similar delay while ULLRLF has 13% less die area and $7 \sim 8\%$ less power. The power consumptions after guided layout are slightly larger than those after automatic layout although delays are reduced. This is because guided floorplanning is primarily for delay reduction on the critical paths. Region constraints manually insert placement boundaries and affect global optimization. The overall placement quality can degrade because of overconstraining [20].

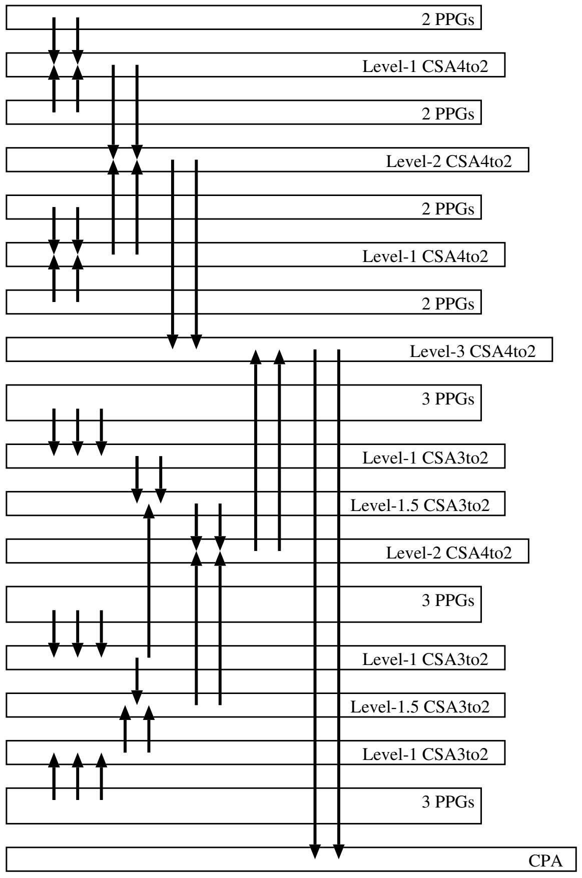


Figure 5.15: Floorplan of tree9to4 (n=32).

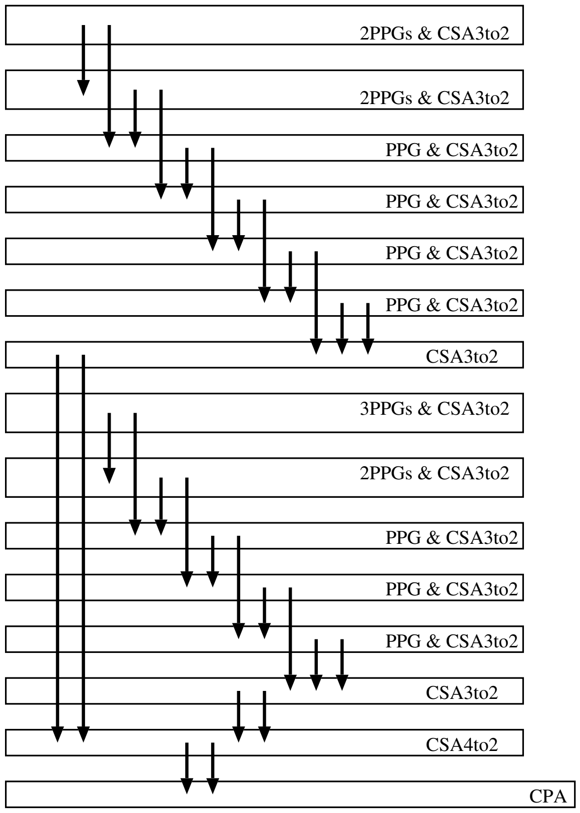


Figure 5.16: Floorplan of ULLRLF (n=32).

Table 5.5: Power/area/delay after layout with guided floorplanning (32-bit)

Schemes	Power(mW) $(100MHz)$		CArea(μm^2)	Rate	Delay(ns)
	Random	Djpeg			
tree9to4	45.53 (1.00)	21.09 (1.00)	76434 (1.00)	63%	**6.90 (1.00)**
ULLRLF	**41.72 (0.92)**	**19.60 (0.93)**	**74598 (0.98)**	**70%**	6.99 (1.01)

5.7 Summary

In this chapter, we have studied power optimization by designing high-performance array multipliers. SALRLF multipliers that integrates array splitting, LRLF, and signal flow optimization are proposed as an alternative to tree multipliers for $n \leq 32$. From the basic LRLF structure, two types of further splitting are considered: even/odd and upper/lower. Optimizations of the designs and signal flows in [3:2]-CSAs and [4:2]-CSAs are performed. The final product is obtained with a prefix adder optimized to match the non-uniform arrival profile of the inputs. For fair comparison, tree multipliers are also designed using a special [9:4]-CSAs with $3T_{XOR2}$ delay and signal flow optimization. We find that upper/lower splitting outperforms even/odd splitting after layout although even/odd splitting is a little better in gate delay at the logic level. Automatic layout experiments for $n = 24$ and $n = 32$ indicate that the proposed ULLRLF array multipliers consume $6 \sim 10\%$ less power than tree multipliers while keeping similar area and delay. For $n = 32$, layout experiments with guided floorplanning show that the floorplan of ULLRLF is more regular and easier to control in layout optimization. After guided layout, the delay in both schemes is improved by 4% while ULLRLF array multiplier has 13% less die area and 8% less power than tree multiplier.

CHAPTER 6

Signal Gating in Linear Array Multipliers

6.1 Introduction

Previous work on low-power multipliers often treats multipliers with little consideration of application data characteristics. This is especially true for power optimization techniques at the circuit and logic levels [2][6][86][47][68]. As power dissipation is directly related to data switching patterns, traditional multiplier optimization leads to limited power reduction. Although many application data have large dynamic ranges with frequent short-precision numbers [15][18][115][70], multipliers are designed for the maximum possible precision. Under short-precision input data, portions of a multiplier work only on the sign extensions and produce unnecessary full-precision results. To eliminate unnecessary switching activities, signal gating can be applied to deactivate the sign-extension portion.

In this chapter, we first review previous work on signal gating and its application to low-power multipliers. Then, we generalize and propose a class of signal gating schemes including one-dimensional (1-D) gating and two-dimensional (2-D) gating. These signal gating schemes are then applied to linear array multipliers. The applications to high-performance array and tree multipliers are described in Chapter 7. Multimedia test data traced from a MediaBench program [79] are used to evaluate proposed signal gating schemes because multiplication is a fundamental operation in most multimedia programs and multimedia data often have large dynamic ranges. Signal gating is an external effort for power reduction as it does not change the basic algorithm or architecture of multipliers.

Signal gating has been a general technique to reduce dynamic power in CMOS digital systems. Signal gating achieves this goal by freezing the input of a logic block with gating logic. Gating logic can be latches, registers, AND/OR gates, or tri-state buffers. When a gating signal is active, the outputs of the gated logic block are frozen at previous old values, or set to fixed values, to reduce switching activity. The most common form of signal gating is clock gating at system and architecture levels. In a microprocessor, functional units can be deactivated by disabling their clock signals [57]. In a pipelined CPU datapath, the clocks of some pipeline registers can be disabled if the following functional units are inactive [129]. Even within a pipelined arithmetic unit, clock gating can be applied carefully to disable stages with zero input values [128]. At the gate level, individual flip-flops or gates can be activated or deactivated according to their local behaviors. Lang *et. al.* [76] investigated several gated flip-flop structures in which the clock is disabled when input signal D is the same as output signal Q of previous clock period. Strollo *et. al.* [116] further removed the clock duty-cycle limitation of previous work by gating on both latches in a flip-flop. For arbitrary combinational logic, Tiwari *et. al.* [122] presented a general approach of latch gating called guarded evaluation, which identifies gating candidates by analyzing *observability don't care (ODC)* set. At the bottom circuit level, ancillary gating logic is merged into functional logic cells to achieve efficiency. Dougherty and Thomas [39], for example, presented internal gating techniques which is built into existing library cell adders at the transistor level.

In the above gating techniques, arithmetic blocks are either considered as black boxes at high levels or as random logic at low levels. The structural and computational characteristics of arithmetic logic are neglected. When an arithmetic block is treated as a black box, the power saving is small because the block is powered down only when the whole unit can be inactive. When signal gating is applied at gate and circuit levels, the gating overhead is very large. To achieve more power saving, signal gating on portions of arithmetic units has been proposed recently. Several arithmetic types have been studied, such as sign-detection arithmetic [42][44], accumulator [43], block-serial ALU [24], and adder/incrementor [62]. Here we only consider signal gating in multipliers.

In [17][18], gating on portions of input signals in functional units is considered at the architecture level according to the precisions of operands. There are several limitations in their work. First, they assume that power consumption in multipliers scales linearly with the datapath width. This is generally not true because the area of a multiplier increases quadratically with the width and the switching capacitance is roughly proportional to the area by a high-level estimation [23][67]. Second, there are little considerations of arithmetic details, data switching pattern and gating overheads, which affects the evaluation accuracy. In addition, gating on portions of input signals without internal modification does not produce correct results in multipliers.

In [13], precision-based operation scenarios are formalized and an adaptive power-aware system construction methodology is proposed. In such a system, a set of functional units with different fixed widths is provided and only one of them is enabled according to input data precision. An example of such a system is shown in Figure 6.1. One drawback of this approach is the large area overhead. Moreover, the wire capacitance and signal fanouts are increased significantly.

Besides these theoretical approaches, there is some work on signal-gated multipliers with implementation details. In [70], a linear array multiplier with one

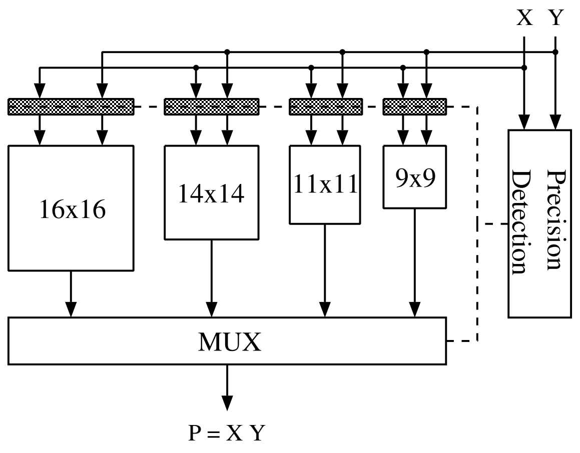
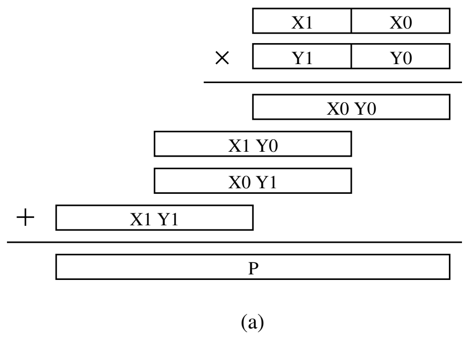


Figure 6.1: The proposed 4-point ensemble multiplier system in [13].

hardwired constant input is partitioned into 4 bit-slices for different data precisions. According to the precision of input data, from 0 up to 4 slices are activated. Such a multiplier is designed for asynchronous matrix-vector multiplication in 2-D DCT/IDCT. In [32], signal gating is applied in two's-complement array multipliers according to the precision of the product. In [50], an array multiplier is partitioned into four equal-size smaller multipliers so that each can be clock-gated separately. This scheme only works on SM or unsigned array multipliers and the partition is not precision-optimized. Figure 6.2 depicts the proposed low-power architecture in [50], which is a straightforward design with much longer delay. In [134], the PPs are generated dynamically in reduced two's-complement number representation according to the actual precision of the input multiplicand data. The sign-extension bits in full-width partial products are changed into constant 0's, which reduces the switching activity in PP reduction step. The Booth recoded PP generators are modified to include the ability of signal gating so that the outputs could be set to 0's. The overhead consists of dynamic precision detection logic and complicated PP generation logic.



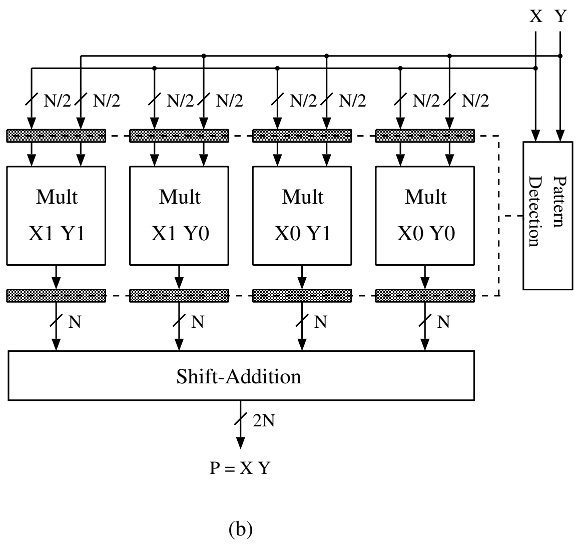


Figure 6.2: The proposed low-power multiplier in [50].

6.2 Signal Gating Schemes

In this section, we generalize various 1-D signal gating algorithms and propose several 2-D gating schemes for low-power multipliers. Which gating scheme should be chosen depends on design objectives and arithmetic features of the target applications. The power reduction depends on the frequency of short-precision data.

6.2.1 General Description

We assume two's-complement multiplication with m-bit multiplicand X, n-bit multiplier Y, and $(m+n)$-bit product P. The width of the gated portion of X is denoted as G_X. Similarly, we define G_Y and G_P. We also define the widths of un-gated portions: $NG_X = m - G_X$, $NG_Y = n - G_Y$, $NG_P = m + n - G_P$. Each w-bit two's-complement number can be partitioned into two parts: sign extension bits (leading zeros or ones), and significant bits including the sign bit. The sign extension part is denoted as E and the significant part as D. W_E and W_D denote the lengths of E and D, respectively. Therefore, $w = W_E + W_D$ and $1 \leq W_D \leq w$.

One of our design goals is to make the signal gating structure transparent to the rest of the system with respect to functionality. This is achieved by providing full-width input/output data interfaces. Inside the multiplier, gating logic is inserted at some predetermined positions to identify portions of the circuit as gating candidates. Whether the candidate portion is actually gated or not depends on runtime data precisions. The general behavior of signal gating is described in Figure 6.3. In this graph, the width(s) of the gating-candidate portion(s) is denoted as some general G. The first step is to detect the width(s), W_E, of sign-extension portion(s) of the variable(s) under gating consideration. If $W_E \geq G$, the gating logic is activated and the multiplier works as a short-precision multiplier. The result is then restored to the full width by sign extension. Otherwise, the multiplier works as a normal full-width multiplier. The final output product is always in full precision of $(m+n)$ bits.

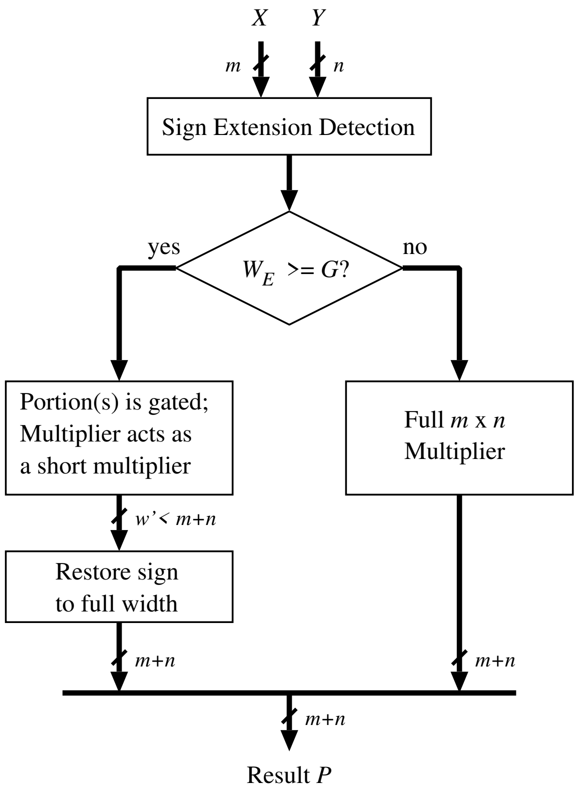


Figure 6.3: General behavior of signal gating schemes.

6.2.2 Static Gating versus Dynamic Gating

The ideal signal gating would dynamically deactivate the corresponding sign-extension portion for each operand with $W_E > 0$. This would be attractive because the circuit adaptively responds to actual precision of operands. However, such a dynamic gating requires gating logic for every bit, which is unacceptable because of the large delay/area/power overhead. Alternatively, gating logic is statically fixed at predetermined positions. When $W_E \geq G$, the candidate portion of width G is gated to be inactive. Algorithms can be developed to decide the optimal gating position based on the data precision statistics.

In static gating with only one fixed gating position, no circuit portion is gated in all cases of $W_E < G$. When $W_E \geq G$, only a portion of width G is gated although more can be gated. Hence, static gating cannot maximize the power reduction when the precisions of input data are distributed evenly over a large range. To take advantage of the adaptive deactivation in dynamic gating, hybrid gating can be developed. In hybrid gating, multiple gating positions are predetermined and multiple gating lines are implemented. According to the actual data precisions, one gating line is selected and the related circuit portion is gated. The overhead increases as more gating lines are added and there is some point that multiple gating lines would bring no power saving or even power increase.

The bit-vectors of most multimedia data can be divided into three regions: MSB region for sign bits, linear region for correlated data bits, and LSB region for uncorrelated random data bits [75][105]. For these data, one gating line located somewhere in the linear region is usually sufficient to achieve good power reduction with small overhead. Therefore, we restrict our discussion to static gating with one gating line per variable in multipliers.

6.2.3 One-Dimensional Signal Gating

1-D gating inserts a single gating line and partitions a circuit module into two parts according to the precision of one variable. In multiplication, there are three variables: multiplicand X, multiplier Y, and product P. Correspondingly, there are three gating choices, namely G-X, G-Y, G-P, as illustrated in Figure 6.4. In these diagrams, the PP bit matrices are shown as parallelograms. The shaded portion in each scheme is inactive when the gating condition holds.

G-X Scheme

In 6.4(a), the left portion corresponding to X is gated when $W_{E_X} \geq G_X$. The

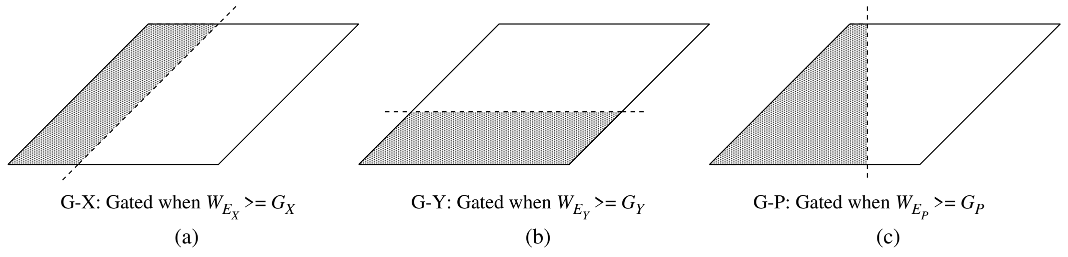

G-X: Gated when W_{E_X} >= G_X
(a)
G-Y: Gated when W_{E_Y} >= G_Y
(b)
G-P: Gated when W_{E_P} >= G_P
(c)

Figure 6.4: 1-D gating diagrams.

high-level algorithm of G-X is described in Figure 6.5. The previous work in [70] belongs to this G-X scheme although it only deals with a specific multiplier with one constant input. The work in [134] is a dynamic G-X gating version as it sets the sign-extension bits in PPs to zeros according to the actual precisions.

```
– step 1: sign extension detection –
$xg = \begin{cases} 0 & \text{if } x_{m-1} = x_{m-2} = \cdots = x_{NG_X-1} \\ 1 & \text{otherwise} \end{cases}$
– step 2: multiplication –
if $xg = 0$ then
    $X^{(NG)} = -x_{NG_X-1}2^{NG_X-1} + \sum_{j=0}^{NG_X-2} x_j 2^j$
    $P^{(NG)} = X^{(NG)} \times Y = \sum_{i=0}^{n} X^{(NG)} y_i 2^i - X^{(NG)} y_{n-1} 2^{n-1}$
    $p_i = p_{NG_X+n-1} \quad i = NG_X + n, \cdots, m+n-1$
else
    $P = X \times Y = \sum_{i=0}^{n-2} X y_i 2^i - X y_{n-1} 2^{n-1}$
end if
```

Figure 6.5: High-level algorithm of G-X.

G-Y Scheme

In 6.4(b), the bottom portion corresponding to Y is gated when $W_{E_Y} \geq G_Y$. The high-level algorithm is described in Figure 6.6. Radix-4 recoding changes sign-extension bits in Y to 0's, which can be viewed as one form of G-Y gating.

```
– step 1: sign extension detection –
$yg = \begin{cases} 0 & \text{if } y_{n-1} = y_{n-2} = \cdots = y_{NG_Y-1} \\ 1 & \text{otherwise} \end{cases}$
– step 2: multiplication –
if $yg = 0$ then
    $Y^{(NG)} = -y_{NG_Y-1}2^{NG_Y-1} + \sum_{i=0}^{NG_Y-2} y_i 2^i$
    $P^{(NG)} = X \times Y^{(NG)} = \sum_{i=0}^{NG_Y-2} Xy_i 2^i - Xy_{NG_Y-1}2^{NG_Y-1}$
    $p_i = p_{m+NG_Y-1} \quad i = m + NG_Y, \cdots, m+n-1$
else
    $P = X \times Y = \sum_{i=0}^{n-2} Xy_i 2^i - Xy_{n-1}2^{n-1}$
end if
```

Figure 6.6: High-level algorithm of G-Y.

G-P Scheme

In 6.4(c), the left portion corresponding to P is gated when $W_{E_P} \geq G_P$. However, it is not easy to detect W_{E_P} because $W_{E_P} = W_{E_X} + W_{E_Y}$ and there are multiple combinations of W_{E_X} and W_{E_Y} for a given W_{E_P}. One way to simplify the detection of W_{E_P} is to set G_X and G_Y separately so that $G_P = G_X + G_Y$ and the gating criteria are $W_{E_X} \geq G_X$ and $W_{E_Y} \geq G_Y$. Again, there are several possibilities of the locations of G-P gating line and there is minor difference in gated PP bits between each location. When $m > NG_P > n$, the high-level algorithm description is shown in Figure 6.7. The G-P scheme was first proposed in [32] as a partially guarded computation technique.

```
– step 1: sign extension detection –
$g = \begin{cases} 0 & \text{if } x_{m-1} = x_{m-2} = \cdots = x_{NG_X-1} \text{ and} \\ & \quad y_{n-1} = y_{n-2} = \cdots = y_{NG_Y-1} \\ 1 & \text{otherwise} \end{cases}$
– step 2: multiplication –
if $g = 0$ then
    $P^{(NG)} = \sum_{i=0}^{n-2} y_i 2^i (\sum_{j=0}^{NG_P-i-1} x_j 2^j) - y_{n-1} 2^{n-1} \sum_{j=0}^{NG_P-n} x_j 2^j$
    $p_i = p_{NG_P-1} \quad i = NG_P, \cdots, m+n-1$
else
    $P = X \times Y = \sum_{i=0}^{n-2} X y_i 2^i - X y_{n-1} 2^{n-1}$
end if
```

Figure 6.7: High-level algorithm of G-P ($m > NG_P > n$).

6.2.4 Two-Dimensional Signal Gating

2-D gating inserts two gating lines for two input variables, X and Y. If the two gating lines are jointly activated, the scheme is denoted as 2-D G-J. If they are separately activated, the scheme is 2-D G-G (the most general case). Figure 6.8 illustrates the 2-D G-J and G-G schemes. In 6.8(a), the shaded portion corresponding to upper bits of X and Y is jointly gated when both $W_{E_X} \geq G_X$ and $W_{E_Y} \geq G_Y$ hold. In 6.8(b), the left portion of X is gated if only $W_{E_X} \geq G_X$ holds; the bottom portion of Y is gated if only $W_{E_Y} \geq G_Y$ holds; all shaded regions are jointly gated if both $W_{E_X} \geq G_X$ and $W_{E_Y} \geq G_Y$ hold.

It is obvious that the 1-D G-X and G-Y schemes are special cases of 2-D gating. The 2-D gating schemes are generally more efficient in power reduction than the 1-D scheme because more regions are gated. Algorithm descriptions of the G-J scheme and the most comprehensive G-G scheme are given in Figure 6.9

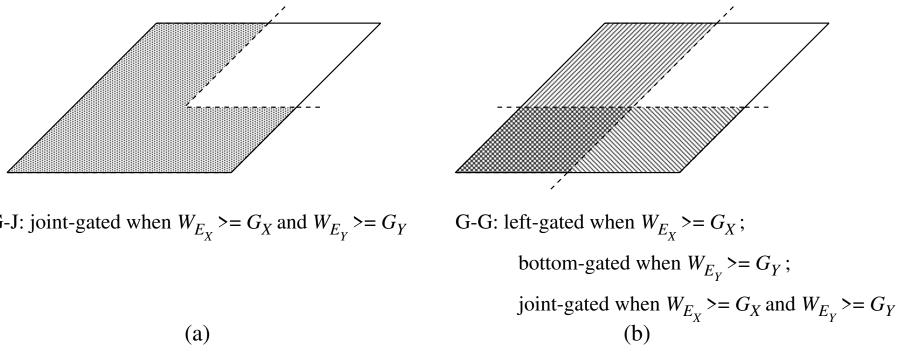

G-J: joint-gated when $W_{E_X} >= G_X$ and $W_{E_Y} >= G_Y$
G-G: left-gated when $W_{E_X} >= G_X$;
bottom-gated when $W_{E_Y} >= G_Y$;
joint-gated when $W_{E_X} >= G_X$ and $W_{E_Y} >= G_Y$
(a)
(b)

Figure 6.8: 2-D gating diagrams.

and 6.10, respectively. With respect to previous work, the scenario-adaptive ensemble system in [13] and the small-multiplier scheme in [50] can be viewed as special architecture-level 2-D gating schemes.

```
– step 1: sign extension detection –
$g = \begin{cases} 0 & \text{if } x_{m-1} = x_{m-2} = \cdots = x_{NG_X-1} \text{ and} \\ & \quad y_{n-1} = y_{n-2} = \cdots = y_{NG_Y-1} \\ 1 & \text{otherwise} \end{cases}$
– step 2: multiplication –
if $g = 0$ then
    $X^{(NG)} = -x_{NG_X-1}2^{NG_X-1} + \sum_{j=0}^{NG_X-2} x_j 2^j$
    $Y^{(NG)} = -y_{NG_Y-1}2^{NG_Y-1} + \sum_{i=0}^{NG_Y-2} y_i 2^i$
    $P^{(NG)} = X^{(NG)} \times Y^{(NG)} = \sum_{i=0}^{NG_Y-2} X^{(NG)} y_i 2^i - X^{(NG)} y_{NG_Y-1} 2^{NG_Y-1}$
    $p_i = p_{NG_X+NG_Y-1} \quad i = NG_X + NG_Y, \cdots, m+n-1$
else
    $P = X \times Y = \sum_{i=0}^{n-2} X y_i 2^i - X y_{n-1} 2^{n-1}$
end if
```

Figure 6.9: High-level algorithm of G-J.

```
– step 1: sign extension detection –
$xg = \begin{cases} 0 & \text{if } x_{m-1} = x_{m-2} = \cdots = x_{NG_X-1} \\ 1 & \text{otherwise} \end{cases}$
$yg = \begin{cases} 0 & \text{if } y_{n-1} = y_{n-2} = \cdots = y_{NG_Y-1} \\ 1 & \text{otherwise} \end{cases}$
– step 2: multiplication –
if $xg = 0$ and $yg = 1$ then
    $X^{(NG)} = -x_{NG_X-1}2^{NG_X-1} + \sum_{j=0}^{NG_X-2} x_j 2^j$
    $P^{(NG)} = X^{(NG)} \times Y = \sum_{i=0}^{n} X^{(NG)} y_i 2^i - X^{(NG)} y_{n-1} 2^{n-1}$
    $p_i = p_{NG_X+n-1} \quad i = NG_X + n, \cdots, m+n-1$
else if $xg = 1$ and $yg = 0$ then
    $Y^{(NG)} = -y_{NG_Y-1}2^{NG_Y-1} + \sum_{i=0}^{NG_Y-2} y_i 2^i$
    $P^{(NG)} = X \times Y^{(NG)} = \sum_{i=0}^{NG_Y-2} X y_i 2^i - X y_{NG_Y-1} 2^{NG_Y-1}$
    $p_i = p_{m+NG_Y-1} \quad i = m + NG_Y, \cdots, m+n-1$
else if $xg = 0$ and $yg = 0$ then
    $X^{(NG)} = -x_{NG_X-1}2^{NG_X-1} + \sum_{j=0}^{NG_X-2} x_j 2^j$
    $Y^{(NG)} = -y_{NG_Y-1}2^{NG_Y-1} + \sum_{i=0}^{NG_Y-2} y_i 2^i$
    $P^{(NG)} = X^{(NG)} \times Y^{(NG)} = \sum_{i=0}^{NG_Y-2} X^{(NG)} y_i 2^i - X^{(NG)} y_{NG_Y-1} 2^{NG_Y-1}$
    $p_i = p_{NG_X+NG_Y-1} \quad i = NG_X + NG_Y, \cdots, m+n-1$
else
    $P = X \times Y = \sum_{i=0}^{n-2} X y_i 2^i - X y_{n-1} 2^{n-1}$
end if
```

Figure 6.10: High-level algorithm of G-G.

6.3 Bit-level Implementation

The gating schemes described above are conceptually simple at the algorithm level. Careful bit-level design is necessary in order to minimize the gating overhead and evaluate realistically the gating effect on power/delay/area. Major considerations include sign-extension detection, modification on gating boundary signals, final CPA sharing and sign extension restoring. Because of the structure difference, linear array multipliers and high-performance multipliers (ULLRLF and tree) are considered separately in this chapter and the next chapter. Input data of a multiplier are stored in two registers. Upon each clock rising edge, new data are loaded into registers and the combinational multiplier works on the loaded data. Signal gating is applied to both input registers and combinational multiplication logic.

6.3.1 Sign-Extension Detection Logic

Sign-extension detection (SED) is the first step in signal gating schemes. For a variable D with the upper d bits as the gating candidate, the upper $(d+1)$ bits are checked in order to detect if the upper d bits are sign extension bits. One extra bit is necessary because the un-gated region needs at least one sign bit for correct two's-complement computation when the d bits are gated.

The detection logic itself simply decides if all input bits are identical, as shown in Figure 6.11. The OR signal of the w bits $(w = d + 1)$ indicates all inputs are 0's whereas the NAND signal indicates all inputs are 1's. Either case would set '0' to the gating control signal g, which is '0'-active. g is used to disable the clock signal of input registers as well as those signals going into the gated combinational portion(s).

To protect the clock signal from glitches and ensure correct timing, g is latched to be gl before controlling the clock and is registered to be gll before controlling the combinational circuit. The timing diagram is illustrated in Figure 6.12. For $gclk$, the potential glitches from g is isolated by the latch when clk is '1' and is

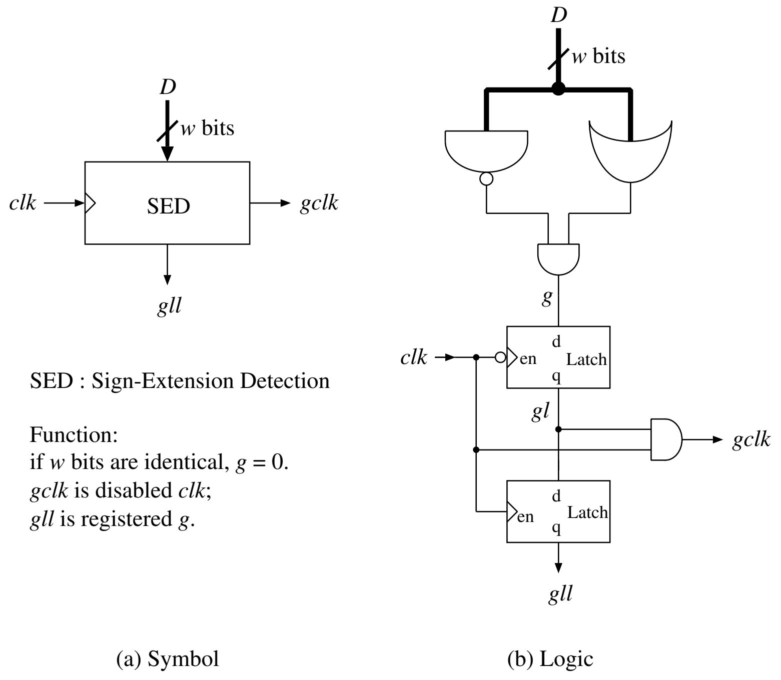


Figure 6.11: Sign-extension detection logic.

eliminated by the AND gate when clk is '0'. Two latches in series with opposite clock controls form a D flip-flop. Therefore, g is registered at clk's rising edge at the same time as data are stored into input registers.

When w is large, SED requires large fan-in AND/OR gates. To reduce the detection time, a tree structure of logic depth $(\log w + 1)$ is built. Using 2-input gates, an 8-bit tree structure is shown in Figure 6.13. In actual implementation, AND/OR gates are replaced by NAND/NOR gates for better delay and area. Gates with more than two inputs may be used to reduce the number of logic levels. However, each level would have larger delay because of the larger gate size. The decision depends on the implementation technology. If the data being checked have a natural skew of arrival times, the skew should be taken into account and a linear structure may help reduce glitches.

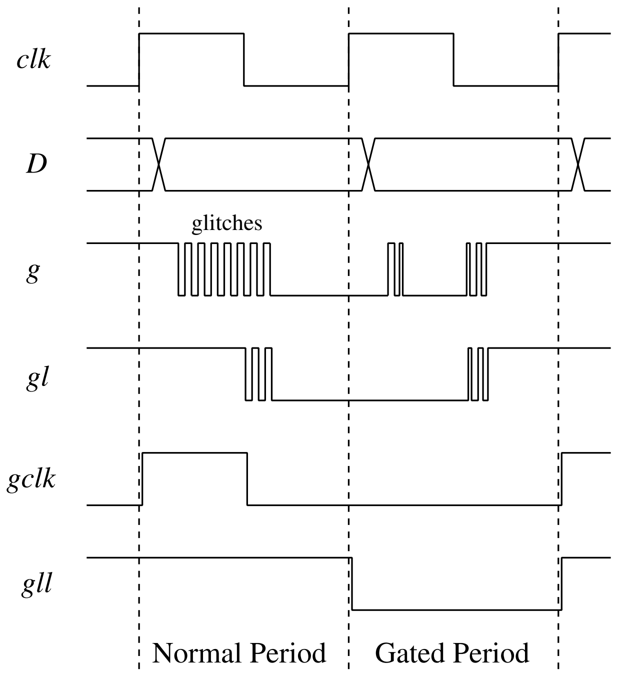


Figure 6.12: Timing diagram of SED with glitch elimination.

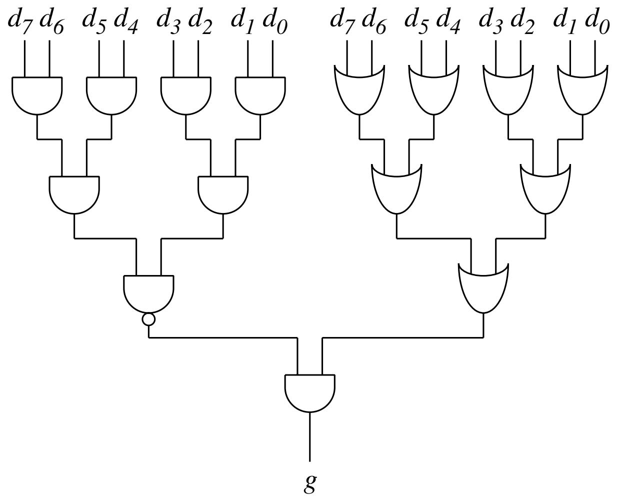

d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0
d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0
g

Figure 6.13: Tree structure of SED.

6.3.2 Signal Gating Logic

Signal gating logic prevents unwanted signal switching activities from propagating forward. There are many ways to implement signal gating logic [130]. The simplest one is to insert a basic gate such as AND2 in the signal propagation path, as shown in Figure 6.14a. For AND, when the control signal *ctl* is set to '0', the output xo becomes '0' and any further switching of input x would have no effect on xo. Note that the change of xo to '0' may trigger switching activities in xo's downstream circuit. Another method is to use a latch in Figure 6.14b as blocking logic. When *ctl* is '1', the latch is transparent to x except for small increase in propagation delay; when *ctl* becomes '0', xo holds the value of x at the time right before *ctl* changes. As xo remains at the old value when x is gated, it prevents signal switching in the downstream circuit. The disadvantage of latch-based gating is the area overhead. Sometimes, a tristate buffer in Figure 6.14c or a transmission gate in Figure 6.14d can be used in place of a latch if the floating node problem can be solved properly [130]. When *ctl* is '0', xo becomes a floating node as all transistors driving the node are turned off. When the node is left alone for a long time, charge leakage can cause the node voltage to float to intermediate values and large short-circuit current may occur. In our study, latches are chosen because the logic depth of gated downstream circuits could be as large as $O(n) + O(m)$ in $m \times n$-bit multipliers and the power cost of resetting signals to '0' by AND2 is often higher.

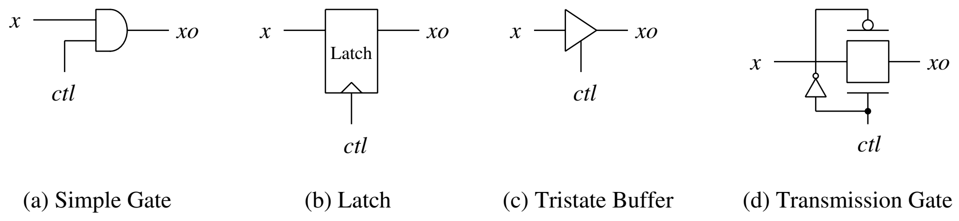


Figure 6.14: Various implementations of gating logic.

6.3.3 Modification in Array Multiplier Core

To minimize gating overhead, it is better to insert gating logic on the bit level. In $m \times n$-bit radix-2 two's-complement multiplication,

$$pp_{i,j} = \begin{cases} y_i x_j & i = 0, \cdots, n-2;\ j = 0, \cdots, m-2 \\ & \text{or } i = n-1;\ j = m-1 \\ (y_i x_j)' & i = 0, \cdots, n-2;\ j = m-1 \\ & \text{or } i = n-1;\ j = 0, \cdots, m-2 \end{cases} \tag{6.1}$$

$$PP_i = \sum_{j=0}^{m-1} pp_{i,j} 2^j \quad i = 0, \cdots, n-1 \tag{6.2}$$

$$P = \sum_{i=0}^{n-1} PP_i 2^i + 2^{n-1} + 2^{m-1} + 2^{m+n-1} \tag{6.3}$$

in which 2^{n-1}, 2^{m-1}, and 2^{m+n-1} correspond to three constant 1's. If $m = n$, $(2^{n-1} + 2^{m-1})$ can be pre-computed as 2^m. FAs with constant inputs are reduced to be half adders or inverters. The MSB bit p_{m+n-1} can be computed separately as $y_{n-1} \oplus x_{m-1}$. In signal gating designs, separate sign computation simplifies the restoring steps of sign extension bits. Figure 6.15 gives an example of a 5×5-bit two's-complement carry-save linear array multiplier. Input data X and Y are stored in two input registers. The new input values are marked with superscript $^{(n)}$. This is the baseline two's-complement array multiplier considered for signal gating.

6.3.3.1 1-D G-X

For 1-D G-X, Figure 6.16 illustrates the modifications in the PP bit matrix when the shaded portion is gated. The number of PP bits in the shaded portion is $(G_X \cdot n)$. All signals entering this region are kept inactive: Y signals are frozen by gating logic, the upper G_X-bit X signals are frozen as the corresponding input

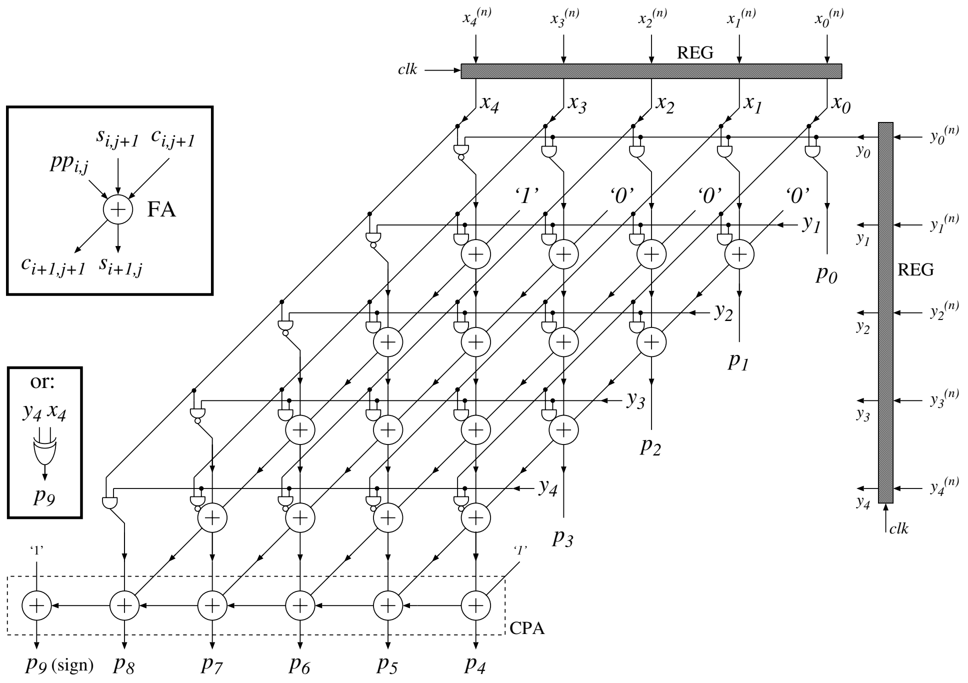


Figure 6.15: A baseline radix-2 two's-complement array multiplier.

registers are clock-gated. Along the gating line, PP bits are inverted to become sign bits when $xgll = 0$

$$pp_{i,j} = \begin{cases} {pp_{i,j}}^{(b)} \odot xgll & i = 0, \cdots, n-1;\ j = NG_X - 1 \\ {pp_{i,j}}^{(b)} & \text{otherwise} \end{cases} \tag{6.4}$$

where $xgll$ is the gating control signal and the signals with superscript $^{(b)}$ are the signals defined in the baseline multiplier. "$\odot$" is equivalence operation XNOR. When $xgll$ is active, the '1' originally in column $(m-1)$ is moved to column $(NG_X - 1)$ and added as $c_{1,NG_X-1} = (xgll)'$. Signals that come from the gated region into un-gated region are set to 0 by $xgll$

$$s_{i,NG_X} = {s_{i,NG_X}}^{(b)} \cdot xgll \quad i = 1, \cdots, n-1 \tag{6.5}$$

A G_X-bit multiplexer called SMUX is used to select between the normal CPA output ${p_i}^{(b)}$ and the sign bits indicated by the dashed line, which can be expressed as

$$p_i = p_i^{(b)} \cdot xgll + p_{m+n-1} \cdot (xgll)' \quad i = NG_X + n - 1, \cdots, m + n - 2 \qquad (6.6)$$

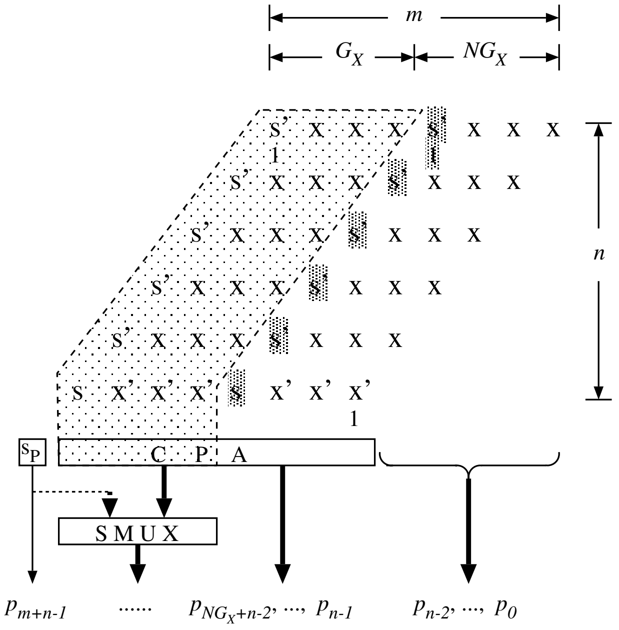


Figure 6.16: The 1-D G-X scheme for two's-complement array multiplier.

Each m-bit PP in Figure 6.16 reduces to be an NG_X-bit PP when gated, which is similar to the reduced two's-complement representation in [134]. The difference is that the leading G_X bits are set to 0's in [134] while the G_X bits here are kept at previous values. Note that when the G_X bits are set to 0's, signal switching activities happen in the shaded portion and also propagate into the un-shaded portion, which consumes unnecessary power.

6.3.3.2 1-D G-Y

Gating on Y can be achieved by exchanging X and Y in the G-X scheme as the multiplicand X and the multiplier Y are exchangeable. However, it is still meaningful to study the G-Y scheme because the length of PP reduction is shorter when gated and it is the basis of 2-D gating. Figure 6.17 describes the modifications in the PP bit matrix when the shaded portion is gated. The number of PP bits in the shaded portion is $(G_Y \cdot m)$. The G-Y scheme is more complicated than the G-X scheme because the gated PPs have to be bypassed. Along the gating line, the PP bits are inverted as follows

$$pp_{i,j} = \begin{cases} pp_{i,j}{}^{(b)} \odot ygll & i = NG_Y - 1;\ j = 0, \cdots, n-1 \\ pp_{i,j}{}^{(b)} & \text{otherwise} \end{cases} \tag{6.7}$$

When $xgll$ is active, the special '1' in column $(n-1)$ is moved to column (NG_Y-1) but still handled in the final CPA. The PP bits along the gating line corresponds to $-Xy_{NG_Y-1}2^{NG_Y-1}$ in Figure 6.6.

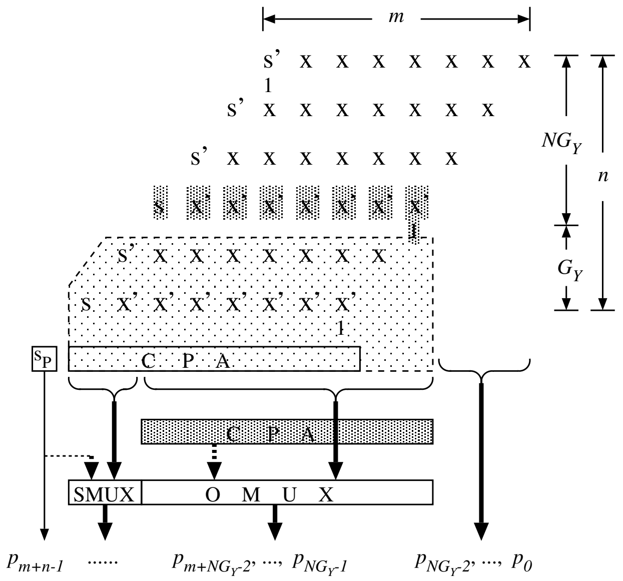


Figure 6.17: The 1-D G-Y scheme for two's-complement array multiplier.

Two final CPAs are provided separately for the un-gated and the gated case. These two CPAs are merged and shared in actual implementations. In the un-gated case, an m-bit OMUX and a G_Y-bit SMUX jointly select the normal product output indicated by the solid lines. In the gated case, OMUX selects dashed signals from the second CPA and SMUX selects sign bits. The operation of SMUX can be expressed as

$$p_i = {p_i}^{(b)} \cdot ygll + p_{m+n-1} \cdot (ygll)' \quad i = m + NG_Y - 1, \cdots, m + n - 2 \tag{6.8}$$

6.3.3.3 1-D G-P

For the 1-D G-P scheme, there is no change in the PP bit matrix. Only logic for blocking signals into the gated region and multiplexers for sign extension are needed, as shown in Figure 6.18. The number of PP bits in the shaded portion depends on the relative values of m, n, G_P. If $m \geq n$, this number can be expressed as

$$Nbit_{shaded} = \begin{cases} \frac{G_P(G_P+1)}{2} & \text{if } G_P \leq n \\ \frac{n(2G_P-n+1)}{2} & \text{if } n < G_P \leq m \\ \frac{n(2m-n+1)}{2} + \frac{(G_P-m)(2n-1-G_P+m)}{2} & \text{if } G_P > m \end{cases} \tag{6.9}$$

If $m < n$, this number is counted as

$$Nbit_{shaded} = \begin{cases} \frac{G_P(G_P+1)}{2} & \text{if } G_P \leq m \\ \frac{m(2G_P-m+1)}{2} & \text{if } m < G_P \leq n \\ \frac{m(2n-m+1)}{2} + \frac{(G_P-n)(2m-1-G_P+n)}{2} & \text{if } G_P > n \end{cases} \tag{6.10}$$

Whether G-P is superior to G-X and G-Y depends on the gating positions and the switching activity in the gated region. A high-level estimation is to compare the number of gated PP bits. For example, if $G_P < n$ and $G_X > n/2$, G-X would gate more PP bits than G-P; if $G_P \geq n$ and $G_X \leq n/2$, G-P would gate more bits. Accurate estimation requires a consideration of gating overhead and data characteristics.

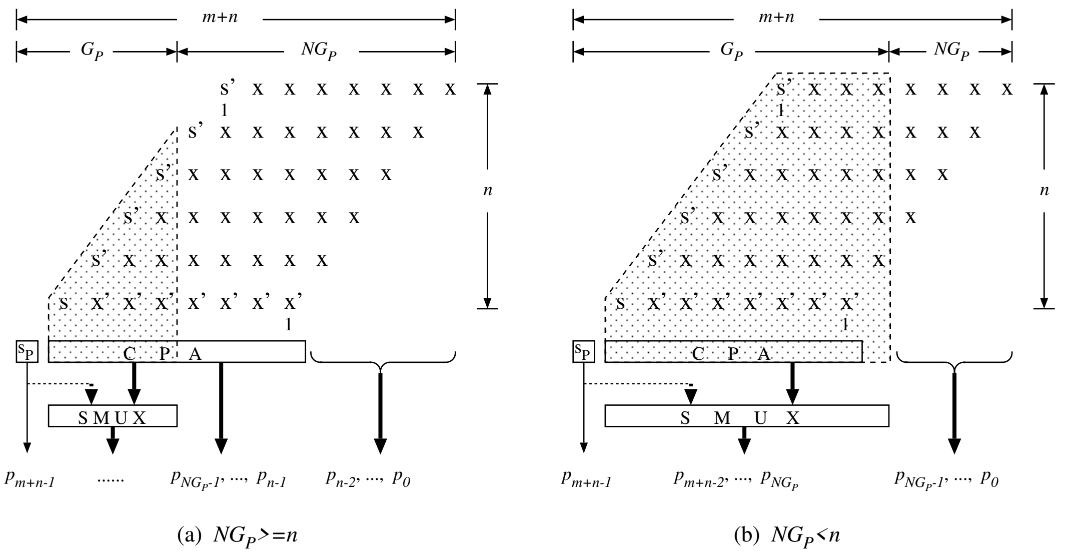


Figure 6.18: The 1-D G-P scheme for two's-complement array multiplier.

6.3.3.4 2-D G-J

In 2-D G-J, the registered gating control signal gll becomes '0' only when both $W_{E_X} \geq G_X$ and $W_{E_Y} \geq G_Y$ hold. Figure 6.19 illustrates major modifications in the bit matrix for the 2-D G-J scheme. The number of PP bits in the shaded portion is $(G_X \cdot n + G_Y \cdot NG_X)$. The multiplier performs $NG_X \times NG_Y$-bit multiplication when gating is active. PP bits are modified as follows

$$pp_{i,j} = \begin{cases} {pp_{i,j}}^{(b)} \odot gll & i = 0, \cdots, NG_Y - 2;\ j = NG_X - 1 \text{ or} \\ & i = NG_Y - 1;\ j = 0, \cdots, NG_X - 2 \\ {pp_{i,j}}^{(b)} & \text{otherwise} \end{cases} \tag{6.11}$$

The special '1' in column $(NG_X - 1)$ is controlled by gll

$$c_{1,NG_X-1} = (gll)'$$

Another '1' in column $(NG_Y - 1)$ is implemented in the CPA. When $NG_X = NG_Y$, these two 1's may be pre-added to be '10' and implemented as

$$c_{1,NG_X} = (gll)'$$

Signals that come from the gated region into un-gated region are also controlled by gll

$$s_{i,NG_X} = s_{i,NG_X}{}^{(b)} \cdot gll \quad i = 1, \cdots, NG_Y - 1 \tag{6.12}$$

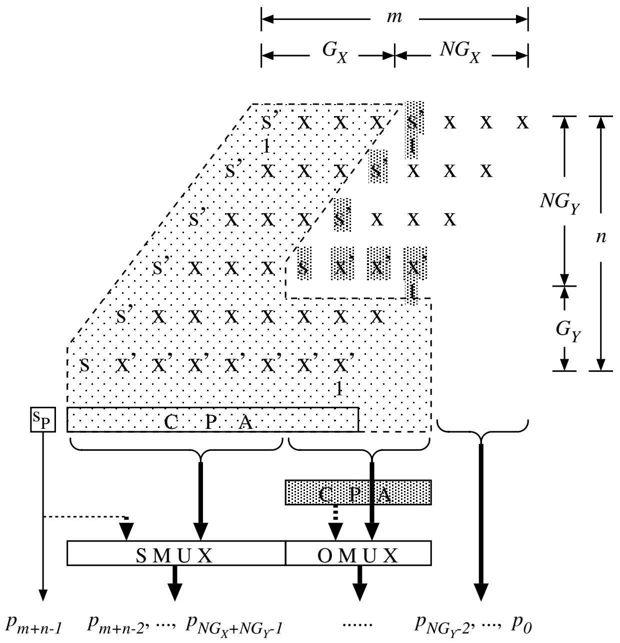


Figure 6.19: The 2-D G-J scheme for two's-complement array multiplier.

Similar to 1-D G-Y, two final CPAs are provided here. In the un-gated case, the NG_X-bit OMUX and the $(G_X + G_Y)$-bit SMUX jointly select the normal product output indicated by the solid lines. In the gated case, OMUX selects dashed signals from the second CPA and SMUX selects sign bits. The operation of SMUX can be expressed as

$$p_i = p_i{}^{(b)} \cdot gll + p_{m+n-1} \cdot (gll)' \quad i = NG_X + NG_Y - 1, \cdots, m + n - 2 \tag{6.13}$$

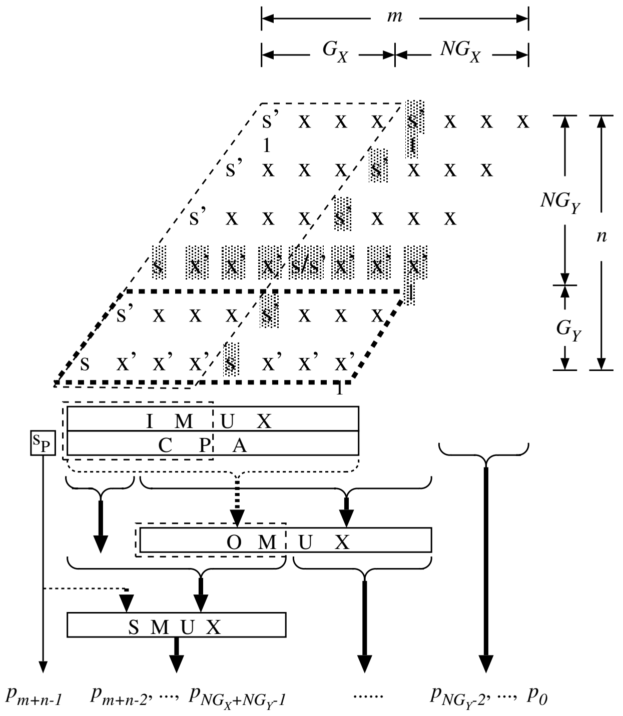
6.3.3.5 2-D G-G

The 2-D G-G scheme can be viewed as the combination of 1-D G-X, 1-D G-Y and 2-D G-J. There are two gating control signals, $xgll$ and $ygll$. In Figure 6.20, the PP bit matrix is partitioned into four portions: top-right (TR), top-left (TL), bottom-right (BR) and bottom-left (BL). When only $xgll$ is active, the left two portions (TL and BL) are gated. When only $ygll$ is active, the bottom two portions (BR and BL) are gated. When both are active, three portions (TL, BL and BR) are gated. TR is always un-gated while BL is gated by either active $xgll$ or active $ygll$. The total number of PP bits being gating candidates is $(G_X \cdot n + G_Y \cdot NG_X)$, the same as in 2-D G-J. The PP bits along the two gating lines are modified as follows

$$pp_{i,j} = \begin{cases} {pp_{i,j}}^{(b)} \odot xgll & i = 0, \cdots, NG_Y - 2;\ j = NG_X - 1 \\ {pp_{i,j}}^{(b)} \odot ygll & i = NG_Y - 1;\ j = 0, \cdots, NG_X - 2 \\ {pp_{i,j}}^{(b)} \odot (xgll \odot ygll) & i = NG_Y - 1;\ j = NG_X - 1 \\ {pp_{i,j}}^{(b)} \odot xgll_y & i = NG_Y, \cdots, N - 1;\ j = NG_X - 1 \\ {pp_{i,j}}^{(b)} \odot ygll_x & i = NG_Y - 1; j = NG_X, \cdots, M - 1 \\ {pp_{i,j}}^{(b)} & \text{otherwise} \end{cases} \tag{6.14}$$

where $xgll_y$ and $ygll_x$ are two special latch-controlled signals. $xgll_y$ is controlled by $ygll$: when $ygll = 1$, $xgll_y$ is essentially $xgll$ propagating transparently through the latch; when $ygll = 0$, $xgll_y$ is frozen at the old value of $xgll$. $ygll_x$ is controlled by $xgll$. These two signals can be replaced by the normal gating signals $xgll$ and $ygll$ without affecting the computation. The reason of using these two special signals is to keep even $xgll$ and $ygll$ inactive in gated regions.

The special '1' in column $(NG_X - 1)$ is controlled by $xgll$ as $c_{1,NG_X-1} = (xgll)'$. The '1' in column $(NG_Y - 1)$ is handled in the final CPA. Signals that come from



m
G_X
NG_X
NG_Y
n
G_Y
IMUX
CPA
OMUX
SMUX
s_P
p_{m+n-1}
$p_{m+n-2}, \ldots, p_{NG_X+NG_Y-1}$
......
$p_{NG_Y-2}, \ldots, p_0$

Figure 6.20: The 2-D G-G scheme for two's-complement array multiplier.

the gated region into the un-gated region are modified as

$$s_{i,NG_X} = \begin{cases} s_{i,NG_X}{}^{(b)} \cdot xgll & i = 1, \cdots, NG_Y - 1 \\ s_{i,NG_X}{}^{(b)} \cdot xgll_y & i = NG_Y, \cdots, N-1 \end{cases} \tag{6.15}$$

Only one final CPA is provided and the sharing is discussed here in detail. This CPA is shared by all computation cases. The inputs of the CPA are selected through the input multiplexer IMUX

$$CPA_{in1} = \begin{cases} (s_{NG_Y,m-1}, s_{NG_Y,m-2}, \cdots, s_{NG_Y,0}) & \text{if} \quad ygll = 0 \\ (s_{n,m-1}, s_{n,m-2}, \cdots, s_{n,0}) & \text{if} \quad ygll = 1 \end{cases} \tag{6.16}$$

$$CPA_{in2} = \begin{cases} (c_{NG_Y,m-1}, c_{NG_Y,m-2}, \cdots, c_{NG_Y,1}, 0) & \text{if} \quad ygll = 0 \\ (c_{n,m-1}, c_{n,m-2}, \cdots, c_{n,1}, 0) & \text{if} \quad ygll = 1 \end{cases} \tag{6.17}$$

If $ygll = 1$, the inputs are the normal sum/carry signals from the last row. If

$ygll = 0$, the inputs come from the $(NG_Y - 1)$-th row. In this case, the outputs of the CPA have to be shifted G_Y-bit right. This shift is implemented in OMUX where the dashed input indicates the data to be shifted. In correspondence with X gating in the array, IMUX, CPA and OMUX are also partitioned and the left portions of them are gated when $xgll = 0$.

Finally, SMUX is used for the restoring operation of sign-extension bits. SMUX handles four sign situations

$$xgll = 1, ygll = 1 \quad : \quad \underbrace{p_{m+n-1}}_{sign\ bit}, \cdots, p_{NG_X+n-1}, p_{NG_X+n-2}, \cdots, p_0 \tag{6.18}$$

$$xgll = 0, ygll = 1 \quad : \quad \underbrace{p_{m+n-1}, \cdots, p_{NG_X+n-1}}_{sign\ bits}, p_{NG_X+n-2}, \cdots, p_0 \tag{6.19}$$

$$xgll = 1, ygll = 0 \quad : \quad \underbrace{p_{m+n-1}, \cdots, p_{m+NG_Y-1}}_{sign\ bits}, p_{m+NG_Y-2}, \cdots, p_0 \tag{6.20}$$

$$xgll = 0, ygll = 0 \quad : \quad \underbrace{p_{m+n-1}, \cdots, p_{NG_X+NG_Y-1}}_{sign\ bits}, p_{NG_X+NG_Y-2}, \cdots, p_0 \tag{6.21}$$

where each bit in a sign region is identical. The MSB bit p_{m+n-1} always indicates the product sign s_P. s_P is thus computed separately in order to simplify the sign extension operation. Denote SMUX's solid-line input signals as ${p_i}^{(b)}$. The SMUX operation is divided into three parts and each part has a different control signal. For $i = max(NG_X + n, m + NG_Y) - 1, \cdots, m + n - 2$,

$$p_i = {p_i}^{(b)} \cdot (xgll \cdot ygll) + s_P \cdot (xgll \cdot ygll)' \tag{6.22}$$

For $i = min(NG_X + n, m + NG_Y) - 1, \cdots, max(NG_X + n, m + NG_Y) - 2$,

$$p_i = \begin{cases} {p_i}^{(b)} \cdot xgll + s_P \cdot (xgll)' & \text{if } NG_X + n < m + NG_Y \\ {p_i}^{(b)} \cdot ygll + s_P \cdot (ygll)' & \text{if } NG_X + n \geq m + NG_Y \end{cases} \tag{6.23}$$

For $i = NG_X + NG_Y - 1, \cdots, min(NG_X + n, m + NG_Y) - 2$,

$$p_i = {p_i}^{(b)} \cdot (xgll + ygll) + s_P \cdot (xgll + ygll)' \tag{6.24}$$

For $i = 0, 1, \cdots, NG_X + NG_Y - 2$, no sign-bit multiplexer is needed.

6.3.3.6 An Example of 2-D G-G in Linear Array Multiplier

Figure 6.20 shows an example implementation of the general gating scheme – 2-D G-G. It is modified from the baseline 5 × 5-bit two's-complement multiplier in Figure 6.15 by inserting two gating lines at $G_X = 2$ and $G_Y = 2$. Each register is partitioned into two parts: one is clocked by the normal *clk*, the other is clocked by the gating-controlled signal *xgclk* or *ygclk*. SED is the sign-extension detection logic discussed in Section 6.3.1.

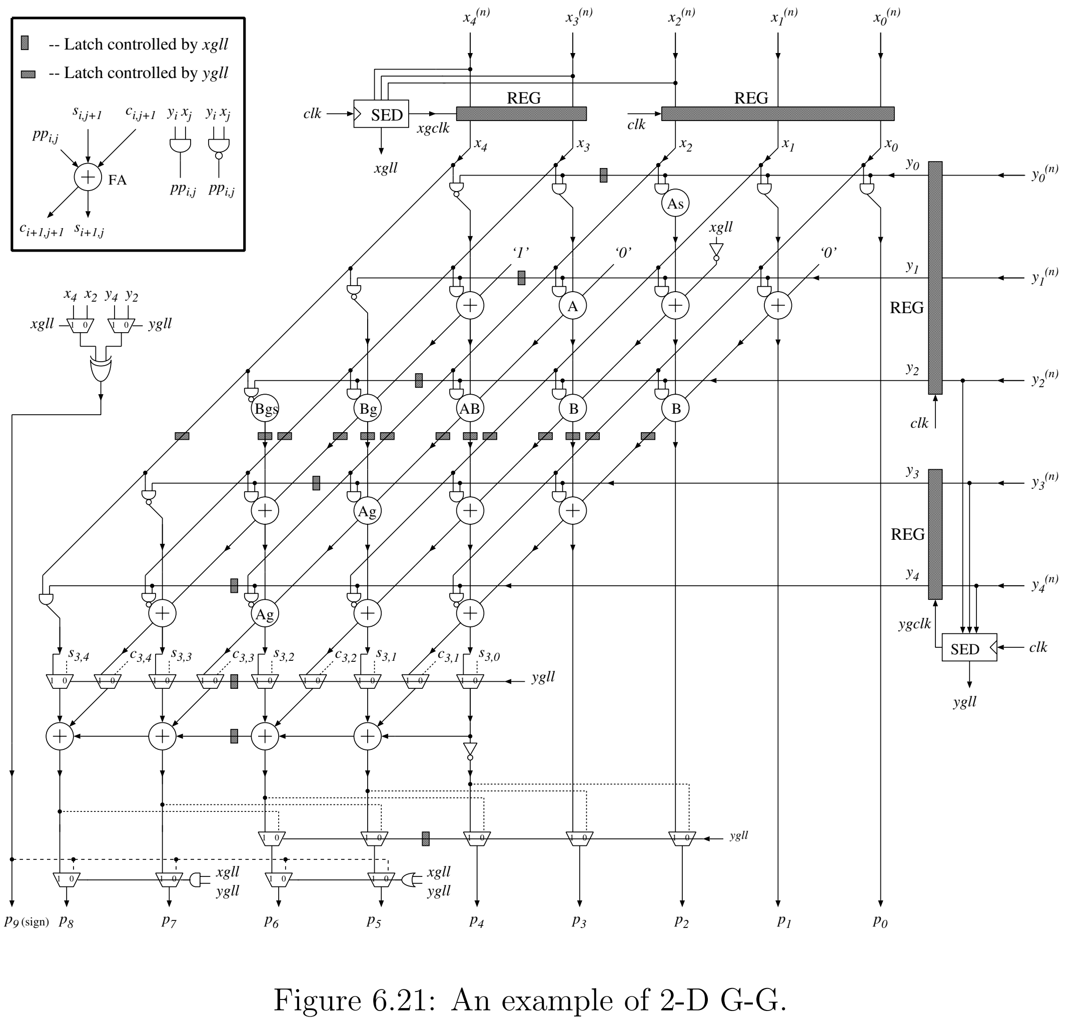


Figure 6.21: An example of 2-D G-G.

In the multiplication core, two gating lines partition the array into four parts.

The horizontal X gating line is controlled by $xgll$ and the Y gating line is controlled by $ygll$. Latches along the X gating line block those Y signals entering the left region. Latches along the Y gating line block those X signals as well as carry/sum signals entering the bottom region. Gating-boundary cells, labeled by letters in the diagram, are specially modified from the baseline cells, as shown in Figure 6.22.

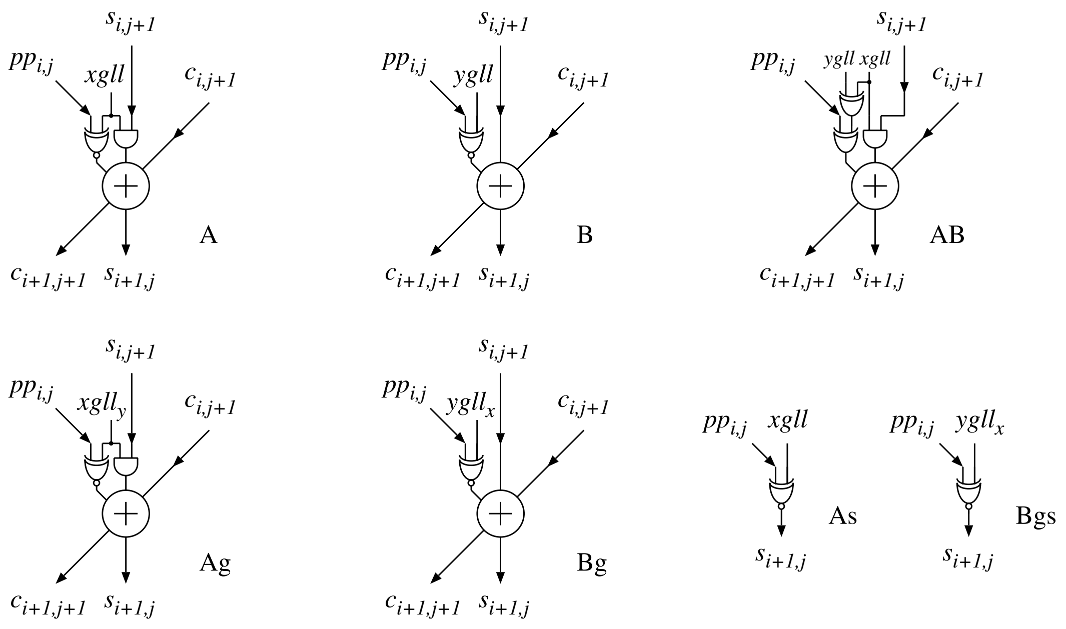


Figure 6.22: Special cells in 2-D G-G.

The final CPA is shared through multiplexed inputs and outputs. When $ygll$ is active, the CPA accepts dashed-line signals from the Y gating boundary and its output is shifted 2-bit right. The CPA itself and its input/output multiplexers are also gated in the higher 2-bit cells when $xgll$ is active. The MSB sign of the product is computed separately. When $xgclk$ is active, x_2 instead of x_4 indicates the sign because x_4 is frozen. When $ygclk$ is active, y_2 instead of y_4 indicates the sign. Because $NG_X + n = m + NG_Y = 8$, only two multiplexer regions with different control signals are needed for the sign restoration described in

Equation 6.22-6.24.

6.4 Experimental Evaluation

The proposed schemes have been implemented and simulated using the design methodology described in Appendix B. In our experiment, the baseline scheme is a 32×32-bit two's-complement carry-save linear array multiplier with two input registers. The test data set with large dynamic range is *djpeg*. *random* is also used in order to test the power overhead. Among three 1-D gating schemes, G-P initially reported in [32] is chosen because it consumes less power than the other two schemes for *djpeg*. For the proposed 2-D gating schemes, the gating lines are inserted at $G_X = 22$ and $G_Y = 16$. The method of gating position selection is also explained in Appendix B. The power results are shown in Table 6.1 and the area/delay results are shown in Table 6.2. The values in parentheses are normalized values. The power consumption is measured at $50MHz$. *CArea* is the total cell area and the routable rate is 85%. T_{SED} is the critical path delay of sign-extension detection logic. T_{core} is the critical path delay in the computation stage including input registers.

Table 6.1: Power comparison of 2-D signal gating schemes

Schemes	Power(mW) ($50MHz$)	
	random	djpeg
r2-array-reg	**58.07 (1.00)**	21.56 (1.00)
1-D G-P	59.57 (1.03)	7.15 (0.33)
2-D G-J	59.23 (1.02)	3.92 (0.18)
2-D G-G	64.43 (1.11)	**3.18 (0.15)**

Table 6.2: Delay/area comparison of 2-D signal gating schemes

Schemes	CArea(μm^2)	Delay(ns)		
		T_{SED}	T_{core}	Total
r2-array-reg	**98325 (1.00)**	0	16.49	**16.49 (1.00)**
1-D G-P	104456 (1.06)	1.22	15.79	17.01 (1.03)
2-D G-J	103085 (1.05)	1.05	16.96	18.04 (1.09)
2-D G-G	110081 (1.12)	0.81	17.23	18.04 (1.09)

It can be seen that signal gating techniques are very efficient in power reduction for input data with a large dynamic range such as *djpeg*. Overall, signal gating has achieved 67-85% power reduction from the baseline multiplier. 1-D G-P reduces the power to 33%. 2-D gating schemes are generally more efficient and further reduce the power to 15-18%. Compared to G-J, G-G consumes 19% less power at the cost of 7% more area. From G-P to G-G, the improvement is more than 50% with only 5% overhead in area. Under random test data, G-P only consumes 2% more power and G-G consumes 11% more power. The delay overhead of 2-D gating schemes are only 6% compared to 1-D G-P and 9% compared to the baseline multiplier.

6.5 Summary

In this chapter, we have proposed 2-D signal gating techniques and generalized a class of signal gating schemes for multipliers with large-dynamic-range input data. This class includes 1-D gating schemes (G-X, G-Y, and G-P) and 2-D gating schemes (G-J and G-G). In 2-D gating, gating lines are provided for both operands. Different regions of the multiplier are dynamically deactivated according to the precision information of each operand. These signal gating schemes are applied into traditional linear array multipliers. The gating overhead has been minimized by careful design of sign-extension logic, gating control logic, gating boundary modification, final CPA sharing and sign-extension restoration. Experiments with large-dynamic-range data show that 2-D G-J achieves 45% power reduction and 2-D G-G achieves 56% power reduction with small overhead in area and delay compared to previous work using 1-D G-P.

CHAPTER 7

Signal Gating in High-Performance Multipliers

7.1 Introduction

All work we are aware of on signal gating for multipliers, including our work in Chapter 6 and previous work in [70][32][50][134], has focused on conventional linear array multipliers. In high-performance applications, tree multipliers are used more frequently. In Chapter 2, we have shown that tree multipliers also consume less power than traditional linear array multipliers although tree multipliers have larger chip area. As signal gating is demonstrated to be very efficient in linear array multipliers with large-dynamic-range data in Chapter 6, we extend it to high-performance multipliers as an additional effort for power saving. Because 2-D G-G is the most power-efficient gating scheme, we only consider 2-D G-G signal gating in this chapter. First, we apply 2-D G-G signal gating in high-performance ULLRLF array multipliers. The gating boundary modification in PP bit matrix and resource sharing to reduce overhead are described in detail. Then, we apply 2-D G-G in tree multipliers. Because of the complex tree structure, gating boundary modification is more complex and each CSA reduction level is considered separately. Finally, experiments with layout are conducted to evaluate the power, delay, and area characteristics of the proposed signal gating schemes.

7.2 Signal Gating in ULLRLF Array Multiplier

The two main parts of bit-level design of signal gating are the PP bit matrix modifications along gating boundaries and resource sharing. In Chapter 5 (Figure 5.1), 1's for sign-extension are placed together in the first row for reduced area and delay. In signal gating, this handling of sign-extension 1's is inconvenient for boundary modification. Therefore, we arrange constant 1's back to the left of each row as in Chapter 4 (Figure 4.2).

7.2.1 Modification of PP Bit Matrix

In ULLRLF, the PP bit matrix has already been partitioned into upper and lower parts. If the optimal gating boundary of Y lies in one of these parts, 2-D G-G in ULLRLF is very similar to that in traditional linear array multipliers (Chapter 6). Here we consider the case where the Y gating boundary is exactly the upper/lower partitioning boundary. The X gating boundary remains flexible and can be inserted at any place in the array. A high-level 2-D gating diagram in ULLRLF is illustrated in Figure 7.1. The PP bit matrix is partitioned into four portions: top-right (TR), top-left (TL), bottom-right (BR) and bottom-left (BL). Different from the 2-D gating for R-L multipliers in Figure 6.8b, the region that is always active is BR rather than TR. The left portion of X is gated if only $W_{E_X} \geq G_X$ holds; the upper portion of Y is gated if only $W_{E_Y} \geq G_Y$ holds; all shaded regions are jointly gated if both $W_{E_X} \geq G_X$ and $W_{E_Y} \geq G_Y$ hold.

Figure 7.2 gives a bit-level diagram of 2-D G-G in $m \times n$-bit ULLRLF with $m = n = 24$ and $G_X = G_Y = 12$. The G-Y gating line goes along the upper/lower partitioning boundary. The correction bit $cor_{n/4-1}$ is included in the active lower region in order to complete smaller $m \times NG_Y$-bit multiplication. To

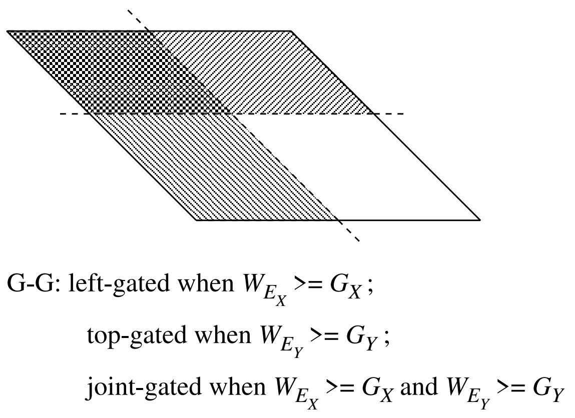

G-G: left-gated when $W_{E_X} >= G_X$;
top-gated when $W_{E_Y} >= G_Y$;
joint-gated when $W_{E_X} >= G_X$ and $W_{E_Y} >= G_Y$

Figure 7.1: High-level diagram of 2-D G-G for ULLRLF.

avoid structure changes in the PPR array, $cor_{n/4-1}$ is postponed to the final [4:2]-CSA instead of being added immediately. No modification in the PP bit matrix is needed for G-Y because of the existing upper/lower partitioning. When $ygll$ is active, all PP bits above the G-Y gating line are gated. The G-X gating line goes through the upper array and lower array in a stair-case way. When $xgll$ is active, all PP bits on the left portion of the G-X gating line are gated.

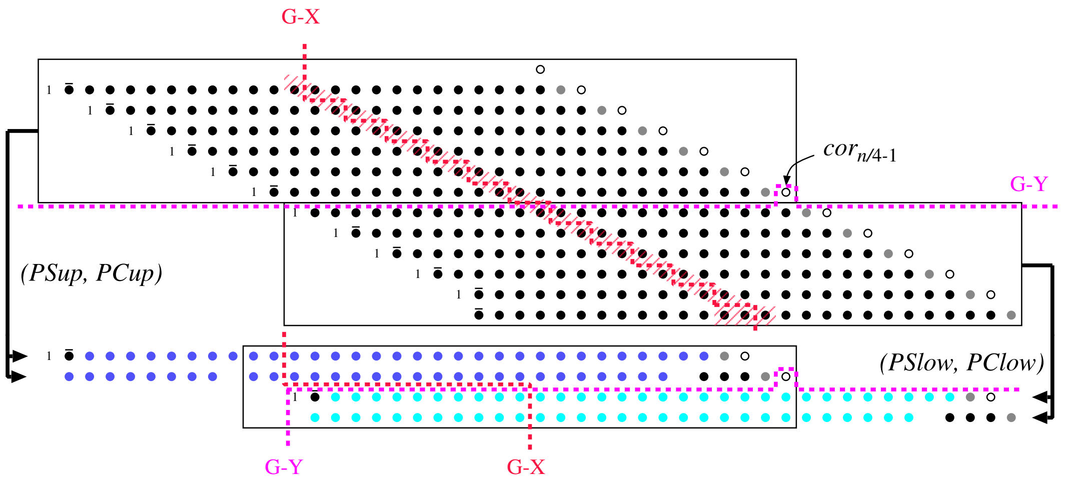


Figure 7.2: Bit-level diagram of 2-D G-G in ULLRLF.

The shaded region of PP bits along the boundary is enlarged in Figure 7.3.

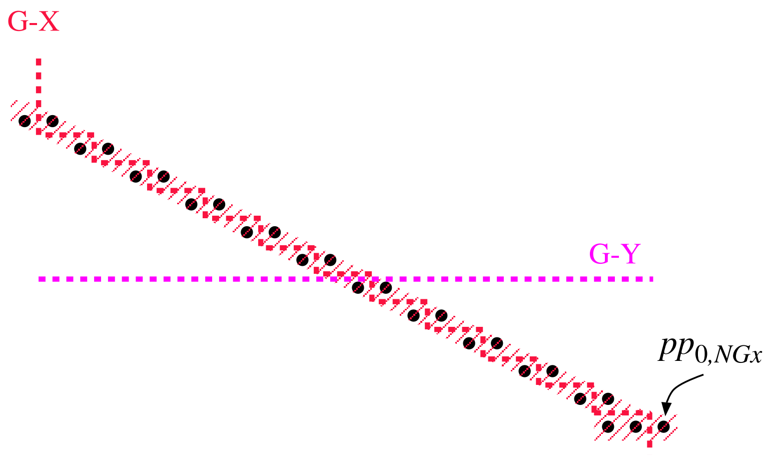


Figure 7.3: The shaded region in Figure 7.2.

The modifications of these bits are as follows. The shaded bits to the right of the G-X gating line are inverted when $xgll = 0$ or $xgll_y = 0$

$$pp_{i,NG_X} = \begin{cases} {pp_{i,NG_X}}^{(b)} \odot xgll & i = 1, \cdots, n/4-1 \\ {pp_{i,NG_X}}^{(b)} \odot xgll_y & i = n/4, \cdots, n/2-1 \end{cases} \tag{7.1}$$

pp_{0,NG_X} is not included as the sign bits in pp_0 are handled differently. Signals with superscript $^{(b)}$ are the original signals in the baseline multiplier. The shaded bits to the left of the line are set to '1' when $xgll = 0$ or $xgll_y = 0$

$$pp_{i,NG_X+1} = \begin{cases} {pp_{i,NG_X+1}}^{(b)} + xgll' & i = 1, \cdots, n/4-1 \\ {pp_{i,NG_X+1}}^{(b)} + xgll'_y & i = n/4, \cdots, n/2-1 \end{cases} \tag{7.2}$$

pp_{0,NG_X+1} is not included. Signal $xgll_y$ is latch-controlled $xgll$: when $ygll = 1$, $xgll_y$ is essentially $xgll$ propagating transparently through the latch; when $ygll = 0$, $xgll_y$ is frozen at the old value of $xgll$. $xgll_y$ is to keep $xgll$ inactive in the gated upper region when G-Y is applied. For the shaded bits in PP_0, the modifications are special because of the signal extension

$$pp_{0,NG_X} = {pp_{0,NG_X}}^{(b)} \tag{7.3}$$

$$pp_{0,NG_X+1} = {pp_{0,NG_X}}^{(b)} \cdot xgll' + {pp_{0,NG_X+1}}^{(b)} \cdot xgll \quad (7.4)$$

$$pp_{0,NG_X+2} = {pp'_{0,NG_X}}^{(b)} \cdot xgll' + {pp_{0,NG_X+2}}^{(b)} \cdot xgll \quad (7.5)$$

The resulting bit matrix corresponds to smaller $NG_X \times n$-bit multiplication. When G-X and G-Y are jointly active, the bit matrix corresponds to smaller $NG_X \times NG_Y$-bit multiplication in the bottom-right region.

7.2.2 Resource Sharing

In LRLF for each half PPs, an (n/2-1)-bit [4:2]-CSA is used to add the left-side vectors from leapfrog PPR and an (m-3)-bit [3:2]-CSA is used to add the right-side vectors. In G-X, these extra CSAs (eCSAs) need to be shared between gating and non-gating. To simplify the sharing, the left-side vectors are precomputed so that only [3:2]-CSAs are used. The modified LRLF for upper half PPs is shown in Figure 7.4. A special cell, FAP1, is introduced to compute $(A + B + C + 1)$, as illustrated in Figure 7.5. The logic expressions of FAP1 are

$$Stmp = A \oplus B \oplus C \quad (7.6)$$

$$Cout1 = A \cdot B + B \cdot C + A \cdot C \quad (7.7)$$

$$Cout2 = Stmp \quad (7.8)$$

$$Sum = Stmp' \quad (7.9)$$

The inverter in FAP1 is shown as a small circle in Figure 7.4. Now there are at most three vectors from PPR and only one type of CSA is used.

In G-X, the adder cells along the gating boundary are modified to execute smaller multiplication, as shown in Figure 7.6. The 1's in Equation 7.2 during G-X are implemented by modifying FAs corresponding to ${PP_{i,NG_X+1}}^{(b)}$ to be FAP1's. For two left-most adders each row (except the first row) in the right region, Pin A's select inputs between normal PP bits and partial sum (PS) bits

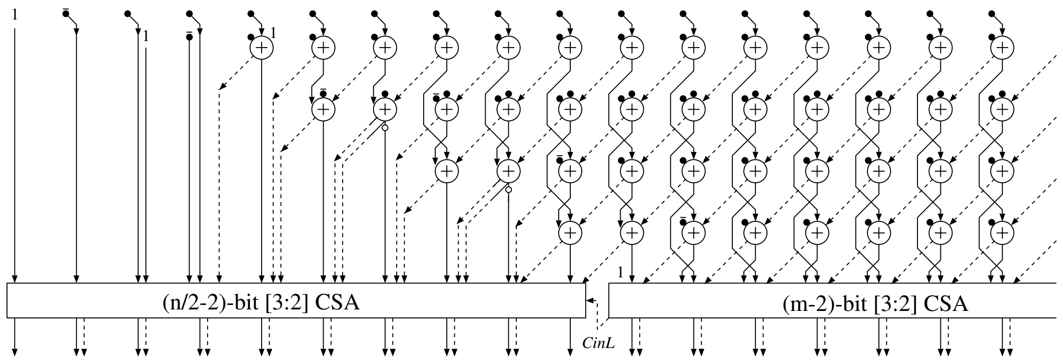


Figure 7.4: LRLF for upper PPs with only [3:2]-CSAs.

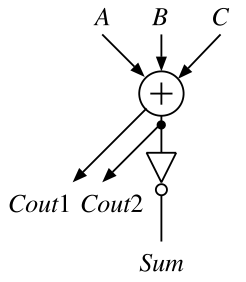


Figure 7.5: Cell FAP1 for $(A + B + C + 1)$.

from the previous row

$$A_{i,NG_X+1} \quad = \quad xgll \cdot PP_{i,NG_X+1}{}^{(b)} + xgll' \cdot PS_{i-1,NG_X+1} \qquad (7.10)$$

$$A_{i,NG_X+2} \quad = \quad xgll \cdot PP_{i,NG_X+2}{}^{(b)} + xgll' \cdot PS_{i-1,NG_X+2} \qquad (7.11)$$

All vectors reaching the dashed thick line are fed into eCSA and the original paths entering the left portion are frozen by latches. Multiplexer IMUX selects the inputs of eCSA between the vectors from normal PPR and the vectors from the G-X boundary.

Assume that the carry-save vectors from the upper PPR portion are $PSup_i$ and $PCup_i$ ($i = 0, \cdots, m + n - 1$) and the vectors from the lower PPR portion are $PSlow_i$ and $PClow_i$ ($i = 0, \cdots, m + n - 1$). Because of shifting,

$$PSup_i \quad = \quad PCup_i = 0 \qquad i = 0, \cdots, n/2 - 2 \qquad (7.12)$$

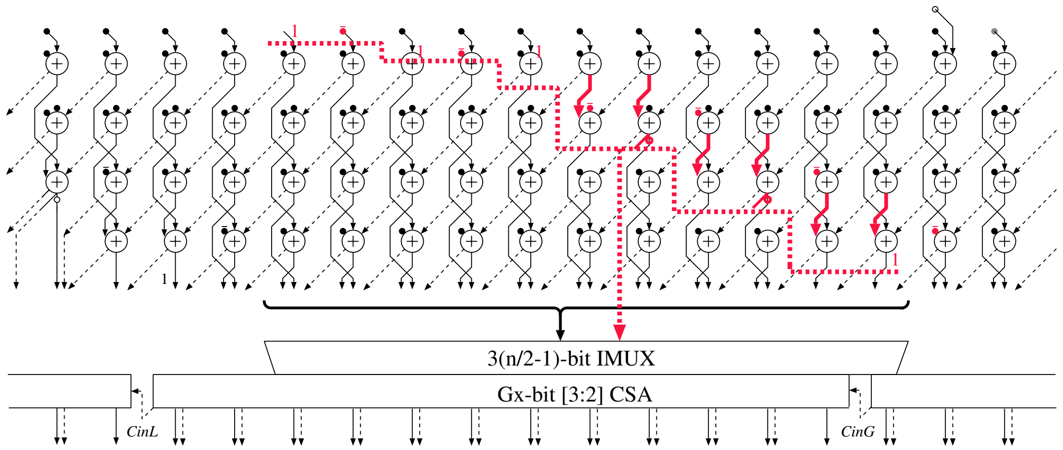


Figure 7.6: Signal gating in LRLF for upper PPs.

$$PSlow_i \quad = \quad PClow_i = 0 \qquad i = m + n/2, \cdots, m + n - 1 \tag{7.13}$$

An $(m + 3)$-bit [4:2]-CSA adds these four vectors. As shown in Figure 7.2, this [4:2]-CSA is shared in four situations: G-X, G-Y, G-J, and no gating. When $xgll$ is active, the following bits of $PSlow_i$ and $PClow_i$ on the left of the G-X line in the [4:2]-CSA are set to 0

$$PSlow_i \quad = \quad PSlow_i^{(b)} \cdot xgll \qquad i = n/2 + NG_X, \cdots, n + NG_X \tag{7.14}$$

$$PClow_i \quad = \quad PClow_i^{(b)} \cdot xgll \qquad i = n/2 + NG_X, \cdots, n + NG_X \tag{7.15}$$

No action is taken on $PSup_i$ and $PCup_i$ with $i = n + NG_x + 1, m + n/2 + 2$ because these bits are out of the product width in G-X. When $ygll$ is active, the following bits of $PSup_i$ and $PCup_i$ in the [4:2]-CSA are set to 0

$$PSup_i \quad = \quad PSup_i^{(b)} \cdot ygll \qquad i = NG_Y + 1, \cdots, m + NG_Y \tag{7.16}$$

$$PCup_i \quad = \quad PCup_i^{(b)} \cdot ygll \qquad i = NG_Y + 1, \cdots, m + NG_Y \tag{7.17}$$

The sharing of final CPA is the same as in 2-D G-G gating for linear array multipliers. Here we do not repeat the description in Section 6.3.3.5 of Chapter 6.

7.3 Signal Gating in Tree Multiplier

Tree structures are quite different from array structures. It is difficult to create a parameterized description for tree multiplier design. Instead, an individual description is used for each multiplier with a different Y operand width. On the physical design level, it is very complex to layout a tree in a rectangular floor-planning. For signal gating, CSAs at each level of a tree require separate considerations. For these reasons, we only consider 2-D G-G for a specific $m \times n$-bit ($m = n = 32$) tree multiplier optimized with [9:4]-CSAs proposed in Chapter 5. To simplify description, $G_Y = n/2$ as in the case of ULLRLF. G_X is flexible between 1 to m. Figure 7.7 gives a high-level diagram of 2-D G-G in this tree multiplier. For G-Y, only one gating line is inserted. For G-X, gating lines are inserted in CSAs on all levels.

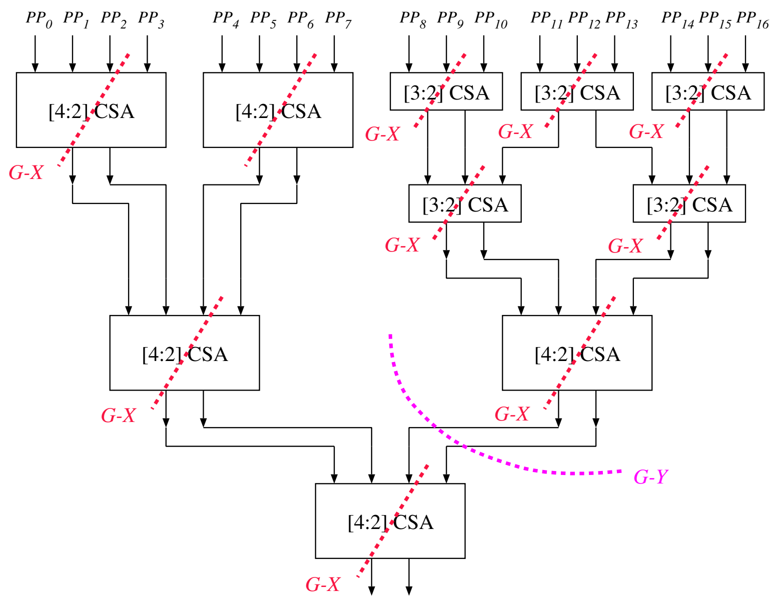

PP_0 PP_1 PP_2 PP_3
PP_4 PP_5 PP_6 PP_7
PP_8 PP_9 PP_{10}
PP_{11} PP_{12} PP_{13}
PP_{14} PP_{15} PP_{16}
[4:2] CSA
[3:2] CSA
G-X
G-Y

Figure 7.7: 2-D G-G in a tree multiplier.

To facilitate the design of gating boundaries, constant 1's are also placed on the left of each row as in ULLRLF signal gating. The bit-level diagram of the tree multiplier in Figure 7.7 is shown in Figure 7.8. Each box corresponds to a CSA and there are four levels of PP reduction. The G-Y gating line partitions the PPs into group $(PP_0, \cdots, PP_7)$ and group $(PP_8, \cdots, PP_{16})$ (PP_{16} only has one bit cor_{15}). The correction bit cor_7 is included in the top region to complete smaller $m \times NG_Y$-bit multiplication. When $ygll$ is active, all PP bits in the bottom region are gated. The G-X gating line goes through all CSAs in a staircase way and $G_X = 16$. When $xgll$ is active, all PP bits on the left portion of the G-X gating line are gated. The shaded region of PP bits along the boundary is enlarged in Figure 7.9. These bits are modified in exact the same way as in ULLRLF. These modifications are incorporated into actual designs by changing the inputs of CSAs.

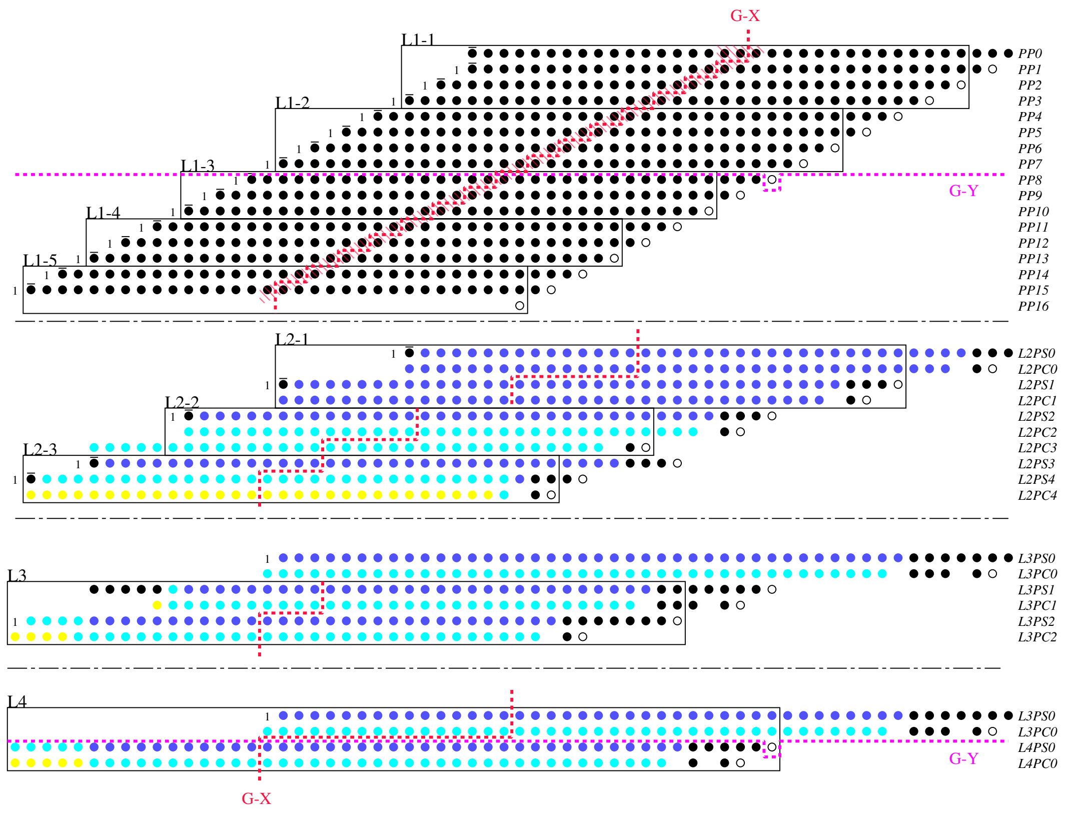


Figure 7.8: Bit-level diagram of 2-D G-G in a tree multiplier.

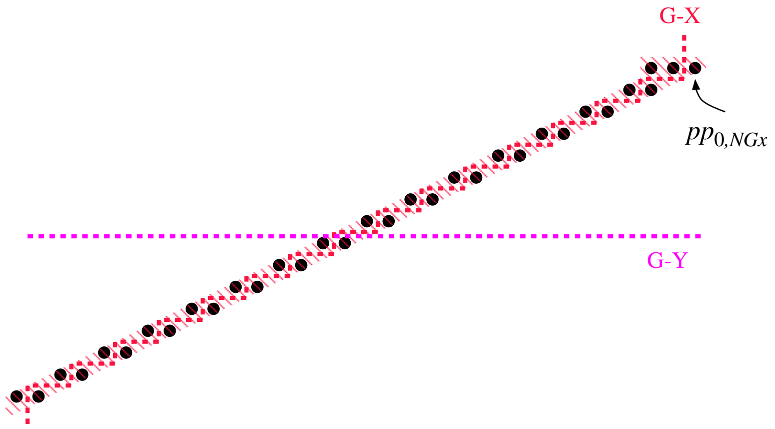


Figure 7.9: The shade region in Figure 7.8.

In L1-1, the inputs of 7-bit [4:2]-CSAs across the G-X line are modified

$$A_{NG_X+1} = {pp_{0,NG_X}}^{(b)} \cdot xgll' + {pp_{0,NG_X+1}}^{(b)} \cdot xgll \tag{7.18}$$

$$A_{NG_X+2} = {pp'_{0,NG_X}}^{(b)} \cdot xgll' + {pp_{0,NG_X+2}}^{(b)} \cdot xgll \tag{7.19}$$

$$A_i = {pp_{0,i}}^{(b)} \cdot xgll \quad i = NG_Y + 3, \cdots, NG_Y + 7 \tag{7.20}$$

$$B_{NG_X+2} = {pp'_{1,NG_X}}^{(b)} \cdot xgll' + {pp_{1,NG_X}}^{(b)} \cdot xgll \tag{7.21}$$

$$B_{NG_X+3} = {pp_{1,NG_X+1}}^{(b)} + xgll' \tag{7.22}$$

$$B_i = {pp_{1,i}}^{(b)} \cdot xgll \quad i = NG_Y + 4, \cdots, NG_Y + 7 \tag{7.23}$$

$$C_{NG_X+4} = {pp'_{2,NG_X}}^{(b)} \cdot xgll' + {pp_{2,NG_X}}^{(b)} \cdot xgll \tag{7.24}$$

$$C_{NG_X+5} = {pp_{2,NG_X+1}}^{(b)} + xgll' \tag{7.25}$$

$$C_i = {pp_{2,i}}^{(b)} \cdot xgll \quad i = NG_Y + 6, NG_Y + 7 \tag{7.26}$$

$$D_{NG_X+6} = {pp'_{3,NG_X}}^{(b)} \cdot xgll' + {pp_{3,NG_X}}^{(b)} \cdot xgll \tag{7.27}$$

$$D_{NG_X+7} = {pp_{3,NG_X+1}}^{(b)} + xgll' \tag{7.28}$$

where the indices of A, B, C, D are numbered according to the positions in full $(m + n)$-bit width. When $xgll$ is active, these inputs are set to constant 1's or inverted if they are PP bits requiring modification, or set to 0 to prevent gated

PP bits from affecting the addition. In L1-2, the inputs of 8-bit [4:2]-CSAs across the G-X line are modified

$$A_{NG_X+8} = {pp'_{4,NG_X}}^{(b)} \cdot xgll' + {pp_{4,NG_X}}^{(b)} \cdot xgll \tag{7.29}$$

$$A_{NG_X+9} = {pp_{4,NG_X+1}}^{(b)} + xgll' \tag{7.30}$$

$$A_i = {pp_{4,i}}^{(b)} \cdot xgll \quad i = NG_Y + 10, \cdots, NG_Y + 15 \tag{7.31}$$

$$B_{NG_X+10} = {pp'_{5,NG_X}}^{(b)} \cdot xgll' + {pp_{5,NG_X}}^{(b)} \cdot xgll \tag{7.32}$$

$$B_{NG_X+11} = {pp_{5,NG_X+1}}^{(b)} + xgll' \tag{7.33}$$

$$B_i = {pp_{5,i}}^{(b)} \cdot xgll \quad i = NG_Y + 12, \cdots, NG_Y + 15 \tag{7.34}$$

$$C_{NG_X+12} = {pp'_{6,NG_X}}^{(b)} \cdot xgll' + {pp_{6,NG_X}}^{(b)} \cdot xgll \tag{7.35}$$

$$C_{NG_X+13} = {pp_{6,NG_X+1}}^{(b)} + xgll' \tag{7.36}$$

$$C_i = {pp_{6,i}}^{(b)} \cdot xgll \quad i = NG_Y + 14, NG_Y + 15 \tag{7.37}$$

$$D_{NG_X+14} = {pp'_{7,NG_X}}^{(b)} \cdot xgll' + {pp_{7,NG_X}}^{(b)} \cdot xgll \tag{7.38}$$

$$D_{NG_X+15} = {pp_{7,NG_X+1}}^{(b)} + xgll' \tag{7.39}$$

In L1-3, the inputs of 6-bit [3:2]-CSAs across the G-X line are modified

$$A_{NG_X+16} = {pp'_{8,NG_X}}^{(b)} \cdot xgll'_y + {pp_{8,NG_X}}^{(b)} \cdot xgll_y \tag{7.40}$$

$$A_{NG_X+17} = {pp_{8,NG_X+1}}^{(b)} + xgll'_y \tag{7.41}$$

$$A_i = {pp_{8,i}}^{(b)} \cdot xgll \quad i = NG_Y + 18, \cdots, NG_Y + 21 \tag{7.42}$$

$$B_{NG_X+18} = {pp'_{9,NG_X}}^{(b)} \cdot xgll'_y + {pp_{9,NG_X}}^{(b)} \cdot xgll_y \tag{7.43}$$

$$B_{NG_X+19} = {pp_{9,NG_X+1}}^{(b)} + xgll'_y \tag{7.44}$$

$$B_i = {pp_{9,i}}^{(b)} \cdot xgll \quad i = NG_Y + 20, NG_Y + 21 \tag{7.45}$$

$$C_{NG_X+20} = {pp'_{10,NG_X}}^{(b)} \cdot xgll'_y + {pp_{10,NG_X}}^{(b)} \cdot xgll_y \tag{7.46}$$

$$C_{NG_X+21} = {pp_{10,NG_X+1}}^{(b)} + xgll'_y \tag{7.47}$$

The same modifications (with different bits) happen in the inputs of 6-bit [3:2]-CSAs in L1-4. In L1-5, the inputs of 4-bit [2:2]-CSAs across the G-X line are modified

$$A_{NG_X+28} = {pp'_{14,NG_X}}^{(b)} \cdot xgll'_y + {pp_{14,NG_X}}^{(b)} \cdot xgll_y \tag{7.48}$$

$$A_{NG_X+29} = {pp_{14,NG_X+1}}^{(b)} + xgll'_y \tag{7.49}$$

$$A_i = {pp_{14,i}}^{(b)} \cdot xgll \quad i = NG_Y + 30, NG_Y + 31 \tag{7.50}$$

$$B_{NG_X+30} = {pp'_{15,NG_X}}^{(b)} \cdot xgll'_y + {pp_{15,NG_X}}^{(b)} \cdot xgll_y \tag{7.51}$$

$$B_{NG_X+31} = {pp_{15,NG_X+1}}^{(b)} + xgll'_y \tag{7.52}$$

$xgll_y$ is used in L1-3, L1-4, and L1-5 because these CSAs are also in the G-Y region.

From the second level, CSAs accept partial sums and carries (PS's and PC's) from the previous level. Because of the gating in the first level, gating lines are inserted in subsequent levels and CSAs across the boundaries are also modified. In L2-1, the inputs of 8-bit [4:2]-CSAs across the G-X line are set to 0 when $xgll$ is active

$$A_i = {L2PS_{0,i}}^{(b)} \cdot xgll \quad i = NG_X + 8, \cdots, NG_X + 15 \tag{7.53}$$

$$B_i = {L2PC_{0,i}}^{(b)} \cdot xgll \quad i = NG_X + 8, \cdots, NG_X + 15 \tag{7.54}$$

In L2-2, the inputs of 6-bit [3:2]-CSAs across the G-X line are set to 0 when $xgll_y$ is active

$$A_i = {L2PS_{2,i}}^{(b)} \cdot xgll_y \quad i = NG_X + 22, \cdots, NG_X + 27 \tag{7.55}$$

$$B_i = {L2PC_{2,i}}^{(b)} \cdot xgll_y \quad i = NG_X + 22, \cdots, NG_X + 27 \tag{7.56}$$

In L2-3, the inputs of 4-bit [3:2]-CSAs across the G-X line are set to 0 when $xgll_y$ is active

$$A_i = {L2PS_{2,i}}^{(b)} \cdot xgll_y \quad i = NG_X + 28, \cdots, NG_X + 31 \tag{7.57}$$

$$B_i = {L2PC_{2,i}}^{(b)} \cdot xgll_y \quad i = NG_X + 28, \cdots, NG_X + 31 \tag{7.58}$$

In L3, the inputs of 4-bit [3:2]-CSAs across the G-X line are set to 0 when $xgll_y$ is active

$$\begin{aligned} A_i &= L3PS_{1,i}{}^{(b)} \cdot xgll_y \quad i = NG_X + 28, \cdots, NG_X + 31 & (7.59) \\ B_i &= L3PC_{1,i}{}^{(b)} \cdot xgll_y \quad i = NG_X + 28, \cdots, NG_X + 31 & (7.60) \end{aligned}$$

The final [4:2]-CSA L4 is shared in four situations: G-X, G-Y, G-J, and no gating. When $xgll$ is active, the input bits on the left of G-X line are set to 0

$$\begin{aligned} A_i &= L3PS_{0,i}{}^{(b)} \cdot xgll \quad i = NG_X + 16, \cdots, NG_X + 31 & (7.61) \\ B_i &= L3PC_{0,i}{}^{(b)} \cdot xgll \quad i = NG_X + 16, \cdots, NG_X + 31 & (7.62) \end{aligned}$$

No action is taken on the bits $i = NG_X + 32, \cdots, m + n$ because these bits are out of the product width in G-X. When $ygll$ is active, the input bits below the G-Y line are modified

$$\begin{aligned} C_i &= L4PS_{0,i}{}^{(b)} \cdot xgll \quad i = NG_Y, \cdots, m + n & (7.63) \\ D_i &= L4PC_{0,i}{}^{(b)} \cdot xgll \quad i = NG_Y, \cdots, m + n & (7.64) \end{aligned}$$

7.4 Experimental Evaluation

The 2-D G-G signal gating schemes for high-performance ULLRLF and tree multipliers have been implemented and evaluated using the methodology described in Appendix B. In our experiments, the baseline schemes are 32×32-bit ULLRLF and tree multipliers with input registers. Because of the difference in handling constant 1's, these multipliers are named with "*new*". The test data set is *djpeg* having a large number of short-precision data. *random* is also used in order to test the power overhead. The gating lines are inserted at $G_X = 22$ and $G_Y = 16$, as explained in Appendix B. The power/delay/area comparison results after automatic layout are shown in Table 7.1 and Table 7.2. The values in parentheses are normalized values. The power consumption is measured at 100 MHz. *CArea* is the total cell area and the routable rate is 70%. T_{SED} is the critical path delay of sign-extension detection logic. T_{core} is the delay in the computation stage including input registers.

Table 7.1: Power comparison of 2-D G-G high-performance multipliers

Schemes	Power(mW) (100MHz)	
	random	djpeg
newUL-reg	**46.94 (1.00)**	22.52 (1.00)
newUL-GG	57.65 (1.23)	**7.90 (0.35)**
newtree-reg	52.49 (1.00)	26.06 (1.00)
newtree_GG	62.09 (1.08)	9.05 (0.35)

Table 7.2: Delay/area comparison of 2-D G-G high-performance multipliers

Schemes	CArea(μm^2)	Delay(ns)		
		T_{SED}	T_{core}	Total
newUL-reg	**80126 (1.00)**	0	7.49	7.49 (1.00)
newUL-GG	90874 (1.13)	0.79	8.06	8.85 (1.18)
newtree-reg	82358 (1.00)	0	6.98	**6.98 (1.00)**
newtree_GG	89780 (1.09)	0.74	7.25	7.99 (1.15)

Under *djpeg* test data, 2-D G-G reduces the power in newUL-reg and newtree-reg by 65%. Between newUL-GG and newtree-GG, newUL-GG consumes 13% less power under *djpeg*. Under *random*, 2-D G-G increases the power in newUL-reg by 23% and the power in newtree-reg by 8%. Although the power increase percentage in newUL-GG is larger under *random*, the power in newUL-GG is still

7% less than that in newtree-GG.

In newUL, 2-D G-G causes 13% area overhead and 18% delay overhead. In newtree, 2-D G-G causes 9% area overhead and 15% delay overhead. The larger area overhead in newUL is because the signals entering the left region in the array are latched while there are no such latches in the tree. The larger delay overhead in newUL comes from IMUX for sharing extra [3:2]-CSA (Figure 7.4). Despite of the larger area/delay overhead, newUL-GG consumes less power than newtree-GG because of the structure advantage.

Because constant 1's are put on the left of each PP row and more gates are required, the area of newUL is slightly larger than that of the original ULLRLF. The delay is also slightly larger because there are a few more latest-arriving bits from the middle of PPR. Thus, newtree-reg has 7% less delay than newUL-reg while the delays of tree and ULLRLF are very close in Chapter 5.

7.5 Summary

In this chapter, we have proposed, implemented, and evaluated 2-D G-G signal gating schemes for high-performance array and tree multipliers. In ULLRLF array multipliers, G-Y gating line follows the boundary of existing upper/lower partitioning. G-X gating line goes through the upper and lower LRLF arrays. Because of the regularity of the array structure, modifications of PP bit matrix and resource sharing are relatively simple. In tree multipliers, G-Y gating line follows the existing partitioning of tree branches. G-X line goes through all PPR CSAs. Because of the irregularity of the tree PPR structure, modifications along G-X line are different for CSAs at different levels and have to be handled separately. Experimental results indicate that 2-D G-G is also quite efficient in high-performance multipliers, with 65% power reduction under test data with large dynamic range. Although the area/delay overhead of 2-D G-G in ULLRLF is larger than that in tree, the power consumption of 2-D G-G ULLRLF is 7 $\sim$ 13% smaller.

CHAPTER 8

Conclusion and Future Work

8.1 Research Contributions

In this dissertation, we have investigated high-level optimization techniques for low-power multiplier design. At the arithmetic algorithm and architecture level, we have addressed the low-power design problem from two aspects: internal efforts considering multiplier algorithm/architectures and external efforts considering input data characteristics. For internal efforts, we considered recoding optimization for partial product generation, operand representation optimization, and structure optimization of partial product reduction. For external efforts, we considered signal gating to deactivate portions of a full-precision multiplier. Two classes of combinational multipliers are considered: linear array multipliers and tree multipliers. Layout experiments are performed to evaluate all the optimization techniques studied. The main contributions are as follows:

- Radix-4 parallel recoding with $/neg/two/one$ control signals, unique-zero handling and neg-first delay balance is proposed for high-performance multipliers. Radix-4 serial recoding with a simplified digit set and $P2/P1/N1$ control signals is proposed for linear array multipliers. Multipliers with new recoding schemes have achieved over 14% power reduction compared to those with non-optimized recoding schemes. Using the proposed recoding design, the power consumptions of radix-4 multipliers are shown to be much better than that of radix-2 multipliers.

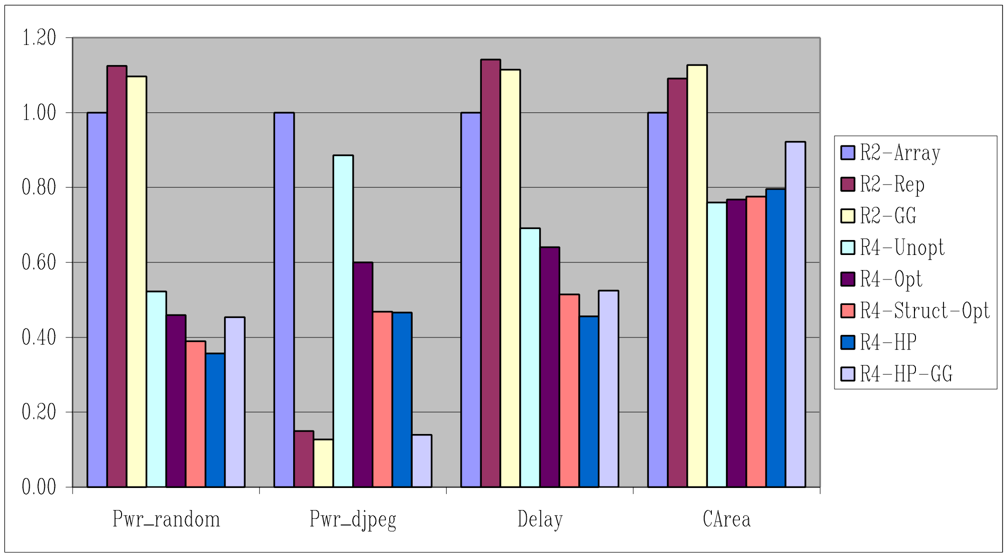
- Operand representation conversion schemes for two's-complement multipliers are proposed to utilize the low-power feature of sign-and-magnitude representation. The delay overhead in 2C-SM conversion is reduced by postponing the "+1" step in conversion and integrating it into the PP reduction process. The overhead in SM-2C conversion is reduced by converting two SM vectors from PPR directly to two 2C vectors before final CPA. For test data with a large dynamic range, the power consumption can be reduced by over 60% with only $7 \sim 9\%$ more area and $13 \sim 23\%$ more delay.

- From the basic L-R linear array multipliers, several structure optimization techniques are proposed: signal flow optimization by switching carry and sum signals in [3:2]-CSA based array, carry-ripple adders for PP reduction, [4:2]-CSAs for PP reduction, even/odd (leapfrog) split structure and upper/lower split structure. Because the optimizations are at the architecture and algorithm level, both power reduction and delay reduction have been achieved. Among different optimization techniques for L-R array multipliers, L-R-CSSC is a primary choice with a very simple structure if power is the critical concern. When small power-delay product is the main goal, L-R-leapfrog and L-R-CSSC-U/L split array structures are better candidates with a simple structure.

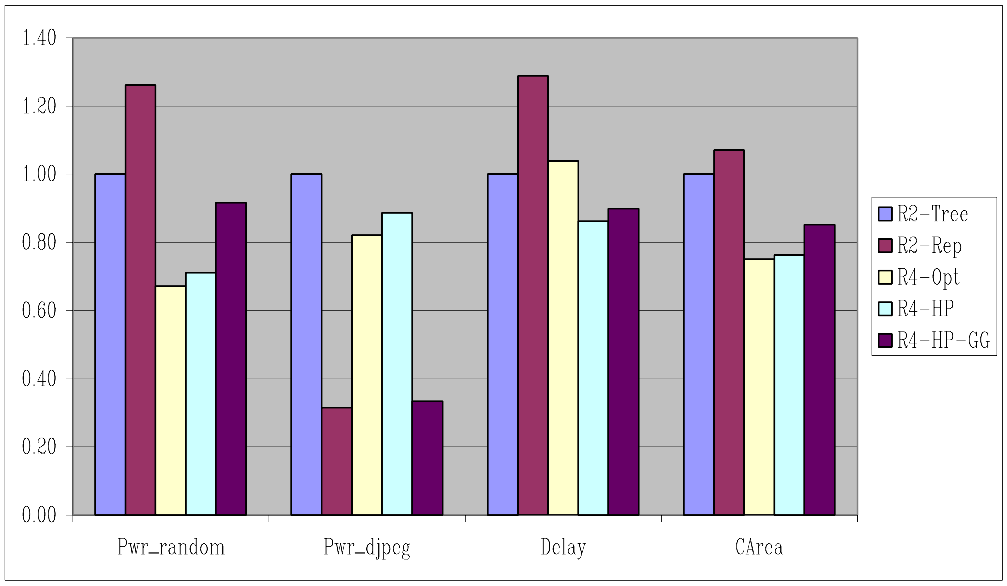
- To approach the performance of tree multipliers while maintaining the regularity of array structure, ULLRLF array multipliers that integrate array splitting, L-R-leapfrog, and signal flow optimization are proposed for $n \leq 32$. The delay of tree multipliers is also reduced by using a [9:4]-CSA with $3T_{XOR2}$ delay. Layout experiments indicate that ULLRLF and tree multipliers have similar delay while ULLRLF array multipliers consume $6 \sim 10\%$ less power. The floorplan of a ULLRLF array multiplier is also more regular and easier to control in layout optimization than that of a tree multiplier. With the smallest power consumption and relatively simple structure, ULLRLF becomes the best choice of high-performance multipliers for $n \leq 32$.

- 2-D signal gating schemes are proposed and studied in both traditional linear array multipliers and the proposed high-performance multipliers. Both 2-D signal gating and representation conversion are only useful for large-dynamic-range data. Representation conversion technique is simpler while signal gating is more power-efficient. In 2-D gating, different regions of the multiplier are dynamically deactivated according to the precision information of two operands. The gating overhead has been minimized by careful design of sign-extension logic, gating control logic, gating boundary modification, final CPA sharing and sign-extension restoration. In traditional array multipliers, 2-D signal gating schemes have achieved $45 \sim 56\%$ power reduction with small overhead in area and delay compared to previous work 1-D G-P under large-dynamic-range test data. In high-performance multipliers, 2-D gating is also very efficient. Because of the irregularity of the tree structure, signal gating in tree multipliers requires much more design efforts than in ULLRLF array multipliers. ULLRLF array multipliers with 2-D general gating also consume $7 \sim 13\%$ less power than tree multipliers in our experiment.

Figure 8.1 and 8.2 summarize main experimental results of optimization techniques studied in this dissertation. The results are normalized to reduce the effects of input registers, different CPAs and clock frequencies. Data in Figure 8.1 are grouped as power/delay/area characteristics. Data in Figure 8.2 are grouped

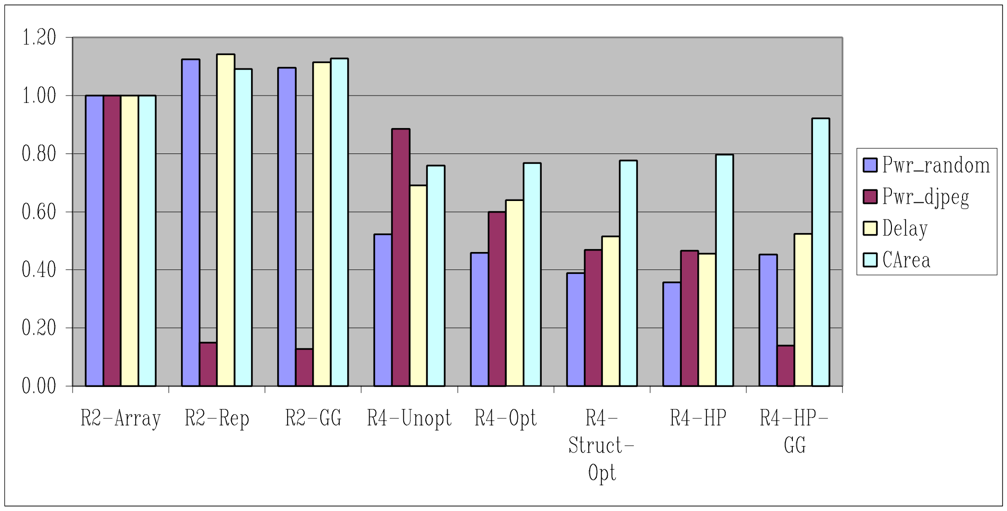


(a)

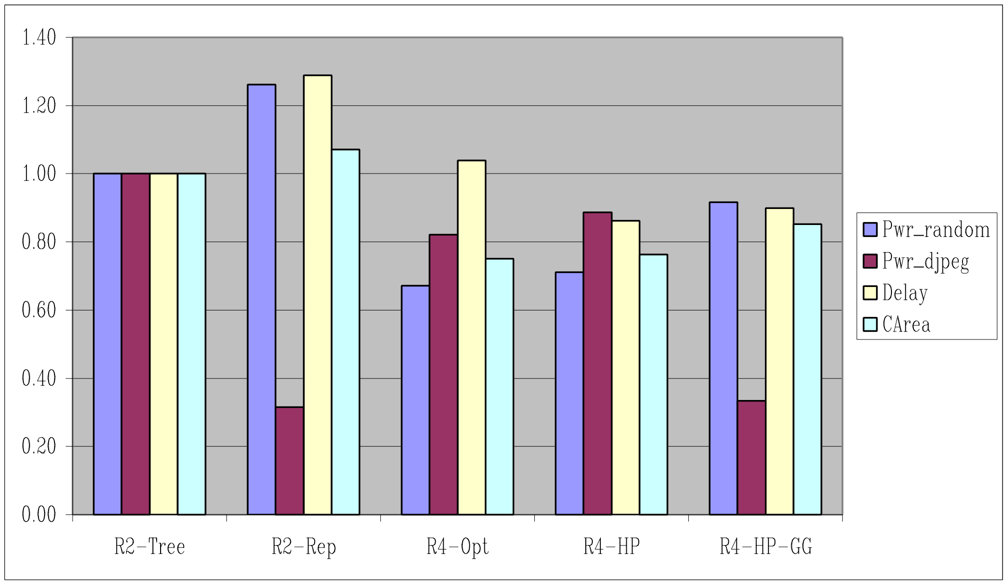


(b)

Figure 8.1: Normalized experimental results of optimization techniques in (a) array multipliers, (b) tree multipliers.



(a)



(b)

Figure 8.2: Another view of normalized experimental results in (a) array multipliers, (b) tree multipliers.

as individual schemes. *Pwr_random* and *Pwr_djpeg* are power consumptions under *random* and *djpeg* test data, respectively. *CArea* is the total cell area. For array multipliers, R2-Array is the baseline radix-2 32×32-bit multiplier, R2-Rep is the multiplier with representation optimization, R2-GG is with 2-D G-G gating, R4-Unopt is with un-optimized radix-4 recoding, R4-Opt is with optimized recoding, R4-Struct-Opt is with optimized recoding and reduction structure, R4-HP is with optimized recoding and high-performance structure, R4-HP-GG is with R4-HP and 2-D G-G gating. For tree multipliers, R2-Tree is the baseline radix-2 32×32-bit multiplier, R2-Rep is the multiplier with representation optimization, R4-Opt is with optimized radix-4 recoding, R4-HP is with optimized recoding and high-performance structure, R4-HP-GG is with R4-HP and 2-D G-G gating. In array multipliers, both power and delay are reduced steadily from R4-Unopt to R4-HP. In tree multipliers, radix-4 recoding reduces area and power while structure optimization reduces the delay. For large-dynamic-range data, the power reductions in representation optimization and signal gating techniques are significant with small area and delay overhead.

Overall, this dissertation has shown that high-level optimization techniques that consider the arithmetic computation features and application-specific data characteristics are very efficient for low-power multiplier design.

8.2 Future Work

As an attempt to develop arithmetic algorithm and architecture level optimization techniques for low-power multiplier design, the research presented in this dissertation has achieved good results and demonstrated the efficiency of high-level optimization techniques. However, there are limitations in our work and several future research directions are possible.

One possible direction is radix higher-than-4 recoding. We have only considered radix-4 recoding as it is a simple and popular choice. Higher-radix recoding further reduces the number of PPs and thus has the potential of power saving. Because of the difficulty of generating hard PPs such as $3X$, higher-radix recoding may increase critical path delay and design complexity which are negative factors for power. Thus, there is power/delay/area tradeoff in higher-radix recoding.

Another direction is to extend our work on signal gating to specialized multipliers such as partitionable multipliers and saturating multipliers. Partitionable multipliers are becoming important because the data precisions vary widely in different applications [78]. For signal gating, the multipliers have already been partitioned into several regions for gating. These partitioning techniques can be extended to the design of partitionable multipliers and signal gating can be applied as well. Saturating multipliers are often desirable in signal processing and graphics applications [109]. In some cases when saturation conditions can be determined in advance (e.g., by examining the MSB bits of two operands), the multiplier can be turned off and replaced by simple saturation logic [49].

The third direction is data-dependent power modeling. The experimental results in our research have been obtained for two specific data sets by using sophisticated CAD tools from the logic level to the physical level. It is not clear how the relative relations of different schemes would change when data with different statistical characteristics are applied. It is desirable to develop reliable data-dependent power models so that a methodology can be developed to automatically select the best schemes for target applications.

APPENDIX A

Abbreviations

1-D G-X	one-dimensional gating based on the precision of operand X
1-D G-Y	one-dimensional gating based on the precision of operand Y
1-D G-P	one-dimensional gating based on the precision of product P
2C	two's-complement
2C-SM-2C	straightforward 2C-SM and SM-2C conversion
2C-P1-2C	2C-SM conversion with postponed "+1"
2C-P1-CS	2C-SM with postponed "+1" and SM-2C using carry-save addition
2-D G-J	two-dimensional joint gating
2-D G-G	two-dimensional general gating
CLA	carry-lookahead adder
CPA	carry-propagate adder
CRA	carry-ripple adder
CSA	carry-save adder
CSSC	signal flow with Cin-to-B and Sin-to-C connection between FAs
CSELA	carry-select adder
CSUMA	conditional-sum adder
EOLRLF	even/odd split LRLF array multiplier
FA	full adder
GEF	generalized earliest-first algorithm for adder generation

HA	half adder
LR	left-to-right
LRCF	LR carry-free array multiplier
LRLF	LR leapfrog array multiplier
LR-CSSC	LR [3:2]-CSA array multiplier with CSSC flow
LR-CSSC-U/L	upper/lower split LR array multiplier with CSSC flow
LSB	least-significant bit
MSB	most-significant bit
NPR3	new parallel recoding with three control signals
NSR3	new serial recoding with three control signals
NSR4	new serial recoding with four control signals
OTFC	on-the-fly conversion
PA	prefix adder
PP	partial product
PPG	partial product generation
PPR	partial product reduction
PR3	parallel recoding with three control signals
PR4	parallel recoding with four control signals
PR5	parallel recoding with five control signals
REC	recoding
SALRLF	split array LRLF multiplier
SED	sign-extension detection logic
SFO	signal flow optimization for adder connections
SR3	serial recoding with three control signals
SM	sign-and-magnitude
ULLRLF	upper/lower split LRLF array multiplier

APPENDIX B

Design and Experimental Methodology

This appendix explains the design and simulation methodologies that have been used to obtain the experimental results in this dissertation. There are many choices for logic styles, design descriptions, and simulation methods. Our choices and the reasons are given below.

B.1 Logic Style

CMOS (Complementary Metal Oxide Semiconductor) is the primary technology choice in the semiconductor industry because of its many nice features such as small area, low power, relatively simple fabrication process [104]. The main alternative is ECL (Emitter Coupled Logic) which offers a speed advantage over CMOS at the expense of large area and increased power consumption [12]. As the technology scales, however, CMOS speed is catching up because the inferior scaling properties of bipolar devices in ECL [104]. The speed of CMOS could also be improved by using BiCMOS and dynamic CMOS. As our primary goal is low power multiplier design, we choose static CMOS as the underlying technology.

Even within static CMOS logic style, there are many variations used for arithmetic circuits [98][2][137]: conventional CMOS, transmission-gate CMOS, complementary pass-transistor logic (CPL), etc. Many existing designs of multipliers rely on NMOS pass transistor logic to achieve smaller area and delay. Because of the poor driving capability and degraded high voltage level in NMOS pass transistor logic, careful custom transistor-level design is necessary for circuit robustness. Pull-up PMOS transistors are often needed to force a node to be high. To reduce the number of inverters in pass transistor logic, dual-rail signals are often used. Dual-rail signals increase wiring complexity and switching capacitance significantly. On the other hand, conventional CMOS logic style is robust with respect to voltage scaling and transistor sizing. Conventional CMOS logic style also has the advantages of generality and ease-of-use as standard cell based technology libraries and logic synthesis techniques are well developed and widely used. Zimmermann and Fichtner [137] has shown that CMOS logic style is a good choice in most cases if low voltage, low power, and small power-delay products are of major concern. For these reasons, we choose CMOS standard cell logic style in this dissertation.

B.2 Artisan Standard Cell Library

The CMOS standard cell library we have used is Artisan TSMC 0.18μm 1.8-Volt standard-cell library [10]. The area/delay/power characteristics of some basic cells are simplified and shown in Table B.1. All cells in this table have unit drive strength (X1). The area unit is μm^2. The dynamic power unit is $\mu W/Hz$. The unit of pin capacitance is pF. The delay unit is ns. P_{dyn}, C_{pin}, t_{pLH}, and t_{pHL} are the averages of individual values for each input pin. The values in this table are listed only for high-level quick estimation purpose. For gate-level simulation tools such as Synopsys VSS [119], the library provides look-up table (LUT) based non-linear delay and power models. The values in LUTs are characterized by electrical simulations of each cell for all possible input transitions and for a wide range of fan-in and fan-out conditions. Non-linear LUT-based models are more accurate than analytical models if the size of LUT is large enough so that the interpolations for cell conditions with no table entries also have good accuracy.

Table B.1: Area/delay/power characteristics of some Artisan library cells

Cells	$Area$	P_{dyn}	C_{pin}	t_{pLH}	t_{pHL}
AND2	5.0 x 2.6	0.0214	0.0020	$0.086 + 4.522C_L$	$0.118 + 2.450C_L$
AOI22	5.0 x 3.3	0.0265	0.0045	$0.076 + 6.723C_L$	$0.035 + 2.725C_L$
AOI222	5.0 x 5.3	0.0430	0.0050	$0.146 + 8.642C_L$	$0.049 + 2.739C_L$
INV	5.0 x 1.3	0.0114	0.0036	$0.023 + 4.512C_L$	$0.014 + 2.401C_L$
MX2	5.0 x 5.3	0.0301	0.0040	$0.098 + 4.520C_L$	$0.120 + 2.474C_L$
MX4	5.0 x 13.2	0.0510	0.0062	$0.137 + 4.528C_L$	$0.174 + 2.579C_L$
MXI2	5.0 x 4.6	0.0247	0.0048	$0.051 + 4.891C_L$	$0.047 + 2.831C_L$
MXI4	5.0 x 15.2	0.0456	0.0038	$0.205 + 4.521C_L$	$0.199 + 2.486C_L$
NAND2	5.0 x 2.0	0.0152	0.0039	$0.031 + 4.512C_L$	$0.019 + 2.784C_L$
NOR2	5.0 x 2.0	0.0149	0.0041	$0.041 + 6.722C_L$	$0.020 + 2.407C_L$
OAI22	5.0 x 4.0	0.0279	0.0046	$0.068 + 6.732C_L$	$0.037 + 2.723C_L$
OAI222	5.0 x 5.9	0.0414	0.0049	$0.107 + 6.739C_L$	$0.063 + 3.021C_L$
OAI32	5.0 x 4.6	0.0348	0.0048	$0.095 + 7.794C_L$	$0.046 + 2.734C_L$
OR2	5.0 x 2.6	0.0203	0.0025	$0.066 + 4.517C_L$	$0.140 + 2.475C_L$
XNOR2	5.0 x 5.3	0.0445	0.0059	$0.125 + 4.521C_L$	$0.125 + 2.470C_L$
XOR2	5.0 x 5.3	0.0421	0.0059	$0.119 + 4.517C_L$	$0.127 + 2.474C_L$
TLAT	5.0 x 7.3	0.0215	0.0034	$0.167 + 4.520C_L$	$0.181 + 2.457C_L$
DFF	5.0 x 11.2	0.0313	0.0023	$0.209 + 4.403C_L$	$0.223 + 2.457C_L$

Artisan library also provides some synthesis optimized arithmetic cells, such as 1-bit full adder, Booth recoder, [4:2] compressor. As the structures of these basic cells are also one of our study objectives, we don't use Artisan arithmetic cells in order to provide a fair comparison of different schemes.

B.3 Power/Delay/Area Estimation

The best way to compare different schemes is to fabricate each design and measure the power/delay/area characteristics of actual chips. However, the fabrication is a rather time-consuming and expensive process, which makes it impractical to explore many design alternatives. In this dissertation, we focus on high-level power optimization techniques which are technology-independent as long as the assumptions of static CMOS technology hold. With the advancement of CAD tools, cell-based full-timing gate-level simulation (FTGS) with back-annotated layout information can achieve the accuracy of within 10-15% of SPICE results [130][120]. As we compare different schemes under the same experimental setting and and the primary concern is the relative performance, the absolute errors tend to go in the same direction and thus have little effect on the relative comparison. We choose Synopsys design environment including Design Compiler (DC) [118], VHDL Simulator VSS [119], and Power Compiler [120], and Cadence layout environment including Silicon Ensemble Place-and-Route [19] and Pearl Timing Analyzer [21]. DC analyzes VHDL designs, optimizes and maps designs into target standard-cell technologies. Silicon Ensemble accepts the netlist and design constraints from DC and performs timing-driven automatic layout. The layout parameters are extracted from the layout and back-annotated into Synopsys tools for accurate estimation. With given test data, VHDL simulator collects switching activity information by dynamic timing simulation. Power Compiler is a gate-level power optimization and synthesis system [28]. In our experiments, we only use Power Compiler as a power estimation tool based on switching and capacitance information. Using non-linear LUT-based delay and power models in Artisan standard cell FTGS library, the power estimation is based on FTGS which can capture spurious transitions or glitches.

All schemes have been implemented in technology-independent structural VHDL descriptions. For schemes requiring internal module-level optimizations (e.g., ULLRLF), VHDL generation programs are written to facilitate the optimizations and automatically produce optimized VHDL descriptions. These VHDL designs are described with user-definable parameters, such as the multiplication size, recoding methods, and signal gating positions. The regularity of multiplier structures is kept by describing designs in a hierarchical way. The fundamental components such as full adders are described in switching expressions for technology independence. VHDL simulation is first conducted to verify the correctness of each design. Test data are random data and some special boundary data. VHDL designs are then mapped into Artisan TSMC $0.18\mu m$ 1.8-Volt standard-cell library. We set a loose delay objective during mapping because our logic-level design has determined the delay range and aggressive gate-level delay minimization may lead to much larger area and power. The main objective is set to minimize area because area minimization in general helps power saving for a given design. The switching expressions are written in the way that they can be easily mapped to most common library cells. For a few cases when the expressions do not map to the desired cells such as OAI32, explicit manual mapping is performed. Block boundary optimization is allowed in order to simplify inputs with constant values across design hierarchy. According to our specification of fanout limits, buffers are automatically inserted by DC.

There are two main ways to estimate interconnect effects in power/delay/area characteristics. One is to extract interconnect information from the actual layout for very accurate estimation. The other is to use wire-load models at the gate level to estimate the effects of wire length and fanout on the resistance, capacitance, and area of interconnects [118]. Semiconductor vendors provide wire load models, based on statistical information specific to the vendors' process. These models include coefficients for area, capacitance, and resistance per unit length, and a fanout-to-length table for estimating net lengths. For schemes with similar interconnect structures, the use of wire-load models in estimation provides a fast and fairly accurate approach. For schemes with quite different interconnect structure, different wire-load models have to be applied carefully according to the interconnect complexity. To get accurate information of interconnects, we choose to perform standard-cell based automatic layout design for all schemes we have studied using Cadence Silicon Ensemble. The design constraints in DC are passed into Silicon Ensemble for consistency. The row utilization rates for layouts are initially set at a high value (85%), and decrease 5% each time if the routing cannot be completed. We name the rate with the densest successful layout as a *routable rate*. The actual die area of the core region (excluding IOs) on a chip is the total cell area (*CArea*) of a scheme divided by the row utilization rate. Experiments show that the routable rates roughly reflect the interconnect complexity of distinct multiplier schemes. In addition, the larger the area, the lower the routable rate. For power analysis, we find that the cell area rather than the die area has a better relation with the power because the layout effects and the routable rates are less predictable in our experiments. For high-performance multipliers with more complex interconnect structures, layouts with guided floorplanning are also performed by specifying placement regions. Guided floorplanning is primarily for delay reduction on the critical paths. Region constraints manually insert placement boundaries and affect global optimization. The overall placement quality can degrade because of overconstraining [20].

The layout information is extracted in several forms. Resistance and capacitance parasitics of interconnects are reported in reduced Standard Parasitic Format (RSPF). Based on RSPF data, Pearl static timing analyzer is used to calculate delay in Standard Delay Format (SDF) that shows interconnect and cell delay limits. With back-annotated SDF information, Synopsys VSS performs full-timing gate-level simulation for given test data and captures switching activity information in Switching Activity Interchange Format (SAIF). Power Compiler annotates the design with SAIF information and performs power analysis.

B.4 Test Data Sets

Power dissipation is directly related to input data characteristics. One scheme may consume less power for certain data patterns but consume more power for other data patterns. Therefore, we prepared two test data sets in order to capture power features in different application environments. One test data set is *random*, consisting of 32-bit pseudo-random data. The static probability of each bit being '1' in *random* is 0.5 and the toggle rate of each bit is 0.25. The second data set is *djpeg*, gathered by tracing the execution of 32×32-bit multiplication in a 32-bit JPEG-decoding program with a typical image input (from MediaBench Suite [79]). The program is compiled for the SPARC v8 architecture and traced using Shade [117]. *djpeg* data have a large dynamic range and most data precisions are less than 16-bit. The probability of each bit being sign-extension bit is shown in Figure B.1. For operand1 (X), the probability of being sign-extension bit becomes 0.9997 from the 13-th bit. For operand2 (Y), the probability becomes 0.9996 from the 18-th bit. Each probability line can be divided into three regions: MSB region for sign extension bits, linear region for correlated data bits, and LSB region for uncorrelated data bits. In our signal gating experiment, the gating positions are chosen to be some point in the linear regions. In this case, they are the 9-th bit position in operand1 ($NG_X = 10$) and the 15-th bit position in operand2 ($NG_Y = 16$).

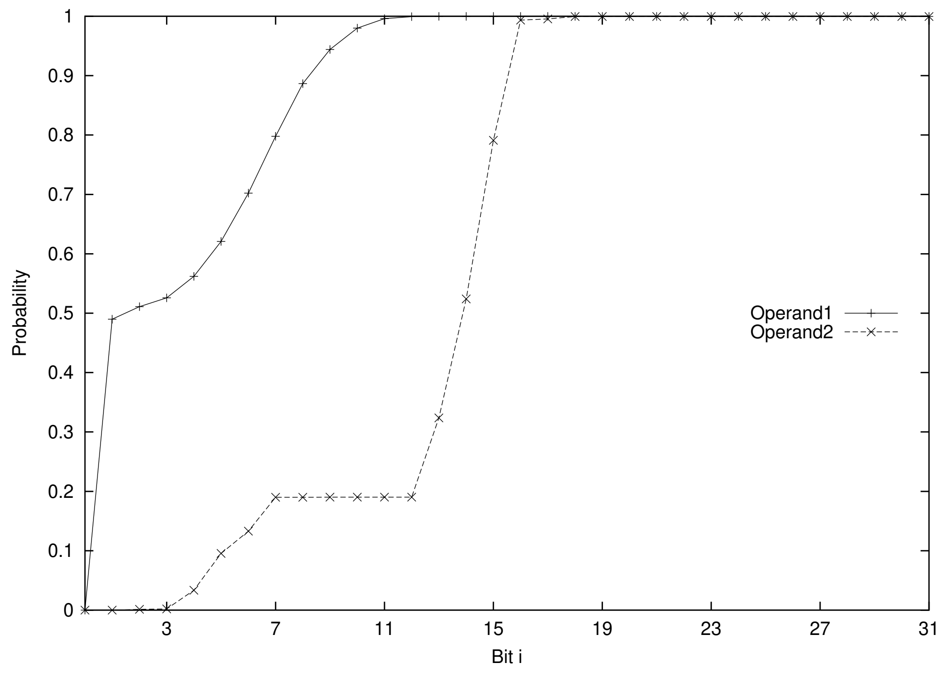


Figure B.1: Probability of being extension bit in *djpeg* data.

Although most MSB bits in *djpeg* are sign extension bits, they switch frequently at the probability of $0.27 \sim 0.33$. Figure B.2 shows the static probability and the toggle rate of each bit in *djpeg*. The static probability of MSB bits being '1' indicates the probability of negative numbers. For operand1, the static probability and toggle rate do not change much. The range of static probability is $0.22 \sim 0.27$ and the range of toggle rate is $0.26 \sim 0.31$. Most data for operand1 are probably inverse Discrete Cosine Transform (IDCT) coefficients that are a set of constant values for a given algorithm. In custom DSP chips, multipliers with constant coefficients can be simplified by recoding these coefficients [45]. In programmable DSP chips and microprocessors, general-purpose multipliers are provided and shared among various computing situations, which is our focus.

For operand2, the static probability and the toggle rate change dramatically in the LSB region because of the input image data.

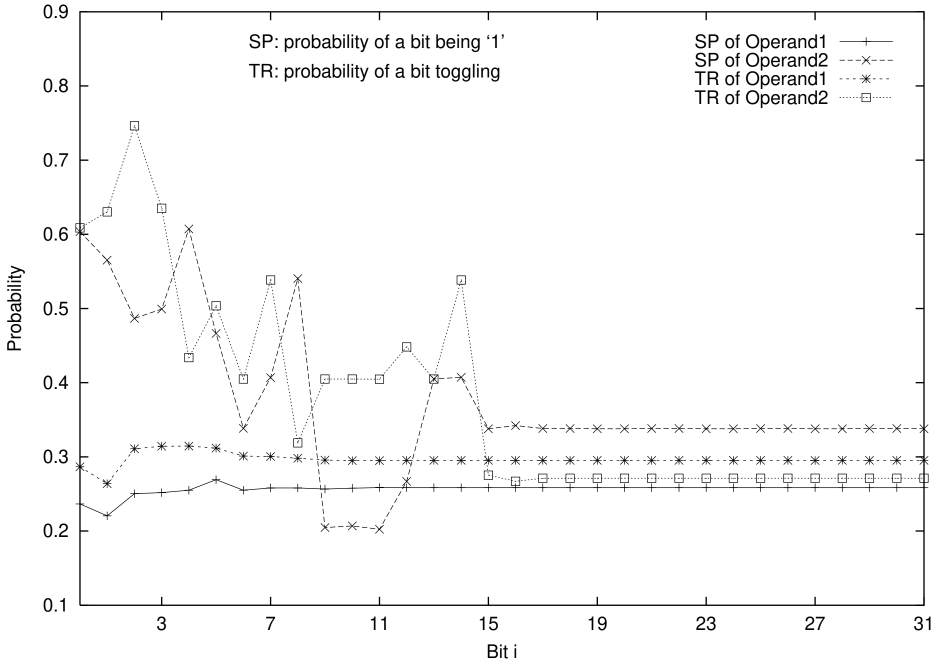


Figure B.2: Static probability and toggle rate of each bit in *djpeg* data.

References

[1] A. Abnous and J. Rabaey, "Ultra-low-power domain-specific multimedia processors," in *VLSI Signal Processing, IX*, pp.461-470, Oct. 1996.

[2] I. Abu-Khater, A. Bellaouar, and M. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers", *IEEE J. Solid-State Circuits*, vol.31, no.10, pp.1535-1546, Oct. 1996.

[3] H. A. Al-Twaijry, *Area and Performance Optimized CMOS Multipliers.* Ph.D. dissertation, Stanford University, Aug. 1997.

[4] M. Alidina, *et. al.*, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Systems*, vol.2, no.4, pp.426-436, Dec. 1994.

[5] F.S. Anderson, *et. al.*, "The IBM system 360/91 floating point execution unit," *IBM J. Res. Develop.*, vol.11, pp.34-53, Jan. 1967.

[6] E. de Angel and E.E. Swartzlander, Jr., "Survey of low power techniques for VLSI design," in *Proc. 8th Annual IEEE Int. Conf. Innovative Systems in Silicon*, pp.159-169, Oct. 1996.

[7] E. de Angel and E.E. Swartzlander, Jr., "Low power parallel multipliers," in *VLSI Signal Processing, IX*, pp.199-208, Oct. 1996.

[8] E. de Angel, *Low Power Digital Multiplication.* Ph.D. dissertation, The University of Texas at Austin, 1996.

[9] E. de Angel and E.E. Swartzlander, Jr., "Switching activity in parallel multipliers," in *Proc. 35th Asilomar Conf. Signals, Systems and Computers*, pp.857-860, Nov. 2001.

[10] *TSMC* 0.18μm *Process 1.8-Volt SAGE-X Standard Cell Library Databook.* Artisan Components, Inc., Oct. 2001.

[11] A. Bellaouar and M. Elmasry, *Low-power Digital VLSI Design: Circuits and Systems.* Kluwer Academic Publishers, 1995.

[12] G.W. Bewick, *Fast Multiplication: Algorithms and Implementation.* Ph.D. dissertation, Stanford University, Feb. 1994.

[13] M. Bhardwaj, R. Min, and A. Chandrakasan, "Power-aware systems," in *Proc. 34th Asilomar Conf. Signals, Systems and Computers*, vol.2, pp.1695-1701, Nov. 2000.

[14] K.C. Bickerstaff, E.E. Swartzlander, Jr., and M.J. Schulte, "Analysis of column compression multipliers," in *Proc. 15th IEEE Symp. Computer Arithmetic*, pp.33-29, 2001.

[15] B. Bishop, T.P. Kelliher, and M.J. Irwin, "A detailed analysis of MediaBench," in *Proc. 1999 IEEE Workshop on Signal Processing Systems*, pp.448-455, 1999.

[16] M. Borah, R.M. Owens, and M.J. Irwin, "High-throughput and low-power DSP using clocked-CMOS circuitry," in *Proc. 1995 Int. Symp. Low Power Design*, pp.139-144, Apr. 1995.

[17] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. 5th Int. Symp. High-Performance Computer Architecture*, pp.13-22, 1999.

[18] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance," *ACM Trans. Computer Systems*, vol.18, no.2, pp.89-126, May 2000.

[19] *Envisa Silicon Ensemble Reference – Version 5.3.* Cadence Design Systems, Inc, Nov. 1999.

[20] *Envisa Ultra Placer Reference – Version 5.1.* Cadence Design Systems, Inc, Dec. 1999.

[21] *Envisa Ultra Placer Reference – Version 5.1.* Cadence Design Systems, Inc, Dec. 1999.

[22] T.K. Callaway, *Area, Delay, and Power Modeling of CMOS Adders and Multipliers.* Ph.D. dissertation, The University of Texas at Austin, 1996.

[23] T.K. Callaway and E.E. Swartzlander, Jr., "Power-delay characteristics of CMOS multipliers," in *Proc. 13th IEEE Int. Symp. Computer Arithmetic*, pp.26-32, 1997.

[24] R. Canal, A. Gonzalez, and J.E. Smith, "Very low power pipelines using significance compression," in *Proc. 33rd Annual IEEE/ACM Int. Symp. Microarchitecture*, pp.181-190, 2000.

[25] F. Catthoor, *Unified Low-power Design Flow for Data-dominated Multimedia and Telecom Applications.* Kluwer Academic Publishers, 2000.

[26] A.P. Chandrakasan, S. Sheng and R.W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol.27, no.4, pp.473-484, Apr. 1992.

[27] A.P. Chandrakasan and R.W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol.83, no.4, pp.498-523, Apr. 1995.

[28] B. Chen and I. Nedelchev, "Power compiler: a gate-level power optimization and synthesis system," in *Proc. 1997 IEEE Int. Conf. Computer Design*, pp.74-79, Oct. 1997.

[29] L.-H. Chen, W.-L. Liu, and O.T.-C. Chen, "Determination of radix numbers of the Booth algorithm for the optimized programmable FIR architecture," in *Proc. 2000 IEEE Int. Symp. Circuits and Systems*, vol.2, pp.345-348, May 2000.

[30] B.S. Cherkauer and E.G. Friedman, "A hybrid radix-4/radix-8 low power signed multiplier architecture," *IEEE Trans. Circuits and Systems – II: Analog and Digital Signal Processing*, vol.44, no.8, pp.656-659, Aug. 1997.

[31] C. Chien, *Digital Radio Systems on a Chip: A Systems Approach.* Kluwer Academic Publishers, 2001.

[32] J. Choi, J. Jeon, and K. Choi, "Power minimization of functional units by partially guarded computation," in *Proc. 2000 Int. Symp. Low Power Electronics and Design*, pp.131-136. Jul. 2000.

[33] K. Choi and M. Song, "Design of a high performance 32*32-bit multiplier with a novel sign select Booth encoder," in *Proc. 2001 IEEE Int. Symp. Circuits and Systems*, vol.2, pp.701-704, May 2001.

[34] Y. Choi and Earl E. Swartzlander, Jr., "Design of a hybrid prefix adder for non-uniform input arrival times," in *Proc. SPIE 2002 Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, July 2002.

[35] H.-C. Chow and I-C. Wey, "A 3.3V 1GHz high speed pipelined booth multiplier," in *Proc. 2002 IEEE Int. Symp. Circuits and Systems*, vol.1, pp.457-460, May 2002.

[36] L. Ciminiera and P. Montuschi, "Carry-save multiplication schemes without final addition," *IEEE Trans. Comput.*, vol.45, no.9, pp.1050-1055, Sept. 1996.

[37] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol.34, pp.349-356, Mar. 1965.

[38] A.N. Danysh and E.E. Swartzlander, Jr. "A recursive fast multiplier," in *Proc. 32th Asilomar Conf. Signals, Systems and Computers*, pp.197-201, Nov. 1998.

[39] W.E. Dougherty and D.E. Thomas, "Modeling and automating selection of guarding techniques for datapath elements," in *Proc. 1999 Int. Symp. Low Power Electronics and Design*, pp.182-187, Aug. 1999.

[40] M.D. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Trans. Comput.*, vol.C-36, no.7, pp.895-897, July 1987.

[41] M.D. Ercegovac and T. Lang, "Fast multiplication without carry-propagate addition," *IEEE Trans. Comput.*, vol.39, no.11, pp.1385-1390, Nov. 1990.

[42] M.D. Ercegovac and T. Lang, "Reducing transition counts in arithmetic circuits," in *Proc. 1994 IEEE Symp. Low Power Electronics*, pp.64-65, 1994.

[43] M.D. Ercegovac and T. Lang, "Low-power accumulator (correlator)," in *Proc. 1995 IEEE Symp. Low Power Electronics* , pp.30-31, Oct. 1995.

[44] M.D. Ercegovac, C.A. Fabian, and T. Lang, "On reducing transition counts in sign detection circuits," in *Proc. 30th Asilomar Conf. Signals, Systems and Computers*, pp.596-599, Nov. 1996.

[45] M.D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann Publishers, Elsevier Science Ltd., 2003.

[46] M.D. Ercegovac and T. Lang, "Comments on "A carry-free 54b*54b multiplier using equivalent bit conversion algorithm"," *IEEE J. Solid State Circuits*, vol.38, no.1, pp.160-161, Jan. 2003.

[47] C.A. Fabian and M.D. Ercegovac, "Input synchronization in low power CMOS arithmetic circuit design," in *Proc. 31th Asilomar Conf. Signals, Systems and Computers*, pp.172-176, Nov. 1997.

[48] A.M. Fahim and M.I. Elmasry, "Low-power high-performance arithmetic circuits and architectures," *IEEE J. Solid-State Circuits*, vol.37, no.1, pp.90-94, Jan. 1997.

[49] Farzan Fallah, personal communication, Fujitsu Laboratories of America, Inc., 2002.

[50] A.A. Fayed and M.A. Bayoumi, "A novel architecture for low-power design of parallel multipliers," in *Proc. IEEE Computer Society Workshop on VLSI*, pp.149-54, Apr. 2001.

[51] J. Frenkil, "A multi-level approach to low-power IC design," *IEEE Spectrum Magazine*, pp.54-60, Feb. 1998.

[52] R. Fried, "Minimizing energy dissipation in high-speed multipliers," in *Proc. 1997 Int. Symp. Low Power Electronics and Design*, pp.214-219, Aug. 1997.

[53] G. Gerosa, *et. al.*, "A 2.2W, 80MHz superscalar RISC microprocessor," *IEEE J. Solid-State Circuits*, vol.29, no.12, pp.1440-1454, Dec. 1994.

[54] A. Goldovsky, *et. al.*, "Design and implementation of a 16 by 16 low-power two's complement multiplier," in *Proc. 2000 IEEE Int. Symp. Circuits and Systems*, vol.5, pp.345-348, 2000.

[55] G. Goto, *et. al.*, "A 54*54-b regularly structured tree multiplier," *IEEE J. Solid-State Circuits*, vol.27, pp.1229-1236, Sept. 1992.

[56] G. Goto, *et. al.*, "A 4.1-ns compact 54*54-b multiplier utilizing sign-select Booth encoders," *IEEE J. Solid-State Circuits*, vol.32, pp.1676-1682, Nov. 1997.

[57] M.K. Gowan, L.L. Biro, and D.B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor," in *Proc. 35th Design and Automation Conf*, pp.726-731, 1998.

[58] C.-Y. Han, H.-J. Park, and L.-S. Kim, "A low-power array multiplier using separated multiplication technique," *IEEE Trans. Circuits and Systems – II: Analog and Digital Signal Processing*, vol.48, no.9, pp.866-871, Sept. 2001.

[59] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd Edition. Morgan Kaufmann Publishers, Inc., 1996.

[60] X. Huang, *et. al.*, "High-performance VLSI multiplier with a new redundant binary coding," *Journal of VLSI Signal Processing*, vol.3, pp.283-291, Oct. 1991.

[61] X. Huang, W.-J. Liu, and B.W.Y. Wei, "A high-performance CMOS redundant binary multiplication-and-accumulation (MAC) unit," *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol.41, pp.33-39, Jan. 1994.

[62] Z. Huang and M.D. Ercegovac, "On signal-gating schemes for low-power adders," in *Proc. 35th Asilomar Conf. Signals, Systems and Computers*, pp.867-871, Nov. 2001.

[63] N. Itoh, *et. al.*, "A 600-MHz 54*54-bit multiplier with rectangular-styled Wallace tree," *IEEE J. Solid-State Circuits*, vol.36, no.2, p.249-257, Feb. 2001.

[64] *International Technology Roadmap for Semiconductors*, 2001 Edition.

[65] J. Iwamura, *et. al.*, "A high speed and low power CMOS/SOS multiplier-accumulator," *Microelectronics Journal*, vol.14, no.6, pp.49-57, Nov.-Dec. 1983.

[66] J. Kao, S. Narendra, and A. Chandrakasan, "subthreshold leakage modeling and reduction techniques," in *Proc. 2002 Int. Conf. Computer-Aided Design*, pp.141-148, Nov. 2002.

[67] G. Keane, J. Spanier, and R. Woods, "The impact of data characteristics and hardware topology on hardware selection for low power DSP," in *Proc. 1998 Int. Symp. Low Power Electronics and Design*, pp.94-96, Aug. 1998.

[68] K-Y. Khoo, Z. Yu, and A.N. Willson, Jr. "Improved-Booth encoding for low-power multipliers," in *Proc. 1999 IEEE Int. Symp. Circuits and Systems*, vol.1, pp.62-65, May 1999.

[69] J. Kim and E.E. Swartzlander, Jr., "Improving the recursive multiplier," in *Proc. 34th Asilomar Conf. Signals, Systems and Computers*, pp.1320-1324, Nov. 2000.

[70] K. Kim, P.A. Beerel, and Y. Hong, "An asynchronous matrix-vector multiplier for discrete cosine transform," in *Proc. 2000 Int. Symp. Low Power Electronics and Design*, pp.256-261, Jul. 2000.

[71] S. Knowles, "A family of adders," in *Proc. 14th IEEE Symp. Computer Arithmetic*, pp.30-34, 1999.

[72] U. Ko, P.T. Balsara, and W. Lee, "A self-timed method to minimize spurious transitions in low power CMOS circuits," in *Proc. 1994 IEEE Symp. Low Power Electronics*, pp.62-63, Oct. 1994.

[73] R.K. Kolagotla, H.R. Srinivas, and G.F. Burns, "VLSI implementation of a 200-MHz 16*16 left-to-right carry-free multiplier in 0.35 mu m CMOS technology for next-generation DSPs," in *Proc. IEEE 1997 Custom Integrated Circuits Conf.*, pp.469-472, May 1997.

[74] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[75] P.E. Landman and J.M. Rabaey, "Architectural power analysis: the dual bit type method," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.3, no.2, pp.173-187, 1995.

[76] T. Lang, E. Musoll, and J. Cortadella, "Individual flip-flops with gated clocks for low power datapaths," *IEEE Trans. Circuits and Systems – II: Analog and Digital Signal Processing*, vol.44, no.6, pp.507-516, June 1997.

[77] C.F. Law, S.S. Rofail, and K.S. Yeo, "A low-power 16×16-b parallel multiplier utilizing pass-transistor logic," *IEEE J. Solid State Circuits*, vol.34, no.10, pp.1395-1399, Oct. 1999.

[78] Ruby B. Lee, "Computer arithmetic – a processor architect's perspective," in *Proc. 15th IEEE Symp. Computer Arithmetic*, Keynote Presentation, June 2001.

[79] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 30th Annual IEEE/ACM Int. Symp. Microarchitecture*, pp.330-335, Dec. 1997.

[80] J. Leijten, J. van Meerbergen, and J. Jess, "Analysis and reduction of glitches in synchronous networks," in *Proc. 1995 European Design and Test Conf.*, pp.398-403, March 1995.

[81] M.J.G. Lewis, *Low Power Asynchronous Digital Signal Processing.* Ph.D. dissertation, University of Manchester, Oct. 2000.

[82] Y.J. Lim, *et. al.*, "A statistical approach to the estimation of delay-dependent switching activities in CMOS combinational circuits," in *Proc. 33rd Design Automation Conference*, pp.445-450, June 1996.

[83] O.L. MacSorley, "High-speed arithmetic in binary computers," *IRE Proceedings*, vol.49, pp.67-91, 1961.

[84] S.S. Mahant-Shetti, P.T. Balsara, and C. Lemonds, "High performance low power array multiplier using temporal tiling," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.7, no.1, p.121-124, March 1999.

[85] H. Makino, *et. al.*, "An 8.8-ns 54*54-bit multiplier with high speed redundant binary architecture," *IEEE J. Solid-State Circuits*, vol.31, pp.773-783, June 1996.

[86] P. Meier, R. Rutenbar, and L. Carley, "Exploring multiplier architecture and layout for low power," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp.513-516, May 1996.

[87] P.C.H. Meier, *Analysis and Design of Low Power Digital Multipliers.* Ph.D. dissertation, Carnegie Mellon University, 1999.

[88] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. 1993 Int. Conf. Computer-Aided Design*, pp.398-402, Nov. 1993.

[89] J. Mori, *et. al.*, "A 10 ns 54*54-b parallel structured full array multiplier with 0.5 mu m CMOS technology", *IEEE J. Solid-State Circuits*, vol.26, p.600-606, Apr. 1991.

[90] K. Muhammad, D. Somasekhar, and K. Roy, "Switching characteristics of generalized array multiplier architectures and their applications to low power design," in *Proc. 1999 IEEE Int. Conf. Computer Design*, pp.230-235, Oct. 1999.

[91] E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," in *Proc. 1995 Int. Symp. Low Power Design*, pp.99-104, Apr. 1995.

[92] E. Musoll and J. Cortadella, "Low-power array multipliers with transition-retaining barriers," in *Proc. Int. Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)*, pp.227-238, October 1995.

[93] M. Nagamatsu, *et. al.*, "A 15-ns 32*32-b CMOS multiplier with an improved parallel structure," *IEEE J. Solid-State Circuits*, vol.25, pp.494-497, Apr. 1990.

[94] F. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.13, no.2, pp.310-323, Feb. 1993.

[95] F. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.2, no.4, pp.446-455, Dec. 1994.

[96] F. Najm, "Low-pass filter for computing the transition density in digital circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, pp.1123-1131, Sep. 1994.

[97] C.J. Nicol and P. Larsson, “Low power multiplication for FIR filters,” in *Proc. 1997 Int. Symp. Low Power Electronics and Design*, pp.76-9, Aug. 1997.

[98] N. Ohkubo, *et. al.*, “A 4.4 ns CMOS 54*54-b multiplier using pass-transistor multiplexer,” *IEEE J. Solid-State Circuits*, vol.30, pp.251-257, March 1995.

[99] V.G. Oklobdzija, D. Villeger, and S.S. Liu, “A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach,” *IEEE Trans. Comput.*, vol.45, no.3, pp.294-306, March 1996.

[100] Y. Oowaki, *et. al.*, “A sub-10-ns 16*16 multiplier using 0.6-um CMOS technology,” *IEEE J. Solid-State Circuits*, vol.SC-22, no.5, pp.762-767, Oct. 1987.

[101] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[102] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, New-York, 2000.

[103] Chan-Ho Park, *et. al.*, “Asynchronous array multiplier with an asymmetric parallel array structure,” in *Proc. 2001 Conference on Advanced Research in VLSI*, pp.202-212, March 2001.

[104] J.M. Rabaey, *Digital Integrated Circuits: a Design Perspective.* Prentice Hall, 1996.

[105] S. Ramprasad, N.R. Shanbha, and I.N. Hajj, “Analytical estimation of signal transition activity from word-level statistics”, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.16, no.7, pp.718-733, July 1997.

[106] K. Roy and S.C. Prasad, *Low-power CMOS VLSI Circuit Design.* Wiley&Son, 2000.

[107] T. Sakuta, W. Lee, and P.T. Balsara, “Delay balanced multipliers for low power/low voltage DSP core,” in *Proc. 1995 IEEE Symp. Low Power Electronics*, pp.36-37, Oct. 1995.

[108] H. Sam and A. Gupta, “A generalized multibit recoding of two’s complement binary numbers and its proof with application in multiplier implementations,” *IEEE Trans. Comput.*, vol.39, no.8, pp.1006-1015, Aug. 1990.

[109] M.J. Schulte, *et. al.*, "Integer multiplication with overflow detection or saturation," *IEEE Trans. Comput.*, vol.49, no.7, pp.681-691, July 2000.

[110] P.-M. Seidel, L.D. McFearin, and D.W. Matula, "Binary multiplication radix-32 and radix-256," in *Proc. 15th IEEE Symp. Computer Arithmetic*, pp.23-32, June 2001.

[111] N.-Y. Shen and O.T.-C. Chen, "Low-power multipliers by minimizing switching activities of partial products," in *Proc. 2002 IEEE Int. Symp. Circuits and Systems*, vol.4, pp.93-96, May 2002.

[112] P.J. Song and G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE J. Solid-State Circuits*, vol.26, pp.1184-1198, Sept. 1991.

[113] P.F. Stelling and V.G. Oklobdzija, "Designing optimal hybrid final adders in a parallel multiplier using conditional sum blocks," in *Proc. 15th IMACS World Congress Scientific Computation, Modeling, and Applied Math.*, Aug. 1997.

[114] P.F. Stelling, *et. al.*, "Optimal circuits for parallel multipliers," *IEEE Trans. Comput.*, vol.47, no.3, pp.273-285, March 1998.

[115] M. Stephenson, J. Babb, and S. Amarashinghe, "Bitwidth analysis with application to silicon compilation", *ACM SIGPLAN Notices*, vol.35, no.5, pp.108-120, May 2000.

[116] A.G.M. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in *Proc. 2000 Int. Symp. Low Power Electronics and Design*, pp.114-119, 2000.

[117] *Shade User's Manual.* Sun Microsystems, Inc., 1993.

[118] *Design Compiler User Guide.* Synopsys, Inc., Nov. 2000.

[119] *VHDL Simulator Reference Manual.* Synopsys, Inc., Nov. 2000.

[120] *Power Compiler Reference Manual.* Synopsys, Inc., Nov. 2000.

[121] N. Takagi and T. Horiyama, "A high-speed reduced-size adder under left-to-right input arrival," *IEEE Trans. Comput.*, vol.48, no.1, pp.76-80, Jan. 1999.

[122] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.17, no.10, pp.1051-1060, Oct. 1998.

[123] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Inc., 1983.

[124] S. Vassiliadis, E.M. Schwarz, and B.M. Sung, "Hard-wired multipliers with encoded partial products," *IEEE Trans. Comput*, vol.40, no.11, pp.1181-1197, Nov. 1991.

[125] C.S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computers*, vol.EC-13, pp.14-17, Feb. 1964.

[126] A. Weinberger, "4:2 carry-save adder module," *IBM Technical Disclosure Bull.*, vol.23, Jan. 1981.

[127] N.H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design.* Addison-Wesley Publishing Company, 1993.

[128] T. Xanthopoulos and A.P. Chandrakasan, "A low-power IDCT macrocell for MPEG-2 MP@ML exploiting data distribution properties for minimal activity," *IEEE J. Solid-State Circuits*, vol.34, No.5, pp.693-703, 1999.

[129] Wu Ye and M.J. Irwin, "Power analysis of gated pipeline registers," in *12th Annual IEEE Int. ASIC/SOC Conf*, pp.281-285, 1999.

[130] G.K. Yeap, *Practical Low Power Digital VLSI Design.* Kluwer Academic Publishers, 1998.

[131] W.-C. Yeh and C.-W. Jen, "High-speed Booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol.49, no.7, pp.692-701, July 2000.

[132] W.-C. Yeh, *Arithmetic Module Design and its Application to FFT.* Ph.D. dissertation, National Chiao-Tung University, 2001.

[133] Z. Yu, L. Wasserman, and A.N. Willson, Jr. "A painless way to reduce power dissipation by over 18% in Booth-encoded carry-save array multipliers for DSP," in *Proc. 2000 IEEE Workshop on Sigal Processing Systems*, pp.571-580, Oct. 2000.

[134] Z. Yu, *et. al.*, "The use of reduced two's-complement representation in low-power DSP design," in *Proc. 2002 IEEE Int. Symp. Circuits and Systems*, vol.1, pp.77-80, May 2002.

[135] Z. Yu, *The Use of Signal Representations in Synthesis and Low-Power Designs for Data-Path Circuits*. Ph.D. dissertation, University of California, Los Angeles, 2002.

[136] M. Zheng and A. Albicki, "Low power and high speed multiplication design through mixed number representations," in *Proc. 1995 IEEE Int. Conf. Computer Design*, pp.566-570, Oct. 1995.

[137] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," *IEEE J. Solid-State Circuits*, vol.32, no.7, pp.1079-1090, July 1997.

[138] R. Zimmermann, *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. Ph.D. dissertation, Swiss Federal Institute of Technology, Zurich, 1997.