Novel modulo multipliers for moduli 2^n-1 , 2^n and 2^{n+1} .

Ramya Muralidharan

2012

Ramya M. (2012). Novel modulo multipliers for moduli 2^n-1 , 2^n and 2^{n+1} . Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/50689>

<https://doi.org/10.32657/10356/50689>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**NOVEL MODULO MULTIPLIERS FOR
MODULI 2^N-1 , 2^N AND 2^N+1**

**RAMYA MURALIDHARAN
SCHOOL OF ELECTRICAL AND ELECTRONIC
ENGINEERING**

2012

**NOVEL MODULO MULTIPLIERS FOR
MODULI 2^N-1 , 2^N AND 2^N+1**

RAMYA MURALIDHARAN

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2012

Acknowledgement

First and foremost, I would like to thank Associate Professor Chang Chip-Hong for his expert guidance and endless support during my Ph. D. candidature at the School of Electrical and Electronic Engineering of Nanyang Technological University. His enthusiasm for quality academic research and his high standards motivated me to strive hard throughout the duration of my Ph. D. I am indebted to him for many insightful discussions and constructive criticism which were instrumental in shaping this thesis. I sincerely appreciate the effort and time he spent in reviewing this thesis and the manuscripts of our publications.

I would also like to thank Mrs. Chang specially for organizing our trip to attend the IEEE International Symposium on Circuits and Systems (ISCAS) every year and several other social gatherings.

I would like to express my gratitude to Professor Thambipillai Srikanthan, Ms. Nah Kiat Joo, Mr. Chua Ngee Tat and Ms. Merilyn Yap of the Centre for High Performance Embedded System (CHiPES) for their assistance. I would like to thank Associate Professor Jong Ching Chuen for the opportunity to work with him on a related project for a year.

Special thanks to my friends for the good times we shared in Singapore. Their continued encouragement and personal support was invaluable to me.

Table of Contents

Acknowledgement.....	i
Table of Contents.....	ii
Summary.....	vii
List of Figures.....	ix
List of Tables.....	xii
List of Acronyms.....	xv
Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Research objectives.....	3
1.3 Major contributions.....	4
1.4 Organization of the thesis.....	5
Chapter 2 Background and Literature Survey.....	8
2.1 Overview of Residue Number System.....	8
2.2 Binary-to-residue converter.....	11
2.2.1 Binary-to residue modulo 2^n-1 converter.....	12
2.2.2 Binary-to residue modulo 2^n converter.....	13
2.2.3 Binary-to residue modulo 2^n+1 converter.....	13
2.3 Residue-to-binary converter.....	14

2.3.1 Residue-to-binary converter for special moduli set.....	17
2.4 Residue arithmetic units.....	17
2.4.1 Modulo m adder.....	18
2.4.2 Modulo 2^n-1 adder.....	19
2.4.2.1 Parallel prefix modulo 2^n-1 adder.....	20
2.4.2.2 Parallel prefix modulo 2^n-1 adder with unrolled c_{out}	23
2.4.2.3 Parallel prefix modulo 2^n-1 adder with Ling carry.....	24
2.4.2.4 Single representation of zero in modulo 2^n-1 adder.....	27
2.4.2.5 Multi-operand modulo 2^n-1 adder (MOMA, 2^n-1).....	27
2.4.3 Modulo 2^n+1 adder.....	29
2.4.3.1 Parallel prefix modulo 2^n+1 adder.....	31
2.4.3.2 Parallel prefix modulo 2^n+1 adder with unrolled c_{out}	31
2.4.3.3 Parallel prefix modulo 2^n+1 adder with Ling carry.....	34
2.4.3.4 Handling zero in modulo 2^n+1 adder.....	36
2.4.3.5 Multi-operand modulo 2^n+1 adder (MOMA, 2^n+1).....	37
2.4.4 Modulo m multiplier.....	39
2.4.5 Modulo 2^n-1 multiplier.....	41
2.4.5.1 Non-encoded modulo 2^n-1 multiplier.....	41
2.4.5.2 Radix-4 Booth encoded modulo 2^n-1 multiplier.....	42
2.4.6 Modulo 2^n+1 multiplier.....	45
2.4.6.1 Non-encoded modulo 2^n+1 multiplier.....	46
2.4.6.2 Radix-4 Booth encoded modulo 2^n+1 multiplier.....	49

Chapter 3 Radix-8 Booth Encoded Modulo 2^n-1 Multiplier for Imbalanced Word-length moduli set based RNS.....	53
3.1 Introduction.....	53
3.2 Radix-8 Booth encoded modulo 2^n-1 multiplication algorithm.....	55
3.3 Proposed radix-8 Booth encoded modulo 2^n-1 multiplier.....	58
3.3.1 Generation of partially-redundant and biased hard multiple.....	58
3.3.2 Generation of partially-redundant and biased simple multiples.....	60
3.3.3 Generation of partially-redundant and biased partial products.....	61
3.3.3.1 Computation of Compensation Constant (CC).....	61
3.3.3.2 Generation of PP_i s.....	68
3.3.4 Accumulation of partial products.....	70
3.4 Performance analysis.....	71
3.5 Summary.....	75
 Chapter 4 Radix-8 Booth Encoded Modulo 2^n-1 and Modulo 2^n+1 Multipliers for Balanced Word-length moduli set based RNSs.....	 77
4.1 Introduction.....	77
4.2 Radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multiplication algorithms.....	78
4.3 Proposed modulo 2^n-1 and modulo 2^n+1 Hard Multiple Generators (HMGs).....	81
4.3.1 Modulo 2^n-1 HMG.....	81
4.3.2 Modulo 2^n+1 HMG.....	84
4.4 Proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers.....	88
4.4.1 Modulo 2^n-1 multiplier.....	89
4.4.1.1 Generation of partial products.....	89

4.4.1.2 Accumulation of partial products.....	90
4.4.2 Modulo 2^n+1 multiplier.....	91
4.4.2.1 Generation of partial products.....	91
4.4.2.2 Generation of bias K_i	91
4.4.2.3 Accumulation of partial products.....	95
4.5 Performance analysis.....	96
4.6 Summary.....	101
Chapter 5 Radix-4 and Radix-8 Booth Encoded Multi-modulus Multipliers.....	102
5.1 Introduction.....	102
5.2 Preliminaries.....	103
5.3 Radix- 2^2 Booth encoded multi-modulus multiplier.....	104
5.3.1 Multi-modulus partial product generation.....	104
5.3.2 Multi-modulus bias generation.....	109
5.3.3 Multi-modulus partial product accumulation.....	112
5.4 Radix- 2^3 Booth encoded multi-modulus multiplier.....	113
5.4.1 Multi-modulus partial product generation.....	113
5.4.2 Multi-modulus hard multiple generation.....	116
5.4.3 Multi-modulus bias generation.....	118
5.4.4 Multi-modulus partial product accumulation.....	121
5.5 Performance analysis.....	122
5.6 Summary.....	125
Chapter 6 Conclusions and Future Work.....	126

6.1 Conclusions.....	126
6.2 Future work.....	128
Author's Publications.....	130
References.....	132

Summary

Long word-length integer multiplication is widely acknowledged as the bottleneck operation in public key cryptographic and signal processing algorithms. Residue Number System (RNS) has emerged as a promising alternative number representation for the design of faster and low power multipliers owing to its merit to distribute a long integer multiplication into several shorter and parallel modulo multiplications. To maximize the advantages offered by the RNS multiplier, judicious choice of moduli that constitute the RNS base and design of efficient modulo multipliers are imperative. In this thesis, special modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers are studied. By manipulating the number theoretic properties of special moduli, 2^n-1 , 2^n and 2^n+1 , new low-power and low-area modulo multipliers are proposed.

The modulo 2^n-1 multiplier is typically the non-critical datapath among all modulo multipliers in the RNS multiplier. This timing slack can be exploited to lower the area as well as power dissipation without compromising the performance of the RNS multiplier. A family of radix-8 Booth encoded modulo 2^n-1 multipliers with delay adaptable to match the RNS delay is proposed. The modulo 2^n-1 multiplier delay is made scalable by controlling the word-length k of the Ripple Carry Adder (RCA) that computes the necessary hard multiple, *i.e.*, three time the multiplicand, of the radix-8 Booth encoding algorithm. The hard multiple and the simple multiples are consistently represented in partially-redundant biased forms. The compensation constant that negates the effect of the biased representation is proven to be a single constant n -bit word for all valid combinations of n and k . The adaptive delay of the modulo 2^n-1 multiplier is corroborated by synthesis results based on CMOS implementations. In an imbalanced word-length moduli set based RNS multiplier, where the critical modulo m multiplier delay is significantly greater than the non-critical modulo 2^n-1 multiplier delay, $k = n$ and $k = n/3$ when n is not divisible by three and divisible by three, respectively, are recommended for maximal area-power savings.

New radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers that are equally applicable in critical and non-critical modulo channels as well as balanced and imbalanced

word-length moduli sets are also proposed. Custom adders called Hard Multiple Generators (HMGs) that exclusively compute the required hard multiples of radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multiplications are designed. The parallel-prefix implementations of the proposed modulo 2^n-1 and modulo 2^n+1 HMGs employ the fewest number of prefix levels and hence are the fastest adders for this application. The modulo-reduced partial products were generated with no accompanying bias in the proposed modulo 2^n-1 multiplier while the inevitable bias was succinctly expressed as three n -bit words in the proposed modulo 2^n+1 multiplier. The savings in area and power dissipation of the proposed radix-8 Booth encoded modulo multipliers over radix-4 Booth encoded and non-encoded modulo multipliers in the $\{2^n-1, 2^n, 2^n+1\}$ based RNS multiplier are substantiated by synthesis results based on CMOS implementations.

Radix-4 and radix-8 Booth encoded modulo 2^n multipliers are introduced. Furthermore, a new radix-4 Booth encoded modulo 2^n+1 multiplier with architecture similar to the corresponding radix-4 Booth encoded modulo 2^n-1 and modulo 2^n multipliers is proposed. The equivalences in modulo negation, modulo reduction of binary weight, modulo multiplication by powers-of-two and two-operand modulo addition for the special moduli, 2^n-1 , 2^n , 2^n+1 are demonstrated. With this correlation among modulo 2^n-1 , modulo 2^n and modulo 2^n+1 operations as the basis, radix-4 and radix-8 Booth encoded multi-modulus multiplier architectures that perform modulo multiplication for the three special moduli successively are developed.

List of Figures

Figure 1.1 Architecture of RNS multiplier.....	2
Figure 2.1 Two's complement adder with $c_{-1} = 0$: (a) Sklansky structure (b) Kogge-Stone structure (c) Implementation of pre-processing, prefix and post-processing operators.....	21
Figure 2.2 Two's complement adder with c_{-1}	22
Figure 2.3 Modulo 2^n-1 adder.....	23
Figure 2.4 Modulo 2^n-1 adder with unrolled c_{out}	24
Figure 2.5 (a) Modulo 2^n-1 adder with Ling carry (b) Implementation of pre-processing and post-processing stages.....	26
Figure 2.6 Example of an 8-bit CSA.....	28
Figure 2.7 Example of an 8-bit EAC-CSA.....	28
Figure 2.8 CSA tree implementation of (MOMA, 2^n-1).....	29
Figure 2.9 Diminished-1 modulo 2^n+1 adder.....	31
Figure 2.10 Diminished-1 modulo 2^n+1 adder with unrolled c_{out}	34
Figure 2.11 Diminished-1 modulo 2^n+1 adder with Ling carry.....	35
Figure 2.12 Example of an 8-bit CEAC-CSA.....	38
Figure 2.13 CSA tree implementation of (MOMA, 2^n+1).....	38
Figure 2.14 MPPG for modulo 2^n-1 multiplier.....	42
Figure 2.15 MPPA for modulo 2^n-1 multiplier.....	42
Figure 2.16 (a) MPPG for radix-4 Booth encoded modulo 2^n-1 multiplier (b) Radix-4 BE (c) Radix-4 BS.....	44

Figure 2.17 MPPA for radix-4 Booth encoded modulo 2^n-1 multiplication.....	45
Figure 2.18 MPPG for modulo 2^n+1 multiplier.....	48
Figure 2.19 MPPA for modulo 2^n+1 multiplier.....	49
Figure 2.20 (a) MPPG for radix-4 Booth encoded modulo 2^n+1 multiplier (b) Radix-4 MBE (c) Radix-4 MBS (d) Radix-4 MBS*	52
Figure 3.1 Generation of hard multiple $ +3X _{2^n-1}$ using n -bit RCAs.....	57
Figure 3.2 Generation of partially-redundant hard multiple $ +3X _{2^n-1}$ using k -bit RCAs.....	58
Figure 3.3 Generation of partially-redundant biased hard multiple $ B+3X _{2^n-1}$ using k -bit RCAs.....	59
Figure 3.4 Generation of partially-redundant biased simple multiples.....	61
Figure 3.5 Modulo 2^n-1 addition of $B_k^0 \parallel B_k^0 \parallel \dots \parallel B_k^0$ and $B_k^1 \parallel B_k^1 \parallel \dots \parallel B_k^1$	64
Figure 3.6 Modulo-reduced partial products and CC in partially-redundant biased representation for $ X \cdot Y _{2^s-1}$	69
Figure 3.7 MPPG for radix-8 Booth encoded modulo 2^n-1 multiplier.....	70
Figure 3.8 (a) Bit-slice of radix-8 Booth Encoder (BE) (b) Bit-slice of radix-8 Booth Selector (BS).....	70
Figure 3.9 MPPA for radix-8 Booth encoded modulo 2^n-1 multiplier.....	71
Figure 4.1 Modulo 2^n-1 Hard Multiple Generator.....	84
Figure 4.2 Modified prefix operator.....	88
Figure 4.3 Modulo 2^n+1 Hard Multiple Generator.....	88
Figure 4.4 MPPG for radix-8 Booth encoded modulo 2^n-1 multiplier.....	89

Figure 4.5 (a) Bit-slice of radix-8 Booth Encoder (BE) (b) Bit-slice of radix-8 Booth Selector (BS).....	90
Figure 4.6 MPPA for radix-8 Booth encoded modulo 2^n-1 multiplier.....	90
Figure 4.7 MPPG for radix-8 Booth encoded modulo 2^n+1 multiplier.....	91
Figure 4.8 MPPA for radix-8 Booth encoded modulo 2^n+1 multiplier.....	95
Figure 5.1 (a) Bit slice of radix- 2^2 Booth Encoder (BE2) (b) Implementation of 3:1 multiplexer MUX3 (c) Multi-modulus radix- 2^2 Booth Encoder.....	106
Figure 5.2 (a) Bit slice of radix- 2^2 Booth Selector (BS2) (b) Multi-modulus generation of PP_i	109
Figure 5.3 Multi-modulus accumulation of partial products.....	113
Figure 5.4 (a) Bit slice of radix- 2^3 Booth Encoder (BE3) (b) Multi-modulus radix- 2^3 Booth Encoder.....	114
Figure 5.5 (a) Bit slice of radix- 2^3 Booth Selector (BS3) (b) Multi-modulus generation of PP_i	116
Figure 5.6 (a) Multi-modulus HMG (b) Circuit implementation of pre-processing, prefix and post-processing operators.....	118
Figure 5.7 Multi-modulus accumulation of partial products.....	121

List of Tables

Table 2.1 Depth of k -operand CSA tree.....	29
Table 2.2 Modulo 2^n-1 reduced partial products.....	41
Table 2.3 Modulo 2^n-1 reduced partial products for radix-4 Booth encoding.....	44
Table 2.4 Modulo 2^n+1 reduced partial products [Wang96b].....	47
Table 2.5 Modulo 2^n+1 reduced partial products [Efst05].....	47
Table 2.6 Modulo 2^n+1 reduced partial products for radix-4 Booth encoding [Sous05].....	50
Table 2.7 Modulo 2^n+1 reduced partial products for radix-4 Booth Encoding [Chen10].....	51
Table 3.1 Modulo 2^n-1 reduced multiples and partial products for radix-8 Booth encoding..	56
Table 3.2 Compensation Constant when n is not divisible by three.....	66
Table 3.3 Compensation Constant when n is divisible by three.....	68
Table 3.4 Synthesis results when n is not divisible by three.....	72
Table 3.5 Synthesis results when n is divisible by three.....	72
Table 3.6 Dynamic and leakage power dissipations when n is not divisible by three.....	73
Table 3.7 Dynamic and leakage power dissipations when n is divisible by three.....	73
Table 3.8 Delay-constrained area and power results of modulo 2^n-1 multipliers for $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ RNSs.....	74
Table 3.9 Delay-constrained area and power results of modulo 2^n-1 multipliers for $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ RNS.....	74
Table 3.10 Normalized area of logic modules.....	75
Table 3.11 Normalized area expressions of multipliers.....	75

Table 3.12 Comparison of normalized area.....	75
Table 4.1 Modulo 2^n-1 reduced partial products for radix-8 Booth encoding.....	81
Table 4.2 Modulo 2^n+1 reduced partial products for radix-8 Booth encoding.....	81
Table 4.3 Modulo 2^n+1 reduced partial products for the encoded multiplier digits ± 3	92
Table 4.4 Dynamic bias for the encoded multiplier digits.....	92
Table 4.5 Truth table for Boolean functions of a, b, c, d, e	93
Table 4.6 Area and delay evaluation of modulo 2^n-1 multipliers.....	97
Table 4.7 Power dissipation evaluation of modulo 2^n-1 multipliers.....	97
Table 4.8 Area and delay evaluation of modulo 2^n+1 multipliers.....	97
Table 4.9 Power dissipation evaluation of modulo 2^n+1 multipliers.....	97
Table 4.10 Area comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1	98
Table 4.11 Total power dissipation comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1	98
Table 4.12 Delay comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1	99
Table 4.13 Normalized area of logic modules.....	99
Table 4.14 Normalized area expressions of modulo 2^n-1 multipliers.....	100
Table 4.15 Normalized area expressions of modulo 2^n+1 multipliers.....	100
Table 4.16 Normalized area comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1	101
Table 5.1 Radix- 2^2 Booth encoding.....	105
Table 5.2 Modulo m reduced partial products for radix- 2^2 Booth encoding.....	107
Table 5.3 Bias for the modulus 2^n	109

Table 5.4 Dynamic bias for the modulus 2^n+1	110
Table 5.5 Radix- 2^3 Booth encoding.....	113
Table 5.6 Modulo m reduced partial products for radix- 2^3 Booth encoding.....	115
Table 5.7 Bias for the modulus 2^n	119
Table 5.8 Dynamic bias for the modulus 2^n+1	119
Table 5.9 Area, delay and total power dissipation of proposed radix- 2^k Booth encoded multi-modulus multipliers.....	122
Table 5.10 Area of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers.....	123
Table 5.11 Delay of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers.....	123
Table 5.12 Total power dissipation of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers.....	123
Table 5.13 Area, delay and total power dissipation of radix- 2^k Booth encoded $\{2^n-1, 2^n, 2^n+1\}$ based RNS multipliers.....	124
Table 5.14 Percentage savings in area, delay and total power dissipation of proposed multi-modulus multipliers over RNS multipliers.....	124

List of Acronyms

AOI	AND-OR-INVERT
BCD	Binary Coded Decimal
BE	Booth Encoder
BS	Booth Selector
CC	Compensation Constant
CEAC	Complementary-End-Around-Carry
CRT	Chinese Remainder Theorem
CSA	Carry Save Adder
DR	Dynamic Range
EAC	End-Around-Carry
FA	Full Adder
HA	Half Adder
HMG	Hard Multiple Generator
lsb	Least Significant Bit
LUT	Look Up Table
MBE	Modified Booth Encoder
MBS	Modified Booth Selector
MPPA	Modulo-reduced Partial Product Accumulation
MPPG	Modulo-reduced Partial Product Generation
MRC	Mixed Radix Conversion
msb	Most Significant Bit
RCA	Ripple Carry Adder
RNS	Residue Number System

Chapter 1

Introduction

1.1 Motivation

Binary multiplication is a ubiquitous operation in cryptographic cores, graphics and signal processors. Owing to the pervasiveness of this operation, the delay of the binary multiplier frequently constrains the processor speed. Hence, there has been an unending research interest in algorithms and architectures to accelerate multiplications [Boot51], [Macs61], [Wall64], [Dadd65], [Wein81], [Naga90], [Song91], [Oklo96], [Stel98], [Yeh00], [Kang06]. These multiplication acceleration techniques can be broadly classified as: (a) methods to expedite the generation of partial products such as Booth encoding algorithm (b) methods to accelerate the summation of partial products such as counter and compressor tree based accumulation. In contemporary multiplier design, it is customary to employ a hybrid of techniques from both categories. As the dynamic range of signal processing and cryptographic applications is ever increasing, the effectiveness and adequacy of the aforementioned hardware acceleration techniques cannot be guaranteed in very long word-length multiplications of the future.

Residue Number System (RNS), an unconventional and non-weighted number representation, has emerged as a viable solution to implement long multiplications. RNS facilitates design of high speed multipliers by its virtue to decompose an integer multiplication into several small word-length and parallel modulo multiplications [Schi09], [Baja04], [Noza01], [Stou01]. Furthermore, as the modulo multiplications are independent of each other, an error in one residue channel will not be propagated to other channels. This fault tolerance offered by RNS becomes a valuable feature in deep submicron VLSI multipliers at low voltage operation.

Despite the advantages of the RNS based multiplier, its use has been rather restricted. The main barrier in the widespread use of RNS multiplier is the additional hardware required for the conversion between binary number system and RNS as well as the concurrent multiplications in several modulo channels. The decomposition of a binary number into its residues is known as the binary-to-residue or forward conversion. Conversely, the composition of the residue back to a binary number is known as the residue-to-binary or reverse conversion. Thus, a complete RNS multiplier consists of three components: a binary-to-residue converter, parallel modulo multipliers and a residue-to-binary converter as illustrated in Fig. 1.1.

Fig. 1.1 Architecture of RNS multiplier

By employing RNS for applications involving repetitive computations like repeated modulo multiplications in cryptographic algorithm and multiply-add operations in the sum-of-product kernels of signal processing algorithm, the hardware overhead incurred from the one-time forward and reverse conversions can be justified. However, the hardware cost of parallel modulo multiplications is still sizeable. To sustain the competitive advantages of the RNS based multiplier, the research emphasis has shifted markedly to the area-power efficient implementation of concurrent modulo multiplications in recent years.

To this end, techniques such as multi-modulus and multi-function architectures to minimize the hardware redundancy as well as multi-threshold voltage and multi-supply voltage designs to lower the power dissipation have been suggested [Pali99], [Kour10], [Card05]. Such

control techniques are intended for algorithm level design space exploration and are equally applicable to all moduli forms. For architecture level simplification of the modulo multiplier, the form of the modulus is perceived to be a decisive factor. In contrast to general moduli, special moduli of forms 2^n and $2^n \pm 1$ have been found to possess unique number theoretic properties. The full-combinatorial based implementation of modulo multiplier using the properties of special modulo arithmetic have received wide spread attention [Hias92], [Wrzy93], [Wang96a], [Wang96b], [Ma98], [Zimm99], [Efst04a], [Efst05], [Sous05], [Verg07], [Chen10].

While the performance of existing modulo $2^n - 1$ and modulo $2^n + 1$ multipliers is acceptable, it is by no means superlative. There is undeniably room for improvement in the performance metrics of the modulo $2^n - 1$, modulo 2^n and modulo $2^n + 1$ multipliers by the ingenious use of the number theoretic properties of special modulo arithmetic and therein lies the motivation behind this research work.

1.2 Research objectives

The prime objective of this research is to develop efficient architectures for modulo $2^n - 1$, modulo 2^n and modulo $2^n + 1$ multipliers. Firstly, well-established number theoretic properties of modulo arithmetic for special moduli $2^n - 1$, 2^n and $2^n + 1$ will be studied. Existing modulo $2^n - 1$, modulo 2^n and modulo $2^n + 1$ adders and multipliers will be systematically reviewed. The performance critical computations in modulo multiplications will be identified. By capitalizing on the modulo arithmetic properties, new designs for modulo $2^n - 1$, modulo 2^n and modulo $2^n + 1$ multipliers as well as their constituent components will be proposed. The VLSI metrics, *i.e.*, area, delay and total power dissipation of the proposed modulo multipliers will be evaluated for application in RNSs based on imbalanced and balanced word-length special moduli sets.

In order to fulfil the main objective of this research, the following specific issues have been identified and focussed on in the thesis.

- (a) To investigate Booth encoding technique for modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications. In particular radix-4 and radix-8 Booth encoding algorithms will be considered.
- (b) To overcome the modulo-reduced hard multiple generation problem of radix-8 Booth encoding technique.
- (c) To devise ingenious solutions for generating the inevitable bias in modulo 2^n and modulo 2^n+1 multiplications.
- (d) To identify equivalent operations in modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications for exploration of unified multiplier architectures.

1.3 Major contributions

The main contributions of the research work performed are highlighted as follows.

- (a) The first-ever radix-8 Booth encoded modulo 2^n-1 multiplier is proposed for application in the non-critical modulo channel of imbalanced word-length moduli set based RNS multiplier. The non-criticality of the modulo 2^n-1 channel is exploited for area-power savings by intentionally operating the modulo 2^n-1 multiplier at a slower speed that nearly matches RNS multiplier speed. The delay match is achieved by varying the word-length of the small adders that compute the necessary hard multiple of the radix-8 Booth encoded modulo 2^n-1 multiplication in a partially-redundant biased form. Formal criteria for the selection of the adder word-length are established by analyzing its effect on the multiplier delay. By the number theoretic properties of modulo 2^n-1 arithmetic, it is proven that for a given n , there exist a number of feasible values of the adder word-length such that the bias due to the partially redundant biased representation can be counteracted by a single constant n -bit word that can be precomputed at design time.
- (b) Novel radix-8 Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers are proposed for use in non-critical and critical modulo channels as well as balanced and imbalanced word-length moduli sets based RNS multipliers. By reformulating the carry

equations of modulo 2^n-1 , modulo 2^n and modulo 2^n+1 additions with the multiplicand and two times the multiplicand as addends, custom adders that exclusively generate the necessary modulo-reduced hard multiple of the radix-8 Booth encoded multiplication are developed. The proposed custom adders implemented as parallel-prefix structures outperform the generic two-operand modulo adders in area, delay and power dissipation simultaneously. In the proposed modulo 2^n-1 multiplier, no additional bias is incurred, while the aggregate bias is expressed as a single n -bit word in the proposed modulo 2^n multiplier and as three n -bit words in the proposed modulo 2^n+1 multiplier.

(c) New radix-4 Booth encoded modulo 2^n and mod 2^n+1 multipliers with architectures comparable to existing radix-4 Booth encoded modulo 2^n-1 multiplier are proposed. As the baseline modulo 2^n-1 multiplier lacks a bias component, minimizing the hardware overhead in generating and accumulating the inevitable bias in the proposed modulo 2^n and modulo 2^n+1 multipliers is emphasized. The aggregate bias in the proposed radix-4 Booth encoded modulo 2^n and modulo 2^n+1 multiplier is reformulated as a single and two n -bit words, respectively. In both multipliers, the aggregate bias is generated by merely hardwiring the outputs of the Booth encoder blocks.

(d) Multi-modulus multiplier architectures for the special moduli 2^n-1 , 2^n and 2^n+1 using radix-4 as well as radix-8 Booth encoding techniques are developed. By taking advantage of the equivalences in key operations such as negation, reduction of binary weight, multiplication by powers-of-two and two-operand addition among the three moduli, the control circuit required for a unified modulo multiplication is simplified.

1.4 Organization of the thesis

This thesis is organized into six Chapters. In Chapter 1, the motivation, the objective and the key contributions of the research work are detailed.

In Chapter 2, the fundamentals of RNS and modulo arithmetic are described. The three main components of the RNS processor are identified as binary-to-residue converter, modulo arithmetic unit and residue-to-binary converter. The binary-to-residue and residue-to-binary conversion techniques are reviewed for the general as well as special moduli, 2^n-1 , 2^n and

2^n+1 . Addition and multiplication algorithms for modulo arithmetic are presented. The two-operand and multi-operand modulo adders for the special moduli, 2^n-1 and 2^n+1 , are comprehensively surveyed. Subsequently, existing non-encoded and radix-4 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers are also reviewed.

The main contributions of this research are presented in Chapters 3 to 5. In Chapter 3, radix-8 Booth encoded multiplication technique is investigated for modulo 2^n-1 arithmetic. The non-trivial computation of the hard multiple, *i.e.*, three times the multiplicand, is identified as the critical operation. A novel technique to generate the hard multiple in partially redundant and biased representation using small word-length adders is proposed. The simple multiples and thus all modulo-reduced partial products are uniformly generated in the partially redundant and biased forms. The constant that negates the effect of the biased representation is derived and expressed as an n -bit word with a specific repetitive pattern of logic ones and zeros. The proposed hard multiple generation technique is proven to be advantageous in RNS multipliers based on imbalanced word-length moduli sets wherein the modulus 2^n-1 constitutes the non-critical channel. By equalizing the non-critical modulo 2^n-1 multiplier delay to the critical modulo m multiplier delay using adder word-length manipulation, significant reductions in area and power dissipation of the RNS multiplier are demonstrated.

Radix-8 Booth encoded multiplication scheme is extended to modulo 2^n+1 multiplier in Chapter 4. New application specific adders called as Hard Multiple Generators (HMGs) that compute solely the modulo-reduced hard multiple of the radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multiplications are proposed. The generation and accumulation of the $\lfloor n/3 \rfloor + 1$ partial products in the proposed modulo 2^n-1 multiplier are detailed. In the proposed modulo 2^n+1 multiplier, the aggregate bias is derived and expressed as only three partial products. Subsequently, the generation and accumulation of the $\lfloor n/3 \rfloor + 6$ partial products in the proposed modulo 2^n+1 multiplier are described. The savings in area and total power dissipation achieved by the proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers over radix-4 Booth encoded and non-encoded modulo multipliers are demonstrated in the balanced word-length moduli set $\{2^n-1, 2^n, 2^n+1\}$ based RNS multiplier.

In Chapter 5, modulo multiplier that is capable of performing modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications simultaneously or successively are explored. Firstly, new modulo 2^n and modulo 2^n+1 multipliers using radix-4 Booth encoding algorithm are described. Furthermore, a radix-8 Booth encoded modulo 2^n multiplier employing a modulo 2^n HMG is proposed. By identifying equivalent operations among the proposed modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers, radix-4 and radix-8 Booth encoded variable multi-modulus multiplier is proposed. The performance of the proposed multi-modulus multiplier is compared against the conventional single modulus multipliers for $\{2^n-1, 2^n, 2^n+1\}$ based RNS.

Finally, Chapter 6 summarizes the results achieved in this research work and outlines topics that are worthy of further research based on the insights from the content presented in this thesis.

Chapter 2

Background and Literature Survey

2.1 Overview of Residue Number System

An integer number system is defined as a set of integers along with the arithmetic operations that can be performed on the integers. A number system is said to be weighted if there exists a set of weights w_i such that any number X in the system can be represented as

$$X = \sum_{i=1}^n x_i \cdot w_i \quad (2.1)$$

where x_i is the i -th digit from the set of permissible digits. If w_i are successive powers of the same number known as radix, then the number system is a fixed-radix system. Well known examples of weighted fixed-radix systems are decimal system of radix 10 and binary system of radix 2. A number system in which the weights are not successive powers of the radix is a mixed-radix system. An example of the weighted mixed-radix system is the Binary Coded Decimal (BCD) system. The advantages of the weighted decimal and binary systems are: relative magnitude comparison is simplified to digit by digit comparison, scaling by a power of the radix is performed by simple shift operations to the left or right, extending the range of the number system is easily realized by adding more digit positions and overflow detection is easily mechanized.

In both decimal and binary systems, truly parallel arithmetic operation in which all digits are processed concurrently is not feasible as every digit of the result depends on all digits of the operands of equal or lower significance. The limitation on speed of computation due to carry propagation between digits is inherent to weighted number systems. Residue Number System (RNS), a non-weighted number system based on modulo arithmetic, offers an ingenious solution to the carry propagation problem of conventional number system. Arithmetic

operations like addition, subtraction, multiplication, squaring and exponentiation when implemented in RNS can achieve high speed of operation compared to decimal or binary system [Szab67], [Sode86].

RNS is defined by a base that consists of a set of N integers, $\{L_1, L_2, \dots, L_N\}$ where L_i is known as the modulus and the moduli are pair-wise relatively prime. For unambiguous representation, the Dynamic Range (DR) of the RNS is given by the product of all moduli in the base, *i.e.*, $L = \prod_{i=1}^N L_i$. The DR can also be expressed as l bits where $l = \lceil \log_2 L \rceil$ and $\lceil a \rceil$ is the smallest integer greater than or equal to a . An integer X within the DR is represented in RNS by a set of N residues $\{x_1, x_2, \dots, x_N\}$, where x_i is the residue of X modulo L_i . x_i is also known as the i -th residue digit of X and can be expressed as

$$\begin{aligned} x_i &= |X|_{L_i}, i = 1, 2 \dots N \\ &= X - q_i \cdot L_i, i = 1, 2 \dots N \end{aligned} \quad (2.2)$$

where q_i and x_i are the quotient and remainder from the division of X by L_i . x_i can only take values from the set $[0, L_i - 1]$.

For RNS of base $\{L_1, L_2, \dots, L_N\}$, let $X = \{x_1, x_2, \dots, x_N\}$ and $Y = \{y_1, y_2, \dots, y_N\}$ be the residue representation of the operands. Then the residue representation of the result from the arithmetic operation $Z = X \circ Y$ is given by

$$\{z_1, z_2, \dots, z_N\} = \{|x_1 \circ y_1|_{L_1}, |x_2 \circ y_2|_{L_2}, \dots, |x_N \circ y_N|_{L_N}\} \quad (2.3)$$

where ‘ $\circ$ ’ can be operations such as addition, subtraction, multiplication, squaring and exponentiation. It can be observed that the i -th residue digit of Z depends on only the i -th residue digits of X and Y . The operation $x_i \circ y_i$ is performed in a unit corresponding to the modulus L_i (also known as modulo channel). As there is no carry-propagation between the modulo channels, the arithmetic operation can be performed in parallel in the N modulo channels independently. Since the residue digits x_i s are considerably smaller than X , the modulo channel operates on reduced word-length operands. The reduced length of intra-channel carry propagation chain and the absence of inter-channel carry propagation lead to faster computation in RNS when compared to its decimal and binary system counterparts.

Operations in each modulo channel are based on modulo (also known as modular or residue) arithmetic. Key identities of residue arithmetic that are recurrent in this thesis are summarized below.

$$|L \cdot x|_L = 0 \quad (2.4)$$

$$|x \pm y|_L = ||x|_L \pm |y|_L|_L \quad (2.5)$$

$$|x \cdot y|_L = ||x|_L \cdot |y|_L|_L \quad (2.6)$$

$$|-x|_L = |L - x|_L \quad (2.7)$$

where $|L - x|_L$ is called the additive inverse of x modulo L .

The multiplicative inverse of x modulo L is defined as $|x^{-1}|_L$ such that $|x \cdot x^{-1}|_L = 1$.

Since binary system is the predominant number system employed in digital applications, a RNS based implementation consists of three main components, *i.e.*, binary-to-residue converter, residue-to-binary converter and residue arithmetic units. The selection of the moduli that comprise the base is crucial to the performance and hardware complexity of the RNS based implementation. The moduli can be categorized as general and special moduli. The former encompasses moduli of no specific form while the latter refers to moduli of forms 2^n-1 , 2^n and 2^n+1 , which possess good number theoretic properties for efficient implementations of modulo operations. Various moduli sets based on special moduli 2^n , 2^n-1 and 2^n+1 have been suggested in literature. These moduli sets can be classified based on their cardinality as: (a) Three-moduli sets, such as $\{2^n-1, 2^n, 2^n+1\}$, $\{2^n, 2^n-1, 2^{n-1}-1\}$ and $\{2^n, 2^n-1, 2^{n+1}-1\}$; (b) Four-moduli sets, such as $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$; (c) High cardinality moduli sets (cardinality greater than four), such as $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$. Moduli set of cardinality greater than three of the form $\{2^n-1, 2^n, 2^n+1, m_i, \dots, m_j\}$ that contains the standard three moduli set, $\{2^n-1, 2^n, 2^n+1\}$ as its subset is known as a superset. The word-length of the modulus is defined as the number of bits required for the representation of the residues of the modulus. Based on the word-length

of the constituent moduli, the moduli sets are categorized as (a) Balanced word-length moduli sets like $\{2^n-1, 2^n, 2^n+1\}$, where the word-length of each moduli is n bits; (b) Imbalanced word-length moduli sets like $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, where the word-length of only the modulus $2^{2n}+1$ is $2n$ bits. Furthermore, the moduli sets can be grouped on the basis of their DR as: (a) $3n$ -bit DR moduli sets, such as $\{2^n-1, 2^n, 2^n+1\}$, $\{2^n, 2^n-1, 2^{n-1}-1\}$ and $\{2^n, 2^n-1, 2^{n+1}-1\}$; (b) $4n$ -bit DR moduli sets, such as $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$; (c) High DR moduli sets, such as $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ with $5n$ -bit DR and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ with $6n$ -bit DR.

The special moduli possess number theoretic properties that facilitate design of efficient binary-to-residue converter, residue-to-binary converter as well as residue arithmetic units. In the following, the three components of a RNS based implementation are described with emphasis on the special moduli.

2.2 Binary-to residue-converter

In the binary-to-residue converter, also known as forward converter, the operands represented in binary system are converted into their residue representation. The conversion of operand X from binary to residue representation is given by (2.2). The residue digit x_i corresponding to each L_i can be computed in parallel. There are three main approaches to forward conversion. In the first approach, all values required by the conversion are precomputed and stored in memory or Look Up Tables (LUTs) [Parh94]. The second approach involves the use of arithmetic units along with smaller memory. In both these techniques, the size of the memory grows exponentially with the word-length of the moduli. The last and recent approach is memoryless and uses only arithmetic circuits [Prem02], [Prem06]. In [Pies91], [Pies94], [Pies02] and [Verg10], binary to residue conversion was simplified to multi-operand modulo addition using the periodicity of modulo-reduced powers-of-two series. The periodicity properties for the special moduli, 2^n-1 and 2^n+1 are expressed as *Properties 2.1* and *2.2*, respectively. In addition, *Properties 2.3* and *2.4* are the simplified expressions for modulo 2^n-1 and modulo 2^n+1 negations, respectively.

Property 2.1: As $\left|2^n\right|_{2^n-1} = 1$, for any non-negative integers i and k ,

$$\left|2^{k \cdot n + i}\right|_{2^n-1} = \left|2^i\right|_{2^n-1} \quad (2.8)$$

Property 2.2: As $\left|2^n\right|_{2^n+1} = -1$, for any non-negative integers i and k ,

$$\left|2^{k \cdot n + i}\right|_{2^n+1} = \begin{cases} \left|2^i\right|_{2^n+1} & \text{if } k \text{ is even} \\ \left|-2^i\right|_{2^n+1} & \text{if } k \text{ is odd} \end{cases} \quad (2.9)$$

Property 2.3: Let $\bar{X}$ denote the one's complement of X . By the definition of additive inverse in (2.7), modulo 2^n-1 negation is given by

$$\left|-X\right|_{2^n-1} = \left|2^n-1-X\right|_{2^n-1} = \bar{X} \quad (2.10)$$

Property 2.4: From (2.7), modulo 2^n+1 negation is given by

$$\left|-X\right|_{2^n+1} = \left|2^n+1-X\right|_{2^n+1} = \bar{X} + 2 \quad (2.11)$$

2.2.1 Binary-to-residue modulo 2^n-1 converter

Let $X_{m-1:0}$ be the m -bit binary operand in excess of the modulus, i.e., $m > n$. Starting from the least significant bit (lsb), the m bits of $X_{m-1:0}$ are partitioned into groups of n bits, i.e., $X_{m-1:\lfloor m/n \rfloor n}, \dots, X_{3n-1:2n}, X_{2n-1:n}, X_{n-1:0}$. If m is not divisible by n , then the most significant bit (msb) positions are padded with $\lfloor m/n \rfloor n + n - m$ zeros so that the last group is also of n bits. The residue modulo 2^n-1 becomes

$$\left|X_{m-1:0}\right|_{2^n-1} = \left|X_{m-1:\lfloor m/n \rfloor n} \cdot 2^{\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} \cdot 2^{2n} + X_{2n-1:n} \cdot 2^n + X_{n-1:0} \cdot 2^0\right|_{2^n-1} \quad (2.12)$$

On simplifying the powers-of-two terms using *Property 2.1*, (2.12) becomes

$$\left|X_{m-1:0}\right|_{2^n-1} = \left|X_{m-1:\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} + X_{2n-1:n} + X_{n-1:0}\right|_{2^n-1} \quad (2.13)$$

Equation (2.13) can be efficiently implemented in hardware using a Multi-Operand Modulo 2^n-1 Adder denoted as (MOMA, 2^n-1) with $\lfloor m/n \rfloor + 1$ operands.

As an example, let $X_{m-1:0}$ be $127180_{10} = 11111000011001100_2$ and the modulus be $2^4-1 = 15$. $X_{m-1:0}$ is partitioned into five groups, $X_{3:0} = 1100_2 = 12_{10}$, $X_{7:4} = 1100_2 = 12_{10}$, $X_{11:8} = 0000_2 = 0_{10}$, $X_{15:12} = 1111_2 = 15_{10}$, $X_{16} = 0001_2 = 1_{10}$. The residue is given by the modulo reduced sum of the five groups, *i.e.*, $|12+12+0+15+1|_{15} = 10_{10}$.

2.2.2 Binary-to-residue modulo 2^n converter

The forward conversion for the modulus 2^n is achieved by simply discarding the bits of binary weight greater than 2^{n-1} . The conversion can be expressed mathematically as

$$|X_{m-1:0}|_{2^n} = |X_{m-1:\lfloor m/n \rfloor n} \cdot 2^{\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} \cdot 2^{2n} + X_{2n-1:n} \cdot 2^n + X_{n-1:0} \cdot 2^0|_{2^n} \quad (2.14)$$

By simplifying (2.14) using (2.4),

$$|X_{m-1:0}|_{2^n} = |X_{m-1:\lfloor m/n \rfloor n} \cdot 0 + \dots + X_{3n-1:2n} \cdot 0 + X_{2n-1:n} \cdot 0 + X_{n-1:0} \cdot 2^0|_{2^n+1} = |X_{n-1:0}|_{2^n} \quad (2.15)$$

Hence, the residue of $X_{m-1:0}$ modulo 2^n is the least significant n bits of $X_{m-1:0}$.

Consider the example of $X_{m-1:0} = 127180_{10} = 11111000011001100_2$ and the modulus of $2^4 = 16$. The residue is equivalent to the least significant four bits, *i.e.*, $1100_2 = 12_{10}$.

2.2.3 Binary-to-residue modulo 2^n+1 converter

The m bits of $X_{m-1:0}$ are partitioned into groups of n bits, *i.e.*, $X_{m-1:\lfloor m/n \rfloor n}, \dots, X_{3n-1:2n}, X_{2n-1:n}, X_{n-1:0}$, beginning from the lsb while padding the msb positions with necessary zeros. The residue modulo 2^n+1 is given by

$$|X_{m-1:0}|_{2^n+1} = |X_{m-1:\lfloor m/n \rfloor n} \cdot 2^{\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} \cdot 2^{2n} + X_{2n-1:n} \cdot 2^n + X_{n-1:0} \cdot 2^0|_{2^n+1} \quad (2.16)$$

By *Property 2.2*, the powers-of two are modulo reduced leading to

$$|X_{m-1:0}|_{2^n+1} = \begin{cases} \left| X_{m-1:\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} - X_{2n-1:n} + X_{n-1:0} \cdot 2^0 \right|_{2^n+1} & \text{if } \lfloor m/n \rfloor \text{ is even} \\ \left| -X_{m-1:\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} - X_{2n-1:n} + X_{n-1:0} \cdot 2^0 \right|_{2^n+1} & \text{if } \lfloor m/n \rfloor \text{ is odd} \end{cases} \quad (2.17)$$

Using *Property 2.4*, the negative term in (2.17) is simplified to a one's complemented vector with a correction bias of two.

$$|X_{m-1:0}|_{2^n+1} = \begin{cases} \left| X_{m-1:\lfloor m/n \rfloor n} + \dots + X_{3n-1:2n} + (\bar{X}_{2n-1:n} + 2) + X_{n-1:0} \cdot 2^0 \right|_{2^n+1} & \text{if } \lfloor m/n \rfloor \text{ is even} \\ \left| (\bar{X}_{m-1:\lfloor m/n \rfloor n} + 2) + \dots + X_{3n-1:2n} + (\bar{X}_{2n-1:n} + 2) + X_{n-1:0} \cdot 2^0 \right|_{2^n+1} & \text{if } \lfloor m/n \rfloor \text{ is odd} \end{cases} \quad (2.18)$$

The residue is computed as the sum of $\lfloor m/n \rfloor + 1$ n -bit binary vectors such that the odd-indexed vectors are inverted and a correction bias of two is added for each inverted vector. The summation is performed by a (MOMA, 2^n+1).

As an example, let $X_{m-1:0}$ be $127180_{10} = 11111000011001100_2$ and the modulus be $2^4+1 = 17$. $X_{m-1:0}$ is partitioned into five binary vectors, $X_{3:0} = 1100_2 = 12_{10}$, $X_{7:4} = 1100_2 = 12_{10}$, $X_{11:8} = 0000_2 = 0_{10}$, $X_{15:12} = 1111_2 = 15_{10}$, $X_{16} = 0001_2 = 1_{10}$. The odd-indexed vectors are inverted, *i.e.*, $\bar{X}_{7:4} = 0011_2 = 3_{10}$ and $\bar{X}_{15:12} = 0000_2 = 0_{10}$. The residue is given by the modulo reduced sum of the three even-indexed vectors, the two inverted odd-indexed vectors and a correction bias of four, *i.e.*, $|12+3+0+0+1+4|_{17} = 3_{10}$.

2.3 Residue-to-binary converter

In the residue-to-binary converter, also known as reverse converter, the operand represented in RNS is converted into binary system. Unlike the results of forward conversion and residue arithmetic operations that depend on only the modulus L_i , the result of reverse conversion depends on all the moduli L_1 to L_N of the base. The two classical approaches to converting a number from its residue form to binary form are Chinese Remainder Theorem (CRT) and

Mixed Radix Conversion (MRC). The binary number X of residue representation $\{x_1, x_2, \dots, x_N\}$ in RNS $\{L_1, L_2, \dots, L_N\}$ is derived using CRT as

$$X = \left| \sum_{i=1}^N \hat{L}_i \cdot x_i \cdot \left| \frac{1}{\hat{L}_i} \right|_{L_i} \right|_L \quad (2.19)$$

where

$$L = \prod_{i=1}^N L_i \text{ and } \hat{L}_i = \frac{L}{L_i} \quad (2.20)$$

The term $\left| \frac{1}{\hat{L}_i} \right|_{L_i}$ is the multiplicative inverse of $\hat{L}_i$ modulo L_i . The advantage of the CRT is that

the partial sum, $\hat{L}_i \cdot x_i \cdot \left| \frac{1}{\hat{L}_i} \right|_{L_i}$ can be computed in parallel and added before the modulo L

reduction. On the downside, the modulo L reduction of the sum can be cumbersome.

On the other hand, the MRC technique eliminates the final modulo reduction step of CRT while being implemented in a sequential approach. The binary number X of residue representation $\{x_1, x_2, \dots, x_N\}$ in RNS $\{L_1, L_2, \dots, L_N\}$ can be represented in the mixed-radix form as

$$X = a_1 + a_2 L_1 + a_3 L_1 L_2 + \dots + a_N \prod_{i=1}^{N-1} L_i \quad (2.21)$$

where a_i is the mixed-radix coefficient. The a_i s are determined one digit at a time starting from a_1 as shown in (2.22).

$$\begin{aligned}
a_1 &= x_1 \\
a_2 &= \left| \left(x_2 - a_1 \right) \left| \frac{1}{L_1} \right|_{L_2} \right|_{L_2} \\
a_3 &= \left| \left(\left(x_3 - a_1 \right) \left| \frac{1}{L_1} \right|_{L_3} - a_2 \right) \left| \frac{1}{L_2} \right|_{L_3} \right|_{L_3} \\
&\dots \\
a_n &= \left| \left(\left(\left(x_n - a_1 \right) \left| \frac{1}{L_1} \right|_{L_n} - a_2 \right) \left| \frac{1}{L_2} \right|_{L_n} - \dots - a_{n-1} \right) \left| \frac{1}{L_{n-1}} \right|_{L_n} \right|_{L_n}
\end{aligned} \tag{2.22}$$

Improved conversion algorithms, namely new CRT-I and new CRT-II, have been proposed in [Wang98] and [Wang00]. Using new CRT-I, the binary number X of the residue representation $\{x_1, x_2, \dots, x_N\}$ in RNS $\{L_1, L_2, \dots, L_N\}$ is computed as

$$X = x_1 + L_1 \left| k_1 (x_2 - x_1) + k_2 L_2 (x_3 - x_2) + \dots + k_{n-1} L_2 L_3 \dots L_{n-1} (x_n - x_{n-1}) \right|_{L_2 L_3 \dots L_N} \tag{2.23}$$

where

$$\begin{aligned}
&\left| k_1 L_1 \right|_{L_2 L_3 \dots L_n} = 1 \\
&\left| k_2 L_1 L_2 \right|_{L_3 L_4 \dots L_n} = 1 \\
&\dots \\
&\left| k_{n-1} L_1 L_2 \dots L_{n-1} \right|_{L_n} = 1
\end{aligned} \tag{2.24}$$

The binary number X of the residue representation $\{x_1, x_2, \dots, x_N\}$ in RNS $\{L_1, L_2, \dots, L_N\}$ is computed in new CRT – II using the algorithm, *translate* as shown below.

Algorithm *translate* $((x_1, x_2, \dots, x_N), X)$

(1) If $n > 2$, let $t = \lfloor n/2 \rfloor$, then

translate $((x_1, \dots, x_t), N_1)$, $M_1 = L_1 \dots L_t$

translate $((x_{t+1}, \dots, x_N), N_2)$, $M_2 = L_{t+1} \dots L_N$

findno (N_1, N_2, M_1, M_2, X)

(2) If $n = 2$, then *findno* (x_1, x_2, L_1, L_2, X)

Procedure *findno* (x_1, x_2, L_1, L_2, X)

- (1) Find a k_0 such that $k_0 \cdot L_2 = 1 \bmod L_1$
- (2) $X = x_2 + L_2 \left\lfloor k_0 (x_1 - x_2) \right\rfloor_{L_1}$

2.3.1 Residue-to-binary converter for special moduli set

Efficient memoryless residue-to-binary converters for the ubiquitous three moduli set $\{2^n-1, 2^n, 2^n+1\}$ have been proposed in [Andr88], [Pies95], [Dhur98], [Bhar98] and [Wang02]. In [Hias98], a reverse converter for the three-moduli set $\{2^n, 2^n-1, 2^{n-1}-1\}$ that avoids the use of the modulus 2^n+1 was suggested. Reverse converters for a similar three moduli set $\{2^n, 2^n-1, 2^{n+1}-1\}$ were proposed in [Math00] and [Moha07c].

In [Bhar99], the four-moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$ consisting of two moduli of the form 2^n+1 was proposed. A more efficient four-moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ was proposed in [Vino00]. Reverse converters suggested in [Bhar99] and [Vino00] were improved in [Cao05] and [Moha07b] by employing the best available reverse converter for the subset, $\{2^n-1, 2^n, 2^n+1\}$ followed by applying the MRC technique for the result and the remaining residues. A similar technique was adopted in the design of residue-to-binary converter for the analogous four-moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$ in [Cao05] and for the five moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ in [Cao07]. Reverse converters for the imbalanced word-length moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ based on new CRT – I and for $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ as well as $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ based on new CRT – II were proposed in [Cao03] and [Mola10], respectively.

2.4 Residue arithmetic units

RNS is frequently used for applications involving repeated addition and multiplication. LUT based implementations of modulo adder and multiplier were presented for small word-length modulus prior to the advent of VLSI technologies. Such LUT based techniques are not ideal for modern applications of high dynamic range due to the exponential increase in the size and

cost of the required tables. Full combinatorial circuits have become the standard in the design of modulo adders and multipliers at present.

2.4.1 Modulo m adder

Modulo adders for general moduli based on two's complement adders were proposed in [Bayo87], [Dugd92], [Hias02]. Let m be the modulus of word-length n , *i.e.*, $n = \lceil \log_2 m \rceil$.

The modulo m addition of n -bit addends, X and Y , can be expressed mathematically as

$$|S|_m = \begin{cases} X + Y & \text{if } X + Y < m \\ X + Y - m & \text{if } X + Y \geq m \end{cases} \quad (2.25)$$

As $m < 2^n$, Z is defined as $2^n - m$. Then, (2.25) is equivalent to

$$|S|_m = \begin{cases} |X + Y|_{2^n} & \text{if } X + Y < m \\ |X + Y + Z|_{2^n} & \text{if } m \leq X + Y < 2^n \\ |X + Y + Z|_{2^n} & \text{if } X + Y \geq 2^n \end{cases} \quad (2.26)$$

A direct implementation of (2.26) uses two two's complement adders: one adder computes the sum of X and Y while the other adder computes the sum of X , Y and Z . The sum outputs of both adders are connected to a multiplexer. The logical disjunction of the carry-out of the two adders selects the correct sum [Bayo87]. A two-cycle implementation using one two's complement adder and a feedback register was detailed in [Dugd92]. In the first cycle of addition, X and Y are selected as the addends, and the sum as well as the carry-out are registered. In the second cycle of addition, Z and the sum output from the first cycle are selected as addends. It must be pointed out that the area of [Bayo87] and the delay of [Dugd92] are nearly twice those of the corresponding two's complement adder.

By using the number theoretic properties of modulo $2^n - 1$ and modulo $2^n + 1$ arithmetic, various modulo adders with area-time complexity similar to a two's complement adder have been proposed in literature [Efst94], [Zimm99], [Kala00], [Verg02], [Dimi03], [Efst04b], [Dimi05b], [Verg08], [Verg09].

2.4.2 Modulo 2^n-1 adder

In modulo 2^n-1 arithmetic, a dual representation of zero is commonly employed. As 2^n-1 is congruent to zero modulo 2^n-1 , zero is represented by an n -bit binary string of all zeros or all ones. Modulo 2^n-1 addition of n -bit addends, X and Y , can be formulated as

$$\begin{aligned} |S|_{2^n-1} &= |X+Y|_{2^n-1} \\ &= \begin{cases} X+Y & \text{if } X+Y < 2^n-1 \\ X+Y-2^n+1 & \text{if } X+Y \geq 2^n-1 \end{cases} \end{aligned} \quad (2.27)$$

When $X+Y=2^n-1$, $|S|_{2^n-1}=0=2^n-1=X+Y$. Therefore, (2.27) can be rewritten as

$$\begin{aligned} |S|_{2^n-1} &= \begin{cases} X+Y & \text{if } X+Y \leq 2^n-1 \\ X+Y-2^n+1 & \text{if } X+Y > 2^n-1 \end{cases} \\ &= \begin{cases} X+Y & \text{if } X+Y < 2^n \\ X+Y-2^n+1 & \text{if } X+Y \geq 2^n \end{cases} \\ &= \begin{cases} |X+Y|_{2^n} & \text{if } X+Y < 2^n \\ |X+Y+1|_{2^n} & \text{if } X+Y \geq 2^n \end{cases} \\ &= |X+Y+c_{out}|_{2^n} \end{aligned} \quad (2.28)$$

where c_{out} is the carry-out from the n -bit addition of X and Y [Zimm99]. Hence, a modulo 2^n-1 addition is equivalent to an n -bit end-around-carry (EAC) addition.

As an example, consider the modulus $2^4-1=15_{10}$ and the addends $X=8_{10}=1000_2$ and $Y=12_{10}=1100_2$. The 4-bit addition of X and Y results in $sum=0100_2$ and $c_{out}=1_2$. $|X+Y|_{15}$ is then given by the addition of sum and c_{out} , *i.e.*, $0101_2=5_{10}$.

The straightforward implementations of EAC addition include: (a) A two cycle implementation where the sum and the carry-out from the first cycle addition of X and Y are added in the second cycle; (b) A single cycle implementation where two adders are used to compute $X+Y$ and $X+Y+1$, and the correct sum is selected in a multiplexer. Furthermore, high-speed and reduced-area modulo 2^n-1 adders have been proposed in [Efst94], [Zimm99], [Kala00], [Dimi03], [Dimi05b]. In [Efst94], modulo 2^n-1 adders based on one-level and two-level Carry-Look-Ahead addition algorithms were proposed. Direct implementation of EAC

addition using either two CLA adders or two cycles was considered. Furthermore, the two-step EAC addition was replaced with a single step addition by unrolling the equation of the carry-out, c_{out} . By considering the term c_{out} as the carry-in to the adder, faster and better structured implementations were suggested [Efst94]. Fast modulo 2^n-1 adders were proposed by treating the carry propagation as a prefix problem in [Zimm99], [Kala00], [Dimi03], [Dimi05b].

2.4.2.1 Parallel prefix modulo 2^n-1 adder

The carry computation in two's complement addition of X and Y is a classic prefix problem as the carry c_i from the bit position i is a function of all inputs $x_j \in [x_i, \dots, x_0]$ and $y_j \in [y_i, \dots, y_0]$ such that $j \leq i$, as shown in (2.29).

$$c_i = x_i \cdot y_i + (x_i + y_i) \cdot c_{i-1} \quad (2.29)$$

The sum computation using the carry equation (2.29) is implemented in three stages: pre-processing, prefix computation and post-processing stages. The computation in each stage is given below. In the following analysis, the carry-in, c_{-1} is assumed to be zero.

Pre-processing stage: For $i = 0$ to $n-1$,

$$\begin{aligned} g_i &= x_i \cdot y_i \\ p_i &= x_i + y_i \\ h_i &= x_i \oplus y_i \end{aligned} \quad (2.30)$$

where g_i , p_i and h_i are the generate, propagate and half-sum bits, respectively, at bit position i .

Prefix computation stage: For $i = 0$ to $n-1$, the carry $c_i = G_i$ is computed as

$$\begin{aligned} (G_i, P_i) &= \begin{cases} (g_0, p_0) & \text{if } i = 0 \\ (g_i, p_i) \bullet (G_{i-1}, P_{i-1}) & \text{if } 1 \leq i \leq n-1 \end{cases} \\ &= \begin{cases} (g_0, p_0) & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) & \text{if } 1 \leq i \leq n-1 \end{cases} \end{aligned} \quad (2.31)$$

where, G_i and P_i are the group-generate and group-propagate signals and the prefix operator ' $\bullet$ ' is defined as

$$(g_i, p_i) \bullet (g_j, p_j) = (g_i + p_i \cdot g_j, p_i \cdot p_j) \quad (2.32)$$

Post-processing stage:

$$s_i = \begin{cases} h_0 & \text{if } i = 0 \\ h_i \oplus c_{i-1} & \text{if } 1 \leq i \leq n-1 \end{cases} \quad (2.33)$$

Many parallel prefix networks representing different tradeoffs between the number of prefix levels, fanout and wiring tracks have been described, such as Sklansky, Kogge-Stone, Brent-Kung, Han-Carlson, Knowles and Ladner-Fischer [Harr03], [Skl60], [Kogg73], [Bren82], [Han87], [Know01], [Ladn80]. Figs. 2.1 (a) and (b) show the parallel-prefix two's complement adder with c_{-1} assumed to be zero for $n = 8$ using Sklansky and Kogge-Stone structures, respectively. The symbols, '□' and '◇', represent the pre-processing and post-processing operators, respectively. The symbol '●' represents the prefix operator and '○' denotes the buffer. The circuit implementations of the operators are illustrated in Fig. 2.1(c).

Fig. 2.1 Two's complement adder with $c_{-1} = 0$: (a) Sklansky structure (b) Kogge-Stone structure (c)

Implementation of pre-processing, prefix and post-processing operators

Let (G'_i, P'_i) be the group-generate and group-propagate signals with a carry-in $c_{-1} \in \{0,1\}$. Then,

$$(G'_i, P'_i) = \begin{cases} (g_0, p_0) \bullet c_{-1} & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet c_{-1} & \text{if } 1 \leq i \leq n-1 \end{cases} \quad (2.34)$$

As the prefix operator is associative, (2.34) can be simplified using (2.31) as follows.

$$(G'_i, P'_i) = (G_i, P_i) \bullet c_{-1} \text{ if } 0 \leq i \leq n-1 \quad (2.35)$$

Equation (2.35) implies that a two's complement adder with c_{-1} can be implemented by including an additional row of prefix blocks to the parallel prefix structure of an adder without c_{-1} . This is illustrated in Fig. 2.2 for $n = 8$. A modulo $2^n - 1$ adder is illustrated for $n = 8$ in Fig. 2.3. In Fig. 2.3, the EAC addition is realized by employing the carry-out, c_{n-1} as the carry-in c_{-1} . The adder employs $\lceil \log_2 n \rceil + 1$, *i.e.*, 4 prefix levels to compute the carries.

Fig. 2.2 Two's complement adder with c_{-1}

Fig. 2.3 Modulo 2^n-1 adder

2.4.2.2 Parallel prefix modulo 2^n-1 adder with unrolled c_{out}

Let c'_i be the carry from the bit position i and let (G'_i, P'_i) be the group-generate and group-propagate signals of the modulo 2^n-1 addition. For modulo 2^n-1 addition, c_{-1} in (2.34) is replaced by c_{n-1} or equivalently, G_{n-1} . Then, (G'_i, P'_i) becomes

$$\begin{aligned}
 (G'_i, P'_i) &= \begin{cases} (g_0, p_0) \bullet c_{n-1} & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \cdots (g_0, p_0) \bullet c_{n-1} & \text{if } 1 \leq i \leq n-1 \end{cases} \\
 &= \begin{cases} (g_0, p_0) \bullet (G_{n-1}, P_{n-1}) & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \cdots (g_0, p_0) \bullet (G_{n-1}, P_{n-1}) & \text{if } 1 \leq i \leq n-1 \end{cases} \\
 &= \begin{cases} (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \cdots (g_0, p_0) & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \cdots (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \cdots (g_0, p_0) & \text{if } 1 \leq i \leq n-1 \end{cases} \quad (2.36)
 \end{aligned}$$

Property 2.5: For the prefix operator, it can be shown that

$$(G_i, P_i) \bullet (g_j, p_j) \cdots (g_k, p_k) \bullet (G_i, P_i) = (G_i, P_i) \bullet (g_j, p_j) \cdots (g_k, p_k) \quad (2.37)$$

By *Property 2.5*, the redundant terms in (2.36) can be eliminated. The simplified carry equation for modulo 2^n-1 addition becomes

$$(G'_i, P'_i) = (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \bullet \dots \bullet (g_{i+1}, p_{i+1}) \quad (2.38)$$

Equation (2.38) implies that in a modulo 2^n-1 addition, the group-generate $G'_i (= c'_i)$ and the group-propagate P'_i signals are functions of not only the generate, g_i and propagate, p_i signals at bit positions 0 through i , but also of the generate and propagate signals at bit positions $i+1$ through $n-1$ [Kala00]. The modulo 2^n-1 adder, where the generate and the propagate signals are recirculated, is illustrated for $n = 8$ in Fig. 2.4. The adder employs $\lceil \log_2 n \rceil = 3$ prefix levels.

Fig. 2.4 Modulo 2^n-1 adder with unrolled c_{out}

2.4.2.3 Parallel prefix modulo 2^n-1 adder with Ling carry

Ling adder is a variation of CLA adder. The equation for the traditional carry c_i is simplified by factoring the common propagate term p_i to create the Ling carry H_i . H_i can be computed faster than the corresponding c_i due to its simpler Boolean equation. But the derivation of the final sum requires a multiplexer that selects either the half-sum bit h_i or $h_i \oplus p_{i-1}$ according to H_{i-1} [Ling81], [Dimi05a]. In [Dimi05b], the parallel prefix modulo 2^n-1 adder using Ling carry was proposed. The prefix adder of [Dimi05b] is described for the example $n = 8$ below.

From (2.38), the carry c_0' of modulo 2^8-1 addition is given by

$$\begin{aligned} c_0' = & g_0 + p_0g_7 + p_0p_7g_6 + p_0p_7p_6g_5 + p_0p_7p_6p_5g_4 + p_0p_7p_6p_5p_4g_3 \\ & + p_0p_7p_6p_5p_4p_3g_2 + p_0p_7p_6p_5p_4p_3p_2g_1 \end{aligned} \quad (2.39)$$

Since $g_i \cdot p_i = g_i$,

$$\begin{aligned} c_0' = & p_0(g_0 + g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4 + p_7p_6p_5p_4g_3) \\ & + p_0(p_7p_6p_5p_4p_3g_2 + p_7p_6p_5p_4p_3p_2g_1) \\ = & p_0H_0 \end{aligned} \quad (2.40)$$

where H_i is the Ling carry for modulo 2^n-1 addition. H_0 is further simplified as

$$\begin{aligned} H_0 = & (g_0 + g_7) + (p_7p_6) \cdot (g_6 + g_5) + (p_7p_6) \cdot (p_5p_4) \cdot (g_4 + g_3) \\ & + (p_7p_6) \cdot (p_5p_4) \cdot (p_3p_2) \cdot (g_2 + g_1) \end{aligned} \quad (2.41)$$

Let

$$\begin{aligned} G_i^* = & \begin{cases} g_0 + g_7 & \text{if } i = 0 \\ g_i + g_{i-1} & \text{if } 1 \leq i \leq 7 \end{cases} \\ P_i^* = & \begin{cases} p_0 \cdot p_7 & \text{if } i = 0 \\ p_i \cdot p_{i-1} & \text{if } 1 \leq i \leq 7 \end{cases} \end{aligned} \quad (2.42)$$

Equation (2.41) can be rewritten as

$$\begin{aligned} H_0 = & G_0^* + P_7^* \cdot G_6^* + P_7^* \cdot P_5^* \cdot G_4^* + P_7^* \cdot P_5^* \cdot P_3^* \cdot G_2^* \\ = & (G_0^*, P_7^*) \cdot (G_6^*, P_5^*) \cdot (G_4^*, P_3^*) \cdot (G_2^*, P_1^*) \end{aligned} \quad (2.43)$$

Similar equations for H_1 to H_7 are given by

$$\begin{aligned}
H_1 &= (G_1^*, P_0^*) \cdot (G_7^*, P_6^*) \cdot (G_5^*, P_4^*) \cdot (G_3^*, P_2^*) \\
H_2 &= (G_2^*, P_1^*) \cdot (G_0^*, P_7^*) \cdot (G_6^*, P_5^*) \cdot (G_4^*, P_3^*) \\
H_3 &= (G_3^*, P_2^*) \cdot (G_1^*, P_0^*) \cdot (G_7^*, P_6^*) \cdot (G_5^*, P_4^*) \\
H_4 &= (G_4^*, P_3^*) \cdot (G_2^*, P_1^*) \cdot (G_0^*, P_7^*) \cdot (G_6^*, P_5^*) \\
H_5 &= (G_5^*, P_4^*) \cdot (G_3^*, P_2^*) \cdot (G_1^*, P_0^*) \cdot (G_7^*, P_6^*) \\
H_6 &= (G_6^*, P_5^*) \cdot (G_4^*, P_3^*) \cdot (G_2^*, P_1^*) \cdot (G_0^*, P_7^*) \\
H_7 &= (G_7^*, P_6^*) \cdot (G_5^*, P_4^*) \cdot (G_3^*, P_2^*) \cdot (G_1^*, P_0^*)
\end{aligned} \tag{2.44}$$

Fig. 2.5(a) shows the parallel prefix implementation of (2.44) with $\lceil \log_2 n \rceil - 1 = 2$ prefix levels. The computations in the pre-processing and the post-processing stages shown in Fig. 2.5(b) differ from that of Fig. 2.1(c). In the pre-processing stage, G_i^* is computed using two AND gates and an OR gate while P_i^* is computed using two OR gates and an AND gate. h_i is also computed using an XOR gate. In the post-processing stage, the sum s_i is generated in a multiplexer, where H_{i-1} (H_{n-1} for $i = 0$) selects either h_i or $h_i \oplus p_{i-1}$.

Fig. 2.5 (a) Modulo 2^n-1 adder with Ling carry (b) Implementation of pre-processing and post-processing stages

2.4.2.4 Single representation of zero in modulo 2^n-1 adder

Modulo 2^n-1 addition, when implemented as an EAC addition, leads to dual representation of zero. If a single representation is desired, minor modification to the adders in Figs. 2.3 – 2.5 is necessary. In a modulo 2^n-1 adder, the result $\underbrace{1\dots1}_n$ occurs only if the addends are bitwise complement of one another. The term T is defined as the logical conjunction of h_i , $i = 0, 1, \dots, n-1$. As h_i is the XOR of the addend bits, T denotes the condition that the addends are bitwise complement. A single representation of zero can then be achieved by computing the sum using the modified equation,

$$s_i = (h_i \oplus c_{i-1}) \cdot \bar{T} \quad (2.45)$$

2.4.2.5 Multi-operand modulo 2^n-1 adder (MOMA, 2^n-1)

Multi-operand modulo addition is crucial to forward conversion, modulo multiplication and modulo squaring. As the name suggests, in a (MOMA, 2^n-1) more than two, *i.e.*, $k > 2$, n -bit operands are summed. The functionality of a (MOMA, 2^n-1) is expressed as

$$|S|_{2^n-1} = \left| \sum_{i=0}^{k-1} X_i \right|_{2^n-1} \quad (2.46)$$

In the straightforward implementation of (MOMA, 2^n-1), the operands can be added sequentially using a single two-operand modulo 2^n-1 adder and a register to hold the partial sum. The total number of cycles required to compute the sum is $k-1$. Alternatively, a tree of $k-1$ two-operand modulo 2^n-1 adders can be used to perform the summation in $\lceil \log_2 k \rceil$ cycles. However these implementations are constrained by the delay of the two-operand modulo 2^n-1 adder.

Fast (MOMA, 2^n-1) using Carry Save Adders (CSAs) has been proposed in [Zimm99]. An n -bit CSA adds three n -bit operands, X , Y and Z , without carry propagation and results in a redundant sum represented by an n -bit sum vector, $S = s_{n-1} \dots s_1 s_0$ and an n -bit carry vector, $C = c_{n-1} \dots c_1 c_0$, *i.e.*,

$$\begin{aligned}
X + Y + Z &= \underbrace{c_{n-1} \dots c_1 c_0}_{n+1} 0 + \underbrace{s_{n-1} \dots s_1 s_0}_n \\
&= 2C + S
\end{aligned} \tag{2.47}$$

The n -bit CSA consists of n Full Adders (FAs) such that the FAs operate in parallel without carry propagation between them. Fig. 2.6 illustrates an 8-bit CSA.

Fig. 2.6 Example of an 8-bit CSA

Since modulo $2^n - 1$ addition is equivalent to EAC addition, (2.47) is modified for EAC addition as follow.

$$\left| X + Y + Z \right|_{2^n - 1} = \left| \underbrace{c_{n-2} \dots c_0 c_{n-1}}_n + \underbrace{s_{n-1} \dots s_1 s_0}_n \right|_{2^n - 1} \tag{2.48}$$

Fig. 2.7 illustrates an 8-bit EAC-CSA.

Fig. 2.7 Example of an 8-bit EAC-CSA

A (MOMA, $2^n - 1$) can be built to add k operands, by arranging $k - 2$ n -bit EAC-CSAs in an array or tree structure for addition in linear or logarithmic time, respectively, followed by a two-operand modulo $2^n - 1$ adder to sum the final S and C vectors. The resultant circuit is very regular since the carry-outs are fed back into the adder structure as carry-ins. Fig. 2.8 shows the CSA tree implementation of (MOMA, $2^n - 1$) for $n = 8$ and $k = 5$. The five addends are

represented as X_0, X_1, X_2, X_3 and X_4 . The final two-operand modulo 2^n-1 adder can be implemented as either Fig. 2.3, Fig. 2.4 or Fig. 2.5.

Fig. 2.8 CSA tree implementation of (MOMA, 2^n-1)

The depth $D(k)$, *i.e.*, the number of FAs in the critical path of a k -operand CSA tree, is given by the function

$$D(k) = 1 + D(\lceil 2k/3 \rceil) \quad (2.49)$$

$D(k)$ for k in the range $[3, 94]$ is shown in Table 2.1.

Table 2.1 Depth of k -operand CSA tree

k	3	4	5 – 6	7 – 9	10 – 13	14 – 19	20 – 28	29 – 42	43 – 63	64 – 94
$D(k)$	1	2	3	4	5	6	7	8	9	10

2.4.3 Modulo 2^n+1 adder

The residues of the special modulus 2^n+1 in the range $[0, 2^n]$ necessitate $n+1$ bits for their representation but only 2^n+1 out of the 2^{n+1} possible representations are utilized. Furthermore, the residues of the special moduli 2^n-1 and 2^n require only n bits for their representations. To

limit the number of bits in the representation of residues modulo 2^n+1 to n bits, diminished-1 representation was introduced [Leib76]. In this system, the number X is represented by $X' = X-1$. Therefore, the numbers in the range $[1, 2^n]$ are denoted as $[0, 2^n-1]$. The zero operand is not used directly in the computation as its result or any result that is a zero can be easily derived and indicated by a flag bit. Let X and Y be the addends and S be their sum. Modulo 2^n+1 addition in diminished-1 representation is given by

$$\begin{aligned} |S|_{2^n+1} &= |X + Y|_{2^n+1} \\ |S' + 1|_{2^n+1} &= |X' + 1 + Y' + 1|_{2^n+1} \\ |S'|_{2^n+1} &= |X' + Y' + 1|_{2^n+1} \end{aligned} \quad (2.50)$$

Equation (2.50) implies that in a diminished-1 adder, the result, S' is the sum of the addends, X' and Y' , and a constant one. Equation (2.50) can be rewritten as

$$\begin{aligned} |S'|_{2^n+1} &= \begin{cases} X' + Y' + 1 & \text{if } X' + Y' + 1 \leq 2^n \\ X' + Y' + 1 - 2^n - 1 & \text{if } X' + Y' + 1 > 2^n \end{cases} \\ &= \begin{cases} X' + Y' + 1 & \text{if } X' + Y' + 1 \leq 2^n \\ X' + Y' - 2^n & \text{if } X' + Y' + 1 > 2^n \end{cases} \end{aligned} \quad (2.51)$$

As S' is represented using only n bits, (2.51) is reformulated as

$$\begin{aligned} |S'| &= \begin{cases} X' + Y' + 1 & \text{if } X' + Y' + 1 < 2^n \\ X' + Y' - 2^n & \text{if } X' + Y' + 1 \geq 2^n \end{cases} \\ &= \begin{cases} |X' + Y' + 1|_{2^n} & \text{if } X' + Y' + 1 < 2^n \\ |X' + Y'|_{2^n} & \text{if } X' + Y' + 1 \geq 2^n \end{cases} \\ &= |X' + Y' + \bar{c}_{out}|_{2^n} \end{aligned} \quad (2.52)$$

where c_{out} is the carry-out from the n -bit addition of X' and Y' [Zimm99]. Hence, a diminished-1 modulo 2^n+1 addition is equivalent to an n -bit complementary-end-around-carry (CEAC) addition.

As an example, consider the modulus $2^4+1 = 17$ and the addends, $X = 8_{10} = 1000_2$ and $Y = 12_{10} = 1100_2$. The corresponding diminished-1 addends are $X' = 0111_2$ and $Y' = 1011_2$. The 4-bit addition of X' and Y' results in $sum = 0010_2$ and $c_{out} = 1_2$. Then, $|S'|_{17}$ is given by the addition of sum and $\bar{c}_{out}$, i.e., $0010_2 = 2_{10}$.

In [Verg02], modulo 2^n+1 addition based on one-level and two-level CLA adders were suggested. Modulo 2^n+1 adders based on parallel-prefix structures were proposed for diminished-1 representation in [Zimm99], [Verg02], [Verg09] and for weighted binary representation in [Efst04b]. A unifying approach for both diminished-1 and weighted binary additions was described in [Verg08].

2.4.3.1 Parallel prefix modulo 2^n+1 adder

A diminished-1 modulo 2^n+1 adder is illustrated in Fig. 2.9 for $n = 8$ [Zimm99]. In Fig. 2.9, the CEAC addition is implemented by considering the bit-complement of the carry-out $\bar{c}_{n-1}$ as the carry-in c_{-1} . The number of prefix levels used is $\lceil \log_2 n \rceil + 1 = 4$.

Fig. 2.9 Diminished-1 modulo 2^n+1 adder

2.4.3.2 Parallel prefix modulo 2^n+1 adder with unrolled c_{out}

Let (G_i, P_i) and (G'_i, P'_i) be the group-generate and group-propagate signal pairs of the binary and modulo 2^n+1 additions, respectively. By replacing c_{-1} in (2.34) with $\bar{c}_{n-1} = \bar{G}_{n-1}$, (G'_i, P'_i) becomes

$$\begin{aligned}
(G'_i, P'_i) &= \begin{cases} (g_0, p_0) \bullet \bar{c}_{n-1} & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet \bar{c}_{n-1} & \text{if } 1 \leq i \leq n-1 \end{cases} \\
&= \begin{cases} (g_0, p_0) \bullet (\bar{G}_{n-1}, P_{n-1}) & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet (\bar{G}_{n-1}, P_{n-1}) & \text{if } 1 \leq i \leq n-1 \end{cases}
\end{aligned} \tag{2.53}$$

By defining the complement of a group-generate and group-propagate signal pair (G, P) as $\overline{(G, P)} = (\bar{G}, \bar{P})$, (2.53) is modified to

$$\begin{aligned}
(G'_i, P'_i) &= \begin{cases} (g_0, p_0) \bullet \overline{(g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \bullet \dots \bullet (g_0, p_0)} & \text{if } i = 0 \\ (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet \overline{(g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \bullet \dots \bullet (g_0, p_0)} & \text{if } 1 \leq i \leq n-1 \end{cases}
\end{aligned} \tag{2.54}$$

Property 2.6: For the prefix operator, it can be shown that

$$(G_i, P_i) \bullet \overline{(g_j, p_j) \bullet \dots \bullet (g_k, p_k) \bullet (G_i, P_i)} = (G_i, P_i) \bullet \overline{(g_j, p_j) \bullet \dots \bullet (g_k, p_k)} \tag{2.55}$$

By eliminating the redundant terms using *Property 2.6*, (2.54) is simplified to

$$(G'_i, P'_i) = (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \dots \bullet (g_0, p_0) \bullet \overline{(g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \bullet \dots \bullet (g_{i+1}, p_{i+1})} \tag{2.56}$$

Equation (2.56) implies that in a modulo 2^n+1 adder, the carry at position i depends not only on the bits in positions i to 0 but also on the bits in positions $n-1$ to $i+1$. However, (2.56) cannot always be implemented using $\lceil \log_2 n \rceil$ prefix levels. To this end, the carry equations are reformulated using the following property of the prefix operator.

Property 2.7:

$$(g, p) \bullet \overline{(G, P)} = \overline{(\bar{p}, \bar{g})} \bullet (G, P) \tag{2.57}$$

For example, when $n = 8$, the carries, G'_0 to G'_7 of modulo 2^8+1 additions are given by

$$\begin{aligned}
(G'_0, P'_0) &= (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1)} \\
(G'_1, P'_1) &= (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2)} \\
(G'_2, P'_2) &= (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3)} \\
(G'_3, P'_3) &= (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4)} \\
(G'_4, P'_4) &= (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5)} \\
(G'_5, P'_5) &= (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6)} \\
(G'_6, P'_6) &= (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7)} \\
(G'_7, P'_7) &= (g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0)
\end{aligned} \tag{2.58}$$

The equation for (G'_i, P'_i) is reformulated by using *Property 2.7* k times recursively, where

$$k = \begin{cases} 1+i & \text{if } i < n/2-1 \\ 1+i-n/2 & \text{if } n/2-1 < i < n-1 \\ 0 & \text{if } i = n/2-1 \text{ or } n-1 \end{cases} \tag{2.59}$$

Then, (2.58) becomes

$$\begin{aligned}
(G'_0, P'_0) &= (\overline{p_0}, \overline{g_0}) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1)} \\
(G'_1, P'_1) &= (\overline{p_1}, \overline{g_1}) \bullet (\overline{p_0}, \overline{g_0}) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2)} \\
(G'_2, P'_2) &= (\overline{p_2}, \overline{g_2}) \bullet (\overline{p_1}, \overline{g_1}) \bullet (\overline{p_0}, \overline{g_0}) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3)} \\
(G'_3, P'_3) &= (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4)} \\
(G'_4, P'_4) &= (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (\overline{p_0}, \overline{g_0}) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5)} \\
(G'_5, P'_5) &= (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (\overline{p_1}, \overline{g_1}) \bullet (\overline{p_0}, \overline{g_0}) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6)} \\
(G'_6, P'_6) &= (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0) \bullet \overline{(g_7, p_7)} \\
(G'_7, P'_7) &= (g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \bullet (g_0, p_0)
\end{aligned} \tag{2.60}$$

Fig. 2.10 shows the parallel-prefix implementation of (2.60) in three prefix levels [Verg02].

Fig. 2.10 Diminished-1 modulo 2^n+1 adder with unrolled c_{out}

2.4.3.3 Parallel prefix modulo 2^n+1 adder with Ling carry

The parallel prefix modulo 2^n+1 adder employing Ling carries was presented in [Verg09].

The design is explained for $n = 8$ as follows. From (2.60), the carry c'_0 is given by

$$\begin{aligned}
 c'_0 &= g_0 + p_0 \bar{p}_7 + p_0 \bar{g}_7 \bar{p}_6 + p_0 \bar{g}_7 \bar{g}_6 \bar{p}_5 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{p}_4 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{p}_3 \\
 &\quad + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{p}_2 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{g}_2 \bar{p}_1 \\
 &= p_0 (g_0 + \bar{p}_7 + \bar{g}_7 \bar{p}_6 + \bar{g}_7 \bar{g}_6 \bar{p}_5 + \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{p}_4 + \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{p}_3) \\
 &\quad + p_0 (\bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{p}_2 + \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{g}_2 \bar{p}_1) \\
 &= p_0 H_0
 \end{aligned} \tag{2.61}$$

where the Ling carry, H_0 is given by

$$\begin{aligned}
 H_0 &= (g_0 + \bar{p}_7) + (\bar{g}_7 \bar{g}_6) \cdot (\bar{p}_6 + \bar{p}_5) + (\bar{g}_7 \bar{g}_6) \cdot (\bar{g}_5 \bar{g}_4) \cdot (\bar{p}_4 + \bar{p}_3) \\
 &\quad + (\bar{g}_7 \bar{g}_6) \cdot (\bar{g}_5 \bar{g}_4) \cdot (\bar{g}_3 \bar{g}_2) \cdot (\bar{p}_2 + \bar{p}_1)
 \end{aligned} \tag{2.62}$$

Let (G_i^*, P_i^*) be defined as

$$\begin{aligned}
G_i^* &= \begin{cases} g_0 + \bar{p}_7 & \text{if } i = 0 \\ g_i + g_{i-1} & \text{if } 1 \leq i \leq 7 \end{cases} \\
P_i^* &= \begin{cases} p_0 \cdot \bar{g}_7 & \text{if } i = 0 \\ p_i \cdot p_{i-1} & \text{if } 1 \leq i \leq 7 \end{cases}
\end{aligned} \tag{2.63}$$

Equation (2.62) can be rewritten as

$$\begin{aligned}
H_0 &= G_0^* + \bar{G}_7^* \cdot \bar{P}_6^* + \bar{G}_7^* \cdot \bar{G}_5^* \cdot \bar{P}_4^* + \bar{G}_7^* \cdot \bar{G}_5^* \cdot \bar{G}_3^* \cdot \bar{P}_2^* \\
&= (G_0^*, \bar{G}_7^*) \cdot (\bar{P}_6^*, \bar{G}_5^*) \cdot (\bar{P}_4^*, \bar{G}_3^*) \cdot (\bar{P}_2^*, \bar{G}_1^*)
\end{aligned} \tag{2.64}$$

Similarly, H_1 to H_7 are given by

$$\begin{aligned}
H_1 &= (G_1^*, P_0^*) \cdot (\bar{P}_7^*, \bar{G}_6^*) \cdot (\bar{P}_5^*, \bar{G}_4^*) \cdot (\bar{P}_3^*, \bar{G}_2^*) \\
H_2 &= (G_2^*, P_1^*) \cdot (G_0^*, \bar{G}_7^*) \cdot (\bar{P}_6^*, \bar{G}_5^*) \cdot (\bar{P}_4^*, \bar{G}_3^*) \\
H_3 &= (G_3^*, P_2^*) \cdot (G_1^*, P_0^*) \cdot (\bar{P}_7^*, \bar{G}_6^*) \cdot (\bar{P}_5^*, \bar{G}_4^*) \\
H_4 &= (G_4^*, P_3^*) \cdot (G_2^*, P_1^*) \cdot (G_0^*, \bar{G}_7^*) \cdot (\bar{P}_6^*, \bar{G}_5^*) \\
H_5 &= (G_5^*, P_4^*) \cdot (G_3^*, P_2^*) \cdot (G_1^*, P_0^*) \cdot (\bar{P}_7^*, \bar{G}_6^*) \\
H_6 &= (G_6^*, P_5^*) \cdot (G_4^*, P_3^*) \cdot (G_2^*, P_1^*) \cdot (G_0^*, \bar{G}_7^*) \\
H_7 &= (G_7^*, P_6^*) \cdot (G_5^*, P_4^*) \cdot (G_3^*, P_2^*) \cdot (G_1^*, P_0^*)
\end{aligned} \tag{2.65}$$

Fig. 2.11 shows the parallel-prefix implementation of (2.65) in $\lceil \log_2 n \rceil - 1 = 2$ levels [Verg09].

Fig. 2.11 Diminished-1 modulo 2^n+1 adder with Ling carry

2.4.3.4 Handling zero in modulo 2^n+1 adder

Any number, X that is a zero in weighted-binary representation is denoted in diminished-1 representation by setting the zero flag bit to logic one, *i.e.*, $x'_n = 1$ while $X' = \underbrace{0 \cdots 0}_n$. The following four distinct cases where the addend or the sum is a zero in weighted-binary or diminished-1 representations are considered.

Case 1: Either of the addends is a zero in weighted-binary representation, *i.e.*, $x'_n = 1$, $X' = \underbrace{0 \cdots 0}_n$ or $y'_n = 1$, $Y' = \underbrace{0 \cdots 0}_n$.

If one of the addends is a zero, then the sum equals the other addend. In other words, $x'_n = 1$ implies that $s'_n = y'_n$ and $S' = Y'$. Similarly, $y'_n = 1$ implies that $s'_n = x'_n$ and $S' = X'$.

Case 2: Both the addends are zeros in weighted-binary representation, *i.e.*, $x'_n = y'_n = 1$, $X' = Y' = \underbrace{0 \cdots 0}_n$.

This is a trivial case. $x'_n = y'_n = 1$ implies that $s'_n = 1$ and $S' = \underbrace{0 \cdots 0}_n$.

Case 3: The sum is a zero in weighted-binary representation *i.e.*, $s'_n = 1$, $S' = \underbrace{0 \cdots 0}_n$.

This case differs from *Case 2* because the sum modulo 2^n+1 can be a zero even when both addends are non-zero. This scenario occurs when the addends in diminished-1 representation are bitwise complement of one another. The half-sum signal h_i computed in the pre-processing stage denotes if the bits, x_i and y_i are complement to each other. The term T is defined as the logical conjunction of h_i , $i = 0, 1, \dots, n-1$. It is used to identify the case when the addends are bitwise complement of each other.

For example, consider the modulus $2^4+1 = 17_{10}$ and the addends, $X = 11_{10} = 1011_2$ and $Y = 6_{10} = 0110_2$. The bitwise complement addends in diminished-1 representation are given by $x'_4 = 0_2$, $X' = 1010_2$ and $y'_4 = 0_2$, $Y' = 0101_2$. Then, the 4-bit CEAC addition of X' and Y' results in $S' = 0000_2$. Furthermore, the zero flag bit, s'_4 is set to 1₂.

Case 4: The sum is a zero in diminished-1 representation, *i.e.*, $s'_n = 0$, $S' = \underbrace{0 \cdots 0}_n$.

In both *Cases 3* and *4*, the diminished-1 representation of the sum S' is $\underbrace{0 \cdots 0}_n$. However, in *Case 3*, the zero flag bit is 1 (denoting a zero in weighted-binary representation) while in *Case 4*, the zero flag bit is 0 (denoting a zero in diminished-1 representation).

For example, let the addends X be $15_{10} = 1111_2$ and Y be $3_{10} = 0011_2$. Then the diminished-1 representation of the addends is $x'_4 = 0_2$, $X' = 1110_2$, $y'_4 = 0_2$ and $Y' = 0010_2$. The 4-bit CEAC addition of X' and Y' leads to $S' = 0000_2$. Since X' and Y' are not bitwise complement of each other, the zero flag bit s'_4 is not set to 1, *i.e.*, $s'_4 = 0$.

2.4.3.5 Multi-operand modulo 2^n+1 adder (MOMA, 2^n+1)

By extending the definition of two-operand diminished-1 modulo 2^n+1 addition in (2.50) to multi-operand, the functionality of diminished-1 (MOMA, 2^n+1) can be expressed mathematically as

$$|S'|_{2^n+1} = \left| \sum_{i=0}^{k-1} X'_i + (k-1) \right|_{2^n+1} \quad (2.66)$$

In other words, in a diminished-1 (MOMA, 2^n+1), the result, S' is the sum of k addends, X'_i and the constant $k-1$. Fast CSA array and tree based implementation of (MOMA, 2^n+1) have been proposed in [Zimm99]. To implement a diminished-1 modulo 2^n+1 addition, *i.e.*, a CEAC addition of X' , Y' and Z' in a CSA, (2.47) is modified to

$$|X' + Y' + Z' + 1|_{2^n+1} = \left| \underbrace{c_{n-2} \cdots c_0 \bar{c}_{n-1}}_n + \underbrace{s_{n-1} \cdots s_1 s_0}_n \right|_{2^n+1} \quad (2.67)$$

Fig. 2.12 illustrates an 8-bit CEAC-CSA.

Fig. 2.12 Example of an 8-bit CEAC-CSA

A diminished-1 (MOMA, 2^n+1) can then be designed to add k operands by arranging $k-2$ n -bit CEAC-CSAs in an array or a tree structure followed by a two-operand modulo 2^n+1 adder. From (2.66), the constant to be incorporated in the diminished-1 (MOMA, 2^n+1) is $k-1$. From (2.67), it can be observed that a CEAC-CSA sums not only the addends but also a correction constant of one. Hence, the $k-2$ CEAC-CSAs inherently add a constant of $k-2$. Eventually, the final two-operand adder adds a constant one, thus bringing the total constant to the required $k-1$. Fig. 2.13 depicts the diminished-1 (MOMA, 2^n+1) for $n = 8$ and $k = 5$. The final two-operand adder can be implemented as a fast parallel-prefix adder of Fig. 2.9, Fig. 2.10 or Fig. 2.11.

Fig. 2.13 CSA tree implementation of (MOMA, 2^n+1)

2.4.4 Modulo m multiplier

Modulo m multiplication of two operands, X and Y , can be expressed mathematically as

$$Z = \begin{cases} X \cdot Y & \text{if } X \cdot Y < m \\ |X \cdot Y|_m & \text{if } X \cdot Y \geq m \end{cases} \quad (2.68)$$

A number of techniques exist for modulo multiplication.

- a) The index calculus technique replaces modulo m multiplication by modulo $m-1$ addition. If m is prime, there exists a primitive radix r such that its powers modulo m cover the set $[1, m-1]$. By using the isomorphism, the product of two residues is transformed into the sum of their indices where the index as well as the inverse index transforms can be stored in LUTs [Szab67], [Jull80], [Radh92].

$$|X \cdot Y|_m = \left| \left| r^x \right|_m \cdot \left| r^y \right|_m \right|_m = \left| r^{x+y} \right|_{m-1} \quad (2.69)$$

In [Dugd94], a multiplication technique for a non-prime modulus is described. If the modulus can be decomposed into two or more co-prime factors, then multiplication can be performed as a set of concurrent multiplication operations using the co-prime factors as moduli. If the factor is a prime, the multiplication is performed using the index calculus method, else the multiplication is performed using LUTs.

- b) The quarter-square technique is equally applicable to prime and non-prime moduli. Using this technique, the product is determined as

$$|X \cdot Y|_m = \frac{\left| (X+Y)^2 \right|_m - \left| (X-Y)^2 \right|_m}{4} \quad (2.70)$$

LUT based implementations of the quarter-square technique have been proposed in [Sode80] [Tayl81].

- c) The Montgomery modular multiplication algorithm computes the product Z as $|X \cdot Y \cdot r^{-n}|_m$, where r is the radix and n is the number of digits in the representation of X , Y and m [Mont85]. Let r and m be relatively prime, then the algorithm is given by

```

 $Z = 0$ 
for  $i = 0$  to  $n-1$  do
begin
(1)  $Q_i = ((Z_0 + X_i Y_0) (r - m_0)^{-1}) \bmod r$ 
(2)  $Z = (Z + X_i Y + Q_i m) \text{ div } r$ 
end

```

where X_i is the i -th digit of X .

- d) Modulo multiplications based on binary multiplication have been suggested in [Alia91] and [Hias00]. In [Alia91], (2.68) was reformulated as (2.71) and implemented using three binary multipliers.

$$Z = XY - m \left\lfloor \frac{XY}{m} \right\rfloor \quad (2.71)$$

In [Hias00], modulo m multiplication was modified to

$$|Z|_m = |A + aC + |D2^{2n-1-k} + B2^{n-1}|_m|_m \quad (2.72)$$

where

$$\begin{aligned}
a &= |2^n|_m \\
k &= 1 + \lfloor \log_2 a \rfloor \\
A &= \sum_{i=0}^{n-2} z_i 2^i \\
B &= z_{n-1} \\
C &= \sum_{i=n}^{2n-2-k} z_i 2^{i-n} \\
D &= \sum_{i=2n-1-k}^{2n-1} z_i 2^{i-(2n-1-k)}
\end{aligned} \quad (2.73)$$

The implementation of (2.72) consisted of basically an $n \times n$ binary multiplier and a $(n-1-k) \times k$ binary multiplier.

2.4.5 Modulo 2^n-1 multiplier

Full combinatorial implementations of modulo 2^n-1 multiplier using the properties of modulo arithmetic have been proposed in [Wang96a], [Zimm99], [Efst04a]. In [Wang96a] and [Zimm99], no encoding scheme was used on the multiplier bits and the number of partial products was n . In [Zimm99] and [Efst04a], radix-4 Booth encoding was employed and the number of partial products was reduced to $\lfloor n/2 \rfloor + 1$ and $\lceil n/2 \rceil$, respectively, where $\lfloor a \rfloor$ represents the greatest integer smaller than or equal to a .

2.4.5.1 Non-encoded modulo 2^n-1 multiplier

Let X , Y and Z be the n -bit multiplicand, multiplier and product, respectively. Modulo 2^n-1 multiplication can be formulated as

$$\begin{aligned} |Z|_{2^n-1} &= |X \cdot Y|_{2^n-1} \\ &= \left| \sum_{i=0}^{n-1} X \cdot y_i \cdot 2^i \right|_{2^n-1} \\ &= \left| \sum_{i=0}^{n-1} PP_i \right|_{2^n-1} \end{aligned} \quad (2.74)$$

where the n -bit vector PP_i is tabulated in Table 2.2. The implementation of (2.74) involves two stages, namely: (a) Modulo-reduced Generation of Partial Products (MPPG) and (b) Modulo-reduced Accumulation of Partial Products (MPPA).

Table 2.2 Modulo 2^n-1 reduced partial products

y_i	PP_i
0	0...0
1	$x_{n-1-i} \dots x_0 \ x_{n-1} \dots x_{n-i}$

In the MPPG stage, the n PP_i s are generated in accordance to Table 2.2. As the PP_i is either X when y_i is 1 or 0 when y_i is 0, the MPPG is implemented by an array of $n \times n$ AND gates. For $n = 8$, Fig. 2.14 illustrates the 8 PP_i s that can be computed using 64 AND gates.

$PP_0 =$	x_7y_0	x_6y_0	x_5y_0	x_4y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0
$PP_1 =$	x_6y_1	x_5y_1	x_4y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1	x_7y_1
$PP_2 =$	x_5y_2	x_4y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2	x_7y_2	x_6y_2
$PP_3 =$	x_4y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3	x_7y_3	x_6y_3	x_5y_3
$PP_4 =$	x_3y_4	x_2y_4	x_1y_4	x_0y_4	x_7y_4	x_6y_4	x_5y_4	x_4y_4
$PP_5 =$	x_2y_5	x_1y_5	x_0y_5	x_7y_5	x_6y_5	x_5y_5	x_4y_5	x_3y_5
$PP_6 =$	x_1y_6	x_0y_6	x_7y_6	x_6y_6	x_5y_6	x_4y_6	x_3y_6	x_2y_6
$PP_7 =$	x_0y_7	x_7y_7	x_6y_7	x_5y_7	x_4y_7	x_3y_7	x_2y_7	x_1y_7

Fig. 2.14 MPPG for modulo 2^n-1 multiplier

Next, in the MPPA stage, the n PP_i s from the preceding MPPG stage are added using a (MOMA, 2^n-1). Fig. 2.15 illustrates the CSA tree based implementation of a (MOMA, 2^n-1) that adds the 8 partial products, PP_0 to PP_7 generated in Fig 2.14. An 8-bit EAC-CSA consisting of 8 FAs as shown in Fig. 2.7 is represented by a single block in Fig. 2.15 for ease of illustration. The total number of EAC-CSAs required is $8-2 = 6$.

Fig. 2.15 MPPA for modulo 2^n-1 multiplier

2.4.5.2 Radix-4 Booth encoded modulo 2^n-1 multiplier

Booth encoding of the multiplier bits has been recommended in [Zimm99] and [Efst04a] for improved area-delay-power performance. In the radix-4 Booth encoding algorithm, successive overlapping triplets of bits of the multiplier Y are examined and encoded as an

element of the set $\{-2, -1, 0, +1, +2\}$ by a Booth Encoder (BE) block. The encoded multiplier digit, along with the multiplicand X , are used for forming a partial product. Every bit of each partial product is generated by a Booth Selector (BS) block.

In [Zimm99], the multiplier bits were encoded to $\lfloor n/2 \rfloor + 1$ digits as shown below.

$$Y = \sum_{i=0}^{n-1} y_i 2^i = \sum_{i=0}^{\lfloor n/2 \rfloor} \underbrace{(y_{2i-1} + y_{2i} - 2y_{2i+1})}_{\{-2, -1, 0, +1, +2\}} 2^{2i} \quad (2.75)$$

where $y_{-1} = y_{n-1}$.

In [Efst04a], (2.75) was further simplified for even n such that the multiplier was represented using only $\lceil n/2 \rceil$ digits. The Booth encoding of the multiplier in [Efst04a] is given by

$$Y = \sum_{i=0}^{n-1} y_i 2^i = \sum_{i=0}^{\lceil n/2 \rceil - 1} \underbrace{(y_{2i-1} + y_{2i} - 2y_{2i+1})}_{\{-2, -1, 0, +1, +2\}} 2^{2i} \quad (2.76)$$

By using the radix-4 Booth encoded multiplier in (2.76), modulo $2^n - 1$ multiplication is expressed as

$$\begin{aligned} |Z|_{2^n-1} &= |X \cdot Y|_{2^n-1} \\ &= \left| \sum_{i=0}^{n-1} X \cdot y_i 2^i \right|_{2^n-1} \\ &= \left| \sum_{i=0}^{\lceil n/2 \rceil - 1} X \cdot \underbrace{(y_{2i-1} + y_{2i} - 2y_{2i+1})}_{\{-2, -1, 0, +1, +2\}} 2^{2i} \right|_{2^n-1} \\ &= \left| \sum_{i=0}^{\lceil n/2 \rceil - 1} PP_i \right|_{2^n-1} \end{aligned} \quad (2.77)$$

where PP_i is tabulated in Table 2.3.

Table 2.3 Modulo 2^n-1 reduced partial products for radix-4 Booth encoding

y_{2i+1}	y_{2i}	y_{2i-1}	$y_{2i-1} + y_{2i} - 2y_{2i+1}$	PP_i
0	0	0	0	$0 \dots 0$
0	0	1	+1	$x_{n-1-2i} x_{n-2-2i} \dots x_0 x_{n-1} \dots x_{n-2i}$
0	1	0	+1	$x_{n-1-2i} x_{n-2-2i} \dots x_0 x_{n-1} \dots x_{n-2i}$
0	1	1	+2	$x_{n-2-2i} x_{n-3-2i} \dots x_0 x_{n-1} \dots x_{n-1-2i}$
1	0	0	-2	$\bar{x}_{n-2-2i} \bar{x}_{n-3-2i} \dots \bar{x}_0 \bar{x}_{n-1} \dots \bar{x}_{n-1-2i}$
1	0	1	-1	$\bar{x}_{n-1-2i} \bar{x}_{n-2-2i} \dots \bar{x}_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$
1	1	0	-1	$\bar{x}_{n-1-2i} \bar{x}_{n-2-2i} \dots \bar{x}_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$
1	1	1	0	$1 \dots 1$

The generation of the $\lceil n/2 \rceil = 4$ PP_i s according to Table 2.3 in the MPPG stage is illustrated in Fig. 2.16(a) for $n = 8$. The MPPG consists of $\lceil n/2 \rceil = 4$ BE as well as $n \lceil n/2 \rceil = 32$ BS blocks. A bit slice of the BE and BS blocks are shown in Figs. 2.16 (b) and (c), respectively. In Fig. 2.17, the accumulation of the $\lceil n/2 \rceil = 4$ PP_i s in the MPPA stage is shown for $n = 8$. Similar to Fig. 2.15, an 8-bit EAC-CSA made up of 8 parallel FAs is depicted as a single block in Fig. 2.17 for simplicity.

 Fig. 2.16 (a) MPPG for radix-4 Booth encoded modulo 2^n-1 multiplier (b) Radix-4 BE (c) Radix-4 BS

Fig. 2.17 MPPA for radix-4 Booth encoded modulo 2^n-1 multiplication

2.4.6 Modulo 2^n+1 multiplier

Various modulo 2^n+1 multipliers employing different number representations for the multiplicand, the multiplier and the product have been proposed in literature [Zimm99], [Hias92], [Wrzy93], [Verg07], [Wang96b], [Efst05], [Ma98], [Sous05], [Chen10]. In [Zimm99], modulo 2^n+1 multipliers, for use specifically in International Data Encryption Algorithm applications, were proposed using a proprietary number representation, where all numbers are represented in the weighted-binary form except 2^n , which is represented as an all zeros vector. Modulo 2^n+1 multiplier based on a binary multiplier was described in [Hias92]. [Hias92] relied on subtracting the n msbs of the binary multiplier result from its lsbs to derive the modulo-reduced product. In [Wrzy93], a modulo 2^n+1 multiplier using weighted-binary representation was suggested by exploiting the redundancy in the representation. The partial product matrix was divided into four groups and simplified by the observation that the bits from only one group can be different from zero. Hence, the partial product matrix was reduced from an $(n+1) \times (n+1)$ bit array to an $n \times n$ bit array. In [Verg07], the correction terms of [Wrzy93] were further simplified and represented as a single additional partial product.

However, the diminished-1 number system proposed in [Leib76] is most prevalent in modulo 2^n+1 multiplier designs [Wang96b], [Efst05], [Ma98], [Sous05], [Chen10]. The diminished-1 multiplier can be further classified on the basis of the encoding employed for the multiplier

bits: (a) Non-encoded multipliers, *e.g.*, [Wang96b] and [Efst05] (b) Radix-4 Booth encoded multipliers, *e.g.*, [Ma98], [Sous05] and [Chen10].

2.4.6.1 Non-encoded modulo 2^n+1 multiplier

Let $X' = \sum_{i=0}^{n-1} x_i 2^i$, $Y' = \sum_{i=0}^{n-1} y_i 2^i$ and $Z' = \sum_{i=0}^{n-1} z_i 2^i$ be the n -bit diminished-1 representations of the multiplicand, the multiplier and the product, respectively. Then the modulo 2^n+1 multiplication in diminished-1 representation is given by

$$\begin{aligned} |Z|_{2^n+1} &= |X \cdot Y|_{2^n+1} \\ |Z' + 1|_{2^n+1} &= |(X' + 1) \cdot (Y' + 1)|_{2^n+1} \\ |Z' + 1|_{2^n+1} &= |X' \cdot Y' + X' + Y' + 1|_{2^n+1} \\ |Z'|_{2^n+1} &= |X' \cdot Y' + X' + Y'|_{2^n+1} \end{aligned} \quad (2.78)$$

From (2.78), in diminished-1 representation, the final product is given by the sum of the multiplicand X' , the multiplier Y' as well as the product of the multiplicand and the multiplier $X'Y'$.

In [Wang96b], (2.78) was reformulated as

$$\begin{aligned} |Z'|_{2^n+1} &= \left| \sum_{i=0}^{n-1} y_i 2^i X' + \sum_{i=0}^{n-1} y_i 2^i + X' \right|_{2^n+1} \\ &= \left| \sum_{i=1}^{n-1} y_i 2^i X' + \sum_{i=1}^{n-1} y_i 2^i + (1 + y_0) X' + y_0 \right|_{2^n+1} \\ &= \left| \sum_{i=1}^{n-1} (PP_i - y_i 2^i + y_i) + \sum_{i=1}^{n-1} y_i 2^i + (1 + y_0) X' + y_0 \right|_{2^n+1} \\ &= \left| \sum_{i=1}^{n-1} PP_i + Q + R \right|_{2^n+1} \end{aligned} \quad (2.79)$$

where $Q = \sum_{i=1}^{n-1} y_i$ is the number of 1's in the most significant $n-1$ bits of Y' , $R = (1 + y_0)X' + y_0$.

PP_i is given by Table 2.4.

Table 2.4 Modulo 2^n+1 reduced partial products [Wang96b]

y_i	PP_i
0	$0\dots 0$
1	$x_{n-1-i} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-i}$

From (2.66), a k -operand (MOMA, 2^n+1) inherently adds the constant $k-1$ to the result. Therefore in (2.79), by using an $(n+1)$ -operand (MOMA, 2^n+1) to sum the $n-1$ PP_i 's, Q and R , Z' is incremented by n . To negate this effect, a constant n is subtracted from (2.79) and (2.79) is simplified to

$$|Z'|_{2^n+1} = \left| \sum_{i=1}^{n-1} PP_i + \bar{S} + R + 1 \right|_{2^n+1} \quad (2.80)$$

where S is the number of zeros in the most significant $n-1$ bits of Y' determined by an $(n-1)$ -bit counter. Moreover, the $n+1$ partial products PP_i , S and R were added in the CSA tree based implementation of $(n+1)$ -operand (MOMA, 2^n+1), while the final addition of 1 was performed in the two-operand modulo 2^n+1 adder.

In [Efst05], modulo 2^n+1 multiplication in diminished-1 representation of (2.78) was implemented as

$$\begin{aligned} |Z|_{2^n+1} &= |X' \cdot Y' + X' + Y'|_{2^n+1} \\ &= \left| \sum_{i=0}^{n-1} y_i 2^i X + X' + Y' \right|_{2^n+1} \\ &= \left| \sum_{i=0}^{n-1} (PP_i + C_i) + X' + Y' \right|_{2^n+1} \end{aligned} \quad (2.81)$$

where PP_i is tabulated in Table 2.5 and $C_i = 2^n(2^i-1)$ is the accompanying correction term.

Table 2.5 Modulo 2^n+1 reduced partial products [Efst05]

y_i	PP_i
0	$\underbrace{0\dots 0}_{n-i} \underbrace{1\dots 1}_i$
1	$x_{n-1-i} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-i}$

The sum of all correction terms, including the correction $-(n+2)$ to negate the constant incurred from the $(n+3)$ -operand (MOMA, 2^n+1), was shown to be equal to one. The aggregate correction term was included as a partial product by considering its diminished-1 form, *i.e.*, an n -bit all zeros vector, denoted as C' . Hence, (2.81) is simplified to

$$|Z|_{2^n+1} = \left| \sum_{i=0}^{n-1} PP_i + X' + Y' + C' \right|_{2^n+1} \quad (2.82)$$

Fig. 2.18 illustrates the required modulo-reduced partial products for $n = 8$. The eight partial products PP_0 to PP_7 can be implemented by an array of 8×8 AND/NAND gates and the remaining three partial products, PP_8 , PP_9 and PP_{10} , can be directly hardwired to X' , Y' and C' , respectively in the MPPG stage.

$PP_0 =$	x_7y_0	x_6y_0	x_5y_0	x_4y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0
$PP_1 =$	x_6y_1	x_5y_1	x_4y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1	$\overline{x_7y_1}$
$PP_2 =$	x_5y_2	x_4y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2	$\overline{x_7y_2}$	$\overline{x_6y_2}$
$PP_3 =$	x_4y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3	$\overline{x_7y_3}$	$\overline{x_6y_3}$	$\overline{x_5y_3}$
$PP_4 =$	x_3y_4	x_2y_4	x_1y_4	x_0y_4	$\overline{x_7y_4}$	$\overline{x_6y_4}$	$\overline{x_5y_4}$	$\overline{x_4y_4}$
$PP_5 =$	x_2y_5	x_1y_5	x_0y_5	$\overline{x_7y_5}$	$\overline{x_6y_5}$	$\overline{x_5y_5}$	$\overline{x_4y_5}$	$\overline{x_3y_5}$
$PP_6 =$	x_1y_6	x_0y_6	$\overline{x_7y_6}$	$\overline{x_6y_6}$	$\overline{x_5y_6}$	$\overline{x_4y_6}$	$\overline{x_3y_6}$	$\overline{x_2y_6}$
$PP_7 =$	x_0y_7	$\overline{x_7y_7}$	$\overline{x_6y_7}$	$\overline{x_5y_7}$	$\overline{x_4y_7}$	$\overline{x_3y_7}$	$\overline{x_2y_7}$	$\overline{x_1y_7}$
$PP_8 =$	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
$PP_9 =$	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
$PP_{10} =$	0	0	0	0	0	0	0	0

Fig. 2.18 MPPG for modulo 2^n+1 multiplier

Fig. 2.19 illustrates the CSA tree based implementation of the MPPA stage, where the 11 modulo-reduced partial products are accumulated in 9 CEAC-CSAs. In Fig. 2.19, an 8-bit CEAC-CSA consisting of 8 FAs, as shown in Fig. 2.12, is illustrated as a single block.

Fig. 2.19 MPPA for modulo 2^n+1 multiplier

2.4.6.2 Radix-4 Booth encoded modulo 2^n+1 multiplier

Modulo 2^n+1 multiplication using radix-4 Booth encoding technique for diminished-1 representation was introduced in [Ma98]. In [Ma98], the number of modulo-reduced partial products were lowered to $\lceil n/2 \rceil + 1$ and each partial product was $n+1$ bits wide. However, the accumulation of partial products was performed in an integer CSA array and the S and C vector outputs from the CSA accumulation were $(n + \log_2 n)$ bits wide that needed to be modulo-reduced further. The modulo-reduction entailed two additional CSAs in series with the CSA array. This technique for MPPA has been superseded by the use of the regular CEAC-CSA tree in [Sous05] and [Chen10].

In [Sous05], a unified architecture for modulo 2^n+1 multiplier using diminished-1 and weighted-binary number representations was proposed. The modulo 2^n+1 multiplication in diminished-1 representation was given by

$$\begin{aligned} |Z'+1|_{2^n+1} &= |(X'+1) \cdot (Y'+1)|_{2^n+1} \\ |Z'|_{2^n+1} &= |X' \cdot Y' + x_n Y' + y_n X' + (x_n \vee y_n) 2^n|_{2^n+1} \end{aligned} \quad (2.83)$$

where x_n is the zero flag bit and is set to 1 when $X'+1 = 0$.

On the other hand, the modulo 2^n+1 multiplication using $(n+1)$ -bit wide weighted-binary representation was given by

$$|Z|_{2^n+1} = |X_{n-1:0} \cdot Y_{n-1:0} - x_n Y_{n-1:0} - y_n X_{n-1:0} + (x_n \wedge y_n)|_{2^n+1} \quad (2.84)$$

A technique to compute the common term in (2.83) and (2.84), *i.e.*, the product of n -bit wide X' (or $X_{n-1:0}$) and Y' (or $Y_{n-1:0}$) was suggested. The common term was reformulated using the radix-4 Booth encoding technique as follows.

$$|X' \cdot Y'|_{2^n+1} \left(\text{or } |X_{n-1:0} \cdot Y_{n-1:0}|_{2^n+1} \right) = \sum_{i=0}^{\lfloor n/2 \rfloor} (PP_i + CT_i) \quad (2.85)$$

where PP_i and CT_i are generated according to Table 2.6.

Table 2.6 Modulo 2^n+1 reduced partial products for radix-4 Booth encoding [Sous05]

y_{2i+1}	y_{2i}	y_{2i-1}	$y_{2i-1} + y_{2i} - 2y_{2i+1}$	PP_i	CT_i
0	0	0	0	$\underbrace{0 \dots 0}_{n-2i} \underbrace{1 \dots 1}_{2i}$	-2^{2i}
0	0	1	+1	$x_{n-2i-1} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$	-2^{2i}
0	1	0	+1	$x_{n-2i-1} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$	-2^{2i}
0	1	1	+2	$x_{n-2i-2} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i-1}$	-2^{2i+1}
1	0	0	-2	$\bar{x}_{n-2i-2} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i-1}$	2^{2i+1}
1	0	1	-1	$\bar{x}_{n-2i-1} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i}$	2^{2i}
1	1	0	-1	$\bar{x}_{n-2i-1} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i}$	2^{2i}
1	1	1	0	$\underbrace{1 \dots 1}_{n-2i} \underbrace{0 \dots 0}_{2i}$	2^{2i}

The summation of $\lceil n/2 \rceil + 1$ CT_i was expressed mathematically as

$$\sum_{i=0}^{\lfloor n/2 \rfloor} (\bar{y}_{2i} y_{2i+1} \vee \bar{y}_{2i} y_{2i-1} \vee y_{2i-1} y_{2i+1}) 2^{2i} + \bar{y}_{2i+1} 2^{2i+1} + CT_0 - 4y_1 + 6 + 2\bar{n}_0 \quad (2.86)$$

where $n_0 = 1$ for odd n and $n_0 = 0$ otherwise. Furthermore, (2.86) was independently simplified for diminished-1 and weighted-binary representations by adding the terms apart from the common term in (2.83) and (2.84), respectively to CT_0 . PP_i and CT were shown to be generated using multiplexers and the resultant partial products were accumulated in a CEAC-CSA tree. For diminished-1 representation, the number of partial products was lowered to $\lfloor n/2 \rfloor + 1$ and $\lfloor n/2 \rfloor + 2$ for the cases of even and odd n , respectively.

In [Chen10], a more regular partial product generation scheme and a simpler expression for correction term were derived for radix-4 Booth encoded modulo 2^n+1 multiplication using diminished-1 representation. The product was given by

$$|Z'|_{2^n+1} = \left| \sum_{i=0}^{k-1} PP_i + \bar{C} + C' + k + 1 \right|_{2^n+1} \quad (2.87)$$

where $C = \sum_{i=0}^{k-1} c_i$ and C' is the diminished-1 representation of the constant one, *i.e.*, an all-zeros vector. PP_i as well as c_i are tabulated in Table 2.7.

Table 2.7 Modulo 2^n+1 reduced partial products for radix-4 Booth Encoding [Chen10]

y_{2i+1}	y_{2i}	y_{2i-1}	$y_{2i-1} + y_{2i} - 2y_{2i+1}$	PP_i	c_i
0	0	0	0	$\underbrace{0 \dots 0}_{n-2i} \underbrace{1 \dots 1}_{2i}$	2^{2i}
0	0	1	+1	$x_{n-2i-1} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$	0
0	1	0	+1	$x_{n-2i-1} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i}$	0
0	1	1	+2	$x_{n-2i-2} \dots x_0 \bar{x}_{n-1} \dots \bar{x}_{n-2i-1}$	0
1	0	0	-2	$\bar{x}_{n-2i-2} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i-1}$	0
1	0	1	-1	$\bar{x}_{n-2i-1} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i}$	0
1	1	0	-1	$\bar{x}_{n-2i-1} \dots \bar{x}_0 x_{n-1} \dots x_{n-2i}$	0
1	1	1	0	$\underbrace{0 \dots 0}_{n-2i} \underbrace{1 \dots 1}_{2i}$	2^{2i}

The generation of PP_i using Modified Booth Encoder (MBE) and Modified Booth Selector (MBS and MBS*) blocks is shown in Fig. 2.20. Moreover, $\bar{C}$ is generated in the form $1\bar{c}_{2(k-1)}1 \dots 1\bar{c}_21\bar{c}_0$ when n is even and $\bar{c}_{2(k-1)}1 \dots 1\bar{c}_21\bar{c}_0$ when n is odd, where c_{2i} is simply the

XOR of the $1x$ and $2x$ outputs from the MBE block. The k PP_i 's, $\bar{C}$ and C' are accumulated in a $(k+2)$ -operand (MOMA, 2^n+1), which inherently adds the constant $k+1$ in (2.87).

Fig. 2.20 (a) MPPG for radix-4 Booth encoded modulo 2^n+1 multiplier (b) Radix-4 MBE (c) Radix-4 MBS (d) Radix-4 MBS*

Chapter 3

Radix-8 Booth Encoded Modulo 2^n-1 Multiplier for Imbalanced Word-length Moduli Set based RNS

3.1 Introduction

The ubiquitous special moduli set is the three moduli set $\{2^n-1, 2^n, 2^n+1\}$ with a DR of only $3n$ bits. It is obvious that the DR of an existing moduli set can be extended by appending many small word-length moduli or a few large word-length moduli. In order to facilitate design of efficient modulo arithmetic units, it is worthwhile to keep the additional moduli in forms of 2^n or $2^n\pm 1$. Due to the conflicting requirements that the additional moduli be modulo-arithmetic friendly and at the same time be relatively prime to the moduli in the existing set, the number of additional moduli that can be included to expand an existing moduli set is limited. Hence, high-DR moduli sets with imbalanced moduli word-lengths are unavoidable. To transcend the $3n$ -bit limit, moduli sets, $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, $\{2^{2n}-1, 2^n, 2^{2n}+1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ with DR of $5n$ bits and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ with DR of $6n$ bits have been proposed recently [Cao03], [Moha07a], [Mola10]. These high DR moduli sets are crucial to public key cryptographic applications where key-sizes in the range of 512~1024 bits and 160~512 bits are typical in RSA and ECC algorithms, respectively.

The delay of an integer multiplication in RNS domain based on the $6n$ -bit DR moduli set of [Mola10] for example, is governed by the delay of the modulo $2^{2n}+1$ multiplier. As the time complexity of partial product summation by a CSA tree and a two-operand parallel-prefix adder is a logarithmic function of n , the critical path delay can be modeled as $O(\log 2n)$, but the delays of the modulo 2^n-1 and modulo 2^n+1 multipliers are only $O(\log n)$. This speedup of around $1/(\log_2 n + C) \times 100\%$, where C is a constant, by modulo 2^n-1 and modulo 2^n+1 multipliers over the critical path delay is of no consequence. The non-critical modulo

multipliers can be made to operate at a slower speed that nearly matches the delay of the critical modulo multiplier. In doing so, the timing slack freed up from the modulo 2^n-1 and modulo 2^n+1 multipliers can be effectively explored for more area and power efficient architectures without compromising the overall system performance. This approach to reduce the system area and power consumption is an architectural design space exploration and can be implemented with any standard cell library. It does not require multiple supply voltages, multiple threshold voltages or control circuitries for the generation and scaling of voltage and frequency in order to exploit the timing surplus in the non-critical paths for power saving.

This chapter focuses on one of the two special moduli, *i.e.*, 2^n-1 . The first-ever family of radix-8 Booth encoded modulo 2^n-1 multipliers with delay adaptable to match the RNS multiplier delay is proposed.

The properties of modulo 2^n-1 arithmetic were most effectively exploited for FA based implementation of modulo 2^n-1 multiplier in [Wang96a], [Zimm99] and [Efst04a]. As explained in Chapter 2, the number of partial products in the non-encoded multiplier of [Wang96a] and the radix-4 Booth encoded multipliers of [Zimm99] and [Efst04a] were n , $\lfloor n/2 \rfloor + 1$ and $\lceil n/2 \rceil$, respectively. With higher radix Booth encoding the number of partial products is reduced by more than half and consequently, significant reduction in silicon area and power dissipation is feasible [Cher97], [Flynn01]. The radix-8 Booth encoding reduces the number of partial products to $\lfloor n/3 \rfloor + 1$, which is more aggressive than the radix-4 Booth encoding. However, in the radix-8 Booth encoded modulo 2^n-1 multiplication, not all modulo-reduced partial products can be generated using the bitwise circular-left-shift operation and bitwise inversion. Particularly, the performance overhead in computing the hard multiple $|+3X|_{2^n-1}$ is by no means trivial and hence, the use of Booth encoding for modulo 2^n-1 multipliers has been restricted to only radix-4 in literature.

This chapter is organized as follows. Section 3.2 describes the radix-8 Booth encoding algorithm for modulo 2^n-1 multiplication and highlights the issue of hard multiple generation. An efficient technique to generate the hard multiple in a partially-redundant biased representation is presented in Section 3.3. Furthermore in Section 3.3, a family of radix-8 Booth encoded modulo 2^n-1 multipliers with adaptive delay is described. The

adaptive delay of the proposed family of modulo 2^n-1 multipliers is corroborated and the performance of the proposed multipliers is compared against the radix-4 Booth encoded modulo 2^n-1 multiplier [Efst04a] in Section 3.4. A brief summary is presented in Section 3.5.

3.2 Radix-8 Booth encoded modulo 2^n-1 multiplication algorithm

In the following, dual representation of zeros, *i.e.*, n -bit all zeros and n -bit all ones is used to represent the residues of the modulus 2^n-1 and all equations are assumed to be modulo-reduced by 2^n-1 unless otherwise specified. Let $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{i=0}^{n-1} y_i \cdot 2^i$ and $Z = \sum_{i=0}^{n-1} z_i \cdot 2^i$ represent the multiplicand, the multiplier and the product of modulo 2^n-1 multiplication, respectively. Then, modulo 2^n-1 multiplication can be expressed as

$$\begin{aligned} |Z|_{2^n-1} &= |X \cdot Y|_{2^n-1} \\ &= \left| \sum_{i=0}^{n-1} X \cdot y_i 2^i \right|_{2^n-1} \end{aligned} \quad (3.1)$$

The radix-8 Booth encoding algorithm can be viewed as a digit set conversion of four consecutive overlapping multiplier bits, $y_{3i+2} y_{3i+1} y_{3i} (y_{3i-1})$ to a signed digit, d_i , $d_i \in [-4, 4]$, for $i = 0, 1, \dots, \lfloor n/3 \rfloor$. The digit set conversion is formally expressed as

$$d_i = y_{3i-1} + y_{3i} + 2y_{3i+1} - 4y_{3i+2} \quad (3.2)$$

where $y_{-1} = y_n = y_{n+1} = y_{n+2} = 0$ [Fly01]. By encoding the multiplier bits, modulo 2^n-1 multiplication is simplified to

$$|Z|_{2^n-1} = \left| \sum_{i=0}^{\lfloor n/3 \rfloor} X \cdot d_i 2^{3i} \right|_{2^n-1} \quad (3.3)$$

The following three properties are employed to simplify the modulo 2^n-1 multiplier design.

Property 3.1: As a corollary of the periodicity property of modulo-reduced powers-of-two expressed as *Property 2.1*, the modulo reduction of the binary weight of x in excess of the modulus is given by

$$\left| x \cdot 2^{n+i} \right|_{2^n-1} = x \cdot 2^i \quad (3.4)$$

Property 3.2: Let $CLS(X, j)$ denote the circular-left-shift of X by j bit positions. Using *Property 3.1* to modulo-reduce the binary weights in excess of the modulus, it can be shown that modulo 2^n-1 multiplication of X by a positive power-of-two term 2^j is equivalent to a CLS operation on X .

$$\begin{aligned} \left| X \cdot 2^j \right|_{2^n-1} &= \left| \sum_{i=0}^{n-1} x_i \cdot 2^{i+j} \right|_{2^n-1} \\ &= \sum_{i=0}^{n-j-1} x_i \cdot 2^{i+j} + \sum_{i=n-j}^{n-1} x_i \cdot 2^{i+j-n} \\ &= CLS(X, j) \end{aligned} \quad (3.5)$$

Property 3.3: By substituting for the additive inverse $-X$ as $\bar{X}$ using *Property 2.3*, modulo 2^n-1 multiplication of X by a negative power-of-two term (-2^j) can be shown to be equivalent to a CLS operation on $\bar{X}$.

$$\begin{aligned} \left| X \cdot (-2^j) \right|_{2^n-1} &= \left| \bar{X} \cdot 2^j \right|_{2^n-1} \\ &= CLS(\bar{X}, j) \end{aligned} \quad (3.6)$$

By *Properties 3.2* and *3.3*, the modulo-reduced multiples, $\left| X \cdot d_i \right|_{2^n-1}$ and partial products, $\left| X \cdot d_i \cdot 2^{3i} \right|_{2^n-1}$ for all possible values of the radix-8 Booth encoded multiplier digit d_i are tabulated in Table 3.1. In Table 3.1, the modulo-reduced partial products for $d_i \in \{\pm 0, \pm 1, \pm 2, \pm 4\}$ are replaced by simple bitwise inversion and/or bitwise circular-left-shift of X . In contrast, in the computation of the hard multiple $\left| +3X \right|_{2^n-1}$, the modulo 2^n-1 addition of X and $2X$ is unavoidable.

Table 3.1 Modulo 2^n-1 reduced multiples and partial products for radix-8 Booth encoding

d_i	$\left d_i \cdot X \right _{2^n-1}$	$\left X \cdot d_i \cdot 2^{3i} \right _{2^n-1} = PP_i$	d_i	$\left d_i \cdot X \right _{2^n-1}$	$\left X \cdot d_i \cdot 2^{3i} \right _{2^n-1} = PP_i$
+0	$\underbrace{0 \cdots 0}_n$	$CLS(\underbrace{0 \cdots 0}_n, 3i)$	-0	$\underbrace{1 \cdots 1}_n$	$CLS(\underbrace{1 \cdots 1}_n, 3i)$
+1	X	$CLS(X, 3i)$	-1	$\bar{X}$	$CLS(\bar{X}, 3i)$
+2	$CLS(X, 1)$	$CLS(X, 3i+1)$	-2	$CLS(\bar{X}, 1)$	$CLS(\bar{X}, 3i+1)$
+3	$\left +3X \right _{2^n-1}$	$CLS(\left +3X \right _{2^n-1}, 3i)$	-3	$\overline{\left +3X \right _{2^n-1}}$	$CLS(\overline{\left +3X \right _{2^n-1}}, 3i)$
+4	$CLS(X, 2)$	$CLS(X, 3i+2)$	-4	$CLS(\bar{X}, 2)$	$CLS(\bar{X}, 3i+2)$

In Chapter 2, modulo 2^n-1 addition was shown to be equivalent to an n -bit EAC addition. Numerous high-speed parallel-prefix implementations of EAC addition were proposed in [Zimm99], [Kala00], [Dimi03], [Dimi05b]. However, the use of such fast adders to compute the hard multiple contradicts the objective to exploit the timing slack of the non-critical modulo 2^n-1 multiplier channel. Of all possible two operand adder implementations, the Ripple Carry Adder (RCA) has indubitably the least area and dynamic power dissipation [Raba03], [Nage96]. Fig. 3.1 illustrates the computation of $|+3X|_{2^n-1}$ by an n -bit EAC addition of $|X|_{2^n-1}$ and $|2X|_{2^n-1} = CLS(X,1)$ using RCAs for $n = 8$. The addends $|X|_{2^n-1}$ and $|2X|_{2^n-1}$ are added with carry propagation through FAs, and the resultant sum and carry-out are added with carry propagation through half adders (HAs), as shown in Fig. 3.1.

Fig. 3.1 Generation of hard multiple $|+3X|_{2^n-1}$ using n -bit RCAs

This technique for $|+3X|_{2^n-1}$ computation involves two n -bit carry-propagate additions in series such that the carry propagation length is twice the operand length, n . Since the absolute difference in delay between the non-critical modulo 2^n-1 channel and the critical modulo m channel in a RNS multiplier depends on the degree of imbalance in the word-lengths of moduli, the delays cannot be equalized by arbitrarily fixing the carry propagation length to $2n$ bits. Instead, it is proposed to adaptively control the modulo 2^n-1 multiplier delay to match the critical m modulo multiplier delay by representing the hard multiple in a partially-redundant form [Bewi94].

3.3 Proposed radix-8 Booth encoded modulo 2^n-1 multiplier

3.3.1 Generation of partially-redundant and biased hard multiple

Let $|X|_{2^n-1}$ and $|2X|_{2^n-1}$ be added by a group of $M (= n/k)$ k -bit RCAs such that there is no carry propagation between the adders. Fig. 3.2 shows this addition for $n = 8$ and $k = 4$, where the sum and carry-out bits from the RCA block j are represented as s_i^j and c_i^j for $i \in [0, k-1]$ and $j \in [0, M-1]$, respectively. In Fig. 3.2, the carry-out of RCA 0, c_3^0 , is not propagated to the carry input of RCA 1 but preserved as one of the partial product bits to be accumulated. However, the binary weight of the carry-out c_3^1 of RCA 1 has exceeded the modulus and has to be modulo reduced before it can be accumulated.

By *Property 3.1*, the binary weight of c_3^1 can be reduced from 2^8 to 2^0 . Thus, c_3^1 is inserted at the lsb position in Fig. 3.2. It should be stressed that the carry-out c_3^1 is a partial-carry propagated through only k most significant FAs and hence, is different from the EAC bit in the modulo 2^n-1 addition of X and $2X$, i.e., c_7 of Fig.3.1.

Fig. 3.2 Generation of partially-redundant hard multiple $|+3X|_{2^n-1}$ using k -bit RCAs

From Fig. 3.2, the partially-redundant form of $|+3X|_{2^n-1}$ is given by the partial-sum and partial-carry pair (S, C) , where

$$\begin{aligned} S &= s_{k-1}^{M-1} s_{k-2}^{M-1} \cdots s_0^{M-1} \cdots s_{k-1}^0 s_{k-2}^0 \cdots s_0^0 \\ C &= \underbrace{0 \cdots 0}_{k-1} c_{k-1}^{M-2} \cdots \underbrace{0 \cdots 0}_{k-1} c_{k-1}^0 \underbrace{0 \cdots 0}_{k-1} c_{k-1}^{M-1} \end{aligned} \quad (3.7)$$

Since modulo 2^n-1 negation is equivalent to bitwise complementation by *Property 2.3*, the

negative hard multiple in the partially-redundant form, $|-3X|_{2^n-1} = (\bar{S}, \bar{C})$ is given by

$$\begin{aligned}\bar{S} &= \bar{s}_{k-1}^{M-1} \bar{s}_{k-2}^{M-1} \dots \bar{s}_0^{M-1} \dots \bar{s}_{k-1}^0 \bar{s}_{k-2}^0 \dots \bar{s}_0^0 \\ \bar{C} &= \underbrace{1 \dots 1}_{k-1} \bar{c}_{k-1}^{M-2} \dots \underbrace{1 \dots 1}_{k-1} \bar{c}_{k-1}^0 \underbrace{1 \dots 1}_{k-1} \bar{c}_{k-1}^{M-1}\end{aligned}\quad (3.8)$$

To avoid having long strings of ones in $\bar{C}$, an appropriate bias B , is added to the partially-redundant form of the hard multiple (S, C) such that both the positive and negative hard multiples are sparse [Bewi94]. The value of B is chosen as

$$B = \sum_{j=0}^{M-1} 2^{k \cdot j} = \underbrace{0 \dots 0 1}_{k} \dots \underbrace{0 \dots 0 1}_{k} \quad (3.9)$$

Since B is chosen to be a binary word that has logic ones at bit positions 2^{kj} and logic zeros at other bit positions, $|B + 3X|_{2^n-1}$ can be generated by simple XNOR and OR operations on the bits of S and C at bit positions 2^{kj} as illustrated in Fig. 3.3.

Fig. 3.3 Generation of partially-redundant biased hard multiple $|B + 3X|_{2^n-1}$ using k -bit RCAs

In general, the partially-redundant biased hard multiple $|B + 3X|_{2^n-1}$ is given by the partial-sum and partial-carry pair (BS, BC) such that

$$\begin{aligned}BS &= s_{k-1}^{M-1} s_{k-2}^{M-1} \dots s_0^{M-1} \dots s_{k-1}^0 s_{k-2}^0 \dots s_0^0 \\ BC &= \underbrace{0 \dots 0}_{k-2} b c_{k-1}^{M-2} 0 \dots \underbrace{0 \dots 0}_{k-2} b c_{k-1}^0 \underbrace{0 \dots 0}_{k-2} b c_{k-1}^{M-1} 0\end{aligned}\quad (3.10)$$

where

$$bs_0^j = \begin{cases} \overline{s_0^j \oplus c_{k-1}^{j-1}} & \text{when } j \neq 0 \\ \overline{s_0^0 \oplus c_{k-1}^{M-1}} & \text{when } j = 0 \end{cases} \quad (3.11)$$

and

$$bc_{k-1}^j = \begin{cases} s_0^{j+1} + c_{k-1}^j & \text{when } j \neq M-1 \\ s_0^0 + c_{k-1}^{M-1} & \text{when } j = M-1 \end{cases} \quad (3.12)$$

for $j = 0, 1, \dots, M-1$.

Let

$$\begin{aligned} \overline{BS} &= \overline{s_{k-1}^{M-1} s_{k-2}^{M-1} \dots bs_0^{M-1} \dots s_{k-1}^0 s_{k-2}^0 \dots bs_0^0} \\ \overline{BC} &= \underbrace{0 \dots 0}_{k-2} \overline{bc_{k-1}^{M-2}} 0 \dots \underbrace{0 \dots 0}_{k-2} \overline{bc_{k-1}^0} 0 \underbrace{0 \dots 0}_{k-2} \overline{bc_{k-1}^{M-1}} 0 \end{aligned} \quad (3.13)$$

It can be verified that the sum of (BS, BC) and $(\overline{BS}, \overline{BC})$ modulo 2^n-1 is $|2B|_{2^n-1}$. Therefore,

$(\overline{BS}, \overline{BC})$ represents the partially-redundant form of $|B-3X|_{2^n-1}$.

3.3.2 Generation of partially-redundant and biased simple multiples

Similar to the hard multiple, other Booth encoded multiples corresponding to $d_i \in \{\pm 0, \pm 1, \pm 2, \pm 4\}$ listed in Table 3.1 must also be biased and generated in partially-redundant forms.

Fig. 3.4 shows the biased simple multiples, $|B+0|_{2^n-1}$, $|B+X|_{2^n-1}$, $|B+2X|_{2^n-1}$ and $|B+4X|_{2^n-1}$ represented in a partially-redundant form for $n = 8$ and $k=4$. From Fig. 3.4, it can be seen that the generation of the partially-redundant and biased simple multiples involves only shift and selective complementation of the multiplicand bits without additional hardware overhead.

0	0	0	1	0	0	0	1	$B+0$
		0				0		
x_7	x_6	x_5	$\bar{x}_4$	x_3	x_2	x_1	$\bar{x}_0$	$B+X$
		x_4				x_0		
x_6	x_5	x_4	$\bar{x}_3$	x_2	x_1	x_0	$\bar{x}_7$	$B+2X$
		x_3				x_7		
x_5	x_4	x_3	$\bar{x}_2$	x_1	x_0	x_7	$\bar{x}_6$	$B+4X$
		x_2				x_6		

Fig. 3.4 Generation of partially-redundant biased simple multiples

3.3.3 Generation of partially-redundant and biased partial products

Using the partially-redundant and biased representation for the partial products, radix-8 Booth encoded modulo 2^n-1 multiplication of (3.3) is reformulated as

$$\begin{aligned}
 |Z|_{2^n-1} &= \left| \sum_{i=0}^{\lfloor n/3 \rfloor} (B + d_i X) \cdot 2^{3i} - \sum_{i=0}^{\lfloor n/3 \rfloor} B \cdot 2^{3i} \right|_{2^n-1} \\
 &= \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + CC \right|_{2^n-1}
 \end{aligned} \tag{3.14}$$

where the Compensation Constant (CC) that negates the effect of the bias B is defined as

$$CC = \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} B \cdot 2^{3i} \right|_{2^n-1} \tag{3.15}$$

In the following subsections, the computation of CC and the generation of $\lfloor n/3 \rfloor + 1$ PP_i s are described.

3.3.3.1 Computation of Compensation Constant (CC)

As CC depends on the bias $B = \sum_{i=0}^{M-1} 2^{kj}$ or equivalently on the RCA word-length k , the criteria for selecting k is stipulated first.

Criterion 1: As the residues of modulus 2^n-1 are represented using only n bits, it is

imperative that k divides n . $k = 1$ is a trivial case and is excluded from this consideration. This criterion is expressed as $k \mid n, k \neq 1$.

Criterion 2: Since each partial product in radix-8 Booth encoding is shifted by three bits relative to the previous partial product, k must not be a multiple of three. Therefore, $3 \nmid k$.

In this section, CC of (3.15) is shown to be a single constant binary word with a specific repetitive pattern of logic ones and zeros for different choices of k stipulated by *Criteria 1* and 2. Due to *Criterion 2*, the proof is constructed separately for $3 \nmid n$ and $3 \mid n$.

A. n is not divisible by three, i.e., $3 \nmid n$

Since n is not divisible by three, k can be chosen as any integer in the set $[n, n/2, n/4, n/5, \dots, 2]$.

Case A.1: If $k = n$, then $M = n/k = 1$ and $B = 1$ from (3.9). By substituting $B = 1$ in (3.15),

$$CC = \left\lfloor - \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} \right\rfloor_{2^n-1} \quad (3.16)$$

Since $3 \nmid n$, $\text{mod}(n, 3) = 1$ or 2.

Case A.1.1: $\text{mod}(n, 3) = 1$

The upper limit of the summation of (3.16) is replaced by $(n-1)/3$.

$$CC = \left\lfloor - \sum_{i=0}^{(n-1)/3} 2^{3i} \right\rfloor_{2^n-1} \quad (3.17)$$

By the definition of additive inverse in *Property 2.3*, (3.17) can be simplified to an n -bit one's complementation of $\sum_{i=0}^{(n-1)/3} 2^{3i}$. Therefore,

$$CC = \sum_{i=0}^{(n-4)/3} (2^{3i+1} + 2^{3i+2}) \quad (3.18)$$

Case A.1.2: $\text{mod}(n, 3) = 2$

The upper limit of the summation of (3.16) is replaced by $(n-2)/3$.

$$\begin{aligned} CC &= \left| - \sum_{i=0}^{(n-2)/3} 2^{3i} \right|_{2^n-1} \\ &= \sum_{i=0}^{(n-2)/3} 2^{3i+1} + \sum_{i=0}^{(n-5)/3} 2^{3i+2} \end{aligned} \quad (3.19)$$

In both cases, from (3.18) and (3.19), CC is an n -bit binary word consisting of logic one at bit positions $3i+1$ and $3i+2$ and logic zero at other bit positions.

Case A.2: For any other legitimate value of k , $k \neq n$, from (3.9),

$$\begin{aligned} B &= \underbrace{0 \cdots 01}_k \parallel \underbrace{0 \cdots 01}_k \parallel \cdots \parallel \underbrace{0 \cdots 01}_k \\ &= \underbrace{B_k^0 \parallel B_k^0 \parallel \cdots \parallel B_k^0}_n \end{aligned} \quad (3.20)$$

where $\parallel$ is the concatenation operator and B_k^0 is a k -bit substring with only its lsb set to logic one. By substituting B into (3.15) and applying *Property 3.2* to simplify the multiplication by 2^{3i} ,

$$CC = \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} CLS(B_k^0 \parallel B_k^0 \parallel \cdots \parallel B_k^0, 3i) \right|_{2^n-1} \quad (3.21)$$

Property 3.4:

$$\begin{aligned} CLS(B, j) &= CLS(B_k^0 \parallel B_k^0 \parallel \cdots \parallel B_k^0, j) \\ &= CLS(B_k^0, j) \parallel CLS(B_k^0, j) \parallel \cdots \parallel CLS(B_k^0, j) \\ &= B_k^j \parallel B_k^j \parallel \cdots \parallel B_k^j \end{aligned} \quad (3.22)$$

where B_k^j is a k -bit substring with a logic one at the $(j \bmod k)$ -th bit position.

Simplifying the *CLS* operation in (3.21) by *Property 3.4*,

$$CC = \left\lfloor - \sum_{i=0}^{\lfloor n/3 \rfloor} B_k^{3i} \parallel B_k^{3i} \parallel \dots \parallel B_k^{3i} \right\rfloor_{2^n-1} \quad (3.23)$$

Fig. 3.5 illustrates the two-operand modulo 2^n-1 addition of n -bit binary strings, each composed of identical substrings of the form $B_k^j \parallel B_k^j \parallel \dots \parallel B_k^j$. In Fig. 3.5, $B_k^0 \parallel \dots \parallel B_k^0$ and $B_k^1 \parallel \dots \parallel B_k^1$ are considered as the addends.

$B_k^0 \parallel \dots \parallel B_k^0$	0 ... 0 0 1	0 ... 0 0 1	...	0 ... 0 0 1
$B_k^1 \parallel \dots \parallel B_k^1$	0 ... 0 1 0	0 ... 0 1 0	...	0 ... 0 1 0
Sum	0 ... 0 1 1	0 ... 0 1 1	...	0 ... 0 1 1
Carry	(0) 0 ... 0 0 0	0 ... 0 0 0	...	0 ... 0 0 0
EAC				0
Final sum	0 ... 0 1 1	0 ... 0 1 1		0 ... 0 1 1

Fig. 3.5 Modulo 2^n-1 addition of $B_k^0 \parallel B_k^0 \parallel \dots \parallel B_k^0$ and $B_k^1 \parallel B_k^1 \parallel \dots \parallel B_k^1$

From Fig. 3.5, the substring at the output, 0...011 can be computed by a k -bit EAC addition, *i.e.*, modulo 2^k-1 addition of B_k^0 and B_k^1 . By extending the above observation to multi-operand addition, *Property 3.5* shows that the modulo 2^n-1 summation of binary strings, each composed of identical k -bit substrings, is equivalent to the modulo 2^k-1 summation of the substrings followed by concatenation.

Property 3.5:

$$\begin{aligned} & \left\lfloor (B_k^0 \parallel B_k^0 \parallel \dots \parallel B_k^0) + (B_k^1 \parallel B_k^1 \parallel \dots \parallel B_k^1) + \dots + (B_k^j \parallel B_k^j \parallel \dots \parallel B_k^j) \right\rfloor_{2^n-1} \\ &= \left\lfloor B_k^0 + B_k^1 + \dots + B_k^j \right\rfloor_{2^k-1} \parallel \left\lfloor B_k^0 + B_k^1 + \dots + B_k^j \right\rfloor_{2^k-1} \parallel \dots \parallel \left\lfloor B_k^0 + B_k^1 + \dots + B_k^j \right\rfloor_{2^k-1} \end{aligned} \quad (3.24)$$

Simplifying the modulo summation operation in (3.23) using *Property 3.5*, CC is given by

$$CC = \left\lfloor - \sum_{i=0}^{\lfloor n/3 \rfloor} B_k^{3i} \right\rfloor_{2^k-1} \parallel \dots \parallel \left\lfloor - \sum_{i=0}^{\lfloor n/3 \rfloor} B_k^{3i} \right\rfloor_{2^k-1} \quad (3.25)$$

For ease of exposition, CC is derived for $k = n/2$ and *Properties 3.4* and *3.5* can be applied to

the remaining cases of k to show that CC is indeed a constant binary word. From (3.25), for $k = n/2$, CC is given by

$$CC = CC_{n/2} \parallel CC_{n/2} \quad (3.26)$$

where

$$\begin{aligned} CC_{n/2} &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} B_{n/2}^{3i} \right|_{2^{n/2}-1} \\ &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} CLS \left(\underbrace{0 \dots 01}_{n/2}, 3i \right) \right|_{2^{n/2}-1} \\ &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} \right|_{2^{n/2}-1} \end{aligned} \quad (3.27)$$

The binary exponents greater than $n/2$ in (3.27) can be modulo reduced by *Property 3.1* to

$$CC_{n/2} = \left| - \sum_{i=0}^{\lfloor k/3 \rfloor} 2^{3i} - \sum_{i=\lfloor k/3 \rfloor+1}^{\lfloor n/3 \rfloor} 2^{3i-k} \right|_{2^{n/2}-1} \quad (3.28)$$

where $k = n/2$ and $\text{mod}(k, 3) = 1$ or 2 since $3 \nmid k$.

Case A.2.1: $\text{mod}(k, 3) = 1$

By substituting $\lfloor k/3 \rfloor = (k-1)/3$ in the summation bounds of (3.28),

$$CC_{n/2} = \left| - \sum_{i=0}^{(k-1)/3} 2^{3i} - \sum_{i=(k+2)/3}^{(2k-2)/3} 2^{3i-k} \right|_{2^{n/2}-1} \quad (3.29)$$

By adjusting the indices of the second summation,

$$\begin{aligned} CC_{n/2} &= \left| - \sum_{i=0}^{(k-1)/3} 2^{3i} - \sum_{i=0}^{(k-1)/3-1} 2^{3i+2} \right|_{2^{n/2}-1} \\ &= \sum_{i=0}^{(k-1)/3-1} 2^{3i+1} \end{aligned} \quad (3.30)$$

Case A.2.2: $\text{mod}(k, 3) = 2$

By substituting $\lfloor k/3 \rfloor = (k-2)/3$ in the summation bounds of (3.28),

$$\begin{aligned}
 CC_{n/2} &= \left| - \sum_{i=0}^{(k-2)/3} 2^{3i} - \sum_{i=(k+1)/3}^{(2k-1)/3} 2^{3i-k} \right|_{2^{n/2}-1} \\
 &= \left| - \sum_{i=0}^{(k-2)/3} 2^{3i} - \sum_{i=0}^{(k-2)/3} 2^{3i+1} \right|_{2^{n/2}-1} \\
 &= \sum_{i=0}^{(k-2)/3-1} 2^{3i+2}
 \end{aligned} \tag{3.31}$$

From the final expressions of (3.30) and (3.31), $CC_{n/2}$ for *Cases A.2.1* and *A.2.2* is an $n/2$ -bit binary word consisting of logic one at bit positions $3i+1$ and $3i+2$, respectively and logic zero elsewhere.

The binary value of CC for the various combinations of n and k can be precomputed using similar derivation when n is not a multiple of three. Table 3.2 summarizes CC for $k = n$, $n/2$ and $n/4$. For $k = n/2$ and $n/4$, the substrings $CC_{n/2}$ and $CC_{n/4}$ are listed and CC can be obtained by $CC_{n/2} \parallel CC_{n/2}$ and $CC_{n/4} \parallel CC_{n/4} \parallel CC_{n/4} \parallel CC_{n/4}$, respectively. In Table 3.2, $\underbrace{\text{###}}_{\times i}$ denote that the string ### is repeated i times.

Table 3.2 Compensation Constant when n is not divisible by three

n	$CC(k=n)$	$CC_{n/2}(k=n/2)$	$CC_{n/4}(k=n/4)$
8	$\underbrace{10110}_{\times 2}$	$\underbrace{0010}_{\times 1}$	10
16	$\underbrace{0110}_{\times 5}$	$\underbrace{00100}_{\times 2}$	$\underbrace{0110}_{\times 1}$
20	$\underbrace{10110}_{\times 6}$	$\underbrace{0010}_{\times 3}$	$\underbrace{10110}_{\times 1}$
28	$\underbrace{0110}_{\times 9}$	$\underbrace{00100}_{\times 4}$	$\underbrace{0110}_{\times 2}$
32	$\underbrace{10110}_{\times 10}$	$\underbrace{0010}_{\times 5}$	$\underbrace{10110}_{\times 2}$
40	$\underbrace{0110}_{\times 13}$	$\underbrace{00100}_{\times 6}$	$\underbrace{0110}_{\times 3}$
44	$\underbrace{10110}_{\times 14}$	$\underbrace{0010}_{\times 7}$	$\underbrace{10110}_{\times 3}$
52	$\underbrace{0110}_{\times 17}$	$\underbrace{00100}_{\times 8}$	$\underbrace{0110}_{\times 4}$
56	$\underbrace{10110}_{\times 18}$	$\underbrace{0010}_{\times 9}$	$\underbrace{10110}_{\times 4}$
64	$\underbrace{0110}_{\times 21}$	$\underbrace{00100}_{\times 10}$	$\underbrace{0110}_{\times 5}$

B. n is divisible by three, i.e., $3|n$

When n is divisible by three, the choice of k is limited to the integers in the set $[n/3, n/6, n/9, \dots, 2]$ that are not divisible by three according to *Criterion 2*. The derivation of CC for $k = n/3$ is illustrated here and CC for all other cases of k can be similarly derived.

From (3.25), for $k = n/3$,

$$CC = CC_{n/3} \parallel CC_{n/3} \parallel CC_{n/3} \quad (3.32)$$

where

$$\begin{aligned} CC_{n/3} &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} B_{n/3}^{3i} \right|_{2^{n/3}-1} \\ &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} CLS \left(\underbrace{0 \dots 01}_{n/3}, 3i \right) \right|_{2^{n/3}-1} \\ &= \left| - \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} \right|_{2^{n/3}-1} \end{aligned} \quad (3.33)$$

The binary exponents greater than $n/3$ in (3.33) are modulo reduced by *Property 3.1* to

$$CC_{n/3} = \left| - \sum_{i=0}^{\lfloor k/3 \rfloor} 2^{3i} - \sum_{i=\lfloor k/3 \rfloor+1}^{\lfloor 2k/3 \rfloor} 2^{3i-k} - \sum_{i=\lfloor 2k/3 \rfloor+1}^{k-1} 2^{3i-2k} - \sum_{i=k}^k 2^{3i-3k} \right|_{2^{n/3}-1} \quad (3.34)$$

where $k = n/3$ and $\text{mod}(k, 3) = 1$ or 2 since $3 \nmid k$.

Case B.1: $\text{mod}(k, 3) = 1$

By substituting $\lfloor k/3 \rfloor = (k-1)/3$ in the summation bounds of (3.34),

$$\begin{aligned} CC_{n/3} &= \left| - \sum_{i=0}^{(k-1)/3} 2^{3i} - \sum_{i=(k+2)/3}^{(2k-2)/3} 2^{3i-k} - \sum_{i=(2k+1)/3}^{k-1} 2^{3i-2k} - 1 \right|_{2^{n/3}-1} \\ &= \left| - \sum_{i=0}^{(k-1)/3} 2^{3i} - \sum_{i=0}^{(k-1)/3-1} 2^{3i+2} - \sum_{i=0}^{(k-1)/3-1} 2^{3i+1} - 1 \right|_{2^{n/3}-1} \\ &= \sum_{i=1}^{(k-1)/3} 2^{3i} + \sum_{i=0}^{(k-1)/3-1} 2^{3i+1} + \sum_{i=0}^{(k-1)/3-1} 2^{3i+2} \end{aligned} \quad (3.35)$$

Case B.2: $\text{mod}(k, 3) = 2$

By substituting $\lfloor k/3 \rfloor = (k-2)/3$ in the summation bounds of (3.34),

$$\begin{aligned} CC_{n/3} &= \left| - \sum_{i=0}^{(k-2)/3} 2^{3i} - \sum_{i=(k+1)/3}^{(2k-1)/3} 2^{3i-k} - \sum_{i=2(k+1)/3}^{k-1} 2^{3i-2k} - 1 \right|_{2^{n/3}-1} \\ &= \sum_{i=1}^{(k-2)/3} 2^{3i} + \sum_{i=0}^{(k-2)/3} 2^{3i+1} + \sum_{i=0}^{(k-2)/3-1} 2^{3i+2} \end{aligned} \quad (3.36)$$

From (3.35) and (3.36), $CC_{n/3}$ is an $n/3$ -bit binary word that is set to logic one in all but the lsb position in both cases.

The binary values of CC for different combinations of n and k , when n is a multiple of three, are reported in Table 3.3. For k chosen as $n/3$ and $n/6$, CC is given by $CC_{n/3} \parallel CC_{n/3} \parallel CC_{n/3}$ and $CC_{n/6} \parallel CC_{n/6} \parallel CC_{n/6} \parallel CC_{n/6} \parallel CC_{n/6} \parallel CC_{n/6}$, respectively.

Table 3.3 Compensation Constant when n is divisible by three

n	$CC_{n/3} (k = n/3)$	$CC_{n/6} (k = n/6)$
12	1110	10
24	$\underbrace{1111110}_{\times 1}$	1110
48	$\underbrace{1111110}_{\times 4}$	$\underbrace{1111110}_{\times 1}$
60	$\underbrace{1111110}_{\times 5}$	$\underbrace{1111110}_{\times 2}$

3.3.3.2 Generation of PP_i s

From (3.14), PP_i is defined as

$$PP_i = \left| 2^{3i} (B + d_i \cdot X) \right|_{2^n-1} \quad (3.37)$$

Using *Property 3.2*, the modulo 2^n-1 multiplication by 2^{3i} in (3.37) is efficiently implemented as bitwise circular-left-shift of the biased multiple, $(B + d_i \cdot X)$. For $n = 8$ and $k = 4$, Fig. 3.6 illustrates the partial product matrix of $|X \cdot Y|_{2^8-1}$ with $(\lfloor n/3 \rfloor + 1)$ partial products in partially-redundant biased representation. Each PP_i consists of an n -bit vector, $pp_{i7} \dots pp_{i1}$ pp_{i0} and a vector of $n/k = 2$ redundant carry bits, q_{i1} and q_{i0} . Since q_{i0} and q_{i1} are the carry-out bits of the RCAs, they are displaced by k -bit positions for a given PP_i . The bit, q_{ij} is displaced circularly to the left of $q_{(i-1)j}$ by 3 bits, i.e., q_{20} and q_{21} are displaced circularly to the left of

q_{10} and q_{11} by 3 bits, respectively and q_{10} and q_{11} are in turn displaced to the left of q_{00} and q_{01} by 3 bits, respectively. The last partial product in Fig. 3.6 is CC . As the generation of CC involves merely the assignment of logic constants to appropriate bit positions, it can be directly hardwired as a constant partial product without any logic circuitry. For the example considered, CC is derived from Table 3.2.

	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
•						d_2	d_1	d_0
	pp_{07}	pp_{06}	pp_{05}	pp_{04}	pp_{03}	pp_{02}	pp_{01}	pp_{00}
			q_{01}				q_{00}	
	pp_{17}	pp_{16}	pp_{15}	pp_{14}	pp_{13}	pp_{12}	pp_{11}	pp_{10}
				q_{10}				q_{11}
	pp_{27}	pp_{26}	pp_{25}	pp_{24}	pp_{23}	pp_{22}	pp_{21}	pp_{20}
	q_{20}				q_{21}			
	0	0	1	0	0	0	1	0

Fig. 3.6 Modulo-reduced partial products and CC in partially-redundant biased representation for $|X \cdot Y|_{2^s-1}$

The generation of the modulo-reduced partial products, PP_0 , PP_1 and PP_2 , in a partially-redundant biased representation using Booth Encoder (BE) and Booth Selector (BS) blocks is illustrated in Fig. 3.7. The BE block produces a signed digit represented by a sign bit s_i and one-hot encoded magnitude bits denoted by $m1_i$, $m2_i$, $m3_i$ and $m4_i$ from four adjacent overlapping multiplier bits, $y_{3i+2}y_{3i+1}y_{3i}y_{3i-1}$ as illustrated in Fig. 3.8(a). The one-hot encoded magnitude bits are then used to select either the multiplicand, one-bit shift multiplicand, sum of multiplicand and one-bit shift multiplicand (hard multiple) or two-bit shift multiplicand. The sign bit determines if the selected word needs to be complemented. A bit-slice of the radix-8 BS for the partial product bit pp_{ij} is shown in Fig. 3.8(b). To generate a single PP_i in the partially-redundant biased representation, $n+n/k = 10$ BS blocks are required.

Fig. 3.7 MPPG for radix-8 Booth encoded modulo 2^n-1 multiplier

Fig. 3.8 (a) Bit-slice of radix-8 Booth Encoder (BE) (b) Bit-slice of radix-8 Booth Selector (BS)

3.3.4 Accumulation of partial products

From Fig. 3.6, as the bit positions of q_{ij} do not overlap, they can be merged into a single partial product for accumulation. As M partial-carry bits (q_{ij}) are introduced by each partially-redundant partial product, the number of additional partial products resulting from merging the non-overlapping q_{ij} bits is given by Q .

$$Q = \left\lceil \frac{(\lfloor n/3 \rfloor + 1) \cdot M}{n} \right\rceil$$

$$= \left\lceil \frac{(\lfloor n/3 \rfloor + 1)}{k} \right\rceil \quad (3.38)$$

For $n = 8$ and $k = 4$, the CSA tree based accumulation of $\lfloor n/3 \rfloor + 1$ modulo-reduced partial products PP_i , Q partial products made up of merged q_{ij} bits and CC in a $(\lfloor n/3 \rfloor + 2 + Q)$ -operand, *i.e.*, five-operand (MOMA, $2^n - 1$), is illustrated in Fig. 3.9. As detailed in Section 2.4.2.5, the five-operand (MOMA, $2^n - 1$) consists of three 8-bit EAC-CSAs followed by a two-operand modulo $2^n - 1$ adder.

Fig. 3.9 MPPA for radix-8 Booth encoded modulo $2^n - 1$ multiplier

3.4 Performance analysis

In this section, the performance of the proposed family of modulo $2^n - 1$ multipliers with different suitably chosen RCA word-length k is evaluated. For experimental analysis, k is selected as n , $n/2$ and $n/4$ to satisfy *Criteria 1* and *2* when n is not divisible by three. When n is divisible by three but not by higher powers of three, k is selected as $n/3$ and $n/6$. The proposed multipliers are also compared against the recent radix-4 Booth encoded modulo $2^n - 1$ multiplier [Efst04a].

The proposed modulo $2^n - 1$ multiplier designs for various feasible combinations of n and k were specified in VHDL, synthesized using Synopsys Design Compiler (V2004.06-SP2) and

mapped to TSMC 0.18 μm 1.8V CMOS standard-cell library. The designs were synthesized under nominal synthesis design environment, *i.e.*, 25°C and 1.8V initially before the timing constraint from the RNS is imposed. The area and delay synthesis results are shown in Table 3.4 for n that is not divisible by three and in Table 3.5 for n that is divisible by three. As the dynamic power dissipation of a combinational circuit is dependent on the input pattern, a Monte Carlo simulation method [Burc93], [Satz06] using a finite number of randomly generated test patterns is adopted to estimate the average power dissipation with 99.9% confidence that the error is bounded below 3% for a data rate of 20 Msamples/s. The average dynamic power and the leakage power are listed in Tables 3.6 and 3.7 for n not divisible and divisible by three, respectively. The synthesis results in Tables 3.4 – 3.7 show that, without exerting specific timing optimization effort by the tool, k has an influence on the area and power dissipation of the proposed multiplier and its value can be adapted according to the timing surplus. Among the proposed family of modulo 2^n-1 multipliers, the multipliers with k equal to n and $n/3$ have the least area and power dissipation for n that is not divisible by three and for n that is divisible by three, respectively. This implies that for a high-DR RNS with a system delay in excess of the modulo 2^n-1 multiplier delay, the multiplier can be designed with the maximum feasible value of k to substantially reduce the area and power dissipation.

Table 3.4 Synthesis results when n is not divisible by three

n	k					
	n		$n/2$		$n/4$	
	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)
16	16026	6.99	16602	5.63	18208	5.18
20	23477	8.34	24136	6.20	26192	6.13
28	44703	10.50	45624	7.77	48555	6.72
32	56691	11.59	57909	8.38	60730	7.69
40	87730	13.62	89134	10.05	92656	8.23
44	103391	14.81	104851	10.45	109232	8.56
52	144422	16.76	146018	12.29	151148	9.13
56	168768	21.53	172051	13.43	178621	10.63
64	221082	20.15	224821	17.30	231550	10.99

Table 3.5 Synthesis results when n is divisible by three

n	k			
	$n/3$		$n/6$	
	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)
12	11459	4.90	12883	4.87
24	37302	6.45	40209	6.33
48	130800	9.78	137410	8.62
60	207194	12.57	212277	9.24

Table 3.6 Dynamic and leakage power dissipations when n is not divisible by three

n	k					
	n		$n/2$		$n/4$	
	Dynamic (mW)	Leakage (μ W)	Dynamic (mW)	Leakage (μ W)	Dynamic (mW)	Leakage (μ W)
16	0.560	0.153	0.569	0.158	0.642	0.176
20	0.832	0.223	0.849	0.229	0.935	0.249
28	1.598	0.415	1.622	0.423	1.744	0.452
32	2.024	0.522	2.053	0.530	2.194	0.560
40	3.160	0.796	3.185	0.808	3.332	0.841
44	3.720	0.929	3.761	0.943	3.939	0.983
52	5.220	1.290	5.255	1.300	5.482	1.350
56	5.732	1.460	5.796	1.480	6.083	1.554
64	6.717	1.950	6.765	1.970	7.026	2.042

Table 3.7 Dynamic and leakage power dissipation when n is divisible by three

n	k			
	$n/3$		$n/6$	
	Dynamic (mW)	Leakage (μ W)	Dynamic (mW)	Leakage (μ W)
12	0.392	0.109	0.443	0.124
24	1.329	0.345	1.450	0.377
48	4.732	1.172	4.998	1.228
60	6.647	1.809	6.899	1.884

The proposed multiplier and [Efst04a] are compared in three special moduli sets, $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ [Cao03], $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ [Mola10], that possess 2^n-1 as one of its moduli. The system delays of these RNSs are set by the critical modulo $2^{2n}+1$ and modulo $2^{2n+1}-1$ multipliers, which were implemented as described in [Sous05] and [Efst04a], respectively. As the delay of the critical modulo channel of $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ is in large excess than the delay of 2^n-1 modulo channel, the proposed partially-redundant modulo 2^n-1 multiplier is designed with the maximum feasible value of k , *i.e.*, n when n is not divisible by three and $n/3$ when n is divisible by three. The synthesis results for the proposed modulo 2^n-1 multiplier and [Efst04a] constrained by the system delay are shown in Table 3.8 for moduli sets, $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$, and in Table 3.9 for $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$. The percentage saving in area-power-product is also tabulated, where the power includes both dynamic and leakage power dissipations. From Tables 3.8 and 3.9, the proposed multiplier reduces the area as well as total power dissipation simultaneously. The percentage savings in area-power-product ranges from 2% to 35% and 1% to 15 % for the range of n considered in Tables 3.8 and 3.9, respectively.

Table 3.8 Delay-constrained area and power results of modulo 2^n-1 multipliers for $\{2^n-1, 2^n, 2^{n+1}, 2^{2n+1}\}$ and $\{2^n-1, 2^{n+1}, 2^{2n}, 2^{2n+1}\}$ RNSs

n	Delay (ns)	Proposed			[Efst04a]			% Savings in Area Power Product
		Area (μm^2)	Dynamic (mW)	Leakage (μW)	Area (μm^2)	Dynamic (mW)	Leakage (μW)	
32	3.96	75585	18.59	0.47	78360	20.47	0.56	12
40	4.06	122019	29.33	0.74	122747	29.79	0.87	2
44	4.49	134150	28.90	0.82	143311	32.15	1.04	16
48	4.56	165681	36.71	1.06	174709	38.87	1.25	10
52	4.43	199507	43.15	1.21	200641	43.88	1.44	2
56	4.56	235199	48.19	1.40	232512	56.12	1.58	13
60	4.76	255061	54.09	1.61	267093	56.66	1.93	8
64	5.13	267099	51.71	1.72	304934	69.73	2.20	35

Table 3.9 Delay-constrained area and power results of modulo 2^n-1 multipliers for $\{2^n-1, 2^n, 2^{n+1}, 2^{2n+1}-1\}$ RNS

n	Delay (ns)	Proposed			[Efst04a]			% Savings in Area Power Product
		Area (μm^2)	Dynamic (mW)	Leakage (μW)	Area (μm^2)	Dynamic (mW)	Leakage (μW)	
32	3.9	74015	20.44	0.49	78669	20.5	0.57	6
40	4.11	120019	31.74	0.77	120871	31.66	0.89	1
44	4.33	139000	30.97	0.85	143743	33.00	1.03	9
48	4.34	171465	36.41	1.06	174582	38.7	1.25	7
52	4.35	196340	42.96	1.19	202461	45.00	1.46	7
56	4.39	244533	48.22	1.41	232941	56.45	1.58	10
60	4.61	256628	53.54	1.60	267868	57.48	1.96	11
64	4.83	275881	58.21	1.76	302339	62.30	2.18	15

To show that the reduction in logic complexity of the proposed design is not due to the mapping of specific cell library, the architectures of [Efst04a] and the proposed multipliers are compared theoretically using the normalized area model. In this model, the logic modules employed in both multipliers are identified and the area of each logic module is normalized to the area of the inverter for the target cell library. The normalized areas of the logic modules for TSMC 0.18 μm CMOS standard cell library are shown in Table 3.10 [TSMC03]. The normalized area expressions of [Efst04a] and the proposed multiplier are expressed in Table 3.11 and the comparison of normalized area is presented in Table 3.12. The final parallel-prefix adder is excluded from the theoretical evaluation as it is common to both designs. From Table 3.12, the proposed design reduces the area of [Efst04a] for $n > 28$ and the trend is well correlated with the actual synthesis results.

Table 3.10 Normalized area of logic modules

Logic module	Notation	Normalized area
Inverter	INV	1
2-input AND	AND2	2
3-input AND	AND3	2.5
2-input OR	OR2	2
4-input OR	OR4	3
2-input XOR	XOR2	4
2-input XNOR	XNOR2	4
Half Adder	HA	5.5
Full Adder	FA	10.5

Table 3.11 Normalized area expressions of multipliers

Multiplier	Component	Normalized-area of component	Number of component
Proposed	k -bit RCA	$k - 1A_{FA} + A_{HA} + A_{OR2} + A_{XNOR2}$ $= 10.5k + 1$	M
	BE	$3A_{INV} + 3A_{AND2} + A_{AND3} + 3A_{XOR2}$ $= 23.5$	$\lfloor n/3 \rfloor + 1$
	BS	$4A_{AND2} + A_{OR4} + A_{XOR2} = 15$	$(\lfloor n/3 \rfloor + 1) \cdot (n + M)$
	FA	$A_{FA} = 10.5$	$n(\lfloor n/3 \rfloor + Q)$
	Total	$25.5n(\lfloor n/3 \rfloor + 1) + 10.5n + 38.5(\lfloor n/3 \rfloor + 1) + 1$ if $k = n$ $25.5n(\lfloor n/3 \rfloor + 1) + 21n + 68.5(\lfloor n/3 \rfloor + 1) + 3$ if $k = n/3$	
[Efst04a]	BE	$A_{INV} + A_{AND2} + 2A_{XOR2} = 11$	$n/2$
	BS	$2A_{AND2} + A_{OR2} + A_{XOR2} = 10$	$n(n/2)$
	FA	$A_{FA} = 10.5$	$n(n/2 - 2)$
	Total	$10.25n^2 - 15.5n$	

Table 3.12 Comparison of normalized area

n	Proposed	[Efst04a]
28	7820	7602
32	9736.5	10000
40	15240	15780
44	17871	19162
48	22984	22872
52	25108	26910
56	28453	31276
60	34832	35970
64	37424	40992

3.5 Summary

The modulo $2^n - 1$ multiplier is usually the non-critical channel among all modulo channels in high-DR RNS multiplier. By exploiting this timing slack the system area and power consumption can be reduced without compromising the system performance. With this precept, a family of radix-8 Booth encoded modulo $2^n - 1$ multipliers with delay adaptable to

the RNS multiplier delay was proposed. The delay of the proposed multiplier was made scalable by controlling the word-length k of the parallel RCAs that were used in the computation of the requisite hard multiple of the radix-8 Booth encoded multiplication in a partially-redundant form.

Formal criteria for the selection of the RCA word-length were established. It was proven that for a given n , there exist a number of feasible values of k such that the total bias incurred from the partially-redundant biased partial products can be counteracted by only a single constant binary string. This compensation constant for different valid combinations of n and k can be precomputed at design time using the number theoretic properties of modulo 2^n-1 arithmetic and hardwired as a partial product. The trade-offs between the RCA word-length and the VLSI performance metrics, *i.e.*, area, delay and power dissipation of the proposed modulo 2^n-1 multiplier were analyzed by means of CMOS implementations. For maximal area and power savings, $k = n$ when n is not divisible by three and $k = n/3$ when n is divisible by three, were recommended for the RCA word-length when the RNS multiplier delay exceeded the non-critical modulo 2^n-1 multiplier delay substantially. Conversely, when the RNS multiplier and the modulo 2^n-1 multiplier delays were nearly balanced, RCA word-lengths of $n/4$ and $n/6$ were recommended when n is not divisible and divisible by three, respectively. From synthesis results constrained by the delay of the RNS multiplier, it was shown that the proposed multiplier simultaneously reduces area as well as power dissipation of the radix-4 Booth encoded multiplier in medium to large word-length multiplication.

Chapter 4

Radix-8 Booth Encoded Modulo 2^n-1 and Modulo 2^n+1 Multipliers for Balanced Word-length Moduli Set based RNS

4.1 Introduction

The challenge in high radix, *i.e.*, radix greater than four Booth encoded modulo multiplier design is the non-trivial generation of the modulo-reduced hard multiple(s). In Chapter 3, the first-ever radix-8 Booth encoded modulo 2^n-1 multiplier that dealt with the hard multiple generation problem was presented. The delay of the proposed modulo 2^n-1 multiplier was made adaptive to match the delay of the RNS multiplier closely. Nevertheless, this method of hard multiple generation is not without drawbacks. Because all partial products are consistently generated in the partially-redundant and biased form, for every n -bit partial product vector, an extra (n/k) -bit vector of partial-carry bits is ineluctably introduced. The partial-carry bits of the partially-redundant representation are merged into Q additional partial products and an n -bit compensation constant is introduced to offset the biased representation. So the total number of partial products is $\lfloor n/3 \rfloor + 2 + Q$. This is still $(Q + 1)$ more than the ideal number of partial products, *i.e.*, $\lfloor n/3 \rfloor + 1$, achievable by radix-8 Booth encoding technique. As a result, the use of delay-adaptive modulo 2^n-1 multiplier is restricted to a RNS where the modulus 2^n-1 forms the non-critical channel. Furthermore, for area-power advantages, the timing slack between the non-critical modulo 2^n-1 channel and the critical modulo m channel must be sufficiently large. Hence Chapter 3 addresses RNSs with imbalanced word-length moduli sets. In the RNS based on the balanced word-length moduli set such as $\{2^n-1, 2^n, 2^n+1\}$, there is not enough timing slack to be exploited and thus the delay-adaptive multiplier design technique is ineffective in this case.

Furthermore, it is impractical to extend the delay-adaptive technique to modulo 2^n+1 multiplier design. This is because modulo 2^n+1 negation, *CCLS* operation and modulo 2^n+1 addition entail additional correction terms. These terms, together with the $Q + 1$ additional partial products generated by the partially-redundant and biased representation, may offset the number of partial products reduced by the radix-8 Booth encoding. Thus, the delay-adaptive multiplier design technique is not comprehensive and not equally applicable to all moduli.

In this Chapter, a new radix-8 Booth encoded modulo 2^n-1 multiplier and the first radix-8 Booth encoded modulo 2^n+1 multiplier are proposed. Unlike Chapter 3, no restriction is imposed on the moduli set, *i.e.*, the proposed multipliers are equally applicable in critical and non-critical moduli channels as well as balanced and imbalanced word-length moduli sets. A new solution to the hard multiple generation is suggested. Customized adders that compute only the modulo-reduced hard multiples of radix-8 Booth encoding algorithm, *i.e.*, $|+3X|_{2^n-1}$ and $|+3X|_{2^n+1}$, are introduced. The proposed adders are by far the most area-delay-power efficient customized adders for this application.

This chapter is organized as follows. In Section 4.2, the radix-8 Booth encoded multiplication algorithms for modulo 2^n-1 and modulo 2^n+1 multiplications are described. The expressions of modulo 2^n-1 and modulo 2^n+1 reduced partial products are derived and simplified. Section 4.3 describes the proposed solution to modulo 2^n-1 and modulo 2^n+1 hard multiple generations. The proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers are described in Section 4.4. In Section 4.5, the performances of the proposed multipliers are evaluated and compared against radix-4 Booth encoded and non-encoded modulo multipliers. A brief summary is presented in Section 4.6.

4.2 Radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multiplication algorithms

This section presents the preliminaries of radix-8 Booth encoded modulo multiplication using binary representation with dual zeros for modulo 2^n-1 arithmetic and diminished-1

representation for modulo 2^n+1 arithmetic. All equations are assumed to be modulo-reduced by m , $m \in \{2^n-1, 2^n+1\}$ unless otherwise specified. Let $X = \sum_{i=0}^{n-1} x_i 2^i$, $Y = \sum_{i=0}^{n-1} y_i 2^i$ and $Z = \sum_{i=0}^{n-1} z_i 2^i$ be the n -bit multiplicand, multiplier and product, respectively. Then, the modulo 2^n-1 and modulo 2^n+1 multiplications can be expressed as

$$|Z|_m = \begin{cases} |X \cdot Y|_m = \left| \sum_{i=0}^{n-1} X \cdot y_i \cdot 2^i \right|_m & \text{if } m = 2^n - 1 \\ |X \cdot Y + X + Y|_m = \left| \sum_{i=0}^{n-1} X \cdot y_i \cdot 2^i + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (4.1)$$

Using the radix-8 Booth encoding, the n multiplier bits $y_i \in [0, 1]$ are encoded into $\lfloor n/3 \rfloor + 1$ signed multiplier digits $d_i \in [-4, 4]$ as explained in Chapter 3. Hence (4.1) is simplified to

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} X \cdot d_i \cdot 2^{3i} \right|_m & \text{if } m = 2^n - 1 \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} X \cdot d_i \cdot 2^{3i} + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (4.2)$$

In addition to *Properties 3.1 – 3.3* of modulo 2^n-1 arithmetic, the following three properties of modulo 2^n+1 arithmetic are used to simplify the modulo 2^n-1 and modulo 2^n+1 multiplier designs.

Property 4.1: As a corollary of *Property 2.2*, the modulo reduction of the binary weight of x in excess of the modulus is given by

$$\begin{aligned} |x \cdot 2^{n+i}|_{2^n+1} &= |-x \cdot 2^i|_{2^n+1} \\ &= \bar{x} \cdot 2^i - 2^i \end{aligned} \quad (4.3)$$

where $\bar{x}$ is the bit complement of x .

Property 4.2: Let $CCLS(X, j)$ denote the circular left shift of X by j bit positions, where the j lsbs are inverted. Using *Property 4.1* to modulo-reduce the binary weights in excess of the modulus, the modulo 2^n+1 multiplication of X by a positive power-of-two term 2^j is given by

$$\begin{aligned}
|2^j \cdot X|_{2^n+1} &= \left| \sum_{i=0}^{n-1} x_i 2^{i+j} \right|_{2^n+1} \\
&= \sum_{i=0}^{n-1-j} x_i 2^{i+j} + \sum_{i=n-j}^{n-1} \bar{x}_i 2^{i+j-n} - 2^{i+j-n} \\
&= CCLS(X, j) - 2^j + 1
\end{aligned} \tag{4.4}$$

Property 4.3: By substituting for the additive inverse $-X$ as $\bar{X} + 2$ from *Property 2.4*, the modulo 2^n+1 multiplication of X by -2^j is given by

$$\begin{aligned}
|-2^j \cdot X|_{2^n+1} &= |2^j (\bar{X} + 2)|_{2^n+1} \\
&= CCLS(\bar{X}, j) + 2^j + 1
\end{aligned} \tag{4.5}$$

Properties 4.2 and *4.3* imply that the modulo 2^n+1 multiplications by 2^j and -2^j can be realized by *CCLS* operations on X with a bias of $-2^j + 1$ and on $\bar{X}$ with a bias of $2^j + 1$, respectively.

The generation of $|X \cdot d_i \cdot 2^{3i}|_m$ can be described by a *CLS* vector PP_i using *Properties 3.2 – 3.3* for $m = 2^n - 1$ as shown in Table 4.1 and by the sum of a *CCLS* vector PP_i and a predefined bias K_i using *Properties 4.2 – 4.3* for $m = 2^n + 1$ as shown in Table 4.2. Hence (4.2) can be further simplified to

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i \right|_m & \text{if } m = 2^n - 1 \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} (PP_i + K_i) + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \tag{4.6}$$

In Tables 4.1 and 4.2, the modulo-reduced partial products $|d_i X \cdot 2^{3i}|_m$, for $d_i \in \{\pm 0, \pm 1, \pm 2, \pm 4\}$ can be implemented by selective bitwise shift and selective bitwise inversion. On the other hand, the generation of the hard multiple $|+3X|_m$ and the corresponding modulo-reduced partial product $|+3X \cdot 2^{3i}|_m$ requires a two-operand modulo m adder to compute the sum of X and $|2X|_m$. To minimize the hardware overhead, it is proposed to replace the two-operand modulo adders by new customized adders called Hard Multiple Generators (HMGs) that

exclusively generate the hard multiples of radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers.

Table 4.1 Modulo 2^n-1 reduced partial products for radix-8 Booth encoding

d_i	$ X \cdot d_i \cdot 2^{3i} _{2^n-1} = PP_i$	d_i	$ X \cdot d_i \cdot 2^{3i} _{2^n-1} = PP_i$
+0	$CLS(\underbrace{0 \cdots 0}_n, 3i)$	-0	$CLS(\underbrace{1 \cdots 1}_n, 3i)$
+1	$CLS(X, 3i)$	-1	$CLS(\bar{X}, 3i)$
+2	$CLS(X, 3i+1)$	-2	$CLS(\bar{X}, 3i+1)$
+3	$CLS(+3X _{2^n-1}, 3i)$	-3	$CLS(+3X _{2^n-1}, 3i)$
+4	$CLS(X, 3i+2)$	-4	$CLS(\bar{X}, 3i+2)$

Table 4.2 Modulo 2^n+1 reduced partial products for radix-8 Booth encoding

d_i	$ X \cdot d_i \cdot 2^{3i} _{2^n+1} = PP_i + K_i$		d_i	$ X \cdot d_i \cdot 2^{3i} _{2^n+1} = PP_i + K_i$	
	PP_i	K_i		PP_i	K_i
+0	$CCLS(\underbrace{0 \cdots 0}_n, 3i)$	-2^{3i+1}	-0	$CCLS(\underbrace{1 \cdots 1}_n, 3i)$	2^{3i+1}
+1	$CCLS(X, 3i)$	-2^{3i+1}	-1	$CCLS(\bar{X}, 3i)$	2^{3i+1}
+2	$CCLS(X, 3i+1)$	-2^{3i+1+1}	-2	$CCLS(\bar{X}, 3i+1)$	2^{3i+1+1}
+3	$CCLS(+3X _{2^n+1}, 3i)$	-2^{3i+1}	-3	$CCLS(+3X _{2^n+1}, 3i)$	2^{3i+1}
+4	$CCLS(X, 3i+2)$	-2^{3i+2+1}	-4	$CCLS(\bar{X}, 3i+2)$	2^{3i+2+1}

4.3 Proposed modulo 2^n-1 and modulo 2^n+1 Hard Multiple Generators (HMGs)

In this section, the carry equations of modulo 2^n-1 and modulo 2^n+1 additions are reformulated by considering X and $2X$ as the addends. The proposed HMGs are detailed for $n = 8$ for illustration but the derivation is applicable to any n .

4.3.1 Modulo 2^n-1 HMG

In Chapter 2, the carry for modulo 2^n-1 addition with unrolled carry-out bit was shown to be a function of the generate and propagate signals at not only bit positions 0 through i but also bit positions $i+1$ through $n-1$ [Kala00]. Mathematically, for $i = 0$ to $n-1$,

$$c_i = (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \cdots \bullet (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet (g_{n-2}, p_{n-2}) \bullet \cdots \bullet (g_{i+1}, p_{i+1}) \quad (4.7)$$

Let $X = \sum_{i=0}^7 x_i 2^i$ be the 8-bit multiplicand. The hard multiple $|+3X|_{2^8-1}$ is generated by adding X and $|2X|_{2^8-1}$. By *Property 3.2*, modulo 2^n-1 multiplication by a positive power-of-two term is equivalent to a circular-left-shift operation. Hence the addend $|2X|_{2^8-1}$ is simplified to

$$|2X|_{2^8-1} = CLS(X, 1) \quad (4.8)$$

For the addends X and $CLS(X, 1)$ the generate g_i and propagate p_i bits can be defined as

$$(g_0, p_0) = (x_0 \cdot x_7, x_0 + x_7); \dots; (g_7, p_7) = (x_7 \cdot x_6, x_7 + x_6) \quad (4.9)$$

The bit correlation between the addends is exploited to simplify the logical conjunction of adjacent g_i and p_i bits as given by *Property 4.4*.

Property 4.4: For the addition of X and $CLS(X, 1)$,

$$p_i \cdot g_{i-1} = \begin{cases} (x_0 + x_7) \cdot (x_7 \cdot x_6) = x_7 \cdot x_6 = g_7 & i = 0 \\ (x_1 + x_0) \cdot (x_0 \cdot x_7) = x_0 \cdot x_7 = g_0 & i = 1 \\ (x_i + x_{i-1}) \cdot (x_{i-1} \cdot x_{i-2}) = x_{i-1} \cdot x_{i-2} = g_{i-1} & 2 \leq i \leq 7 \end{cases} \quad (4.10)$$

c_0 can be calculated using (4.7) as

$$\begin{aligned} c_0 &= (g_0, p_0) \cdot (g_7, p_7) \cdot (g_6, p_6) \cdot (g_5, p_5) \cdot (g_4, p_4) \cdot (g_3, p_3) \cdot (g_2, p_2) \cdot (g_1, p_1) \\ &= g_0 + p_0 g_7 + p_0 p_7 g_6 + p_0 p_7 p_6 g_5 + p_0 p_7 p_6 p_5 g_4 + p_0 p_7 p_6 p_5 p_4 g_3 \\ &\quad + p_0 p_7 p_6 p_5 p_4 p_3 g_2 + p_0 p_7 p_6 p_5 p_4 p_3 p_2 g_1 \end{aligned} \quad (4.11)$$

By replacing, $p_0 g_7$ and $p_i g_{i-1}$ by g_7 and g_{i-1} , respectively using *Property 4.4*, (4.11) can be simplified for the computation of the hard multiple as follows.

$$\begin{aligned} c_0 &= (g_0 + g_7) + p_0 p_7 (g_6 + g_5) + p_0 p_7 p_6 p_5 (g_4 + g_3) + p_0 p_7 p_6 p_5 p_4 p_3 (g_2 + g_1) \\ &= g_0^* + p_0^* g_6^* + p_0^* p_6^* g_4^* + p_0^* p_6^* p_4^* g_2^* \\ &= (g_0^*, p_0^*) \cdot (g_6^*, p_6^*) \cdot (g_4^*, p_4^*) \cdot (g_2^*, p_2^*) \end{aligned} \quad (4.12)$$

where the modified-generate g_i^* and modified-propagate p_i^* bits are defined as

$$\begin{aligned}
g_i^* &= \begin{cases} g_0 + g_7 = x_7 \cdot (x_0 + x_6) & i=0 \\ g_1 + g_0 = x_0 \cdot (x_1 + x_7) & i=1 \\ g_i + g_{i-1} = x_{i-1} \cdot (x_i + x_{i-2}) & 2 \leq i \leq 7 \end{cases} \\
p_i^* &= \begin{cases} p_0 \cdot p_7 = x_7 + x_0 \cdot x_6 & i=0 \\ p_1 \cdot p_0 = x_0 + x_1 \cdot x_7 & i=1 \\ p_i \cdot p_{i-1} = x_{i-1} + x_i \cdot x_{i-2} & 2 \leq i \leq 7 \end{cases}
\end{aligned} \tag{4.13}$$

Hence the carry equations of the modulo 2^8-1 HMG using modified-generate and modified-propagate bits are given by

$$\begin{aligned}
c_0 &= (g_0^*, p_0^*) \bullet (g_6^*, p_6^*) \bullet (g_4^*, p_4^*) \bullet (g_2^*, p_2^*) \\
c_1 &= (g_1^*, p_1^*) \bullet (g_7^*, p_7^*) \bullet (g_5^*, p_5^*) \bullet (g_3^*, p_3^*) \\
c_2 &= (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \bullet (g_6^*, p_6^*) \bullet (g_4^*, p_4^*) \\
c_3 &= (g_3^*, p_3^*) \bullet (g_1^*, p_1^*) \bullet (g_7^*, p_7^*) \bullet (g_5^*, p_5^*) \\
c_4 &= (g_4^*, p_4^*) \bullet (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \bullet (g_6^*, p_6^*) \\
c_5 &= (g_5^*, p_5^*) \bullet (g_3^*, p_3^*) \bullet (g_1^*, p_1^*) \bullet (g_7^*, p_7^*) \\
c_6 &= (g_6^*, p_6^*) \bullet (g_4^*, p_4^*) \bullet (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \\
c_7 &= (g_7^*, p_7^*) \bullet (g_5^*, p_5^*) \bullet (g_3^*, p_3^*) \bullet (g_1^*, p_1^*)
\end{aligned} \tag{4.14}$$

Using only $n/2$ modified-generate and modified-propagate bit pairs (g_i^*, p_i^*) , the carry equations of modulo 2^n-1 HMG for any n are given by

$$c_i = \begin{cases} (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \bullet \cdots \bullet (g_0^*, p_0^*) \bullet (g_{n-2}^*, p_{n-2}^*) \bullet \cdots \bullet (g_{i+2}^*, p_{i+2}^*) & \text{for even } i \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \bullet \cdots \bullet (g_1^*, p_1^*) \bullet (g_{n-1}^*, p_{n-1}^*) \bullet \cdots \bullet (g_{i+2}^*, p_{i+2}^*) & \text{for odd } i \end{cases} \tag{4.15}$$

where

$$\begin{aligned}
g_i^* &= \begin{cases} g_0 + g_{n-1} = x_{n-1} \cdot (x_0 + x_{n-2}) & i=0 \\ g_1 + g_0 = x_0 \cdot (x_1 + x_{n-1}) & i=1 \\ g_i + g_{i-1} = x_{i-1} \cdot (x_i + x_{i-2}) & 2 \leq i \leq n-1 \end{cases} \\
p_i^* &= \begin{cases} p_0 \cdot p_{n-1} = x_{n-1} + x_0 \cdot x_{n-2} & i=0 \\ p_1 \cdot p_0 = x_0 + x_1 \cdot x_{n-1} & i=1 \\ p_i \cdot p_{i-1} = x_{i-1} + x_i \cdot x_{i-2} & 2 \leq i \leq n-1 \end{cases}
\end{aligned} \tag{4.16}$$

Fig. 4.1 shows the proposed modulo 2^n-1 HMG for $n = 8$. Each operator ‘■’ in the pre-

processing stage contains an OR-AND gate, an AND-OR gate and a XOR gate for the generation of g_i^* , p_i^* and h_i , respectively. In the prefix computation stage, the carry c_i is computed from $n/2 = 4$ (g_i^*, p_i^*) pairs in $\lceil \log_2 n \rceil - 1 = 2$ levels of prefix operators, ‘ $\bullet$ ’. Finally, each operator ‘ $\diamond$ ’ in the post-processing stage computes a sum bit s_i by the XOR of h_i from the pre-processing stage and c_{i-1} from the prefix-computation stage.

Fig. 4.1 Modulo 2^n-1 Hard Multiple Generator

4.3.2 Modulo 2^n+1 HMG

In Chapter 2, diminished-1 modulo 2^n+1 addition was expressed as

$$|S|_{2^n+1} = |X + Y + 1|_{2^n+1} \quad (4.17)$$

In other words, the result of a diminished-1 modulo 2^n+1 addition is the sum of the addends as well as a constant one. The carry equation for diminished-1 modulo 2^n+1 addition with unrolled carry-out was shown to be

$$c_i = (g_i, p_i) \bullet \dots \bullet (g_0, p_0) \bullet \overline{(g_{n-1}, p_{n-1}) \bullet \dots \bullet (g_{i+1}, p_{i+1})} \quad (4.18)$$

Let $X = \sum_{i=0}^7 x_i 2^i$ be the 8-bit multiplicand. By *Property 4.2*,

$$\lfloor 2 \cdot X \rfloor_{2^n+1} = CCLS(X, 1) - 1 \quad (4.19)$$

To simplify the computation of the hard multiple $\lfloor +3X \rfloor_{2^n+1}$ by exploiting the bit correlation between the addends, the addends are expressed as X and $CCLS(X, 1) = \lfloor 2X \rfloor_{2^n+1} + 1$. From (4.17), the diminished-1 modulo 2^n+1 addition of these addends is

$$\lfloor X + CCLS(X, 1) + 1 \rfloor_{2^n+1} = \lfloor X + 2 \cdot X + 1 + 1 \rfloor_{2^n+1} = \lfloor 3 \cdot X + 2 \rfloor_{2^n+1} \quad (4.20)$$

Hence in the proposed HMG, by considering the addends as X and $CCLS(X, 1)$, the hard multiple is generated along with a constant bias of two. Instead of removing this constant bias from the sum, which will increase the critical path delay of the HMG, it is negated in the partial product accumulation stage as detailed in the following section.

For the addends X and $CCLS(X, 1)$ the generate g_i and propagate p_i bits can be defined as

$$(g_0, p_0) = (x_0 \cdot \bar{x}_7, x_0 + \bar{x}_7); \dots; (g_7, p_7) = (x_7 \cdot x_6, x_7 + x_6) \quad (4.21)$$

Property 4.5: For the addition of X and $CCLS(X, 1)$,

$$p_0 \cdot \bar{p}_7 = (x_0 + \bar{x}_7) \cdot \overline{(x_7 + x_6)} = \bar{p}_7 \quad (4.22)$$

$$p_i \cdot g_{i-1} = \begin{cases} (x_1 + x_0) \cdot (x_0 \cdot \bar{x}_7) = x_0 \cdot \bar{x}_7 = g_0 & i=1 \\ (x_i + x_{i-1}) \cdot (x_{i-1} \cdot x_{i-2}) = x_{i-1} \cdot x_{i-2} = g_{i-1} & 2 \leq i \leq 7 \end{cases} \quad (4.23)$$

$$\bar{g}_i \cdot \bar{p}_{i-1} = \begin{cases} \overline{(x_1 \cdot x_0)} \cdot \overline{(x_0 + \bar{x}_7)} = \overline{x_0 + \bar{x}_7} = \bar{p}_0 & i=1 \\ \overline{(x_i \cdot x_{i-1})} \cdot \overline{(x_{i-1} + x_{i-2})} = \overline{x_{i-1} + x_{i-2}} = \bar{p}_{i-1} & 2 \leq i \leq 7 \end{cases} \quad (4.24)$$

From (4.18), c_0 can be written as

$$\begin{aligned} c_0 &= (g_0, p_0) \bullet \overline{(g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1)} \\ &= g_0 + p_0 \bar{p}_7 + p_0 \bar{g}_7 \bar{p}_6 + p_0 \bar{g}_7 \bar{g}_6 \bar{p}_5 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{p}_4 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{p}_3 \\ &\quad + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{p}_2 + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 \bar{g}_2 \bar{p}_1 \end{aligned} \quad (4.25)$$

Using *Property 4.5*, (4.25) is simplified for the computation of the hard multiple as follows.

$$\begin{aligned}
c_0 &= (g_0 + \bar{p}_7) + p_0 \bar{g}_7 (\bar{p}_6 + \bar{p}_5) + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 (\bar{p}_4 + \bar{p}_3) + p_0 \bar{g}_7 \bar{g}_6 \bar{g}_5 \bar{g}_4 \bar{g}_3 (\bar{p}_2 + \bar{p}_1) \\
&= g_0^* + p_0^* \bar{p}_6^* + p_0^* \bar{g}_6^* \bar{p}_4^* + p_0^* \bar{g}_6^* \bar{g}_4^* \bar{p}_2^* \\
&= (g_0^*, p_0^*) \bullet (\bar{p}_6^*, \bar{g}_6^*) \bullet (\bar{p}_4^*, \bar{g}_4^*) \bullet (\bar{p}_2^*, \bar{g}_2^*)
\end{aligned} \tag{4.26}$$

where the modified-generate g_i^* and modified-propagate p_i^* bits are defined as

$$\begin{aligned}
g_i^* &= \begin{cases} g_0 + \bar{p}_7 = \bar{x}_7 \cdot (x_0 + \bar{x}_6) & i=0 \\ g_1 + g_0 = x_0 \cdot (x_1 + \bar{x}_7) & i=1 \\ g_i + g_{i-1} = x_{i-1} \cdot (x_i + x_{i-2}) & 2 \leq i \leq 7 \end{cases} \\
p_i^* &= \begin{cases} p_0 \cdot \bar{g}_7 = \bar{x}_7 + x_0 \cdot \bar{x}_6 & i=0 \\ p_1 \cdot p_0 = x_0 + x_1 \cdot \bar{x}_7 & i=1 \\ p_i \cdot p_{i-1} = x_{i-1} + x_i \cdot x_{i-2} & 2 \leq i \leq 7 \end{cases}
\end{aligned} \tag{4.27}$$

The carry equation for the modulo 2^8+1 HMG using the modified-generate and modified-propagate bits is summarized as

$$\begin{aligned}
c_0 &= (g_0^*, p_0^*) \bullet (\bar{p}_6^*, \bar{g}_6^*) \bullet (\bar{p}_4^*, \bar{g}_4^*) \bullet (\bar{p}_2^*, \bar{g}_2^*) \\
c_1 &= (g_1^*, p_1^*) \bullet (\bar{p}_7^*, \bar{g}_7^*) \bullet (\bar{p}_5^*, \bar{g}_5^*) \bullet (\bar{p}_3^*, \bar{g}_3^*) \\
c_2 &= (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \bullet (\bar{p}_6^*, \bar{g}_6^*) \bullet (\bar{p}_4^*, \bar{g}_4^*) \\
c_3 &= (g_3^*, p_3^*) \bullet (g_1^*, p_1^*) \bullet (\bar{p}_7^*, \bar{g}_7^*) \bullet (\bar{p}_5^*, \bar{g}_5^*) \\
c_4 &= (g_4^*, p_4^*) \bullet (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \bullet (\bar{p}_6^*, \bar{g}_6^*) \\
c_5 &= (g_5^*, p_5^*) \bullet (g_3^*, p_3^*) \bullet (g_1^*, p_1^*) \bullet (\bar{p}_7^*, \bar{g}_7^*) \\
c_6 &= (g_6^*, p_6^*) \bullet (g_4^*, p_4^*) \bullet (g_2^*, p_2^*) \bullet (g_0^*, p_0^*) \\
c_7 &= (g_7^*, p_7^*) \bullet (g_5^*, p_5^*) \bullet (g_3^*, p_3^*) \bullet (g_1^*, p_1^*)
\end{aligned} \tag{4.28}$$

Using only $n/2$ modified-generate and modified-propagate bit pairs (g_i^*, p_i^*) , the carry equations of modulo 2^n+1 HMG for any n are given by

$$c_i = \begin{cases} (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \bullet \cdots \bullet (g_0^*, p_0^*) \bullet (\bar{p}_{n-2}^*, \bar{g}_{n-2}^*) \bullet \cdots \bullet (\bar{p}_{i+2}^*, \bar{g}_{i+2}^*) & \text{for even } i \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \bullet \cdots \bullet (g_1^*, p_1^*) \bullet (\bar{p}_{n-1}^*, \bar{g}_{n-1}^*) \bullet \cdots \bullet (\bar{p}_{i+2}^*, \bar{g}_{i+2}^*) & \text{for odd } i \end{cases} \tag{4.29}$$

where

$$\begin{aligned}
g_i^* &= \begin{cases} g_0 + \bar{p}_{n-1} = \bar{x}_{n-1} \cdot (x_0 + \bar{x}_{n-2}) & i = 0 \\ g_1 + g_0 = x_0 \cdot (x_1 + \bar{x}_{n-1}) & i = 1 \\ g_i + g_{i-1} = x_{i-1} \cdot (x_i + x_{i-2}) & 2 \leq i \leq n-1 \end{cases} \\
p_i^* &= \begin{cases} p_0 \cdot \bar{g}_{n-1} = \bar{x}_{n-1} + x_0 \cdot \bar{x}_{n-2} & i = 0 \\ p_1 \cdot p_0 = x_0 + x_1 \cdot \bar{x}_{n-1} & i = 1 \\ p_i \cdot p_{i-1} = x_{i-1} + x_i \cdot x_{i-2} & 2 \leq i \leq n-1. \end{cases}
\end{aligned} \tag{4.30}$$

To implement (4.28), a modified prefix operator that computes $(g_i^*, p_i^*) \bullet (g_j^*, p_j^*)$ as well as $(\bar{p}_i^*, \bar{g}_i^*) \bullet (\bar{p}_j^*, \bar{g}_j^*)$ from the inputs (g_i^*, p_i^*) and (g_j^*, p_j^*) , using fewer logic gates is proposed. This implementation makes best use of the AND-OR-INVERTER (AOI) gates commonly available in CMOS standard-cell libraries.

For (g_i^*, p_i^*) defined in (4.27), it can be easily verified that

$$\begin{aligned}
g_i^* \cdot p_i^* &= g_i^* \\
\bar{p}_i^* \cdot \bar{g}_i^* &= \bar{p}_i^*
\end{aligned} \tag{4.31}$$

Using (4.31), the output of the modified prefix operator in the first prefix level, $l = 1$ can be computed as follows.

$$\begin{aligned}
(g_i^*, p_i^*) \bullet (g_j^*, p_j^*) &= (g_i^* + p_i^* \cdot g_j^*, p_i^* \cdot p_j^*) \\
&= (\bar{g}_i^* \cdot \bar{g}_j^* \cdot p_i^*, p_i^* \cdot p_j^*) \\
(\bar{p}_i^*, \bar{g}_i^*) \bullet (\bar{p}_j^*, \bar{g}_j^*) &= (\bar{p}_i^* + \bar{g}_i^* \cdot \bar{p}_j^*, \bar{g}_i^* \cdot \bar{g}_j^*) \\
&= (\bar{p}_i^* \cdot p_j^* \cdot \bar{g}_i^*, \bar{g}_i^* \cdot \bar{g}_j^*)
\end{aligned} \tag{4.32}$$

The implementation of the modified prefix operator is shown in Fig. 4.2. Fig. 4.3 shows the proposed modulo 2^n+1 HMG for $n = 8$. The dotted line indicates that the modified-generate and modified-propagate signals have been pair-wise swapped and complemented from the outputs of the preprocessing stage to produce $(\bar{p}_i^*, \bar{g}_i^*)$. The computations in the pre-processing ‘■’, prefix ‘●’ and post-processing ‘◇’ operators are identical to those in Fig. 4.1. All but four operators in the first level of the prefix computation stage are simplified to the modified prefix operator of Fig. 4.2. In Fig. 4.3, the carry c_i is computed in $\lceil \log_2 n \rceil - 1 = 2$ levels of prefix operators.

Fig. 4.2 Modified prefix operator

Fig. 4.3 Modulo 2^n+1 Hard Multiple Generator

4.4 Proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers

At the outset, the simplified expressions of partial products for radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers were tabulated in Tables 4.1 and 4.2, respectively. Subsequently, novel designs for generating the hard multiples were proposed. This section completes the architecture of the proposed multipliers with circuits for generating and accumulating the modulo-reduced partial products.

4.4.1 Modulo 2^n-1 multiplier

4.4.1.1 Generation of partial products

For $n = 8$, the generation of the $\lfloor n/3 \rfloor + 1 = 3$ modulo 2^n-1 reduced partial products, PP_0 , PP_1 and PP_2 , according to Table 4.1 is shown in Fig. 4.4. The number of radix-8 Booth Encoder (BE) blocks required to encode n multiplier bits into multiplier digits is $\lfloor n/3 \rfloor + 1$. To generate a single PP_i , n Booth Selector (BS) blocks are needed. Then the total number of BS blocks required for the generation of $\lfloor n/3 \rfloor + 1$ PP_i s is $n(\lfloor n/3 \rfloor + 1)$. Figs 4.5(a) and (b) show the bit slice implementations of the BE and BS blocks, respectively. In Fig. 4.5(b), h_j represents the j -th bit of the hard multiple generated according to the proposed modulo 2^n-1 HMG depicted in Fig. 4.1.

Fig. 4.4 MPPG for radix-8 Booth encoded modulo 2^n-1 multiplier

Fig. 4.5 (a) Bit-slice of radix-8 Booth Encoder (BE) (b) Bit-slice of radix-8 Booth Selector (BS)

4.4.1.2 Accumulation of partial products

From (4.6), the total number of modulo-reduced partial products to be accumulated is $\lfloor n/3 \rfloor + 1$. The CSA tree based accumulation of the partial products in a $(\lfloor n/3 \rfloor + 1)$ -operand (MOMA, $2^n - 1$) is illustrated in Fig. 4.6 for $n = 8$. The $(\lfloor n/3 \rfloor + 1)$ -operand (MOMA, $2^n - 1$) is designed with $\lfloor n/3 \rfloor - 1$ EAC-CSAs followed by a two-operand modulo $2^n - 1$ adder.

Fig. 4.6 MPPA for radix-8 Booth encoded modulo $2^n - 1$ multiplier

4.4.2 Modulo 2^n+1 multiplier

4.4.2.1 Generation of partial products

The generation of modulo 2^n+1 reduced partial products, PP_0 , PP_1 and PP_2 , according to Table 4.2, is illustrated in Fig. 4.7 for $n = 8$. Again, the number of BE blocks required to encode the n multiplier bits is $\lfloor n/3 \rfloor + 1$ and the total number of BS blocks needed to generate $\lfloor n/3 \rfloor + 1$ PP_i s is $n(\lfloor n/3 \rfloor + 1)$. In Fig. 4.7, the inputs x_{j-3i-1} and x_{j-3i-2} when $j = 3i$, the input x_{j-3i-2} when $j = 3i+1$ and the outputs at the least-significant $3i$ bit positions of the BS blocks are inverted to implement the *CCLS* operation. Furthermore, the input h_j to the BS block represents the j -th bit of the hard multiple generated according to the proposed modulo 2^n+1 HMG shown in Fig. 4.3.

Fig. 4.7 MPPG for radix-8 Booth encoded modulo 2^n+1 multiplier

4.4.2.2 Generation of bias K_i

From (4.6), for radix-8 Booth encoded modulo 2^n+1 multiplier, besides the $\lfloor n/3 \rfloor + 1$ PP_i , additional $\lfloor n/3 \rfloor + 1$ biasing constants K_i need to be generated. Furthermore, $|3 \cdot X + 2|_{2^n+1}$

instead of $|3 \cdot X|_{2^n+1}$ is generated by the proposed modulo 2^n+1 HMG depicted in Fig. 4.3. The additional bias of 2 can be accounted for by modifying the PP_i and K_i corresponding to the Booth encoded digit $d_i = \pm 3$ as follows and the modified PP_i and K_i are shown in Table 4.3.

$$\begin{aligned}
|+3 \cdot X \cdot 2^{3i}|_{2^n+1} &= |(3 \cdot X + 2) \cdot 2^{3i}|_{2^n+1} - |2 \cdot 2^{3i}|_{2^n+1} \\
&= CCLS(|3X + 2|_{2^n+1}, 3i) - 2^{3i} + 1 - 2 \cdot 2^{3i} \\
&= CCLS(|3X + 2|_{2^n+1}, 3i) - 2^{3i+1} - 2^{3i} + 1
\end{aligned} \tag{4.33}$$

Table 4.3 Modulo 2^n+1 reduced partial products for the encoded multiplier digits ± 3

d_i	$ X \cdot d_i \cdot 2^{3i} _{2^n+1} = PP_i + K_i$	
	PP_i	K_i
+3	$CCLS(+3X + 2 _{2^n+1}, 3i)$	$-2^{3i+1} - 2^{3i} + 1$
-3	$CCLS(+3X + 2 _{2^n+1}, 3i)$	$2^{3i+1} + 2^{3i} + 1$

Every Booth encoded multiplier digit d_i generates a modulo-reduced partial product $|X \cdot d_i \cdot 2^{3i}|_{2^n+1}$ that involves a modulo 2^n+1 multiplication by the power-of-two term 2^{3i} . By *Property 4.2*, modulo 2^n+1 multiplication by 2^{3i} introduces a bias of $-2^{3i}+1$. Hence K_i can be divided into two parts, $KS_i = -2^{3i}+1$ and $KD_i = K_i - KS_i$. KS_i is a static bias, which is specific to the radix of the Booth encoding algorithm and independent of d_i , whereas KD_i is a dynamic bias which is dependent on d_i . From Tables 4.2 and 4.3, K_i and KD_i for all d_i are tabulated in Table 4.4.

Table 4.4 Dynamic bias for the encoded multiplier digits

d_i	K_i	$KD_i = K_i + 2^{3i} - 1$	d_i	K_i	$KD_i = K_i + 2^{3i} - 1$
+0	$-2^{3i} + 1$	0	-0	$2^{3i} + 1$	2^{3i+1}
+1	$-2^{3i} + 1$	0	-1	$2^{3i} + 1$	2^{3i+1}
+2	$-2^{3i+1} + 1$	-2^{3i}	-2	$2^{3i+1} + 1$	$2^{3i} + 2^{3i+1}$
+3	$-2^{3i} + 1 - 2^{3i+1}$	-2^{3i+1}	-3	$2^{3i} + 1 + 2^{3i+1}$	2^{3i+2}
+4	$-2^{3i+2} + 1$	$-2^{3i} - 2^{3i+1}$	-4	$2^{3i+2} + 1$	$2^{3i+2} + 2^{3i}$

The fixed KS_i can be easily aggregated into a single constant binary word. To minimize the number of binary words due to KD_i , let $KD_i = -2^{3i}a - 2^{3i+1}b + 2^{3i+1}c + 2^{3i+1}d + 2^{3i+2}e$, where $a, b, c, d, e \in \{0, 1\}$. The truth table of a, b, c, d, e with Boolean inputs, $s_i, m1_i, m2_i, m3_i, m4_i$, is given in Table 4.5. Those minterms not listed in Table 4.5 are don't cares.

Table 4.5 Truth table for Boolean functions of a, b, c, d, e

d_i	KD_i	$d_i = (-1)^{s_i} (m1_i + 2(m2_i) + 3(m3_i) + 4(m4_i))$					$KD_i = -2^{3i}a - 2^{3i+1}b + 2^{3i+1}c + 2^{3i+1}d + 2^{3i+2}e$				
		s_i	$m4_i$	$m3_i$	$m2_i$	$m1_i$	a	b	c	d	e
+0	0	0	0	0	0	0	0	0	0	0	0
+1	0	0	0	0	0	1	0	0	0	0	0
+2	-2^{3i}	0	0	0	1	0	1	0	0	0	0
+3	-2^{3i+1}	0	0	1	0	0	0	1	0	0	0
+4	$-2^{3i} - 2^{3i+1}$	0	1	0	0	0	1	1	0	0	0
-0	2^{3i+1}	1	0	0	0	0	0	0	1	0	0
-1	2^{3i+1}	1	0	0	0	1	0	0	1	0	0
-2	$-2^{3i} + 2^{3i+1} + 2^{3i+1}$	1	0	0	1	0	1	0	1	1	0
-3	$-2^{3i+1} + 2^{3i+1} + 2^{3i+2}$	1	0	1	0	0	0	1	1	0	1
-4	$-2^{3i} - 2^{3i+1} + 2^{3i+1} + 2^{3i+1} + 2^{3i+2}$	1	1	0	0	0	1	1	1	1	1

Minimization by a 5-variable Karnaugh map yields $a = m2_i \vee m4_i$, $b = m3_i \vee m4_i$, $c = s_i$, $d = (m2_i \vee m4_i) \wedge s_i$ and $e = (m3_i \vee m4_i) \wedge s_i$. Hence

$$KD_i = -(m2_i \vee m4_i)2^{3i} - (m3_i \vee m4_i)2^{3i+1} + s_i 2^{3i+1} + ((m2_i \vee m4_i) \wedge s_i)2^{3i+1} + ((m3_i \vee m4_i) \wedge s_i)2^{3i+2} \quad (4.34)$$

By applying *Property 4.1* to the negatively weighted $(m2_i \vee m4_i)$ and $(m3_i \vee m4_i)$ terms,

$$KD_i = (\overline{m2_i \vee m4_i})2^{3i} - 2^{3i} + (\overline{m3_i \vee m4_i})2^{3i+1} - 2^{3i+1} + s_i 2^{3i+1} + ((m2_i \vee m4_i) \wedge s_i)2^{3i+1} + ((m3_i \vee m4_i) \wedge s_i)2^{3i+2} \quad (4.35)$$

For the case of $\text{mod}(n, 3) = 2$, the aggregate sum of KS_i and KD_i is given by

$$\begin{aligned} \sum_{i=0}^{\lfloor n/3 \rfloor} K_i &= \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{(\overline{m2_i \vee m4_i})2^{3i} + (\overline{m3_i \vee m4_i})2^{3i+1} + s_i \cdot 2^{3i+1}}_{KD_i} \\ &\quad + \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{((m2_i \vee m4_i) \wedge s_i)2^{3i+1} + ((m3_i \vee m4_i) \wedge s_i)2^{3i+2}}_{KD_i} \\ &\quad + \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{-2^{3i} - 2^{3i+1}}_{KD_i} + \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{-2^{3i} + 1}_{KS_i} \\ &= K1 + K2 + K3 + \sum_{i=0}^{\lfloor n/3 \rfloor} (-2^{3i+2}) + \lfloor n/3 \rfloor + 1 \end{aligned} \quad (4.36)$$

where

$$\begin{aligned}
K1 &= \sum_{i=0}^{\lfloor n/3 \rfloor} \left(\overline{m2_i \vee m4_i} \right) 2^{3i} + \left(\overline{m3_i \vee m4_i} \right) 2^{3i+1} \\
K2 &= \sum_{i=0}^{\lfloor n/3 \rfloor} \left((m2_i \vee m4_i) \wedge s_i \right) 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} \left((m3_i \vee m4_i) \wedge s_i \right) 2^{3i+2} \\
K3 &= \sum_{i=0}^{\lfloor n/3 \rfloor} s_i 2^{3i+1}
\end{aligned} \tag{4.37}$$

By substituting for K_i in (4.6) by (4.37),

$$|Z|_{2^{n+1}} = \underbrace{\sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + X + Y + K1 + K2 + K3}_{\lfloor n/3 \rfloor + 6} + \sum_{i=0}^{\lfloor n/3 \rfloor} \left(-2^{3i+2} \right) + \lfloor n/3 \rfloor + 1 \tag{4.38}$$

Equation (4.38) implies that Z is a sum of $\lfloor n/3 \rfloor + 6$ partial products and a constant term $\sum_{i=0}^{\lfloor n/3 \rfloor} \left(-2^{3i+2} \right) + \lfloor n/3 \rfloor + 1$. It will be shown that the constant term can be merged with the partial product $K3$ and hence the constant term is not considered as an additional partial product to be accumulated.

Then the accumulation of the $\lfloor n/3 \rfloor + 6$ partial products can be realized by an $\lfloor n/3 \rfloor + 6$ -operand (MOMA, 2^n+1). In Section 2.4.3.5, a k -operand (MOMA, 2^n+1) was shown to inherently add a constant of $k-1$. In other words, Z is incremented by a constant $\lfloor n/3 \rfloor + 5$ when the $\lfloor n/3 \rfloor + 6$ partial products are accumulated using a (MOMA, 2^n+1). To negate this effect, $\lfloor n/3 \rfloor + 5$ is subtracted from Z in (4.38), giving

$$\begin{aligned}
|Z|_{2^{n+1}} &= \underbrace{\sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + X + Y + K1 + K2 + K3}_{\lfloor n/3 \rfloor + 6} + \sum_{i=0}^{\lfloor n/3 \rfloor} \left(-2^{3i+2} \right) - 4 \\
&= \underbrace{\sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + X + Y + K1 + K2 + K3}_{\lfloor n/3 \rfloor + 6} + \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i+1}
\end{aligned} \tag{4.39}$$

The term $\sum_{i=1}^{\lfloor n/3 \rfloor} 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i+1}$ can be merged with $K3$ and the final product is given by

$$|Z|_{2^n+1} = \underbrace{\sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + X + Y + K1 + K2 + K3'}_{\lfloor n/3 \rfloor + 6} \quad (4.40)$$

where $K3' = \sum_{i=1}^{\lfloor n/3 \rfloor} 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} s_i \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor} s_i \cdot 2^{3i+2}$.

Similarly, by suitably modifying the dynamic bias corresponding to the most significant signed multiplier digit, *i.e.*, $KD_{\lfloor n/3 \rfloor}$, the aggregate sum of $\lfloor n/3 \rfloor + 1$ bias K_i can be reduced to three n -bit words for the cases of $\text{mod}(n,3) = 0$ and 1 as well.

4.4.2.3 Accumulation of partial products

From (4.40), the total number of partial products to be accumulated is $\lfloor n/3 \rfloor + 6$. The CSA tree based accumulation of the partial products in an $\lfloor n/3 \rfloor + 6$ -operand (MOMA, 2^n+1) is illustrated in Fig. 4.8 for $n = 8$. The $\lfloor n/3 \rfloor + 6$ -operand (MOMA, 2^n+1) consists of $\lfloor n/3 \rfloor + 4$ CEAC CSAs followed by a two-operand modulo 2^n+1 adder.

Fig. 4.8 MPPA for radix-8 Booth encoded modulo 2^n+1 multiplier

4.5 Performance analysis

In this section, the performances of the proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers are compared against non-encoded and radix-4 Booth encoded modulo multipliers for application in the three moduli set $\{2^n-1, 2^n, 2^n+1\}$ RNS. The following five configurations of RNS multipliers are considered. Type A radix-8 Booth encoded RNS multiplier uses the proposed modulo 2^n-1 and modulo 2^n+1 multipliers (Prop.). Types B and C radix-4 Booth encoded RNS multipliers use the same modulo 2^n-1 multiplier [Efst04a], but the former uses the design [Chen10] and the latter uses the design [Sous05] for the modulo 2^n+1 multiplier. Types D and E non-encoded RNS multipliers use the modulo 2^n-1 multiplier of [Zimm99]. While Type D uses the diminished-one modulo 2^n+1 multiplier of [Efst05], Type E uses the modulo 2^n+1 multiplier for binary representation of [Verg07].

In all modulo 2^n-1 and modulo 2^n+1 multipliers, the final two-operand adder was consistently implemented by a Sklansky parallel-prefix structure with an additional prefix level for EAC and CEAC additions, respectively [Zimm99]. The multipliers, Prop., [Zimm99], [Efst04a], [Efst05], [Sous05], [Chen10] and [Verg07], were specified in VHDL and synthesized by Synopsys Design Compiler tool (version D-2010.03) under nominal synthesis environment, *i.e.*, 25°C and 1.8V based on TSMC 0.18 μ m 1.8V CMOS standard-cell library. The designs were optimized for area and delay independently. The average dynamic power dissipation is also estimated based on the Monte Carlo method with 99.9% confidence that the error is bounded below 3% for a clock period of 50ns. The minimum achievable area from area-constrained optimization, the least achievable critical path delay from delay-constrained optimization, the average dynamic power dissipation and the leakage power dissipation for modulo 2^n-1 and modulo 2^n+1 multipliers are tabulated in Tables 4.6-4.9. The individual results in Tables 4.6 - 4.9 are presented as precursors to the comparison of Types A-E RNS multipliers in Tables 4.10 - 4.12.

Table 4.6 Area and delay evaluation of modulo 2^n-1 multipliers

n	Area (μm^2)			Delay (ns)		
	Prop.	[Efst04a]	[Zimm99]	Prop.	[Efst04a]	[Zimm99]
32	61535	63920	93564	3.55	3.31	3.50
36	80871	80465	118127	3.70	3.54	3.75
40	95796	99060	145696	4.17	4.04	3.72
44	112059	119570	176139	4.14	4.01	4.34
48	136462	142130	209589	4.30	4.00	4.43
52	155608	166645	245781	4.28	4.09	4.37
56	176122	193050	284979	4.39	4.17	4.37
60	206383	221371	327051	4.68	4.69	4.57
64	229791	251745	372134	4.67	4.74	5.08

Table 4.7 Power dissipation evaluation of modulo 2^n-1 multipliers

n	Dynamic power dissipation (mW)			Leakage power dissipation (μW)		
	Prop.	[Efst04a]	[Zimm99]	Prop.	[Efst04a]	[Zimm99]
32	0.690	0.725	0.868	0.392	0.472	0.688
36	0.899	0.909	0.981	0.514	0.592	0.868
40	1.061	1.115	1.301	0.609	0.727	1.069
44	1.242	1.343	1.562	0.711	0.875	1.291
48	1.474	1.610	1.826	0.865	1.038	1.533
52	1.715	2.020	2.125	0.985	1.214	1.796
56	1.857	2.230	2.319	1.113	1.404	2.080
60	2.047	2.421	2.409	1.304	1.607	2.384
64	2.140	2.614	2.454	1.452	1.824	2.710

Table 4.8 Area and delay evaluation of modulo 2^n+1 multipliers

n	Area (μm^2)					Delay (ns)				
	Prop.	[Chen10]	[Sous05]	[Efst05]	[Verg07]	Prop.	[Chen10]	[Sous05]	[Efst05]	[Verg07]
32	77724	85411	152152	100111	95317	3.90	3.14	3.48	3.62	3.60
36	99565	106824	190739	125275	119883	4.19	3.92	3.80	3.77	3.95
40	116783	130767	233822	153400	147406	4.25	3.84	4.15	4.29	4.02
44	135294	157285	281273	184302	177709	4.61	4.02	4.18	4.36	4.42
48	162288	186132	333218	218158	210966	4.62	3.75	4.17	4.37	4.32
52	183727	217420	389395	254702	246912	4.59	3.95	4.51	4.53	4.52
56	206569	251070	450068	294200	285810	4.77	4.48	5.05	4.58	4.54
60	239580	287114	515119	336518	327530	4.98	4.42	5.01	4.65	4.61
64	266421	325894	584648	381790	372204	4.85	4.62	4.78	4.90	4.92

Table 4.9 Power dissipation evaluation of modulo 2^n+1 multipliers

n	Dynamic power dissipation (mW)					Leakage power dissipation (μW)				
	Prop.	[Chen10]	[Sous05]	[Efst05]	[Verg07]	Prop.	[Chen10]	[Sous05]	[Efst05]	[Verg07]
32	0.843	1.211	1.544	0.898	0.927	0.495	0.666	0.731	0.717	0.683
36	1.077	1.649	2.012	1.237	1.164	0.631	0.832	0.914	0.896	0.858
40	1.270	2.086	2.480	1.441	1.368	0.740	1.018	1.117	1.095	1.054
44	1.446	2.524	3.004	1.777	1.694	0.855	1.221	1.340	1.314	1.268
48	1.715	2.869	3.549	2.114	2.008	1.023	1.443	1.584	1.554	1.504
52	2.099	3.389	4.150	2.431	2.306	1.157	1.683	1.848	1.812	1.758
56	2.267	3.664	4.652	2.593	2.460	1.298	1.943	2.132	2.091	2.034
60	2.469	4.056	4.883	2.709	2.560	1.503	2.220	2.438	2.390	2.328
64	2.602	4.262	5.086	2.801	2.655	1.667	2.516	2.764	2.709	2.642

The areas of Types A-E RNS multipliers are computed from Tables 4.6 and 4.8 and compared in Table 4.10. Similarly, the total power dissipations of Types A-E RNS multipliers are calculated from Tables 4.7 and 4.9 and compared in Table 4.11. The area and power dissipation of the modulo 2^n multiplier are common to all five RNS multipliers and can therefore be excluded. In Table 4.12, the critical path delay of the RNS multiplier is determined by either the modulo 2^n-1 multiplier delay of Table 4.6 or the modulo 2^n+1 multiplier delay of Table 4.8.

Table 4.10 Area comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1

n	Area (μm^2)					% area saving of Type A over			
	Type A (Prop.)	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])
32	139259	149331	216072	193675	188881	6.7	35.5	28.0	26.2
36	180436	187289	271204	243402	238010	3.6	33.4	25.8	24.1
40	212579	229827	332882	299096	293102	7.5	36.1	28.9	27.4
44	247353	276855	400843	360441	353848	10.6	38.2	31.3	30.0
48	298750	328262	475348	427747	420555	8.9	37.1	30.1	28.9
52	339335	384065	556040	500483	492693	11.6	38.9	32.1	31.1
56	382691	444120	643118	579179	570789	13.8	40.4	33.9	32.9
60	445963	508485	736490	663569	654581	12.2	39.4	32.7	31.8
64	496212	577639	836393	753924	744338	14.0	40.6	34.1	33.3

Table 4.11 Total power dissipation comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1

n	Total power dissipation (mW)					% power saving of Type A over			
	Type A (Prop.)	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])
32	1.533	1.937	2.270	1.767	1.796	20.8	32.4	13.2	14.6
36	1.977	2.559	2.922	2.219	2.146	22.7	32.3	10.9	7.8
40	2.332	3.202	3.596	2.744	2.671	27.1	35.1	15.0	12.6
44	2.689	3.869	4.349	3.341	3.258	30.4	38.1	19.5	17.4
48	3.190	4.481	5.161	3.943	3.837	28.8	38.1	19.0	16.8
52	3.816	5.411	6.173	4.559	4.434	29.4	38.1	16.3	13.9
56	4.126	5.897	6.885	4.916	4.783	30.0	40.0	16.0	13.7
60	4.518	6.480	7.308	5.122	4.973	30.2	38.1	11.7	9.1
64	4.745	6.880	7.704	5.260	5.114	31.0	38.4	9.7	7.2

From Table 4.10, Type A RNS multiplier achieves about 4~14%, 33~40%, 26~34% and 24~33% area savings over radix-4 Booth encoded Type B and Type C RNS multipliers as well as non-encoded Type D and Type E RNS multipliers, respectively. From Table 4.11, Type A RNS multiplier also dissipates about 21~31%, 32~40%, 10~19% and 7~17% less power than Types B-E RNS multipliers, respectively. The area and power reduction are more prominent for large n , which occurs in applications for which the use of RNS is most desirable.

Table 4.12 Delay comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1

n	Delay (ns)					% delay increase of Type A over			
	Type A (Prop.)	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])
32	3.90	3.31	3.48	3.62	3.60	17.8	12.0	7.7	8.3
36	4.19	3.92	3.80	3.77	3.95	6.8	10.2	11.1	6.0
40	4.25	4.04	4.15	4.29	4.02	5.1	2.4	-0.9	5.7
44	4.61	4.02	4.18	4.36	4.42	14.6	10.2	5.7	4.2
48	4.62	4.00	4.17	4.43	4.43	15.5	10.7	4.2	4.2
52	4.59	4.09	4.51	4.53	4.52	12.2	1.7	1.3	1.5
56	4.77	4.48	5.05	4.58	4.54	6.4	-5.5	4.1	5.0
60	4.98	4.69	5.01	4.65	4.61	6.1	-0.5	7.0	8.0
64	4.85	4.74	4.78	5.08	5.08	2.3	1.4	-4.5	-4.5

From Table 4.12, Type A RNS multiplier incurs maximum delay penalties of 17%, 12%, 11% and 8% over Types B-E RNS multipliers, respectively. In general, the percentage increase in delay is comparatively smaller than the percentage reduction in area and power dissipation.

To show that the reduction in logic complexity of the proposed multipliers is not due to the mapping of specific cell library, the architectures of Types A-E RNS multipliers are compared theoretically using the normalized area model. The areas of logic modules normalized to the area of an inverter for the TSMC 0.18 μ m CMOS standard-cell library are tabulated in Table 4.13 [TSMC03]. Using Table 4.13, the normalized area expressions of the proposed modulo 2^n-1 multiplier, [Efst04a] and [Zimm99] are tabulated in Table 4.14 while those of the proposed modulo 2^n+1 multiplier, [Efst05], [Sous05], [Chen10] and [Verg07] are tabulated in Table 4.15.

Table 4.13 Normalized area of logic modules

Logic module	Notation	Normalized area
Inverter	INV	1
2-input AND	AND2	2
3-input AND	AND3	2.5
2-input OR	OR2	2
4-input OR	OR4	3
2-input NAND	NAND2	1.5
2-input NOR	NOR2	1.5
3-input AOI	AOI3	2
2-input XOR	XOR2	4
2-input XNOR	XNOR2	4
Half Adder	HA	5.5
Full Adder	FA	10.5
2:1 Multiplexer	MUX2	4

Table 4.14 Normalized area expressions of modulo 2^n-1 multipliers

Multiplier	Component	Normalized area of component	Number of component
Prop.	HMG pre-processing	$2A_{AND2} + 2A_{OR2} + A_{XOR2} = 12$	n
	HMG prefix	$2A_{AND2} + A_{OR2} = 6$	$n(\lceil \log_2 n \rceil - 1)$
	HMG post-processing	$A_{XOR2} = 4$	n
	BE	$3A_{INV} + 3A_{AND2} + A_{AND3} + 3A_{XOR2} = 23.5$	$\lfloor n/3 \rfloor + 1$
	BS	$4A_{AND2} + A_{OR4} + A_{XOR2} = 15$	$n(\lfloor n/3 \rfloor + 1)$
	FA	$A_{FA} = 10.5$	$n(\lfloor n/3 \rfloor - 1)$
	Total	$25.5n(\lfloor n/3 \rfloor) + 23.5(\lfloor n/3 \rfloor + 1) + 14.5n + 6n(\lceil \log_2 n \rceil)$	
[Efst04a]	BE	$A_{INV} + A_{AND2} + 2A_{XOR2} = 11$	$n/2$
	BS	$2A_{AND2} + A_{OR2} + A_{XOR2} = 10$	$n(n/2)$
	FA	$A_{FA} = 10.5$	$n(n/2 - 2)$
	Total	$10.25n^2 - 15.5n$	
[Zimm99]	AND	$A_{AND2} = 2$	$n(n)$
	FA	$A_{FA} = 10.5$	$n(n-2)$
	Total	$12.5n^2 - 21n$	

Table 4.15 Normalized area expressions of modulo 2^n+1 multipliers

Multiplier	Component	Normalized area of component	Number of component
Prop.	HMG pre-processing	$2A_{AND2} + 2A_{OR2} + 1A_{XOR2} = 12$	n
	HMG modified prefix	$2A_{AOI3} + 3A_{INV} = 7$	$n - 4$
	HMG prefix	$2A_{AND2} + A_{OR2} = 6$	$2n\lceil \log_2 n \rceil - 6n + 12$
	HMG post-processing	$A_{XOR2} = 4$	n
	BE	$3A_{INV} + 3A_{AND2} + A_{AND3} + 3A_{XOR2} = 23.5$	$\lfloor n/3 \rfloor + 1$
	BS	$4A_{AND2} + A_{OR4} + A_{XOR2} = 15$	$n(\lfloor n/3 \rfloor + 1)$
	FA	$A_{FA} = 10.5$	$n(\lfloor n/3 \rfloor + 4)$
	Inverter	$A_{INV} = 1$	$1.5(\lfloor n/3 \rfloor)(\lfloor n/3 \rfloor + 1) + 3(\lfloor n/3 \rfloor) + 13$
	Total	$1.5(\lfloor n/3 \rfloor)^2 + 25.5n(\lfloor n/3 \rfloor + 1) + 28(\lfloor n/3 \rfloor + 1) + 18.5n + 12n(\lceil \log_2 n \rceil) + 52.5$	
[Chen10]	CTG	$A_{XOR2} = 4$	$n/2$
	BE	$2A_{AND2} + 2A_{NOR2} + A_{XOR2} + A_{XNOR2} = 15$	$n/2$
	BS	$2A_{AND2} + A_{OR2} + 2A_{XOR2} = 14$	$n(n/2)$
	FA	$A_{FA} = 10.5$	$n(n/2)$
	Inverter	$A_{INV} = 1$	$(n/2)(n/2) + n/2 + 1$
	Total	$n^2/4 + 24.5n^2/2 + 10n + 1$	
[Sous05]	8:1 Multiplexer	$7A_{MUX2} = 28$	$n(n/2 + 1)$
	FA	$A_{FA} = 10.5$	$n(n/2)$
	Inverter	$A_{INV} = 1$	$n(n/2) + n/2$
	Total	$39.5n^2/2 + 28.5n$	
[Efst05]	AND	$A_{AND2} = 2$	$n^2/2 + n/2$
	NAND	$A_{NAND2} = 1.5$	$n^2/2 - n/2$
	FA	$A_{FA} = 10.5$	$n(n+1)$
	Total	$12.25n^2 + 10.75n$	
[Verg07]	AND	$A_{AND2} = 2$	$n(n/2) + 7n/2$
	NAND	$A_{NAND2} = 1.5$	$(n-1)(n-2)/2$
	OR	$A_{OR2} = 2$	$n+3$
	NOR	$A_{NOR2} = 1.5$	$n-1$
	FA	$A_{FA} = 10.5$	$n(n-1)$
	Total	$12.25n^2 - 2.25n + 6$	

From the normalized area expressions of their constituent modulo 2^n-1 and modulo 2^n+1 multiplier implementations shown in Tables 4.14 and 4.15, the normalized areas of Types A-E RNS multipliers are calculated and tabulated in Table 4.16 for comparison. The percentage savings in normalized area is well correlated with the percentage area savings obtained from the synthesis results of Table 4.10.

Table 4.16 Normalized area comparison of RNS multipliers based on moduli 2^n-1 and 2^n+1

n	Normalized area					% normalized area saving of Type A over			
	Type A (Prop.)	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])	Type B ([Efst04a], [Chen10])	Type C ([Efst04a], [Sous05])	Type D ([Zimm99], [Efst05])	Type E ([Zimm99], [Verg07])
32	21841	23121	31136	25016	24606	5.5	29.8	12.6	11.2
36	28964	29287	39348	31707	31245	1.1	26.3	8.6	7.3
40	34207	36181	48520	39190	38676	5.4	29.4	12.7	11.5
44	39861	43803	58652	47465	46899	8.9	32.0	16.0	15.0
48	48472	52153	69744	56532	55914	7.0	30.5	14.2	13.3
52	55155	61231	81796	66391	65721	9.9	32.5	16.9	16.0
56	62249	71037	94808	77042	76320	12.3	34.3	19.2	18.4
60	72924	81571	108780	88485	87711	10.6	32.9	17.5	16.8
64	81047	92833	123712	100720	99894	12.6	34.4	19.5	18.8

4.6 Summary

New area-power efficient modulo 2^n-1 and modulo 2^n+1 multipliers employing radix-8 Booth encoding scheme were proposed. In the proposed modulo 2^n-1 multiplier, the modulo-reduced partial products were generated with no accompanying bias. Thus the number of modulo-reduced partial products in modulo 2^n-1 multiplier was lowered to $\lfloor n/3 \rfloor + 1$, which is the best achievable partial product count using radix-8 Booth encoding scheme. For the proposed modulo 2^n+1 multiplier, the aggregate sum of biasing constants imposed by hard multiple and partial product generations was partitioned into multiplier-dependent dynamic bias and multiplier-independent static bias. The bias was succinctly expressed as three n -bit binary words generated by hardwiring the Booth encoder outputs and constant ones to appropriate bit positions via at most two levels of AND gates. Consequently, the number of modulo-reduced partial products was reduced to $\lfloor n/3 \rfloor + 6$ in the modulo 2^n+1 multiplier. All in all, the customized hard multiple generation, simpler correction term for modulus 2^n+1 and fewer modulo-reduced partial products collectively improve the area-power performance of the proposed modulo multipliers over the existing radix-4 Booth encoded and non-encoded modulo 2^n-1 and 2^n+1 multipliers for the three moduli set $\{2^n-1, 2^n, 2^n+1\}$ RNS.

Chapter 5

Radix-4 and Radix-8 Booth Encoded Multi-Modulus Multipliers

5.1 Introduction

An architecture that performs the same modulo operation for more than one modulus is called multi-modulus architecture [Pali99]. Multi-modulus architectures are particularly employed in variable word-length and fault-tolerant processors [Card02], [Jenk92], [Jenk93a], [Jenk93b]. In the variable word-length processor, multi-modulus architecture facilitates the selection of appropriate moduli to achieve the desired word-length. In the fault-tolerant processor, the redundant modulo channels are ideal candidates for multi-modulus architectures.

The two types of multi-modulus architectures are fixed and variable. The modulo operations corresponding to the different moduli are performed simultaneously in a fixed architecture and serially in a variable architecture. In contrast to the single modulus architecture, multi-modulus architecture exploits hardware reuse amongst the modulo channels. Hence both fixed and variable multi-modulus architectures result in considerable savings in area as well as power dissipation. However when compared to the single modulus architecture, the fixed architecture incurs no delay penalty while the variable architecture causes a marginal increase in delay.

In this Chapter, two multi-modulus multiplier architectures for the triple moduli set $\{2^n-1, 2^n, 2^n+1\}$ are described. The former uses the radix-2² while the latter uses the radix-2³ Booth encoding technique. As the Booth encoder is the only component that can be shared amongst the modulo 2^n-1 , the modulo 2^n and the modulo 2^n+1 multipliers, fixed multi-modulus architecture has limited advantage. Hence, variable architectures are employed for the

proposed radix-2² as well as radix-2³ Booth encoded multi-modulus multipliers. For maximal hardware reuse, a new radix-2² Booth encoded modulo 2ⁿ+1 multiplier with architecture similar to that of the modulo 2ⁿ-1 multiplier is introduced. Furthermore, radix-2² and radix-2³ Booth encoded modulo 2ⁿ multipliers are described. An efficient technique to generate the bias associated with modulo 2ⁿ negation is also proposed.

The Chapter is organized as follows. In Section 5.2, the properties of modulo 2ⁿ arithmetic are presented. Sections 5.3 and 5.4 describe the proposed radix-2² and radix-2³ Booth encoded multi-modulus multipliers, respectively with emphasis on multi-modulus partial product, hard multiple and bias generations as well as multi-modulus partial product accumulation. In Section 5.5, the performance of the proposed multi-modulus multipliers are evaluated and compared against {2ⁿ-1, 2ⁿ, 2ⁿ+1} based RNS multipliers. A brief summary is presented in Section 5.6.

5.2 Preliminaries

In the following, binary representation with dual zeros for modulo 2ⁿ-1 arithmetic and diminished-1 representation for modulo 2ⁿ+1 arithmetic are considered. All equations are assumed to be modulo-reduced by m , $m \in \{2^n-1, 2^n, 2^n+1\}$ unless otherwise specified. Let $X = \sum_{i=0}^{n-1} x_i 2^i$, $Y = \sum_{i=0}^{n-1} y_i 2^i$ and $Z = \sum_{i=0}^{n-1} z_i 2^i$ be the n -bit multiplicand, multiplier and product, respectively. The modulo m multiplication can be expressed as follows.

$$|Z|_m = \begin{cases} |X \cdot Y|_m = \left| \sum_{i=0}^{n-1} X \cdot y_i 2^i \right|_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ |X \cdot Y + X + Y|_m = \left| \sum_{i=0}^{n-1} X \cdot y_i 2^i + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.1)$$

In the radix-2^k Booth encoding algorithm, k adjacent multiplier bits $y_{ki+k-1}y_{ki+k-2} \dots y_{ki+1}y_{ki}$ and an overlap multiplier bit y_{ki-1} are scanned and converted to a signed digit d_i , $d_i \in [-2^{k-1}, 2^{k-1}]$. Thus, the n multiplier bits are encoded as $\lfloor n/k \rfloor + 1$ multiplier digits. The digit set conversion is given by

$$d_i = -2^{k-1} y_{ki+k-1} + \sum_{j=0}^{k-2} 2^j y_{ki+j} + y_{ki-1} \quad (5.2)$$

In addition to *Properties 3.1 – 3.3* of modulo 2^n-1 arithmetic and *Properties 4.1 – 4.3* of modulo 2^n+1 arithmetic, the following properties of modulo 2^n arithmetic are employed in the design of multi-modulus multiplier architectures.

Property 5.1: By (2.4), the modulo reduction of the binary weight of x in excess of the modulus becomes

$$\left| x \cdot 2^{n+i} \right|_{2^n} = 0 \quad (5.3)$$

Property 5.2: Let $LS(X, j)$ denote the left shift of X by j bit positions where the j lsbs are zeros. As a corollary of *Property 5.1*, the modulo 2^n multiplication of X by a positive power-of-two term, 2^j is equivalent to a LS operation.

$$\begin{aligned} \left| 2^j \cdot X \right|_{2^n} &= \left| \sum_{i=0}^{n-1} x_i 2^{i+j} \right|_{2^n} \\ &= \sum_{i=0}^{n-1-j} x_i 2^{i+j} \\ &= LS(X, j) \end{aligned} \quad (5.4)$$

Property 5.3: In modulo 2^n arithmetic, as $-X$ is equivalent to the two's complement of X , *i.e.*, $\bar{X} + 1$, the modulo 2^n multiplication of X by -2^j is given by

$$\begin{aligned} \left| -2^j \cdot X \right|_{2^n} &= \left| 2^j (\bar{X} + 1) \right|_{2^n} \\ &= LS(\bar{X}, j) + 2^j \end{aligned} \quad (5.5)$$

5.3 Radix-2² Booth encoded multi-modulus multiplier

5.3.1 Multi-modulus partial product generation

By substituting for $k = 2$ into (5.2), the radix-2² Booth encoding of the multiplier is given by

$$\sum_{i=0}^{n-1} y_i \cdot 2^i = (y_0 + 2y_1) + \sum_{i=1}^{n/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) \cdot 2^{2i} + y_{n-1} \cdot 2^n \quad (5.6)$$

where n is even. The term $y_{n-1}2^n$ can be modulo-reduced using *Properties 3.1, 4.1* and *5.1* as follows.

$$\left| y_{n-1} \cdot 2^n \right|_m = \begin{cases} y_{n-1} & \text{if } m = 2^n - 1 \\ 0 & \text{if } m = 2^n \\ \bar{y}_{n-1} - 1 & \text{if } m = 2^n + 1 \end{cases} \quad (5.7)$$

Hence, (5.6) can be simplified to

$$\begin{aligned} \sum_{i=0}^{n-1} y_i \cdot 2^i &= \begin{cases} \sum_{i=0}^{n/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) \cdot 2^{2i} & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ -1 + \sum_{i=0}^{n/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) \cdot 2^{2i} & \text{if } m = 2^n + 1 \end{cases} \\ &= \begin{cases} \sum_{i=0}^{n/2-1} d_i \cdot 2^{2i} & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ -1 + \sum_{i=0}^{n/2-1} d_i \cdot 2^{2i} & \text{if } m = 2^n + 1 \end{cases} \end{aligned} \quad (5.8)$$

where

$$y_{-1} = \begin{cases} y_{n-1} & \text{if } m = 2^n - 1 \\ 0 & \text{if } m = 2^n \\ \bar{y}_{n-1} & \text{if } m = 2^n + 1 \end{cases} \quad (5.9)$$

The radix- 2^2 Booth encoding of the three overlapping bits $y_{2i+1}y_{2i}(y_{2i-1})$ to d_i is shown in Table 5.1.

Table 5.1 Radix- 2^2 Booth encoding

Multiplier bits $y_{2i+1}y_{2i}(y_{2i-1})$	Signed multiplier digit d_i	Multiplier bits $y_{2i+1}y_{2i}(y_{2i-1})$	Signed multiplier digit d_i
000	+0	100	-2
001	+1	101	-1
010	+1	110	-1
011	+2	111	-0

The radix- 2^2 Booth encoded digit d_i is formatted using three bits: a sign bit s_i and one-hot encoded magnitude bits, $m1_i$ and $m2_i$. The circuit implementation of a bit slice of the radix- 2^2 Booth Encoder (BE2) with inputs, y_{2i+1} , y_{2i} , y_{2i-1} , and outputs, s_i , $m1_i$, $m2_i$, is shown in Fig. 5.1(a). To select the appropriate y_{-1} bit corresponding to the modulus 2^n-1 , 2^n and 2^n+1 , a

specific 3:1 multiplexer (MUX3) depicted in Fig. 5.1(b) is used. $n/2$ BE2 and one MUX3 blocks are required to encode the n -bit multiplier, as shown in Fig. 5.1(c) for $n = 4$.

Fig. 5.1 (a) Bit slice of radix- 2^2 Booth Encoder (BE2) (b) Implementation of 3:1 multiplexer (MUX3) (c) Multi-modulus radix- 2^2 Booth Encoder

Using the radix- 2^2 Booth encoded multiplier form of (5.8), (5.1) becomes

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{n/2-1} X \cdot d_i \cdot 2^{2i} \right|_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ \left| \sum_{i=0}^{n/2-1} X \cdot d_i \cdot 2^{2i} + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.10)$$

The generation of the modulo-reduced partial products $|X \cdot d_i \cdot 2^{2i}|_m$ can be simplified by the number theoretic properties of modulo m arithmetic. By *Properties 3.2* and *3.3*, the modulo 2^n-1 multiplications of X by 2^j and -2^j can be implemented by *CLS* operations on X and $\bar{X}$, respectively. By *Properties 4.2* and *4.3*, the modulo 2^n+1 multiplications by 2^j and -2^j can be realized by *CCLS* operations on X with a bias of $-2^j + 1$ and on $\bar{X}$ with a bias of $2^j + 1$, respectively. Similarly by *Properties 5.2* and *5.3*, the modulo 2^n multiplication of X by 2^j and -2^j can be realized by *LS* operations on X with no bias and on $\bar{X}$ with a bias of 2^j , respectively. $|X \cdot d_i \cdot 2^{2i}|_m$ is shown to be equivalent to a *CLS* vector PP_i for the modulus 2^n-1 , the sum of a *LS* vector PP_i and a constant bias K_i for the modulus 2^n and the sum of a *CCLS* vector PP_i and a constant bias K_i for the modulus 2^n+1 in Table 5.2.

Table 5.2 Modulo m reduced partial products for radix- 2^2 Booth encoding

d_i	$PP_i = \left\lfloor X \cdot d_i \cdot 2^{2i} \right\rfloor_{2^n-1}$	$PP_i + K_i = \left\lfloor X \cdot d_i \cdot 2^{2i} \right\rfloor_{2^n}$	$PP_i + K_i = \left\lfloor X \cdot d_i \cdot 2^{2i} \right\rfloor_{2^n+1}$
	PP_i	PP_i K_i	PP_i K_i
+0	$\underbrace{0 \cdots 0}_n$	$\underbrace{0 \cdots 0}_n$ 0	$\underbrace{0 \cdots 0}_{n-2i} \underbrace{1 \cdots 1}_{2i}$ $-2^{2i}+1$
+1	$\underbrace{x_{n-1-2i} \cdots x_0}_{n-2i} \underbrace{x_{n-1} \cdots x_{n-2i}}_{2i}$	$\underbrace{x_{n-1-2i} \cdots x_0}_{n-2i} \underbrace{0 \cdots 0}_{2i}$ 0	$\underbrace{x_{n-1-2i} \cdots x_0}_{n-2i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-2i}}_{2i}$ $-2^{2i}+1$
+2	$\underbrace{x_{n-2-2i} \cdots x_0}_{n-2i} \underbrace{x_{n-2} \cdots x_{n-1-2i}}_{2i}$	$\underbrace{x_{n-2-2i} \cdots x_0}_{n-2i} \underbrace{00 \cdots 0}_{2i}$ 0	$\underbrace{x_{n-2-2i} \cdots x_0}_{n-2i} \underbrace{\bar{x}_{n-2} \cdots \bar{x}_{n-1-2i}}_{2i}$ $-2^{2i+1}+1$
-0	$\underbrace{1 \cdots 1}_n$	$\underbrace{1 \cdots 1}_{n-2i} \underbrace{0 \cdots 0}_{2i}$ 2^{2i}	$\underbrace{1 \cdots 1}_{n-2i} \underbrace{0 \cdots 0}_{2i}$ $2^{2i}+1$
-1	$\underbrace{\bar{x}_{n-1-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-2i}}_{2i}$	$\underbrace{\bar{x}_{n-1-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{0 \cdots 0}_{2i}$ 2^{2i}	$\underbrace{\bar{x}_{n-1-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{x_{n-1} \cdots x_{n-2i}}_{2i}$ $2^{2i}+1$
-2	$\underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{\bar{x}_{n-2} \cdots \bar{x}_{n-1-2i}}_{2i}$	$\underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{10 \cdots 0}_{2i}$ 2^{2i}	$\underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{x_{n-2} \cdots x_{n-1-2i}}_{2i}$ $2^{2i+1}+1$

The modulo 2^n reduced partial products for $d_i = +2$ and $d_i = -2$ are given by

$$\begin{aligned}
 \lfloor X \cdot (+2) \cdot 2^{2i} \rfloor_{2^n} &= LS(X, 2i+1) \\
 &= \underbrace{x_{n-2-2i} \cdots x_0}_{n-2i-1} \underbrace{0 \cdots 0}_{2i+1} \\
 &= \underbrace{x_{n-2-2i} \cdots x_0}_{n-2i} \underbrace{00 \cdots 0}_{2i}
 \end{aligned} \tag{5.11}$$

$$\begin{aligned}
 \lfloor X \cdot (-2) \cdot 2^{2i} \rfloor_{2^n} &= LS(\bar{X}, 2i+1) + 2^{2i+1} \\
 &= \underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i-1} \underbrace{0 \cdots 0}_{2i+1} + 2^{2i+1} \\
 &= \underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{00 \cdots 0}_{2i} + 2^{2i+1}
 \end{aligned} \tag{5.12}$$

From Table 5.2, for the modulus 2^n , it can be observed that the most significant $n-2i$ bits of PP_i s corresponding to $d_i = +0$ and -0 are bitwise complement of each other. Similarly, the most significant $n-2i$ bits of PP_i s corresponding to $d_i = +1$ and -1 are also bitwise complement of each other. However, from (5.11) and (5.12), in the PP_i s for $d_i = +2$ and -2 , only the most significant $n-2i-1$ bits are bitwise complement of each other. The bias corresponding to $d_i = -2$ is suitably modified such that the most significant $n-2i$ bits of the PP_i for $d_i = -2$ are bitwise complement of the PP_i for $d_i = +2$.

$$\lfloor X \cdot (-2) \cdot 2^{2i} \rfloor_{2^n} = \underbrace{\bar{x}_{n-2-2i} \cdots \bar{x}_0}_{n-2i} \underbrace{10 \cdots 0}_{2i} + 2^{2i} \tag{5.13}$$

Hence in Table 5.2, for the modulus 2^n and $d_i = -2$, the expressions of PP_i and d_i derived in (5.13) are employed.

Equation (5.10) can then be reformulated as

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{n/2-1} PP_i \right|_m & \text{if } m = 2^n - 1 \\ \left| \sum_{i=0}^{n/2-1} PP_i + \sum_{i=0}^{n/2-1} K_i \right|_m & \text{if } m = 2^n \\ \left| \sum_{i=0}^{n/2-1} PP_i + \sum_{i=0}^{n/2-1} K_i + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.14)$$

In the following, the generation of the $n/2$ PP_i for the three moduli, 2^n-1 , 2^n and 2^n+1 is described. Fig. 5.2(a) shows the circuit implementation of a bit slice of the radix- 2^2 Booth Selector (BS2) that generates a single bit of PP_i i.e., pp_{ij} , by selecting a bit of either the multiplicand or one-bit shifted multiplicand and conditionally inverting it. From Table 5.2, it can be observed that the least significant $2i$ bits of PP_i for modulo 2^n-1 , 2^n and 2^n+1 are pp_{ij} , 0 and $\overline{pp_{ij}}$, respectively. Hence, MUX3 of Fig. 5.1(b) can be used at the output of the BS2 blocks in the least significant $2i$ bit positions to select the appropriate pp_{ij} . Furthermore, when $j = 2i$, the input $x_{(j-2i-1) \bmod n}$ to the BS2 block is x_{n-1} , 0 and $\bar{x}_{n-1}$ for modulo 2^n-1 , 2^n and 2^n+1 , respectively. Thus this input to the BS2 block is selected using a MUX3. The generation of the $n/2$ PP_i using BS2 and MUX3 blocks for the three moduli is shown in Fig. 5.2(b) for $n = 4$. The number of BS2 and MUX3 blocks required in the PP_i generation is $n^2/2$ and $n^2/4$, respectively.

Fig. 5.2 (a) Bit slice of radix- 2^2 Booth Selector (BS2) (b) Multi-modulus generation of PP_i

5.3.2 Multi-modulus bias generation

Analogous to the generation of PP_i using BS2 blocks, the bias K_i can be generated by decoding d_i appropriately. This technique, while being straightforward, has an obvious limitation. By considering K_i as an n -bit binary word, the $n/2$ K_i doubles the number of partial products to be accumulated. Hence, the reduction in the number of partial products achieved by the radix- 2^2 Booth encoding technique is essentially nullified. An efficient technique to include K_i with minimal hardware circuitry based on the properties of modulo 2^n and modulo 2^n+1 arithmetic is proposed.

From Table 5.2, d_i and K_i for the modulus 2^n are tabulated in Table 5.3. In order to determine the relation between d_i and K_i , d_i is represented by the three bits, s_i , $m1_i$ and $m2_i$ while K_i is expressed as a function of the bit a .

Table 5.3 Bias for the modulus 2^n

d_i	K_i	$d_i = (-1)^{s_i} (m1_i + 2(m2_i))$			$K_i = (2^{2i})a$
		s_i	$m2_i$	$m1_i$	a
+0	0	0	0	0	0
+1	0	0	0	1	0
+2	0	0	1	0	0
-0	2^{2i}	1	0	0	1
-1	2^{2i}	1	0	1	1
-2	2^{2i}	1	1	0	1

From the 3-variable $(s_i, m1_i, m2_i)$ Karnaugh map with don't care minterms 011 and 111, it can be shown that $a = s_i$. Hence, the aggregate bias is given by the simple expression:

$$\left| \sum_{i=0}^{n/2-1} K_i \right|_{2^n} = \sum_{i=0}^{n/2-1} s_i \cdot 2^{2i} = K1 \quad (5.15)$$

$K1$ can be generated by merely hardwiring the s_i output of the BE2 blocks to appropriate bit positions.

For the modulus 2^n+1 , K_i is divided into a static bias KS_i and a dynamic bias KD_i . $KS_i = (-2^{ki}+1) = (-2^{2i}+1)$ is specific to the radix of the Booth encoding k but independent of d_i . On the other hand, $KD_i = K_i - KS_i$ is dependent on d_i . From Table 5.2, d_i and KD_i for the modulus 2^n+1 are tabulated in Table 5.4. d_i is represented by the three bits, s_i , $m1_i$ and $m2_i$ while K_i is expressed as a function of the bits a , b and c .

Table 5.4 Dynamic bias for the modulus 2^n+1

d_i	$KD_i = K_i - KS_i = K_i + 2^{2i} - 1$	$d_i = (-1)^{s_i} (m1_i + 2(m2_i))$			$KD_i = (-2^{2i})a + (2^{2i+1})b + (2^{2i+1})c$		
		s_i	$m2_i$	$m1_i$	a	b	C
+0	0	0	0	0	0	0	0
+1	0	0	0	1	0	0	0
+2	-2^{2i}	0	1	0	1	0	0
-0	2^{2i+1}	1	0	0	0	1	0
-1	2^{2i+1}	1	0	1	0	1	0
-2	$2^{2i} + 2^{2i+1} = -2^{2i} + 2^{2i+1} + 2^{2i+1}$	1	1	0	1	1	1

The dynamic bias can be minimized by a three-variable $(s_i, m1_i, m2_i)$ Karnaugh map, which results in $a = m2_i$, $b = s_i$ and $c = (s_i \wedge m2_i)$.

$$KD_i = (-m2_i)2^{2i} + (s_i)2^{2i+1} + (s_i \wedge m2_i)2^{2i+1} \quad (5.16)$$

By applying *Property 4.1* to the negatively weighted term $(-m2_i)2^{2i}$, (5.16) becomes

$$KD_i = (\overline{m2_i})2^{2i} - 2^{2i} + (s_i)2^{2i+1} + (s_i \wedge m2_i)2^{2i+1} \quad (5.17)$$

In the next stage, the partial products will be accumulated in a CEAC-CSA tree based $(n/2+3)$ -operand (MOMA, 2^n+1) that inherently increments the result by $n/2+2$. To negate this effect, the constant $n/2+2$ is subtracted from the aggregate bias. Hence, the aggregate of the static bias, the dynamic bias and the constant $-n/2-2$ is given by

$$\begin{aligned}
\left| \sum_{i=0}^{n/2-1} K_i \right|_{2^n+1} &= \sum_{i=0}^{n/2-1} K S_i + \sum_{i=0}^{n/2-1} K D_i - n/2 - 2 \\
&= \sum_{i=0}^{n/2-1} (-2^{2i} + 1) + \sum_{i=0}^{n/2-1} \overline{m2_i} \cdot 2^{2i} - 2^{2i} + s_i \cdot 2^{2i+1} + (s_i \wedge m2_i) 2^{2i+1} - n/2 - 2 \\
&= \sum_{i=0}^{n/2-1} \overline{m2_i} \cdot 2^{2i} + s_i \cdot 2^{2i+1} + (s_i \wedge m2_i) 2^{2i+1} + 2^{2i} \\
&= K1 + K2
\end{aligned} \tag{5.18}$$

where

$$\begin{aligned}
K1 &= \sum_{i=0}^{n/2-1} \overline{m2_i} \cdot 2^{2i} + \sum_{i=0}^{n/2-1} s_i \cdot 2^{2i+1} \\
K2 &= \sum_{i=0}^{n/2-1} 2^{2i} + \sum_{i=0}^{n/2-1} (s_i \wedge m2_i) 2^{2i+1}
\end{aligned} \tag{5.19}$$

Both $K1$ and $K2$ can be implemented by hardwiring the outputs $m2_i$ and s_i of BE2 as well as logic ones to appropriate bit positions via at most one level of AND gates.

From the expressions of $\left| \sum_{i=0}^{n/2-1} K_i \right|_m$ in (5.15) and (5.18), the aggregate bias for the three moduli, 2^n-1 , 2^n , 2^n+1 is given by the sum of only two n -bit words, $K1$ and $K2$ as follows.

$$K1 = \begin{cases} \sum_{i=0}^{n/2-1} 0 \cdot 2^{2i} + \sum_{i=0}^{n/2-1} 0 \cdot 2^{2i+1} & \text{if } m = 2^n - 1 \\ \sum_{i=0}^{n/2-1} s_i \cdot 2^{2i} + \sum_{i=0}^{n/2-1} 0 \cdot 2^{2i+1} & \text{if } m = 2^n \\ \sum_{i=0}^{n/2-1} \overline{m2_i} \cdot 2^{2i} + \sum_{i=0}^{n/2-1} s_i \cdot 2^{2i+1} & \text{if } m = 2^n + 1 \end{cases} \tag{5.20}$$

$$K2 = \begin{cases} \sum_{i=0}^{n/2-1} 0 \cdot 2^{2i} + \sum_{i=0}^{n/2-1} 0 \cdot 2^{2i+1} & \text{if } m = 2^n - 1 \text{ or } 2^n \\ \sum_{i=0}^{n/2-1} 1 \cdot 2^{2i} + \sum_{i=0}^{n/2-1} (s_i \wedge m2_i) \cdot 2^{2i+1} & \text{if } m = 2^n + 1 \end{cases} \tag{5.21}$$

Equation (5.20) can be implemented by using $n/2$ 3:1 multiplexers to select the coefficients for the first summation from $\{0, s_i, \overline{m2_i}\}$ at the $2i$ bit positions and $n/2$ 2:1 multiplexers (or equivalently AND gates) to select the coefficients for the second summation from $\{0, s_i\}$ at

the $2i+1$ bit positions. The 3:1 multiplexer used in the generation of $K1$ is a generic 3:1 multiplexer and not the MUX3 of Fig. 5.1(b) as its inputs are not correlated. Similarly (5.21) can be realized by employing n 2:1 multiplexers (or equivalently AND gates) to select the coefficients for the first summation from $\{0, 1\}$ at the $2i$ bit positions and the coefficients for the second summation from $\{0, s_i \wedge m2_i\}$ at the $2i+1$ bit positions.

5.3.3 Multi-modulus partial product accumulation

By replacing $\left| \sum_{i=0}^{n/2-1} K_i \right|_m$ in (5.14) with $K1$ of (5.20) and $K2$ of (5.21), the radix- 2^2 Booth encoded multi-modulus multiplication is given by (5.22). In (5.22), an n -bit all zeros partial product is included for the moduli 2^n-1 and 2^n so that the number of partial products to be accumulated is $n/2+3$ for all three moduli.

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{n/2-1} PP_i + K1 + K2 + 0 \right|_m & \text{if } m = 2^n - 1 \text{ or } 2^n \\ \left| \sum_{i=0}^{n/2-1} PP_i + K1 + K2 + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.22)$$

The modulo 2^n-1 , modulo 2^n and modulo 2^n+1 additions are equivalent to EAC, carry-ignore and CEAC additions, respectively. Hence, a multi-modulus addition can be implemented using a MUX3 in the carry feedback path that selects from $\{c_{out}, 0, \overline{c}_{out}\}$. The CSA tree based accumulation of $n/2+3$ partial products in a multi-modulus MOMA is illustrated in Fig. 5.3 for $n = 4$. The number of MUX3 blocks needed is $n/2+2$.

Fig. 5.3 Multi-modulus accumulation of partial products

5.4 Radix-2³ Booth encoded multi-modulus multiplier

5.4.1 Multi-modulus partial product generation

By substituting $k = 3$ into (5.2), the radix-2³ Booth encoding of the multiplier for the three moduli, 2^n-1 , 2^n and 2^n+1 is given by

$$\begin{aligned} \sum_{i=0}^{n-1} y_i \cdot 2^i &= \sum_{i=0}^{\lfloor n/3 \rfloor} (y_{3i-1} + 2y_{3i} + 2y_{3i+1} - 4y_{3i+2}) \cdot 2^{3i} \\ &= \sum_{i=0}^{\lfloor n/3 \rfloor} d_i \cdot 2^{3i} \end{aligned} \quad (5.23)$$

where $y_{-1} = y_n = y_{n+1} = y_{n+2} = 0$. The radix-2³ Booth encoding of the four overlapping bits $y_{3i+2}y_{3i+1}y_{3i}(y_{3i-1})$ to d_i is shown in Table 5.5.

Table 5.5 Radix-2³ Booth encoding

Multiplier bits $y_{3i+2}y_{3i+1}y_{3i}(y_{3i-1})$	Signed multiplier digit d_i	Multiplier bits $y_{3i+2}y_{3i+1}y_{3i}(y_{3i-1})$	Signed multiplier digit d_i
0000	+0	1000	-4
0001	+1	1001	-3
0010	+1	1010	-3
0011	+2	1011	-2
0100	+2	1100	-2
0101	+3	1101	-1
0110	+3	1110	-1
0111	+4	1111	-0

The radix-2³ Booth encoded multiplier digit is formatted using 5 bits: a sign bit s_i and four one-hot encoded magnitude bits $m1_i$, $m2_i$, $m3_i$ and $m4_i$. A bit slice of the radix-2³ Booth Encoder (BE3) is shown in Fig. 5.4(a). $\lfloor n/3 \rfloor + 1$ BE3 blocks are used to encode the n -bit multiplier as shown in Fig. 5.4(b) for $n = 4$.

Fig. 5.4 (a) Bit slice of radix-2³ Booth Encoder (BE3) (b) Multi-modulus radix-2³ Booth Encoder

By replacing the multiplier in (5.1) with its radix-2³ Booth encoded form of (5.23),

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} X \cdot d_i \cdot 2^{3i} \right|_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} X \cdot d_i \cdot 2^{3i} + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.24)$$

By *Properties 3.2, 3.3, 4.2, 4.3, 5.2 and 5.3*, the modulo-reduced partial products, $|X \cdot d_i \cdot 2^{3i}|_m$ are simplified and tabulated in Table 5.6. In Table 5.6, all partial products except $|+3X \cdot 2^{3i}|_m$ and $|-3X \cdot 2^{3i}|_m$, are shown to be generated by *CLS*, *LS* and *CCLS* operations on the multiplicand X when m is $2^n - 1$, 2^n and $2^n + 1$, respectively. In the subsequent section, a multi-modulus Hard Multiple Generator (HMG) that generates an n -bit H of either $|+3X|_{2^n - 1}$, $|+3X|_{2^n}$ or $|+3X + 2|_{2^n + 1}$ according to whether the selected modulus is $2^n - 1$, 2^n or $2^n + 1$, respectively is proposed. Thus the partial products corresponding to $d_i = +3$ can be generated by *CLS*, *LS* and *CCLS* operations on H for the modulus $2^n - 1$, 2^n and $2^n + 1$, respectively, as shown in Table 5.6.

Table 5.6 Modulo m reduced partial products for radix-2³ Booth encoding

d_i	$PP_i = \left\lfloor X \cdot d_i \cdot 2^{3i} \right\rfloor_{2^n-1}$	$PP_i + K_i = \left\lfloor X \cdot d_i \cdot 2^{3i} \right\rfloor_{2^n}$		$PP_i + K_i = \left\lfloor X \cdot d_i \cdot 2^{3i} \right\rfloor_{2^n+1}$	
	PP_i	PP_i	K_i	PP_i	K_i
+0	$\underbrace{0 \cdots 0}_n$	$\underbrace{0 \cdots 0}_n$	0	$\underbrace{0 \cdots 0}_{n-3i} \underbrace{1 \cdots 1}_{3i}$	$-2^{3i}+1$
+1	$\underbrace{x_{n-1-3i} \cdots x_0}_{n-3i} \underbrace{x_{n-1} \cdots x_{n-3i}}_{3i}$	$\underbrace{x_{n-1-3i} \cdots x_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	0	$\underbrace{x_{n-1-3i} \cdots x_0}_{n-3i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-3i}}_{3i}$	$-2^{3i}+1$
+2	$\underbrace{x_{n-2-3i} \cdots x_0}_{n-3i} \underbrace{x_{n-2} \cdots x_{n-1-3i}}_{3i}$	$\underbrace{x_{n-2-3i} \cdots x_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	0	$\underbrace{x_{n-2-3i} \cdots x_0}_{n-3i} \underbrace{\bar{x}_{n-2} \cdots \bar{x}_{n-1-3i}}_{3i}$	$-2^{3i+1}+1$
+3	$CLS(\lceil +3X \rceil_{2^n-1}, 3i)$	$LS(\lceil +3X \rceil_{2^n}, 3i)$	0	$CCLS(\lceil +3X \rceil_{2^n+1}, 3i)$	$-2^{3i}+1$
	$\underbrace{h_{n-1-3i} \cdots h_0}_{n-3i} \underbrace{h_{n-1} \cdots h_{n-3i}}_{3i}$	$\underbrace{h_{n-1-3i} \cdots h_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	0	$\underbrace{h_{n-1-3i} \cdots h_0}_{n-3i} \underbrace{\bar{h}_{n-1} \cdots \bar{h}_{n-3i}}_{3i}$	$-2^{3i+1}-2^{3i}+1$
+4	$\underbrace{x_{n-3-3i} \cdots x_0}_{n-3i} \underbrace{x_{n-1} \cdots x_{n-2-3i}}_{3i}$	$\underbrace{x_{n-3-3i} \cdots x_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	0	$\underbrace{x_{n-3-3i} \cdots x_0}_{n-3i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-2-3i}}_{3i}$	$-2^{3i+2}+1$
-0	$\underbrace{1 \cdots 1}_n$	$\underbrace{1 \cdots 1}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	2^{3i}	$\underbrace{1 \cdots 1}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	$2^{3i}+1$
-1	$\underbrace{\bar{x}_{n-1-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-3i}}_{3i}$	$\underbrace{\bar{x}_{n-1-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	2^{3i}	$\underbrace{\bar{x}_{n-1-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{x_{n-1} \cdots x_{n-3i}}_{3i}$	$2^{3i}+1$
-2	$\underbrace{\bar{x}_{n-2-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{\bar{x}_{n-2} \cdots \bar{x}_{n-1-3i}}_{3i}$	$\underbrace{\bar{x}_{n-2-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{1 \cdots 0}_{3i}$	2^{3i}	$\underbrace{\bar{x}_{n-2-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{x_{n-2} \cdots x_{n-1-3i}}_{3i}$	$2^{3i+1}+1$
-3	$CLS(\lceil +3X \rceil_{2^n-1}, 3i)$	$LS(\lceil +3X \rceil_{2^n}, 3i)$	2^{3i}	$CCLS(\lceil +3X \rceil_{2^n+1}, 3i)$	$2^{3i}+1$
	$\underbrace{\bar{h}_{n-1-3i} \cdots \bar{h}_0}_{n-3i} \underbrace{\bar{h}_{n-1} \cdots \bar{h}_{n-3i}}_{3i}$	$\underbrace{\bar{h}_{n-1-3i} \cdots \bar{h}_0}_{n-3i} \underbrace{0 \cdots 0}_{3i}$	2^{3i}	$\underbrace{\bar{h}_{n-1-3i} \cdots \bar{h}_0}_{n-3i} \underbrace{h_{n-1} \cdots h_{n-3i}}_{3i}$	$2^{3i+1}+2^{3i}+1$
-4	$\underbrace{\bar{x}_{n-3-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{\bar{x}_{n-1} \cdots \bar{x}_{n-2-3i}}_{3i}$	$\underbrace{\bar{x}_{n-3-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{1 \cdots 0}_{3i}$	2^{3i}	$\underbrace{\bar{x}_{n-3-3i} \cdots \bar{x}_0}_{n-3i} \underbrace{x_{n-1} \cdots x_{n-2-3i}}_{3i}$	$2^{3i+2}+1$

Hence, the radix-2³ Booth encoded modulo m multiplication can be reformulated as

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i \right|_m & \text{if } m = 2^n - 1 \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + \sum_{i=0}^{\lfloor n/3 \rfloor} K_i \right|_m & \text{if } m = 2^n \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + \sum_{i=0}^{\lfloor n/3 \rfloor} K_i + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.25)$$

The generation of PP_i for the three moduli, 2^n-1 , 2^n and 2^n+1 , according to Table 5.6 is described firstly. Fig. 5.5(a) shows a bit slice of the radix-2³ Booth Selector (BS3) that selects a bit from the multiplicand, the one-bit shifted multiplicand, the hard multiple or the two-bit shifted multiplicand, followed by a conditional bit inversion. Fig. 5.5(b) shows the generation of $\lfloor n/3 \rfloor + 1$ PP_i using $n(\lfloor n/3 \rfloor + 1)$ BS3 blocks for $n = 4$. MUX3 blocks are used at the output of the BS3 blocks in the least significant $3i$ bit positions to select from pp_{ij} , 0 or $\overline{pp_{ij}}$

when the modulus is 2^n-1 , 2^n or 2^n+1 , respectively. When $j = 3i$, MUX3 blocks are used at the inputs, x_{j-3i-1} and x_{j-3i-2} , of the BS3 block to select from $\{x_{n-1}, 0, \bar{x}_{n-1}\}$ and $\{x_{n-2}, 0, \bar{x}_{n-2}\}$, respectively. Likewise, when $j = 3i+1$ a MUX3 block is used at the input x_{j-3i-2} of the BS3 block to select from $\{x_{n-1}, 0, \bar{x}_{n-1}\}$. The number of MUX3 blocks needed is $(\lfloor n/3 \rfloor + 1)(1.5\lfloor n/3 \rfloor + 2) = 1.5\lfloor n/3 \rfloor^2 + 3.5\lfloor n/3 \rfloor + 2$.

Fig. 5.5 (a) Bit slice of radix-2³ Booth Selector (BS3) (b) Multi-modulus generation of PP_i

5.4.2 Multi-modulus hard multiple generation

An application-specific parallel-prefix adder known as HMG that computes only the sum of X and $\lfloor 2 \cdot X \rfloor_m$ is employed to generate the hard multiple $\lfloor +3 \cdot X \rfloor_m$. The HMG is designed by reformulating the carry equations of modulo 2^n-1 , 2^n and 2^n+1 additions using the bit correlation between the addends, X and $\lfloor 2 \cdot X \rfloor_m$. From Section 4.3, the simplified carry equation for multi-modulus HMG is given by (5.26) and (5.27) for even and odd cases of n , respectively.

$$c_i = \begin{cases} (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (g_{n-2}^*, p_{n-2}^*) \cdots \bullet (g_{i+2}^*, p_{i+2}^*) & \text{if } m = 2^n - 1 \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (0, 0) \cdots \bullet (0, 0) & \text{if } m = 2^n \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (\bar{p}_{n-2}^*, \bar{g}_{n-2}^*) \cdots \bullet (\bar{p}_{i+2}^*, \bar{g}_{i+2}^*) & \text{if } m = 2^n + 1 \end{cases} \quad (5.26)$$

$$c_i = \begin{cases} (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (g_{n-1}^*, p_{n-1}^*) \cdots \bullet (g_{i+2}^*, p_{i+2}^*) & \text{if } m = 2^n - 1 \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (0, 0) \cdots \bullet (0, 0) & \text{if } m = 2^n \\ (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \cdots \bullet (g_0^*, p_0^*) \bullet (\bar{p}_{n-1}^*, \bar{g}_{n-1}^*) \cdots \bullet (\bar{p}_{i+2}^*, \bar{g}_{i+2}^*) & \text{if } m = 2^n + 1 \end{cases} \quad (5.27)$$

where the modified-generate and modified-propagate signals pair (g_i^*, p_i^*) is defined as follows.

$$\begin{aligned} (g_0^*, p_0^*) &= \begin{cases} (x_{n-1} \cdot (x_0 + x_{n-2}), x_{n-1} + (x_0 \cdot x_{n-2})) & \text{if } m = 2^n - 1 \\ (0 \cdot (x_0 + 0), 0 + (x_0 \cdot 0)) = (0, 0) & \text{if } m = 2^n \\ (\bar{x}_{n-1} \cdot (x_0 + \bar{x}_{n-2}), \bar{x}_{n-1} + (x_0 \cdot \bar{x}_{n-2})) & \text{if } m = 2^n + 1 \end{cases} \\ (g_1^*, p_1^*) &= \begin{cases} (x_0 \cdot (x_1 + x_{n-1}), x_0 + (x_1 \cdot x_{n-1})) & \text{if } m = 2^n - 1 \\ (x_0 \cdot (x_1 + 0), x_0 + (x_1 \cdot 0)) = (x_0 \cdot x_1, x_0) & \text{if } m = 2^n \\ (x_0 \cdot (x_1 + \bar{x}_{n-1}), x_0 + (x_1 \cdot \bar{x}_{n-1})) & \text{if } m = 2^n + 1 \end{cases} \\ (g_i^*, p_i^*) &= (x_{i-1} \cdot (x_i + x_{i-2}), x_{i-1} + (x_i \cdot x_{i-2})) \end{aligned} \quad (5.28)$$

Fig. 5.6(a) shows the proposed multi-modulus HMG for $n = 8$. It consists of three stages: pre-processing, prefix computation and post-processing. Fig. 5.6(b) shows the gate level implementation of the operators in each stage.

In the preprocessing stage, each operator ‘■’ computes the (g_i^*, p_i^*) using AND-OR and OR-AND gates. The t_i bit is also computed by the XOR operation on the i -th bits of the two addends. Two MUX3 blocks are required at the input of the pre-processing operator to select from $\{x_{n-1}, 0, \bar{x}_{n-1}\}$ and $\{x_{n-2}, 0, \bar{x}_{n-2}\}$ for computing (g_0^*, p_0^*) and (g_1^*, p_1^*) , respectively.

In the prefix-computation stage, c_i is computed using the ‘●’ prefix operators in only $\lceil \log_2 n \rceil - 1$ levels. To implement the multi-modulus carry generation of (5.26) and (5.27), dual prefix operators are used in the first level such that the input to one prefix operator is (g_i^*, p_i^*) while the input to the other prefix operator is selected from $\{(g_i^*, p_i^*), (0, 0)\}$,

$(\bar{p}_i^*, \bar{g}_i^*)$ using MUX3 blocks. The number of MUX3 blocks required is $n-2$ and the number of dual prefix operators in each prefix level l is $n-2^{l+1}$, $1 \leq l \leq \lceil \log_2 n \rceil - 1$.

In the post-processing stage, each operator ‘ $\diamond$ ’ computes a bit of the hard multiple h_i by XOR of c_{i-1} and t_i . When $i = 0$, a MUX3 is employed at the c_{i-1} input of the post-processing operator to select from $\{c_{n-1}, 0, \bar{c}_{n-1}\}$ to implement the EAC, carry-ignore and CEAC additions, respectively.

Fig. 5.6 (a) Multi-modulus HMG (b) Circuit implementation of pre-processing, prefix and post-processing operators

5.4.3 Multi-modulus bias generation

From Table 5.6, d_i and K_i for the modulus 2^n are tabulated in Table 5.7.

Table 5.7 Bias for the modulus 2^n

d_i	K_i	$d_i = (-1)^{s_i} (m1_i + 2(m2_i) + 3(m3_i) + 4(m4_i))$					$K_i = (2^{3i})a$
		s_i	$m4_i$	$m3_i$	$m2_i$	$m1_i$	
+0	0	0	0	0	0	0	0
+1	0	0	0	0	0	1	0
+2	0	0	0	0	1	0	0
+3	0	0	0	1	0	0	0
+4	0	0	1	0	0	0	0
-0	2^{3i}	1	0	0	0	0	1
-1	2^{3i}	1	0	0	0	1	1
-2	2^{3i}	1	0	0	1	0	1
-3	2^{3i}	1	0	1	0	0	1
-4	2^{3i}	1	1	0	0	0	1

From Table 5.7, a can be deduced as s_i and the aggregate bias can be simplified to:

$$\left| \sum_{i=0}^{\lfloor n/3 \rfloor} K_i \right|_{2^n} = \sum_{i=0}^{\lfloor n/3 \rfloor} s_i \cdot 2^{3i} = K1 \quad (5.29)$$

For the modulus 2^n+1 , K_i is again divided into a static bias $KS_i = (-2^{3i}+1)$ that depends on the radix of the Booth encoding and a dynamic bias $KD_i = K_i - KS_i$ that depends on the multiplier digit d_i . From Table 5.6, d_i and KD_i for the modulus 2^n+1 are tabulated in Table 5.8.

Table 5.8 Dynamic bias for the modulus 2^n+1

d_i	$d_i = (-1)^{s_i} (m1_i + 2(m2_i) + 3(m3_i) + 4(m4_i))$					$KD_i = K_i - KS_i$ $= K_i + 2^{3i} - 1$	$KDi = (-2^{3i})a + (-2^{3i+1})b + (2^{3i+1})c + (2^{3i+1})d + (2^{3i+2})e$				
	s_i	$m4_i$	$m3_i$	$m2_i$	$m1_i$		a	b	c	d	E
+0	0	0	0	0	0	0	0	0	0	0	0
+1	0	0	0	0	1	0	0	0	0	0	0
+2	0	0	0	1	0	-2^{3i}	1	0	0	0	0
+3	0	0	1	0	0	-2^{3i+1}	0	1	0	0	0
+4	0	1	0	0	0	$-2^{3i} - 2^{3i+1}$	1	1	0	0	0
-0	1	0	0	0	0	2^{3i+1}	0	0	1	0	0
-1	1	0	0	0	1	2^{3i+1}	0	0	1	0	0
-2	1	0	0	1	0	$-2^{3i} + 2^{3i+1} + 2^{3i+1}$	1	0	1	1	0
-3	1	0	1	0	0	$-2^{3i+1} + 2^{3i+1} + 2^{3i+2}$	0	1	1	0	1
-4	1	1	0	0	0	$-2^{3i} - 2^{3i+1} + 2^{3i+1} + 2^{3i+1} + 2^{3i+2}$	1	1	1	1	1

By minimizing using a 5-variable ($s_i, m1_i, m2_i, m3_i, m4_i$) Karnaugh map, the aggregate of the static bias, the dynamic bias and the constant $-\lfloor n/3 \rfloor - 5$ that negates the inclusion of a constant by the $(\lfloor n/3 \rfloor + 6)$ -operand (MOMA, 2^n+1) is given by

$$\left\lfloor \sum_{i=0}^{\lfloor n/3 \rfloor} K_i \right\rfloor_{2^n+1} = \sum_{i=0}^{\lfloor n/3 \rfloor} KS_i + \sum_{i=0}^{\lfloor n/3 \rfloor} KD_i - \lfloor n/3 \rfloor - 5 \quad (5.30)$$

$$= K1 + K2 + K3$$

where

$$\begin{aligned} K1 &= \sum_{i=0}^{\lfloor n/3 \rfloor} (\overline{m2_i \vee m4_i}) \cdot 2^{3i} + (\overline{m3_i \vee m4_i}) \cdot 2^{3i+1} \\ K2 &= \sum_{i=0}^{\lfloor n/3 \rfloor} ((m2_i \vee m4_i) \wedge s_i) \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} ((m3_i \vee m4_i) \wedge s_i) \cdot 2^{3i+2} \\ K3 &= \sum_{i=1}^{\lfloor n/3 \rfloor} 1 \cdot 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} \overline{s_i} \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} s_i \cdot 2^{3i+2} \end{aligned} \quad (5.31)$$

From (5.29) and (5.31), the aggregate bias for the three moduli, 2^n-1 , 2^n and 2^n+1 , is given by only three n -bit words as follows.

$$K1 = \begin{cases} \sum_{i=0}^{\lfloor n/3 \rfloor} 0 \cdot 2^{3i} + 0 \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 0 \cdot 2^{3i+2} & \text{if } m = 2^n - 1 \\ \sum_{i=0}^{\lfloor n/3 \rfloor} s_i \cdot 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} 0 \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 0 \cdot 2^{3i+2} & \text{if } m = 2^n \\ \sum_{i=0}^{\lfloor n/3 \rfloor} (\overline{m2_i \vee m4_i}) \cdot 2^{3i} + \overline{m3_i \vee m4_i} \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 0 \cdot 2^{3i+2} & \text{if } m = 2^n + 1 \end{cases} \quad (5.32)$$

$$K2 = \begin{cases} \sum_{i=0}^{\lfloor n/3 \rfloor} 0 \cdot 2^{3i} + 0 \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 0 \cdot 2^{3i+2} & \text{if } m = 2^n - 1 \text{ or } 2^n \\ \sum_{i=0}^{\lfloor n/3 \rfloor} 0 \cdot 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} ((m2_i \vee m4_i) \wedge s_i) \cdot 2^{3i+1} & \text{if } m = 2^n + 1 \\ + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} ((m3_i \vee m4_i) \wedge s_i) \cdot 2^{3i+2} & \end{cases} \quad (5.33)$$

$$K3 = \begin{cases} \sum_{i=0}^{\lfloor n/3 \rfloor} 0 \cdot 2^{3i} + 0 \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 0 \cdot 2^{3i+2} & \text{if } m = 2^n - 1 \text{ or } 2^n \\ \sum_{i=1}^{\lfloor n/3 \rfloor} 1 \cdot 2^{3i} + \sum_{i=0}^{\lfloor n/3 \rfloor} \overline{s_i} \cdot 2^{3i+1} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} s_i \cdot 2^{3i+2} & \text{if } m = 2^n + 1 \end{cases} \quad (5.34)$$

$K1$ can be generated using 3:1 multiplexers at $3i$ bit positions along with 2:1 multiplexers at $3i+1$ and $3i+2$ bit positions. Both $K2$ and $K3$ can be generated by 2:1 multiplexers at all bit positions.

5.4.4 Multi-modulus partial product accumulation

By replacing $\left\lfloor \sum_{i=0}^{\lfloor n/3 \rfloor} K_i \right\rfloor_m$ in (5.25) with $K1, K2$ and $K3$ from (5.32) – (5.34), the radix-2³ Booth encoded multi-modulus multiplication is given by (5.35). Two n -bit all zeros partial products are included for the moduli 2^n-1 and 2^n so that the number partial products to be accumulated is $\lfloor n/3 \rfloor + 6$ for all three moduli.

$$|Z|_m = \begin{cases} \left\lfloor \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + K1 + K2 + K3 + 0 + 0 \right\rfloor_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ \left\lfloor \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + K1 + K2 + K3 + X + Y \right\rfloor_m & \text{if } m = 2^n + 1 \end{cases} \quad (5.35)$$

The CSA tree based accumulation of $\lfloor n/3 \rfloor + 6$ in a multi-modulus MOMA using MUX3 blocks in the carry feedback path is illustrated in Fig. 5.7 for $n = 4$.

Fig. 5.7 Multi-modulus accumulation of partial products

5.5 Performance analysis

In this section, the performances of the proposed radix-2² and radix-2³ Booth encoded multi-modulus multipliers are evaluated. For comparison, the following four configurations of $\{2^n-1, 2^n, 2^n+1\}$ based RNS multipliers are considered. In Type A RNS multiplier, all three modulo multipliers are radix-2³ Booth encoded. Type A RNS multiplier consists of modulo 2^n-1 and modulo 2^n+1 multipliers described in Chapter 4 [Mura11c] and modulo 2^n multiplier based on Section 5.4. While in Types B – D RNS multipliers, all three modulo multipliers are radix-2² Booth encoded. In the Types B – D RNS multipliers, the multiplier of [Efst04a] for the modulus 2^n-1 and the multiplier based on Section 5.3 for the modulus 2^n are employed. The multipliers of [Mura11b], [Chen10] and [Sous05] are employed for the modulus 2^n+1 in Type B, Type C and Type D, respectively.

The multipliers were specified in VHDL and synthesized by Synopsys Design Compiler tool (version D-2010.03) under nominal synthesis environment, *i.e.*, 25°C and 1.8V based on TSMC 0.18μm 1.8V CMOS standard-cell library. The designs were optimized for area and delay independently. The average dynamic power dissipation is also estimated based on the Monte Carlo method with 99.9% confidence that the error is bounded below 3%. The minimum achievable area from area-constrained optimization, the least achievable critical path delay from delay-constrained optimization, the total power dissipation (sum of both dynamic and leakage power dissipations) for the proposed radix-2^k Booth encoded multi-modulus multipliers are tabulated in Table 5.9 when $k = 2$ and $k = 3$.

Table 5.9 Area, delay and total power dissipation of proposed radix-2^k Booth encoded multi-modulus multipliers

n	Area (μm ²)		Delay (ns)		Total power dissipation (mW)	
	$k = 3$	$k = 2$	$k = 3$	$k = 2$	$k = 3$	$k = 2$
32	86396	81034	4.33	4.07	0.833	0.927
36	110170	100553	4.76	4.43	1.097	1.160
40	128661	122301	4.56	4.37	1.290	1.413
44	147964	145539	4.86	4.31	1.519	1.684
48	177263	171552	4.87	4.50	1.743	2.001
52	200172	199703	5.06	4.85	2.130	2.147
56	224545	229934	4.96	4.80	2.293	2.333
60	260280	262273	5.06	4.97	2.532	2.522
64	288814	296861	5.14	4.91	2.494	2.826

The areas, delays and total power dissipations of the individual radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers that comprise Types A – D RNS multipliers are tabulated in Tables 5.10, 5.11 and 5.12, respectively.

Table 5.10 Area of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers

n	Area (μm^2)							
	Mod 2^n-1		Mod 2^n		Mod 2^n+1			
	$k=3$	$k=2$	$k=3$	$k=2$	$k=3$	$k=2$		
	[Mura11c]	[Efst04a]			[Mura11c]	[Mura11b]	[Chen10]	[Sous05]
32	61535	63920	61854	65253	77724	74032	85411	152152
36	80871	80465	80299	81972	99565	92104	106824	190739
40	95796	99060	95397	100740	116783	112285	130767	233822
44	112059	119570	111833	121423	135294	134393	157285	281273
48	136462	142130	136409	144156	162288	158602	186132	333218
52	155608	166645	155728	168844	183727	184821	217420	389395
56	176122	193050	176415	195422	206569	212982	251070	450068
60	206383	221371	206848	223916	239580	243113	287114	515119
64	229791	251745	230426	254459	266421	275346	325894	584648

Table 5.11 Delay of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers

n	Delay (ns)							
	Mod 2^n-1		Mod 2^n		Mod 2^n+1			
	$k=3$	$k=2$	$k=3$	$k=2$	$k=3$	$k=2$		
	[Mura11c]	[Efst04a]			[Mura11c]	[Mura11b]	[Chen10]	[Sous05]
32	3.55	3.31	2.98	2.90	3.9	3.48	3.14	3.48
36	3.70	3.54	3.21	3.04	4.19	4.12	3.92	3.80
40	4.17	4.04	3.18	3.35	4.25	4.02	3.84	4.15
44	4.14	4.01	3.34	3.47	4.61	4.05	4.02	4.18
48	4.30	4.00	3.38	3.43	4.62	4.26	3.75	4.17
52	4.28	4.09	3.4	3.60	4.59	4.73	3.95	4.51
56	4.39	4.17	3.71	4.02	4.77	4.78	4.48	5.05
60	4.68	4.69	4.03	3.97	4.98	4.87	4.42	5.01
64	4.67	4.74	4.05	4.10	4.85	4.77	4.62	4.78

Table 5.12 Total power dissipation of radix- 2^k Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers

n	Power dissipation (mW)							
	Mod 2^n-1		Mod 2^n		Mod 2^n+1			
	$k=3$	$k=2$	$k=3$	$k=2$	$k=3$	$k=2$		
	[Mura11c]	[Efst04a]			[Mura11c]	[Mura11b]	[Chen10]	[Sous05]
32	0.690	0.725	0.588	0.585	0.843	0.812	1.211	1.544
36	0.899	0.909	0.753	0.725	1.077	1.102	1.649	2.012
40	1.061	1.115	0.890	0.903	1.270	1.357	2.087	2.481
44	1.242	1.343	1.054	1.074	1.452	1.607	2.525	3.005
48	1.474	1.611	1.232	1.418	1.716	1.923	2.870	3.550
52	1.715	2.021	1.415	1.611	2.100	2.159	3.390	4.151
56	1.858	2.231	1.523	1.726	2.268	2.441	3.665	4.654
60	2.048	2.422	1.789	1.810	2.470	2.515	4.058	4.885
64	2.141	2.615	1.879	1.882	2.610	2.690	4.264	5.088

In Table 5.13, the areas of Types A – D RNS multipliers are computed as the sum of the areas of the constituent modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers from Table 5.10. Similarly, the total power dissipations of Types A – D RNS multipliers are computed from Table 5.12 and tabulated in Table 5.13. Whereas in Table 5.13, the critical path delays of Types A – D RNS multipliers are determined as the maximum value amongst the delays of the constituent modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers from Table 5.11.

Table 5.13 Area, delay and total power dissipation of radix- 2^k Booth encoded $\{2^n-1, 2^n, 2^n+1\}$ based RNS multipliers

n	Area (μm^2)				Delay (ns)				Total power dissipation (mW)			
	$k=3$		$k=2$		$k=3$		$k=2$		$k=3$		$k=2$	
	Type A	Type B	Type C	Type D	Type A	Type B	Type C	Type D	Type A	Type B	Type C	Type D
32	201113	203205	214584	281325	3.9	3.48	3.31	3.48	2.122	2.123	2.522	2.855
36	260735	254541	269261	353176	4.19	4.12	3.92	3.8	2.730	2.737	3.285	3.648
40	307976	312085	330567	433622	4.25	4.04	4.04	4.15	3.222	3.377	4.106	4.500
44	359186	375386	398278	522266	4.61	4.05	4.02	4.18	3.750	4.026	4.943	5.424
48	435159	444888	472418	619504	4.62	4.26	4.00	4.17	4.423	4.952	5.899	6.579
52	495063	520310	552909	724884	4.59	4.73	4.09	4.51	5.231	5.791	7.023	7.784
56	559106	601454	639542	838540	4.77	4.78	4.48	5.05	5.649	6.399	7.623	8.611
60	652811	688400	732401	960406	4.98	4.87	4.69	5.01	6.308	6.748	8.291	9.118
64	726638	781550	832098	1090852	4.85	4.77	4.74	4.78	6.631	7.189	8.763	9.587

From Table 5.9 and 5.13, the percentage savings in area and total power dissipation and the percentage increase in delay of the proposed radix- 2^3 Booth encoded multi-modulus multiplier over Type A RNS multiplier and of the proposed radix- 2^2 Booth encoded multi-modulus multiplier over Types B – D RNS multipliers are tabulated in Table 5.14.

Table 5.14 Percentage savings in area, delay and total power dissipation of proposed multi-modulus multipliers over RNS multipliers

n	% savings in area over				% savings in delay over				% savings in total power dissipation over			
	$k=3$		$k=2$		$k=3$		$k=2$		$k=3$		$k=2$	
	Type A	Type B	Type C	Type D	Type A	Type B	Type C	Type D	Type A	Type B	Type C	Type D
32	57.04	60.12	62.23	71.19	-11.02	-16.95	-22.96	-16.95	60.72	56.31	63.23	67.51
36	57.74	60.49	62.65	71.52	-13.60	-7.52	-13.01	-16.57	59.80	57.60	64.66	68.18
40	58.22	60.81	63.00	71.79	-7.29	-8.16	-8.16	-5.3	59.95	58.13	65.57	68.58
44	58.80	61.22	63.45	72.13	-5.42	-6.41	-7.21	-3.11	59.47	58.17	65.93	68.95
48	59.26	61.43	63.68	72.30	-5.41	-5.63	-12.5	-7.91	60.59	59.59	66.07	69.58
52	59.56	61.61	63.88	72.45	-10.23	-2.53	-18.58	-7.53	59.27	62.92	69.42	72.41
56	59.83	61.77	64.04	72.57	-3.98	-0.41	-7.14	4.95	59.40	63.53	69.39	72.90
60	60.12	61.90	64.18	72.69	-1.6	-2.05	-5.97	0.79	59.85	62.61	69.57	72.33
64	60.25	62.01	64.32	72.78	-5.97	-2.93	-3.58	-2.71	62.38	60.69	67.75	70.52

From Table 5.14, by replacing the three individual modulo multipliers in the $\{2^n-1, 2^n, 2^n+1\}$ based RNS multiplier by the proposed multi-modulus multiplier, an average reduction of 60% in area and total power dissipation can be observed. Specifically, the proposed radix-2³ Booth encoded multi-modulus multiplier achieves a maximum area and power dissipation savings of 60% over the radix-2³ Booth encoded Type A RNS multiplier. Similarly the proposed radix-2² Booth encoded multi-modulus multiplier reduces the area and total power dissipation of the radix-2² Booth encoded Types B – D RNS multipliers by 72% maximally. However the proposed radix-2³ Booth encoded multi-modulus multiplier increases the critical path delay of Type A RNS multiplier by no more than 13%. The maximum delay penalties of the proposed radix-2² Booth encoded multi-modulus multiplier over Types B – D RNS multipliers are 16%, 22% and 16%, respectively.

5.6 Summary

Multi-modulus architectures for radix-2² and radix-2³ Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications were explored. To exploit hardware reuse amongst the three modulo channels, new modulo 2^n and modulo 2^n+1 multipliers architectures were proposed. The proposed multi-modulus multipliers were compared against RNS multipliers employing Booth encoding technique of the same radix. From synthesis results based on CMOS implementations, the proposed radix-2² Booth encoded multi-modulus multiplier achieves maximal area and power dissipation savings of 72% with a maximum delay penalty of 22%. Moreover, the proposed radix-2³ Booth encoded multi-modulus multiplier lowers the area as well as power dissipation by 60% maximally with a marginal delay increase of 13% in the worst case.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

As RNS is often adopted to implement repetitive multiplications of cryptographic and signal processing algorithms, the research emphasis has shifted markedly to the area-power efficient implementation of concurrent modulo multiplications in recent years. In particular, full-combinatorial modulo multiplier implementations by exploiting the number theoretic properties of special moduli 2^n-1 , 2^n and 2^n+1 are predominant in RNS research. While high-radix Booth encoding algorithm has been demonstrated to reduce area and power consumption in two's complement multiplier design, there are significant challenges in gaining comparable advantages from Booth encoding in modulo multiplications. Specifically, in radix-8 Booth encoded modulo multiplication, the non-trivial computation of the hard multiple is widely acknowledged as the bottleneck operation. For this reason, the radix of Booth encoding has been limited to four in existing modulo multiplier designs.

The work focused on designing modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multipliers using radix-4 as well as radix-8 Booth encoding algorithms. The key arithmetic operations in Booth encoded modulo multiplication, *i.e.*, modulo negation, modulo reduction of binary weight, modulo multiplications by positive and negative power-of-two terms were uniformly reformulated for the three moduli 2^n-1 , 2^n and 2^n+1 . Novel solutions to generate the hard multiple of radix-8 Booth encoded modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications were proposed. In both radix-4 and radix-8 Booth encoded modulo multiplier designs, techniques to generate the inherent bias with minimal hardware overhead were emphasized.

In short, the contributions of this thesis are summarized as follows.

The first-ever radix-8 Booth encoded modulo 2^n-1 multiplier for application in the non-critical modulo channel of RNSs was proposed. A technique to generate the modulo-reduced hard multiple in a partially-redundant and biased representation using RCAs was suggested. The RCA word-length was varied to match the non-critical modulo 2^n-1 multiplier delay to the critical modulo m multiplier delay of the RNS. By intentionally operating the non-critical modulo 2^n-1 multiplier at a slower speed, significant savings in system area and system power dissipation was observed without compromising the system speed. The proposed delay-adaptive multiplier design technique was shown to be most advantageous in imbalanced word-length moduli set based RNSs where the timing slack between the non-critical modulo 2^n-1 and critical modulo m channels is in large excess.

As the delay-adaptive radix-8 Booth encoded modulo 2^n-1 multiplier design is applicable only in the non-critical channel of imbalanced word-length moduli sets, a radically different approach to modulo-reduced hard multiple generation was proposed. A custom adder, *i.e.*, HMG that exclusively computes the modulo 2^n-1 and modulo 2^n+1 reduced hard multiples of the radix-8 Booth encoding algorithm was designed. Both the modulo 2^n-1 and modulo 2^n+1 HMGs were the most area-delay-power efficient adders for this application. New radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers using the proposed HMGs for application in both critical and non-critical channels as well as balanced and imbalanced word-length moduli sets were developed. In the $\{2^n-1, 2^n, 2^n+1\}$ based RNS multiplier, the proposed radix-8 Booth encoded modulo 2^n-1 and modulo 2^n+1 multipliers outperformed radix-4 Booth encoded as well as non-encoded modulo 2^n-1 and modulo 2^n+1 multipliers.

Due to the similarity amongst *CLS*, *LS* and *CCLS* operations as well as EAC, carry-ignore CEAC additions, moduli 2^n-1 , 2^n , 2^n+1 multiplier architectures are prime candidates for resource sharing. A new radix-4 Booth encoded modulo 2^n+1 multiplier with hardware complexity similar to the radix-4 Booth encoded modulo 2^n-1 multiplier was designed. In addition, modulo 2^n multipliers using radix-4 and radix-8 Booth encoding algorithms were proposed for the first time and used as the base architectures for the unification of modulo multiplication in the popular three moduli set $\{2^n-1, 2^n, 2^n+1\}$ based RNS. With the proposed radix-4 and radix-8 Booth encoded modulo multipliers as the basis, multi-modulus multiplier architectures that perform the modulo 2^n-1 , modulo 2^n and modulo 2^n+1 multiplications in serial were developed. As three parallel modulo 2^n-1 , modulo 2^n and modulo 2^n+1

multipliers were replaced by a single multi-modulus multiplier, the savings in area and total power dissipation of 60% were feasible theoretically.

6.2 Future work

Based on the research presented in this thesis, relevant topics for further exploration have been identified.

1. The number of partial products to be generated and accumulated in a modulo multiplier is $\lfloor n/k \rfloor + 1$ and it diminishes inversely with the radix 2^k of Booth encoding algorithm. Thus, the design of multipliers employing Booth encoding with radix greater than eight is an alluring option. But on closer examination, it is observed that the number of hard multiples increases to $2^{k-1} - 2$. For example, in radix- 2^4 Booth encoding technique, the $2^{4-1} - 2 = 6$ hard multiples are $+3X$, $+5X$, $+7X$, $-3X$, $-5X$ and $-7X$, where X is the multiplicand. Both $+6X$ and $-6X$ are not considered as distinct hard multiples as they can be computed by shifting the hard multiples $+3X$ and $-3X$, respectively. The computations of the hard multiples require modulo adders that fall on the critical path of the multiplier. Similar to the HMGs presented in Chapters 4 and 5, fast adders for the generation of other hard multiples can be designed by making use of the properties of modulo arithmetic. Furthermore, common computations amongst the generation of various hard multiples can be isolated and performed in a shared unit.
2. A hybrid radix-4/radix-8 Booth encoded modulo multiplier design is a potential research direction. In this technique, a part of the multiplier bits is encoded using the radix-4 Booth encoding algorithm while the remainder of the multiplier bits is encoded using the radix-8 Booth encoding algorithm [Cher97]. The generation and accumulation of the modulo-reduced radix-4 Booth encoded partial products are performed while the hard multiple is computed in a fast adder. The optimal number of multiplier bits under each Booth encoding scheme depends on the delay of hard multiple generation.

3. Alternatively, the multiplier bits encoded consistently using the same radix- 2^k ($k > 2$) Booth encoding can be partitioned into two groups. The hard multiples corresponding to the first group can be expressed redundantly as pairs or triplets of simple multiples. Similar to the hybrid design, the partial products corresponding to the former group are generated and accumulated simultaneously with the computation of the hard multiple. For example, in a radix-8 Booth encoded modulo 2^n-1 multiplier, the n multiplier bits can be partitioned into two groups of $n/2$ bits. The hard multiple is expressed as the redundant pair $(1X, 2X)$ in the former group and as the calculated sum $3X$ in the latter group. The trade-off between the multiplier bit group length and the area-delay-power performance of the modulo multiplier can be investigated.

4. Modulo squaring is a special modulo multiplication operation where the multiplicand and the multiplier are identical. By manipulating the redundancy in the partial product matrix, efficient squarer circuits have been proposed in [Verg05], [Mura09a] where the operand bits are non-encoded and [Baka11] where both the multiplicand and multiplier bits are radix-4 Booth encoded. The high radix Booth encoding techniques suggested in this thesis can be extended to modulo 2^n-1 , modulo 2^n and modulo 2^n+1 squarer circuits for improved area-delay-power metrics. Modulo squaring is also the backbone of modulo exponentiation. As modulo exponentiation can be performed by decomposing it into a series of modulo squaring operations, there is more room for optimization by unifying the modulo exponentiations for special moduli in a multi-modulus architecture, which can be very useful for the implementation of cryptographic algorithms.

Author's Publications

Journal publications

[Mura11a] R. Muralidharan and C. H. Chang, "Radix-8 Booth encoded modulo 2^n-1 multipliers with adaptive delay for high dynamic range Residue Number System," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 58, no. 5, pp. 982 – 993, May. 2011.

[Mura11c] R. Muralidharan and C. H. Chang, "Area-power efficient modulo 2^n-1 and modulo 2^n+1 multipliers for $\{2^n-1, 2^n, 2^n+1\}$ based RNS," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, accepted for publication, Dec. 2011.

[Mura12] R. Muralidharan and C. H. Chang, "Radix- 2^2 and radix- 2^3 Booth encoded multi-modulus multipliers for moduli 2^n-1 , 2^n and 2^n+1 ," Submitted to *IEEE Transactions on Circuits and Systems – I: Regular Papers*, Jul. 2012.

Conference publications

[Satz08] R. K. Satzoda, R. Muralidharan and C. H. Chang, "Programmable LSB-first and MSB-first modular multipliers for ECC in $GF(2^m)$," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Seattle, WA, pp. 808 – 811, May 2008.

[Mura08] R. Muralidharan, C. H. Chang and C. C. Jong, "A low complexity modulo 2^n+1 squarer design," in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, Macau, pp. 1296 – 1299, Nov. 2008.

[Mura09a] R. Muralidharan and C. H. Chang, "Fixed and variable multi-modulus squarer architectures for triple moduli base of RNS," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Taipei, Taiwan, pp. 441 – 444, May 2009.

[Mura09b] R. Muralidharan and C. H. Chang, "Hard multiple generator for higher radix modulo 2^n-1 multipliers," in *Proceedings of 12th IEEE International Symposium on Integrated Circuits*, Singapore, pp. 546 – 549, Dec. 2009.

[Mura10] R. Muralidharan and C. H. Chang, "Fast hard multiple generators for radix-8 Booth encoded modulo 2^n-1 and 2^n+1 multipliers," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Paris, France, pp. 717 – 720, May 2010.

[Mura11b] R. Muralidharan and C. H. Chang, "A simple radix-4 Booth encoded modulo 2^n+1 multiplier," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Rio de Janeiro, Brazil, pp. 1163 – 1166, May 2011.

References

- [Alia91] G. Alia and E. Martinelli, “A VLSI modulo m multiplier,” *IEEE Transactions on Computers*, vol. 40, no. 7, pp. 873 – 878, Jul. 1991.
- [Andr88] S. Andraos and H. Ahmad, “A new efficient memoryless residue to binary converter,” *IEEE Transactions on Circuits and Systems*, vol. 35, no. 11, pp. 1441 – 1444, Nov. 1988.
- [Baja04] J. C. Bajard and L. Imbert, “A full RNS implementation of RSA,” *IEEE Transactions on Computers – Brief Contributions*, vol. 53, no. 6, pp. 769 – 774, Jun. 2004.
- [Baka11] D. Bakalis, H. T. Vergos and A. Spyrou, “Efficient modulo $2^n \pm 1$ squarers,” *Integration, the VLSI Journal*, vol. 44, no. 3, pp. 163 – 174, Jun. 2011.
- [Bayo87] M. Bayoumi, G. Jullien and W. Miller, “A VLSI implementation of residue adders,” *IEEE Transactions on Circuits and Systems*, vol. 34, no. 3, pp. 284 – 288, Mar. 1987.
- [Bewi94] G. W. Bewick, “Fast multiplication: algorithms and implementation,” Ph.D. thesis, Stanford University, Feb. 1994.
- [Bhar98] M. Bhardwaj, A. B. Premkumar and T. Srikanthan, “Breaking the $2n$ -bit carry propagation barrier in residue to binary conversion for the $\{2^n-1, 2^n, 2^n+1\}$ moduli set,” *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 45, no. 9, pp. 998 – 1002, Sep. 1998.
- [Bhar99] M. Bhardwaj, T. Srikanthan and C. T. Clarke, “A reverse converter for the 4-moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$,” *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, pp. 168 – 175, Apr. 1999.
- [Boot51] A. D. Booth, “A signed binary multiplication technique,” *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236 – 240, 1951.

- [Bren82] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260 – 264, Mar. 1982.
- [Burc93] R. Burch, F. N. Najm, P. Yang, and T. N. Trick, "A Monte Carlo approach for power estimation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 1, pp. 63-71, Mar. 1993.
- [Cao03] B. Cao, C. H. Chang and T. Srikanthan, "An efficient reverse converter for the 4-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ based on the New Chinese Remainder Theorem," *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 50, no. 10, pp. 1296 – 1303, Oct. 2003.
- [Cao05] B. Cao, T. Srikanthan and C. H. Chang, "Efficient reverse converters for four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$ " *IEE Computers and Digital Techniques*, vol. 152, no. 5, pp. 687 – 696, Sep. 2005
- [Cao07] B. Cao, C. H. Chang and T. Srikanthan, "A residue-to-binary converter for a new five-moduli set," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 54, no. 5, pp. 1041 – 1049, May 2007.
- [Card02] G. C. Cardarilli, A. Del Re, A. Nannarelli, M. Re, "Residue number system reconfigurable datapath," *Proceedings of IEEE International Symposium on Circuits and Systems*, Phoenix-Scottsdale, AZ, vol. 2, pp. 756 – 759, May 2002.
- [Card05] G. C. Cardarilli, A. Del Re, A. Nannarelli, M. Re, "Low power and low leakage implementation of RNS FIR filters," *Proceedings of 39th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, pp. 1620 – 1624, Nov. 2005.
- [Chen10] J. W. Chen and R. H. Yao, "Efficient modulo 2^n+1 multipliers for diminished-1 representation," *IET Circuits, Devices and Systems*, vol. 4, no. 4, pp. 291 – 300, Jul. 2010.
- [Cher97] B. S. Cherkauer and E. G. Friedman, "A hybrid radix-4/radix-8 low power signed multiplier architecture," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 44, no. 8, pp. 656-659, Aug. 1997.

- [Dhur98] A. Dhurkadas, "Comments on 'A high speed realization of a residue to binary number system converter'," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 45, no. 3, pp. 446 – 447, Mar. 1998.
- [Dadd65] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349 – 356, 1965.
- [Dimi03] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos and C. Efstathiou, "A family of parallel-prefix modulo 2^n-1 adders," *Proceedings of International Conference on Application-specific Systems, Architectures and Processors*, The Hague, The Netherlands, pp. 326 – 336, Jun. 2003.
- [Dimi05a] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI Ling adders," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 225 – 231, Feb. 2005.
- [Dimi05b] G. Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos and C. Efstathiou, "New architectures for modulo 2^n-1 adders," in *Proceedings of 12th IEEE International Conference on Electronics, Circuits and Systems*, Gammarth, Tunisia, pp. 1 – 4, Dec. 2005.
- [Dugd92] M. Dugdale, "VLSI implementation of residue adders based on binary adders," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 325 – 329, May 1992.
- [Dugd94] M. Dugdale, "Residue multipliers using factored decomposition," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 41, no. 9, pp. 623 – 627, Sep. 1994.
- [Efst94] C. Efstathiou, D. Nikolos and J. Kalamatiaos, "Area-time efficient modulo 2^n-1 adder design," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 41, no. 7, pp. 463 – 467, Jul. 1994.
- [Efst04a] C. Efstathiou, H. T. Vergos and D. Nikolos, "Modified Booth modulo 2^n-1 multiplier," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 370 – 374, Mar. 2004.

- [Efst04b] C. Efstathiou, H. T. Vergos and D. Nikolos, “Fast parallel-prefix modulo 2^n+1 adders,” *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1211 – 1216, Sep. 2004.
- [Efst05] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos and D. Nikolos, “Efficient Diminished-1 modulo 2^n+1 multipliers,” *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 491 – 496, Apr. 2005.
- [Flyn01] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*, John Wiley & Sons, Inc., 2001.
- [Han87] T. Han and D. A. Carlson, “Fast area-efficient VLSI adders,” *Proceedings of 8th Symposium on Computer Arithmetic*, Como, Italy, pp. 49 – 56, May 1987.
- [Harr03] D. Harris, “A taxonomy of parallel prefix networks,” *Proceedings of 37th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, vol. 2, pp. 2213 – 2217, Nov. 2003.
- [Hias92] A. Hiasat, “New memoryless mod $(2^n\pm 1)$ residue multiplier,” *Electronics Letters*, vol. 28, no. 3, pp. 314 – 315, Jan. 1992.
- [Hias98] A. A. Hiasat and H. S. Abdel-Aty-Zohdy, “Residue-to-binary arithmetic converter for the moduli set $(2^k, 2^k-1, 2^{k-1}-1)$,” *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 45, no. 2, pp. 204 – 209, Feb. 1998.
- [Hias00] A. A. Hiasat, “New efficient structure for a modular multiplier for RNS,” *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 170 – 174, Feb. 2000.
- [Hias02] A. A. Hiasat, “High-speed and reduced-area modular adder structures for RNS,” *IEEE Transactions on Computers*, vol. 51, no. 1, pp. 84 – 89, Jan. 2002.
- [Jenk92] W. K. Jenkins and B. A. Schnaufer, “Fault tolerant architectures for efficient realization of common DSP kernels,” *Proceedings of 35th Midwest Symposium on Circuits and Systems*, Washington, DC, vol. 2, pp. 1320 – 1323, Aug. 1992.

- [Jenk93a] W. K. Jenkins and A. J. Mansen, "Variable word length DSP using serial-by-modulus residue arithmetic," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, vol. 3, pp. 89 – 92, Apr. 1993.
- [Jenk93b] W. K. Jenkins, B. A. Schnaufer and A. J. Mansen, "Combined system-level redundancy and modular arithmetic for fault tolerant digital signal processing," *Proceedings of 11th Symposium on Computer Arithmetic*, Windsor, Canada, pp. 28 – 35, Jul. 1993.
- [Jull80] G. A. Jullien, "Implementation of multiplication, modulo a prime number with applications to number theoretic transforms," *IEEE Transactions on Computers*, vol. C-29, no. 10, pp. 899 – 905, Oct. 1980.
- [Kala00] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos and J. Kalamatianos, "High-speed parallel-prefix modulo (2^n-1) adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673 – 680, Jul. 2000.
- [Kang06] J. Y. Kang and J. L. Gaudiot, "A simple high-speed multiplier design," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1253 – 1258, Oct. 2006.
- [Know01] S. Knowles, "A family of adders," *Proceedings of 15th IEEE Symposium on Computer Arithmetic*, Vail, CO, pp. 277 – 281, Jun. 2001.
- [Kogg73] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence relations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786 – 793, Aug. 1973.
- [Kour10] I. Koretas and V. Paliouras, "RNS multi-voltage low power multiply-add unit," *Proceedings of 17th IEEE International Conference on Electronics, Circuits and Systems*, Athens, Greece, pp. 9 – 12, Dec. 2010.
- [Ladn80] R. E. Ladner and M. J. Fischer, "Parallel prefix computation" *Journal of the ACM*, vol. 27, no. 4, pp. 831 – 838, Oct. 1980.

- [Leib76] L. Leibowitz, "A simplified binary arithmetic for the Fermat number transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, no. 5, pp. 356 – 359, Oct. 1976.
- [Ling81] H. Ling, "High-speed binary adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156 – 166, Mar. 1981.
- [Ma98] Y. Ma, "A simplified architecture for modulo (2^n+1) multiplication," *IEEE Transactions on Computers*, vol. 47, no. 3, pp. 333 – 337, Mar. 1998.
- [Macs61] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proceedings of IRE*, vol. 49, no. 1, pp. 67 – 91, Jan. 1961.
- [Math00] J. Mathew, D. Radhakrishnan and T. Srikanthan, "Fast residue-to-binary converter architectures," *Proceedings of 42nd Midwest Symposium on Circuits and Systems*, Las Cruces, NM, vol.2, pp. 1090 – 1093, Aug. 1999.
- [Moha07a] P. V. A. Mohan, "Reverse converters for a new moduli set $\{2^{2n}-1, 2^n, 2^{2n}+1\}$," *Circuits, Systems and Signal Processing*, vol. 26, no. 2, pp. 215-227, Apr. 2007
- [Moha07b] P. V. A. Mohan and A. B. Premkumar, "RNS-to-binary converters for two four-moduli sets $\{2^n-1, 2^n, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^{n+1}+1\}$," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 54, no. 6, pp. 1245 – 1254, Jun. 2007.
- [Moha07c] P. V. A. Mohan, "RNS-to-binary converter for a new three-moduli set $\{2^{n+1}-1, 2^n, 2^n-1\}$," *IEEE Transactions on Circuits and Systems – II: Express Briefs*, vol. 54, no. 9, pp. 775 – 779, Sep. 2007.
- [Mola10] A. S. Molahosseini, K. Navi, C. Dadkhah, O. Kavehei and S. Timarchi "Efficient reverse converter designs for the new 4-moduli sets $\{2^n-1, 2^n, 2^{n+1}-1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ based on New CRTs," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 57, no. 4, pp. 823 – 835, Apr. 2010.

- [Mont85] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519 – 521, Apr. 1985.
- [Naga90] M. Nagamatsu, S. Tanaka, J. Mori, K. Hirano, T. Noguci and K. Hatanaka, "A 15-ns 32x32-b CMOS multiplier with an improved parallel structure," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 494 – 497, Apr. 1990.
- [Nage96] C. Nagendra, M. J. Irwin, R. M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 689 – 702, Oct. 1996.
- [Noza01] H. Nozaki, M. Motoyama, A. Shimbo and S. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery algorithm," *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Paris, France, pp. 364 – 376, May 2001.
- [Oklo96] 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 Transactions on Computers*, vol. 45, no. 3, pp. 294 – 306, Mar. 1996.
- [Pali99] V. Paliouras and T. Stouraitis, "Multi-function architectures for RNS processors," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 46, no. 8, pp. 1041 – 1054, Aug. 1999.
- [Parh94] B. Parhami and C.Y. Hung, "Optimal table lookup schemes for VLSI implementation of input/output conversions and other residue number operations," *Proceedings of VII Workshop on VLSI Signal Processing*, La Jolla, CA, pp. 470 – 481, Oct. 1994.
- [Pies91] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *Proceedings of 10th IEEE Symposium on Computer Arithmetic*, Grenoble, France, pp. 100 – 107, Jun. 1991.
- [Pies94] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Transactions on Computers*, vol. 423, no. 1, pp. 68 – 77, Jan. 1994.

- [Pies95] S. J. Piestrak, "A high-speed realization of a residue to binary number system converter," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 42, no. 10, pp. 661 – 663, Oct. 1995.
- [Pies02] S. J. Piestrak, "Design of squarers modular A with low-level pipelining," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 49, no. 1, pp. 31 – 41, Jan. 2002.
- [Prem02] A. B. Premkumar, "A formal framework for conversion from binary to residue numbers," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 49, no. 2, pp. 135 – 144, Feb. 2002.
- [Prem06] A. B. Premkumar, E. L. Ang and E. M. K. Lai, "Improved memoryless RNS forward converters based on the periodicity of residues," *IEEE Transactions on Circuits and System – II: Express Briefs*, vol. 53, no. 2, pp. 133 – 137, Feb. 2006.
- [Raba03] J. M. Rabaey, A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits*, Prentice Hall, 2003.
- [Radh92] D. Radhakrishnan and Y. Yuan, "Novel approaches to the design of RNS multipliers," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 39, no. 1, pp. 52 – 57, Jan. 1981.
- [Satz06] R. K. Satzoda, C. H. Chang and T. Srikanthan, "Monte Carlo statistical analysis for dynamic power simulation of RTL designs using Synopsys Power Compiler," *Synopsys Users' Group Conference*, Singapore, Jun. 2006. Online publication: <http://snug-universal.org/papers/papers.htm>.
- [Schi09] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas and T. Stouraitis, "An RNS implementation of an F_p Elliptic curve point multiplier," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 56, no. 6, pp. 1202 – 1213, Jun. 2009.

- [Skl60] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, vol. EC-9, no. 2, pp. 226 – 231, Jun. 1960.
- [Sode80] M. A. Soderstrand and C. Vernia, "A high-speed low-cost modulo P_i multiplier with RNS arithmetic applications," *Proceedings of IEEE*, vol. 68, no. 4, pp. 529 – 532, Apr. 1980.
- [Sode86] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, 1986.
- [Song91] P. J. Song and G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1184 – 1198, Sep. 1991.
- [Sous05] L. Sousa and R. Chaves, "A universal architecture for designing efficient modulo 2^n+1 multipliers," *IEEE Transactions on Circuits and Systems – I: Regular Papers*, vol. 52, no. 6, pp. 1166 – 1178, Jun. 2005.
- [Stel98] P. F. Stelling, C. U. Martel, V. G. Oklobdzija and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Transactions on Computers*, vol. 47, no. 3, pp. 273 – 285, Mar. 1998.
- [Stou01] T. Stouraitis and V. Paliouras, "Considering the alternatives in low-power design," *IEEE Circuits and Devices Magazine*, vol. 17, no. 4, pp. 22 – 29, Jul. 2001.
- [Szab67] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill, 1967.
- [Tayl81] F. Taylor, "Large moduli multipliers for signal processing," *IEEE Transactions on Circuits and Systems*, vol. 28, no.7, pp. 731 – 736, Jul. 1981.
- [TSMC03] *TSMC 0.18 μ m Process 1.8Volt Sage-X™ Standard Cell Library Databook*, Artisan Components Inc., Sept. 2003.

- [Verg02] H. T. Vergos, C. Efstathiou and D. Nikolos, “Diminished-one modulo 2^n+1 adder design,” *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1389 – 1399, Dec. 2002.
- [Verg05] H. T. Vergos and C. Efstathiou, “Diminished-1 modulo 2^n+1 squarer design,” *IEE Computers and Digital Techniques*, vol. 152, no. 5, pp. 561 – 566, Sep. 2005.
- [Verg07] H. T. Vergos and C. Efstathiou, “Design of efficient modulo 2^n+1 multipliers,” *IET Computers and Digital Techniques*, vol. 1, no. 1, pp. 49 – 57, Jan. 2007.
- [Verg08] H. T. Vergos and C. Efstathiou, “A unifying approach for weighted and diminished-1 modulo 2^n+1 addition,” *IEEE Transactions on Circuits and Systems – II: Express Briefs*, vol. 55, no. 10, pp. 1041 – 1045, Oct. 2008.
- [Verg09] H. T. Vergos and C. Efstathiou, “Efficient modulo 2^n+1 adder architectures,” *Integration, the VLSI Journal*, vol. 42, no. 2, pp. 149 – 157, Feb. 2009.
- [Verg10] H. T. Vergos, D. Bakalis and C. Efstathiou, “Fast modulo 2^n+1 multi-operand adders and residue generators,” *Integration, the VLSI Journal*, vol. 43, no. 1, pp. 42 – 48, Jan. 2010.
- [Vino00] A. P. Vinod and A. B. Premkumar, “A memoryless reverse converter for the 4-moduli superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$,” *Journal of Circuits, Systems, and Computers*, vol. 10, no. 1&2, pp. 85 – 99, 2000.
- [Wall64] C. S. Wallace, “A suggestion for a fast multiplier,” *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14 – 17, Feb. 1964.
- [Wang96a] Z. Wang, G. A. Jullien and W. C. Miller, “An algorithm for multiplication modulo (2^n-1) ,” in *Proceedings of 39th IEEE Midwest Symposium on Circuits and Systems*, Ames, IA, pp. 1301–1304, Aug. 1996.
- [Wang96b] Z. Wang, G. A. Jullien and W. C. Miller, “An efficient tree architecture for modulo 2^n+1 multiplication,” *Journal of VLSI Signal Processing*, vol. 14, no. 3, pp. 241–248, Dec. 1996.

- [Wang98] Y. Wang, "New Chinese Remainder Theorems," *Proceedings of 32nd Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, vol. 1, pp. 165 – 171, Nov. 1998.
- [Wang00] Y. Wang, "Residue-to-binary converters based on new Chinese remainder theorems," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 47, no. 3, pp. 197 – 205, Mar. 2000.
- [Wang02] Y. Wang, X. Song, M. Aboulhamid and H. Shen, "Adder based residue to binary number converters for $(2^n-1, 2^n, 2^n+1)$," *IEEE Transactions on Signal Processing*, vol. 50, no. 7, pp. 1772 – 1779, Jul. 2002.
- [Wein81] A. Weinberger, "4:2 Carry-save adder module," *IBM Technical Disclosure Bulletin*, vol. 23, Jan. 1981.
- [Wrzy93] A. Wrzyszc and D. Milford, "A new modulo 2^a+1 multiplier," *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, MA, pp. 614 – 617, Oct. 1993.
- [Yeh00] W. C. Yeh and C. W. Jen, "High-speed Booth encoded parallel multiplier design," *IEEE Transactions on Computers*, vol. 49, no.7, pp. 692 – 701, Jul. 2000.
- [Zimm99] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n\pm 1)$ addition and multiplication," *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, pp. 158 – 167, Apr. 1999.