VLSI Efficient Architectures for Triple Moduli Based RNS Computations

Cao Bin

School of Computer Engineering

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

2006

Acknowledgements

I wish to thank my supervisor, Professor Thambipillai Srikanthan, for his supervision and academic motivation throughout this research programme. In particular, I wish to thank him for teaching me some of the philosophical aspects of research and some intangible skills, which I consider are the most invaluable knowledge I have acquired in this research programme.

I would also like to thank Associate Professor Chang Chip Hong, for providing me the research suggestions, for discussions during the development of the algorithms and the organization of the large amount of the experimental results, and for spending much of his valuable time to correct and improve the manuscripts of the published papers and this thesis.

I would like to thank Dr. Wu Jigang, Mr. Shubi Menon, Dr. Gu Jiangmin, Ms Ye Zhi and Mr. Satzoda Ravi Kumar, for their friendship, help, discussions, suggestions and collaboration pertaining to the work in this research programme.

To the staff of Centre for High Performance Embedded Systems, I wish to convey my appreciation to all of them for their kind and friendly assistance.

I would like to express my gratitude to my parents for their support and encourage. I am grateful to my wife, Zheng Zhen （郑榛）, for her love and unwavering support throughout this research programme. This dissertation is specially dedicated to her.

Last but not least, I wish to thank God, our heavenly Father, the Creator of heaven and earth, for daily providence.

Summary

The inherent parallel computations of the residue number system (RNS) arithmetic have been shown to accelerate the computations of certain class of applications. The triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ has emerged as a prerogative choice of RNS computations due to its VLSI efficient properties. In an attempt to increase the cardinality without incurring unacceptable hardware complexity, a number of moduli sets have been proposed in the literature. It is typical for contributions to examine the various computation units of an RNS in isolation and constraints such as power, delay and area measures have not been examined in an integrated manner.

In this research work, we have made a thorough evaluation of existing contributions on the reverse converters based on the triple moduli and proposed new reverse converters and architectures. In addition, we have proposed reverse converters for new moduli sets in an attempt to realize VLSI efficient alternatives. In particular, detailed studies on the power, delay and area measures of the reverse converters for four different special moduli sets based on the triple moduli have been undertaken. These include two existing moduli sets and two new moduli sets with cardinality of 4 and 5 respectively. Improvements to the existing reverse converters were made through a new multi-level reverse conversion algorithm. These reverse converters have been fully implemented and simulated at the register-transfer level running on commercial synthesis tools using a 0.35µm CMOS standard cell library.

Based on the observation that the most critical residue channel in the RNS's residue arithmetic unit (RAU) is associated with the $2^n + 1$ modulus, a general structure for the diminished-one modulo $2^n + 1$ Multi-Operand Modulo Adder (MOMA) has been developed. Since the proposed MOMA has the same structure as that of modulo $2^n - 1$, the performance of the $2^n + 1$ channel is comparable to that of modulo $2^n - 1$ channel.

A generic inner product RNS processor that can be programmed to support a predefined number of modular operations has been employed to evaluate the overall performance of the RNS processor with forward converter, RAUs and the reverse

converter. The proposed MOMA has also been employed in the forward converter to improve efficiency. Detailed evaluations on the performance of this generic RNS processor have been undertaken for five moduli sets based on the triple moduli.

Extensive simulations to compute power, delay and area measures for RNS with a predefined dynamic range and input data width have been carried out to establish the relationship among the degree of parallelism, power consumption and area. Our evaluations affirm that higher-cardinality RNS processor is superior despite the increased complexity of the corresponding reverse converters. For example, for a 32-bit inner product processor with 32 8-bit input signals, the five-moduli set based RNS achieves 31.1% and 12.5% reduction in area cost and dynamic power dissipation, and performs 14.1% faster than the triple-moduli set RNS. It is also evident that the impact of the reverse conversion overhead of these special moduli sets rapidly diminishes with the increase in the number of iterative computations within the RNS computation unit. Finally, the proposed integrated environment for evaluating the RNS processor has provided for a holistic approach to examine the implications that the cardinality and types of moduli sets have on power, delay and area measures.

CONTENTS

List of Figures

List of Tables

List of Abbreviations

CLA	Carry Look-ahead Adder
CMOS	Complementary Metal-Oxide-Semiconductor
CEAC	Complement End-Around Carry
CRT	Chinese Remainder Theorem
CPA	Carry Propagate Adder
CSA	Carry Save Adder
DSP	Digital Signal Processing
EAC	End-Around Carry
EDA	Electronic Design Automation
FA	Full Adder
HA	Half Adder
LSB	Least Significant Bit
LUT	Lookup Table
MAC	Multiply-Accumulate
MOMA	Multi-Operand Modular Addition/Adder
MRC	Mixed Radix Conversion
MSB	Most Significant Bit
RAU	Residue Arithmetic Unit
RCA	Ripple Carry Adder
RNS	Residue Number System
ROM	Read-Only Memory
RTL	Register Transfer Level
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
XOR	Exclusive OR

List of Symbols

$\{P_1, P_2, \ldots, P_n\}$	Moduli set with moduli $P_1, P_2, \ldots, P_n$.
$(x_1, x_2, x_3, \ldots, x_n)$	Residue representation where x_i, $i = 1, 2, \ldots, n$ is the residue.
$\lvert X \rvert_P$	Remainder of X with respect to the modulus P
$\lvert 1/M \rvert_P$ or $\left\lvert \frac{1}{M} \right\rvert_P$	Multiplicative inverse of M with respect to P
$CLS_p(x, r)$	Circular left shift of the p-bit binary number, x by r bits.
$CCLS_p(x, r)$	Complementary circular left shift of the p-bit binary number, x by r bits.
$\overline{X}$	1's complement of a bit, X or a number, X.
$(ABCD)_2$	Binary representation where A, B, C, D are binary digits.
$(ABCD)_{16}$	Hexadecimal representation where A, B, C, D are Hexadecimal digits.
$\oplus$	XOR.
$\lceil \ \rceil$	Ceiling operator
$\lfloor \ \rfloor$	Floor operator.
$d(A)$	Diminished-one expression of an integer, A.
$(abcd)^{(k)}$	A string of k repetitions of the binary string, $abcd$.
$gcd(m, p)$	Greatest common divisor of m and p.

Chapter 1

Introduction

1.1 Motivation

Residue Number System (RNS) is a non-weighted number system, whereby large binary numbers can be decomposed into smaller residues in such a way that there is no carry propagation across different residue channels. In other words, operations such as addition, subtraction and multiplication can be performed on each residue channel concurrently and independently [Sza67] [Sod86]. The inherent parallelism coupled with reduced bit-width operations implies the potential enhancement on the speed of computations, and the amenability to other VLSI design attributes. The trend in digital signal processing and general computing has grown towards greater memory capacity, higher resolution and higher workload in order to deliver the desired performance, scalability and reliability for transaction-heavy applications. Therefore, technology and industry are regularly poised for a broad transition to double the bus-width from the present computer architecture. Cycle time for effective migration path to the next higher-bit computing has been continuously shortened and it is foreseen that computer arithmetics of more than 100 bits will soon prevail [Parh00] [Wal02]. The inter-channel carry free characteristic of residue arithmetic becomes very attractive then as it eliminates the complex VLSI interconnections associated with large operand arithmetic by the localization of the interconnects to within each residue channel. Therefore, there has been a rekindle of interest in the use of RNS for VLSI arithmetic circuits since its inception in 1950's.

Another useful feature provided by RNS-based computation is the property of fault tolerance. Since no communication is needed among residue channels in RNS, if an error occurs in one residue channel during an arithmetic operation, it will not be propagated into other residue channels. This is the basis for fault tolerant computing

that is inherent in RNS. In a redundant RNS [Sod86], if an error occurs in one residue channel, the erroneous residue can be discarded without affecting the final result provided that there is enough dynamic range in the reduced system to contain the result. This type of flexibility is not simply available in the weighted number systems, where the loss of some digits will annihilate the result.

Though initially articulated for general purpose computer arithmetic, the advantages of RNS for such general purpose systems were proven to be obscure because of the difficulties associated with sign detection, magnitude comparison and general division. Research into more specialized computational applications has spawned such applications like RNS-based Digital Signal Processing (DSP) [Bar80] [Huan81] [Mill84] [Tayl90] [Kal01] and applications based on the fault tolerance property of RNS [Man72] [Sod86] [Kri92]. The use of RNS has also been found recently in applications of image processing [Rej00] [Ram00] and cryptography [Zimm94] [Noz01] [Imb02] [Yen03].

Although RNS offers many advantages over the weighted number system, it is still not widely employed in applications where it is feasible such as high performance DSPs and fault-tolerant systems. The main obstacles to the widespread use of RNS are the extra overheads, such as the forward conversion of the input data, the reverse conversion of the output data, and the residue arithmetic operations, which are more complex than the corresponding binary arithmetic operations. The need for conversion into and out of RNS arises because most of the digital systems and their associated peripherals are designed based on the weighted number system. The decomposition of a binary number into its residues is called the *forward conversion.* Conversely, the composition of the residues back to a weighted number is called the *reverse conversion.* Thus, a complete RNS system includes a forward converter, a reverse converter and a Residue Arithmetic Unit (RAU) comprising the required modular operations in the residue domain. Figure 1.1 shows the block diagram of a typical RNS system.

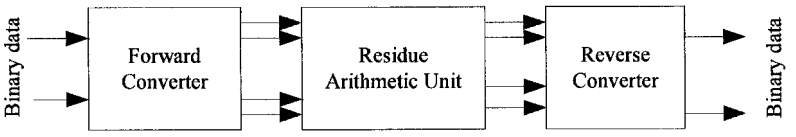


Figure 1.1 *Architecture of Residue Number System (RNS)*

Among these overheads, the forward conversion is the least severe one because it can be performed concurrently, and sometimes it can be eliminated depending on the input word length. The reverse conversion requires considerably more silicon area, delay and power consumption than the forward conversion. For many years, the Chinese Remainder Theorem (CRT) and the Mixed Radix Conversion (MRC) have been used for solving the reverse conversion problem, and only until recently, the New CRTs proposed by Wang [Wang98] [Wang00] are used for this purpose. In applications where RNS has significant advantages, the arithmetic operations in the RAU are performed in parallel and iteratively, and the RAU takes a significant portion of the hardware cost, delay, and power consumption of the entire RNS. The efficiency of a RAU is largely dependent on the moduli set chosen for the RNS. As the modular operations for special moduli are more efficient than those for the general format moduli, special moduli sets have been used more extensively than the general moduli sets to reduce the hardware complexities in the implementation of the forward converters, reverse converters, and RAUs.

Among the special moduli, the moduli of $2^n \pm 1$ type (i.e., expressible in the form 2^n or $2^n \pm 1$) are more attractive due to their efficiency for the design of not only the forward and reverse converters, but also the RAUs. The triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ was the first successful special moduli set introduced. It has gained unprecedented popularity by virtue of its inherent number theoretic properties in the CRT algorithm for the reverse converter and for its efficient implementations of RAUs. The cardinality of the triple-moduli set does not provide sufficient headroom to meet the performance for some high-dynamic range computations, and for fault-tolerant applications. Higher cardinality RNS's based on special four-moduli and five-moduli sets have been introduced [Ska98] [Bha99] [Ska99b] [Vin00]. Some of the

moduli of these RNS's are not of the $2^n \pm 1$ type, and are therefore not efficient for the RAUs even if they possess efficient reverse converter architectures. Without loss of generality, throughout this thesis, we refer to the moduli set, $\{2^n - 1, 2^n, 2^n + 1\}$ as the triple moduli set and those special moduli sets that are composed of only moduli of the forms 2^n and $2^n \pm 1$, irrespective of the number of moduli, the triple moduli based RNS.

The number of moduli used in a RNS dictates the degree of parallelism that can be achieved. In addition, using more moduli for a fixed dynamic application implies smaller wordlength for every residue channel; hence better speedup can be gained. On the other hand, higher cardinality leads to increased complexity of the reverse converter. As the residue channels are running in parallel, the latency or the worst case delay is determined by the slowest residue channel. Therefore, the balance of the channel wordlength of a moduli set is another important criterion to speed up the RNS as a whole. Although many special moduli set RNS's have been proposed, the relative VLSI performance metrics such as hardware cost, delay, and power consumption for different special moduli sets with different dynamic ranges remain highly obscure today, owing to the conflicting desiderata of the reverse conversion and the RAU for different cardinality moduli sets, or even moduli sets of the same cardinality.

Given a myriad of moduli sets at the disposal of the computer architects, they would prefer to use the best moduli set for a given application; but what actually defines the "best" moduli set? Is it the fastest, or the smallest, or the most fault-tolerant or the one that consumes the least power? It is envisaged that the answer to this question will not be definitive according to the arguments of intricate trade-offs mentioned above. According to Mead and Conway [Mead79], the greatest challenge that VLSI presents to computer science is that of developing a theory of computation that accommodates a more general model of costs involved in computing. Therefore, the research work of this thesis is inspired by the desire to provide an insight into the design tradeoffs that can reduce power consumption, minimize cost and enhance performance for various high-cardinality triple moduli based RNS's introduced and studied in this thesis. The intention is not to simply rank different RNS's according to a fixed criterion, but to evaluate the characteristics of different moduli sets with a holistic approach. Besides

aiding a designer in selecting an appropriate moduli set with favorable VLSI characteristics for his application, this research is motivated by the potential exploitation of emerging theorems to spawn efficient reverse conversion algorithms and modular arithmetic modules for some special moduli sets, especially for those based on the celebrated triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. We are also aspired to explore new moduli sets which can lead to good reverse converter architectures than the existing RNS's of the same cardinality. In view of the relatively sparse number of high-cardinality special moduli sets, we believe that there is a promising research opportunity for original high-cardinality special moduli sets with good number theoretic properties that make them efficient for both the reverse converters as well as the RAUs.

1.2 Research Objectives

The objective of this thesis is to explore the VLSI efficient architectures for realizing the critical building blocks of RNS, which is based on triple-moduli set or their extended supersets. The ultimate aim is to provide an insight into the design tradeoffs of different special moduli sets at different dynamic ranges, taking into account the overall consideration of the power, delay and area constraints for the entire RNS built around those supersets so defined. The research involves the proposal of new special moduli sets, and exploiting their number theoretic properties to develop novel algorithms for more efficient reverse converters. The study will also include exploring better design of reverse converters and several obligatory components of the RAU in the critical channel of the RNS's of interest. VLSI metrics in terms of area, delay and power consumption of these RNS's at different dynamic ranges will also be analyzed and evaluated.

The three integral parts of the RNS, namely the forward converter, the reverse converter and the RAU will first be explored independently for different moduli sets. Special emphasis will be placed on the reverse conversion algorithms and high cardinality moduli sets as reverse converters are far more complex than the forward converters, especially when the number of moduli is high. Given the main objective, the following scope has been covered in this thesis.

- Proposal of new efficient reverse converters for existing and new triple moduli based four-moduli supersets.
- Proposal of reverse conversion algorithm for new high-cardinality triple moduli based supersets. Examining the area-time-power metrics of their reverse converters.
- Evaluation and analysis of the implication of cardinality, dynamic ranges on area, delay and power dissipation performances of the reverse converters.
- Exploration of unified structure to support modular operations of the critical channel of the triple-moduli set based RNS's.
- Evaluation and analysis of the relative merits of the complete RNS's for different triple moduli based supersets under varying dynamic ranges and arithmetic operations.

It is hoped that based on the results of our research, a larger design space is formulated from which a designer can choose the right architectures and special moduli sets with the desired characteristics constrained by his application requirements.

1.3 Major Contributions of the Thesis

A number of contributions have been made to allow interesting implementations in terms of tradeoffs between area, delay and power consumption for the triple moduli based RNS's. We have proposed a new reverse conversion algorithm to devise the area-power efficient reverse converters for two akin four-moduli sets. Two new special moduli set RNS's along with their VLSI efficient reverse converters are also proposed. In addition, we have developed a general architecture for the diminished-one Multi-Operand Modular Addition (MOMA), which can be used in modular arithmetic circuit for the RAU and in the forward converters. Most importantly, a detailed evaluation of VLSI performance metrics of various RNS's has been undertaken. We describe these contributions individually as follows.

Firstly, we propose a new method to devise the reverse conversion algorithms for two existing four-moduli sets. These two akin moduli sets are $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$

and $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, and they are valid for even value of n. These moduli sets feature the important property of balanced residue channel widths and maximal utilization of dynamic range. Exploiting the fact that they are extensions of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, our new method possesses a two-level hierarchy. In the first level, we adopt the reverse converter for the triple-moduli set, and in the second level, we use MRC for the resultant two-moduli set $\{2^n(2^{2n} - 1), 2^{n \pm 1} - 1\}$ to obtain the final result. One advantage of the new method is that we can use the efficient structures for the reverse converters for the triple-moduli set which are available in the literature. For these four-moduli sets, computational complexity analysis and simulation results have revealed that our proposed reverse converters are faster, consumed lower power and occupied smaller area than the existing reverse converters.

Secondly, we propose a new reverse converter for the four-moduli set, $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ with proof of its relative primality. An elegant formulation of its reverse conversion algorithm based on the New CRT [Wang00] has been derived. The new moduli set is also an extension of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. Due to the special property of this moduli set and the use of New CRT, the proposed reverse converter is as efficient as that of the triple-moduli set. Even though it is not balanced, a rigorous performance comparison presented in Chapter 7 of this thesis shows that the proposed reverse converter is not only as area efficient as that for the triple-moduli set, but the overall performance in terms of delay and power consumption, is superior to that of the triple-moduli set. The reasons for its transcendent performance over the celebrated triple-moduli set are due to its small-sized residue channels and increased parallelism for the RAU at no loss of efficiency in its reverse converter.

In addition, we propose a new five-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, and its reverse converter. The moduli are all of $2^n \pm 1$ type, and co-prime when n is even. Comparing with some existing five-moduli sets where the moduli are not all of $2^n \pm 1$ type, our proposed moduli set is more efficient for the modular operations of the RAU. The advantages are evidently translated into a more efficient RNS. The reverse conversion algorithm is based the on the work of the first contribution mentioned above. The algorithm also comprises two levels. In the first level, the

reverse converter for the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ is adopted, and in the second level, MRC is used for the two-moduli set $\{2^n(2^{2n} - 1)(2^n - 1), 2^{n-1} - 1\}$.

The fourth contribution involves a critical component of the RAU, the modulo $2^n + 1$ MOMA. The modulo $2^n + 1$ modular arithmetic operations are often manipulated using the diminished-1 representation as it helps to reduce the width of the operands to only n bits. Moreover, some operations such as negation, multiplications by powers of 2 can be performed readily. However, the general structure of MOMA is not available in the literature. In this thesis, we propose a generalized structure for the diminished-one modulo $2^n + 1$ MOMA. The proposed structure is similar to the structure of modulo $2^n - 1$ MOMA, which consists of a modular Wallace tree. It exhibits a more regular VLSI structure than its binary counterpart.

The last but the most significant contribution involves the comprehensive analysis and evaluation of VLSI measures for RNS's. Altogether 5 special moduli sets based on the triple moduli have been chosen for optimal VLSI implementation of their forward and reverse converters. A generic RNS structure of an inner product step processor with varying number of operations is proposed to adapt to different dynamic ranges. For the first time a systematic approach to analyze the complete VLSI performances in terms of area, delay and power consumption based on synthesis results has been carried out for the RNS's built around the triple moduli based supersets. The breakeven points of area, power and delay for different superset RNS's in terms of the number of modular operations have been established. Our evaluation shows that the reverse converter constitutes a small fraction of the hardware cost and power consumption for dedicated DSP applications. We conclude that with highly efficient modular operations, higher-cardinality RNS benefits significantly from the increased parallelism despite the increased complexity of the reverse converter. This is the fundamental reason for the exploration of higher cardinality RNS.

1.4 Organization of the Thesis

This thesis is arranged into eight chapters. In Chapter 1, we presented the motivations for our research work, the objectives, followed by the major contributions of this

thesis.

In Chapter 2, we provide a literature review on RNS, which includes the residue number representation, residue arithmetic, the forward converter, and the basic computational units in RNS. We will review the basic computational units in RNS, such as binary adders, Carry Save Adders (CSA), and the diminished-one systems which are usually applied to the critical modulo $2^n + 1$ operations. We also review the reverse converters based on the traditional CRT, MRC and the emerging New CRTs that promise great potential for the discovery of new reverse conversion algorithms. A survey of the reverse converters for special moduli sets, particularly the moduli sets of $2^n \pm 1$ type will also be provided in this chapter. In our review of the residue arithmetic operations, we present the fundamentals of the modular adders, MOMAs and modular multipliers. Due to the importance of special moduli sets of $2^n \pm 1$ type, special attention has been paid to the time critical modulo $2^n + 1$ arithmetic operations.

In Chapter 3, we propose a new reverse conversion method for two akin 4-moduli sets, $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, which are both extensions of the triple-moduli set. Both moduli sets are valid for even value of n. The new reverse conversion method is based on a two-level decomposition of the moduli sets. In the first level, a reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is adopted. The second level uses MRC for the resulting two-moduli sets, $\{2^n(2^{2n} - 1), 2^{n+1} - 1\}$ and $\{2^n(2^{2n} - 1), 2^{n-1} - 1\}$ corresponding to the two supersets of interest. We show that the closed-form expressions of the MRC coefficients can be calculated by solving two modular equations. The reverse converters for these two moduli sets are very similar in structure and performance. For a given dynamic range, one of them will supplement the other as an alternative to minimize the wastage of dynamic range.

In Chapter 4, we propose a new four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, with a dynamic range of $5n$ bits. Comparing with other four-moduli sets, this moduli set is co-prime for any value of the integer n, and has a larger dynamic range. Therefore, a smaller value of n is needed for a fixed dynamic range. Using the New CRT, we derive the closed-form expressions for the multiplicative inverses required for the reverse converter. Due to the special form of the fourth modulus, the reverse

converter only consists of four CSAs with end-around carry (EAC) and one $4n$-bit 1's complement adder, which is very similar to the reverse converter for the triple-moduli set. The performances in terms of area and time complexity of the proposed reverse converters are evaluated and compared with the reverse converters of two other four-moduli sets towards the end of the chapter.

In Chapter 5, we propose a new five-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, with even value of n, where the moduli are all of $2^n \pm 1$ type. Comparing to some other five-moduli sets where the moduli are not all of $2^n \pm 1$ type, this five-moduli set is more efficient for the forward converters and the RAU due to the fact that modulo $2^n \pm 1$ operations are more efficient than the modular operations for general moduli. The reverse conversion algorithm is based on a two-level method. In the first level, the reverse converter of the moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ is adopted, and in the second level, the MRC for the two-moduli set $\{2^n(2^{2n} - 1)(2^n - 1), 2^{n-1} - 1\}$ is used to obtain the final result.

Chapter 6 focuses on the essential modulo operations used in the RAUs. We propose a general structure for the diminished-one MOMA. MOMA is an important component of the modular multiplier, and it can also be used separately in the forward converter and as embedded core in some cryptographic processing. The proposed architecture consists of a Wallace tree of universal CSAs with complement EAC (CEAC) followed by a 2-input diminished-one carry propagate adder (CPA). The notion of auxiliary ports is introduced to form a network of companion residue counters to calculate the constant residues introduced intrinsically by the CSA with CEAC tree. With elegant derivation, we prove that the companion residue counter network is redundant as the final constant residue of the MOMA is fixed and can be absorbed into the final diminished-one CPA. The hardware cost and delay of this highly regular Wallace tree architecture approach those of the modulo $2^n - 1$ CSA tree with EAC.

In Chapter 7, we investigate the VLSI performances of RNS's based on the supersets build around the triple-moduli $\{2^n - 1, 2^n, 2^n + 1\}$ in a holistic manner. The evaluation includes the study of the forward converters and the reverse converters independently,

as well as the complete RNS for an inner product processor of each moduli set. We present the synthesis and simulation results based on the CMOS 0.35μm technology standard cell library for five RNS's with different number of modular operations. For more information about the synthesis tools and technology libraries, please refer to Appendix. The five RNS's investigated are the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, four-moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$, and the five-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$. VLSI measures such as delay, area and power for the forward/reverse converters and the total RNS against the dynamic ranges for different special moduli sets are compared and analyzed. Based on the simulation results, an appraisal of the relative merits of different cardinality moduli sets is provided.

In Chapter 8, we present our conclusions, identify potential future work and make recommendations for further research.

Chapter 2

Background and Literature Review

Although residue arithmetic has been applied successfully to modern DSP applications, it was founded on number theory of great antiquity. The roots of the residue arithmetic date back to the first century A. D., when the earliest recorded work in this field was performed by the Chinese mathematician Sun-Tsu. The puzzle, "what number has the remainder of 2, 3 and 2 when divided by the numbers, 7, 5 and 3, respectively?", was written in the form of a verse around first century A.D. [Gros66] [Tayl84]. The theoretical foundations of RNS theory were developed in the eighteenth and nineteenth centuries by Euler, Fermat, Gauss, and other notable mathematicians. An early attempt to apply RNS to physical problem was made by D. H. Lehmer [Sod86], and from 1950's there was a flurry of research activities about the RNS in the United States. Due to the characteristic of RNS, its applications in general purpose computers were soon reckoned with limited prospect, and many researchers began to investigate its use in more specialized computational applications.

By the time of mid-1960's, digital signal processing was beginning to emerge as a technical subject distinct from the general digital computing. During this era, the properties offered by RNS were exploited and applied in the digital signal processing applications where additions, subtractions and multiplications are the most computational intensive arithmetic operations. The VLSI era arrived in the 1970's, along with some new problems. It was found that the traditional algorithms based on weighted number systems for developing digital signal processing applications suffer from the curse of dimensionality due to the carry chain. It did not exhibit sufficient "parallelism" and "modularity" to fully utilize the advantages of VLSI technology to optimize essential hardware attributes like area, delay, power consumption and interconnection complexity. It should be noted that RNS can provide many good

features to mitigate the problems due to its non-weighted and modular structures.

The aim of this chapter is to provide both background information and literature survey of specific topics of RNS research. It is organized as follows: Section 2.1 reviews the RNS, including the basic concepts of residue number representation, the residue arithmetic, and the forward conversion, followed by the basic computational units within RNS. Section 2.2 gives detailed discussion on reverse conversion, followed by reverse converter architectures for the special moduli sets. Section 2.3 discusses the RAUs, including the modular adders, modular multipliers, and MOMAs.

2.1 Residue Number Systems

2.1.1 Residue Number Representation

The root of residue number system is number theory. The fundamental support of number theory is the numeral system. An integer number system is defined as a numeral system that is governed by a set of rules so that a finite number of integers can be represented based on a positional notation and their arithmetic operations can be performed without ambiguity. A number system is said to be weighted if there exists a set of weights w_i such that for any number x in the system, x can be expressed in the following form:

$$x = \sum_{i=1}^{n} a_i w_i \tag{2-1}$$

where a_i is a digit at the i-th position and it is represented by a symbol from the permissible symbol sets. If the values for w_i are successive powers of the same number, then the number system has a *fixed base* or *fixed radix*. For example, radix 10 with the symbol set {0, 1, 2, …, 9} for the decimal system, and radix 2 for the binary system with two symbols {0, 1}. Number systems in which the weights are not powers of the same radix are called *mixed radix systems*.

Weighted number systems are important because they generally permit convenient magnitude comparison of two numbers by the simple operation on the digits in the

corresponding positions. The two most useful number systems in computer and DSP applications are binary system and decimal number system. The two systems have some important advantages such as: simple relative-magnitude comparison, multiplication or division by a power of the radix can be done by left or right shifting, ease of extending the range of the number system, and ease of overflow detection. However, all the arithmetic operations in both the binary and the decimal number systems cannot be easily parallelized because of the propagation of carries [Sza67].

The residue number system is a non-weighted number system. Although it may not have some of the advantages of the binary or decimal system, it can offer hardware implementation advantages. Addition, subtraction and multiplication operations can be performed in parallel within small wordlength, non-communicating channels, i.e., the operations on each residue channel are independent of the other residue channels.

Unlike the fixed radix number system, the base of RNS consists of an n-tuple of integers, $P_1, P_2, \ldots, P_n$, where each integer is called a modulus and they are pairwise relatively prime. For any given base, which is also called the moduli set, the residue number representation of an integer X is an n-tuple of smaller integers, $(x_1, x_2, \ldots, x_n)$, where x_i is given by

$$x_i = X \mod P_i, \quad i = 1, \ 2, \ \ldots \ , \ n. \tag{2-2}$$

Equation (2-2) can also be expressed as $x_i = |X|_{P_i}$. The product of all moduli, M is called the dynamic range. It is the number of representable values in the RNS given by the moduli set. For convenience, we define the dynamic range, M in terms of $\lfloor \log_2 M + 0.5 \rfloor$ bits, where the function $\lfloor \ \rfloor$ is a floor operation.

Definition 2.1: If $0 < a < m$, and $|a \cdot b|_m = 1$, then a is called the multiplicative inverse of b mod m, and is denoted by $a = |1/b|_m$ or $a = \left|b^{-1}\right|_m$.

The number X can be evaluated from the n-tuple $(x_1, x_2, \ldots, x_n)$ using the Chinese Remainder Theorem (CRT), which is stated as follows:

Theorem 2.1 (CRT): In an RNS with moduli set $\{P_1, P_2, \ldots, P_n\}$, if the residue representation of an integer X $(0 \le X < M)$ is $(x_1, x_2, \ldots, x_n)$, then the weighted number X can be evaluated by the following formula [Sza67]:

$$X = \left| \sum_{i=1}^{n} \left| \frac{1}{\hat{M}_i} \right|_{P_i} \cdot x_i \cdot \hat{M}_i \right|_M \tag{2-3}$$

where $M = \prod_{i=1}^{n} P_i$, and $\hat{M}_i = M/P_i$, $\left|1/\hat{M}_i\right|_{P_i}$ is the multiplicative inverse of $\hat{M}_i$ mod P_i.

2.1.2 Residue Arithmetic

Residue arithmetic is also known as modular arithmetic. Modular addition, modular subtraction and modular multiplication are widely used in RNS-based applications, cryptography, and digital signal processing applications. Modular arithmetic used in RNS is usually referred to as residue arithmetic because each channel is a residue channel. Although the residue arithmetic for general modulus is more complex than the binary arithmetic, it can be performed very efficiently if proper modulus is selected, for example, the modulus of the $2^n \pm 1$ type. Modular squaring [Cao05b] is a special case of modular multiplication. Modular exponentiation can be performed by Montgomery algorithm [Mont85]. Some important axioms of residue arithmetic are summarized below. These properties will be used extensively in this thesis.

Property 2.1: $|Km|_m = 0$, for any integer K.

Property 2.2: $|x \pm Km|_m = |x|_m$, for any integer K.

Property 2.3: $|Km + 1|_m = 1$, for any integer K.

Property 2.4: $|x \pm y|_m = \left| |x|_m \pm |y|_m \right|_m$.

Property 2.5: $|x \cdot y|_m = \left| |x|_m |y|_m \right|_m$.

Property 1.6: $|-x|_m = |(m-1)x|_m$.

Property 2.7: $|Kx|_{Km} = K|x|_m$, for any integer K ($\neq 0$).

Property 2.8: Given two co-prime integers K and m, if $|Ka|_m = |Kb|_m$, then $|a|_m = |b|_m$.

Property 2.9: If p is a prime number, then $|a^p|_p = |a|_p$.

Property 2.10: Multiplying a positive integer, x by 2^r modulo $2^n - 1$ can be accomplished by expressing x in an n-bit binary representation and then shifting it circularly by r bits to the left.

Property 2.11: Multiplying a negative integer, x by 2^r modulo $2^n - 1$ can be accomplished by expressing the one's complement of x in an n-bit binary representation and then shifting it circularly by r bits to the left.

Property 2.12: Arithmetic operation $\Diamond$, such as addition, subtraction, and multiplication in RNS can be performed in parallel over the residues, i.e., if $(x_1, x_2, \ldots, x_n) \Diamond (y_1, y_2, \ldots, y_n) = (z_1, z_2, \ldots, z_n)$, then $z_i = |x_i \Diamond y_i|_{P_i}$.

Property 2.13: $\left|2^{ta}\right|_{2^a-1} = 1$, where t is a nonnegative integer, and the nature number $a > 1$.

Property 2.10 is originally proposed in [Sza67]. As an example, let $x = 6$, $n = 5$ and $r = 3$, then we have

$$\left|x \cdot 2^r\right|_{2^n-1} = CLS_5(6, 3) = CLS_5((00110)_2, 3) = (10001)_2 = 17$$

where the function $CLS_p(x, r)$ is used to denote a circular shift of the p-bit binary number x by r bits to the left. An example of *Property 2.11* is shown as follows. Let $x = -6$, $n = 5$ and $r = 3$, then we have

$$\left|x \cdot 2^r\right|_{2^n-1} = CLS_5(-6, 3) = CLS_5((\overline{00110})_2, 3) = (01110)_2 = 14$$

where $\overline{b}$ indicates the complement of the binary variable, b.

Properties 2.10 and *2.11* are very important because computationally intensive multiplication can be replaced by circular shift. *Property 2.12* shows that the commonly used arithmetic operations are performed on the residues in RNS, and they can be executed simultaneously in the residue channels. *Property 2.13* can be useful

for modulo $2^n - 1$ arithmetic. For a useful algebraic system, the reciprocal of modular multiplication is needed and it is defined by the multiplicative inverse.

For the multiplicative inverse, we have the following property:

Property 2.14: The multiplicative inverse $|1/b|_m$ exists and is unique if and only if $gcd(b, m) = 1$.

2.1.3 Forward Conversion

As binary system is the prevailing number system in most digital systems, efficient methods to convert the data from the binary domain to the residue form and vice versa have been studied. The forward conversion problem is stated as follows: Given an integer X, and the base of the RNS, i.e., the moduli set $P = \{P_1, P_2, \ldots, P_n,\}$, calculate the residues $(x_1, x_2, \ldots, x_n)$ according to Equation (2-2).

Forward conversion can be performed in parallel, because the residue of the i-th residue channel depends only on its moduli P_i and the input binary integer X. Forward conversion can be treated as a problem of multioperand modular addition with different parts of X or terms related to X as operands. There are three methods to implement forward converters. The first method is based on lookup tables, where all possible values required by the conversion are precomputed and stored in read-only memory (ROM) [Jen77] [Parh94]. The second method is to use arithmetic circuits to perform the forward conversion [Alia84] [Capo88] [Pou94a] [Pou94b] [Prem02]. The third is hybrid method, where both arithmetic circuits and small-sized ROMs are involved. The first method will be efficient only for small moduli, because the table size would increase exponentially with the wordlength of the moduli. The complexity of the second method depends on the moduli set. Some new architectures can compute residues for more than one moduli either serially or in parallel by sharing common intermediate results among residue channels, which would increase the resource utilization of the RNS [Pali99]. Some of the forward converters are based on the periodicity of $|2^n|_m$, and the forward conversion can be performed by multioperand modular adder using carry save adders according to this periodic property [Pie94a].

The forward converters for general moduli set are somewhat more complex due to the lack of the useful number theoretic properties of the moduli set. When the modulus has special form, such as that of $2^n \pm 1$, the forward conversion can be greatly simplified due to the special number properties exhibited by the modulo $2^n \pm 1$ arithmetic [Pou94a] [Pou94b] [Pie94a].

2.1.4 Basic Computational Units within Residue Number Systems

RNS is suitable for the applications where only addition, subtraction and multiplication are involved. Therefore, useful basic computational units include the fundamental entities for addition and multiplication, such as Full Adder (FA), Half-Adder (HA), two-operand binary adder, Carry Save Adder (CSA), multioperand adder, and binary multiplier. As the operations on each residue channel are based on modular arithmetic, two-operand modular adder, modular multiplier and modular CSA are also discussed here. The diminished-one system will be introduced for efficient hardware implementation of modulo $2^n + 1$ arithmetic.

2.1.4.1 Half Adder, Full Adder and Unit-Gate Model

One-bit half-adder (HA) and full-adder (FA) are the basic building blocks for arithmetic circuits. A one-bit HA receives two input bits a and b, producing a sum bit, s and a carry bit, c as follows:

$$\begin{aligned} s &= a \oplus b = a \cdot \bar{b} + \bar{a} \cdot b \\ c &= a \cdot b \end{aligned} \quad . \tag{2-4}$$

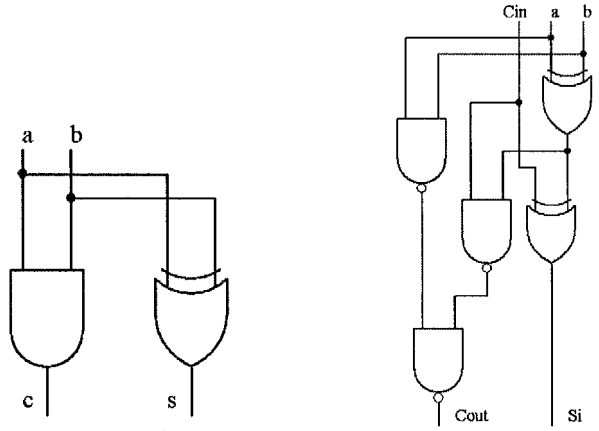
A one-bit FA receives three input bits, a, b and c_{in}, producing a sum bit, s and a carry bit, c_{out} as follows:

$$\begin{aligned} s &= a \oplus b \oplus c_{in} \\ c_{out} &= a \cdot b + a \cdot c_{in} + b \cdot c_{in} \end{aligned} \quad . \tag{2-5}$$

There are many ways to implement HA and FA. Figure 2.1(a) shows one way to implement HA using AND/exclusive-OR (XOR) gate, and Figure 2.1(b) shows one way to implement FA using NAND/Exclusive-OR (XOR) gate. Figure 2.2 shows the

symbols for HA and FA. A one-bit FA is sometimes referred to as a (3; 2)-counter, meaning that it counts the number of 1s among its three input bits and represents the result as a 2-bit number.

Unit-gate model is a simple approach for the area and delay estimation for complex gates and some basic functional modules such as adders and multipliers [Tya93] [Zimm99] [Ver02]. It is very commonly used in the literature, with reasonably good accuracy, for comparing the relative area-time complexity of different circuits. This model assumes that each two-input monotonic gate, excluding the exclusive-OR gate, is counted as equivalent to one elementary gate for both the area and delay costs. An exclusive-OR gate accounts for two elementary gates for both the area and delay costs. Therefore, a FA occupies an area of seven gates and has four gate delays, and a HA occupies an area of three gates and has two gate delay. This model will be used in this work for making comparisons of our proposed multioperand modular adders.



(a) HA (b) FA

Figure 2.1 *Possible implementations for HA and FA*

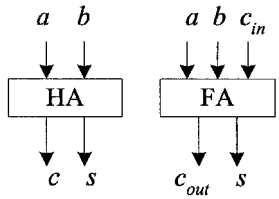


Figure 2.2 *Symbols for HA and FA*

If some of the FA's inputs are fixed to '0' or '1', FA can be simplified [Hwan76]. For example, an FA with a constant input, '1' can be reduced to a pair of two-input

XNOR and OR gates, a FA with a constant input '0' can be reduced to a pair of two-input XOR and AND gates. For simplicity, when one of the inputs is fixed to '0' or '1', the area is reduced to half. If an FA is fed with two constant inputs of '0' and '1', it can be reduced to an inverter.

2.1.4.2 Binary Adder

Binary addition is by far the most frequently used operation both in general-purpose systems and application specific processors. There are many adder structures, such as linear time ripple carry adders (RCA), the square-root time carry skip and carry select adders, and logarithmic time CLA adders [Hwan79][Bren82] [Nag96]. RCA is a linear time adder consists of n FAs for n-bit binary adder. Figure 2.3 shows the structure of a RCA. The delay of the n-bit RCA is $n \cdot t_{FA}$, where t_{FA} is the delay of a FA, because the critical path usually begins at the least significant bit (LSB) position, and proceeds through the carry propagation chain to the most significant bit (MSB) position. The key to fast addition is a low-latency carry network, since once the carry into position i, c_i is know, the sum digit can be determined from the operand digits x_i and y_i by:

$$s_i = x_i \oplus y_i \oplus c_{i-1}. \tag{2-6}$$

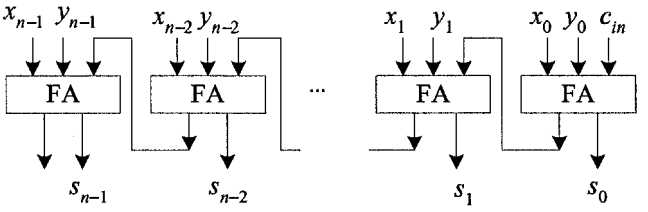


Figure 2.3 *Ripple carry adder (RCA)*

From the point of view of carry propagation and the design of a carry network, the actual operands' digits are not important. What matters is whether in a given position a carry is generated, propagated, or absorbed. In the case of binary addition, the generate, propagate and the absorb signals are characterized as follows:

$$\begin{aligned} g_i &= x_i \cdot y_i \\ p_i &= x_i \oplus y_i \\ t_i &= \overline{x_i + y_i} \end{aligned} \quad . \tag{2-7}$$

The propagate can also be defined as

$$h_i = x_i + y_i \tag{2-8}$$

Using the preceding signals, the carry, c_i can be computed by:

$$c_i = g_i + p_i \cdot c_{i-1} \tag{2-9}$$

CLA adders can achieve a time complexity of $O(\log_2 n)$ and have an area complexity of $O(n\log n)$ [Parh00]. Due to their speed and modularity, CLA adders are as popular in RNS as in normal binary system. There are two types of CLA adders, namely the one-level CLA and the two-level CLA [Efst94]. The n-bit one-level CLA adder consists of three units: the carry propagate/generate (CPG) unit, the carry look-ahead (CLA) unit and the summation (S) unit. When the bit width increases, the number of inputs of the high-order gates in the carry generate logic of a one-level CLA adder also increases. In current VLSI technology, gates with a small number of inputs are much faster than gates with more inputs. Thus, adders with large number of inputs may require two or more levels of carry look-ahead subunits. The n-bit two-level CLA adder consists of five functional units [Efst94]: the carry propagate/generate (CPG) unit, the group propagate/gate (GPG) unit, the between groups carry look-ahead (GBCLA) unit, the group carry look-ahead (GCLA) unit, and the summation (S) unit. The output signals of the CPG unit are divided into $m = \lceil n/k \rceil$ groups. The GPG unit consists of $m-1$ subunits with k-inputs and a subunit with $r = n - k(m-1)$ inputs.

The above discussion is based on the carry propagate definition of Equation (2-7). When the carry propagate is defined by Equation (2-9), the same analysis can also be applied based on the relation that $c_i = g_i + h_i \cdot c_{i-1}$ and $s_i = p_i \oplus c_{i-1}$.

2.1.4.3 Carry Save Adder and Dot Notation

CSA can be used to implement fast arithmetic operations. CSA has three input signals and two output signals, i.e., carry and sum. A one-bit CSA is the same as a FA. Figure 2.4 shows an n-bit CSA, which performs the following function:

$$2 \cdot C + S = X + Y + Z, \tag{2-10}$$

where C and S are the carry and sum outputs of the CSA.

A CSA tree can reduce k n-bit binary numbers to two numbers having the same sum in $O(\log n)$ levels. An additional carry-propagate adder (CPA) is used to reduce the two resulting numbers into a single binary number. The width of the CSAs varies, but generally they are close to n bits. The width of the CPA is at most $n + \log_2 k$ [Parh00].

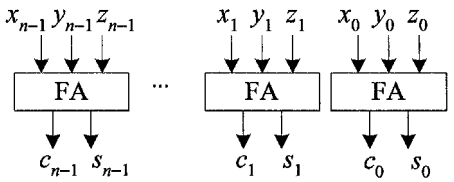


Figure 2.4 *n-bit carry save adder (CSA)*

The Wallace tree is an architecture that uses CSA to accumulate the partial products of multiplication into two binary partial sums with minimal propagation delay. Since each CSA reduces the number of partial products by a factor of 1.5, the smallest height, or the number of levels of the CSA tree, $L(K)$ of an K-input Wallace tree satisfies the following recurrence [Parh00]:

$$L(K) = 1 + L\left(\left\lceil \frac{2K}{3} \right\rceil\right). \tag{2-11}$$

The maximum number of operands, K that can be processed with an L-level CSA tree can also be obtained from a recursive formula:

$$K(L) = \left\lfloor \frac{K(L-1)}{2} \right\rfloor \times 3 + \left| K(L-1) \right|_2 \tag{2-12}$$

for $L = 2, 3, \ldots$ with the initial value of $K(1) = 1$.

Dot notation is a useful way to show the structure of an adder or multiplier, when only the position or alignment of bits, rather than their values, is important. It is typically used for illustrating the input allocations of the 1-bit CSAs of conventional CSA tree for normal binary multiplication. Figure 2.5 shows the dot notation for the addition of seven 6-bit numbers using conventional Wallace CSA tree. The three dots enclosed in a rectangular strip are inputs to a 1-bit CSA. The horizontal lines demarcate the levels of the CSA tree. However, for modular addition with special base, there is some subtle implementation differences of the Wallace tree.

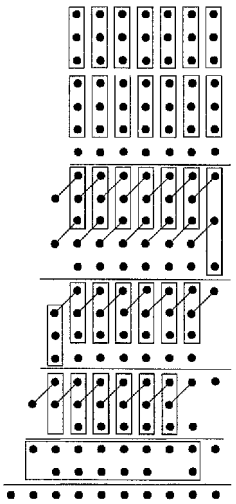


Figure 2.5 *Addition of seven 7-bit numbers using Wallace CSA tree*

For a modulo $2^n - 1$ CSA, Equation (2-10) is rewritten as follows:

$$|X + Y + Z|_{2^n-1} = |2 \times C + S|_{2^n-1} \tag{2-13}$$

Let $C = (C_{n-1}C_{n-2}\ldots C_0)_2$, and $S = (S_{n-1}S_{n-2}\ldots S_0)_2$, then Equation (2-13) becomes

$$\begin{aligned} &|X + Y + Z|_{2^n-1} \\ &= |2 \times (C_{n-1}C_{n-2}\ldots C_0)_2 + S|_{2^n-1} \\ &= |2 \times (0C_{n-2}\ldots C_0)_2 + 2 \times C_{n-1} \times 2^{n-1} + S|_{2^n-1} \\ &= |(C_{n-2}\ldots C_0C_{n-1})_2 + S|_{2^n-1} = |C' + S|_{2^n-1} \end{aligned} \tag{2-14}$$

where

$$C' = \left(C_{n-2}...C_0 C_{n-1}\right)_2 . \tag{2-15}$$

According to Equation (2-15), the carry output of the $2^n - 1$ modulo CSA will be shifted left circularly by one bit position before feeding to the next level of the CSA tree. This operation is called the end-around carry (EAC). Therefore, each level of the modulo $2^n - 1$ CSA tree has a fixed width of n bits. The regularity and localized interconnections make the resultant Wallace tree for the modulo $2^n - 1$ CSA even more appealing than its binary counterpart. Figure 2.6 shows a bit level implementation of a 2-level modulo $2^4 - 1$ CSA tree with EAC.

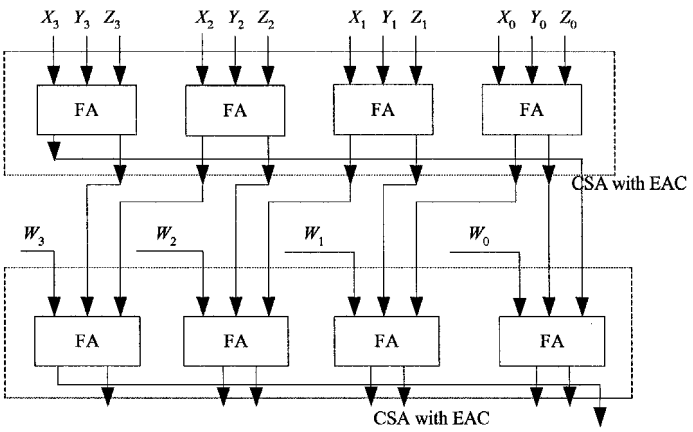


Figure 2.6 *Modulo $2^n - 1$ CSA tree with EAC*

2.1.4.4 Diminished-One System

To develop an efficient coding system for Fermat number arithmetic, Leibowitz proposed the diminished-one system [Leib76], which is not only suitable for Fermat numbers, but also for moduli of the form $2^n + 1$. For modulo $2^n + 1$ arithmetic, the values of the operands can range from 0 to 2^n inclusive. If the operand is zero, the diminished-one system is inhibited and the arithmetic operation can be simplified due to the zero input. Otherwise, the diminished-one representation of an operand, A is defined by:

$$d(A) = A - 1 \tag{2-16}$$

According to Equation (2-16), a diminished-one number, $d(A)$, is confined to n-bit.

The relationship between the binary representation and the diminished-one representation is given as follows:

$$A = d(A) + 1 \tag{2-17}$$

It indicates that a '1' must be added to the diminished-one number to obtain the binary result.

The following axioms for diminished-one additions are obtained according to [Leib76]:

$$d(A_1 + A_2) = |d(A_1) + d(A_2) + 1|_{2^n+1} \tag{2-18}$$

$$d\left(\sum_{i=1}^{k} A_i\right) = \sum_{i=1}^{k}{}_{\oplus}\, d(A_i) = |d(A_1) + \ldots + d(A_k) + k - 1|_{2^n+1} \tag{2-19}$$

Equation (2-18) defines a two-operand diminished-one addition wherein a '1' is added to the n-bit binary sum of the two diminished-one operands. In Equation (2-19), $\Sigma_{\oplus}$ denote a multi-operand diminished-one addition operator wherein multiple pairs of diminished-one additions defined by Equation (2-18) are performed iteratively.

Diminish-one system has the following properties, which lead to more efficient implementations of diminished-one modulo $2^n + 1$ arithmetic relative to their corresponding implementations in normal binary system.

Property 2.15: The negation of a diminished-one number, A is the one's complement of A, i.e., $\overline{A}$.

Property 2.16: The diminished-one representation of the product of modulo $2^n + 1$ multiplication of a positive integer, x and a power-of-two term, 2^r can be accomplished by expressing the diminished-one number $d(x)$ in an n-bit binary representation, complement its MSB, and then shifting it circularly by r bits to the left.

For example, if $x = 11$, $r = 3$, we have

$$d(x\cdot 2^r)=CCLS_n(d(x),r)=CCLS_4((1010)_2,3)=(0\bar{1}\bar{0}\bar{1})_2=2,$$

where the function $CCLS_n(x, r)$ denotes a complementary circular left shift of an n-bit number, x by r bits.

Property 2.17: The diminished-one representation of the product of modulo 2^n+1 multiplication of a negative integer, x and 2^r can be accomplished by expressing the diminished-one number of $-x$, i.e., $\overline{d(-x)}$ in an n-bit binary representation, complement its MSB, and then shifting it circularly by r bits to the left, where $\bar{a}$ denotes the one's complement of the n-bit positive integer a.

For example, if $x=-11$, $r=1$, then

$$d(x\cdot 2^r)=CCLS_n(\overline{d(-x)},r)=CCLS_4((\bar{1}\bar{0}\bar{1}\bar{0})_2,1)=(1011)_2=11.$$

For a diminished-one CSA, let its three diminished-one inputs be $d(A_1)$, $d(A_2)$, and $d(A_3)$. Equation (2-19) can be rewritten as follows:

$$\begin{aligned}
&\sum_{i=1}^{3}{}_{\oplus}\, d(A_i)=|2\times C+S+2|_{2^n+1}= \\
&|2\times(0C_{n-2}\ldots C_0)_2+2\times C_n\times 2^{n-1}+S+2|_{2^n+1} \\
&=|(C_{n-2}\ldots C_0 0)_2+1-C_n+S+1|_{2^n+1} \\
&=|(C_{n-2}\ldots C_0\bar{C}_{n-1})_2+S+1|_{2^n+1} \\
&=|C'+S+1|_{2^n+1}
\end{aligned}, \qquad (2\text{-}20)$$

where

$$C'=(C_{n-2}\ldots C_0\bar{C}_{n-1})_2\,. \qquad (2\text{-}21)$$

It should be noted that the MSB of the carry output of the binary CSA is complemented and then the expression is circularly shifted to the left by one bit. This is called the complementary end-around carry (CEAC). CEAC is different from the EAC defined by Equation (2-15), which is used in the modulo 2^n-1 CSA. To differentiate them, the realization of Equation (2-20) is called the CSA with CEAC.

2.2 Reverse Converters

In RNS, the reverse conversion is a problem of translating a residue representation (x_1, x_2, …, x_n) of a weighted number X with respect to a moduli set $\{P_1, P_2, \ldots, P_n\}$ back to its weighted form. The CRT of Equation (2-3) can be used to perform the reverse conversion. Direct implementation of CRT is VLSI inefficient as it involves large modular multipliers and modular adders. Another approach to implement the reverse conversion is by the Mixed-Radix Conversion (MRC) [Sza67]. A mixed-radix representation is just another form of weighted number representation, and it allows magnitude comparison to be performed easily. The conversion from a residue number to a mixed-radix number is known as the MRC. MRC is a sequential process and often requires a number of look-up tables (LUT). For general moduli sets in practical systems with large dynamic ranges, both approaches have reached the limit of efficiency. Recently, Wang proposed several ingenious methods for the reverse conversion problems, which are called the New CRTs [Wang98] [Wang00]. The new algorithms do not need large modulo adders, and the coefficients involved are smaller compared with the coefficients in CRT and MRC algorithms. However, the reverse converters based on the new CRTs for general moduli sets are still relatively complex. To reduce the complexity of the implementation of RNS, special moduli sets are used extensively and it is very easy to select a suitable moduli set for a RNS of a specific dynamic range because there exists clear relationship between the special moduli sets and the corresponding dynamic ranges.

2.2.1 CRT-Based Reverse Converters

The algorithm for the CRT based reverse converter is given by *Theorem 2.1*, which can be expressed as $X = \left| \sum_{i=1}^{n} \beta_i \right|_M$, where $\beta_i = \left| x_i / \hat{M}_i \right|_{P_i} \hat{M}_i \Big|_M$. A straightforward implementation of this formula is to generate the partial sum, β_i using modular multipliers, and use a multiple-operand modular adder to obtain the final value of X. The terms other than x_i are constants for a specific moduli set. These coefficients can therefore be precomputed and stored in ROMs so that the product of the corresponding coefficient and x_i can be read directly from the ROM. The limitation of this method is the overhead of ROM accesses and address decoding if large numbers

of small lookup tables are used. For memoryless implementation, CRT requires several large modular multipliers and modulo M reduction. Generally speaking, there are two ways to improve the implementation of CRT algorithm. One way is to introduce a scaling factor to avoid the mod M operation. Another way is to use distributed processing elements (PEs), CSA tree and modular adder to implement CRT algorithm.

Soderstrand et al. [Sod83] proposed a CRT-based method to reduce the number of ROMs and the complexities of the reverse converter by using the scaling factor of M. Therefore, the CRT of Equation (2-3) can be written as:

$$\frac{X}{M} = \frac{1}{M}\sum_{i=1}^{n}\left|\hat{M}_i x_i \left|\frac{1}{\hat{M}_i}\right|_{m_i}\right|_M = FRAC\left[\sum_{i=1}^{n} b_{m_i}\right], \tag{2-22}$$

where

$$b_{m_i} = \frac{x_i}{m_i}\left|\frac{1}{\hat{M}_i}\right|_{m_i}. \tag{2-23}$$

As X/M can be greater than unity, only the fractional part of Equation (2-22) is preserved and the integer part is removed from the output of the final adder. The conversion from x_i to b_{m_i} of Equation (2-23) can be implemented by only one level of ROMs. Therefore, only one level of adders is required to sum Equation (2-22). A modification to this technique results in drastic reduction of hardware with only a slight reduction in the dynamic range of the converter. Vu [Vu85] found that some results of the method in [Sod83] are erroneous. He derived the number of bits needed to represent the fractional form accurately in order that the sign detection can be accurately accomplished.

Meehan at el. [Mee90] proposed a similar reverse conversion algorithm to that of [Sod83], where the residues are in a scaled version produced by the scaled forward conversions. In this technique, the forward converter produces a scaled version of the residue x, $x' = \left|2^{-k}x\right|_m$, where k is an implementation-dependent constant. The scaling

of the residue can be done recursively, which makes systolic implementation feasible.

Cardarilli et al. [Card00] proposed an alternative implementation of scaled CRT-based reverse converter where M is the scale factor. Thus, CRT Equation (2-3) can be written as follows:

$$\frac{X}{M} = \left\langle \sum_{i=1}^{n} \frac{1}{m_i} \left| x_i \frac{1}{\hat{M}_i} \right|_{m_i} \right\rangle_1 = \left\langle \sum_{i=1}^{n} \frac{y_i}{m_i} \right\rangle_1 . \tag{2-24}$$

The scaling by M modifies the output range of the resulting CRT, which is now bounded in the interval $0 \le X/M < 1$. This operation introduces a modular operation, $\langle \ldots \rangle_1$, which discards all bits of the integer part of Equation (2-24). The simplification of modulo extraction implies the use of a set of LUTs to store the fractional terms of Equation (2-24). In order to reduce the size of the LUTs and speed up the scaled CRT architecture, the authors proposed a new algorithm, where the LUT stores n correct vectors with wordlength of only $\lceil \log_2 m_i \rceil$ bits. Due to the small number of bits required for each term in Equation (2-24), according to the author, in the case of LUT implementation based on CMOS ROM, the access delay will reduce, and the subsequent accumulator implemented by the CSA tree will require smaller silicon area.

G. Alia and E. Martinelli [Alia84] presented several PE-based VLSI architectures for both forward and reverse conversions based on the direct implementation of CRT. Their reverse conversion algorithm is given as follows. Let $X_1 = (x_1, 0, \ldots, 0)$, $X_2 = (0, x_2, 0, \ldots, 0)$, $\ldots$, $X_k = (0, \ldots, 0, x_k, 0, \ldots, 0)$, $X_n = (0, 0, \ldots, 0, x_n)$, then $X = |X_1 + X_2 + \ldots + X_n|_M$. Moreover, X_i can be expressed as

$$X_i = \left| \sum_{k=0}^{\lceil \log P_i \rceil - 1} b_k^{(i)} Q^{(i,k)} \right|_M \tag{2-25}$$

where $b_k^{(i)}$ is the k-th bit of x_i, and $Q^{(i,k)}$ is the $(n+1)$-bit weighted representation of the RNS number $\left(\underbrace{0, \cdots, 0, 2^k}_{i}, 0, \cdots, 0 \right)$. Hence, X can be obtained by

$$X = \left| \sum_{i=1}^{n} X_i \right|_M = \left| \sum_{i=1}^{n} \left| \sum_{k=0}^{\lceil \log P_i \rceil - 1} b_k^{(i)} Q^{(i,k)} \right|_M \right|_M . \quad (2\text{-}26)$$

Equation (2-26) can be implemented by $\lceil \log_2 M \rceil /2$ Processing Elements (PE), where each PE is able to perform the addition of two $\lceil \log_2 M \rceil$-bit numbers modulo M and receives as inputs a pair of adjacent bits in the string of all residue digits. The reported area complexity is $O(n^2 \log n)$, and the time complexity is $O(\log^2 n)$.

Capocelli *et al.* [Capo88] proposed another VLSI architecture for the reverse conversion, and their work is based on Alia *et al.*'s architecture. In their architecture, only adjacent processors can communicate with each other and this is carried out in constant time. Despite their approach is not better than Alia's approach in terms of area-time complexity, it achieves better area utilization and increased locality. From the VLSI perspective, localized communication implies reduced routing area and capacitance, and minimizes control overheads. This can significantly improve the performance and the power dissipation when advanced deep-submicron technologies are chosen for its implementation.

Elleithy *et al.* proposed a modified architecture with $O(\log n)$ time complexity in [Elle89] [Lee89] [Elle92]. In their architecture, the partial sums are generated by ROMs, and followed by a modular addition. After the modular addition, a so-called Range Determinator is adopted to correct the sum. The last stage is a final corrector, where one CLA adder and a cluster of tri-state multiplexers are used to obtain the final value. This scheme is very costly in terms of the hardware required for the second stage, which is the partial sum modular addition. The authors presented two implementations for the partial sum modular addition, one is based on multilevel CSA tree followed by a CPA [Hwan79], the other is the multi-operand modulo adder proposed in [Elle86].

Barsi and Pinotti [Bar94] proposed an entirely different architecture, which is also based on the CRT. According to CRT,

$$X = \left| \sum_{i=1}^{n} \left| \frac{1}{\hat{M}_i} \right|_{P_i} \cdot x_i \cdot \hat{M}_i \right|_M = \left| \sum_{i=1}^{n} \beta_i \cdot \hat{M}_i \right|_M$$

where $\beta_i = \left| \frac{x_i}{\hat{M}_i} \right|_{P_i}$. The authors suggested to split the term $\hat{M}_i$ into a summation of t terms, i.e., $\hat{M}_i = \sum_{j=0}^{t-1} \mu_{ij} B^j$, where $0 \le \mu_{ij} < B$ and $B = 2^k$, to ensure the uniqueness of the representation. Thus,

$$X = \left| \sum_{i=1}^{n} \beta_i \sum_{j=0}^{t-1} \mu_{ij} B^j \right|_M = \left| \sum_{j=0}^{t-1} \sum_{i=1}^{n} w_{ij} B^j \right|_M = \left| \sum_{j=0}^{t-1} \left\lfloor \sum_{i=1}^{n} w_{ij} / B \right\rfloor B^{j+1} + \sum_{j=0}^{t-1} \left| \sum_{i=1}^{n} w_{ij} \right|_B B^j \right|_M = |Q + R|_M \quad (2\text{-}27)$$

where $w_{ij} = \beta_i \mu_{ij}$, $Q = \sum_{j=0}^{t} q_j B^j$, $R = \sum_{j=0}^{t} r_j B^j$, $q_j = \left\lfloor \sum_{i=1}^{n} w_{i(j-1)} / B \right\rfloor$ $q_0 = 0$, r_0=0, $r_j = \left| \sum_{i=1}^{n} w_{ij} \right|_B$, for j =1, 2, …, t.

The bottleneck of large modular multiplications and modular addition in the direct implementation of CRT is overcome by splitting $\hat{M}_i$ into small groups of k bits so that large coefficient multiplier is broken down into several smaller constant multipliers. The outputs of these multipliers are added up using a CSA tree. This architecture has good area and time complexity in VLSI implementation.

Another important contribution to boost the RNS converter perfromance is the excellent work of Piestrak on Multiple Operand Modular Adder (MOMA). It uses CSA with EAC [Pie94a] by exploiting the periodic properties of the moduli. What is new and significant is the novel implementation of modular adder. This modular adder was adopted in [Pie94b] for the design of his reverse converter. In the reverse converter proposed in [Pie94b], the partial sums are generated by ROMs, and a CSA tree is used to obtain the carry and sum of these partial sums. These two stages are the same as that of Elleithy's method [Elle92], except the next stage, which is distinct in two aspects: (i) it avoids using a CPA that directly follows the CSA tree; and (ii) it

uses a different technique to keep an intermediate result of mod M summation and to extract the final result.

Some researchers [Li91] [Per94] have studied the hierarchical residue number systems (HRNS), where the problem of reverse conversion with n moduli is reduced into sub-problems of 2-moduli in a recursive manner. This pioneering work can be studied in conjunction with the original and more influential idea of the New CRT-II [Wang98] [Wang00]. Li *et al.* [Li91] proposed a modularized decoding algorithm in which each module decodes only 2 residues so that a combination of an adder and a LUT can be used to obtain the result. As their implementation involves the traditional lookup table augmented by addition, the size of the table will increase exponentially with the bit widths of the moduli. The architecture of Perumal's method [Per94] is also hierarchical, but the computing element for the decoding of the two residues is not the same as that of Li's. Let x_1 and x_2 be the residues corresponding to the moduli set $\{m_1, m_2\}$, then X can be calculated by

$$X = \left| x_1 + \left| m_1 \left| \frac{1}{m_1} \right|_{m_2} (x_2 - x_1) \right|_{m_1 m_2} \right|_{m_1 m_2}.$$

Latches are used inside the architecture to increase its throughput rate by pipelining.

Srikanthan *et al.* presented an area–time efficient architecture for the reverse converter for general moduli set, which is called the Compressed Multiply Accumulate (CMAC) converter [Sri98] [Bha99b]. The efficiency of the architecture results from the careful identification and elimination of redundancy in the existing architectures. Specifically, the partial sum generation and addition are merged into a single carry-save addition by using booth recoding techniques. Moreover, modulo multipliers are replaced by simple adders using the bit unfolding and uncorrected residues technique. Finally the modulo correction in [Li91] is adopted at the final stage to obtain the correct result.

The above CRT based architectures of reverse converters are mostly developed and evaluated for general moduli sets. Actually, these CRT implementation techniques are

generally more efficient when it is employed in reverse converters of special moduli sets, which will be discussed later in this chapter

2.2.2 MRC-Based Reverse Converters

Another approach to implement the reverse conversion is by the MRC. An integer X can be expressed in mixed-radix form with respect to the moduli set $\{m_1, m_2, \ldots, m_n\}$ as follows:

$$X = a_n \prod_{i=1}^{n-1} m_i + \cdots + a_3 m_2 m_1 + a_2 m_1 + a_1 \tag{2-28}$$

where a_is are the mixed-radix coefficients. The mixed-radix coefficients can be obtained from the residues by the following equations.

$$\begin{aligned}
a_1 &= x_1 \\
a_2 &= \left| (x_2 - a_1) \left| \frac{1}{m_1} \right|_{m_2} \right|_{m_2} \\
a_3 &= \left| \left((x_3 - a_1) \left| \frac{1}{m_1} \right|_{m_3} - a_2 \right) \left| \frac{1}{m_2} \right|_{m_3} \right|_{m_3} \\
&\cdots \\
a_n &= \left| \left(\left((x_n - a_1) \left| \frac{1}{m_1} \right|_{m_n} - a_2 \right) \left| \frac{1}{m_2} \right|_{m_n} - \cdots - a_{n-1} \right) \left| \frac{1}{m_{n-1}} \right|_{m_n} \right|_{m_n}
\end{aligned} \tag{2-29}$$

It is obvious that the process of obtaining the mixed-radix coefficients is sequential. In order to obtain a_i, a_{i-1} has to be calculated first. To speed up the computation, many lookup tables are required. For a simple 2-moduli set $\{P_1, P_2\}$, the integer X can be converted from its residue representation (x_1, x_2) by

$$X = a_1 + a_2 P_1 = x_1 + P_1 \cdot \left| (x_2 - x_1) \left| \frac{1}{P_1} \right|_{P_2} \right|_{P_2} \tag{2-30}$$

where $\left|1/P_1\right|_{P_2}$ is the multiplicative inverse of P_1 modulo P_2.

A detailed description of the recursive method for obtaining the mixed-radix

coefficients can be found in the textbook written by Szabo and Tanaka [Sza67]. In order to obtain all the n mixed-radix coefficients, a_i (i = 1, 2, …, n), $n(n-1)/2$ modulo subtractions and $n(n-1)/2$ modulo multiplications are required, where n is the cardinality of the moduli set. These operations are nested and cannot be performed in parallel. In general, $(n-1)$ lookup cycles are needed to generate all the coefficients. The hardware requirements include $n(n-1)/2$ tables with size of $2^{K+1} \times K$ bits, $n(n-1)/2$ adders with K-bit operands, and $n(n+1)/2 - 1$ latches, where K is the number of bits per modulus.

Huang [Huan83] presented a so-called fully parallel MRC algorithm for the reverse conversion. A series of modular adder networks is adopted to obtain the final coefficients from the parallel output of the lookup tables. The algorithm operates as follows.

Given a moduli set $\{m_1, m_2, \ldots, m_n\}$, and the residue representation of X is $(x_1, x_2, \ldots, x_n)$. Let $X_1 = (x_1, 0, \ldots, 0)$, $X_2 = (0, x_2, 0, \ldots, 0)$, …, $X_n = (0, \ldots, 0, x_n)$, where X_i can be called the i-th orthogonal projection of X. The i-th orthogonal projection of X has the mixed-radix representation as follows: $X_i = (0,0,\cdots,0,x_{i,i},a_{i,i+1},\cdots,a_{i,n})$, i.e., the mixed radix representation of X_i has $(i-1)$ zeros before the first nontrivial element. Furthermore, let $q_k = \prod_{i=1}^{k-1} m_i$, $q_1 = 1$. Thus, $X_i = \bar{x}_{i,i}q_i + \bar{x}_{i,i+1}q_{i+1} + \cdots + \bar{x}_{i,n}q_n$, where coefficient $\bar{x}_{i,j}$ can be obtained through the LUT. The mixed radix coefficients a_is can be obtained by summing, in modulo m, all the coefficients of X_i, i.e., $\bar{x}_{i,j}$, and the carry generated from the forming logic of a_{i-1}. The conversion delay is two clock cycles, and the hardware costs include $n(n+1)/2$ lookup tables of size $2^b \times b$, $n(n+1)/2$ latches, $n(n+1)/2$ adders and n comparators. Strictly speaking, their MRC implementation is still sequential because in the worst case, the carries could still ripple through the whole adder network.

Chakraborti *et al.* [Chak86] presented an architecture based on Huang's method by combining some of the lookup tables and eliminating the adder networks in the architecture, but some other adders are inserted. The authors assume the ROM and moduli sizes are designed such that two moduli may be used to address a ROM

lookup table. Therefore, it is necessary to separate the original RNS into some sub-RNS containing only two non-zero residues. Unfortunately, this makes the design less scalable. For high cardinality, their method is not obviously advantageous than Huang's.

Yassine et al. [Yas91] proposed an improved MRC method. The authors have carefully selected the moduli sets to lower the computational cost. The MRC algorithm of Equations (2-29) can be written in another form as follow:

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= \left| \left| \frac{1}{m_1} \right|_{m_2} (x_2 - a_1) \right|_{m_2} \\ a_3 &= \left| \left| \frac{1}{m_1 m_2} \right|_{P_3} (x_3 - (a_2 m_1 + a_1)) \right|_{m_3} \\ &\cdots \\ a_n &= \left| \left| \frac{1}{m_1 m_2 \cdots m_{n-1}} \right|_{m_n} (x_n - (a_{n-1} m_{n-2} \cdots m_1 + \cdots + a_2 m_1 + a_1)) \right|_{m_n} \end{aligned} \tag{2-31}$$

If we choose the moduli set such that

$$\left| \frac{1}{\prod_{i=1}^{j-1} m_i} \right|_{m_j} = 1, \tag{2-32}$$

where $j = 2, 3, \ldots, n$, a number of multiplications in Equation (2-31) can be reduced. It becomes

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= |(x_2 - a_1)|_{m_2} \\ a_3 &= |(x_3 - (a_2 m_1 + a_1))|_{m_3} \\ &\cdots \\ a_n &= |(x_n - (a_{n-1} m_{n-2} \cdots m_1 + \cdots + a_2 m_1 + a_1))|_{m_n} \end{aligned} \tag{2-33}$$

In Equation (2-33), only $n(n-1)/2$ subtractions and $n-2$ multiplications are required. However, the criterion given by Equation (2-32) has restricted the number of permissible moduli sets.

Miller et al. presented a parallel MRC architecture based solely on lookup tables [Mill98]. Their ROM addressing method is different from the traditional MRC technique and it eliminates the adder associated with each table in Szabo's method. The removal of adder will increase the address lines from $K + 1$ to $2K$, thus the table size will increase from $2^{K+1} \times K$ in Szabo's method to $2^{2K} \times K$, while the number of tables and lookup cycles remain the same as in Szabo's method.

2.2.3 New CRT-Based Reverse Converters

The New CRTs are proposed by Y. Wang [Wang96] [Wang98] [Wang00] [Wang02]. The following theorem taken from [Wang00] is useful for the understanding of the New CRT-I.

Theorem 2.2: Given the residue number $(x_1, x_2, \ldots, x_n)$ with respect to the moduli set $\{P_1, P_2, \ldots, P_n\}$, the corresponding decimal number X can be computed using the following equation:

$$X = \left| x_1 + k_1 P_1 (x_2 - x_1) + k_2 P_1 P_2 (x_3 - x_2) + \cdots + k_{n-1} P_1 P_2 P_3 \cdots P_{n-1} (x_n - x_{n-1}) \right|_M \tag{2-34}$$

where

$$\begin{aligned} &\left| k_1 P_1 \right|_{P_2 P_3 \cdots P_n} = 1 \\ &\left| k_2 P_1 P_2 \right|_{P_3 P_4 \cdots P_n} = 1 \\ &\cdots \\ &\left| k_{n-1} P_1 P_2 \cdots P_{n-1} \right|_{P_n} = 1. \end{aligned} \tag{2-35}$$

By reorganizing the terms in Equation (2-34), we have

$$X = \left| a_0 x_1 + a_1 P_1 x_2 + \cdots + a_{n-2} P_1 P_2 \cdots P_{n-2} x_{n-1} + a_{n-1} P_1 P_2 \cdots P_{n-1} x_n \right|_M \tag{2-36}$$

where

$$
\begin{aligned}
a_0 &= |1 - k_1 P_1|_M \\
a_1 &= |k_1 - k_2 P_2|_{P_2 P_3 \cdots P_n} \\
&\cdots \\
a_{n-2} &= |k_{n-2} - k_{n-1} P_{n-1}|_{P_{n-1} P_n} \\
a_{n-1} &= |k_{n-1}|_{P_n}
\end{aligned}
\tag{2-37}
$$

It should be noted that Equation (2-36) is not merely an alternative expression of CRT. It provides a number of alternative solutions to different formulations of the reverse conversion problems. Furthermore, we assume the weights, $a_i P_1 P_2 \ldots P_i$ in Equation (2-36) have the following uniquely defined mixed-radix representation:

$$
\begin{aligned}
a_0 &= a_{0,0} + a_{0,1} P_1 + \cdots a_{0,n-1} P_1 P_2 \cdots P_{n-1} \\
a_1 P_1 &= a_{1,1} P_1 + \cdots a_{1,n-1} P_1 P_2 \cdots P_{n-1} \\
&\cdots \\
a_{n-2} P_1 P_2 \cdots P_{n-2} &= a_{n-2,n-2} P_1 P_2 \cdots P_{n-2} + \cdots a_{n-2,n-1} P_1 P_2 \cdots P_{n-1} \\
a_{n-1} P_1 P_2 \cdots P_{n-1} &= a_{n-1,n-1} P_1 P_2 \cdots P_{n-1}
\end{aligned}
\tag{2-38}
$$

Definition 2.3: The matrix

$$
A = \begin{pmatrix}
a_{0,0} & 0 & \cdots & \cdots & 0 \\
a_{0,1} & a_{1,1} & \cdots & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
a_{0,n-2} & a_{1,n-2} & \cdots & a_{n-2,n-2} & 0 \\
a_{0,n-1} & a_{1,n-1} & \cdots & a_{n-2,n-1} & a_{n-1,n-1}
\end{pmatrix}
\tag{2-39}
$$

is called the characteristic matrix of the moduli $\{P_1, P_2, \ldots, P_n\}$, where $a_{j,i} < P_{i+1}$, i.e., every number in the *characteristic matrix* is bounded by P_i. Given the RNS number $X = (x_1, x_2, \ldots, x_n)$, the *first-order radix* is define as a vector B, $B = A \cdot X^T$ where the elements $B_i = a_{0,i} x_1 + a_{1,i} x_2 + \cdots + a_{i,i} x_{i+1}$ is called the i-th pseudo-digit, i.e.,

$$
B = \begin{pmatrix} B_0 \\ B_1 \\ \cdots \\ B_{n-2} \\ B_{n-1} \end{pmatrix} = A \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{n-1} \\ x_n \end{pmatrix}.
\tag{2-40}
$$

The following theorems are directly excerpted from [Wang00].

Theorem 2.3: The decimal number X can be computed as

$$X = \left| B_0 + B_1 P_1 + B_2 P_1 P_2 + \cdots + B_{n-1} P_1 P_2 \cdots P_{n-1} \right|_{P_1 P_2 \cdots P_n} \tag{2-41}$$

where B_is are given by Equations (2-39) and (2-40).

Theorem 2.3 is known as the New Chinese Remainder Theorem I (New CRT-I). The New CRT-I can be further simplified to smaller modulo operations for some special moduli sets [Wang02], as shown in Equation (2-62) when it is applied to the triple moduli set to be discussed later.

Theorem 2.4: Given a general moduli set, $\{P_1, P_2, \ldots, P_n\}$ with $P_1 < P_2 < \ldots < P_n$, the algorithm, *translate*, finds the correct decimal representation of the RNS number $X = (x_1, x_2, \ldots, x_n)$.

Algorithm *translate* $((x_1, x_2, \ldots, x_n), X)$

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

translate$((x_1, x_2, \ldots, x_n), N_1)$, $M_1 = P_1 P_2 \ldots P_t$

translate$((x_{t+1}, x_{t+2}, \ldots, x_n), N_2)$, $M_2 = P_{t+1} P_{t+2} \ldots P_n$,

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

(2) If $n = 2$, then *findno*(x_1, x_2, P_1, P_2, X).

Procedure *findno*(x_1, x_2, P_1, P_2, X)

(1) Find a k_0 such that $k_0 \cdot P_2 = 1 \bmod P_1$,

(2) $X = x_2 + P_2 \left| k_0 (x_1 - x_2) \right|_{P_1}$.

Theorem 2.4 is known as the New Chinese Remainder Theorem II (New CRT-II).

Several reverse converters based on the New CRTs are proposed in [Wang00]. The implementation of the CRT-I involves two steps. The first step is to obtain B, the *first-order radix* of the RNS number $(x_1, x_2, \ldots, x_n)$ by Equations (2-38) to (2-40). The second step is to obtain the final X using Equation (2-41). If the moduli set meets the

following condition, the reverse converter can be implemented without the modulo M operation.

$$m_i > m_1 + m_2 + \ldots + m_{i-1}, \tag{2-42}$$

2.2.4 Reverse Converters for Special Moduli Sets

Our survey and discussion on well-known architectures of reverse converters consider the general moduli sets. A general moduli set RNS has the following disadvantages compared to one that is constructed from the special moduli set.

(1) It is difficult to choose a suitable general moduli set when designing a specified dynamic range RNS, because there is no clear relationship between the dynamic range and the general moduli sets. For special moduli set, it is comparatively easier to choose a moduli set for a specified dynamic range. For example, the dynamic range of the three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is $3n$ bits, and the dynamic range of the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ is $(4n+1)$ bits.

(2) The forward conversion is relatively complex for the general moduli sets compared to the special moduli sets. This is especially true for the special moduli set with $2^n \pm 1$ type moduli.

(3) The RAU will be more efficient and simple for the special moduli sets of the $2^n \pm 1$ type than that of the general moduli sets.

(4) The reverse converters based on the general moduli sets have more complex architectures than that of the special moduli sets. This is true for both the CRT based converters and MRC based converters. Special moduli sets have some inherent number theoretic properties which can be exploited to allow efficient hardware implementations.

In what follows, we will study several celebrated special moduli sets. We will classify the special moduli sets, and then compare their architectures and performances. Not all the special moduli sets will have all the desirable advantages over the general moduli sets. Some of them facilitate efficient implementation of the reverse

converters, while not necessarily efficient in the forward conversion or the RAU implementation; Some moduli sets are far too complex to be designed and implemented because the moduli are not co-prime, and their dynamic ranges are not large enough.

2.2.4.1 Special Moduli Sets

According to the mathematical expressions of the moduli, the special moduli sets can be classified into three categories: (1) $2n + m$ type. Examples of these moduli sets are $\{2n - 1, 2n, 2n + 1\}$ and $\{2n, 2n + 1, 2n + 2\}$. (2) $2^n \pm 1$ type. Examples of these moduli sets are $\{2^n - 1, 2^n, 2^n + 1\}$, $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ and $\{2^{n1} - 1, 2^{n1} + 1, 2^{n2} - 1, 2^{n2} + 1\}$. (3) $n \pm 2^k$ type. One example of these moduli sets is $\{n - 2^k, n + 2^k, m - 2^i, m + 2^i\}$. Based on the cardinality of the moduli sets, these special moduli sets can be further described as: (1) Three-moduli sets. Examples of these moduli sets are $\{2n - 1, 2n, 2n + 1\}$ and $\{2^{n-1} - 1, 2^n, 2^n - 1\}$. (2) Four-moduli sets. Examples of these moduli sets are $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$. (3) High cardinality moduli sets, if the cardinality is larger than 4. For example, the 8-moduli set $\{2^{n-3} + 1, 2^{n-2} - 1, 2^{n-1} + 1, 2^n - 1, 2^n, 2^n + 1, 2^{n+2} - 1, 2^{n+3} + 1\}$ for $n = 4m + 1$, where m is an integer. (4) Conjugate moduli sets, where the cardinality is always even, and the moduli sets are formed by pairs of two moduli of the form $a \pm b$. For example, the moduli sets $\{n - 2^k, n + 2^k, m - 2^i, m + 2^i\}$ and $\{2^{n1} - 1, 2^{n1} + 1, 2^{n2} - 1, 2^{n2} + 1\}$.

If CRT is adopted for the reverse conversion, the ease of calculating the multiplicative inverses determinates the extent to which the architecture can be simplified. The simple relationship between the special moduli sets and their dynamic ranges is another crucial factor. For example, the dynamic range of the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is $3n$ bits, whereas the relationship between the moduli set $\{n - 2^k, n + 2^k, m - 2^i, m + 2^i\}$ and its dynamic range is not clear. For the $2^n \pm 1$ type moduli sets, the forward conversion and the RAU are relatively simple, but for the $2n + m$ type moduli sets, the forward conversion and the RAU implementation are more complex. Another important factor is the balance of the moduli set. In an RNS, the RAUs are executing their operations on each residue channel in parallel, and the execution speed depends

on how evenly the workload is being shared by all the residue channels. If the moduli differ greatly in magnitude, there will be some slower channels that degrade the overall speedup for a given degree of parallelism. Table 2.1 provides a qualitative comparison of the figure-of-merits among these special moduli sets.

Table 2.1 *Comparisons of different special moduli sets*

Moduli Sets	Constraints	Dynamic Range	Multiplicative Inverses	Forward Conversion	RAU	Balance
$2n-1, 2n, 2n+1$	No	Vague	Fair	Poor	Poor	Good
$2n, 2n+1, 2n+2$	No	Vague	Fair	Poor	Poor	Good
$2^n-1, 2^n, 2^n+1$	No	$3n$ bits	Good	Good	Good	Good
$2^{n-1}-1, 2^n, 2^n-1$	No	$3n-1$ bits	Good	Good	Good	Fair
$2^n-1, 2^n, 2^{n+1}-1$	No	$3n+1$ bits	Good	Good	Good	Fair
$2^n-1, 2^n+1, 2^{2n}+1$	No	$4n$ bits	Good	Good	Good	Bad
$n-2^k, n+2^k, m-2^i, m+2^i$	$m_3m_4 - m_2m_1 = 2^R$	Vague	Poor	Poor	Poor	Fair
$2^n-1, 2^n, 2^n+1, 2^{n+1}+1$	$n = 2k+1$	$4n+1$ bits	Fair	Good	Good	Fair
$2^n-1, 2^n, 2^n+1, 2^{n+1}-1$	$n = 2k$	$4n+1$ bits	Good	Good	Good	Fair
$2^{n-1}-1, 2^n-1, 2^n, 2^n+1$	$n = 2k$	$4n-1$ bits	Good	Good	Good	Fair
$2^{n1}-1, 2^{n1}+1, 2^{n2}-1, 2^{n2}+1$	$gcd(2k_i+1, 2k_j+1) = 1,$	Vague	Good	Good	Good	Fair
$2^n-1, 2^n, 2^n+1, 2^{n+1}+1, 2^{n+2}-1$	$n = 2k+1$	$5n+3$ bits	Fair	Good	Good	Fair
$2^{n+1}, 2^n-1, 2^n+1, 2^n-2^{(n+1)/2}+1, 2^n+2^{(n+1)/2}+1$	$n = 2k+1$	$5n$ bits	Good	Poor	Poor	Fair
$2^{3n}, 2^{3n}-1, 2^{3n}+1, 2^{3n}-2^{(3n+1)/2}+1, 2^{3n}+2^{(3n+1)/2}+1$	$n = 2k+1$	$15n-2$ bits	Good	Poor	Poor	Fair
$2^{n-3}+1, 2^{n-2}-1, 2^{n-1}+1, 2^n-1, 2^n, 2^n+1, 2^{n+2}-1, 2^{n+3}+1$	$n = 4k+1$	$8n-1$ bits	Fair	Good	Good	Poor
$2^m-1, \quad 2^{2^0 m}+1, \quad 2^{2^1 m}+1, \ldots, 2^{2^k m}+1$	No	$2^{k+1}*m$ bits	Fair	Good	Good	Poor

2.2.4.2 Reverse Converters for $2n+m$ Type Moduli Sets

Premkumar *et al.* [Prem92] [Prem95] [Prem98] proposed two $2n + m$ *type* moduli sets, $\{2n-1, 2n, 2n+1\}$ and $\{2n, 2n+1, 2n+2\}$. These moduli sets are well balanced, but the forward conversion and the residue arithmetic operations based on these moduli sets are not efficient. The author introduced the concept of residue weights for the reverse conversion. In their algorithms, the integer corresponding to the residue representation that has a 1 in the j-th residue position and zero for all the other residues is designated as the weight of the j-th residue, w_j. In order to recover the

decimal equivalent X from its residue representation, the following formula has to be evaluated.

$$X = \left| \sum_{j=1}^{n} w_j x_j \right|_M \tag{2-43}$$

Hence, in the recovery of the decimal numbers, the weights have to be determined first. The weights of the moduli set $\{2n-1, 2n, 2n+1\}$ are determined as follows:

$$\begin{aligned} w_1 &= \frac{(m_1+1)m_2m_3}{2} \\ w_2 &= m_1(m_2-1)m_3 \\ w_3 &= \frac{m_1m_2(m_3+1)}{2} \end{aligned} \tag{2-44}$$

where $m_1 = 2n-1$, $m_2 = 2n$, $m_3 = 2n+1$.

In [Prem98], the authors proposed two modified architectures, known as the cost-effective (CE) architecture, and the high-speed (HS) architecture. The algorithm is given as follows:

$$X = m_2 \cdot \left| n(x_1 + x_3 - 2x_2) + P_1 \right|_{m_1m_3} + x_2 \tag{2-45}$$

where P_1 is defined by

$P_1 = \frac{x_1 - x_3}{2}$, if x_1 and x_3 are both odd or both even,

$P_1 = \frac{x_1 - x_3 + m_1m_3}{2}$, if x_1 is even and x_3 is odd or vice versa.

The CE version is a direct implementation of the above algorithm, and the HS version achieves higher speed with extra hardware. In both architectures, only small size multipliers, adders and complementers are involved. Let $r = \lceil \log_2(2n) \rceil$, then the delays of the CE and HS versions are approximately $8r$ FA delays and $4r$ FA delays, respectively. The hardware costs are $3r^2+15r$ FAs for the CE version and $3r^2+13r$ FAs for the HS version.

In [Prem95], the author proposed a reverse converter for the moduli set $\{2n, 2n + 1, 2n + 2\}$ with 2 as a common factor. Let $m_1 = 2n + 2$, $m_2 = 2n + 1$ and $m_3 = 2n$. The dynamic range is $\overline{M} = \prod_{i=1}^{3} \mu_i$, where μ_i is a relatively prime set defined as follows:

$$\mu_i = \{n+1, 2n+1, 2n\}, \text{ when } n \text{ is even,}$$
$$\mu_i = \{2n+2, 2n+1, n\}, \text{ when } n \text{ is odd.}$$

For the above sets of μ_i, the weights are calculated as follows:

$$w_1 = \frac{(m_1 + 2)m_2 m_3}{4}$$
$$w_2 = \frac{m_1(m_2 - 2)m_3}{2} \qquad (2\text{-}46)$$
$$w_3 = \frac{m_1 m_2 (m_3 + 2)}{4}$$

Finally, X can be calculated for the two cases as below:

For even $(x_1 + x_3)$

$$X = \left| \frac{m_2 m_3}{2} x_1 - m_1 m_3 x_2 + \frac{m_1 m_2}{2} x_3 \right|_{\overline{M}} \qquad (2\text{-}47)$$

For odd $(x_1 + x_3)$

$$X = \left| \frac{\overline{M}}{2} + \frac{m_2 m_3}{2} x_1 - m_1 m_3 x_2 + \frac{m_1 m_2}{2} x_3 \right|_{\overline{M}} \qquad (2\text{-}48)$$

From the expressions of X in Equations (2-47) and (2-48), the numbers involved in the multiplications are small compared to the numbers involved in the direct implementation of the CRT.

2.2.4.3 Reverse Converters for Conjugate Moduli Sets

There are two main categories of conjugate moduli sets, namely $\{n - 2^k, n + 2^k, m - 2^i, m + 2^i, \ldots\}$ and $\{2^{n1} - 1, 2^{n1} + 1, 2^{n2} - 1, 2^{n2} + 1, \ldots\}$. The first moduli set was proposed by Cardarilli *et al.* [Card98]. The moduli, $m_1 = n - 2^k$ and $m_2 = n + 2^k$ are

conjugates of each other, where n is an odd number. The two moduli m_1 and m_2 are coprime, and their absolute difference is 2^{k+1}. A four-moduli set using two such conjugate pairs is defined by

$$\{m_1, m_2, m_3, m_4\} = \{n - 2^k, n + 2^k, m - 2^i, m + 2^i\}, \tag{2-49a}$$

$$m_4 m_3 - m_1 m_2 = 2^R. \tag{2-49b}$$

The authors of [Card98] proposed some solutions that satisfy Equations (2-49a) and (2-49b) simultaneously.

The architectures of the reverse converters for the above four-moduli sets are based on a conjugated moduli converter module abbreviated as CONV2M. The algorithm for the implementation of CONV2M is summarized as follows. For a pair of conjugate moduli $\{m_1, m_2\}$, the corresponding decimal X can be obtained by

$$X = \frac{m_2 x_1 - m_1 x_2 + \alpha M}{m_2 - m_1} \tag{2-50}$$

where α is an integer. As the difference of the two moduli has to be 2^{k+1}, we let $\alpha = \alpha_0 + \alpha_1 2^{k_1}$, with $k_1 \le (k+1)$, then Equation (2-50) becomes

$$X = \frac{\dfrac{m_2 x_1 - m_1 x_2 + \alpha_0 M}{2^{k_1}} + \alpha_1 M}{2^{k+1-k_1}} \tag{2-51}$$

Equation (2-51) corresponds to two scaling operations. The first scaling factor is 2^{k_1} and the second is $2^{k+1-k_1} = 2^{k_2}$. If k_1 and k_2 are equal, then only one lookup table is needed for these operations. Although this moduli set can be extended to larger cardinality moduli sets, for example, six-moduli or eight-moduli set, it is far too complex to implement them efficiently. Hence, larger cardinality moduli sets are seldom used for this type of conjugate moduli set.

Conjugate moduli set $P = \{2^{n1} - 1, 2^{n1} + 1, 2^{n2} - 1, 2^{n2} + 1, \ldots\}$ is proposed by Skavantzos *et al.* [Ska99a]. In this case, the moduli, $2^n - 1$ and $2^n + 1$ are the conjugates of each other. The reverse converters that rely on such pairs of conjugate

moduli result in efficient two-level hardware implementations, and large dynamic ranges can be achieved without slowing down the RNS processing by enlarging the cardinality of the moduli set. Let $P = \{m_1, m^*_1, m_2, m^*_2, \ldots, m_L, m^*_L\} = \{2^{n1} - 1, 2^{n1} + 1, \ldots, 2^{nL} - 1, 2^{nL} + 1,\}$. The authors suggested some guidelines for the construction of P, and provided efficient implementations for the reverse conversion based on a two-level construction. However, this moduli set is not co-prime, thus, it is not easy to choose a suitable moduli set for a specified dynamic range as the choice is restricted by the downscaling factors. Secondly, the reduced dynamic range due to the non co-prime property also reduces its efficiency. Thirdly, half of the moduli are in form of $2^n + 1$ which is not efficient for both the forward converter and the RAU implementation comparing with the moduli in the form of $2^n - 1$. Except for some special applications which require large dynamic ranges, this category of moduli sets is inferior to the $2^n \pm 1$ type moduli sets.

2.2.4.4 Reverse Converters for the $2^n \pm 1$ Type Moduli Sets

In Sections 2.1.2, 2.1.4.3 and 2.1.4.4 of this chapter, several properties of the modular operations for the moduli of the $2^n \pm 1$ type have been introduced. They make the modular operation very efficient in comparison with those of the general modulus. Consequently, these special moduli sets form RNS's with efficient RAUs. Furthermore, our survey also shows that these moduli sets have efficient reverse converters as well. By the way, a moduli set is said to be a "superset" if it is based on the triple moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ with extended moduli of $2^n \pm 1$ type. For example, the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ is a superset.

There are many moduli sets of this type as listed in Table 2.1. Their reverse conversion algorithms will be discussed here in order of their cardinality.

The triple moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ has gained unprecedented popularity by virtue of its inherent number theoretic properties in CRT algorithm. CRT is often used for its reverse conversion, sometime combining with the MRC method. On the other hand, non-CRT methods are also proposed, but they are less efficient compared with the CRT-based methods.

Bernardson proposed a memoryless reverse converter for this moduli set [Ber85]. Let $X_1 = (x_1, x_3)$ be the integer corresponding to the moduli set $\{2^n - 1, 2^n + 1\}$. According to the CRT, X_1 can be obtained by

$$X_1 = \left| 2^{n-1}\left(2^n(x_1 + x_3) + (x_1 - x_3)\right)\right|_{2^{2n}-1} \tag{2-52}$$

Next, MRC is applied to combine X_1 with x_2 to obtain X by

$$X = X_1 + \left(2^{2n} - 1\right)\left|\left|X_1\right|_{2^n} - x_2\right|_{2^n} \tag{2-53}$$

For Equations (2-52) and (2-53), only modulo $2^{2n} - 1$ additions, adders, subtractors and comparators are involved. However, the hardware cost is high. It requires altogether 9 adders with different widths.

Ibrahim *et al.* proposed a reverse conversion algorithm which is not based on the CRT [Ibra88]. Their algorithm works as follows. Let

$$M = \left|X\right|_{(2^n-1)(2^n+1)} \tag{2-54}$$

thus

$$X = K2^{2n} - K + M \tag{2-55}$$

where $0 \le K < 2^n$ and $0 \le M < (2^{2n} - 1)$, and K is defined by

$$K = \left|\left|M\right|_{2^n} - x_2\right|_{2^n} \tag{2-56}$$

Therefore, M can be calculated from x_1 and x_3 by

$$M = B\left(2^n + 1\right) + x_3 \tag{2-57}$$

where B is defined according to the values of $D = x_1 - x_3$.

Case 1: If D is a nonnegative even integer, then $B = D/2$.

Case 2: If D is an odd integer, then $B = \frac{D + (2^n - 1)}{2}$.

Case 3: If D is a negative even integer, then $B = \frac{D + 2(2^n - 1)}{2}$.

The reverse converter consists of three circuits. The first circuit generates the number M from x_1 and x_3 by Equation (2-57). The second circuit generates K from M and x_2 by Equation (2-56), and the third circuit generates X from M and K using Equation (2-55). A total of two n-bit subtractors, one $3n$-bit subtractor, one $2n$-bit adder and one n-bit adder are needed.

Bi and Jones [Bi88] proposed a reverse conversion architecture based on the Bernardson's algorithm [Ber85]. By simplifying the implementation of Equation (2-52) with $4(n+1)$-bit adders. The total hardware cost is reduced to four n-bit adders and four $(n+1)$-bit adders.

Perhaps the most important work is that of Andraos and Ahmad [And88]. The authors derived the closed form expressions for the multiplicative inverses of the moduli based on the CRT, and *Property 2.10* is adopted to reduce the hardware complexity by circular shifts of the operand bits instead of the computationally intensive multiplications. To avoid the direct implementation of the CRT, both sides of the CRT equation are divided by 2^n and then modulo with respect to the product of the remaining moduli. According to the CRT,

$$X + MA(X) = \sum_{i=1}^{3} \hat{M}_i \left| \frac{1}{\hat{M}_i} \right|_{m_i} x_i \qquad (2\text{-}58)$$

where M is the dynamic range, and $A(X)$ is a nonnegative integer which is a function of X. The multiplicative inverses are derived as follows:

$$\left| \frac{1}{\hat{M}_1} \right|_{m_1} = \left| \frac{1}{2^n (2^n + 1)} \right|_{2^n - 1} = 2^{n-1}$$

$$\left| \frac{1}{\hat{M}_2} \right|_{m_2} = \left| \frac{1}{(2^{2n} - 1)} \right|_{2^n} = 2^n - 1$$

$$\left|\frac{1}{\hat{M}_3}\right|_{m_3} = \left|\frac{1}{2^n(2^n-1)}\right|_{2^n+1} = 2^{n-1}+1$$

Substituting the inverses into Equation (2-58) and dividing both sides of Equation (2-58) by 2^n, the integer part of the result modulo $2^{2n} - 1$ yields:

$$\begin{aligned}\left[\frac{X}{2^n}\right] &= \left|\left(2^{2n-1}+2^{n-1}\right)x_1 + \left(2^{2n}-2^n-1\right)x_2 + \left(2^{2n-1}+2^{n-1}-1\right)x_3\right|_{2^{2n}-1} \\ &= \left|C+B+A-X_3\right|_{2^{2n}-1}\end{aligned} \tag{2-59}$$

where

$$\begin{aligned}A &= \left|\left(2^{2n-1}+2^{n-1}\right)x_3\right|_{2^{2n}-1} \\ B &= \left|\left(2^{2n}-2^n-1\right)x_2\right|_{2^{2n}-1} \\ C &= \left|\left(2^{2n-1}+2^{n-1}\right)x_1\right|_{2^{2n}-1}\end{aligned} \tag{2-60}$$

So X can be calculated by

$$X = 2^n \cdot \left|C - X_3 + A + B\right|_{2^{2n}-1} + x_2 \tag{2-61}$$

S. J. Piestrak [Pie95] used a Multioperand Modular Adder (MOMA) and a 1's complement adder to implement Equation (2-59). According to the implementations of the CPA with EAC, two structures are proposed, which are the cost-effective (CE) and the high-speed (HS) versions.

Bhardwaj et al. [Bha98] proposed a high-speed realization of the reverse converter based on the HS version of Piestrak's architecture. The HS version of the $2n$-bit of CPA with EAC is implemented by two pairs of n-bit CPA performing the summations of $X + Y$ and $X + Y + 1$ in parallel, followed by the two n-bit 2-to-1 multiplexers. The data select signals of the multiplexers are generated from the carry outputs of the four CPAs. So the delay of CPA with EAC in [Bha98] is $t_{CPA}(n) + t_{MUX} + 2t_{NAND}$, which is smaller than the delay in [Pie95], while the area is almost the same.

Dhurkadas [Dhur98] rearranged the bit matrix of A, B, C and $-X_3$ so that the outputs of the bit orientation block comprise only three $2n$-bit strings. Thus the MOMA is

reduced to 3 inputs, and the CSA tree within the (3, $2^{2n} - 1$)-MOMA has only one level. The improvement is one FA delay time, and n FAs can be saved compared with the architecture of [Pie95]. Z. Wang *et al.* [Wan00] also proposed an algorithm to reduce the four-input MOMA to the 3-input MOMA, and the same improvement as [Dhur98] has also been achieved.

Y. Wang *et al.* [Wang02] proposed a new algorithm based on the New CRT and three efficient architectures for this triple moduli set. According to *Theorem 2.2*, Equation (2-34), for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, the binary number $X = (x_1, x_2, x_3)$ can be calculated as

$$X = x_2 + m_1 \left| k_1 (x_3 - x_2) + k_2 m_2 (x_1 - x_3) \right|_{m_2 m_3} \tag{2-62}$$

where $m_1 = 2^n$, $m_2 = 2^n + 1$ and $m_3 = 2^n - 1$.The terms k_1 and k_2 are defined by

$$\left| k_1 m_1 \right|_{m_2 m_3} = 1$$
$$\left| k_2 m_1 m_2 \right|_{m_3} = 1$$

The solutions of the above equations are $k_1 = 2^n$ and $k_2 = 2^{n-1}$. Based on the two multiplicative inverses, the authors proposed several architectures to implement (2-62) by using both $2n$-bit adder and n-bit adders.

Another three moduli of the $2^n \pm 1$ type is $\{2^n, 2^n - 1, 2^{n-1} - 1\}$. Hiasat *et al.* [Hia98] proposed an architecture of the reverse converter for this moduli set. This moduli set has two advantages over the three moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. First, for the latter moduli set, the residue with respect to modulus $2^n + 1$ requires $(n+1)$-bit to represent $2^n + 1$ states, which means that almost half of the states remain unused. Second, the multiplications modulo $2^n + 1$ is not as simple as multiplications modulo $2^n - 1$, thus the RAU for the latter moduli set will be more complex than those for the moduli set $\{2^n, 2^n - 1, 2^{n-1} - 1\}$. The CRT is used for the derivation of the algorithm, and the multiplicative inverses can be expressed in closed forms. However, the architecture proposed by the authors is only comparable to that of the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$.

Mathew *et al.* [Math00] proposed a new three moduli set $\{2^n, 2^n - 1, 2^{n+1} - 1\}$, which has utilized the full dynamic range, while keeping its reverse conversion efficient. The architecture of the reverse converter is based on the CRT and *Property 2.10*, and only three modulo $2^n - 1$ subtractors and one $(2n+1)$-bit regular subtractor are required.

Albeit efficiency, the three-moduli set has insufficient dynamic range and limited parallelism for some high performance and fault-tolerant systems. Four-moduli sets of the $2^n \pm 1$ type have emerged to be the good solutions to mitigate the situation. Bhardwaj *et al.* [Bha99] proposed the reverse converter for the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$, where n must be an odd number. The drawback of this four-moduli set is that there are two moduli in the form of $2^n + 1$, which requires $(n + 1)$-bits to represent $2^n + 1$ states, resulting in almost half of the states being unused.

Vinod and Premkumar [Vin00] proposed a more efficient four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and its corresponding reverse converter. Nevertheless, their reverse conversion algorithm is directly derived from the CRT without leveraging on the special number theoretic properties of the chosen moduli set. Thus, the hardware implementation of the reverse converter is not fully optimized.

Skavantzos and Stouraitis [Ska99b] proposed several grouped-moduli sets in the form of $\{2^a, m_1, m_2, \ldots, m_L, p_1, p_2, \ldots, p_M\}$, where $\prod_{i=1}^{L} m_i = (2^b - 1)$, and $\prod_{i=1}^{M} p_i = (2^c - 1)$. The reverse conversion is performed in a 2-level fashion. The first level consists of two converters operating in parallel, one of which combines the channels mod m_1, mod m_2, …, mod m_L, while the other combines the channels mod p_1, mod p_2, …, mod p_M. The residue channel 2^a will be combined together with either group of channels for m_i or p_i depending on which combination results in the simplest possible converter design. The second level of the reverse converter is a two-moduli set MRC converter combining the two results produced by the two first-level converters. It is noted that the grouped-moduli set should be chosen in such a way as to generate very efficient implementations for all the three converters involved in the reverse conversion. The authors proposed two four-moduli sets, $\{2^n, 2^{n-1} - 1, 2^n - 1, 2^n + 1\}$ and $\{2^{n+1}, 2^{n+1} - 1, 2^n - 1, 2^n + 1\}$ with even number n, one five-moduli set $\{2^{n+1}, 2^n - 1, 2^n + 1, 2^{n+1}$

$-1, 2^{n+1}+1\}$, valid for any integer n, and two seven-moduli set $\{2^{n+3}, 2^n-1, 2^n+1, 2^{n+2}-1, 2^{n+2}+1, 2^{n+2}+2^{(n+3)/2}+1, 2^{n+2}+2^{(n+3)/2}-1\}$ and $\{2^{n+2}, 2^n-1, 2^n+1, 2^{n+2}+2^{(n+1)/2}+1, 2^{n+2}+2^{(n+1)/2}-1, 2^{n+2}-1, 2^{n+2}+1\}$ with odd number n. In Chapter 3, we will present a more efficient reverse converter for these four-moduli sets using MRC, but the algorithm is somewhat different from that of [Ska99b]. The five-moduli set of [Ska99b] is not co-prime, resulting a reduced dynamic range. Although the reverse converters for the seven-moduli sets are efficient, two moduli of the sets are not of the $2^n \pm 1$ type. Therefore, the forward converters and the arithmetic units for the corresponding channels will not be area-time efficient.

Skavantzos [Ska98] proposed an efficient reverse converter for a five-moduli set $\{2^{n+1}, 2^n-1, 2^n+1, 2^n-2^{(n+1)/2}+1, 2^n+2^{(n+1)/2}+1\}$, which is valid for odd values of n, and the dynamic range is $(5n-1)$-bit. The conversion technique is based on integrating the CRT and the MRC, and *Property 2.10* is used to simplify the implementation. This algorithm involves four constant modular multiplications, and there are two moduli which are not of the $2^n \pm 1$ type. Therefore, this moduli set has the same disadvantages as the five-moduli set proposed in [Ska99a].

Conway and Nelson [Con03] proposed the scaled CRT based method for the moduli of the form $2^n \pm 1$, where the scaling factor is $M/2$. Then the scaled value of X can be obtained by:

$$X_f = \left\lfloor \sum_{i=1}^{n} y_i' \left| x_i \frac{1}{\hat{M}_i} \right|_{m_i} \right\rfloor - 2r, \tag{2-63}$$

where $y_i' = 2/m_i$, and r is a nonnegative number to ensure that X_f has a value in the range [0, 2]. Let y_i' be expressed by a b-bit finite number, $\hat{y}_i'$ as follows:

$$\hat{y}_i' = \lfloor 2^b y_i' \rfloor 2^{-b}. \tag{2-64}$$

Consider the moduli of the form $2^n - 1$, Equation (2-64) can be expressed as follows:

$$\hat{y}_i^{'} = \sum_{i=1}^{C_i} 2^{-jn+1}, \tag{2-65}$$

where C_i is the smallest integer that is greater than $\frac{b+1-\log_2(2^n-1)}{n}$. Equations (2-63) and (2-65) eliminate the multiplications when moduli of the form $2^n - 1$ are used.

Similar expressions of $\hat{y}_i^{'}$ for the moduli of the forms $2^n + 1$ and 2^n can be found, and using these expressions will allow the coefficient multiplications to be replaced by simpler shifting and additions. Comparing with the normal CRT-based method and the two-level method for conjugate moduli set in [Ska99a], this method can save the terms to be added, resulting in smaller area and conversion latency.

2.3 Residue Arithmetic Unit

It is conjectured that RNS is not efficient for the general purpose computation, such as division, magnitude comparison, scaling and so on. Fortunately, most of the algorithms encountered in real time DSP are based on multiplication and addition operations. With RNS, binary operations such as addition, subtraction and multiplications for large operands are replaced by a group of corresponding modular operations with smaller operands. Unlike the forward and reverse converters, modulo operations are ineluctable in the RAU and they usually operate in iterations to implement specific applications. Therefore, their performances are critical for a given RNS.

2.3.1 Review of Modular Adders

There are three ways to implement modular addition. One is to use lookup-tables [Ban74] [Jen77], usually implemented on ROMs. The second approach is based on combinatorial implementation [Tay85] [Bay87] [Elle90] [Dug92] [Pie94b] [Zimm99] [Hia02]. The third is a hybrid method [Jen78] [Sod83]. The fundamental drawback of lookup-table based architecture is the exponential growth of table size with the input word length. Therefore, they are inefficient for modulo operations with large moduli. The structures for combinatorial modular operation vary for different moduli. It is

generally agreed that modular additions with general moduli are far more complicated than those with special moduli. A two-input modular adder is the most fundamental computation element. Modular subtraction can be performed by adders using the additive inverse property [Sza67]. According to [Zimm99], binary addition can be formulated as a prefix problem. The same parallel-prefix structures are also applicable to the modular addition. Modular adders with parallel-prefix architectures are called the parallel-prefix modular adders [Dimi03] [Ver02].

Multioperand modular addition is implicitly employed in both the modular multiplication and forward converters for the RNS. MOMA has more than two operands, and often comprises a CSA tree and a two-input modular adder. For simplicity, K-input n-bit modulo M MOMA is denoted as (K, M)-MOMA, which performs the following calculation:

$$U = \left| \sum_{i=1}^{K} A_i \right|_m . \tag{2-66}$$

MOMA can be realized by using two-operand modular adders both sequentially and in tree structure. The sequential implementation of MOMA using a two-operand modular adder and a partial sum register. If one addition is performed per clock cycle, $K - 1$ cycles are required for a (K, M)-MOMA. For higher speed, a tree of two-operand modular adders can be used. Such a tree of two-operand adders requires $K - 1$ modular adders, and the number of levels of carry propagation adders is $O(\log_2 K)$.

The use of two-operand modular adders in multi-operand adder tree is both hardware intensive and slow due to the carry-chain. In binary multi-operand addition, CSAs are usually used for the design of fast tree adders. A CSA tree can reduce K n-bit binary numbers into two numbers having the same sum in $O(\log n)$ levels. The sum and carry outputs of the binary CSA tree are at most ($n + \log_2 K$) bits. Modular CSAs are also used in MOMA. Due to the modular addition, the partial sum after each CSA can be reduced, which results in a more regular CSA tree than the CSA tree for the binary multi-operand adder. For general moduli, reducing the output of the modular CSA will incur additional logic circuits, which in terms increase the area cost and the delay [Koc90] [Alia96].

For special moduli, such as those in the form of $2^n \pm 1$, due to the special properties of $\left|2^n\right|_{2^n-1} = 1$ and $\left|2^n\right|_{2^n+1} = -1$, the corresponding modular CSA tree will have a regular structure. The standard implementation of a modulo $2^n - 1$ adder uses a conventional binary adder with its carry output connected to its carry input to achieve the EAC operation. Bhardwaj *et al.* [Bha98] proposed a high-speed implementation of the CPA with EAC. The new architecture for $2n$-bit CPA with EAC adopts four n-bit binary CPA working in parallel, with different carry inputs, to overcome the $2n$-bit carry propagation barrier in the common high-speed implementation for the CPA with EAC [Pie95]. Efstathiou *et al.* [Efst94] proposed some modulo $2^n - 1$ adders based on the carry look-ahead addition algorithm. One-level and two-level CLA modulo $2^n - 1$ adders with both double and single representations of zero were illustrated. Although the CLA-like modular adder proposed in [Hia02] is meant for the general moduli, it can also be used for special moduli, such as modulus $2^n - 1$. The parallel-prefix structure for modulo $2^n - 1$ adder was proposed in [Zim98], where an additional prefix level is added to propagate the EAC to the sum output. Kalampoukas et al. [Kal00] proposed another high-speed parallel-prefix modulo $2^n - 1$ adders, where the idea of recirculating the generate and propagate signals, instead of the traditional EAC approach, is applied. In [Dim03], the authors extended this idea and proposed a new family of parallel-prefix modulo $2^n - 1$ adders. The performances of these new adders, in terms of area and/or operation speed, outperform all previously reported ones.

For 2-input modulo $2^n + 1$ adder, there are generally two methods, based on two different number representations of the operands. One operates in binary format while the other exploits the diminished-one system. For normal binary approach, the CLA-like modular adder proposed by Hiasat [Hia02] for the general moduli can be adopted to implement the modular adders for special modulus $2^n + 1$ efficiently. Due to the property of $\left|2^n\right|_{2^n+1} = -1$, the CEAC instead of EAC is used for the binary modulo $2^n + 1$ adder [Zim99]. The diminished-one system is more often used for modulo $2^n + 1$ arithmetic. Both CLA-like and parallel-prefix based modulo adder $2^n + 1$ are proposed in [Ver02], where the parallel-prefix based adder is more efficient than those of

[Zim99]. By using select-prefix blocks, the diminished-one modulo $2^n + 1$ adders can achieve the best $A \times T^2$ product in the literature according to [Efst03].

2.3.2 Modular Multipliers

Modular multiplication is a fundamental operation of all RNS related applications and some cryptographic systems. There are many variants and structures of modular multipliers [Sod80][Tay81][Mont85][Rad92][Baja98][MA98][Efst05]. According to the modulus, two general types of modular multipliers exist. One is the multiplier for general modulus, which computes $|A \times B|_m$ for any modulus. The other is the modular multiplier for special form of modulus, where the modulus has either a special mathematical expression or some important number properties. According to the implementation methods, there are ROM-based modular multipliers for small moduli, and arithmetic components based modular multipliers. According to the algorithms, there are index calculus based modular multipliers [Rad92], binary adder/multiplier based modular multipliers [Stou93] [Hia00], Quarter-square multipliers [Sod80] [Tay81], Montgomery modular multipliers [Mont85], etc.. According to the radix, there are radix-2, radix-4, high-radix and redundant modular multipliers. According to the structure, there are non-systolic and systolic array modular multipliers. Depending on whether scaling is required, there are non-scaling modular multipliers and scaling modular multipliers. These classifications are not very strict, and at times fuzzy as there exist variants that fit two or more attributes. For example, some index calculus modular multipliers are implemented based on ROM, while others are best implemented with elementary arithmetic components.

In many cases, modular multiplications can be performed by addressing a look-up table, stored in RAM or ROM [Jull78] [Jull90] [Rad92]. The look-up table based multipliers are only suitable for small moduli. There has been a proliferation of memoryless implementation of modular multipliers. Soderstrand and Vernia [Sod80] proposed a square law modular multiplier, which is also known as Quarter Square multiplier. The basic Quarter Square modular multiplier is further improved by Taylor [Tay81]. Alia and Martinelli [Alia91] proposed a structure for general modular multiplier using binary multipliers and adders. Stouraitis et al. [Stou93] proposed FA-

based architectures for the RNS multiply-addition operations. Elleithy and Bayoumi [Elle95] proposed a systolic architecture for general moduli multiplication using their $\theta(\log n)$ modular adders. Hiasat [Hia00] also proposed an area-time efficient structure for the general moduli multiplications. Montgomery's modular multiplication algorithm is widely used in cryptographic systems and the modulus can be of any format [Mont85] [Eld93]. An RNS Montgomery modular multiplication algorithm is introduced by Bajard et al. [Baja98] [Baja01] by adapting Montgomery's method to the base extension in RNS, so that RNS implementation is now introduced into cryptography.

The modular multiplications for special moduli are of special interest from both the RNS and application points of view. When the modulus is a prime, the modular multiplication can be performed by a method called index calculus, where the multiplication is achieved by the modular addition of the indexes [Sza67] [Jull80]. Dugdale [Dug94] proposed a technique for modular multiplication using factored decomposition where the modulus is a non-prime integer. Skavantzos [Ska92] proposed a modulo $2^n - 1$ multiplier using look-up tables, where the size of the table is reduced at the expense of some additions and squaring operations. Wang et al. [Wan96] proposed a technique for modulo $2^n - 1$ multiplication using a Wallace FA tree and a modular adder, which is also suitable for large modular multiplication. Similar to the realization of modulo $2^n + 1$ addition, there are also two systems for modulo $2^n + 1$ multiplication, i.e., normal representation and diminished-one representation. Curiger et al. [Curi91] reviewed different architectures for multiplication modulo a Fermat number. Their conclusion is as follows: when $n \leq 8$, the look-up table based methods, such as quarter-squared method [Tay81] and index calculus method [Jull80] are competitive; for large n, arithmetic components based methods are superior.

With normal binary representation, Wrzyszcz and Milford [Wrzy93] proposed an efficient modulo $2^n + 1$ multiplier based on the periodic properties of the modulus $2^n + 1$. Chang et al. [Chan85] proposed an improved diminished-one multiplication method over the one proposed by Leibowitz's [Leib76]. The diminished-one multiplication method is further enhanced by Benaissa et al. [Ben88a] [Bena88b] by

removing the initial states of the structure in [Chan85], and the wordlength of the operands is only n-bit. Wang et al. [Wan95] proposed a structure for diminished-one multiplication using a regular CSA with CEAC tree followed by a 2-operand modulo $2^n + 1$ adder. Ma [Ma98] used two modular CSA for the modulo reduction and Booth's recoding method to reduce the number of partial products. Booth's recoding is also used by Zimmerman [Zimm99] for modulo $2^n + 1$ multiplication in both the diminished-one and normal representations, and the structure of [Wan95] has also been further improved by removing the correction term Z. The latest contribution to modulo $2^n + 1$ multiplication is due to [Efst05], where a new formulation of partial products is proposed, and better performance can be achieved.

Typically, a RNS has at least two different moduli, and modular arithmetic operations are performed independently over these residue channels. If the design of different modulo multipliers for all the residue channels is considered as a whole, more efficient area-time structures can be achieved by exploiting the common circuits or other number properties of the moduli sets. Typically, RNS multiplier refers to one that applies to all the residue channels. Hiasat proposed a RNS multiplier which is tailored for moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ [Hia92]. Claudio et al. [Clau95] proposed a new RNS multiplier by using pseudo-RNS, and the moduli must be odd numbers. Paliouras *et al.* [Pali99] [Pali01] proposed low-complexity combinatorial RNS multipliers by sharing hardwares among different residue channels. However, sharing of hardware resources among residue channels will inevitably limit the parallelism of the RNS.

2.4 Conclusions

This chapter introduces the background of RNS. The topics surveyed include the fundamentals of residue arithmetic, the forward converters, the reverse converters, and the RAU. Due to the nature of RNS research, specific emphasis has been put on the literature review of the reverse converters for the special moduli sets. For the reverse converters, classical and contemporary techniques such as CRTs, MRC, and New CRTs have been identified, and a comprehensive study of the reverse converters for different types of special moduli sets have been carried out. In the review of

typical RAU of RNS, we presented the past and recent work on the modular adders and modular multipliers. The survey covers not only those based on the general moduli, but also those on the special moduli, especially the $2^n \pm 1$ type. The arithmetic units considered in the review are MOMAs, modulo $2^n \pm 1$ adders, and modular multipliers.

Although there exist a number of reverse converters, very few have been physically synthesized to verify their performances from the VLSI perspective. In particular, no report has scrutinized the power dissipation issues to provide a better decision trade-off in terms of power efficiency governing the choice of moduli sets. Our literature survey also reveals that most literatures provide only the area and time complexity analysis and comparisons at the architecture level for the reverse converters alone, or the forward converters alone or for basic modules such as modulo adders and modulo multipliers evaluated independently. We argue that such evaluations are unlikely to convincingly infer the actual VLSI performances of the overall system. Since residue arithmetic operations are usually performed iteratively in applications that merit the use of RNS, the types of moduli sets, the cardinality of the moduli sets, the types of modular operations and the number of operations of RAUs should be taken into consideration for the RNS systems.

While it is conjectured that the supersets based on moduli of the $2^n \pm 1$ types are comparatively more efficient in all constituent blocks of an RNS than other special moduli sets, there are not many high cardinality supersets, and some of the existing supersets have not been fully evaluated in terms of their area-delay efficiency. This must be undertaken in order to scrutinize their merits and shortcomings before more efficient converters can be proposed and suitable extensions for high cardinality moduli sets can be considered. Due to their special number properties of triple moduli set and their balanceness, it makes sense to explore new extended sets of the triple moduli to facilitate VLSI efficient RNS implementation. In this perspective, it is important to capitalize on the untapped power of the New CRT to devise VLSI efficient reverse converter, particularly for the special moduli supersets that are extended from the well studied triple moduli set.

In the following chapters, we will propose new reverse converters for the existing triple moduli based supersets as well as for the proposed new high-cardinality moduli supersets. A new unified approach to the design of a critical modular operation will also be proposed. Complete RNS's constructed from these triple moduli based supersets and the triple moduli set itself will all be synthesized and optimized in a platform to emulate generic inner product step processor application. The analysis will provide an insight into their trade-offs in VLSI metrics such as area, delay and power consumptions under different dynamic ranges, and different number of iterations of modulo arithmetic called for by the application.

Chapter 3

New Area-Time Efficient Reverse Converters for Two Akin Four-Moduli Supersets

3.1 Introduction

We will present in this chapter new reverse conversion algorithms for two four-moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ for even values of n. A large portion of this chapter has been published in the *IEE Proceedings of Computers and Digital Techniques* [Cao05a] and presented at the 2003 IEEE International Symposium on Circuits and Systems [Cao03c] and at the 2003 International Symposium on Image and Signal Processing and Analysis [Cao03d].

In Sections 2.1 and 2.2 of Chapter 2, we have discussed several factors that limit the diffusion of RNS-based processors. We acknowledged the necessity to address the overhead incurred in the conversions into and out of the RNS due to the peripheral interfaces of most digital systems today are based on the weighted number system. The forward conversion from binary number to residues is conceivably simple. Efficient architectures have been devised for residue generator based on special moduli of the *2^n-type* (i.e., expressible in the form 2^n or $2^n \pm 1$) [Bi88] [Pie94a] [Pou94a] [Pou94b]. It is the reverse conversion from residues to binary number that causes a significantly higher hurdle that offsets the performance gained in the RNS.

Special moduli sets have been used extensively to reduce the hardware complexity in the implementation of reverse converters. Among which the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ has gained unprecedented popularity by virtue of its inherent number theoretic properties in CRT algorithm. Many reverse converters have been proposed for this triple-moduli set [Ber85] [And88] [Ibra88] [Pie94b] [Pou94c] [Pie95] [Gall97]

[Bha98] [Con99] [Wan00] [Wang02]. Its forward converters and the RAU involve modular adders and modular multipliers in modulo $2^n - 1$ and $2^n + 1$. Due to the special number theoretical properties of the moduli of $2^n \pm 1$ type, the modular operations for these moduli are more efficient than those for general moduli [Bay87] [Ska89] [Alia91] [Ska92] [Efst94] [Wan96] [Kal00] [Hia02] [Ver02] [Dimi03] [Efst05]. Albeit efficiency, the celebrated triple-moduli set has insufficient dynamic range and limited parallelism for some high performance, fault-tolerant applications and cryptosystems. Four-moduli sets of the $2^n \pm 1$ type have emerged to be the good solutions to mitigate this situation.

Several important four-moduli supersets have been proposed on the basis of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, such as $S_4^1 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $S_4^2 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ which are only valid for even values of n, and $S_4^3 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$, which is valid only for odd n. The drawback of S_4^3 is that there are two moduli in the form of $2^n + 1$, which requires $n + 1$ bits to represent $2^n + 1$ states, resulting in almost half of the states being unused; furthermore, the modulo $2^n + 1$ operations are more complex than those for modulo $2^n - 1$, which means that the RAUs for modulo $2^n - 1$ can be implemented and computed more efficiently than those for modulo $2^n + 1$.

In this chapter, we first propose several efficient reverse converters for the four-moduli set S_4^1 with even number n. Our proposed architectures embody an efficient reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ proposed either in [Wan00] or in [Wang02] in the first stage, and MRC technique is applied to the resulting residue and the remaining residue for the reverse conversion. The new architectures are more efficient than the reverse converter proposed in [Vin00] in terms of the area and delay complexities even for the most demanding DSP applications. Similar techniques can also be applied to the reverse conversion problem of an akin four-moduli set S_4^2 with a dynamic range of $4n - 1$ bits, where n must be even. For a given application with a dynamic range slightly larger than $4n + 1$ bits for the first four-moduli set, the second four-moduli set provides a better choice, because not only does it minimize the waste of dynamic range, but it is also equally efficient as the reverse converters for the first four-moduli set in terms of the area and delay

complexities. Skavantzos and Stouraitis [Ska99b] also proposed a reverse converter for the four-moduli set, S_4^2. Their method is based on the decomposition of the moduli set into two levels. The authors first arranged the moduli set as $\{2^n, 2^{n-1} - 1, 2^n - 1, 2^n + 1\}$, where in the first level, two reverse converters are used for two two-moduli sets, $\{2^n, 2^{n-1} - 1\}$ and $\{2^n - 1, 2^n + 1\}$, and in the second level, the MRC method is used for the two-moduli set $\{2^n(2^{n-1} - 1), 2^{2n} - 1\}$. On the other hand, the first level of our proposed structure only consists of one reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, while in the second level, the MRC is applied to the two-moduli set $\{2^n(2^{2n} - 1), 2^{n-1} - 1\}$. If we use the most efficient solution proposed in [Wang02], the hardware cost of the first level of our proposed reverse converter is more economical than that of [Ska99b] without compromising the speed of the first level. By virtue of the reverse conversion of the triple-moduli set, the second level MRC of our proposed method reduces the hardware complexity more than that for [Ska99b], resulting in a simpler implementation. In the later sections, we will show that our algorithm is more efficient than that of [Ska99b].

3.2 Reverse Conversion of $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$

Consider the four-moduli set $S_4^1 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$, where n is any even natural number. Let the RNS representation of the integer X be (x_1, x_2, x_3, x_4), and the binary representations of the residues be as follows: $x_1 = (X_{1,n-1}X_{1,n-2} \ldots X_{1,0})_2$, $x_2 = (X_{2,n-1}X_{2,n-2} \ldots X_{2,0})_2$, $x_3 = (X_{3,n}X_{3,n-1} \ldots X_{3,0})_2$, and $x_4 = (X_{4,n}X_{4,n-1} \ldots X_{4,0})_2$. Our algorithm for the new reverse conversion begins with the recovery of the integer $X^{(1)} = (x_1, x_2, x_3)$ corresponding to the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ via an efficient reverse converter. Next, the MRC method is applied to calculate the final integer $X = (X^{(1)}, x_4)$ corresponding to the moduli set $\{2^n(2^{2n} - 1), 2^{n+1} - 1\}$. The following property and lemmas are needed for our new algorithm.

Lemma 3.1: The multiplicative inverse of $(5 \cdot 2^{n-2} - 1)$ modulo $(2^{n+1} - 1)$ is

$$\left| \frac{1}{5 \cdot 2^{n-2} - 1} \right|_{2^{n+1}-1} = \frac{1}{3}\left(2^{n+2} - 10\right), \tag{3-1}$$

where n is any even number.

Proof: Using the fact that $\left|2^{n+1}\right|_{2^{n+1}-1} = 1$,

$$\left|\frac{1}{3}\left(2^{n+2}-10\right)\cdot\left(5\cdot 2^{n-2}-1\right)\right|_{2^{n+1}-1} = \left|\frac{1}{3}\left(5\cdot 2^{2n}-2^{n+2}-50\cdot 2^{n-2}+10\right)\right|_{2^{n+1}-1}$$

$$= \left|\frac{1}{3}\left(5\cdot 2^{n-1}-2-25\cdot 2^{n-1}+10\right)\right|_{2^{n+1}-1} = \left|\frac{1}{3}\left(8-5\cdot 2^{n+1}\right)\right|_{2^{n+1}-1} = \left|\frac{1}{3}\left(8-5\right)\right|_{2^{n+1}-1} = 1$$

□

Lemma 3.2: The multiplicative inverse of 3 modulo $(2^{n+1} - 1)$ is

$$\left|\frac{1}{3}\right|_{2^{n+1}-1} = \frac{1}{3}\left(2^{n+2}-1\right) = \sum_{i=0}^{n/2} 2^{2i}\ , \tag{3-2}$$

where n is any even number.

Proof:

$$\left|3\cdot\left(\frac{1}{3}\left(2^{n+2}-1\right)\right)\right|_{2^{n+1}-1} = \left|2^{n+2}-1\right|_{2^{n+1}-1} = \left|2\left(2^{n+1}-1\right)+1\right|_{2^{n+1}-1} = 1$$

□

3.2.1 Four-Stage Reverse Converters

Based on *Property 2.10* and *Property 2.11*, *Lemma 3.1* and *Lemma 3.2*, a new algorithm can be developed for the reverse conversion. First, consider the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ and the integer $X^{(1)} = (x_1, x_2, x_3)$, where x_1, x_2 and x_3 are the residues of $X^{(1)}$ with respect to the above triple-moduli set. A triple-moduli set reverse converter [Wang02] is required to obtain $X^{(1)}$ from its residues as follows:

$$X^{(1)} = x_2 + 2^n Y \tag{3-3}$$

where Y is a $2n$-bit number. Further, let

$$Y = Y_1 + 2^n Y_2 = \left(Y_{2,n-1}\cdots Y_{2,0} Y_{1,n-1}\cdots Y_{1,0}\right)_2 \tag{3-4}$$

where Y_1 and Y_2 are the least significant n-bit and the most significant n-bit of Y, respectively.

Next, consider the two-moduli set $\{2^n(2^{2n}-1), 2^{n+1}-1\}$ and $X = (X^{(1)}, x_4)$. Using MRC Equation (2-30), X can be calculated by

$$X = X^{(1)} + 2^n\left(2^{2n}-1\right)\left|k_0\left(x_4 - X^{(1)}\right)\right|_{2^{n+1}-1} \tag{3-5}$$

where

$$\left|k_0 \cdot 2^n\left(2^{2n}-1\right)\right|_{2^{n+1}-1} = 1. \tag{3-6}$$

It is difficult to calculate k_0 directly from Equation (3-6). However, the left-hand side of Equation (3-6) can be simplified as follows:

$$\begin{aligned}
&\left|k_0 2^n\left(2^{2n}-1\right)\right|_{2^{n+1}-1} = \left|k_0 2^n\left|\left(2^{n-1}\left(2^{n+1}-1\right)+2^{n-1}-1\right)\right|_{2^{n+1}-1}\right|_{2^{n+1}-1} \\
&= \left|k_0 2^n\left(2^{n-1}-1\right)\right|_{2^{n+1}-1} = \left|k_0\left|2^{n-2}\left(2^{n+1}-1\right)+2^{n-2}-2^n\right|_{2^{n+1}-1}\right|_{2^{n+1}-1} \\
&= \left|k_0\left(2^{n-2}-2^n\right)\right|_{2^{n+1}-1} = \left|k_0\left(2^{n-2}-2^n+2^{n+1}-1\right)\right|_{2^{n+1}-1} \\
&= \left|k_0\left(5\cdot 2^{n-2}-1\right)\right|_{2^{n+1}-1}
\end{aligned}$$

Therefore, Equation (3-6) becomes:

$$\left|k_0\left(5\cdot 2^{n-2}-1\right)\right|_{2^{n+1}-1} = 1$$

According to *Lemma 3.1*, $k_0 = \frac{1}{3}\left(2^{n+2}-10\right)$. Substituting it into Equation (3-5),

$$X = X^{(1)} + 2^n\left(2^{2n}-1\right)\left|\frac{1}{3}\left(2^{n+2}-10\right)\left(x_4 - X^{(1)}\right)\right|_{2^{n+1}-1} = X^{(1)} + 2^n\left(2^{2n}-1\right)Z \tag{3-7}$$

where Z is a $(n+1)$-bit number defined as follows:

$$Z = \left|\frac{1}{3}\left(2^{n+2}-10\right)\left(x_4 - X^{(1)}\right)\right|_{2^{n+1}-1} = \left|\frac{1}{3}\left(2^{n+2}x_4 - 2^{n+2}X^{(1)} - 10x_4 + 10X^{(1)}\right)\right|_{2^{n+1}-1}$$

According to Equations (3-3) and (3-4), Z can be calculated by

$$\begin{aligned} Z &= \left|\frac{1}{3}\left(2^{n+2}x_4 - 2^{n+2}\left(x_2 + 2^n Y\right) - 10x_4 + 10\left(x_2 + 2^n Y\right)\right)\right|_{2^{n+1}-1} \\ &= \left|\frac{1}{3}\left(2^{n+2}x_4 - 2^{n+2}\left(x_2 + 2^n Y\right) - 10x_4 + 10\left(x_2 + 2^n Y\right)\right) + 2^{n+1}Y\left(2^{n+1}-1\right)\right|_{2^{n+1}-1} \\ &= \left|\frac{1}{3}\left(2^{n+2}x_4 - 10x_4 - 2^{n+2}x_2 + 10x_2 + 2^{n+3}Y\right)\right|_{2^{n+1}-1} \\ &= \left|\frac{1}{3}\left(H + I + J\right)\right|_{2^{n+1}-1} = \left|\left|\frac{1}{3}\right|_{2^{n+1}-1} \left|H + I + J\right|_{2^{n+1}-1}\right|_{2^{n+1}-1} = \left|S \cdot Q\right|_{2^{n+1}-1} \end{aligned} \quad (3\text{-}8)$$

where

$$\begin{aligned} H &= 2^{n+2}x_4 - 10x_4 \\ I &= -2^{n+2}x_2 + 10x_2 \\ J &= 2^{n+3}Y \end{aligned} \quad (3\text{-}9)$$

and

$$\begin{aligned} S &= \left|\frac{1}{3}\right|_{2^{n+1}-1} \\ Q &= \left|H + I + J\right|_{2^{n+1}-1} \end{aligned} \quad (3\text{-}10)$$

In Equations (3-10), the term S is given by *Lemma 3.2*. According to *Property 2.11*, the term Q can be simplified as follows:

$$\begin{aligned} \left|H\right|_{2^{n+1}-1} &= \left|2^{n+2}x_4 - 10x_4\right|_{2^{n+1}-1} = \left|2\left(2^{n+1}-1\right)x_4 - 2^3 \cdot x_4\right|_{2^{n+1}-1} = \left|-2^3 \cdot x_4\right|_{2^{n+1}-1} \\ &= \left|2^{n+1} - 1 - CLS_{n+1}(x_4, 3)\right|_{2^{n+1}-1} = \left(\overline{X}_{4,n-3} \cdots \overline{X}_{4,0}\overline{X}_{4,n}\overline{X}_{4,n-1}\overline{X}_{4,n-2}\right)_2 \end{aligned} \quad (3\text{-}11)$$

$$\begin{aligned} \left|I\right|_{2^{n+1}-1} &= \left|-2^{n+2}x_2 + 10 \cdot x_2\right|_{2^{n+1}-1} = \left|-2\left(2^{n+1}-1\right)x_2 + 2^3 \cdot x_2\right|_{2^{n+1}-1} \\ &= \left|2^3 x_2\right|_{2^{n+1}-1} = CLS_{n+1}(x_2, 3) = \left(X_{2,n-3} \cdots X_{2,0}0X_{2,n-1}X_{2,n-2}\right)_2 \end{aligned} \quad (3\text{-}12)$$

$$\begin{aligned} \left|J\right|_{2^{n+1}-1} &= \left|2^{n+3}Y\right|_{2^{n+1}-1} = \left|2^2\left(2^{n+1}-1\right)Y + 2^2 Y\right|_{2^{n+1}-1} = \left|2^2\left(Y_1 + 2^n Y_2\right)\right|_{2^{n+1}-1} \\ &= \left|2^2 Y_1 + 2^{n+2}Y_2\right|_{2^{n+1}-1} = \left|CLS_{n+1}(Y_1, 2) + 2\left(2^{n+1}-1\right)Y_2 + 2Y_2\right|_{2^{n+1}-1} \\ &= \left|CLS_{n+1}(Y_1, 2) + CLS_{n+1}(Y_2, 1)\right|_{2^{n+1}-1} = \left|J_1 + J_2\right|_{2^{n+1}-1} \end{aligned} \quad (3\text{-}13)$$

In Equation (3-13), J_1 and J_2 are defined by:

$$J_1 = CLS_{n+1}(Y_1, 2) = (Y_{1,n-2} \cdots Y_{1,0} 0 Y_{1,n-1})_2 \tag{3-14}$$

$$J_2 = CLS_{n+1}(Y_2, 1) = (Y_{2,n-1} \cdots Y_{2,0} 0)_2 \tag{3-15}$$

Thus, the term Q can be re-expressed as follows:

$$Q = |H + I + J_1 + J_2|_{2^{n+1}-1}, \tag{3-16}$$

where the terms H, I, J_1 and J_2 are $(n+1)$-bit strings defined by Equations (3-11) to (3-15). According to Equation (3-16), Q can be easily calculated by only one $(4, 2^{n+1} - 1)$-MOMA. Figure 3.1(a) shows the hardware scheme.

According to *Lemma 3.2* and Equations (3-9) to (3-16), Equation (3-8) becomes:

$$Z = |S \cdot Q|_{2^{n+1}-1} = \left|\frac{1}{3}(2^{n+2} - 1)Q\right|_{2^{n+1}-1}$$

Thus, according to *Lemma 3.2* and *Property 2.10*, we have

$$\begin{aligned} Z &= |S \cdot Q|_{2^{n+1}-1} = \left|(2^0 + 2^2 + 2^4 + \cdots + 2^n)Q\right|_{2^{n+1}-1} \\ &= \left|CLS_{n+1}(Q,0) + CLS_{n+1}(Q,2) + \cdots + CLS_{n+1}(Q,n)\right|_{2^{n+1}-1} \\ &= \left|Q^{(0)} + Q^{(2)} + \cdots + Q^{(n)}\right|_{2^{n+1}-1} \end{aligned} \tag{3-17}$$

In Equation (3-17), the terms $Q^{(0)}$, $Q^{(2)}$,..., $Q^{(n)}$ are $(n+1)$-bit circularly left-shifted version of Q. So Z can be easily obtained by one $(n/2+1, 2^{n+1} - 1)$ MOMA. Figure 3.1(b) shows the hardware scheme for the calculation of Z when $n = 8$.

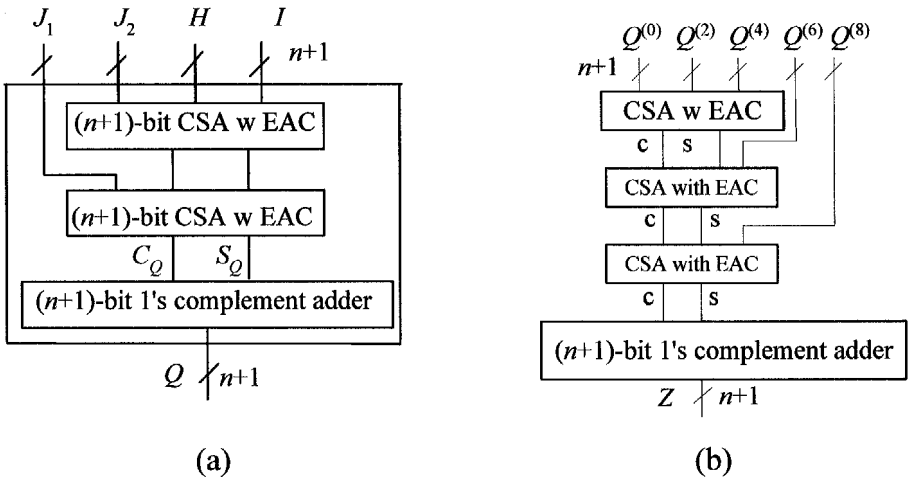


Figure 3.1 *Hardware scheme for the calculation of Q and Z for the moduli set* S_4^1

The final X can be calculated from Equation (3-5) as follows:

$$\begin{aligned} X &= X^{(1)} + 2^n\left(2^{2n}-1\right)Z = x_2 + 2^n Y + 2^{3n} Z - 2^n Z \\ &= x_2 + 2^n\left(Y + 2^{2n} Z - Z\right) = x_2 + 2^n\left(Z^{(1)} - Z\right) = x_2 + 2^n T \end{aligned} \tag{3-18}$$

where

$$\begin{aligned} Z^{(1)} &= 2^{2n} Z + 2^n Y_2 + Y_1 = \left(Z_n \cdots Z_0 Y_{2,n-1} \cdots Y_{2,0} Y_{1,n-1} \cdots Y_{1,0}\right)_2 \\ T &= Z^{(1)} - Z \end{aligned} \tag{3-19}$$

According to Equations (3-18), one $(3n+1)$-bit subtractor is needed to calculate T, and the resulting $(3n+1)$-bit T is concatenated to the n-bit x_2 to obtain the final value of X. Figure 3.2 shows the hardware implementation devised directly from the new algorithm. The first stage is an efficient triple-moduli set reverse converter. The second stage is the calculation of Q, which involves only one $(4, 2^{n+1} - 1)$-MOMA. The third stage is the calculation of Z, which is realized by one $(n/2+1, 2^{n+1} - 1)$-MOMA. The final stage is a $(3n+1)$-bit binary subtractor. Since these four stages are sequential, the total delay of the conversion is the sum of the delay of each individual stage. Nevertheless, this structure lends itself well for pipelining, and high throughput rate can be easily achieved by latches insertion between stages.

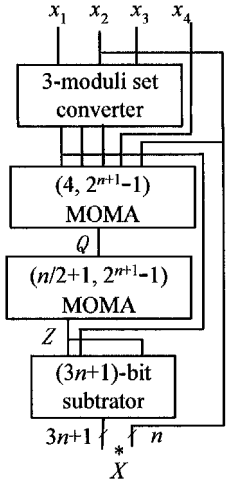


Figure 3.2 *The four-stage reverse converter for the moduli set* S_4^1

3.2.2 Three-Stage Reverse Converters

As each of the middle two stages of the four-stage architecture includes one MOMA, the conversion latency in the combinational circuit implementation can be significantly accelerated by removing the inter-stage MOMA. In fact, the CPAs in the second and the third stages can be merged into one CPA with EAC. The basis for their merging is provided below.

In Figure 3.1(a), if the (n+1)-bit 1's complement adder is saved, the value of Q will be kept in stored carry format as follows:

$$Q = \left|J_1 + J_2 + H + I\right|_{2^{n+1}-1} = C_Q + S_Q \tag{3-20}$$

where C_Q and S_Q are the (n+1)-bit carry and sum outputs of Q, respectively. By removing the CPA of the second stage with the aid of *Property 2.10*, Equation (3-17) has to be adjusted accordingly:

$$\begin{aligned} Z &= \left|S \cdot Q\right|_{2^{n+1}-1} = \left|\left(2^0 + 2^2 + 2^4 + \cdots + 2^n\right)Q\right|_{2^{n+1}-1} \\ &= \left|\left(2^0 + 2^2 + 2^4 + \cdots + 2^n\right)\left(C_Q + S_Q\right)\right|_{2^{n+1}-1} \\ &= \left|CLS_{n+1}\left(C_Q,0\right) + CLS_{n+1}\left(C_Q,2\right) + \cdots + CLS_{n+1}\left(C_Q,n\right) + CLS_{n+1}\left(S_Q,0\right) + \cdots + CLS_{n+1}\left(S_Q,n\right)\right|_{2^{n+1}-1} \\ &= \left|C_Q^{(0)} + C_Q^{(2)} + C_Q^{(4)} + \cdots + C_Q^{(n)} + S_Q^{(0)} + S_Q^{(2)} + S_Q^{(4)} + \cdots + S_Q^{(n)}\right|_{2^{n+1}-1} \end{aligned} \tag{3-21}$$

In Equation (3-21), the terms $S_Q^{(n)}$, $C_Q^{(n)}$ are (n+1)-bit circularly left-shifted versions of S_Q and C_Q, and one (n+2, $2^{n+1} - 1$) MOMA is required for the calculation of Z. The architecture of the second stage for the calculation of Z is shown in Figure 3.3(a). The block Bit Orientation 2 performs the circular left shift operations required by Equation (3-21). After Z has been obtained, the remaining work is the same as that presented in the previous section. Figure 3.3(b) shows the architecture of the three-stage reverse converter. The first and third stages are the same as the four-stage architecture. Since the interim result of Q is in the carry-save format, the number of levels of the CSA tree in the second stage is doubled. Therefore, the hardware cost (FAs) of the three-stage will be larger than that of the four-stage architecture, while the conversion delay

is less than that of the four-stage architecture.

Example 3.1: If the required dynamic range is 16 bits, then $n = 4$. The moduli set is {15, 16, 17, 31}. Let $X = (1, 2, 1, 30)$. Using the reverse converter for the moduli set {15, 16, 17}, we can obtain $Y = (1110\ 1111)_2$, where $Y_1=(1111)_2$, and $Y_2 = (1110)_2$. According to Equations (3-11) to (3-15), $H = (01000)_2$, $I = (10000)_2$, $J_1 = (11101)_2$, $J_2 = (11100)_2$. From Equation (3-16), $Q = (10011)_2 = 19$. From Equation (3-17), $Z = (11011)_2 = 27$. From Equations (3-19), $Z^{(1)} = (1\ 1011\ 1110\ 1111)_2$, and $T = (11011\ 11010100)_2$. From Equation (3-18), $X = (11011\ 11010100\ 0010)_2 = 113986$. We can see that $|113986|_{15} = 1$, $|113986|_{16} = 2$, $|113986|_{17} = 1$, $|113986|_{31} = 30$, therefore, the calculated X is indeed the weighted value of the residue representation (1, 2, 1, 30) with respect to the moduli set {15, 16, 17, 31}.

3.3 Reverse Conversion of $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$

Now let's consider the second four-moduli set $S_4^2 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, where n is any even natural number. Let the RNS representation of the integer X be (x_1, x_2, x_3, x_4), and the binary representations of the residues are the same as that for the moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ except that x_4 has only $n - 1$ bits.

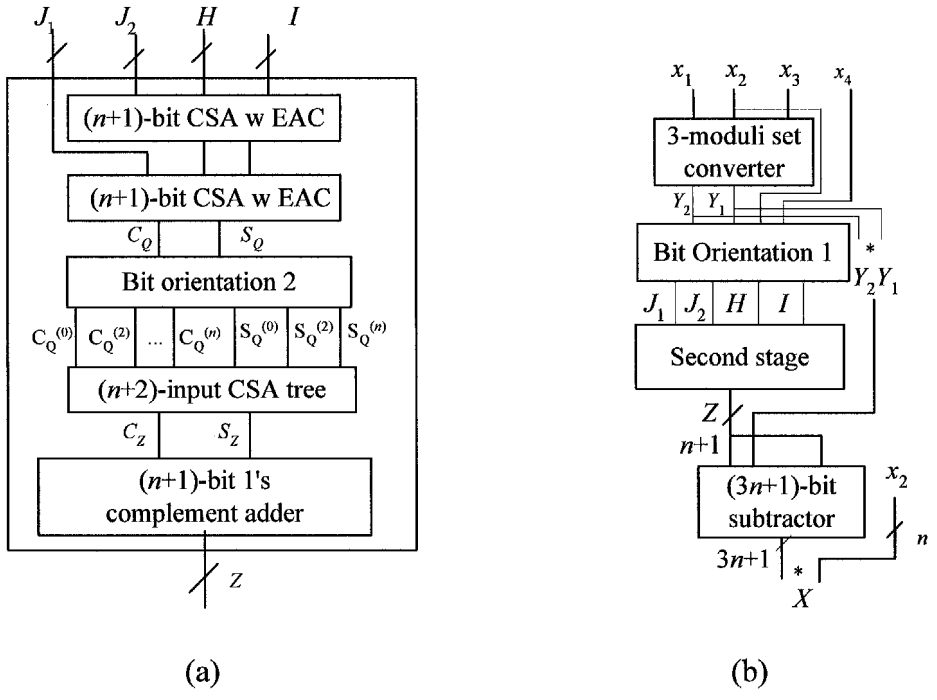


Figure 3.3 *The three-stage reverse converter for moduli set* S_4^1

Similar concept can be applied to the reverse conversion algorithm for this new four-moduli set. First, we obtain the integer $X^{(1)} = (x_1, x_2, x_3)$ corresponding to the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ by an efficient reverse converter as before. Next, MRC technique is applied to calculate the integer $X = (X^{(1)}, x_4)$ corresponding to the two-moduli set $\{2^n(2^{2n} - 1), 2^{n-1} - 1\}$. Besides *Property 2.10 and 2.11*, the following two Lemmas are required for the reverse conversion algorithm.

Lemma 3.3: The multiplicative inverse of $2^n\left(2^{2n}-1\right)$ modulo $(2^{n-1} - 1)$ is

$$\left|\frac{1}{2^n\left(2^{2n}-1\right)}\right|_{2^{n-1}-1} = \frac{1}{3}\left(2^n + 2^{n-2} - 2\right), \tag{3-22}$$

where n is any even number.

Proof: Since

$$\left|2^{2n}-1\right|_{2^{n-1}-1} = \left|\left(2^n-1\right)\left(2^n+1\right)\right|_{2^{n-1}-1} = 3,$$

and

$$\left|2^n\right|_{2^{n-1}-1} = \left|2\left(2^{n-1}-1\right)+2\right|_{2^{n-1}-1} = 2.$$

Using the above two identities, we have

$$\left|\frac{1}{3}\left(2^n + 2^{n-2} - 2\right)2^n\left(2^{2n}-1\right)\right|_{2^{n-1}-1}$$
$$= \left|\frac{1}{3}\left(2 + 2^{n-2} - 2\right)\cdot 2\cdot 3\right|_{2^{n-1}-1} = \left|2^{n-1}\right|_{2^{n-1}-1} = 1$$

Therefore, $\frac{1}{3}\left(2^n + 2^{n-2} - 2\right)$ is the multiplicative inverse of $2^n(2^{2n} - 1)$ modulo $(2^{n-1} - 1)$. □

Lemma 3.4: For even number n, the multiplicative inverse of 3 modulo $(2^{n-1} - 1)$ is

$$\left|\frac{1}{3}\right|_{2^{n-1}-1} = \frac{1}{3}\left(2^n - 1\right) = \sum_{i=0}^{n/2-1} 2^{2i} \tag{3-23}$$

Proof:

$$\left|3\cdot\left(\frac{1}{3}\left(2^n-1\right)\right)\right|_{2^{n-1}-1}=\left|2^n-1\right|_{2^{n-1}-1}=\left|2\cdot\left(2^{n-1}-1\right)+1\right|_{2^{n-1}-1}=1 \qquad \square$$

3.3.1 Four-Stage Reverse Converters

Based on *Property 2.10 and Property 2.11*, *Lemma 3.3* and *Lemma 3.4*, a new algorithm can be developed for the reverse conversion for the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$. The first stage is the same as the reverse converter for the first four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$. For subsequent stages, consider the moduli set $\{2^n(2^{2n} - 1), 2^{n-1} - 1\}$ and $X = (X^{(1)}, x_4)$. Using MRC Equation (2-30), X can be calculated by

$$X = X^{(1)} + 2^n\left(2^{2n}-1\right)\left|k_0\left(x_4 - X^{(1)}\right)\right|_{2^{n-1}-1} \tag{3-24}$$

where

$$\left|k_0\cdot 2^n\left(2^{2n}-1\right)\right|_{2^{n-1}-1}=1. \tag{3-25}$$

According to *Lemma 3.3*, we substitute the multiplicative inverse k_0 into Equation (3-25):

$$\begin{aligned} X &= X^{(1)} + 2^n\left(2^{2n}-1\right)\left|\frac{1}{3}\left(2^n + 2^{n-2} - 2\right)\left(x_4 - X^{(1)}\right)\right|_{2^{n-1}-1} \\ &= X^{(1)} + 2^n\left(2^{2n}-1\right)Z \end{aligned} \tag{3-26}$$

where Z is defined as follows:

$$\begin{aligned} Z &= \left|\frac{1}{3}\left(2^n + 2^{n-2} - 2\right)\left(x_4 - x_2 - 2^n Y\right)\right|_{2^{n-1}-1} \\ &= \left|\left|\frac{1}{3}\right|_{2^{n-1}-1}\left|2^{n-2}\left(x_4 - x_2 - 2Y\right)\right|_{2^{n-1}-1}\right|_{2^{n-1}-1} = \left|P\cdot Q\right|_{2^{n-1}-1} \end{aligned}, \tag{3-27}$$

where P is defined by (3-23) and the term Q is defined as follows:

$$Q = \left|2^{n-2}(x_4 - x_2 - 2Y)\right|_{2^{n-1}-1} = \left|H + I + J\right|_{2^{n-1}-1}, \tag{3-28}$$

where H, I and J can be calculated with the aid of *Property 2.10*.

$$H = \left|2^{n-2}x_4\right|_{2^{n-1}-1} = CLS_{n-1}(x_4, n-2) = \left(X_{4,0}X_{4,n-2}X_{4,n-3}\cdots X_{4,2}X_{4,1}\right)_2 \tag{3-29}$$

$$I = \left|-2^{n-2}x_2\right|_{2^{n-1}-1} \tag{3-30}$$

Since x_2 is an n-bit number, *Property 2.10* can not be applied directly to calculate I. Let $x_2{}^d = x_2 - 2^{n-1}X_{2,n-1}$, i.e., $x_2{}^d = (X_{2,n-2}, X_{2,n-3}, \ldots, X_{2,0})_2$. Thus, we have

$$\left|x_2\right|_{2^{n-1}-1} = \left|X_{2,n-1}2^{n-1} + x_2^d\right|_{2^{n-1}-1} = \left|X_{2,n-1} + x_2^d\right|_{2^{n-1}-1}$$

Now *Property 3.10* can be applied for the calculation of I:

$$\begin{aligned} I &= \left|-2^{n-2}x_2\right|_{2^{n-1}-1} = \left|-2^{n-2}\left(X_{2,n-1} + x_2^d\right)\right|_{2^{n-1}-1} \\ &= \left|-CLS_{n-1}\left(X_{2,n-1}, n-2\right) - CLS_{n-1}\left(x_2^d, n-2\right)\right|_{2^{n-1}-1} = \left|I_1 + I_2\right|_{2^{n-1}-1} \end{aligned} \tag{3-31}$$

where

$$I_1 = \left(\overline{X}_{2,n-1}(1)^{(n-2)}\right)_2, \tag{3-32}$$

$$I_2 = \left(\overline{X}_{2,0}\overline{X}_{2,n-2}\cdots\overline{X}_{2,2}\overline{X}_{2,1}\right)_2. \tag{3-33}$$

In equation (3-32), the notation of $(1)^{(n-2)}$ indicates a string of $n-2$ bits of '1'. The term J in Equation (3-28) is evaluated as follows:

$$J = \left|-2^{n-1}Y\right|_{2^{n-1}-1} = \left|-Y\right|_{2^{n-1}-1} = \left|-\left(Y_1 + 2^n Y_2\right)\right|_{2^{n-1}-1} = \left|-\left(Y_1 + 2Y_2\right)\right|_{2^{n-1}-1}$$

Let $Y_1{}^d = Y_1 - 2^{n-1}Y_{1,n-1}$ and $Y_2{}^d = Y_2 - 2^{n-1}Y_{2,n-1}$, where $Y_1{}^d$ and $Y_2{}^d$ are both $(n-1)$-bit strings. Substituting them into the above identity, we have

$$\begin{aligned} J &= \left|-\left(Y_1^d + 2^{n-1}Y_{1,n-1} + 2\left(Y_2^d + 2^{n-1}Y_{2,n-1}\right)\right)\right|_{2^{n-1}-1} \\ &= \left|-\left(Y_{1,n-1} + 2Y_{2,n-1} + Y_1^d + 2Y_2^d\right)\right|_{2^{n-1}-1} = \left|J_1 + J_2 + J_3\right|_{2^{n-1}-1} \end{aligned} \tag{3-34}$$

where

$$J_1 = \left((1)^{(n-3)}\overline{Y}_{2,n-1}\overline{Y}_{1,n-1}\right)_2 \tag{3-35}$$

$$J_2 = \left(\overline{Y}_{1,n-2} \cdots \overline{Y}_{1,1}\overline{Y}_{1,0}\right)_2 \tag{3-36}$$

$$J_3 = \left(\overline{Y}_{2,n-3} \cdots \overline{Y}_{2,1}\overline{Y}_{2,0}\overline{Y}_{2,n-2}\right)_2 \tag{3-37}$$

According to Equations (3-32) and (3-35), there are some '1's included in both I_1 and J_1, we combine I_1 and J_1 into I_1J_1 as follows:

$$I_1J_1 = \left(\overline{X}_{2,n-1}(1)^{(n-4)}\overline{Y}_{2,n-1}\overline{Y}_{1,n-1}\right)_2 \tag{3-38}$$

Since there are some constant '1's embedded in the strings of I_1J_1, the first CSA in Figure 3.4(a) can be further simplified. According to Equations (3-23) and (3-27), Z can be calculated by

$$\begin{aligned} Z &= \left|P \cdot Q\right|_{2^{n-1}-1} = \left|\left(2^0 + 2^2 + 2^4 + \cdots + 2^{n-2}\right)Q\right|_{2^{n-1}-1} \\ &= \left|CLS_{n-1}(Q,0) + CLS_{n-1}(Q,2) + \cdots + CLS_{n-1}(Q,n-2)\right|_{2^{n-1}-1} \\ &= \left|Q^{(0)} + Q^{(2)} + \cdots + Q^{(n-2)}\right|_{2^{n-1}-1} \end{aligned} \tag{3-39}$$

The terms $Q^{(0)}$, $Q^{(2)}$,..., $Q^{(n-2)}$ in Equation (3-39) are $(n-1)$-bit circularly left-shifted versions of Q. In essence, Z can be easily implemented by one $(n/2, 2^{n-1} - 1)$-MOMA. Figure 3.4(b) shows the hardware scheme for the calculation of Z when $n = 10$. Finally, the value of X can be calculated by

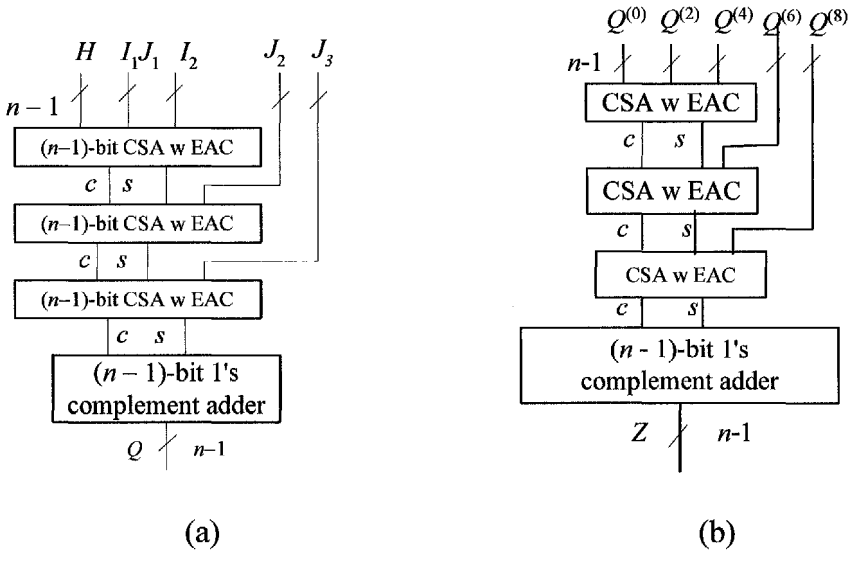


Figure 3.4 *The calculation of Q and Z for the moduli set* S_4^2

$$X = X^{(1)} + 2^n\left(2^{2n}-1\right)Z = x_2 + 2^n Y + 2^{3n} Z - 2^n Z$$
$$= x_2 + 2^n\left(2^{2n} Z + Y - Z\right) = x_2 + 2^n\left(Z^{(1)} - Z\right) \quad , \tag{3-40}$$

where

$$Z^{(1)} = 2^{2n} Z + Y = \left(Z_{n-2} \cdots Z_0 Y_{2,n-1} \cdots Y_{2,0} Y_{1,n-1} \cdots Y_{1,0}\right)_2 \tag{3-41}$$

It is clear from Equation (3-40) that one $(3n-1)$-bit subtractor is needed for the calculation from $Z^{(1)}$ and Z, and then the result is concatenated to the n-bit x_2 to obtain the final value of X. Figure 3.5 shows the hardware implementation devised directly from the algorithm. The first stage is a reverse converter for the popular triple-moduli set. The second stage and third stage are shown in Figure 3.4. The final stage is a $(3n-1)$-bit subtractor. The structure resembles the four-stage architecture for the first four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ presented in Section 3.2.1.

3.3.2 Three-Stage Reverse Converters

Following the same argument of Section 3.3.1, the CPAs with EAC in the second and third stages of the four-stage architecture can be merged into one CPA with EAC, therefore, a new three-stage architectures can be developed to reduce the total conversion delay at the expense of area complexity.

In Figure 3.4(a), if the $(n-1)$-bit 1's complement adder is omitted, the value of Q will be kept in stored carry format. Thus, Z can be expressed by

$$Z = \left|P \cdot Q\right|_{2^{n-1}-1} = \left|\left(2^0 + 2^2 + 2^4 + \cdots + 2^{n-2}\right)\left(C_Q + S_Q\right)\right|_{2^{n-1}-1}$$
$$= \left|CLS_{n-1}\left(C_Q, 0\right) + \cdots + CLS_{n-1}\left(C_Q, n-2\right) + CLS_{n-1}\left(S_Q, 0\right) + \cdots + CLS_{n-1}\left(S_Q, n-2\right)\right|_{2^{n-1}-1} \tag{3-42}$$
$$= \left|C_Q^{(0)} + C_Q^{(2)} + \cdots + C_Q^{(n-2)} + S_Q^{(0)} + S_Q^{(2)} + \cdots + S_Q^{(n-2)}\right|_{2^{n-1}-1}$$

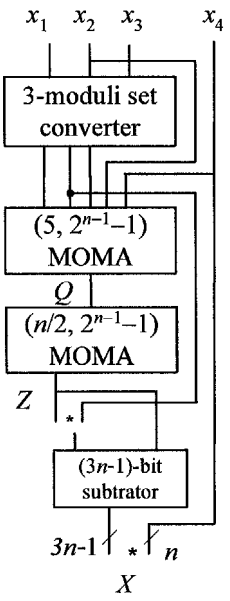


Figure 3.5 *The four-stage reverse converter for the moduli set* S_4^2

In Equation (3-42), the terms $S_Q^{(i)}$, $C_Q^{(i)}$ are $(n-1)$-bit circularly left-shifted versions of S_Q and C_Q, respectively, and only one $(n, 2^{n-1}-1)$-MOMA is required for their modular summation to Z. Figure 3.6(a) shows the second stage of the three-stage architecture.

After Z is obtained, a $(3n-1)$-bit subtractor can be used to calculate X according to Equations (3.40) and (3.41). Figure 3.6(b) shows the three-stage architecture of the reverse converter where the first and third stages remain unchanged from the four-stage architecture. This three-stage architecture can also be pipelined to achieve a high throughput rate.

Example 3.2: Assume the desired dynamic range is 14 bits. For the first four-moduli set, $n = 4$, the resulting dynamic range is 17 bits, 3 bits will be wasted. If the second four-moduli set is used, $n = 4$, and the dynamic range is 15 bits. Hence, there is only 1 surplus bit. Therefore, we choose the four-moduli set as {15, 16, 17, 7}. Let $X = (1, 2, 1, 6)$. Using the reverse converter for the triple-moduli set {15, 16, 17}, we can obtain $Y = (1110\ 1111)_2$, where $Y_1=(1111)_2$, and $Y_2 = (1110)_2$. According to Equation (3-29), $H = (011)_2$. According to Equations (3-33) to (3-38), $I_1J_1 = (100)_2$, $I_2 = (110)_2$, $J_2 = (000)_2$, and $J_3 = (010)_2$. From Equation (3-28), $Q = (001)_2$. From Equation (3-39), $Z = (101)_2$. From Equation (3-41), $Z^{(1)} = (101\ 1110\ 1111)_2$. From Equation (3-40), $X =$

$(0101\ 1110\ 1010\ 0010)_2 = 24226$. We can verify that $|24226|_{15} = 1$, $|24226|_{16} = 2$, $|24226|_{17} = 1$, $|24226|_7 = 6$, so 24226 is indeed the weighted value of the residue representation (1, 2, 1, 6) with respect to the moduli set {15, 16, 17, 7}.

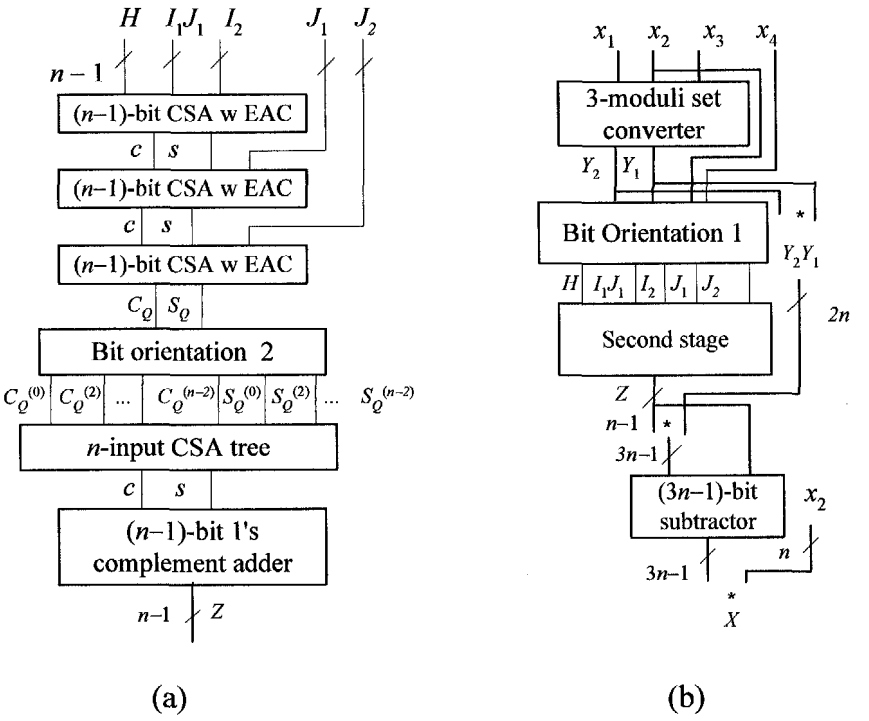


Figure 3.6 *Three-stage reverse converter for the moduli set* S_4^2

3.4 Performance Evaluation and Comparison

In this section, we make the performance evaluation and comparison for the reverse converters of the two moduli sets S_4^1 and S_4^2. For the moduli set S_4^1, the reverse converter proposed in [Vin00] consists of different modular adders and other components, it is not easy to implement and synthesize; therefore, we estimate the hardware costs in terms of the number of primitive logic elements such as FA, and evaluate the critical path delays of the reverse converters. The wire loads are typically assumed to be negligible in such an estimation process. In order to provide a more detailed comparison, we employ two different implementations of CPA with EAC presented in [Pie95] and [Bha98], both the four-stage and three-stage architectures. Hence, there are altogether four versions of implementations to be analyzed for each

four-moduli set: four-stage Cost-Effective (CE), four-stage High Speed (HS), three-stage CE, and three-stage HS. The hardware costs of CSA with EAC employed here are computed based on the basic structure of [Hwan79], which is widely used for complexity analysis as in [Pies95] [Bha98]. For the comparison for the moduli set S_4^2, the reverse converter proposed in [Ska99b] has a close structure as ours; therefore, we not only compare them in terms of primitive components such as modular adders and subtractors, but also provide the comparison on the synthesized results. In order to make the comparison easily to understand, the method proposed in [Efst94] is used to implement the 1's complement adder for both reverse converters instead of the CE and HS versions.

The reverse converter proposed in [Vin00] for the four-moduli set S_4^1 was reported to be the most efficient. The design of [Vin00] consists of 13 adders/subtractors with operand widths ranging from $(n + 1)$ to $(5n + 1)$ bits and some other combinational logic circuits. The authors of [Vin00] do not provide the hardware complexity analysis in terms of the number of FAs at algorithmic level. We have attempted to analyze their proposed architecture to determine the number of FAs required by their reverse converter by ignoring the overhead due to other combinational logic circuits. We use the simple RCA of Figure 2.3 as opposed to the more area intensive CLA for the implementation of the CPA to provide a fair comparison. Moreover, we assume that the modular adder proposed in [Bay87] will be used for the implementation of the modulo $2^n(2^{3n} - 1)$ subtractor of their converter. Based on these criteria, their reverse converter requires $(37n + 14)$ FAs. The conversion delay is $(14n+8)t_{FA}$ as stated in [Vin00].

The comparisons between our proposed reverse converters and the reverse converter of [Vin00] are shown in Table 3.1, where the number l and m are the numbers of levels of CSA with EAC tree of $(n/2 + 1, 2^{n+1} - 1)$-MOMA and $(n + 2, 2^{n+1} - 1)$-MOMA, respectively. The detailed derivation of this table can be found in [Cao05a]. It is clear that our new reverse converters are always faster than the reverse converter of [Vin00]; particularly, our HS version can achieve about three times faster than the converter of [Vin00]. In terms of area complexity, our converters are more economical than that of [Vin00] if the dynamic range is within the typical dynamic

ranges of DSP applications. For example, when the dynamic range (DR) ≤ 208 bits for the four-stage CE version, DR ≤ 160 bits for the four-stage HS version, DR ≤ 104 bits for the three-stage CE version and DR ≤ 84 bits for the three-stage HS version, the hardware costs of our proposed reverse converters are smaller than that of [Vin00]. It should be highlighted that the above comparisons of hardware costs do not include the interconnection complexity. The architecture of [Vin00] is built using adders and subtractors of different operand widths. On the contrary, our architectures are more regular and modular. Thus, the interconnect complexity is envisaged to be smaller than that of [Vin00].

Table 3.1 *Comparisons of the reverse converters for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n+l} - 1\}$

Converters	Hardware	Delay
[Vin00]	$(37n + 14)$ FAs	$(14n + 8)t_{FA}$
Four-stage CE	$(n^2/2 + 11n + 4)$ FAs	$(11n + l + 8)t_{FA}$
Four-stage HS	$(n^2/2 + 17n + 6)$ FAs	$(5n + l + 6)t_{FA}$
Three-stage CE	$(n^2 + 10.5n + 3)$ FAs	$(9n + m + 6)t_{FA}$
Three-stage HS	$(n^2 + 15.5n + 4)$ FAs	$(4.5n + m + 5)t_{FA}$

The reverse converter proposed in [Ska99b] for the same four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ is also based on the two-level CRT-MRC algorithm. Our proposed approach for the first level adopts a reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, whilst the algorithm of [Ska99b] consists of two reverse converters for two moduli sets, namely moduli sets $\{2^n, 2^{n-1} - 1\}$ and $\{2^n - 1, 2^n + 1\}$. In our algorithm, the second level is an MRC for two-moduli set $\{2^n(2^{2n} - 1), 2^{n-1} - 1\}$, while in [Ska99b], MRC is used for a two-moduli set $\{2^n(2^{n-1} - 1), 2^{2n} - 1\}$.

As only the notion and principle are presented in [Ska99b] without the algorithmic details, we can only derive the algorithm with best effort to make a comparison. Without loss of generality, the moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ can be rearranged to $\{2^n, 2^{n-1} - 1, 2^n - 1, 2^n + 1\}$ with the residue set (x_2, x_4, x_1, x_3). According to the MRC Equation (2-30), the mixed radix coefficient is 2^{n-2} for the two-moduli set $\{2^n, 2^{n-1} - 1\}$; therefore, the weighted number for (x_2, x_4) with respect to $\{2^n, 2^{n-1} - 1\}$ can be obtained by

$$Y_{11} = x_2 + 2^n \left| (x_4 - x_2) \cdot 2^{n-2} \right|_{2^{n-1}-1} = x_2 + 2^n \left| CLS_{n-1}(x_4, n-2) + CLS_{n-1}(\bar{x}_2^d, n-2) + \left((0)^{(n-2)} \bar{X}_{2,n-1}\right)_2 \right|_{2^{n-1}-1}, \quad (3\text{-}43)$$

where x_2^d is the least significant $n-1$ bits of x_2, and $X_{2,n-1}$ is the most significant bit of x_2. By using *Property 2.10*, we can implement Equation (3-43) using a $(3, 2^{n-1}-1)$-MOMA where one CSA can be simplified because the third input consists of one variable bit and $n-2$ '0's. According to the simplification method of FA mentioned in section 2.1.4.1, the hardware costs include n inverters, $1+(n-2)/2 = 0.5n$ FAs, and one modulo $2^{n-1}-1$. The delay is $t_{FA} + t_{M1(n-1)}$.

The weighted number of (x_1, x_3) with moduli set $\{2^n - 1, 2^n + 1\}$ is obtained as follows using CRT:

$$\begin{aligned} Y_{12} &= \left| x_1 \left| \frac{1}{2^n+1} \right|_{2^n-1} (2^n - 1) + x_3 \left| \frac{1}{2^n-1} \right|_{2^n+1} (2^n + 1) \right|_{2^{2n}-1} \\ &= \left| x_1 \cdot 2^{2n-1} - x_1 \cdot 2^{n-1} + x_3 \cdot 2^{2n-1} + x_3 \cdot 2^{n-1} \right|_{2^{2n}-1} \\ &= \left| CLS_{2n}(x_1, 2n-1) + CLS_{2n}(\bar{x}_1, n-1) + CLS_{2n}(x_3, 2n-1) + CLS_{2n}(x_3, n-1) \right|_{2^{2n}-1} \end{aligned} \quad . \quad (3\text{-}44)$$

Equation (3-44) can be further simplified because its $2n$-bit inputs include some embedded '0's and '1's. Using *Property 2.10* and *2.11*, we can simplify as follows:

$$CLS_{2n}(x_1, 2n-1) = \left(X_{1,0}(0)^{(n)} X_{1,n-1} \ldots X_{1,1}\right)_2$$

$$CLS_{2n}(\bar{x}_1, n-1) = \left(1 \bar{X}_{1,n-1} \ldots \bar{X}_{1,0} (1)^{(n-1)}\right)_2$$

$$CLS_{2n}(x_3, 2n-1) = \left(X_{3,0}(0)^{(n-1)} X_{3,n} \ldots X_{3,1}\right)_2$$

$$CLS_{2n}(x_3, n-1) = \left(0 X_{3,n} \ldots X_{3,0} (0)^{(n-2)}\right)_2$$

These terms can further be simplified into 3 terms. There are still $2n-3$ '0's that allow the FAs involved to be further simplified. Finally, Equation (3-44) can be implemented by a $(3, 2^{2n}-1)$-MOMA, where the hardware costs include n inverters, $3\times 2n - (2n-3)/2 = 5n + 1.5$ FAs, and one modulo $2^{2n}-1$ adder.

Now let us look at the second level's MRC for the two-moduli set $\{2^n(2^{n-1}-1), 2^{2n}-1\}$ for [Ska99b]. The final value of X corresponding to the residues (Y_{11}, Y_{12}) of the moduli set $\{2^n(2^{n-1}-1), 2^{2n}-1\}$ can be obtained by

$$X = Y_{11} + 2^n\left(2^{n-1}-1\right)a_{21} = \left(Y_{11} + 2^{2n-1}a_{21}\right) - 2^n a_{21} \tag{3-45}$$

where the MRC coefficient a_{21} is defined as

$$a_{21} = \left\| \frac{1}{2^n\left(2^{n-1}-1\right)}\right|_{2^{2n}-1} \left(Y_{12}-Y_{11}\right)\Big|_{2^{2n}-1},$$

where the multiplicative inverse inside a_{21} is $\frac{1}{3}\left(2^{2n+1}-2^{n+2}-2^2\right)$. Therefore, MRC coefficient a_{21} is obtained by

$$a_{21} = \left|\frac{1}{3}\left(2^{2n+1}-2^{n+2}-2^2\right)\left(Y_{12}-Y_{11}\right)\right|_{2^{2n}-1}. \tag{3-46}$$

In Equation (3-46), the multiplicative inverse of 3 modulo $2^{2n}-1$ does not exist, when n is even number. Therefore, the systematic methods to obtain the values of a_{21}, such as Lemmas 3.1 to 3.4, cannot be found. Since there are $n-1$ '1's in the binary format of multiplicative inverse of a_{21}, $2(n-1)$ terms are to be added by using *Property 2.10* and *Property 2.11*. As Y_{11} is $(2n-1)$-bits long, and Y_{12} is $2n$-bit long, the calculation of Equation (3-46) can be simplified in the same way as those for Equation (3-43) and (3-44). Therefore, we can use one $(2n-2, 2^{2n}-1)$-MOMA to implement (3-46). After a_{21} has been computed, one $(3n-1)$-bit binary subtractor is required according to Equation (3-45) because the least significant n bits of $2^n a_{21}$ are zeros. Therefore, the total hardware costs of the second level include $2n-1$ inverters, $(2n-2-2)\times 2n = 4n^2 - 8n$ FAs, one modulo $2^{2n}-1$ adder and one $(3n-1)$-bit binary adder. The delay is $t_{INV} + l_3 * t_{FA} + t_{M1(2n)} + t_{SUB(3n-1)}$.

Therefore, the total hardware costs of the reverse converter [Ska99b] include $3n$ inverters, 1 modulo $2^{2n}-1$ adder, 2 modulo $2^{n-1}-1$ adder, 1 $(3n-1)$-bit binary

subtractor and $0.5n^2 + 2.5n + 1$ FAs. The total conversion delay is $2t_{M1(2n)} + 2t_{INV} + (l_3+1)*t_{FA} + t_{SUB(3n-1)}$.

Table 3.2 and Table 3.3 show the comparison of the hardware costs and delays for our proposed reverse converters and the one from [Ska99b]. In Table 3.2, there are one modulo $2^{2n} - 1$ adder and 2 modulo $2^{n-1} - 1$ adders are used, while in [Ska99b], two modulo $2^{2n} - 1$ adders and 1 modulo $2^{n-1} - 1$ adder are used. According to [Efst94], the hardware cost of modulo $2^{2n} - 1$ adder is larger than those for modulo $2^{n-1} - 1$ adder; therefore, the total hardware cost of ours is smaller than that for [Ska99b]. In Table 3.3, l_3 is the number of levels of one $(2n-2)$-input CSA tree, while l_2 is that of one $n/2$-input CSA tree, therefore, $l_2 < l_3$. The delay difference between the two reverse converters is small, because in our reverse converter, the main delay consists of $t_{M1(2n)} + 2t_{M1(n-1)} + (l_2+5)t_{FA} + t_{SUB(3n-1)}$, while the delay of [Ska99b] consists of $2t_{M1(2n)} + (l_3+1)t_{FA} + t_{SUB(3n-1)}$. Since the delay of the CLA-like modular adder is not linearly proportional to the width of the operand [Efst94], the total conversion delay of our reverse converter is slightly longer than that of [Ska99b].

Table 3.2 *Hardware costs comparison with [Ska99b] for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$

Converter	FA	Modulo $2^{2n} - 1$ adder	Modulo $2^{n-1} - 1$ adder	$(3n-1)$-bit Binary Subtractor	Inverter
Ours (4-stage)	$0.5n^2 + 2.5n + 1$	1	2	1	$3n$
[Ska99b]	$4n^2 - 2.5n + 1.5$	2	1	1	$4n-1$

Table 3.3 *Conversion delay comparison with [Ska99b] for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$

Ours (4-stage)	$t_{MUX} + t_{M1(2n)} + 2t_{INV} + (l_2+5)*t_{FA} + 2t_{M1(n-1)} + t_{SUB(3n-1)}$
[Ska99b]	$2t_{M1(2n)} + 2t_{INV} + (l_3+1)*t_{FA} + t_{SUB(3n-1)}$

Table 3.4 *Synthesized comparisons with [Ska99b] for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$

n		4		8		12		16		20	
Optimized mode		Speed	Area	Speed	Area	Speed	Area	Speed	Area	Speed	Area
Total area	Ours	1764	747.4	4268	1824	7056	3027	10038	4353	13984	5949
	[Ska99b]	2077	851.2	6791	2997	13097	6647	17689	12397	34944	19607
	Δarea (%)	15.1	12.2	37.2	39.1	46.1	54.5	43.3	64.9	60.0	69.7
Total Delay (ns)	Ours	7.61	12.20	9.79	17.31	11.1	20.21	11.79	21.65	12.94	24.02
	[Ska99b]	6.15	10.52	8.19	17.58	9.79	22.93	10.76	23.95	11.33	31.6
	Δdelay(%)	-23.7	-16.0	-19.5	1.5	-13.4	11.9	-11.2	9.6	-14.2	22.1
Power (mW)	Ours	17.73	5.68	44.53	14.50	74.69	24.33	105.78	35.25	150	47.91
	[Ska99b]	21.9	6.67	58.7	24.78	118.1	54.01	158.5	93.72	350.8	150.4
	Δpower(%)	19.4	14.8	24.1	41.5	36.8	55.0	33.3	62.4	57.0	68.1

In order to provide a more realistic comparison, both reverse converters in Table 3.2 and 3.3 are implemented by Avant!'s Libra Passport V.35 libraries. Appendix provides the detailed information about the EDA tools and the technology libraries used in the evaluation. The binary subtractors are from Synopsys's DesignWare basic library, which are based on CLA. Two optimizations are analyzed. One is optimized to achieve minimum area, and second is to achieve minimum critical path delay. In our simulation, the power analysis is invoked without annotating any switching activity. This will give the default power simulation of 50% static probability and a toggle rate of 50% of the fastest clock in the design. The synthesized results for area, delay and power were obtained from the Synopsys's Design Complier. Table 3.4 shows the comparison results. The total area includes both the cell area and the net interconnection area, and is measured in terms of the area of 2-input NAND gate. The unit of the total delay is in nanosecond. The percentage improvements in each performance metric of our design over that of [Ska99b] are also shown in the rows labeled 'Δvalue (%)'. A positive (negative) Δ indicates an improved (inferior) performance of our design over that of [Ska99b]. Figures 3.7 to 3.8 show the comparisons of area, total delay and power consumption. It is clear that our proposed reverse converter uses significantly less hardware and consume much less power than those for [Ska99b] in both optimization modes, up to 3 times saving of hardware cost and power consumption is achieved. For the total delay, in the area optimization mode, ours is faster than [Ska99b] except for $n = 4$, while in the delay optimization mode, the design of [Ska99b] is faster.

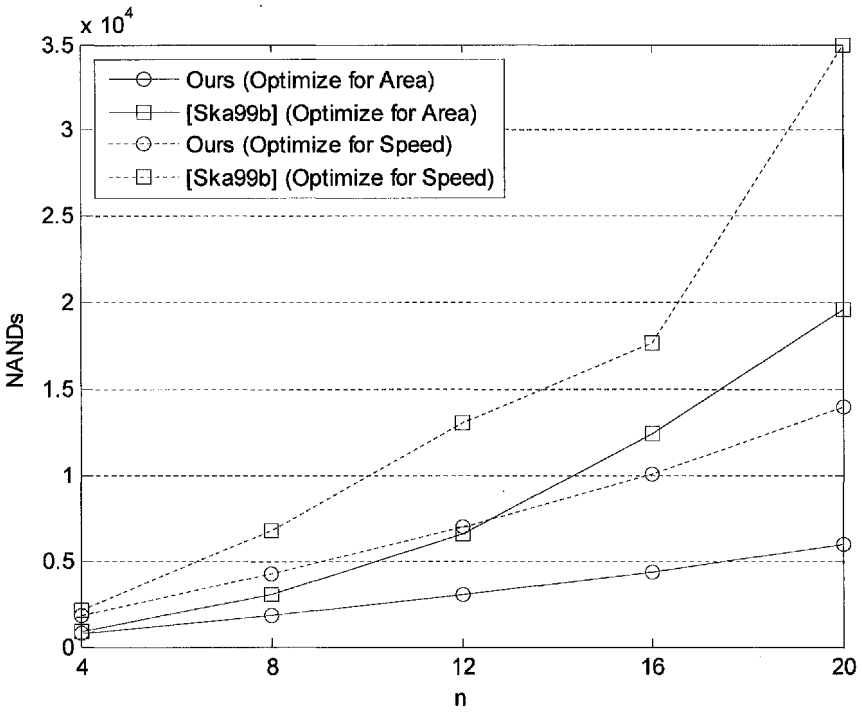


Figure 3.7 *Area comparisons between our reverse converter for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ *and that of [Ska99b]*

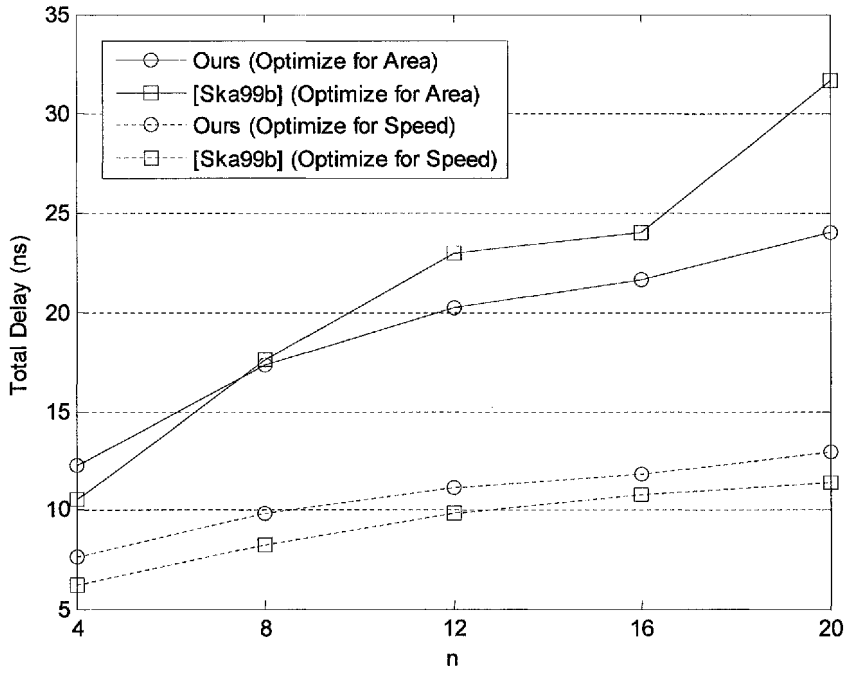


Figure 3.8 *Total delay comparisons between our reverse converter for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ *and that of [Ska99b]*

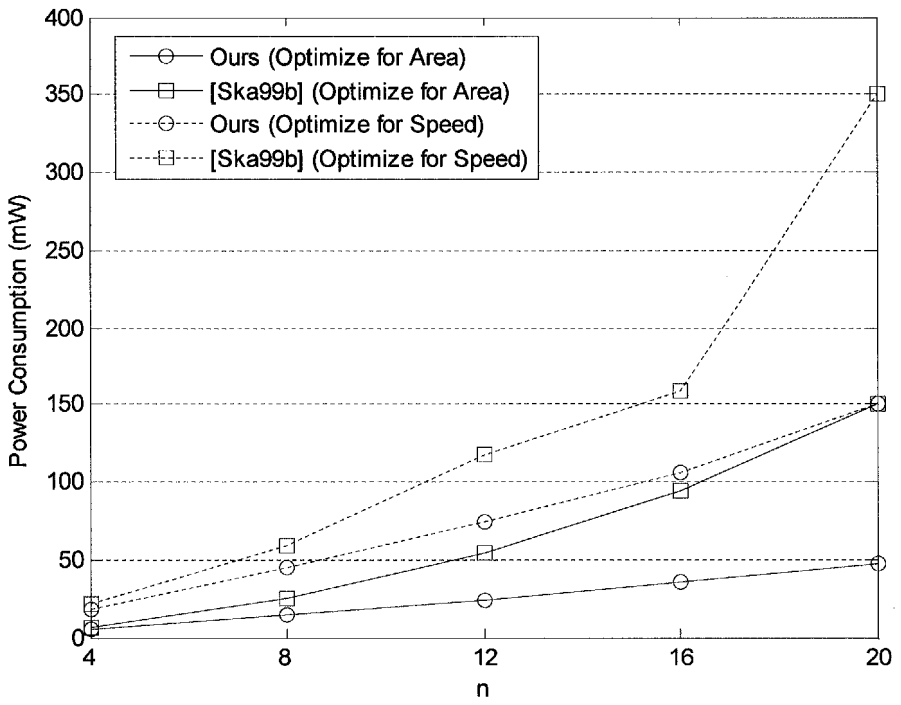


Figure 3.9 *Power consumption comparisons between our proposed reverse converter for* $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ *and that of [Ska99b]*

3.5 Conclusions

In this chapter, we proposed a new reverse conversion algorithm for the four-moduli sets, which is based on a two-level MRC technique. Using this method, we have proposed a number of reverse conversion architectures for the four-moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, where n is an even number. The proposed conversion techniques are based on MRC and the popular triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. The various architectures, for realizing reverse converters for these moduli sets, are all based on FAs thereby providing with more options to maximize the full potential and advantage of advanced design automation tools and optimized cell library. Our analyses show that the proposed reverse converter architectures for the moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ are faster than the best reverse converter reported in the literature. Within the typical dynamic ranges of DSP applications, our new converters are also proven to be area efficient. The reverse converter for the $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ shares the similar conversion algorithm

and the similar architecture as that of the $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$. Comparing with the an existing reverse converter which is also based on a two-level MRC technique, our analysis shows that the proposed reverse converter consumes up to 1/3 of the area of the existing reverse converter. The reasons that account for the lower area cost are twofold. First, in the first level, our reverse converter uses a reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, while the existing reverse converter use two reverse converters. Second, the MRC of the second level of our reverse converter involves only modulo $2^{n-1} - 1$ operations, while the existing reverse converter uses modulo $2^{2n} - 1$ operations, where the wordlength is more than two times that of our reverse converter.

Chapter 4

An Efficient Reverse Converter for a New Four-Moduli Superset

4.1 Introduction

In this chapter, we will introduce a $S_4^4 = \{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ and present an elegant reverse conversion algorithm for this RNS. A large portion of this chapter has been published in the IEEE Transactions of Circuits and Systems – Part I [Cao03a] and presented at the 2003 IEEE International Symposium on Circuits and Systems [Cao03b].

In chapters 2 and 3, we have investigated a number of architectures for the reverse converters of different existing four-moduli supersets, where variants of CRT and MRC are adopted for their implementations. We observed that CRT implementation of reverse converters requires several modular multipliers, and a large modular adder. MRC is a sequential process, and requires some LUTs. Both of them, when applied independently, are not efficient for reverse converters of RNS with large dynamic range. As a result, we have proposed and demonstrated a hybrid CRT and MRC approach in the derivation of efficient reverse converter designs for two akin four-moduli supersets in Chapter 3.

Recently, alternative revolutionary solutions to the traditional CRT for the reverse conversion problem of the general moduli set RNS, known as the New CRTs have been proposed [Wang98] [Wang00]. The New CRTs have inherited the merits of the classical CRT and MRC algorithms. As mentioned in Chapter 2, RNS's with general moduli sets lack in efficient RAUs and the reverse converters. Furthermore, it is difficult to choose a suitable general moduli set for a specific dynamic range RNS.

Applications of the New CRTs to the reverse conversion of general moduli sets also require several modular multipliers and a large modular adder. Under certain constraints such as the moduli sets of Equation (2-42), the New CRTs are found to be more efficient than traditional CRT for the reverse converter design. One application of the New CRTs is manifested by the highly efficient reverse conversion of the celebrated triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ [Wang02].

Inspired by the prominent performance gain of the reverse converter for the triple-moduli set and the potential simplification made possible by the number theoretic interaction of the New CRT, we explore new moduli set which could take advantage of the New CRT to increase the parallelism and extend the dynamic range of the RNS. It is noted that the dynamic range of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is only $3n$ bits. For contemporary signal processing applications, as highlighted in the previous chapter, the parallelism of this moduli set is also limited. It is advantageous to consider higher cardinality moduli supersets with larger dynamic range for higher parallelism, provided that the architectures of their forward/reverse conversions and RAUs can be maintained relatively efficient.

In this chapter, we propose a new four-moduli set $S_4^4 = \{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$. The dynamic range of this moduli set is $5n$ bits. The architecture of the reverse converter is comparable in complexity to the architecture for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. Compared with other four-moduli sets, such as $S_4^1 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ and $S_4^2 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$, our proposed S_4^4 has the following advantages: (1) it has larger dynamic range. (2) it is valid for all integers of n. (3) it has a more efficient architecture for the reverse converter. (4) RAU with the proposed moduli can also be implemented efficiently. New CRT theorem introduced recently has been employed to exploit the special properties of the proposed moduli set where modulo corrections are done without resorting to the costly and time consuming modulo operations. The resulting architecture is notably simple and can be realized in hardware with only bit reorientation and one MOMA. The new reverse converter has superior area-time complexity in comparison with the reverse converters for several four-moduli sets.

4.2 Algorithm for the Reverse Conversion

Before we can apply the New CRT-I to derive the algorithm for the reverse converter of the proposed four-moduli set, we need to prove that these moduli are in fact pairwise relatively prime for the validity of the RNS. The following theorem is useful for this purpose.

Lemma 4.1: For natural number a, b and c, if $a \mid b$ and $a \mid (b + c)$, then $a \mid c$.
Proof: If $a \mid b$, and $a \mid (b + c)$, then $b = ka$, and $b + c = ma$ for some integers k and m where $k \geq 1$ and $m > 1$. Since $b + c > b$, $m > k$. Let the nature number $n = m - k$, then $c = (b + c) - b = ma - ka = (m - k)a = na$. Therefore, $a \mid c$. □

Theorem 4.1: The moduli from the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ are pairwise relatively prime for any natural number n.

Proof: It has been established that the moduli $2^n - 1$, 2^n and $2^n + 1$ are pairwise relatively prime [Sza67]. Hence, we only need to prove that $2^{2n} + 1$ is relatively prime to the other moduli.

Assume $2^{2n} + 1$ is divisible by 2^n, then $2^{2n} + 1 = k \cdot 2^n$, for some natural number $k > 2^n$. Rearranging the above equation, we have $k \cdot 2^n - 2^{2n} = 1 \Rightarrow 2^n \cdot (k - 2^n) = 1$. Since $k > 2^n \Rightarrow k - 2^n > 0$ and 2^n is an even natural number, the identity $2^n \cdot (k - 2^n) = 1$ is invalid. By contradiction, $2^{2n} + 1$ is not divisible by 2^{2n}. Hence, $2^{2n} + 1$ is coprime to 2^n.

Assume that $2^{2n} + 1$ is not coprime to $2^n + 1$, there exists a natural number $a \geq 2$ such that $a = gcd(2^n + 1, 2^{2n} + 1)$. Thus, we have $a \mid 2^{2n} + 1$ and $a \mid 2^n + 1$. Since $2^{2n} + 1 = 2^n(2^n - 1) + 2^n + 1$, $a \mid 2^{2n} + 1$ implies that $a \mid (2^n(2^n - 1) + 2^n + 1)$. According to *Lemma 4.1* and the assumption, $a \mid 2^n(2^n - 1)$. The divisibility property [Rob93] suggests that either (i) $a \mid 2^n$ or (ii) $a \mid 2^n - 1$. (i) If $a \mid 2^n$, based on the above assumption that $a \mid 2^n + 1$, a is the common divisor of $2^n + 1$ and 2^n. This contradicts the well-established fact that $2^n + 1$ is relatively prime to 2^n. (ii) if $a \mid 2^n - 1$, based on the assumption, $a \mid 2^n + 1$ and a is the common divisor of $2^n + 1$ and $2^n - 1$. This is, again, a contradiction to the proven fact that $2^n + 1$ is relatively prime to $2^n - 1$. Since

both cases cannot be established, the assumption that $2^n + 1$ is not prime to $2^{2n} + 1$ is invalid. Therefore, $2^n + 1$ is relatively prime to $2^{2n} + 1$.

Assume that $2^{2n} + 1$ is not coprime to $2^n - 1$, there exists a natural number $a \geq 2$ such that $a = gcd\,(2^n - 1, 2^{2n} + 1)$. So we have $a \mid 2^{2n} + 1$ and $a \mid 2^n - 1$. Since $2^{2n} + 1 = 2^n(2^n - 1) + 2^n + 1$, $a \mid 2^{2n} + 1$ implies that $a \mid (2^n(2^n - 1) + 2^n + 1)$. According to *Lemma 4.1* and the assumption, we have $a \mid 2^n + 1$, so a is the common divisor of $2^n + 1$ and $2^n - 1$. This is in contradiction with the fact that $2^n - 1$ and $2^n + 1$ are relatively prime. Hence we rule out the assumption and conclude that $2^{2n} + 1$ is coprime to $2^n - 1$.

Since $2^{2n} + 1$ is pairwise relatively prime to all the other moduli in the set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, the relative primality of the proposed four moduli set is established. □

With the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, the reverse conversion algorithm from the residues (x_1, x_2, x_3, x_4) to the weighted binary number X is derived in the sequent based on the New CRT-I. Let $P_1 = 2^n$, $P_2 = 2^n + 1$, $P_3 = 2^{2n} + 1$ and $P_4 = 2^n - 1$. According to *Theorem 2.2*, X can be calculated by

$$X = x_1 + P_1\left|k_1(x_3 - x_2) + k_2P_2(x_4 - x_3) + k_3P_2P_3(x_1 - x_4)\right|_{P_2P_3P_4} \tag{4-1}$$

where

$$\left|k_1 \cdot 2^n\right|_{(2^n+1)(2^{2n}+1)(2^n-1)} = 1 \tag{4-2}$$

$$\left|k_2 \cdot 2^n(2^n + 1)\right|_{(2^n-1)(2^{2n}+1)} = 1 \tag{4-3}$$

$$\left|k_3 \cdot 2^n(2^{2n} + 1)(2^n + 1)\right|_{2^n-1} = 1 \tag{4-4}$$

The three multiplicative inverses, k_1, k_2 and k_3 are given as follows:

$$k_1 = 2^{3n} \tag{4-5a}$$

$$k_2 = 2^{3n-2} + 2^{2n-1} - 2^{n-2} \tag{4-5b}$$

$$k_3 = 2^{n-2} \tag{4-5c}$$

The above closed-form expressions of the multiplicative inverses are validated as follows:

Proof of (4-5a):

$$\left|k_1 \cdot 2^n\right|_{(2^n+1)(2^{2n}+1)(2^n-1)} = \left|2^{3n} \cdot 2^n\right|_{2^{4n}-1} = \left|2^{4n}\right|_{2^{4n}-1} = \left|2^{4n} - 1 + 1\right|_{2^{4n}-1} = 1$$ □

Proof of (4-5b):

$$\left|k_2 \cdot 2^n\left(2^n+1\right)\right|_{(2^n-1)(2^{2n}+1)} = \left|\left(2^{3n-2} + 2^{2n-1} - 2^{n-2}\right) \cdot 2^n\left(2^n+1\right)\right|_{2^{3n}-2^{2n}+2^n-1}$$
$$= \left|2^{5n-2} + 2^{4n-1} - 2^{3n-2} + 2^{4n-2} + 2^{3n-1} - 2^{2n-2}\right|_{2^{3n}-2^{2n}+2^n-1}$$
$$= \left|-2^{2n-2}\left(2^{3n} - 2^{2n} + 2^n - 1\right) + 2^{5n-2} + 2^{4n-1} - 2^{3n-2} + 2^{4n-2} + 2^{3n-1} - 2^{2n-2}\right|_{2^{3n}-2^{2n}+2^n-1}$$
$$= \left|2^{4n}\right|_{2^{3n}-2^{2n}+2^n-1} = \left|2^n\left(2^{3n} - 2^{2n} + 2^n - 1\right) + 2^{3n} - 2^{2n} + 2^n\right|_{2^{3n}-2^{2n}+2^n-1}$$
$$= \left|2^{3n} - 2^{2n} + 2^n\right|_{2^{3n}-2^{2n}+2^n-1} = 1$$

Proof of (4-5c):

It is trivial to show that $\left|2^n\right|_{2^n-1} = 1$. Using *Property 2.4* and *2.5*, we have

$$\left|2^n + 1\right|_{2^n-1} = 2, \quad \left|2^{2n} + 1\right|_{2^n-1} = \left|2^n \cdot 2^n + 1\right|_{2^n-1} = 2.$$

Thus,

$$\left|k_3 \cdot 2^n\left(2^{2n}+1\right)\left(2^n+1\right)\right|_{2^n-1} = \left|2^{n-2} \cdot 2^n\left(2^{2n}+1\right)\left(2^n+1\right)\right|_{2^n-1}$$
$$= \left|2^{n-2} \cdot 1 \cdot 2 \cdot 2\right|_{2^n-1} = \left|2^n\right|_{2^n-1} = 1$$ □

Theorem 4.2: In a RNS defined by the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, the weighted binary number X can be calculated from the residues (x_1, x_2, x_3, x_4) by:

$$X = x_2 + 2^n \cdot \left|A \cdot x_1 + B \cdot x_2 + C \cdot x_3 + D \cdot x_4\right|_{2^{4n}-1} \tag{4-6}$$

where

$$A = 2^{n-2}\left(2^{3n} + 2^{2n} + 2^n + 1\right) \tag{4-7a}$$
$$B = -2^{3n} \tag{4-7b}$$
$$C = 2^{3n} - \left(2^n + 1\right)\left(2^{3n-2} + 2^{2n-1} - 2^{n-2}\right) \tag{4-7c}$$
$$D = \left(2^n + 1\right)\left(2^{3n-2} + 2^{2n-1} - 2^{n-2}\right) - 2^{n-2}\left(2^n + 1\right)\left(2^{2n} + 1\right) \tag{4-7d}$$

Proof: By substituting $P_1 = 2^n$, $P_2 = 2^n + 1$, $P_3 = 2^{2n} + 1$, $P_4 = 2^n - 1$ and the values of

k_1 to k_3 from Equations (4-5a) to (4-5c) into Equation (4-1), we have

$$X = x_1 + 2^n \left| \begin{matrix} 2^{3n}(x_3 - x_2) + (2^{3n-2} + 2^{2n-1} - 2^{n-2})(2^n + 1)(x_4 - x_3) \\ + 2^{n-2}(2^n + 1)(2^{2n} + 1)(x_1 - x_4) \end{matrix} \right|_{2^{4n}-1}$$

It should be noted that the residues corresponding to P_1, P_2, P_3 and P_4 are respectively x_2, x_3, x_4 and x_1. The result follows directly by expanding the terms and grouping the coefficients of x_1, x_2, x_3 and x_4. □

4.3 Hardware Realization of the Reverse Converter

We use *Theorem 4.2* as the basis to devise a VLSI efficient reverse converter for the proposed four-moduli set. Before mapping it into a hardware architecture, the expressions of Equations (4-6) and (4-7) can be further simplified to reduce the hardware complexity by exploiting the special properties of the modular M operation.

Let the residues x_1, x_2, x_3 and x_4 be represented by binary strings of different lengths as follows:

$$x_1 = (X_{1,n-1}X_{1,n-2}...X_{1,1}X_{1,0})_2$$
$$x_2 = (X_{2,n-1}X_{2,n-2}...X_{2,1}X_{2,0})_2$$
$$x_3 = (X_{3,n}X_{3,n-1}...X_{3,1}X_{3,0})_2$$
$$x_4 = (X_{4,2n}X_{4,2n-1}...X_{4,1}X_{4,0})_2$$

Furthermore, let

$$Y = \left| \sum_{i=1}^{4} \beta_i \right|_{2^{4n}-1}$$

where

$$\beta_1 = |A \cdot x_1|_{2^{4n}-1}$$
$$\beta_2 = |B \cdot x_2|_{2^{4n}-1}$$
$$\beta_3 = |C \cdot x_3|_{2^{4n}-1}$$
$$\beta_4 = |D \cdot x_4|_{2^{4n}-1}$$

According to *Theorem 4.2*, X can be calculated by

$$X = x_2 + 2^n \cdot Y. \tag{4-8}$$

In what follows, we will apply *Property 2.10* recursively to replace β_i by the re-arrangement of bits from the residues with interleaving strings of binary constants, '0' and '1'. To reduce the complexity of the resulting expressions, a shorthand notation introduced in [Ska98] is used. The shorthand notation $(abcd)^{(k)}$ indicates a string of $4k$-bit vector where the four-bit string '*abcd*' is repeated k times.

Evaluation of β_1

$$\begin{aligned}\beta_1 &= \left|2^{n-2}\left(2^{3n}+2^{2n}+2^n+1\right)\cdot x_1\right|_{2^{4n}-1}\\ &= \left|CLS_{4n}\left\{\left(CLS_{4n}(x_1,3n)+CLS_{4n}(x_1,2n)+CLS_{4n}(x_1,n)+x_1\right),n-2\right\}\right|_{2^{4n}-1}\\ &= CLS_{4n}\left(X_{1,n-1}\ldots X_{1,0}X_{1,n-1}\ldots X_{1,0}X_{1,n-1}\ldots X_{1,0}X_{1,n-1}\ldots X_{1,0},n-2\right)\\ &= \left(X_{1,1}X_{1,0}\left(X_{1,n-1}\ldots X_{1,0}\right)^{(3)}X_{1,n-1}\ldots X_{1,2}\right)_2\end{aligned} \tag{4-9}$$

Evaluation of β_2

$$\beta_2 = \left|-2^{3n}\cdot x_2\right|_{2^{4n}-1} = \left|2^{4n}-1-CLS_{4n}(x_2,3n)\right|_{2^{4n}-1} = \left(\overline{X}_{2,n-1}\ldots\overline{X}_{2,0}(1)^{(3n)}\right)_2 \tag{4-10}$$

Evaluation of β_3

$$\begin{aligned}\beta_3 &= \left|C\cdot x_3\right|_{2^{4n}-1} = \left|\left\{2^{3n}-\left(2^{4n-2}+2^{3n-1}-2^{2n-2}+2^{3n-2}+2^{2n-1}-2^{n-2}\right)\right\}\cdot x_3\right|_{2^{4n}-1}\\ &= \left|\left\{-\left(2^{4n-2}+2^{2n-2}\right)+\left(2^{3n-2}+2^{n-2}\right)\right\}\cdot x_3\right|_{2^{4n}-1}\end{aligned}$$

If we define

$$\beta_{3,1} = \left|-\left(2^{4n-2}+2^{2n-2}\right)\cdot x_3\right|_{2^{4n}-1}$$

$$\beta_{3,2} = \left|\left(2^{3n-2}+2^{n-2}\right)\cdot x_3\right|_{2^{4n}-1}$$

so that $\beta_3 = \left|\beta_{3,1}+\beta_{3,2}\right|_{2^{4n}-1}$. Then, $\beta_{3,1}$ and $\beta_{3,2}$ can be evaluated by recursive applications of *Property 2.10*.

$$\begin{aligned}\beta_{3,1} &= \left|-\left(2^{4n-2}+2^{2n-2}\right)\cdot x_3\right|_{2^{4n}-1} = \left|-2^{2n-2}\left(2^{2n}+1\right)\cdot x_3\right|_{2^{4n}-1}\\ &= \left|-CLS_{4n}\left(x_3+CLS_{4n}(x_3,2n),2n-2\right)\right|_{2^{4n}-1}\end{aligned}$$

where

$$x_3 + CLS_{4n}(x_3, 2n) = \left((0)^{(n-1)} X_{3,n} \ldots X_{3,0} (0)^{(n-1)} X_{3,n} \ldots X_{3,0}\right)_2$$

$$CLS_{4n}(x_3 + CLS_{4n}(x_3, 2n), 2n-2) = \left(X_{3,1} X_{3,0} (0)^{(n-1)} X_{3,n} \ldots X_{3,0} (0)^{(n-1)} X_{3,n} \ldots X_{3,2}\right)_2$$

Eventually, we have

$$\begin{aligned} \beta_{3,1} &= \left|-CLS_{4n}(x_3 + CLS_{4n}(x_3, 2n), 2n-2)\right|_{2^{4n}-1} \\ &= \left|2^{4n} - 1 - CLS_{4n}(x_3 + CLS_{4n}(x_3, 2n), 2n-2)\right|_{2^{4n}-1} \\ &= \left(\overline{X}_{3,1} \overline{X}_{3,0} (1)^{(n-1)} \overline{X}_{3,n} \ldots \overline{X}_{3,0} (1)^{(n-1)} \overline{X}_{3,n} \ldots \overline{X}_{3,2}\right)_2 \end{aligned} \quad (4\text{-}11)$$

Similarly,

$$\begin{aligned} \beta_{3,2} &= \left|\left(2^{3n-2} + 2^{n-2}\right) \cdot x_3\right|_{2^{4n}-1} = \left|2^{n-2}\left(2^{2n} + 1\right) \cdot x_3\right|_{2^{4n}-1} \\ &= CLS_{4n}(x_3 + CLS_{4n}(x_3, 2n), n-2) \\ &= \left(0 X_{3,n} \cdots X_{3,0} (0)^{(n-1)} X_{3,n} \cdots X_{3,0} (0)^{(n-2)}\right)_2 \end{aligned} \quad (4\text{-}12)$$

Evaluation of β_4

$$\begin{aligned} \beta_4 &= \left|\left\{\left(2^n + 1\right)\left(2^{3n-2} + 2^{2n-1} - 2^{n-2}\right) - 2^{n-2}\left(2^n + 1\right)\left(2^{2n} + 1\right)\right\} \cdot x_4\right|_{2^{4n}-1} \\ &= \left|\left(2^{3n-1} - 2^{n-1}\right) \cdot x_4\right|_{2^{4n}-1} = \left|\beta_{4,1} + \beta_{4,2}\right|_{2^{4n}-1} \end{aligned}$$

where

$$\begin{aligned} \beta_{4,1} &= \left|2^{3n-1} \cdot x_4\right|_{2^{4n}-1} = CLS_{4n}(x_4, 3n-1) \\ &= \left(X_{4,n} \cdots X_{4,0} (0)^{(2n-1)} X_{4,2n} \cdots X_{4,n+1}\right)_2 \end{aligned} \quad (4\text{-}13)$$

$$\begin{aligned} \beta_{4,2} &= \left|-2^{n-1} \cdot x_4\right|_{2^{4n}-1} = \left|2^{4n-1} - 1 - CLS_{4n}(x_4, n-1)\right|_{2^{4n}-1} \\ &= \left((1)^{(n)} \overline{X}_{4,2n} \ldots \overline{X}_{4,0} (1)^{(n-1)}\right)_2 \end{aligned} \quad (4\text{-}14)$$

There are $3n$ '1's in Equation (4-10), $2n - 2$ '1's in Equation (4-11), $2n - 2$ '0's in Equation (4-12), $2n - 1$ '0's in Equation (4-13), and $2n - 1$ '1's in Equation (4-14). Equations (4-10) and (4-14) can be merged because there are $4n$ consecutive '1's, which can be eliminated because of the modulo $2^{4n} - 1$ addition. Rearrange the other terms, we can obtain the following expression.

$$Y = \left| \sum_{i=1}^{5} \alpha_i \right|_{2^{4n}-1} \tag{4-15}$$

where

$$\alpha_1 = \left(X_{1,1}X_{1,0}\left(X_{1,n-1}...X_{1,0}\right)^{(3)}X_{1,n-1}...X_{1,2}\right)_2 \tag{4-16a}$$

$$\alpha_2 = \left(\overline{X}_{2,n-1} \cdots \overline{X}_{2,0}\overline{X}_{4,2n} \cdots \overline{X}_{4,0}X_{4,2n-1} \cdots X_{4,n+1}\right)_2 \tag{4-16b}$$

$$\alpha_3 = \left(\overline{X}_{3,1}\overline{X}_{3,0}X_{4,n-2} \cdots X_{4,0}\overline{X}_{3,n} \cdots \overline{X}_{3,0}X_{3,n-1}...X_{3,1}\overline{X}_{3,n} \cdots \overline{X}_{3,2}\right)_2 \tag{4-16c}$$

$$\alpha_4 = \left(0X_{3,n}...X_{3,0}(0)^{(n-1)}X_{3,n}(1)^{(n-1)}X_{3,1}(0)^{(n-2)}\right)_2 \tag{4-16d}$$

$$\alpha_5 = \left(X_{4,n}X_{4,n-1}(1)^{(n-1)}(0)^{(2n-1)}X_{4,2n}(1)^{(n-1)}\right)_2 \tag{4-16e}$$

From Equations (4-15) and (4-16), it can be seen that the calculation of X is magically simple and elegant. It involves only one (5, 2^{4n} – 1)-MOMA, which can be implemented as efficiently as the algorithm of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ with the aid of *Property 2.10*. Similar efficient architecture as that used in the triple-moduli set can be employed to realize our new reverse converter.

Example 4.1: For $n = 3$, the moduli set is {7, 8, 9, 65}. Let $X = (3, 6, 7, 20)$. In binary string notation, $x_1 = (011)_2$, $x_2 = (110)_2$, $x_3 = (0111)_2$, $x_4 = (001\ 0100)_2$. Based on Equation (4-16), $\alpha_1 = (11\ 011\ 011\ 011\ 0)_2 = 3510$, $\alpha_2 = (001\ 1101011\ 01)_2 = 941$, $\alpha_3 = (00\ 00\ 1000\ 11\ \ 10)_2 = 142$, $\alpha_4 = (0\ 0111\ 00\ 0111\ 0)_2 = 910$, $\alpha_5 = (0111\ 0000\ 0011)_2 = 1795$. From Equation (4-15), $Y = |\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5|_{4095} = 3203$. From Equation (3-8), $X = x_2 + 2^3 \times Y = 6 + 8 \times 3203 = 25630$. The causality is verified by $|25630|_7 = 3$, $|25630|_8 = 6$, $|25630|_9 = 7$ and $|25630|_{65} = 20$.

Figure 4.1 shows the architecture of our reverse converter for the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$. The bits orientation block generates α_1, α_2, α_3, α_4, α_5 by simply manipulating the routings of the bits from the input residue numbers of x_1, x_2, x_3 and x_4. The summation can be done by (5, 2^{4n} – 1)-MOMA, which consists of a $4n$-bit 3-level CSA with EAC, and a $4n$-bit 1's complement adder. It should be note that as Y is weighted by 2^n, addition to x_2 in Equation (4-8) incurs no additional hardware and computation cost since x_2 can be directly wired to the right of Y.

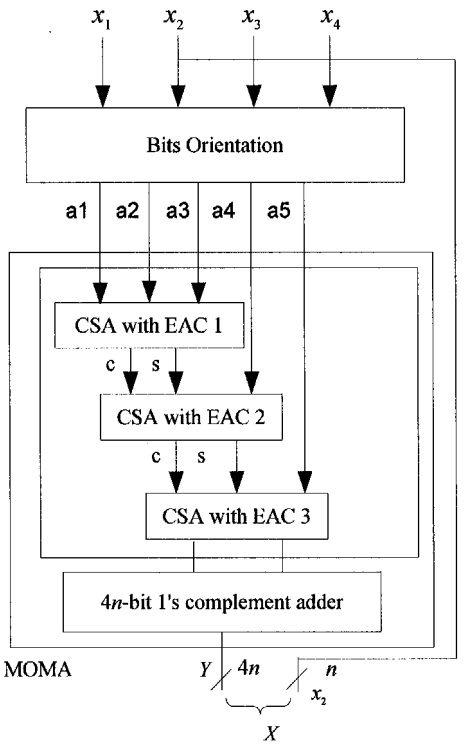


Figure 4.1 *Proposed reverse converter for the four-moduli set* S_4^4

4.4 Performance Evaluation and Comparison

In this section, we will estimate the hardware costs and evaluate the delay of the reverse converter. For ease of comparison with the published results of the relevant reverse converters in this research field, the standard practice of measuring complexity in terms of the number and delay of fundamental logic units like full adder (FA), standard logic gates, etc., is adopted for the reverse converters with generic n. The wire loads are normally neglected.

For the Bits Orientation block of Figure 4.1, $3n+1$ inverters are used for α_2 and $n + 1$ for α_3. The total number of inverters is $4n + 2$. The delay of this block is t_{INV}, which is the delay of an inverter. The delay of the CSA with EAC is $3t_{FA}$ where t_{FA} is the delay of a FA. The number of FAs of CSAs with EAC is $12n$. Since there are some '1's and '0's embedded in α_4 and α_4, according to section 2.1.4.1, some of the FAs of Adder2 and Adder 3 can be simplified.

It is obvious that the critical delay of the entire reverse converter is dominated by the delay of the CPA. If the timing is not critical, then the architectures proposed in

[Efst94] or [Dimi03] can be used for the 1's complement CPA. Otherwise, for the high-speed realization, the method proposed in [Bha98] can be used at the expense of doubling the amount of hardware, by breaking the $4n$-bit carry propagation chain of the final $4n$-bit 1's complement CPA into $2n$-bit carry propagation chains.

As opposed to the other four-moduli supersets, such as $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ [Bha99], and $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [Vin00], which require the parameter, n to be either odd and even, respectively, our new moduli set is co-prime for every nature n. Furthermore, the reverse converters for these two moduli sets require different kinds of adders, subtractors, and even ROMs for their implementation. Therefore, their structures are more complex than our structure shown in Figure 4.1. Actually, the architecture of the reverse converter of our proposed four moduli set is similar to that of the reverse converter for the triple-moduli set [Pie95] shown in Figure 4.2.

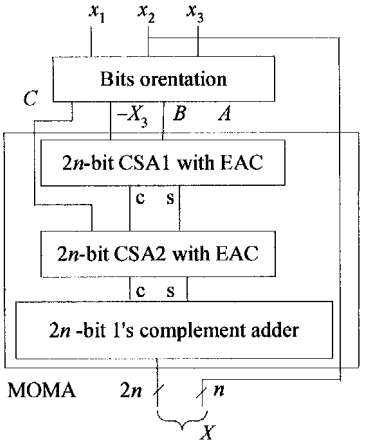


Figure 4.2 *Reverse converter of the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$[Pie95]*

Table 4.1 summarizes the hardware requirements of the reverse converters for these four-moduli sets. Only the number of adders and their operand width are considered for the converter of [Vin00] and ours. This is because for these two converters, the total hardware costs are contributed predominantly by the adders. The simplification method of FAs is also adopted for evaluation of the area complexity of [Vin00]. Furthermore, we assume that the modular adder proposed in [Bay87] will be used for the modular adder/subtractor in [Vin00]. Due to heterogeneous operand width and varying types of adders and subtractors used in [Vin00], it is conjectured that the scalability and modularity of [Vin00] are inferior to ours. The reverse converter in

[Bha98] is implemented with ROM and modular adders, the area complexity of $O(n^2)$ is reported as the equivalent area occupied by full adders from the fabricated circuits.

Table 4.1 *Comparisons of hardware requirements for the reverse converters of* S_4^4

Converters	Hardware cells		Operand width	Amount	Total area
[Vin00]	Binary Adder		$n+2$	1	$O(37n)$
	Binary Adder		$n+3$	1	
	Binary Adder		$2n+3$	2	
	Binary Adder		$3n$	1	
	Binary Adder		$3n+1$	1	
	Binary Subtractor		$n+3$	1	
	Binary Subtractor		$2n$	1	
	Binary Subtractor		$2n+1$	1	
	Binary Subtractor		$3n$	1	
	Binary Subtractor		$5n+1$	1	
	Modular Adder		$2n+3$	1	
	Modular Subtractor		$3n+1$	1	
[Bha98]	N. A.		N. A.	N. A.	$O(n^2)$
Proposed	CE	CPA	$4n$	1	$O(14n)$
		CSA	$4n$	4	
	HS	CPA	$4n$	2	$O(18n)$
		CSA	$4n$	4	

N. A: Not available from [Bha98]

Table 4.2 compares the latencies of these four-moduli set reverse converters. It should be noted that for a specified dynamic range, the value of n for our new converter is generally smaller than the n for the other converters. Therefore, the latencies of our proposed converters are smaller than the other two converters.

The synthesis results provided in Chapter 7 further show that the area, delay and power consumption metrics of converters proposed in this chapter are very close to those for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, and better than those for the reverse converters for S_4^1 and S_4^2. Please refer to the synthesized results in Table 7.6 to Table 7.9, and also the area, total delay and power consumption comparisons in Figure 7.12 to 7.14 of Chapter 7.

Table 4.2 *Delay comparisons of the reverse converters for four-moduli sets*

Converters	Delay
[Vin00]	$t_{MUX} + t_{LUT} + t_{XOR} + (10n + 13)*t_{FA}$
[Bha98]	$(14n + 8)*t_{FA}$
Proposed – HS	$2t_{MUX} + t_{INV} + 2t_{NAND} + (2n + 3)*t_{FA}$
Proposed – CE	$t_{MUX} + t_{INV} + 2t_{NAND} + (8n + 3)*t_{FA}$

4.5 Conclusions

The revolutionary New CRTs have painted a completely different landscape for the RNS research and rekindled the interest to search for new special moduli sets that have never been thought of before. In an earnest attempt to leverage on the strength of the new CRT, we have discovered a new four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$. The new moduli set has a dynamic range of $5n$-bit, and is co-prime for any natural number n. The New CRT I is applied for the development of the reverse conversion algorithm. Closed-form expressions of the New CRT coefficients have been derived. By exploiting the number properties of this moduli set, the new reverse converter completely eliminates the need for large modulo addition with the aid of *Property 2.10*. Only one 5-input MOMA is required which can be easily realized by CSA and 1's complement adder. The complexity analysis of its implementation show that it is as efficient as the most efficient reverse converter for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ while offering a higher degree of parallelism with an additional residue channel.

Chapter 5

A New Five-Moduli Superset and Its Efficient Reverse Converter

5.1 Introduction

This chapter will present a new five-moduli set RNS $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$ and the efficient implementation of its reverse converters. The proposed new moduli set is valid for even n and has a dynamic range of $5n$-bits. This extension from the four-moduli set presented in earlier chapters further increases the parallelism and reduces the size of the each residue channel for a given dynamic range. A part of this chapter has been presented at the 2004 IEEE 2004 International Symposium on Circuits and Systems [Cao04], and a large portion of this chapter is currently being reviewed by the IEEE Transactions on Circuits and Systems I.

In specialized high performance and high precision signal processing applications [Rej00] [Ram00], the level of parallelism provided by the previously presented moduli sets is compromised by the increased dynamic range. The demand for the increased granularity of parallelism and dynamic range calls for the necessity to expand the cardinality of the RNS to beyond four. The challenges of exploring high cardinality special moduli sets are the desiderata of retaining symmetrical or balanced residue channels and the hardware simplicity for the RAUs and the reverse converters at higher parallelism of computation. Unfortunately, the degree of RNS's complexity involved in the hardware implementation of forward/reverse converters and RAUs depends largely on the moduli set. In particular, the reverse converter becomes significantly more difficult to implement efficiently relative to other components of the RNS. For example, if the reverse conversion can only be solved by the traditional CRT, then it is difficult to pipeline the reverse converter, because there is a very large

modular addition. Therefore, the high cardinality moduli set RNS has to solve two problems. One is that the moduli set should possess relatively symmetrical channel widths with low wastage on dynamic ranges. The other is that the reverse converter should be sufficiently fast, or it can be easily pipelined to allow a faster rate of computation. Otherwise, the speedup gained by the RAU will be offset by the long latency of the reverse converters.

To deal with the reverse conversion for high-cardinality moduli sets, Y. Wang proposed the New CRTs [Wang98] [Wang00], which are more efficient than the traditional CRT. However, the large modular additions in the New CRT-based reverse converters cannot be completely eliminated. Skavantzos *et al.* proposed a class of conjugate moduli sets in [Ska99a]. Although they are high-cardinality sets, the moduli are not pairwise relatively prime, resulting in reduced dynamic ranges, asymmetric moduli channel length and long conversion delay. Furthermore, half of the moduli are in the form of $2^n + 1$. These moduli are more efficient than the general moduli, but are otherwise less efficient than those in the form of $2^n - 1$ for the RAUs.

Good high-cardinality sets are hard to come by. Some special five-moduli sets with efficient reverse conversion algorithms have been proposed. Skavantzos proposed a five-moduli set $\{2^{n+1}, 2^n - 1, 2^n + 1, 2^n + 2^{(n+1)/2} + 1, 2^n - 2^{(n+1)/2} + 1\}$ [Ska98]. As two of its moduli are not in the form of 2^n or $2^n \pm 1$, this moduli set is not a superset. The corresponding forward conversion and the residue arithmetic for these two residue channels will be more complex than those for the other 3 residue channels. Mathew *et al.* [Math00] proposed a five-moduli set $\{2^{3n}, 2^{3n} - 1, 2^{3n} + 1, 2^{3n} - 2^{(3n+1)/2} + 1, 2^{3n} + 2^{(3n+1)/2} + 1\}$. Unfortunately, this moduli set suffers from the same disadvantages as [Ska98]. Hiasat [Hia03] proposed another five-moduli set $\{2^{n-2}, 2^n - 1, 2^n + 1, 2^n + 2^{(n+1)/2} + 1, 2^n - 2^{(n+1)/2} + 1\}$ with odd number n, where the reverse converter is more efficient than those for [Ska98] and [Math00]. However, these three moduli sets are very similar and they share similar disadvantages. Skavantzos and Stouraitis [Ska99b] proposed a five-moduli superset $\{2^n, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$, where all the moduli are in the form of 2^n or $2^n \pm 1$ type. Therefore, this moduli set is more advantageous than [Ska98] [Math00] [Hia03]. The disadvantage of this moduli set is that it is not co-prime for any value of n, resulting in the reduced dynamic range and

increased complexity of the reverse conversion algorithm.

In this chapter, we propose a new five-moduli superset $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, which is valid for even values of n. Being a superset, this moduli set is efficient for both the reverse converters and for the modular operations. Furthermore, there is only one modulus in the form of $2^n + 1$. According to [Ska92] [Efst94] [Wan96] [Zimm99] [Ver02] [Dimi03], the operations modulo $2^n - 1$ are more efficient than those for modulo $2^n + 1$. In this perspective, it is better than the five-moduli superset of [Ska99b]. Since the proposed five-moduli set is an extension of the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ presented in chapter 3, we propose an efficient reverse conversion algorithm that is based on the reverse conversion for the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and the MRC technique for a two-moduli set RNS $\{2^n(2^{2n} - 1)(2^{n+1} - 1), 2^{n-1} - 1\}$. The hardware implementation of the proposed reverse converter employs only FAs as the primitive operators. Such adder-based design has the advantage of being design automation friendly. Besides, it can be easily pipelined to attain a high throughput rate.

5.2 Algorithm for the Reverse Conversion

It is trivial to show that the five moduli in the proposed superset, $S = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ is pairwise relatively prime for even value of n. We decompose the superset S into two subsets, $S_1 = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $S_2 = \{2^n(2^{2n}-1)(2^{n+1}-1), 2^{n-1}-1\}$. A reverse converter of the four moduli superset, $S_1 = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ is used to recover an interim integer $X^{(2)}$ from the residues (x_1, x_2, x_3, x_4) of S_1. An efficient reverse converter of S_1 has already been proposed in chapter 3. The final binary equivalent, X of the residues $(x_1, x_2, x_3, x_4, x_5)$ in S can be calculated from the residues $(X^{(2)}, x_5)$ correspond to the two-moduli set $S_2 = \{2^n(2^{2n}-1)(2^{n+1}-1), 2^{n-1}-1\}$ by the MRC Equation (2-10).

According to the conversion algorithm of chapter 3, the interim integer $X^{(2)} = (x_1, x_2, x_3, x_4)$ can be calculated by

$$X^{(2)} = x_2 + 2^n\left(2^{2n}Z + 2^nY_2 + Y_1 - Z\right) = x_2 + 2^n \cdot T \qquad (5\text{-}1)$$

where Y_1, Y_2 are the output signals of the reverse converter for the triple-moduli set $\{2^n-1, 2^n, 2^n+1\}$, Z is the interim result of the reverse converter for the four-moduli superset S_1. Therefore, T is a $(3n+1)$-bit integer.

By applying the MRC equation (2-10) to the resultant two-moduli set S_2, the binary equivalent X of the proposed superset can be obtained from its residues by:

$$X = X^{(2)} + 2^n\left(2^{2n}-1\right)\left(2^{n+1}-1\right)\left|k_0\left(x_5 - X^{(2)}\right)\right|_{2^{n-1}-1} \tag{5-2}$$

where k_0 is the multiplicative inverse of $2^n(2^{2n}-1)(2^{n+1}-1)$ modulo $2^{n-1}-1$, thus

$$\left|k_0 2^n\left(2^{2n}-1\right)\left(2^{n+1}-1\right)\right|_{2^{n-1}-1} = 1 \tag{5-3}$$

Since

$$\left|2^n\right|_{2^{n-1}-1} = \left|2\left(2^{n-1}-1\right)+2\right|_{2^{n-1}-1} = |2|_{2^{n-1}-1},$$

$$\left|2^{2n}-1\right|_{2^{n-1}-1} = \left|2^{n+1}\left(2^{n-1}-1\right)+2\cdot 2^n-1\right|_{2^{n-1}-1} = |3|_{2^{n-1}-1},$$

the left-hand side of the above equation can be simplified as follows:

$$\left|k_0 2^n\left(2^{2n}-1\right)\left(2^{n+1}-1\right)\right|_{2^{n-1}-1} = \left|k_0\cdot 2\cdot 3\cdot 3\right|_{2^{n-1}-1} = \left|18k_0\right|_{2^{n-1}-1} \tag{5-4}$$

The solution to Equation (5-4) is provided by the following Lemma.

Lemma 5.1: For any even number n ($n > 2$), the solution of the modular equation

$$\left|18\cdot k_0\right|_{2^{n-1}-1} = 1 \tag{5-5}$$

is given by:

when $n = 6k-2$, $k = 1, 2, 3, \ldots,$

$$k_0 = 2^{n-3} + \frac{2}{9}\left(2^{n-4}-1\right) = 2^{6k-5} + \frac{2}{9}\left(2^{6k-6}-1\right) \tag{5-6}$$

when $n = 6k$, $k = 1, 2, 3, \ldots,$

$$k_0 = 2^{n-2} + \frac{1}{9}\left(2^{n-1} - 5\right) = 2^{6k-2} + \frac{1}{9}\left(2^{6k-1} - 5\right) \tag{5-7}$$

when $n = 6k + 2$, $k = 1, 2, 3, \ldots,$

$$k_0 = 2^{n-2} + \frac{8}{9}\left(2^{n-2} - 1\right) = 2^{6k} + \frac{8}{9}\left(2^{6k} - 1\right) \tag{5-8}$$

Proof of (5-6): When $n = 6k - 2$, the left-hand side of Equation (5-5) becomes:

$$\left|18 \cdot \left(2^{6k-5} + \frac{2}{9}\left(2^{6k-6} - 1\right)\right)\right|_{2^{6k-3}-1} = \left|5 \cdot 2^{6k-3} - 4\right|_{2^{6k-3}-1} = \left|5 \cdot \left(2^{6k-3} - 1\right) + 1\right|_{2^{6k-3}-1} = 1$$

Proof of (5-7): When $n = 6k$, the left-hand side of Equation (5-5) becomes:

$$\left|18 \cdot \left(2^{6k-2} + \frac{1}{9}\left(2^{6k-1} - 5\right)\right)\right|_{2^{6k-1}-1} = \left|11 \cdot 2^{6k-1} - 10\right|_{2^{6k-1}-1} = \left|11 \cdot \left(2^{6k-1} - 1\right) + 1\right|_{2^{6k-1}-1} = 1$$

Proof of (5-8): When $n = 6k + 2$, the left-hand side of Equation (5-5) becomes:

$$\left|18 \cdot \left(2^{6k} + \frac{8}{9}\left(2^{6k} - 1\right)\right)\right|_{2^{6k+1}-1} = \left|17 \cdot 2^{6k+1} - 16\right|_{2^{6k+1}-1} = \left|17 \cdot \left(2^{6k+1} - 1\right) + 1\right|_{2^{6k+1}-1} = 1$$

This completes the proof.

Although Equation (5-5) can be solved by *Lemma 5.1*, the closed-form expressions provided by *Lemma 5.1* in their original forms are not amenable to hardware realization. For efficient hardware implementation, the expressions of k_0 under the three different ranges of n are expanded into differences of geometrical series so that *Property 2.10* and *Property 2.11* can be exploited to eliminate the logic circuits needed to implement the special power-of-two multiplications modulo $2^n - 1$.

For the case of $n = 6k - 2$,

$$\begin{aligned} k_0 &= 2^{6k-5} + \frac{2}{9}\left(2^{6k-6} - 1\right) = 2^{6k-5} + \sum_{i=0}^{k-2}\left(2^{6i+4} - 2^{6i+1}\right) \\ &= 2^{n-3} + \left(2^4 + 2^{6\cdot 1+4} + \ldots + 2^{6\cdot(k-2)+4}\right) - \left(2^1 + 2^{6\cdot 1+1} + \ldots + 2^{6\cdot(k-2)+1}\right) \end{aligned} \tag{5-9}$$

When $n = 6k$,

$$k_0 = 2^{6k-2} + \frac{1}{9}\left(2^{n-1} - 5\right) = 2^{6k-2} + \sum_{i=0}^{k-1} 2^{6i+2} - \sum_{i=0}^{k-2} 2^{6i+5} - 1$$
$$= 2^{n-2} + \left(2^2 + 2^{6\cdot 1+2} + \ldots + 2^{6\cdot(k-1)+2}\right) - \left(2^0 + 2^5 + 2^{6\cdot 1+5} + \ldots + 2^{6\cdot(k-2)+5}\right) \quad (5\text{-}10)$$

When $n = 6k + 2$,

$$k_0 = 2^{6k} + \frac{8}{9}\left(2^{6k} - 1\right) = \sum_{i=1}^{k}\left(2^{6i} - 2^{6i-1}\right)$$
$$= \left(2^{6\cdot 1} + 2^{6\cdot 2} + \ldots + 2^{6\cdot k}\right) - \left(2^3 + 2^{6\cdot 2-3} + \ldots + 2^{6\cdot k-3}\right) \quad (5\text{-}11)$$

If we let

$$L = \left|\left(x_5 - X^{(2)}\right)\right|_{2^{n-1}-1} \quad (5\text{-}12)$$

and

$$R = \left|k_0\left(x_5 - X^{(2)}\right)\right|_{2^{n-1}-1} = \left|k_0 \cdot L\right|_{2^{n-1}-1}, \quad (5\text{-}13)$$

then Equation (5-4) can be expressed as follows:

$$X = X^{(2)} + 2^n\left(2^{2n} - 1\right)\left(2^{n+1} - 1\right)R \quad (5\text{-}14)$$

By substituting $X^{(2)}$ of Equation (5-1) into Equation (5-14), L can be calculated by

$$L = \left|x_5 - x_2 - 2^n \cdot T\right|_{2^{n-1}-1}$$
$$= \left|x_5 - x_2 - 2\cdot\left(2^{2n}(Z-1) + 2^n Y_2 + Y_1\right)\right|_{2^{n-1}-1} \quad (5\text{-}15)$$
$$= \left|x_5 - x_2 - 6\cdot Z - 4\cdot Y_2 - 2\cdot Y_1\right|_{2^{n-1}-1}$$

For ease of expressing bit level manipulations, the residues are also represented in their binary forms as follows: $x_1 = (X_{1,n-1}X_{1,n-2} \ldots X_{1,0})_2$, $x_2 = (X_{2,n-1}X_{2,n-2} \ldots X_{2,0})_2$, $x_3 = (X_{3,n}X_{3,n-1} \ldots X_{3,0})_2$, $x_4 = (X_{4,n}X_{4,n-1} \ldots X_{4,0})_2$, and $x_5 = (X_{5,n-2}X_{5,n-3} \ldots X_{5,0})_2$. Y_1, Y_2 and Z are similarly expressed in their binary forms as follows: $Y_1 = (Y_{1,n-1}Y_{1,n-2} \ldots Y_{1,0})_2$, $Y_2 = (Y_{2,n-1}Y_{2,n-2} \ldots Y_{2,0})_2$, $Z = (Z_nZ_{n-1} \ldots Z_0)_2$. Furthermore, let $x_2^d = x_2 - 2^{n-1}X_{2,n-1}$, $Z^d = Z - \left(2^n Z_n + 2^{n-1}Z_{n-1}\right)$, $Y_2^d = Y_2 - 2^{n-1}Y_{2,n-1}$, and $Y_1^d = Y_1 - 2^{n-1}Y_{1,n-1}$, where x_2^d, Z^d, Y_2^d, and Y_1^d are the lower (least significant) $n-1$ bits of x_2, Z, Y_2 and Y_1, respectively. By substituting these terms into the expression of L in Equation (5-15), we have

$$L = \left| \begin{matrix} x_5 - x_2^d - 2Z^d - 2^2 Z^d - 2^2 Y_2 - 2Y_1 - 2^{n-1} X_{2,n-1} \\ -2^{n+2} Z_n - 2^{n+1} Z_n - 2^{n+1} Z_{n-1} - 2^n Z_{n-1} - 2^{n+1} Y_{2,n-1} - 2^n Y_{1,n-1} \end{matrix} \right|_{2^{n-1}-1} \quad (5\text{-}16)$$

Property 2.10 and *Property 2.1* can then be used to simplify Equation (5-16) to:

$$L = \left| \sum_{i=1}^{13} L_i \right|_{2^{n-1}-1} \quad (5\text{-}17)$$

where

$$L_1 = \left(X_{5,n-2} X_{5,n-3} \cdots X_{5,1} X_{5,0}\right)_2 \quad (5\text{-}18a)$$

$$L_2 = \left(\overline{X}_{2,n-2} \overline{X}_{2,n-3} \cdots \overline{X}_{2,1} \overline{X}_{2,0}\right)_2 \quad (5\text{-}18b)$$

$$L_3 = CLS_{n-1}\left(\overline{Z}^d, 1\right) = \left(\overline{Z}_{n-3} \ldots \overline{Z}_1 \overline{Z}_0 \overline{Z}_{n-2}\right)_2 \quad (5\text{-}18c)$$

$$L_4 = CLS_{n-1}\left(\overline{Z}^d, 2\right) = \left(\overline{Z}_{n-4} \ldots \overline{Z}_0 \overline{Z}_{n-2} \overline{Z}_{n-3}\right)_2 \quad (5\text{-}18d)$$

$$L_5 = CLS_{n-1}\left(\overline{Y}_2^d, 2\right) = \left(\overline{Y}_{2,n-4} \ldots \overline{Y}_{2,0} \overline{Y}_{2,n-2} \overline{Y}_{2,n-3}\right)_2 \quad (5\text{-}18e)$$

$$L_6 = CLS_{n-1}\left(\overline{Y}_1^d, 1\right) = \left(\overline{Y}_{1,n-3} \cdots \overline{Y}_{1,1} \overline{Y}_{1,0} \overline{Y}_{1,n-2}\right)_2 \quad (5\text{-}18f)$$

$$L_7 = \left((1)^{(n-2)} \overline{X}_{2,n-1}\right)_2 \quad (5\text{-}18g)$$

$$L_8 = \left((1)^{(n-5)} \overline{Z}_n (1)^{(3)}\right)_2 \quad (5\text{-}18h)$$

$$L_9 = \left((1)^{(n-4)} \overline{Z}_n (1)^{(2)}\right)_2 \quad (5\text{-}18i)$$

$$L_{10} = \left((1)^{(n-4)} \overline{Z}_{n-1} (1)^{(2)}\right)_2 \quad (5\text{-}18j)$$

$$L_{11} = \left((1)^{(n-3)} \overline{Z}_{n-1} (1)^{(1)}\right)_2 \quad (5\text{-}18k)$$

$$L_{12} = \left((1)^{(n-4)} \overline{Y}_{2,n-1} (1)^{(2)}\right)_2 \quad (5\text{-}18l)$$

$$L_{13} = \left((1)^{(n-3)} \overline{Y}_{1,n-1} (1)^{(1)}\right)_2 \quad (5\text{-}18m)$$

In Equations (5-18g) to (5-18m), recall that the notation, $(c)^{(k)}$ from [Ska98] is adopted to represent a constant string where the constant, c is repeated k times. There are some embedded constant strings of '1's in the expressions of L_7 to L_{13}. Therefore the modular summation, M of L_7 to L_{13}, can be simplified substantially. Figure 5.1 (a)

shows the architecture of the modular summation M, where C_M and S_M are the $(n-1)$-bit carry and sum outputs of M. One 7-input CSA with EAC tree is used. After the logic simplification, C_M and S_M can be expressed:

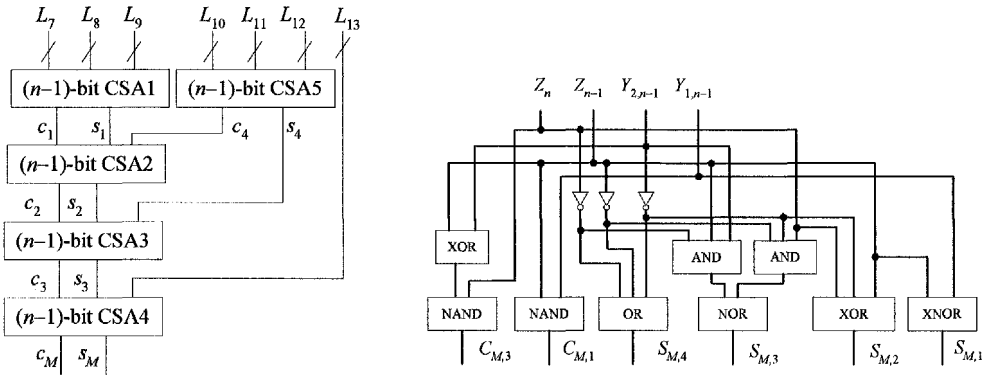
$$C_M = \left((1)^{(n-5)} C_{M,3} (1)^{(1)} C_{M,1} (1)^{(1)}\right)_2 \tag{5-19a}$$

$$S_M = \left((1)^{(n-6)} S_{M,4} S_{M,3} S_{M,2} S_{M,1} \overline{X}_{2,n-1}\right)_2 \tag{5-19b}$$

where $C_{M,3}$, $C_{M,1}$, and $S_{M,1}$ to $S_{M,4}$ can be calculated by the following equations:

$$\begin{aligned}
C_{M,1} &= \overline{Z_{n-1} \cdot Y_{1,n-1}} \\
C_{M,3} &= \overline{Z_n \left(Z_{n-1} \oplus Y_{2,n-1}\right)} \\
S_{M,1} &= \overline{Z_{n-1} \oplus Y_{1,n-1}} \\
S_{M,2} &= Z_n \oplus Z_{n-1} \oplus \overline{Y}_{2,n-1} \\
S_{M,3} &= \overline{\overline{Z}_n Z_{n-1} Y_{2,n-1} + Z_n \overline{Z}_{n-1} \overline{Y}_{2,n-1}} \\
S_{M,4} &= \overline{Z_n \cdot Z_{n-1} \cdot Y_{2,n-1}}
\end{aligned} \tag{5-20}$$

Figure 5.1(b) shows the implementation of Equation (5-20).



(a) *Summation of L_7 to L_{13}* (b) *Simplified calculation of C_M and S_M*

Figure 5.1 *Architecture for the calculation of M, modular summation of L_7 to L_{13}*

Now Equation (5-17) can be rewritten as follows:

$$L = \left| \sum_{i=1}^{6} L_i + 2 \times C_M + S_M \right|_{2^{n-1}-1}. \tag{5-21}$$

Thus, only one $(8, 2^{n-1}-1)$-MOMA is required. The architecture is shown in Figure 5.2. Once L has been obtained, R can be calculated by Equation (5-13) for the three different cases of k_0. When $n = 6k - 2$, the expanded form of k_0 given by Equation (5-

9) is substituted into Equation (5-13) and with the aid of *Property 2.10* and *Property 2.11*, we have

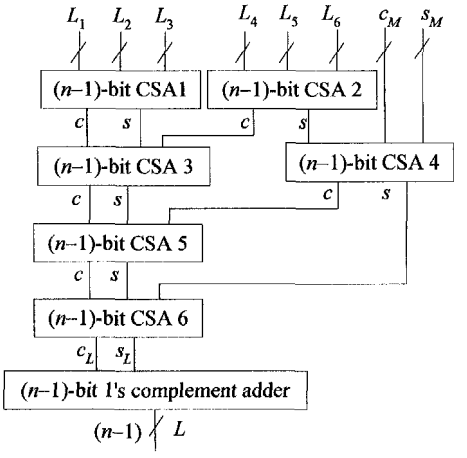


Figure 5.2 *Architecture for the calculation of L for* $\{2^n-1,\ 2^n,\ 2^n+1,\ 2^{n+1}-1,\ 2^{n-1}-1\}$

$$R = \left|\left\{2^{n-3} + \left(2^4 + 2^{6\cdot 1+4} + \cdots 2^{6\cdot(k-2)+4}\right) - \left(2^1 + 2^{6\cdot 1+1} + \cdots 2^{6\cdot(k-2)+1}\right)\right\}\cdot L\right|_{2^{n-1}-1}$$
$$= \left|\begin{matrix} CLS_{n-1}(L, n-3) + CLS_{n-1}(L, 4) + \cdots + CLS_{n-1}(L, 6\cdot(k-2)+4) \\ + CLS_{n-1}(\overline{L}, 1) + CLS_{n-1}(\overline{L}, 7) + \cdots + CLS_{n-1}(\overline{L}, 6\cdot(k-2)+1) \end{matrix}\right|_{2^{n-1}-1} \quad (5\text{-}22)$$

When $n = 6k$, substituting k_0 of Equation (5-10) into Equation (5-13), we have

$$R = \left|\left\{2^{n-2} + \left(2^2 + 2^{6\cdot 1+2} + \cdots 2^{6\cdot(k-1)+2}\right) - \left(2^0 + 2^5 + \cdots + 2^{6\cdot(k-2)+5}\right)\right\}\cdot L\right|_{2^{n-1}-1}$$
$$= \left|\begin{matrix} CLS_{n-1}(L, n-2) + CLS_{n-1}(L, 2) + \cdots + CLS_{n-1}(L, 6\cdot(k-1)+2) \\ + \overline{L} + CLS_{n-1}(\overline{L}, 5) + CLS_{n-1}(\overline{L}, 11) + \cdots + CLS_{n-1}(\overline{L}, 6\cdot(k-1)+5) \end{matrix}\right|_{2^{n-1}-1} \quad (5\text{-}23)$$

When $n = 6k + 2$, substituting k_0 of Equation (5-11) into Equation (5-13), we have

$$R = \left|\left\{\left(2^6 + 2^{6\cdot 2} + \cdots 2^{6\cdot k}\right) - \left(2^3 + 2^{6\cdot 2-3} + \ldots + 2^{6\cdot k-3}\right)\right\}\cdot L\right|_{2^{n-1}-1}$$
$$= \left|\begin{matrix} CLS_{n-1}(L, 6) + CLS_{n-1}(L, 12) + \cdots + CLS_{n-1}(L, 6\cdot k) \\ + CLS_{n-1}(\overline{L}, 3) + CLS_{n-1}(\overline{L}, 9) + \cdots + CLS_{n-1}(\overline{L}, 6\cdot k-3) \end{matrix}\right|_{2^{n-1}-1} \quad (5\text{-}24)$$

Therefore, one $(2k-1,\ 2^{n-1}-1)$-MOMA is required when $n = 6k - 2$ or $n = 6k + 2$, and one $(2k+1,\ 2^{n-1}-1)$-MOMA is required when $n = 6k$. Figure 5.3 shows the architecture for the calculation of R when $k = 3$ and $n = 6k - 2 = 16$, where only one $(5,\ 2^{15}-1)$-MOMA is required.

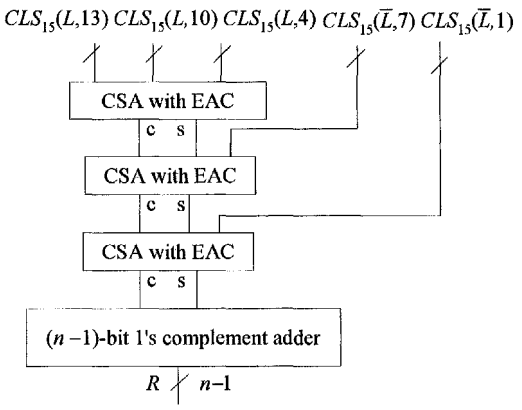


Figure 5.3 *Architecture for the calculation of R when n = 16 (n = 6k – 2)*

After the (n–1)-bit R has been obtained, the final value of X can be calculated by Equation (5-14). Let the binary representation of R be $(R_{n-2}\,R_{n-3}\ldots R_0)_2$. First, let

$$V = \left(2^{2n}-1\right)\left(2^{n+1}-1\right)R\,, \tag{5-25}$$

By expanding the equation of V, we have

$$V = 2^{n+1}\left(2^{2n}R - \left(2^{n-1}R + R\right)\right) + R = 2^{n+1}\cdot U + R \tag{5-26}$$

where U is defined as follows:

$$\begin{aligned} U &= 2^{2n}R - \left(2^{n-1}R + R\right) \\ &= \left(\left(R_{n-2}R_{n-3}\cdots R_1R_0\right)^{(1)}(0)^{(2n)}\right)_2 - \left(\left(R_{n-2}R_{n-3}\cdots R_1R_0\right)^{(2)}\right)_2 . \end{aligned} \tag{5-27}$$

In Equation (5-27), only one (3n–1)-bit binary subtractor is needed for the calculation of U, and the resultant (3n–1)-bit U can be concatenated to the left of R to form the 4n-bit string V by Equation (5-26). By substituting the computed values of $X^{(2)}$ from Equation (5-1) and V from Equation (5-26) into Equation (5-14), we have

$$X = X^{(2)} + 2^n \cdot V = x_2 + 2^n \cdot (T + V) \tag{5-28}$$

A 4n-bit binary adder is required to sum the values of T and V in Equation (5-28) and the sum is concatenated to the left of the residue x_2 to obtain X. Figure 5.4 shows the final architecture of the reverse converter for the proposed five-moduli superset.

Example 5.1: For $n = 6k - 2$ and $k = 2$, then $n = 10$. The proposed five-moduli superset, S is {1023, 1024, 1025, 2047, 511}. Let the RNS representation of X be (100, 200, 100, 30, 6). From the reverse converter for the triple-moduli set {1023, 1024, 1025}, we obtain $Y_1 = (11\ 1111\ 111)_2$ and $Y_2 = (11\ 1001\ 1011)_2$. From the reverse converter for the four-moduli superset, $S_1 = \{1023, 1024, 1025, 2047\}$, $Z = (100\ 0010\ 1100)_2$, $X^{(2)} = (10B\ 39AF\ 4CC8)_{16}$, and $T = (42CE\ 6BD3)_{16}$ are obtained. From Equations (5-18a) to (5-18f), we have $L_1 = (0\ 0000\ 0110)_2$, $L_2 = (1\ 0011\ 0111)_2$, $L_3 = (1\ 1010\ 0111)_2$, $L_4 = (1\ 0100\ 1111)_2$, $L_5 = (1\ 1001\ 0000)_2$ and $L_6 = (0\ 0000\ 0000)_2$. From Equation (5-20), $C_{M,1} = 1$, $C_{M,3} = 0$, $S_{M,1} = 0$, $S_{M,2} = 1$, $S_{M,3} = 1$ and $S_{M,4} = 1$. Thus, according to Equation (5-19), $C_M = (1\ 1111\ 0111)_2$ and $S_M = (1\ 1111\ 1101)_2$. Based on Equations (5-21) and (5-22), $L = (1\ 1011\ 0011)_2$, $R = (1\ 1100\ 0010)_2$. Substituting the value of R into Equation (5-27), we obtain $U = (1C1C\ 7A3E)_{16}$ and Equation (5-26) yields $V = (E0\ E3D1\ F1C2)_{16}$. Finally, by Equation (5-28), $X = (3\ 849A\ 8176\ 54C8)_{16}$.

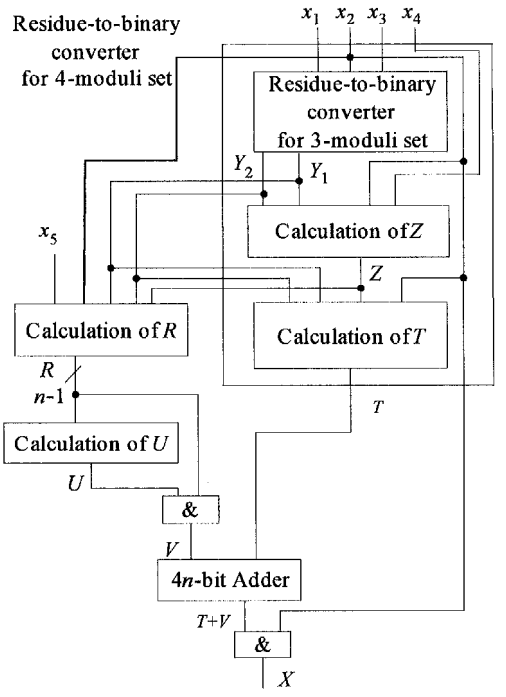


Figure 5.4 *Architecture for the reverse converter for the five-moduli set* $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$

5.3 Performance Evaluation and Comparison

In this section, the performances of our proposed reverse converter for the new five-moduli superset are evaluated theoretically for its area-time complexity. The circuit is also synthesized at gate level to simulate its area, delay and average power consumption and compared against the performances of the reverse converters for other five-moduli sets.

According to Figure 5.4, our proposed reverse converter consists of one four-moduli set reverse converter, calculation of R (including calculation of L), one $(3n-1)$-bit binary subtractor, one $4n$-bit binary adder, and some inverters. The calculation of L consists of one $(8, 2^{n-1}-1)$-MOMA shown in Figure 5.2 and the calculation of C_M and S_M shown in Figure 5.1(b) requires roughly 2 FAs. The calculation of R consist of one $(2k-1, 2^{n-1}-1)$-MOMA for $n = 6k-2$ or $n = 6k+2$, and one $(2k+1, 2^{n-1}-1)$-MOMA for $n = 6k$. When the 4-stage reverse converter for the four-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ is used, the theoretic hardware costs in terms of primitive logic elements such as FA, binary adder and modular adder, are shown in Table 5.1 The total delay of the proposed reverse converter is the sum of the delays of the reverse converter for the triple-moduli set $\{2^n-1, 2^n, 2^n+1\}$, the calculation of Z, calculation of R, the $(3n-1)$-bit subtractor and the $4n$-bit adder. In practical implementation, pipelining is usually applied to achieve a high throughput rate. The maximum pipelining delay is the delay of the $4n$-bit adder by simply inserting latches or registers into the reverse converter without breaking the primitive arithmetic elements.

Table 5.1 *Hardware costs of the proposed reverse converter*

Primitive Elements	$n = 6k-2$	$n = 6k$	$n = 6k+2$
FA	$(5n^2+44n-4)/6$	$(5n^2+52n-12)/6$	$(5n^2+36n+4)/6$
Modulo $2^{n-1}-1$adder	2	2	2
Modulo $2^{n+1}-1$adder	2	2	2
Modulo $2^{2n}-1$adder	1	1	1
$(3n-1)$-bit Subtractor	1	1	1
$(3n+1)$-bit Subtractor	1	1	1
$4n$-bit Adder	1	1	1
Inverter	$6n+1$	$6n+1$	$6n+1$

Since the proposed five-moduli set is an entirely new five-moduli set, it is difficult to make a judicial comparison of its reverse converter against the converters of other

existing five-moduli sets, such as those proposed in [Ska98] [Math00] [Ska99b] [Hia03]. The disadvantages of these moduli sets have been discussed in the introduction section. For the purpose of evaluation, we will provide an impartial comparison of our reverse converter for the new five-moduli superset against those for the existing five-moduli sets [Ska98] [Ska99b] [Hia03]. Following the same basis of comparison adopted in [Hia03], both reverse converters of [Ska98] and [Hia03] were implemented as a stand alone entity and synthesized using the same synthesis tools and digital library as before. For generality, reuse of hardware resources from the RNS processing channels as noted in [Ska98] has not been considered here to avoid the intricate trade-off on performance due to degree of reusability and the accompanying control and timing circuitries.

The implementations of the converters of [Ska99b] and [Hia03] are obtained directly from the algorithms and structures proposed in these papers. Using similar techniques as used in [Math00] and [Hia03], we simplify the modular multipliers for the calculation of A, B, C and D of the converter of [Ska98] into constant multiplications. The simplification is described as follows:

The architecture proposed in [Ska98] consists of two parts. The first part refers to the calculation of A, B, C and D, where four modular multipliers with constants are required. After A, B, C and D have been calculated, Z_1 to Z_{11} can be simply obtained by re-wiring. The second part consists of one (11, 2^{4n} −1)-MOMA. As the implementation of the modular multipliers in the first part was not given in [Ska98], we propose the improved architecture which has been similarly performed by [Math00] for the calculation of A, B, C and D for a fair comparison. According to [Ska98], $A = \left| x_2 \cdot N_2^* \right|_{2^n-1}$, where N_2^* is the multiplicative inverse of $(2^n+1)(2^{2n}+1)$ modulo 2^n-1. By solving the modular equation, we obtain $N_2^* = 2^{n-2}$. Applying *Property 2.10* to the calculation of A gives

$$A = \left| x_2 \cdot 2^{n-2} \right|_{2^n-1} = CLS_n(x_2, n-2), \tag{5-29}$$

This simplification leads to the substitution of the more complex modulo $2^n - 1$ multiplication with constant by a simpler circular shift operation, which can be

achieved by merely re-routing the signals. According to [Ska98], $B=\left|x_3 \cdot N_3^*\right|_{2^n+1}$, $C=\left|x_4 \cdot N_4^*\right|_{2^n+2^{\frac{n+1}{2}}+1}$ and $D=\left|x_5 \cdot N_5^*\right|_{2^n-2^{\frac{n+1}{2}}+1}$, where N_3^* is the multiplicative inverse of $(2^n-1)(2^{2n}+1)$ modulo 2^n+1, N_4^* is the multiplicative inverse of $\left(2^{2n}-1\right)\left(2^n-2^{\frac{n+1}{2}}+1\right)$ modulo $\left(2^n+2^{\frac{n+1}{2}}+1\right)$, and N_5^* is the multiplicative inverse of $\left(2^{2n}-1\right)\left(2^n+2^{\frac{n+1}{2}}+1\right)$ modulo $\left(2^n-2^{\frac{n+1}{2}}+1\right)$. By solving the modular equations, we obtain the multiplicative inverses $N_3^*=2^{n-2}$, $N_4^*=2^{n-2}+2^{\frac{n-5}{2}}$, and $N_5^*=2^{n-2}-2^{\frac{n-5}{2}}$. By exploiting the special formats of the constants, the modular multipliers for the calculation of *B*, *C* and *D* can be further optimized to modular additions but not completely eliminated as in the calculation of *A*. Having proposed notable improvements to the architecture for the reverse converter of [Ska98], we proceed to examine its implications on area, delay and power.

In order to obtain more realistic results, the proposed converter, [Ska98], [Ska98b] and [Hia03] are described by structural VHDL. The EDA tools and the technology libraries are stated in Appendix A.1. The area cost is measured in terms of the number of equivalent NAND gates of the circuit synthesized by the Design Compiler. The total area cost includes both the cell area and the net interconnection area. For completeness, the default power simulation results invoked by Synopsys Design Compiler are included. The dynamic power reported by the Synopsys power compiler models the switching activity in terms of static probability and toggle rate. This power analysis assumed 50% static probability and a toggle rate of 50% of the fastest clock in the design.

Figure 5.5 shows the comparison of the costs of silicon area usage of the reverse converters of our five-moduli superset, Skavantzos' [Ska98], five-moduli superset [Ska99b] and Hiasat's [Hia03], simulated at various dynamic ranges and optimized for minimum silicon area. Figure 5.6 and Figure 5.7 show the total conversion delays and the average power consumptions of these three reverse converters, respectively. The results show that our proposed converter is intermediate between [Hia03] and [Ska98], [Ska99b] for the same dynamic range for all performance metrics except that

when the dynamic range is less than 45-bit, our converter has higher area cost. As the moduli of [Hia03] have been carefully selected in such a way that their closed-form multiplicative inverses are very simple, its reverse converter has the best performance. However, the overall performance of a RNS for an application depends on not only the efficiency of the reverse converter but more critically on the modulo operations in each channel. As the moduli set of [Hia03] is not a superset, some channels of its RNS comprises complicated modulo operations. Although three out of five moduli composing the RNS of [Hia03] are efficient for the residue arithmetic unit, it is the worst-case residue channel that dictates the overall performance of an arithmetic operation in RNS. The critical modulo operations of our proposed RNS and that of [Hia03] are modulo 2^n+1 and modulo $2^n+2^{(n+1)/2}+1$, respectively. Based on the unit gate model (see Section 2.1.4.1), which assumes that each two-input logic gate except the exclusive-OR gate has a delay of unity and the exclusive-OR gate has a delay of 2, the execution latency of addition in modulo $2^n+2^{(n+1)/2}+1$ is approximately $4\log_2 n+7$ designed with the method of [Gar03] as opposed to $2\log_2 n+6$ for modulo 2^n+1 adder of [Imb03]. Even for the same channel width, arithmetic in modulo $2^n\pm1$ is much faster than arithmetic in modulo $2^n+2^{(n+1)/2}+1$. In fact, the execution latencies of the fastest reported modulo 2^n and modulo 2^n-1 adders are equal to $2\log_2 n+3$ [Imb03], making the execution speed in each channel of our proposed moduli set well balanced.

It is noted that the area, delay and power performances of the converter of [Ska98] behave in a quadric function with respect to the dynamic range or the input word-length, whereas the performances of the other two converters vary linearly with the dynamic range. The main reason for the inferior performance of the reverse converter of [Ska98] is that the hardware resources required by the modular multipliers cannot be totally eliminated. Due to the higher hamming weight of the multiplicative inverses for the MRC calculation of [Ska99b], the corresponding MOMAs have more inputs than our algorithm, resulting in an escalation of hardware costs with increase in dynamic ranges. For example, for our proposed algorithm, when $n = 6k$, according to Equations (5-23), one $(n/3+1, 2^{n-1}-1)$ MOMA is required, whereas for the converter of [Ska99b], when $n = 6k - 1$, one $(2n-2, 2^{2n}-1)$ MOMA is required. Not only the size of the MOMA of [Ska99b] is doubled, but also the number of inputs is rapidly expanded to nearly 6 times more than ours.

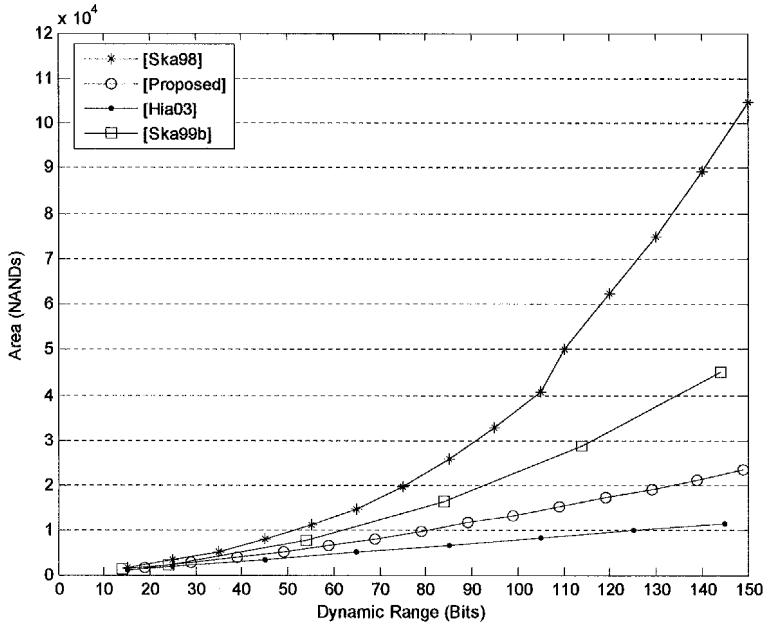


Figure 5.5 *Comparison of area complexity of reverse converters for five-moduli sets*

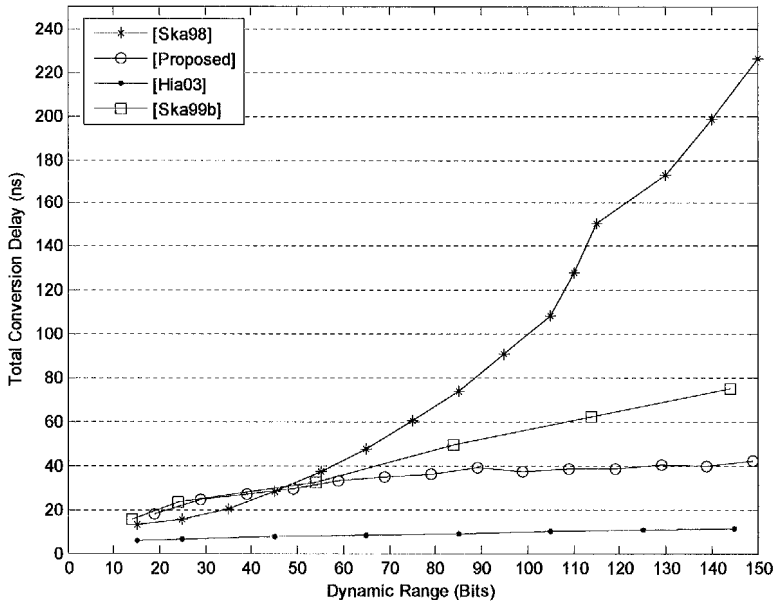


Figure 5.6 *Comparison of worst case conversion delay of reverse converters for five-moduli sets*

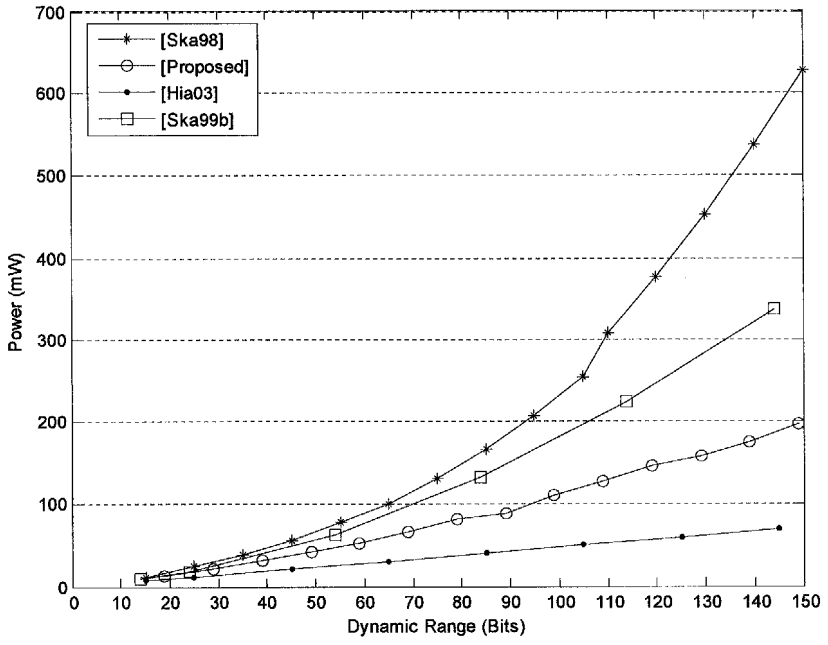


Figure 5.7 *Comparison of average power consumption of reverse converters for five-moduli sets*

5.4 Conclusions

In this chapter, we have proposed a new five-moduli superset $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$ RNS, valid for even values of n. It retains the properties of the popular triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ to provide for increased parallelism and high-speed reverse conversion. When compared with the existing non-superset five-moduli sets, the advantages are obvious because the superset consists of moduli in form of $2^n \pm 1$ is more efficient for the residue arithmetic units. Furthermore, the residue arithmetic units are performed iteratively and the reverse conversion is performed only once when the result is achieved; therefore, the supersets are more suitable for RNS than the non-superset moduli sets. Comparing with the existing non co-prime five-moduli superset, our reverse converter uses less hardware resource due to the elegant multiplicative inverses which have small Hamming weights. Being a fundamentally adder based design, the proposed architecture of the reverse converter benefits from the advanced digital cell-based library and EDA tools. It is more adaptive to optimization provided by silicon compilers under different design constraints, and can be readily pipelined to achieve high throughput rate.

Chapter 6

Modulo 2^n+1 Multioperand Modular Adder

6.1 Introduction

In this chapter, we will introduce a notion of composite carry-save adders with complementary end-around carry (CCSA with CEAC) to generalize the construction of modulo $2^n + 1$ MOMA. The proposed MOMA structure will be used in the implementations for the forward converters and the modulo $2^n + 1$ arithmetics in the next chapter. The work presented in this chapter has been accepted for presentation at the 2005 International Symposium on Circuits and Systems [Cao05c].

Modulo $2^n + 1$ arithmetic is an integral part of RNS's with supersets and is also widely used in digital signal processing, coding and cryptography systems [Sun93] [Zimm94] [Pali99] [Zimm99] [Efst05]. Due to the special number theoretic properties of modulo $2^n \pm 1$ operations, reverse converters for RNS's based on triple moduli set have been proposed and studied in the earlier chapters. This chapter focuses on the special and essential base architectures to the modulo $2^n + 1$ multiplication, i.e., modulo $2^n + 1$ MOMA. An efficient modulo $2^n + 1$ MOMA is not only crucial to the design of modular multiplier for RNS-based applications that use moduli of $2^n + 1$ form, but it is also utilized in the forward converter of such channels. The critical delay in an RNS that is based on the triple-moduli set $\{2^n, 2^n - 1, 2^n + 1\}$ is usually determined by the speed of arithmetic operations in the modulo $2^n + 1$ channels, as demonstrated by the evaluation results of various RNS's in the next chapter. Therefore, a general structure for the design of fast modulo $2^n + 1$ MOMA is of great significance. As MOMAs have already been proposed [Ska89] [Pie94a], and some are also implicitly used in the diminished-one modulo $2^n + 1$ multiplication without a theoretical proof [Curi91] [Zimm94] [Wan95] [Ma98] [Efst05], the aim of this

chapter is to provide new insight into these problems by proposing unified formulation of the generalized architectures from which the VLSI efficient designs are developed.

It is noted from Chapter 2 that MOMA for moduli in the form of $2^n - 1$ has a regular CSA tree followed by a modulo $2^n - 1$ adder. Compared with the modulo $2^n - 1$ operations, fast and hardware efficient modulo $2^n + 1$ operations are more difficult to design and fall short of the performance of the modulo $2^n - 1$ operations. Two approaches to deal with modulo $2^n + 1$ arithmetic, the normal format and the diminished-one format, have been reviewed in Section 2.1.4.4. The diminished-one based method prevails the normal one in MOMA implementation. However, the concomitant difficulty of designing the adder tree of the modulo $2^n + 1$ MOMAs in diminished-one representation remains and a unified mathematical model to the analysis of the diminished-one modulo $2^n + 1$ MOMA is lacking.

In this chapter, we will introduce the concept of composite carry-save adders with complementary end-around carry (CCSA with CEAC) to construct a general modulo $2^n + 1$ MOMA. The notion of auxiliary ports is introduced to form a network of companion residue counters to calculate the constant residues introduced intrinsically by the CEAC tree. With the specific reduction rule of the constant residue counter, we prove that the companion residue counter tree network is redundant as the final constant residue of the MOMA is fixed and can always be absorbed into the final diminished-one CPA. The hardware cost and delay of this highly regular Wallace tree architecture approach those of the modulo $2^n - 1$ CSA tree with EAC. When fast solutions for the two-input modulo $2^n + 1$ adder [Ver02] [Efst03] are employed for its final stage CPA, the proposed modulo $2^n + 1$ MOMA can be as fast as the high-speed modulo $2^n - 1$ MOMA.

6.2 MOMA Modulo $2^n + 1$

6.2.1 Composite CSA with CEAC and Companion Residue Counter

The definitions of MOMA modulo $2^n + 1$ and diminished-one CSA have been given

by Equations (2-19) and (2-20) of Chapter 2. The modulo $2^n - 1$ MOMA consists of a regular Wallace CSA tree with EAC (see Figure 2.6 of chapter 2), and a 2-input modulo $2^n - 1$ adder. The regularity and localized interconnections make the resultant Wallace tree for the modulo $2^n - 1$ MOMA more appealing than its binary counterpart. In order to utilize the same structure for the modulo $2^n + 1$ MOMA using CSA with CEAC, we anatomize the structure of the modulo $2^n - 1$ MOMA. Then, we model the modulo $2^n + 1$ MOMA with a structure similar to that of the modulo $2^n - 1$ MOMA using the notions of composite CSA with CEAC (CCSA with CEAC) and companion residue counter.

First, let us categorize the CSAs with EAC in a modulo $2^n - 1$ MOMA according to their input attributes. Let 'I' denote any addend, '*C*' denote the carry output of a CSA with EAC (*C'* defined by Equation (2-15)) and '*S*' denote the sum output of a CSA with EAC. Altogether seven different CSAs with EAC are classified. (1) Type 3I – all inputs are fed from the addends. (2) Type 2IS – two inputs are fed from the addends and one from the sum output of a CSA in the preceding level. (3) Type ICS – one input are fed from an addend and the other two inputs are fed from the outputs of a CSA in the preceding level. (4) Type 2CS – two inputs are fed from the carry outputs of the CSAs and the other input is fed from the sum output of another CSA in the preceding level. (5) Type C2S – two inputs are fed from the sum outputs of the CSAs and the other input is fed from the carry output of another CSA in the preceding level. (6) Type I2S – two inputs are fed from the sum outputs of two different CSAs in the preceding level and the other input is fed from the addend. (7) Type 3S – all three inputs are fed from the sum outputs from different CSAs in the preceding level.

Although there are 7 different CSAs with EAC within the $2^n - 1$ MOMA, they perform the consistent operation defined by Equations (2-14) and (2-15). On the contrary, for the modulo $2^n + 1$ MOMA, the CSAs with CEAC having different input attributes will behave differently. From Equation (2-19), it is observed that a constant residue of $k - 1$ appears in the diminished-one addition of k addends. Therefore, if standard n-bit CSAs with CEAC are to be used to construct the Wallace tree, the constant residues (e.g., '1' in (2-20)) will have to be cumulated by a separate adder tree for a universal modulo $2^n + 1$ MOMA. In order to construct a universal modulo 2^n

+ 1 MOMA using a unified CSA element, we introduce the notion of a CCSA with CEAC, which consists of a CSA with CEAC and a companion residue counter. The CCSA with CEAC has two sets of ports: primary ports and auxiliary ports, corresponding to the CSA with CEAC and the companion residue counter. The operation of CSA with CEAC is performed via the primary ports while the addition of the constant residues is completed through the auxiliary ports. The primary port holds a diminished-one value, and the corresponding auxiliary port holds its concomitant constant residue. For simplicity, let the primary input X with a concomitant constant residue, a, be labeled as: $X\langle a\rangle$. Then the operation of the CCSA with CEAC is defined as follows:

$$\begin{aligned}&\left|X\langle a\rangle+Y\langle b\rangle+Z\langle c\rangle\right|_{2^n+1}\\&=\left|X+Y+Z+a+b+c\right|_{2^n+1}\\&=\left|C'+S+(a+b+c-1)\right|_{2^n+1}\\&=\left|C'\langle a+b+c-1\rangle+S\langle 0\rangle\right|_{2^n+1}\end{aligned}, \tag{6-1}$$

where C' and S are the CEAC and sum outputs of the CSA with CEAC defined by Equations (2-20) and (2-21), respectively. The constant residue is carried by the term inside the angle bracket of (6-1). Multiple residues due to the MOMA are reduced in a tree of the companion residue counters. We define the operation of the companion residue counter as follows:

$$\begin{cases}d=a+b+c-1\\e=0\end{cases}. \tag{6-2}$$

The companion residue counter functions like a binary three-to-two counter to reduce the constant residues in the same Wallace tree but with a unique reduction rule defined by (6-2), where the output constant residue e is forced to zero, and the constant residue d, typically non-zero is attached to the auxiliary port associated with the primary carry output. With this special reduction rule for the companion residue counter, we will show later that the companion residue counters are virtually non-critical as the final constant residue, d in the auxiliary carry output port will be reduced to a fixed value irrespective of the number of addends, k.

In summary, the ports of a CCSA with CEAC are divided into two parts: the primary ports hold the diminished-one numbers, and the auxiliary ports hold their concomitant constant residues. For simplicity, two ports are superposed. Equation (6-1) defines the CCSA with CEAC operation, where the operations for the primary ports are defined by Equations (2-20) and (2-21), and (6-2) defines the residue counting operation for the auxiliary ports. Figure 6.1(a) shows the symbolic representation of the companion residue counter. A CCSA with CEAC with a companion residue counter is represented by a simplified symbol with a bubble attached to the carry output as shown in Figure 6.1(b). Only the residue d to the auxiliary ports is labeled and its value is enclosed in angle bracket, <·>; therefore, the sum output has no numerical label because its constant residue e is always 0. Initially, the constant residue, carried by the input auxiliary port is '1' when its corresponding primary input port is fed from the diminished-one addends of the MOMA.

Figure 6.1 *(a) The companion residue counter (b) Composite CSA with CEAC (CCSA with CEAC)*

6.2.2 Algorithm for Diminished-One Modulo $2^n + 1$ MOMA

Now let us consider the operations of the CCSAs with CEAC with different input attributes in a diminished-one modulo $2^n + 1$ MOMA. The following theorem determines the constant residues to be attached to the corresponding auxiliary carry output ports of different types of CCSA with CEAC.

Theorem 6.1: If the primary inputs to the an n-bit CCSA with CEAC comprise p n-bit diminished-one addends and q diminished-one numbers $Y_1<a_1>$, .., Y_q $<a_q>$, where $p + q = 3$, Y_1, .., Y_q and a_1, .., a_q are the outputs from the primary and auxiliary ports,

respectively, of some other CCSAs with CEAC, then, the constant residue generated in its auxiliary carry output is $p-1+\sum_{i=1}^{q} a_i$

Proof: According to the definition of CCSA with CEAC in Equation (6-1), if the primary input is the diminished-one addend, the constant residue feeding its auxiliary port is 1. Therefore, the sum of the constant residues to the input auxiliary ports associated with the p diminished-one addends is p. On the other hand, an aggregate constant residue of $\sum_{i=1}^{q} a_i$ is fed into the auxiliary input ports from the auxiliary outputs of the CCSAs with CEAC associated with the q diminished-one numbers. According to the reduction rule of (6-2), Theorem 6.1 is true.

Theorem 6.2: The 7 CCSAs with CEAC with different input attributes can be unified by the CCSA with CEAC defined by Equations (6-1) and (6-2).

Proof: Case 1: Type 3I CCSA with CEAC. 3I CCSA with CEAC performs the modulo $2^n + 1$ addition of three diminished-one addends, A_1, A_2 and A_3, and generates the carry C', sum S and a constant residue. Then it performs:

$$\begin{aligned} &\left|A_1+1+A_2+1+A_3+1\right|_{2^n+1} \\ &=\left|A_1+A_2+A_3+3\right|_{2^n+1} \\ &=\left|C'+S+2\right|_{2^n+1} \\ &=\left|C'\langle 2\rangle+S\right|_{2^n+1} \\ &=\left|A_1\langle 1\rangle+A_2\langle 1\rangle+A_3\langle 1\rangle\right|_{2^n+1} \end{aligned} \tag{6-3}$$

According to the definition of CCSA with CEAC (6-1) and Theorem 6.1, Equation (6-3) implies that the 3I CCSA with CEAC for the modulo $2^n + 1$ MOMA can be directly mapped to a CCSA with CEAC for the diminished-one modulo $2^n + 1$ MOMA, where the three primary input ports are fed from the diminished-one addends of the MOMA. Therefore the inputs to the auxiliary ports are all 1s, and the auxiliary carry output port is 2. Figure 6.2(a) shows the symbolic diagram of the CCSA with CEAC.

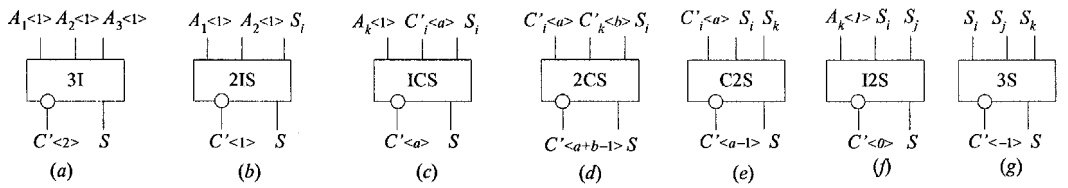


Figure 6.2 *Different types of composite CSA with CEAC*

Case 2: For type 2IS CCSA with CEAC, the two diminished-one inputs are A_1 and A_2, and the third input is the sum output of the CCSA from the preceding level. Therefore, type 2IS CCSA with CEAC performs the following operation:

$$\begin{aligned}
&\left|A_1+1+A_2+1+S_i\right|_{2^n+1}\\
&=\left|A_1+A_2+S_i+2\right|_{2^n+1}\\
&=\left|C'_j+S_j+1\right|_{2^n+1}\\
&=\left|C'_j\langle 1\rangle+S_j\right|_{2^n+1}\\
&=\left|A_1\langle 1\rangle+A_2\langle 1\rangle+S_i\right|_{2^n+1}
\end{aligned} \tag{6-4}$$

For the same reason, the type 2IS CCSA with CEAC for the modulo 2^n+1 MOMA is equivalent to a CCSA with CEAC for the diminished-one modulo 2^n+1 MOMA, where one of the primary input ports is from the sum output of another CCSA with CEAC, and the others are from the diminished-one addends. The auxiliary carry output is 1, which is shown in Figure 6.2(b).

Case 3: For type ICS CCSA with CEAC, two of its inputs are fed from the carry and sum outputs of the CCSA in the preceding level, and the third input is taken from the addend. If the constant residue of the carry output from the preceding CCSA is a, the ICS CCSA with CEAC performs the following operation:

$$\begin{aligned}
&\left|A_k + 1 + C_i^{'}\langle a\rangle + S_i\right|_{2^n+1} \\
&= \left|A_k + C_i^{'} + S_i + a + 1\right|_{2^n+1} \\
&= \left|C_j^{'} + S_j + a\right|_{2^n+1} \\
&= \left|C_j^{'}\langle a\rangle + S_j\right|_{2^n+1} \\
&= \left|A_k\langle 1\rangle + C_i^{'}\langle a\rangle + S_i\right|_{2^n+1}
\end{aligned} \quad . \tag{6-5}$$

Type ICS CCSA with CEAC defined by (6-5) is also equivalent to a CCSA with CEAC for the diminished-one modulo $2^n + 1$ MOMA, which is shown in Figure 6.2(c).

Case 4: For type 2CS CCSA with CEAC, two of its inputs are from the carry outputs of the CCSA in the preceding level, and the third input is taken from the sum output of another CCSA in the preceding level. Assume that the constant residues in the auxiliary ports of the two carry inputs are a and b. The operation of type 2CS CCSA is given by:

$$\begin{aligned}
&\left|C_i^{'}\langle a\rangle + S_i + C_k^{'}\langle b\rangle\right|_{2^n+1} \\
&= \left|C_i^{'} + a + S_i + C_k^{'} + b\right|_{2^n+1}, \\
&= \left|C_j^{'}\langle a + b - 1\rangle + S_j\right|_{2^n+1}
\end{aligned} \tag{6-6}$$

Equation (6-6) implies also a CCSA with CEAC with its two primary carry signals carrying the constant residues a and b, respectively, and one primary sum signal. A constant residue of $a + b - 1$ is generated in its auxiliary carry output port. Figure 6.2(d) shows the type 2CS CCSA with CEAC.

Case 5: The type C2S CCSA with CEAC has three input signals: a pair of carry and sum outputs from the preceding level, and a sum output from another CCSA of the preceding level. Assume that the carry input signal holds the constant residue a in its auxiliary port, then

$$\left|C_i^{'}\langle a\rangle + S_i + S_k\right|_{2^n+1} = \left|C_i^{'} + a + S_i + S_k\right|_{2^n+1} = \left|C_j^{'}\langle a - 1\rangle + S_j\right|_{2^n+1}. \tag{6-7}$$

Equation (6-7) is a CCSA with CEAC where one primary input port is fed from the carry output of the CCSA in the preceding level and has a constant residue a, while other two primary inputs are all taken from the sum outputs of some other CCSAs. Figure 6.2(e) shows the symbolic diagram of type C2S CCSA with CEAC, where the auxiliary carry output port holds the residue, $a-1$.

Case 6: Two inputs of the type I2S CCSA with CEAC are the sum outputs from two CCSAs in the preceding level, and the third input is fed from the input addend. The outputs of the type I2S CCSA with CEAC is obtained as follows:

$$\begin{aligned} &\left|A_k+1+S_i+S_j\right|_{2^n+1} \\ &=\left|C_l'\langle 0\rangle+S_l\right|_{2^n+1} \\ &=\left|A_k\langle 1\rangle+S_i+S_j\right|_{2^n+1} \end{aligned} \quad . \tag{6-8}$$

The I2S CCSA with CEAC defined by Equation (6-8) is again equivalent to a CCSA with CEAC for the diminished-one modulo 2^n+1 MOMA, where one of the primary input ports is from the addend, and two other primary input ports are from the sum outputs of some CCSAs. There is no residue generated from the auxiliary sum output port. Figure 6.1(f) shows the symbolic diagram.

Case 7: The three inputs of the type 3S CCSA with CEAC come from the sum outputs of three CCSAs in the preceding levels. Let the three inputs be S_i, S_j and S_k, then the output of the type 3S CCSA is given by:

$$\left|S_i+S_j+S_k\right|_{2^n+1}=\left|C_l'+S_l-1\right|_{2^n+1}=\left|C_l'\langle -1\rangle+S_l\right|_{2^n+1}. \tag{6-9}$$

The 3S type CCSA with CEAC defined by Equation (6-9) is also a CCSA with CEAC, where the constant residue generated in the auxiliary carry output is –1. Figure 6.1(g) shows the symbolic diagram of type 3S CCSA with CEAC. That ends the proof of Theorem 6.2.

According to Theorem 6.2, different CCSAs with CEAC can be unified to a homogeneous CCSA with CEAC, therefore, the CSA tree can be reduced. The proposed modulo 2^n+1 MOMA consists of a Wallace CCSA with CEAC tree,

followed by a 2-input diminished-one modulo $2^n + 1$ adder. The CCSAs employed in the tree can be any of the 7 CCSAs with CEAC of different input attributes. Therefore, a universal structure for the modulo 2^n +1 MOMA consisting of the unified CCSAs with CEAC can be obtained, wherein the diminished-one values are calculated by the CCSAs with CEAC, and the constant residues are accounted by the companion residue counter. Figure 6.3 shows the architecture of a (23, $2^n + 1$)-MOMA, where only the constant residues in the auxiliary carry output ports are labeled for simplicity. The constant residue generated in the auxiliary carry output of the last CCSA with CEAC is 2. In what follows, we will prove that this final constant residue is not a mere coincidence.

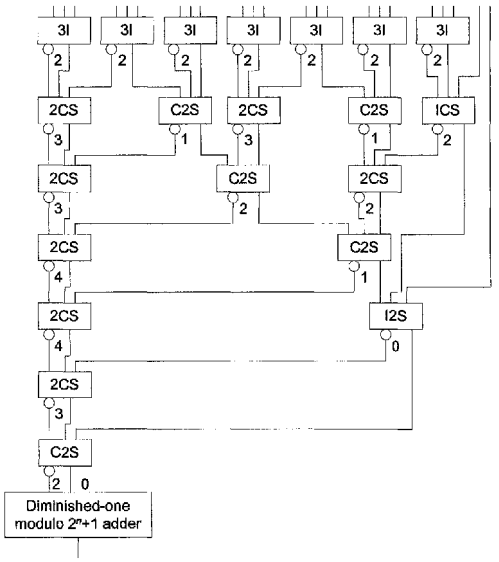


Figure 6.3 *Architecture for (23, 2^n+1)-MOMA using different types of CCSA with CEAC*

Theorem 6.3: The constant residue generated in the auxiliary carry output of the last CCSA with CEAC is always 2 irrespective of the number of addends, K ($K \geq 3$) of the modulo $2^n + 1$ CSA tree.

Proof: According to definition of CCSA with CEAC, if the primary input, X_i (i = 1, 2 or 3) to the CCSA with CEAC of Figure 6.1(b) comes from the primitive addends, the constant residue for the auxiliary input port of Figure 6.1(a) has to be 1. Otherwise, the constant residue is c_j, which is the auxiliary carry output of some CCSA with CEAC, or the constant residue is 0 if the primary input is fed from the auxiliary sum

output of a CCSA with CEAC from the preceding level. Therefore, the input constant residue of a companion residue counter, a_i can have any of the following values:

$$a_i = \begin{cases} 1, & \text{if the primary input is from the addends} \\ 0, & \text{if the primary input is from the sum output of a CCSA with CEAC} \\ c_j, & \text{if the primary input is from the carry output of a CCSA with CEAC} \end{cases} \quad (6\text{-}10)$$

After defining the inputs of the companion residue counter for the CCSA with CEAC, we can generate the structure of the companion residue counter network to accumulate the constant residues. As the CCSA with CEAC tree is a Wallace tree, the companion residue counter network is also a Wallace tree. According to the definition of (6-2), the companion residue counters are formed at the earliest opportunity from left to right, which is based on the same rule as the Wallace tree for the CSA with CEAC. The counters can be allocated using the following algorithm for different number of addends, K ($K \geq 3$).

Companion residue counter allocation algorithm:

1. Input integer K, $K \geq 3$.
2. Generate an array $A = \{a_i \mid a_i = 1, i = 1, 2, \ldots, K\}$.
3. $n = K$; $l = 0$;
4. WHILE $n > 2$ DO
   4.1 Allocate $\left\lfloor \frac{n}{3} \right\rfloor$ companion residue counters for the l-th level. For $j = 1, 2, \ldots, \left\lfloor \frac{n}{3} \right\rfloor$, remove groups of three constants a_{3j-2}, a_{3j-1}, a_{3j} from A to feed the three inputs of the counters from left to right.
   4.2 Calculate the carry and sum outputs of every counter using Equation (6-2). Form a new array of constants B by inserting into B the outputs of the counters from left to right.
   4.3 Append the remaining constants, if any, from A to the end of B.
   4.4 $A <= B$; l++;
   4.5 $n <=$ Numbers of terms in A.
5. END WHILE.

The input of the allocation algorithm is K, and the outputs are B and l. Each iteration in the while loop generates a level of counters. The terms in B correspond to the outputs of the counters in each level. Upon exiting from the while loop, B contains a pair of constants corresponding to the c and s outputs of the last counter, and l is the number of levels of the counter tree.

According to the reduction rule of Equation (6-2), the summation of the outputs of a companion residue counter is 1 less than the summation of the inputs; therefore, if there are K companion residue counters in one level of the counter tree, the summation of the outputs of this level is K less than the summation of the inputs of the same level. Let L be the number of levels in the tree, and the number of counters at each level is K_i for $i = 1, 2, \ldots, L$. Therefore, the summation of the outputs of the first level is $K - K_1$, the summation of the outputs of the second level is $K - K_1 - K_2$, ..., and the summation of the last counter's output is $K - K_1 - K_2 - \ldots - K_L = K - (K_1 + K_2 + \ldots + K_L)$. According to [Hwan79] [Parh00], $K - 2$ companion residue counters are required to reduce K operands to two. Thus, the summation of the outputs of the last counter of the tree is $K - (K - 2) = 2$. Based on the specific reduction rule of Equation (6-2), the output s is always 0. Hence, the output, c will be 2. Therefore, the constant residue in the auxiliary carry output of the last CCSA is always 2. That ends the proof of Theorem 6.3.

The trace of the companion residue counter allocation algorithm for $K = 23$ is illustrated in Figure 6.4. The outputs of the companion residue counters in level 1 is $B = \{2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 1, 1\}$, in level 2 is $B = \{3, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1\}$, ..., and in level 7, $B = \{2, 0\}$.

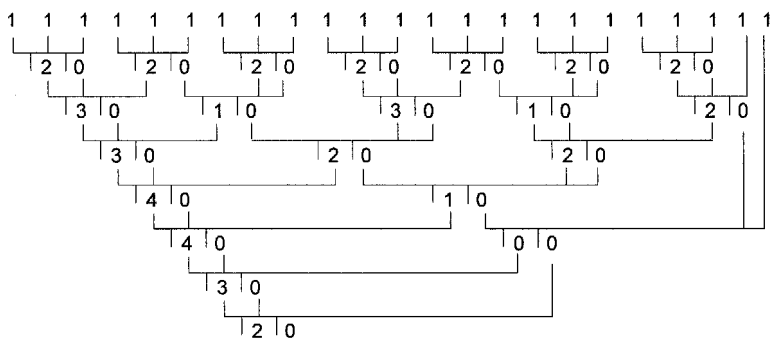


Figure 6.4 *Process of counter allocation algorithm with K = 23*

According to Theorem 6.3, the last constant residue is always 2 irrespective of the number of addends of the diminished-one modulo $2^n + 1$ CCSA tree, which means that there is no need to implement the companion residue counter tree. Furthermore, the last stage of the MOMA is a two-operand diminished-one addition. According to Equation (2-18) of Chapter 2, a constant '1' is added in the diminished-one addition for the diminished-one modulo $2^n + 1$ adder. In converting the diminished-one result back to its binary representation, another '1' is added according to Equation (2-17). Therefore, the final constant residue of '2' held in the auxiliary port of the last CCSA can be used to offset the constant required in the diminished-one operation of the final CPA. Thus, the companion residue counter network can be completely eliminated, and the CCSA with CEAC within the modulo $2^n + 1$ MOMA is replaced by the simple CSA with CEAC defined by Equations (2-20) and (2-21). Finally, the universal modulo $2^n + 1$ MOMA comprises a Wallace CSA with CEAC tree and a 2-input diminished-one CPA, which is shown in Figure 6.5, and it performs the diminished-one MOMA of Equation (2-19). The architecture of a CSA with CEAC tree is the same as that of CSA with EAC except that the EAC is replaced by CEAC. The difference between the modulo $2^n - 1$ and $2^n + 1$ MOMAs is that the inputs and output of the modulo $2^n - 1$ MOMA are n-bit binary numbers, while the inputs and output of the latter are n-bit diminished-one numbers. Another difference is the final modulo CPA. The final modular CPA for the $2^n - 1$ MOMA is a 2-input modulo $2^n - 1$ adder, while that for the $2^n + 1$ MOMA is a 2-input diminished-one modulo $2^n + 1$ adder.

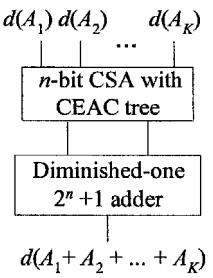


Figure 6.5 *General architecture for diminished-one (K, $2^n + 1$)-MOMA*

Example 6.1: Let $n = 8$ and $K = 7$. One $(7, 2^8 + 1)$-MOMA is shown in Figure 6.6. Given that $A_1 = 126$, $A_2 = 36$, $A_3 = 20$, $A_4 = 200$, $A_5 = 206$, $A_6 = 122$ and $A_7 = 89$, then

their diminished-one values are: $d(A_1) = 125$, $d(A_2) = 35$, $d(A_3) = 19$, $d(A_4) = 199$, $d(A_5) = 205$, $d(A_6) = 121$, and $d(A_7) = 88$. The carry and sum outputs of the CSAs with CEAC are calculated as follows: C'_1 = 67H, S_1 = 4DH, C'_2 = 9AH, S_2 = 73H, C'_3 = 9FH, S_3 = B0H, C'_4 = 66H, S_4 = 5CH, C'_5 = B9H and S_5 = 62H. The output of the diminished-one modulo $2^8 + 1$ adder is given by $|\text{B9H} + 62\text{H} + 1|_{257} = 27$. As $|A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7|_{257} = 28$, the correct result of the diminished-one summation of 27 is obtained.

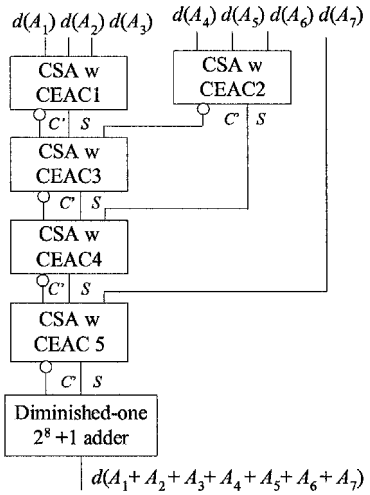


Figure 6.6 *Architecture for diminished-one (7, 2^8 + 1)-MOMA*

6.2.3 Performance Evaluation and Comparison for Modulo 2^n + 1 MOMA

Before the evaluation, we first analyze the gate level implementation of an n-bit CSA with CEAC because it is the basic element of the Wallace CSA with CEAC tree for the proposed MOMA. A CSA with CEAC consists of n FAs, and one inverter for the leftmost FA. An FA with an inverter can be treated as a modified FA, which has the truth table shown in Table 6.1. In Table 6.1, the carry output of the modified FA is complemented as opposed to the FA. A gate-level implementation of the modified FA is shown in Figure 6.7, where the area and delay are the same as those of the FA in terms of the unit–gate model [Zimm99] [Ver02] (see Section 2.1.4.1 of Chapter 2). Therefore, the area of an n-bit CSA with CEAC is $7n$, and the delay is 4.

Table 6.1 *Truth table for the modified FA of CSA with CEAC*

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	0

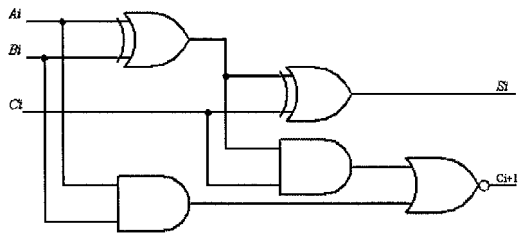


Figure 6.7 *Gate-level implementation of the modified FA*

The maximum number of operands K that can be processed with an L-level CSA with CEAC tree can be obtained from a recursive formula from Equation (2-12). Therefore, the delay of the K-input CSA with CEAC tree is $4L$. As the K-input CSA tree consists of $K-2$ CSAs [Parh00], the area is $7n(K-2)$.

Figure 6.8 shows the dot notation of a $(7, 2^8+1)$-MOMA. The two dots connected by a short slash from the upper right to the lower left direction denotes an FA, and the two dots connected with a long slash from the upper left to the lower right direction denotes a modified FA. The 8 dots in the last row that are enclosed in a box denote a 2-input diminished-one modulo adder. It is evident that the proposed diminished-one MOMA has a very regular structure, which is amenable to VLSI implementation.

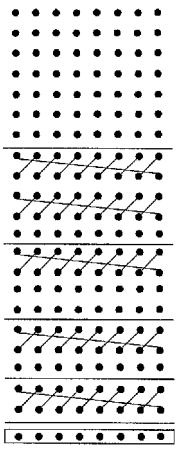


Figure 6.8 *Dot annotation of (7, 2^8 + 1)-MOMA*

The most efficient implementation of the 2-input diminished-one modulo $2^n + 1$ adder has been reported in [Ver02], which can be as fast as the modulo $2^n - 1$ adder. Let the area cost and delay of the modulo $2^n + 1$ adder be $C_{M(2^n+1)}$ and $D_{M(2^n+1)}$, respectively. Then, the total area cost and delay of the proposed universal architecture for modulo $2^n + 1$ MOMA are $7n(K-2)+C_{M(2^n+1)}$ and $4L+D_{M(2^n+1)}$, respectively. The architecture of the proposed diminished-one $2^n + 1$ MOMA shown in Figure 6.5 is similar to that for the modulo $2^n - 1$ MOMA. The delay of the CSA with CEAC tree for modulo $2^n + 1$ MOMA is the same as that of the CSA with EAC tree for modulo $2^n - 1$ MOMA. If the modulo $2^n + 1$ adder in [Ver02] is adopted in our architecture, the proposed MOMA can achieve the same delay as that of the modulo $2^n - 1$ MOMA.

To the best of our knowledge, there is no universal architecture for the diminished-one MOMA reported in the literature. What we will do here is to compare our area-time performance against the architecture for the binary modulo $2^n + 1$ MOMAs. The modulo $2^n + 1$ MOMA proposed in [Ska89] performs the modulo addition sequentially, taking K+1 cycles for K addends. This architecture is not the diminished-one system, and it can be unrolled into a binary CSA tree followed by two binary adders and one modulo $2^n + 1$ adder, as shown in Figure 6.9. The number of levels of the resulting CSA is the same as that of the proposed CSA with CEAC tree for the same K input addends. Each CSA in Figure 6.9 consists of $n + 1$ FAs, while each CSA with CEAC contains only n FAs including the modified FA. The delay of CSA tree in Figure 6.9 is the same as that of the proposed CSA with CEAC tree.

There are two (n+1)-bit binary adders and one counter to carry out the addition sequentially after the CSA tree in Figure 6.9. The last stage is a 2-input carry propagate module $2^n + 1$ adder. Therefore, the area and delay of the modulo $2^n + 1$ MOMA of [Ska89] are $7(n+1)(K-2)+2C_{B(2^{n+1})}+C_{cnt}+C_{M(2^n+1)}$ and $4L+2D_{B(2^{n+1})}+D_{cnt}+D_{M(2^n+1)}$, respectively, where $C_{B(2^{n+1})}$ and $D_{B(2^{n+1})}$ are the area and delay of the (n+1)-bit binary adder, C_{cnt} and D_{cnt} are the area and delay of the counter, and L is the number of levels of the CSA tree.

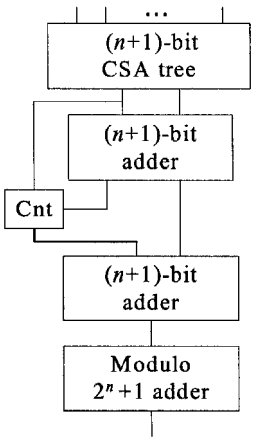


Figure 6.9 *Modulo $2^n + 1$ MOMA proposed in [Ska89]*

The modulo $2^n + 1$ MOMA with normal representation proposed in [Pie94a] is shown in Figure 6.10, which consists of one n-bit CSA with CEAC tree, one n-bit CPA, one (n+1)-bit CPA and one (n+1)-bit MUX according to Table VI of [Pie94a]. The CSAs are also with CEAC, and the number of inputs of the CSA tree is also K, but the least bit position has an input of $2K$ bits due to the normal representation. Thus, the number of levels of the CSA with CEAC tree is bigger than that of our proposed diminished-one MOMA for the same number of inputs K. It is difficult to calculate the delay of the CSA with CEAC tree, because the relation between the number of levels, l and the input K, as defined by Equation (2-12) is not applicable. More area and longer delay will be incurred. Furthermore, the CSA with CEAC tree becomes irregular due to the fact that the input bit number of the LSB positions doubles compared to the other bit positions. Figure 6.11 shows the implementation of [Pie94a] for n = 8, and K = 7. Comparing to our proposed MOMA shown in Figure 6.8, it has longer delay, and its CSA with CEAC tree uses 47 FAs, whereas ours use only 40 FAs. Moreover, its

irregularity is obvious, which will lead to poor layout with low silicon area utilization efficiency.

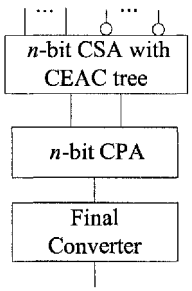


Figure 6.10 *Modulo $2^n + 1$ MOMA proposed in [Pie94a]*

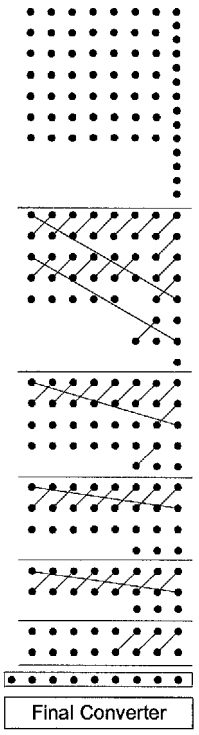


Figure 6.11 *Implementation of (7, 2^8+1)-MOMA of [Pie94a]*

In the unit gate model, the area and delay of a 1-bit MUX are 3 and 2, respectively. Thus, the area and delay of a $(n+1)$-bit MUX are $3(n+1)$ and 2, respectively. According to Table VI of [Pie94a], the area cost of the $2^n + 1$ MOMA in [Pie94a] is $C_{tree}+3(n+1)+C_{B(2^n)}+C_{B(2^{n+1})}$, and the delay is $4L+2+D_{B(2^n)}+D_{B(2^{n+1})}$, where C_{tree}, $C_{B(2^n)}$ and $D_{B(2^n)}$ are respectively, the area cost of the CSA with CEAC tree, area cost of the n-bit binary adder and the delay of an n-bit binary adder. According to the

previous analysis, C_{tree} is bigger than the hardware cost of the CSA with CEAC in Figure 6.5.

Table 6.2 shows the area and delay comparisons of the three modulo $2^n + 1$ MOMAs. It is obvious that the proposed MOMA is more area efficient than those of [Ska89]. From [Ver02], we know that the area and delay of a 2-input diminished-one modulo $2^n + 1$ adder are a little more than those of the n-bit binary adder, but apparently less than double the area and delay of an n-bit binary adder. Therefore, our proposed diminished-one MOMA is more efficient than the binary MOMAs proposed in [Ska89] and [Pie94a]. In Table 6.2, the number of levels of the CSA with CEAC tree for [Pie94a] (L_1) is bigger than that of our proposed MOMA and [Ska89] (L), therefore, the proposed MOMA is also the fastest.

Table 6.2 *Area and delay comparisons of three (K, $2^n + 1$)-MOMAs*

MOMA	Area	Delay
Proposed diminished-one	$7n(K-2)+C_{M(2^n+1)}$	$4L+D_{M(2^n+1)}$
[Ska89]	$7(n+1)(K-2)+2C_{B(2^{n+1})}+C_{cnt}+C_{M(2^n+1)}$	$4L+2D_{B(2^{n+1})}+D_{cnt}+D_{M(2^n+1)}$
[Pie94a]	$C_{tree}+3(n+1)+C_{B(2^n)}+C_{B(2^{n+1})}$	$4L_1+2+D_{B(2^n)}+D_{B(2^{n+1})}$

6.3 Conclusions

In this chapter, we have presented the general architecture for designing the diminished-one modulo $2^n + 1$ MOMAs. The proposed architecture consists of one n-bit CSA with CEAC tree and one 2-input diminished-one modulo $2^n + 1$ adder. The area and delay of the CSA with CEAC tree is the same as that of the CSA with EAC tree of the modulo $2^n - 1$ MOMA, and the architecture of the CSA with CEAC tree is more regular than that of a binary Wallace tree. By using the efficient diminished-one modulo $2^n + 1$ adders in the literature, the proposed modulo $2^n + 1$ MOMA can be as fast as the modulo $2^n - 1$ MOMA. Comparing to the existing modulo $2^n + 1$ MOMAs [Ska89] [Pie94a], the proposed structure has lower hardware costs, and faster speed from the area-time complexity analysis of Table 6.2. Another important contribution of this chapter is the modeling of the generic modulo $2^n + 1$ MOMA using CCSA with CEAC and the formal proof of the redundancy of the companion residue counter

network. The generalization of the diminished-one modulo $2^n + 1$ MOMA and its compatible structure to that of the modulo $2^n - 1$ MOMA can be readily exploited to build a dual function modulo $2^n \pm 1$ MOMAs with minimal additional circuits of multiplexers and selector logics [Men05]. As modulo $2^n + 1$ and modulo $2^n - 1$ arithmetic are the two most frequently used modulo operations, such reconfigurability of the proposed structure has significantly enhanced the versatility of its use in RNS based applications.

Chapter 7

VLSI Performances of Triple Moduli Based RNS's

7.1 Introduction

It is conjectured that there exist intricate trade-offs among RNS's of different special moduli sets and to a large extent, the system architecture and performance are dictated by the dynamic ranges and the types and frequency of modular operations required by the application of interest [Con03] [Con04]. It is not always necessary that the higher cardinality RNS will outperform the triple moduli set in any application or every aspect of VLSI metrics due to the tug of war between the increased parallelism and higher complexity of the forward and reverse converters. To the best of our knowledge, no attempt has been made to quantify the relative performances of different RNS's to determine the critical dynamic range above which the increased parallelism will justify for the conversion overheads. We believe that there are at least two major reasons that tend to discourage such investigations. First, there are myriads of moduli sets, even for the special moduli sets alone. For each moduli set, there are also many different structures and implementations for the forward and reverse converters. Second, residue number systems are architectural level methods beneficial to specific high-speed applications. The operations involved in the residue arithmetic unit are highly dependent on the applications. While it is impossible and also impractical to accurately evaluate the VLSI performances in all aspects of the implementation of residue number systems, we can limit our study of VLSI performances of those special triple moduli based RNS's presented in earlier chapters with a proper design of experiment for some generic

applications. The aim is to provide some useful cues to the designer on the trade-offs of using the special moduli sets of 2^n and $2^n \pm 1$ types for his application.

VLSI cost metrics such as delay, area and power consumption of the forward and reverse converters against the dynamic ranges for five different special moduli set RNS's will be independently studied in this chapter. A simulation environment with a programmable number of multiply and accumulate operations in the residue arithmetic unit is set up to evaluate each complete RNS. The dynamic ranges and input word length have been chosen to possess sufficient generality to represent several important RNS applications. The general structure of the modulo $2^n + 1$ MOMA presented in the previous chapter has been employed for the implementations of the forward converters and modular multipliers in the residue channels involving moduli of $2^n + 1$ type. Carry look-ahead adder is consistently employed for the final-stage two-operand modulo adders and the modular multipliers because they have reasonably good performance and are most widely used. As the primary goal of using RNS is to increase the speed or throughput rate of computations, timing optimization has higher precedence when mapping the RTL designs to gate-level circuits. In comparing the performances, the total latencies of the designs are discussed, particularly for the full combinational implementation of the forward and reverse converters. For the complete system, however, the throughput rate is more important. This is often constrained by the latency of the longest residue arithmetic operation of each RNS. The throughput rates for different RNS's are also related to the number of pipelining stages. Therefore, it is logical to use more pipelined stages for the converters of the higher cardinality RNS's.

To evaluate the VLSI performances of different RNS's and their constituent computational units, the designs are modeled with Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) at Register Transfer Level (RTL). Mentor Graphics' ModelSim is used to verify the functionality of the designs. Upon successful functional verification, the designs are synthesized and optimized by Synopsys Design Compiler (DC). DC uses design rules and optimization constraints to control the synthesis of the design, and to map the technology independent design to a technology

specific gate-level net list. The technology files contain information about the characteristics and functions of each cell provided in a semiconductor vendor's library. Cell characteristics include information such as cell names, pin names, area, delay arcs, and pin loading. The technology library also defines the conditions that must be met for a functional design, which are called design rule constraints. In addition to cell information and design rule constraints, technology libraries specify the operation conditions and wire load specific to the technology. The technology libraries and the EDA tools used in this thesis are stated in the Appendix A.1. To allow a comprehensive analysis, both timing and area driven optimizations were performed. For timing optimization, each design is recursively optimized for speed until the EDA tool is unable to provide a faster design. For area driven optimization, the design is optimized at the best effort of the EDA tool to produce the minimum area possible. The total area cost, measured in terms of the number of equivalent 2-input NAND gates, includes the cell area and the net interconnection area. The average power consumptions are simulated by Synopsys Power Compiler with randomly generated input vectors. For full combinational circuits, a virtual clock is used to input the data for power simulation. The average power consists of dynamic power and static power. The simulation results of delay, area and power are extracted from various reports of DC. Appendix A.2 and A.3 show examples of VHDL codes, scripts and reports.

7.2 Forward Converters for Triple Moduli based RNS's

7.2.1 Forward Converter Architectures

The forward converters for the triple moduli based RNS are studied in this section. The special moduli sets being investigated are: $S_1 = \{2^n - 1, 2^n, 2^n + 1\}$, $S_2 = \{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$, $S_3 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, $S_4 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $S_5 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$. The residue of an integer X modulo 2^n is the n LSBs of X and hence no hardware cost is needed for its forward conversion. As for the moduli $2^n + 1$ and $2^n - 1$, only modulo $2^n - 1$ CSA with EAC or CEAC and modulo $2^n \pm 1$ adder are required for their forward conversions due to the special number properties of modulo $2^n \pm 1$ operations. In what follows, unless otherwise specified, the general

structure of the diminished-one modulo $2^n + 1$ MOMA presented in the previous chapter is assumed in the residue channels for moduli of the $2^n + 1$ type.

For the RNS, S_1, consider the $3n$-bit integer $X = (X_{3n-1}\, X_{3n-2}\, \ldots\, X_1\, X_0)_2$. According to [Pou94a], we have

$$\begin{aligned} x_1 &= |X|_{2^n-1} = |B_1 + B_2 + B_3|_{2^n-1} \\ x_2 &= |X|_{2^n} = B_1 \\ x_3 &= |X|_{2^n+1} = |B_1 - B_2 + B_3|_{2^n+1} \end{aligned} \tag{7-1}$$

where $B_1 = (X_{n-1}\, \ldots\, X_1 X_0)_2$, $B_2 = (X_{2n-1}\, \ldots\, X_{n+1} X_n)_2$, and $B_3 = (X_{3n-1}\, \ldots\, X_{2n+1} X_{2n})_2$. The forward converter of S_1 is shown in Figure 7.1.

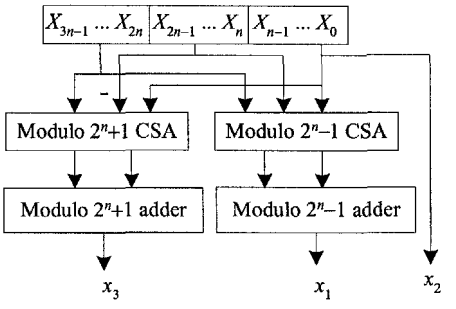


Figure 7.1 *Forward converter for* S_1

For the RNS, S_2, consider the $5n$-bit integer $X = (X_{5n-1} X_{3n-2}\, \ldots\, X_1\, X_0)_2$. Let

$$\begin{aligned} B_0 &= (X_{n-1}\; X_{n-2} \cdots X_0)_2 \\ B_1 &= (X_{2n-1}\; X_{2n-2} \cdots X_n)_2 \\ B_2 &= (X_{3n-1}\; X_{3n-2} \cdots X_{2n})_2 \\ B_3 &= (X_{4n-1}\; X_{4n-2} \cdots X_{3n})_2 \\ B_4 &= (X_{5n-1}\; X_{5n-2} \cdots X_{4n})_2 \end{aligned} \tag{7-2}$$

The forward converters for S_2 can then be expressed by the following equations.

$$x_1 = |X|_{2^n-1} = \left|\sum_{i=0}^{4} B_i\right|_{2^n-1} \tag{7-3}$$

$$x_2 = B_0 \tag{7-4}$$

$$x_3 = |X|_{2^n+1} = |B_0 + B_2 + \overline{B}_1 + \overline{B}_3 + \overline{B}_4 + 4|_{2^n+1} \tag{7-5}$$

$$x_4 = |X|_{2^{2n}+1} = |B_1 B_0 + \overline{B}_3 \overline{B}_2 + B_4 + 2|_{2^{2n}+1} \tag{7-6}$$

In Equation (7-5), $\overline{B}_1$ and $\overline{B}_3$ are 1's complement of B_1 and B_3, respectively. In Equation (7-6), B_1B_0 denotes the concatenation of strings B_1 and B_0. Similarly, $\overline{B}_3\overline{B}_2$ denotes the concatenation of strings $\overline{B}_3$ and $\overline{B}_2$. Figure 7.2 shows the forward converter of S_2. According to Equation (2-20), a constant '1' is invoked by the diminished-one addition for each CSA with CEAC operation. As there are four CSAs with CEAC in the CSA tree, the numeral '4' in Equation (7-5) will be absorbed by the four CSAs with CEAC. Thus, the sixth input to the CSA with CEAC for x_3 is zero. For the same reason, the fourth input to the CSA with CEAC for x_4 is also 0.

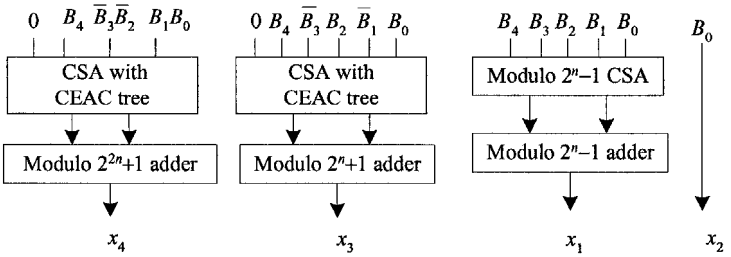


Figure 7.2 *Forward converter for* S_2

For the RNS, S_3, consider the $(4n-1)$-bit integer $X = (X_{4n-2}\, X_{4n-3} \ldots X_1\, X_0)_2$. Let

$$\begin{aligned} B_{10} &= \left(X_{n-1}\, X_{n-2} \ldots X_0\right)_2 \\ B_{11} &= \left(X_{2n-1}\, X_{2n-2} \ldots X_n\right)_2 \\ B_{12} &= \left(X_{3n-1}\, X_{3n-2} \ldots X_{2n}\right)_2 \\ B_{13} &= \left(0 X_{4n-2}\, X_{4n-3} \ldots X_{3n}\right)_2 \end{aligned}, \tag{7-7}$$

$$\begin{aligned} B_{30} &= \left(X_{n-1} \ldots X_0\right)_2 \\ \overline{B}_{31} &= \left(\overline{X}_{2n-1} \ldots \overline{X}_n\right)_2 \\ B_{32} &= \left(X_{3n-1} \ldots X_{2n}\right)_2 \\ \overline{B}_{33} &= \left(1\overline{X}_{4n-2} \ldots \overline{X}_{3n}\right)_2 \end{aligned} \tag{7-8}$$

and

$$\begin{aligned} B_{40} &= \left(X_{n-2} \ldots X_0\right)_2 \\ B_{41} &= \left(X_{2n-3} \ldots X_{n-1}\right)_2 \\ B_{42} &= \left(X_{3n-4} \ldots X_{2n-2}\right)_2 \\ B_{43} &= \left(X_{4n-5} \ldots X_{3n-3}\right)_2 \\ B_{44} &= \left((0)^{(n-3)} X_{4n-2} X_{4n-3} X_{4n-4}\right)_2 \end{aligned} \tag{7-9}$$

Then, the residues for S_4 can be generated as follows:

$$x_1 = \left| \sum_{i=0}^{3} B_{1i} \right|_{2^n-1} \tag{7-10}$$

$$x_2 = B_{10} \tag{7-11}$$

$$x_3 = \left| B_{30} + B_{32} + \overline{B}_{31} + \overline{B}_{33} + 4 \right|_{2^n+1} \tag{7-12}$$

$$x_4 = \left| \sum_{i=0}^{4} B_{4i} \right|_{2^{n-1}-1} \tag{7-13}$$

Figure 7.3 shows the forward converter architecture for S_3.

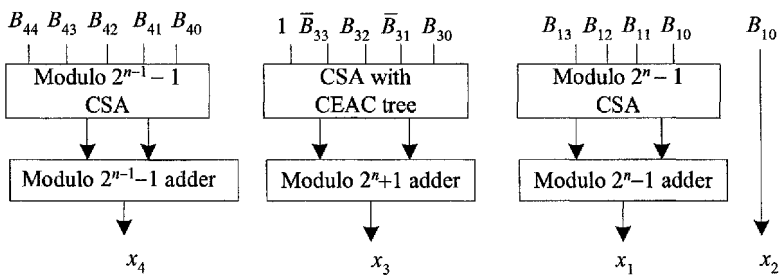


Figure 7.3 *Forward converter for S_3*

For the RNS, S_4, consider the $(4n + 1)$-bit integer, $X = (X_{4n} X_{4n-1} \ldots X_1 X_0)_2$. Let

$$\begin{aligned} B_{10} &= (X_{n-1} \ldots X_0)_2 \\ B_{11} &= (X_{2n-1} \ldots X_n)_2 \\ B_{12} &= (X_{3n-1} \ldots X_{2n})_2 , \\ B_{13} &= (X_{4n-1} \ldots X_{3n})_2 \\ B_{14} &= (0 \ldots 0 X_{4n})_2 \end{aligned} \tag{7-14}$$

$$\begin{aligned} B_{30} &= (X_{n-1} \ldots X_0)_2 \\ \overline{B}_{31} &= (\overline{X}_{2n-1} \ldots \overline{X}_n)_2 \\ B_{32} &= (X_{3n-1} \ldots X_{2n})_2 \\ \overline{B}_{33} &= (\overline{X}_{4n-1} \ldots \overline{X}_{3n})_2 \\ B_{34} &= (0 \ldots 0 X_{4n})_2 \end{aligned} \tag{7-15}$$

and

$$\begin{aligned} B_{40} &= (X_n \ldots X_0)_2 \\ B_{41} &= (X_{2n+1} \ldots X_{n+1})_2 \\ B_{42} &= (X_{3n+2} \ldots X_{2n+2})_2 \\ B_{43} &= (0 \ldots 0 X_{4n} \ldots X_{3n+3})_2 \end{aligned} \tag{7-16}$$

The residues for S_4 can be generated as follows:

$$x_1 = \left| \sum_{i=0}^{4} B_{1i} \right|_{2^n - 1} \tag{7-17}$$

$$x_2 = B_{10} \tag{7-18}$$

$$x_3 = \left| B_{30} + B_{32} + B_{34} + \overline{B}_{31} + \overline{B}_{33} + 4 \right|_{2^n + 1} \tag{7-19}$$

$$x_4 = \left| \sum_{i=0}^{4} B_{4i} \right|_{2^{n+1} - 1} \tag{7-20}$$

Figure 7.4 shows the forward converter architecture for S_4.

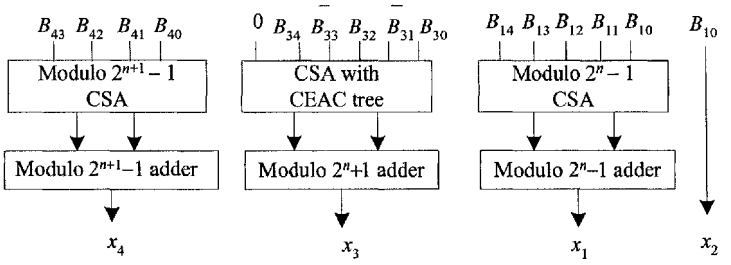


Figure 7.4 *Forward converter for* S_4

For the RNS, S_5, consider the $5n$-bit integer, $X = (X_{5n-1}\, X_{5n-2} \ldots X_1\, X_0)_2$. Let

$$\begin{aligned} B_{10} &= (X_{n-1} \ldots X_0)_2 \\ B_{11} &= (X_{2n-1} \ldots X_n)_2 \\ B_{12} &= (X_{3n-1} \ldots X_{2n})_2 \\ B_{13} &= (X_{4n-1} \ldots X_{3n})_2 \\ B_{14} &= (X_{5n-1} \ldots X_{4n})_2 \end{aligned} \quad , \tag{7-21}$$

$$\begin{aligned} B_{30} &= (X_{n-1} \ldots X_0)_2 \\ \overline{B}_{31} &= (\overline{X}_{2n-1} \ldots \overline{X}_n)_2 \\ B_{32} &= (X_{3n-1} \ldots X_{2n})_2 \\ \overline{B}_{33} &= (\overline{X}_{4n-1} \ldots \overline{X}_{3n})_2 \\ B_{34} &= (X_{5n-1} \ldots X_{4n})_2 \end{aligned} \quad , \tag{7-22}$$

$$\begin{aligned} B_{40} &= (X_n \cdots X_0)_2 \\ B_{41} &= (X_{2n+1} \cdots X_{n+1})_2 \\ B_{42} &= (X_{3n+2} \cdots X_{2n+2})_2 \\ B_{43} &= (X_{4n+3} \cdots X_{3n+3})_2 \\ B_{44} &= ((0)^{(5)} X_{5n-1} \cdots X_{4n+4})_2 \end{aligned} \quad , \tag{7-23}$$

and

$$
\begin{aligned}
B_{50} &= \left(X_{n-2} \cdots X_0\right)_2 \\
B_{51} &= \left(X_{2n-4} \cdots X_{n-3}\right)_2 \\
&\cdots \\
B_{5(k-2)} &= \left(X_{(k-1)\cdot(n-1)-1} \cdots X_{k^2-n-3}\right)_2 \\
B_{5(k-1)} &= \left((0)^{((k-1)\cdot(n-1)-4n-1)} X_{5n-1} \cdots X_{(k-1)\cdot(n-1)}\right)_2
\end{aligned}
\quad , \tag{7-24}
$$

where the integer k in Equation (7.24) is given by

$$k = \lceil 5n/(n-1) \rceil \tag{7-25}$$

Then the residues for S_5 can be generated as follows:

$$x_1 = \left| \sum_{i=0}^{4} B_{1i} \right|_{2^n-1} \tag{7-26}$$

$$x_2 = B_{10} \tag{7-27}$$

$$x_3 = \left| B_{30} + B_{32} + B_{34} + \overline{B}_{31} + \overline{B}_{33} + 4 \right|_{2^n+1} \tag{7-28}$$

$$x_4 = \left| \sum_{i=0}^{4} B_{4i} \right|_{2^{n+1}-1} \tag{7-29}$$

$$x_5 = \left| \sum_{i=0}^{k-1} B_{5i} \right|_{2^{n-1}-1} \tag{7-30}$$

The forward converter's architecture for S_5 is shown in Figure 7.5.

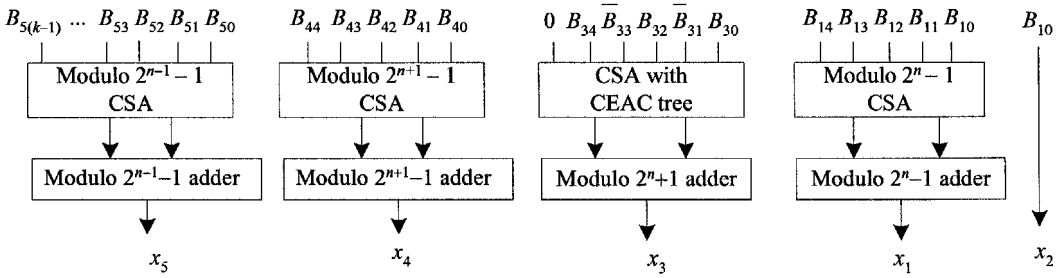


Figure 7.5 *Forward converter for S_5*

7.2.2 Performance Evaluation and Analysis of Forward Converters

To evaluate the VLSI performances of different forward converters, the architectures are modeled with VHDL and synthesized. For the final two-operand modular additions required by the forward converters, the modulo $2^n - 1$ adder proposed in [Efst94] and the modulo $2^n + 1$ adder proposed in [Hia02] are used. The synthesis results for area, delay and power consumption of each forward converter for S_1 to S_5 with different dynamic ranges are tabulated in Tables 7.1 to 7.5. The total area costs, measured in terms of the number of equivalent 2-input NAND gates, include the cell area and the net interconnection area. The average power consumptions are simulated by Synopsys Power Compiler with randomly generated input vectors. Figures 7.6 to 7.8 show the comparison of delay, total area and average power, respectively of all the five converters obtained under timing optimization. From Figure 7.6, the forward converters for S_1 and S_2 have the shortest delay when the dynamic range is less than 60 bits, and the forward converters for S_5 have the shortest delay when the dynamic range is larger than 60 bits. The delay differences of the forward converters for S_3 and S_4 are very small. From Figures 7.7 and 7.8, the area and power consumption of the forward converters for S_2 are higher than the others. Figure 7.9 to 7.11 show the comparisons of area, delay and power, respectively of the forward converters optimized for area. It is expected that the forward converter for higher cardinality RNS is more costly to implement and consume higher power as it has more channels. For example, when the dynamic range is 50-bit, the hardware costs are 1.2K and 2.2K gates for the forward converters for S_1 and S_5, respectively when optimized for the minimum area. It is interesting to note that the speed of the forward converter of S_5 is the fastest when all the designs are optimized for area.

Table 7.1 *VLSI metrics of forward converters for* $S_1 = \{2^n - 1, 2^n, 2^n + 1\}$

n	4	8	12	16	20	24	28	32
Dynamic Range	12	24	36	48	60	72	84	96
Results obtained by Speed Optimization								
Cell area	429	894.7	1368	1922	2543	2819	3413	3909
Routing area	121	261.4	387	579	689	857	1012	1158
Total area	550	1156	1755	2571	3232	3676	4425	5068
Delay (ns)	2.54	2.91	3.02	3.14	3.47	3.50	3.64	3.91
Power (mW)	4.89	10.37	16.76	24.68	31.86	33.91	41.80	48.73
Results obtained by Area Optimization								
Cell area	216.9	428.2	653.2	855.6	1089	1304	1512	1724
Routing area	81.9	162.2	256.1	348.1	446	548	643	742
Total area	298.8	590.5	909.3	1203.7	1535	1852	2156	2466
Delay (ns)	3.82	5.88	7.81	9.77	11.71	13.50	14.87	16.36
Power (mW)	2.22	4.44	6.86	9.01	11.42	13.86	16.07	18.18

Table 7.2 *VLSI metrics of forward converters for* $S_2 = \{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$

n	4	8	12	16	20
Dynamic Range	20	40	60	80	100
Results obtained by Speed Optimization					
Cell area	1066	2295	3466	4651	5725
Routing area	324	699	1039	1395	1743
Total area	1390	2995	4506	6047	7469
Delay (ns)	2.80	3.13	3.43	3.98	4.38
Power (mW)	11.76	26.61	40.31	56.01	67.54
Results obtained by Area Optimization					
Cell area	582	1168	1758	2319	2829
Routing area	220	462	715	965	1197
Total area	803	1630	2474	3285	4026
Delay (ns)	5.75	9.77	13.55	16.22	21.52
Power (mW)	5.79	11.80	17.95	23.72	28.69

Table 7.3 *VLSI metrics for forward converters for* $S_3 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$

n	4	8	12	16	20	24
Dynamic Range	15	31	47	63	79	95
Results obtained by Speed Optimization						
Cell area	677.7	1353.8	2096.6	2957	3600	4474
Routing area	187.1	387.1	621.5	878	1108	1339
Total area	864.8	1740.9	2718.1	3834	4709	5814
Delay (ns)	3.00	3.59	3.57	4.03	3.93	4.46
Power (mW)	8.70	16.17	25.56	36.55	43.08	57.15
Results obtained by Area Optimization						
Cell area	378.3	796.3	1227	1636	2096	2497
Routing area	130.8	277.7	444.8	609	796	981
Total area	509.2	1074.0	1672.8	2245	2892	3478
Delay (ns)	4.32	6.24	8.30	10.26	12.21	14.13
Power (mW)	3.92	8.28	12.82	17.08	21.74	26.14

Table 7.4 *VLSI metrics for forward converters for* $S_4 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$

n	4	8	12	16	20	24
Dynamic Range	17	33	49	65	81	97
Results obtained by Speed Optimization						
Cell area	763.2	1458.7	2203	3089	3732	1370
Routing area	216.2	411.9	638	908	1146	4561
Total area	979.4	1870.6	2842	3997	4878	5932
Delay (ns)	3.00	3.59	3.72	4.02	3.92	4.48
Power (mW)	8.70	17.81	26.88	38.30	44.76	57.93
Results obtained by Area Optimization						
Cell area	459.1	861.9	1295	1706	2157	2571
Routing area	160.5	304.9	468	641	832	1016
Total area	619.6	1166.8	1763	2347	2990	3588
Delay (ns)	4.32	6.23	8.30	10.26	12.20	14.05
Power (mW)	4.69	8.86	13.38	17.69	22.25	26.81

Table 7.5 *VLSI metrics for forward converters for* $S_5 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$

n	4	8	12	16	20
Dynamic Range	20	40	60	80	100
Results obtained by Speed Optimization					
Cell area	1044.2	2045.8	2977.2	4177	5198
Routing area	293.9	580.3	896.2	1274	1583
Total area	1338.1	2626.1	3873.4	5452	6781
Delay (ns)	3.01	3.32	3.48	3.45	3.71
Power (mW)	12.44	24.23	34.27	48.56	60.58
Results obtained by Area Optimization					
Cell area	630.6	1285.6	1948.5	2594	3291
Routing area	228.5	460.9	712.8	959	1243
Total area	859.1	1746.5	2661.3	3553	4534
Delay (ns)	4.46	6.24	8.30	10.27	12.21
Power (mW)	6.61	13.31	20.25	26.99	34.04

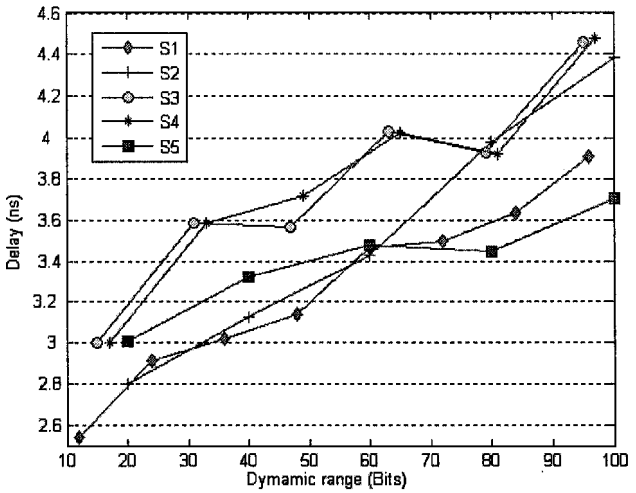


Figure 7.6 *Delay comparison of forward converters when optimized for speed*

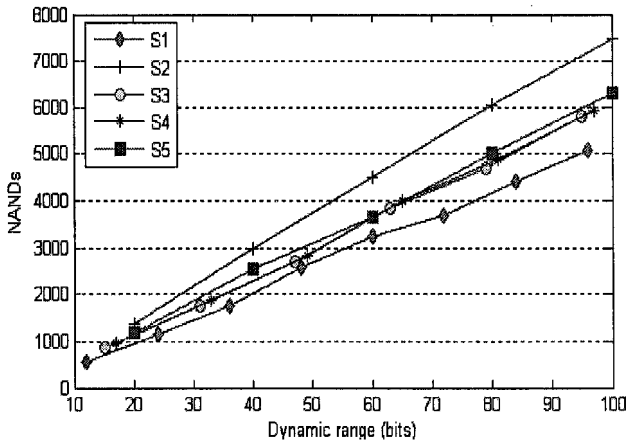


Figure 7.7 *Area comparison of forward converters when optimized for speed*

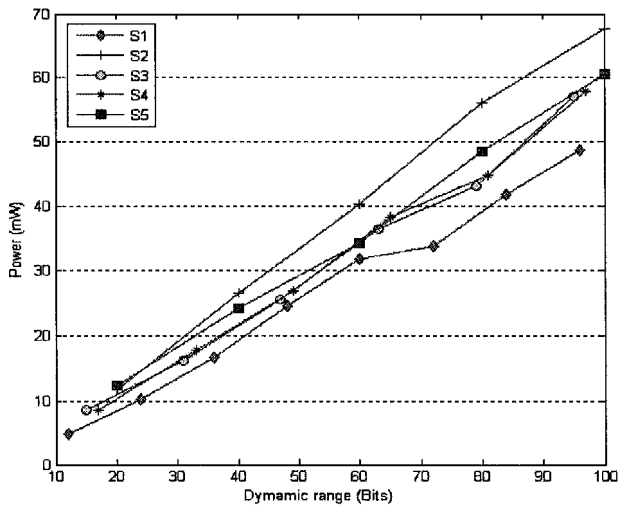


Figure 7.8 *Power comparison of forward converters when optimized for speed*

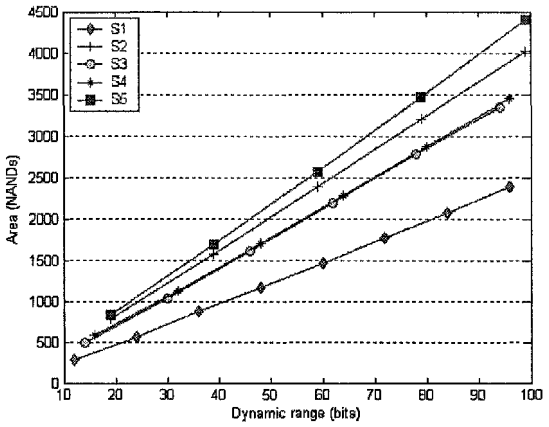


Figure 7.9 *Area comparison of forward converters when optimized for area*

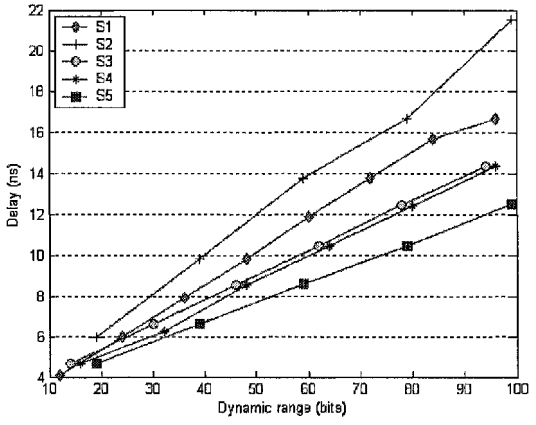


Figure 7.10 *Delay comparison of forward converters when optimized for area*

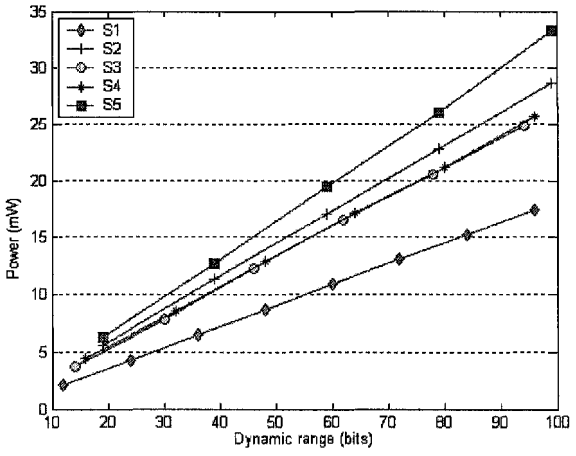


Figure 7.11 *Power comparison of forward converters when optimized for area*

As will be seen from the simulated results of later sections, the forward converter consumes only a very small fraction of the hardware of the entire RNS. The forward conversion is carried out only once per input, while the residue arithmetics are usually performed iteratively for applications that use RNS. In this perspective, higher cardinality RNS is advantageous in view of its smaller dimension residue channels than those of the lower cardinality RNS. Therefore, we will further investigate the VLSI performances of the reverse converters for these RNS's in the next section.

7.3 Reverse Converters for Triple Moduli Based RNS's

The reverse converters for the special moduli sets, S_1 to S_5, are implemented and compared in this section. The reverse converter for the triple moduli set S_1 is implemented based on the Converter II proposed in [Wang02]. The reverse converters for S_2, S_3, S_4 and S_5 have been described and analyzed in the preceding chapter 3, 4 and 5. Their structures are shown in Figure 4.1, Figure 3.5, Figure 3.2, and Figure 5.4, respectively. Four-stage pipelining is applied for the reverse converters for S_3, S_4, and six-stage pipelining is used for S_5. Similar to the prototypes of the forward converters, the 2-input modulo $2^n - 1$ adder proposed in [Efst94] and the modulo $2^n + 1$ adder proposed in [Hia02] are used for the two-operand additions. The binary adders and subtractors are implemented by CLA adders. When n is less than 8, one-level CLA structure is used, and for $n \geq 8$, two-level CLA structure is used. As in Section 7.2, the results are obtained from both area and timing driven optimized circuits. In delay comparison, the total conversion delays and pipelined delays for these reverse converters are compared separately. The pipelined delay is measured based on the critical path of the longest stage of the converter.

The synthesis results are listed in Tables 7.6 to 7.10 for each of these reverse converters. Figures 7.12 to 7.14 show the comparison of the area, total conversion delay and power versus dynamic range for all reverse converters which are optimized for speed. It is evident that the reverse converters for the four-moduli set, S_2 has very similar performances as those for the triple moduli set, S_1. It is also noted that the reverse

converters for S_3 and S_4 have also very similar performances. As expected, the reverse converter for the five-moduli set S_5 is the most costly case, and it consumes the most power and has the longest conversion delays. The pipelined conversion delays are shown in Figure 7.15. Even with pipelining, the six-staged pipelining delays for S_5 still exhibit the longest delays among these reverse converters. Figures 7.16 to 7.19 show the performance comparisons when the reverse converters are optimized for minimum area costs. The comparisons are very similar to those optimized for speed.

Table 7.6 *VLSI metrics of reverse converters for* $S_1 = \{2^n - 1, 2^n, 2^n + 1\}$

n	4	8	12	16	20	24	28	32
Dynamic Range	12	24	36	48	60	72	84	96
Results obtained by Speed Optimization								
Cell area	605.1	1404	2185	2932	3689	4271	5201	6115
Routing area	159.1	387	573	790	991	1227	1422	1676
Total area	764.2	1791	2758	3722	4680	5498	6624	7791
Delay (ns)	2.35	2.67	2.76	2.80	2.88	3.02	3.07	3.22
Power (mW)	6.86	16.68	27.1	35.67	45.7	51.43	62.74	75.76
Results obtained by Area Optimization								
Cell area	241.6	600	889	1182	1471	1876	2055	2513
Routing area	82.2	231	359	489	643	824	932	1132
Total area	323.8	831	1248	1671	2114	2700	2987	3645
Delay (ns)	3.91	4.82	5.12	5.14	5.99	6.01	6.40	6.75
Power (mW)	2.28	6.22	9.36	12.22	15.26	19.73	21.55	26.57

Table 7.7 *VLSI metrics of reverse converters for* $S_2 = \{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$

n	4	8	12	16	20
Dynamic Range	20	40	60	80	100
Results obtained by Speed Optimization					
Cell area	1272	2514	3592	5283	6561
Routing area	340	720	1040	1488	1871
Total area	1612	3234	4632	6771	8432
Delay (ns)	2.81	3.14	3.35	3.41	3.54
Power (mW)	15.77	30.76	43.33	64.88	80.30
Results obtained by Area Optimization					
Cell area	492.3	976.9	1453	1956	2456
Routing area	181.0	365.9	581.9	808.2	1016
Total area	673.3	1342.8	2034.9	2764.2	3472
Delay (ns)	5.07	6.14	6.88	7.66	9.07
Power (mW)	5.26	10.33	15.41	21.02	26.44

Table 7.8 *VLSI metrics of reverse converters for* $S_3 = \{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$

n	4	8	12	16	20	24
Dynamic Range	15	31	47	63	79	95
Results obtained by Speed Optimization						
Cell area	1400.3	3356	5531	7820	10888	13752
Routing area	363.4	911	1525	2218	3096	3991
Total area	1763.7	4267	7056	10038	13984	17743
Delay (ns)	7.61	9.79	11.09	11.79	12.94	13.46
Power (mW)	2.46	2.78	3.23	3.8	4.2	4.41
Results obtained by Area Optimization						
Cell area	552.1	1309.4	2144.	3094	4168	5334
Routing area	195.3	514.5	883.5	1259	1781	2408
Total area	747.4	1823.9	3027.5	4353	5949	7742
Delay (ns)	12.20	17.31	20.21	21.65	24.02	26.39
Power (mW)	3.98	5.18	5.34	5.76	6.28	7.48

Table 7.9 *VLSI metrics of reverse converters for* $S_4 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$

n	4	8	12	16	20	24
Dynamic Range	17	33	49	65	81	97
Results obtained by Speed Optimization						
Cell area	1699	3763	6060	8420	11238	14838
Routing area	445.6	1030	1677	2435	3260	4239
Total area	2144.6	4793	7738	10855	14499	19077
Delay (ns)	7.92	10.31	11.37	11.68	12.86	13.07
Power (mW)	2.60	3.14	3.47	3.70	4.29	4.65
Results obtained by Area Optimization						
Cell area	671.4	1460.4	2332	3317	4423	5663
Routing area	232.7	577.0	948.5	1362	1945	2615
Total area	904.1	2037.4	3280.5	4679	6368	8278
Delay (ns)	12.74	19.16	21.48	21.83	23.79	26.75
Power (mW)	4.03	5.11	5.83	5.90	6.70	7.72

Table 7.10 *VLSI metrics of reverse converters for* $S_5 = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$

n	4	8	12	16	20
Dynamic Range	20	40	60	80	100
Results obtained by Speed Optimization					
Cell area	753.2	6540	10782	15436	20181
Routing area	2834.9	1792	3047	4463	5973
Total area	3588.1	8332	13829	19899	26154
Delay (ns)	12.52	17.32	19.99	20.99	22.35
Power (mW)	4.16	5.48	6.66	7.06	7.41
Results obtained by Area Optimization					
Cell area	1194.1	2667.6	4338	6038	7999
Routing area	426.7	1050.7	1772	2517	3479
Total area	1620.8	3718.3	6110	8555	11479
Delay (ns)	18.00	28.91	34.74	36.41	39.09
Power (mW)	5.57	7.93	10.49	10.61	11.02

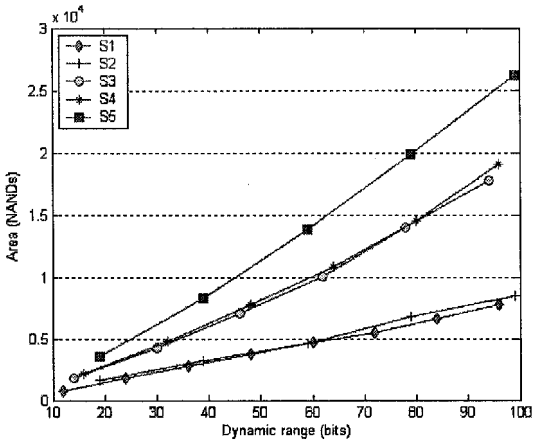


Figure 7.12 *Area comparison of different reverse converters optimized for speed*

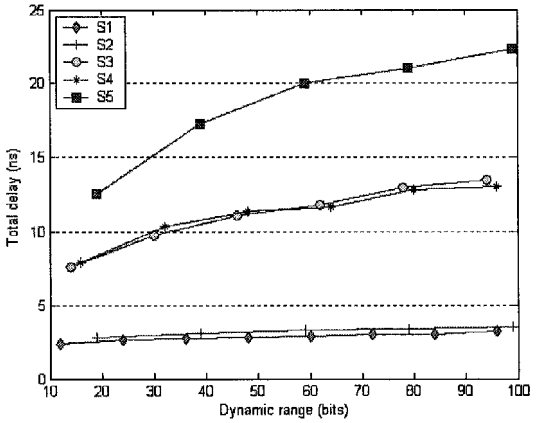


Figure 7.13 *Total delay comparison of different reverse converters optimized for speed*

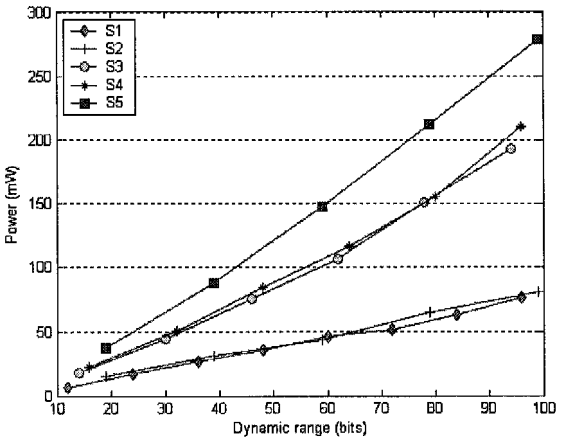


Figure 7.14 *Power comparison of different reverse converters optimized for speed*

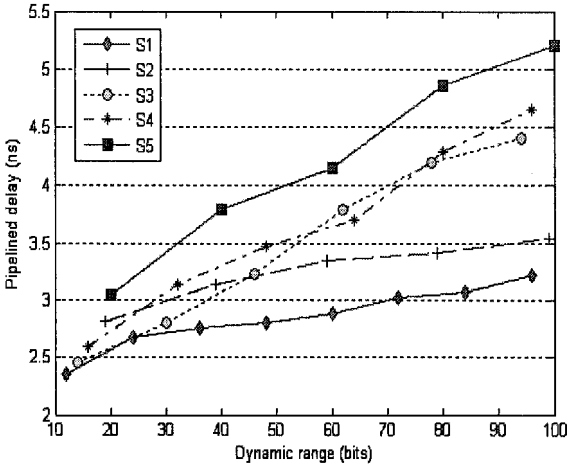


Figure 7.15 *Pipelined delay comparison of different reverse converters optimized for speed*

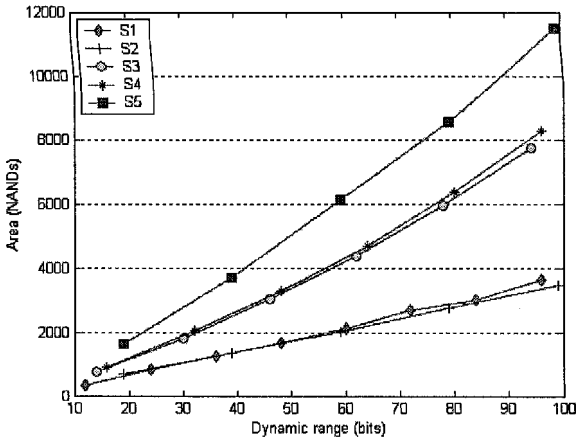


Figure 7.16 *Area comparison of different reverse converters optimized for area*

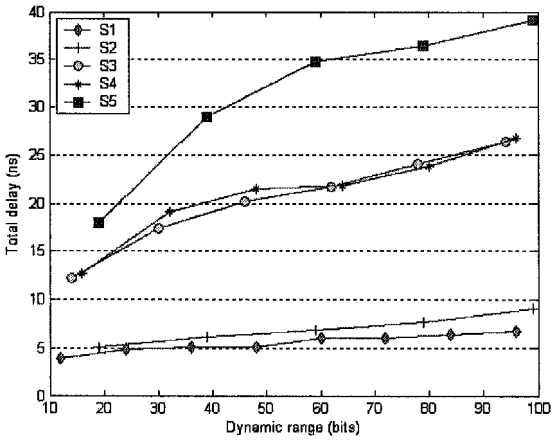


Figure 7.17 *Total delay comparison of different reverse converters optimized for area*

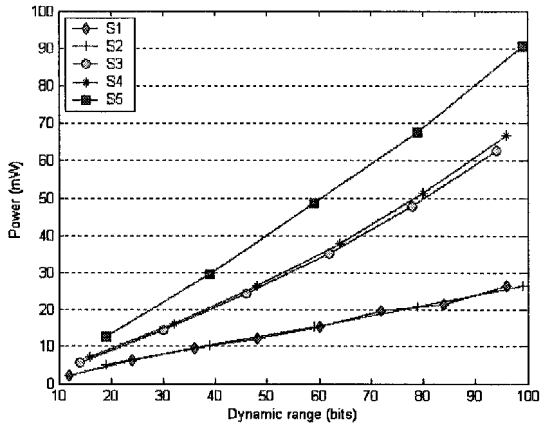


Figure 7.18 *Power comparison of different reverse converters optimized for area*

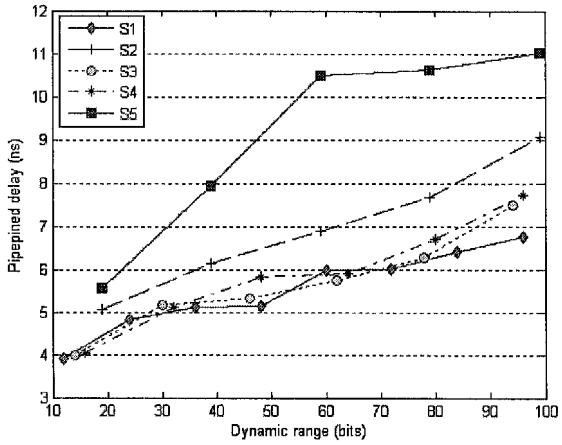


Figure 7.19 *Pipelined delay comparison of different reverse converters optimized for area*

7.4 Residue Number Systems for Triple Moduli Based RNS's

In this section, we present the holistic comparison of VLSI metrics for the five different RNS's completed with forward converters, reverse converters and generic residue arithmetic operations. RNS has been recognized for its merits for high-speed applications where only addition, subtraction and multiplications are involved. Therefore, typical applications that can be maximally benefited from the inherent parallelism of RNS include FIR filters, correlators, convolvers, DFT/FFT/DCT, etc.. Their use is promising in real-time applications, such as radio astronomy, real time image processing systems [Kal01] [Rej00].

Although the underlying algorithms vary from applications to applications, the basic and most commonly used operations required are addition and multiplication. Therefore, a generic operation of computing the inner product, $\sum_{i=0}^{N-1} X_i Y_i$ with programmable number of additions and multiplications is used as a vehicle to compare the overall performances of the RNS's composed from different special moduli sets. For most DSP applications, a 32-bit dynamic range is sufficient and the input signals are around 8-bit wide. For 32-bit RNS with 8-bit inputs, S_1 and S_4 don't require the forward converters because the input signals are already in the residue domain.

7.4.1 RNS Channel Structures

A generic structure is set for the evaluation of the five RNS's. Four sets of registers *A, B, C* and *D* are used to hold the values for calculation and for pipelining for each residue channel. There are two forward converters, one for each of the binary inputs *X* and *Y* (For some RNS's, forward converters are not needed). Figure 7.20 shows the RNS channel structure for the generic application, where the number of modular adders is $N-1$ and the number of modular multipliers is *N*. With this general structure, if the application is a FIR filter, then only one input channel has a forward converter, the other input channel holds the constant FIR coefficients instead of the variables. If the application requires cross-convolution or correlation, both input channels will have forward converters if necessary.

For FIR-like applications, *N* has a very wide range of potential values from less than 10 to several hundreds, and the structure has to be changed slightly. For the convolvers used in radio astronomy, the value of *N* may be around 1000, and the input signals normally have a very short wordlength [Bosi99]. Therefore, it is reasonable to select $N = 8$, 16 and 32 to carry out the comparison for 32-bit RNS. For modulo $2^n + 1$ channels, diminished-one number representation is used. The diminished-one modular adders proposed in [Ver02] are used for the addition of two residue numbers, and the MOMA proposed in

chapter 6 are used for the diminished-one multipliers. For modulo $2^n - 1$ channels, the modulo $2^n - 1$ adders proposed in [Efst94] and the modulo $2^n - 1$ multiplier proposed in [Wan96] are used.

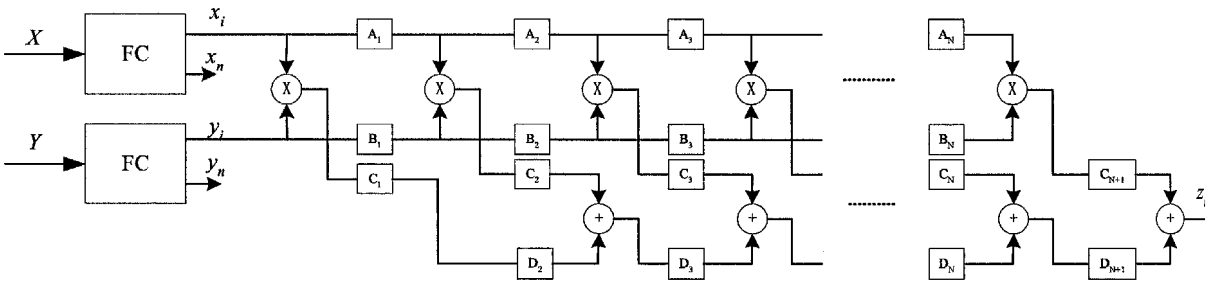


Figure 7.20 *RNS channel structure for the generic application*

Table 7.11 shows the parameters required for different RNS's. In Table 7.11, FC, RC, MUL and ADD are abbreviations for the number of forward converter channels, number of reverse converters, number of modular multipliers and number of modular adders, respectively. Due to the relative primality requirements of the moduli sets, the closest values of n that meet the desirable dynamic ranges are selected. The forward converters in Table 7.11 are simpler than those in Section 7.2, because now the wordlength of the input signals is only one or two bits wider than the corresponding values of n. All the designs are optimized to fulfill the required throughput and to achieve the minimum area cost. All the forward converters are to be completed within one clock cycle. The reverse converters for S_1 and S_2 are to be completed within one clock cycle, and the reverse converters for S_3 and S_4 are four-stage pipelined. The reverse converters for S_5 are pipelined into 6 stages. For the residue operations in each residue channel, the previous modular multiplication and current modular addition are performed in parallel in one cycle. Therefore, the total delays for these RNS's are as follows: $T_1 = T_2 = (N + 2) \times T_{cyc}$, $T_3 = T_4 = (N + 5) \times T_{cyc}$, $T_5 = (N + 7) \times T_{cyc}$, where T_{cyc} is the clock period, and T_i is the total delay for S_i.

Table 7.11 *Parameters for different RNS channels (Dynamic range = 32 bits)*

RNS	S_1			S_2			S_3			S_4			S_5		
n	11			6			8			8			6		
N	8	16	32	8	16	32	8	16	32	8	16	32	8	16	32
FC	0	0	0	4	4	4	2	2	2	0	0	0	8	8	8
RC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
MUL	8×3	16×3	32×3	8×4	16×4	32×4	8×4	16×4	32×4	8×4	16×4	32×4	8×4	16×4	32×4
ADD	7×3	15×3	31×3	7×4	15×4	31×4	7×4	15×4	31×4	7×4	15×4	31×4	7×4	15×4	31×4

7.4.2 Hardware Costs Analysis

The gate counts of different RNS's with N = 8, 16 and 32 are recorded in Table 7.12 where the clock frequency is set to 66.7MHz. The total hardware costs are not exactly equal to the sum of the gate counts of forward converter, reverse converter and residue arithmetics. The reason is that the gate count of the top level design includes wrappers at the module boundaries for interfacing to sub-designs. For ease of comparison, the gate count versus dynamic range for different RNS's is charted in Figure 7.21. Table 7.12 and Figure 7.21 clearly show that the reverse converter of S_5 has the least hardware costs, while the reverse converter of S_1 has the highest hardware costs.

Table 7.12 *RNS hardware costs (Tcyc=15 ns, f = 66.7MHz)*

	Forward Converter	Reverse Converter	Residue arithmetics			Total		
			$N=8$	$N=16$	$N=32$	$N=8$	$N=16$	$N=32$
S_1	0	820	33374	65334	132712	45915	91035	184197
S_2	349	744	28794	58678	116209	40100	80832	160954
S_3	189	1463	25026	50849	102818	35980	71088	142790
S_4	0	1682	28293	56642	114550	39847	76729	154758
S_5	692	2105	21790	45600	89763	33280	65544	127003

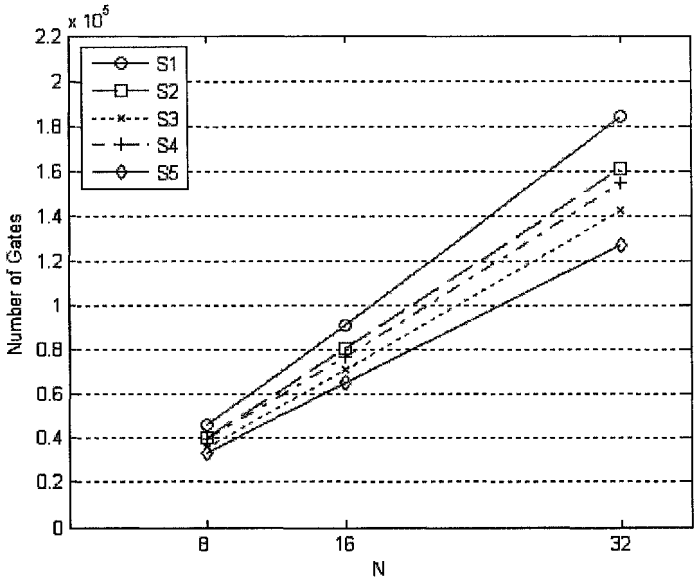


Figure 7.21 *Total areas comparisons for RNS with different values of N*

The breakdown of the hardware costs in terms of the number of gates for the arithmetic operations involved in each residue channel is tabulated in Table 7.13. To show the relative cost of arithmetic operations in different residue channels for each RNS, the data of Table 7.13 are charted in three plots for $N = 8$, 16, 32 in Figure 7.22. With the dynamic range fixed to 32-bit, the values of n for S_1 to S_5 are 11, 6, 8, 8, and 6 bits, respectively. Due to the significantly longer channel length of S_1 over other RNS's, the modulo operations are more costly to implement in every channel of S_1 than those for other moduli sets. For example, from Table 7.13, the gate count of channel 1 of S_1 is approximately 2.7 times higher than those of the corresponding channel of S_2 and S_5. In addition, the proportion of hardware resources used in each RNS channel is relatively balanced except for S_2. Among those balanced RNS's, the hardware costs for modulo 2^n channels are lower than those for modulo $2^n - 1$, and the hardware costs for modulo $2^n + 1$ channels are higher than those for modulo $2^n - 1$. Figure 7.23 shows the relative composition for the RNS, where the residue operations are collectively dumped into the Residue Arithmetic Unit (RAU). It is noted that the RAU occupies the most part of the hardware costs, while the reverse/forward converters consume only a very small portion. For example, when $N = 8$, 16 and 32, RAU occupies 97.6%, 98.7% and 99.4% of the total hardware costs, respectively.

Table 7.13 *Break down of hardware costs for the arithmetic operations of each residue channel (Tcyc=15 ns, f = 66.7MHz)*

N	Channel No.	S_1	S_2	S_3	S_4	S_5
8	1	10492	4414	7032	7030	4316
	2	6519	2356	3588	3580	2457
	3	13063	5179	7906	7906	5287
	4		16045	5600	8567	5606
	5					3424
16	1	22643	9138	12468	12466	9266
	2	11409	5329	7611	7613	5364
	3	23282	9278	14844	14844	10645
	4		29333	9826	15617	10335
	5					7290
32	1	47675	16531	27272	27274	16531
	2	22148	8565	12682	12690	8626
	3	51889	20006	31925	31926	20101
	4		63107	21939	31660	21963
	5					12542

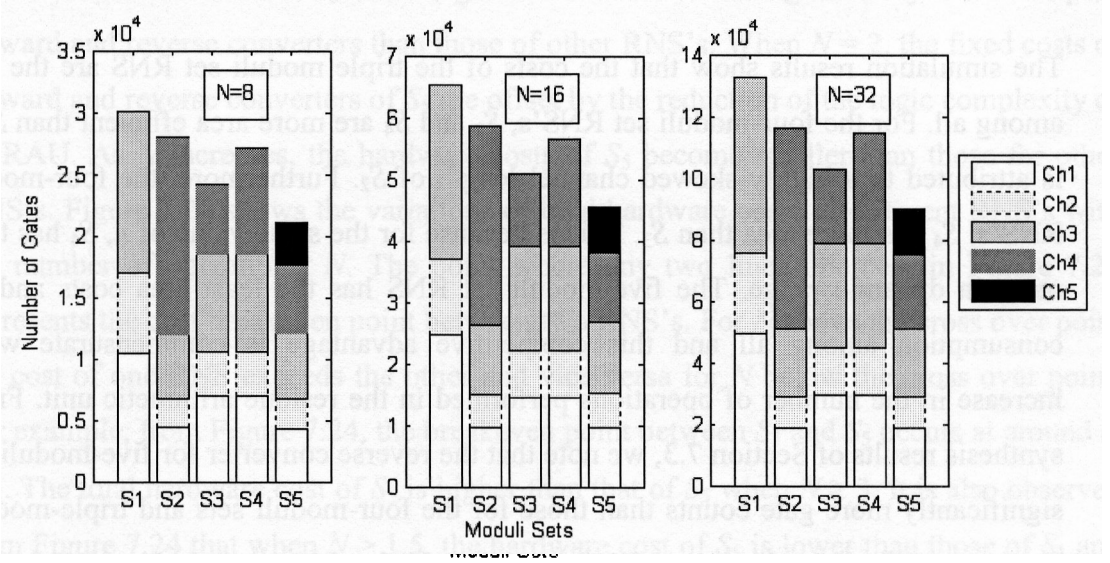


Figure 7.22 *Break down of hardware costs for the arithmetic operations*

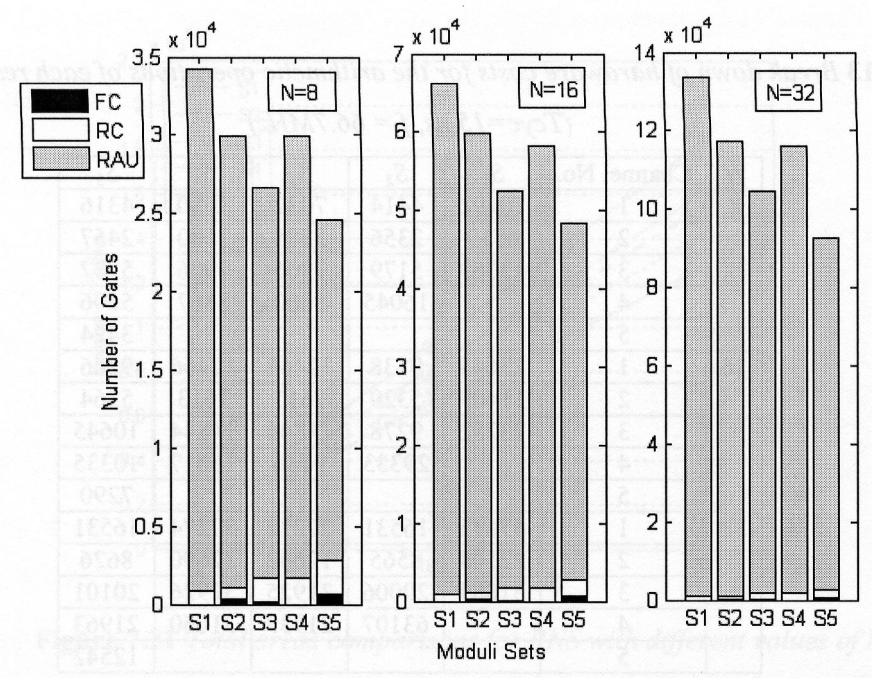


Figure 7.23 *Composition of hardware costs for the RNS*

The simulation results show that the costs of the triple moduli set RNS are the highest among all. For the four-moduli set RNS's, S_3 and S_4 are more area efficient than S_2. This is attributed to a highly skewed channel length of S_2. Furthermore, the four-moduli set RNS's, S_4 has more area than S_3. This is because for the same value of n, S_4 has two bits more in dynamic range. The five-moduli set RNS has the least area costs and power consumption among all and this competitive advantage is commensurate with the increase in the number of operations performed in the residue arithmetic unit. From the synthesis results of Section 7.3, we note that the reverse converter for five-moduli set has significantly more gate counts than those for the four-moduli sets and triple-moduli set. However, the results in Figure 7.23 show that the hardware requirement of the reverse converter constitutes only a very small fraction of the total cost of an RNS. For high-cardinality moduli sets, smaller value of n will suffice to fulfill the dynamic range, resulting in a smaller word length residue for each channel. Thus, the modulo arithmetic operations can be largely simplified. For a fixed dynamic range, the optimization tool needs less effort to meet the timing constraints for the higher cardinality RNS and hence it is more likely to generate lesser gates than the lower cardinality RNS. Consequently, it

is also easier for the high-cardinality RNS to achieve a higher frequency of operation (i.e., higher throughput rate) than that for the lower cardinality RNS.

Since the RAU dominates the cost of RNS, a more insightful benchmarking is provided in Table 7.14 where the trade-off analysis of different RNS's are performed with reference to the complexity of their RAUs. The RAU which is composed of the modulo operations required to perform N multiplications and $N - 1$ additions of Figure 7.20 is used as the basis for comparison. When $N = 1$, the average hardware costs for the RAU are more than those for the forward and reverse converters except for S_5. For higher cardinality moduli sets, each residue channel has a smaller width than that for the smaller cardinality moduli sets. Comparatively, there is more timing margin for the EDA tools to trade for reduced logic area. Therefore, the average hardware cost for each residue channel of the high cardinality moduli sets is smaller than that for the small cardinality moduli sets. When $N = 1$, S_5 has the greatest cost due to its significantly more complex forward and reverse converters than those of other RNS's. When $N = 2$, the fixed costs of forward and reverse converters of S_5 are offset by the reduction of the logic complexity of its RAU. As N increases, the hardware costs of S_5 become smaller than those for other RNS's. Figure 7.24 shows the variations of total hardware costs of different RNS's with the number of operations, N. The point where any two lines intercept in Figure 7.24 represents the cost breakeven point between two RNS's. For N above the cross over point, the cost of one RNS exceeds the other and vice versa for N below the cross over point. For example, from Figure 7.24, the breakeven point between S_3 and S_5 occurs at around $N = 3$. The total hardware cost of S_3 is higher than that of S_5 when $N \geq 3$. It is also observed from Figure 7.24 that when $N > 1.5$, the hardware cost of S_5 is lower than those of S_1 and S_4. When $N \geq 2$, the hardware cost of S_5 is lower than that of S_2.

Table 7.14 *Total costs of various RNS's for different number of operations in their RAUs*

N		1	2	3	4
S_1	RC+FC	820	820	820	820
	RAU	4071	8242	12413	16584
	Total	4891	9062	13233	17404
S_2	RC+FC	1093	1093	1093	1093
	RAU	3499	7098	10697	14296
	Total	4592	8191	11517	15398
S_3	RC+FC	1652	1652	1652	1652
	RAU	3028	6156	9284	12412
	Total	4680	7808	10936	14064
S_4	RC+FC	1682	1682	1682	1682
	RAU	3436	6972	10508	14044
	Total	5118	8654	12190	15726
S_5	RC+FC	2797	2797	2797	2797
	RAU	2623	5346	8069	10792
	Total	5420	8075	10866	13589

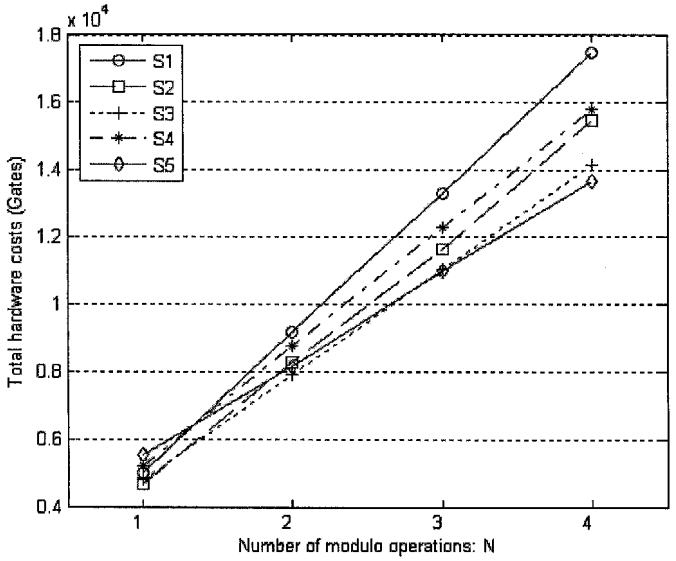


Figure 7.24 *Total hardware costs of different RNS's versus number of operations in their RAUs*

7.4.3 Power Consumption Analysis

The data for average power consumption comparison in mW are recorded in Table 7.15, and Figure 7.25 shows the total power consumption of various RNS's for $N = 8$, 16 and 32. The average power consumptions of the RNS's are more than the sum of those of forward converter, reverse converter and RAU simulated independently. The reason is

that the wrapper logic for boundary interface to the modules has been ignored when they are simulated independently. In general, the power consumption comparisons share similar trends and proportions as those discussed for the gate count earlier. The composition of the power consumptions for every RNS is shown in Figure 7.26. It is evident that the contributions of the forward converter and the reverse converter to the total power consumption are insignificant compared to the RAU. The average power consumptions for the arithmetic operations involved in each residue channel are also tabulated in Table 7.16 and charted in Figure 7.27. Except for S_2, the power consumptions in each residue channels are comparable. Among the first three residue channels, the modulo $2^n + 1$ channels dissipate the most power and the modulo 2^n channels dissipate the least power. Due to the bit width variation for the same value of n, channel 5 of S_5 dissipates the least power and channel 4 of S_2 dissipates the most power of all RNS's. It is interesting to note that the ratio of power dissipations of different channels for the same RNS remains relatively constant as N varies.

Table 7.15 *Comparison of power consumption (mW) of different RNS's*

	FC	RC	Residue arithmetic			Total		
			$N=8$	$N=16$	$N=32$	$N=8$	$N=16$	$N=32$
S_1	0	3.28	152.7	288.8	584.5	875.4	1018	1314
S_2	1.75	4.38	133.4	253.3	523.8	872.1	963.4	1204
S_3	0.11	7.67	122.9	253.0	506.4	865.6	960.4	1187
S_4	0	9.7	138.4	280.6	517.6	870.8	980.5	1210
S_5	3.38	9.17	99.11	212.3	411.5	850.2	945.4	1150

Table 7.16 *Break down of power consumption of arithmetic operations per residue channel*

N	Channel No.	S_1	S_2	S_3	S_4	S_5
8	1	48.66	14.11	34.81	34.86	15.13
	2	35.39	18.98	22.91	22.95	16.05
	3	58.66	29.73	40.01	40.00	28.08
	4		72.62	30.15	36.55	25.66
	5					14.18
16	1	99.04	30.28	72.21	71.67	31.98
	2	70.89	40.27	46.87	46.61	33.64
	3	118.9	63.35	80.24	80.24	57.76
	4		136.4	51.73	73.11	58.25
	5					30.65
32	1	200	70.85	144.3	144.2	63.33
	2	144.3	85.94	94.12	94.33	65.85
	3	240.2	117.14	163.5	164.32	115.4
	4		269.9	104.5	150.8	106.1
	5					60.77

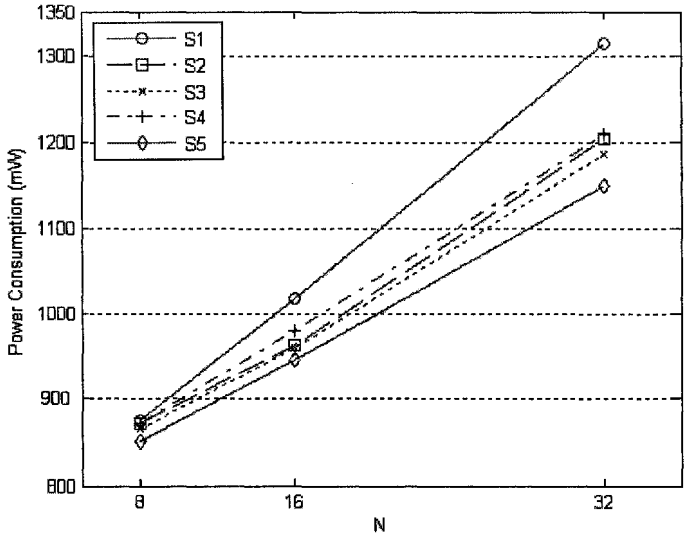


Figure 7.25 *Total power consumptions of RNS's versus number of operations, N*

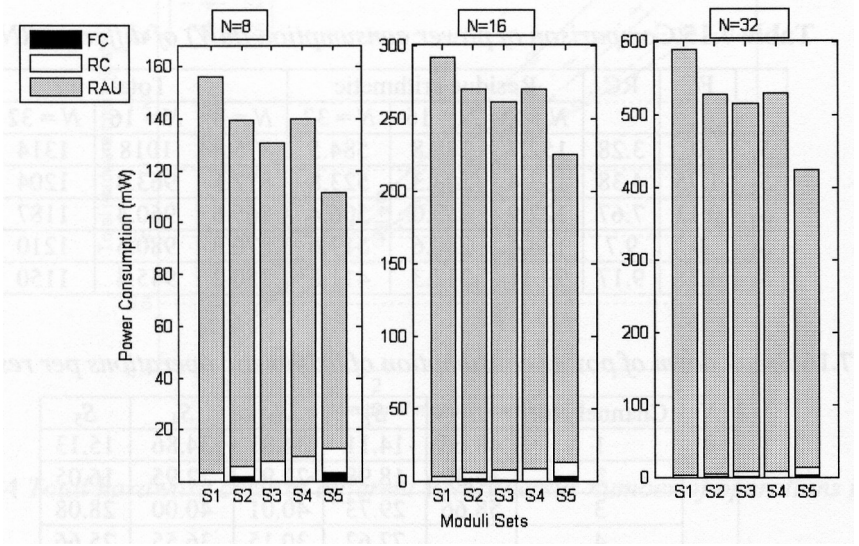


Figure 7.26 *Composition of power consumptions for different RNS's*

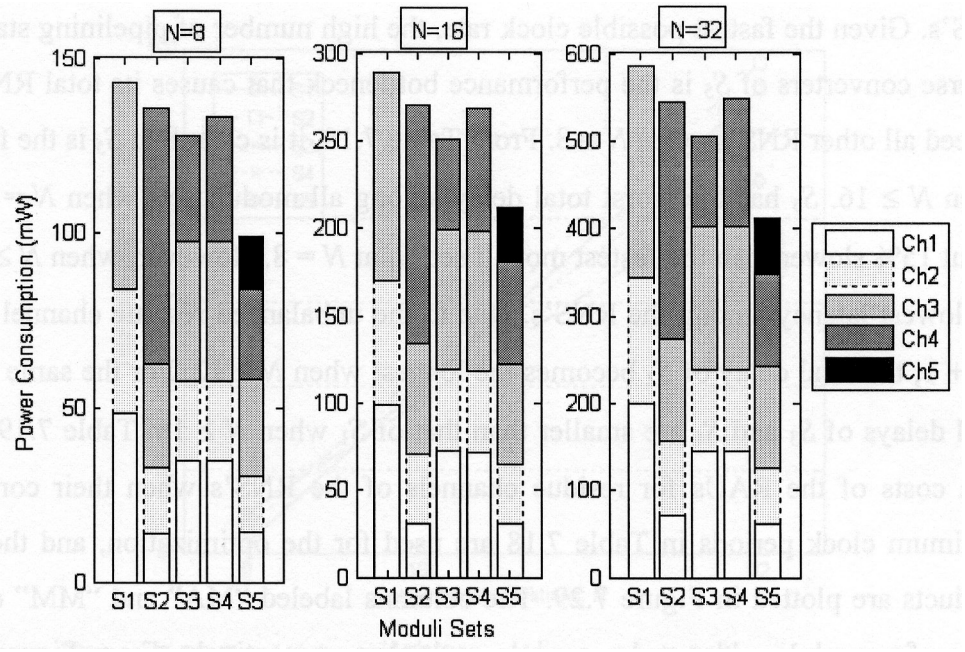


Figure 7.27 *Power consumptions of residue arithmetic computations of RNS apportioned by residue channels*

7.4.4 Delay and Area-Time Analysis

In this section, the total delay and the area-time products are evaluated. The total delays of various RNS's are provided in Table 7.17. In Table 7.17, the clock period is 15 *ns*, and the designs are optimized for minimum area. With fixed cycle time, S_5 has the longest latency because it uses the greatest number of pipelining stages. Now let's evaluate the minimum clock cycles for these RNS's. As assumed in Section 7.4.1, the clock cycle is determined by the delay of the modulo $2^{2n} + 1$ multiplier in channel 4 for S_2 and that of the modulo $2^n + 1$ multiplier in channel 3 for the other four RNS's. The critical channel of S_5 has the smallest word length due to the smallest value of n needed to meet the desired dynamic range. Therefore, S_5 can be executed with the fastest clock rate or the minimum cycle time among the RNS's being compared. For a dynamic range of 32-bit, the minimal clock periods are determined by the delays of modulo $2^{11} + 1$, $2^{12} + 1$, $2^8 + 1$, $2^8 + 1$ and $2^6 + 1$ multipliers for S_1, S_2, S_3 S_4 and S_5, respectively. The minimum clock periods and the total delays for these RNS's are listed in Table 7.18 and plotted in Figure 7.28 to provide an intuitive view of the breakeven number of operations between any two

RNS's. Given the fastest possible clock rate, the high number of pipelining stages for the reverse converters of S_5 is the performance bottleneck that causes its total RNS delay to exceed all other RNS's when $N = 8$. From Table 7.18, it is clear that S_5 is the fastest RNS when $N \geq 16$. S_5 has the worst total delay among all moduli sets when $N = 8$ and it is about 15% slower than the fastest moduli set, S_1 at $N = 8$. However, when $N \geq 14$, S_5 has the lowest latency among the RNS's. Due to the unbalanced residue channel of modulo $2^{2n} + 1$, the total delay of S_2 becomes the longest when $N \geq 12$. For the same reason, the total delays of S_3 and S_4 are smaller than that of S_1 when $N \geq 16$. Table 7.19 shows the area costs of the RAUs for residue channels of the RNS's when their corresponding minimum clock periods in Table 7.18 are used for the optimization, and the area-time products are plotted in Figure 7.29. The columns labeled "MA" and "MM" denotes the costs of a modulo adder and a modulo multiplier, respectively. From Figure 7.29, it is obvious that S_5 has the best overall performance, while S_3 and S_4 have better overall performance than S_1 and S_2, and S_1 has the worst overall performance for $N = 8$ and above. The performance gaps among these modulo widen as N increases.

Table 7.17 *Total delays of the RNS's with cycle time fixed at 15 ns*

Total delay	S_1	S_2	S_3	S_4	S_5
$N = 8$	150	150	195	195	225
$N = 16$	270	270	315	315	345
$N = 32$	510	510	555	555	585

Table 7.18 *Minimum clock periods of RNS's and total RNS delays for different N*

	n	Clock cycles (*ns*)	Total delay (ns)		
			$N = 8$	$N = 16$	$N = 32$
S_1	11	4.42	42.2	79.56	150.28
S_2	6	4.61	46.1	82.98	156.74
S_3	8	3.81	49.53	80.01	140.97
S_4	8	3.81	49.53	80.01	140.97
S_5	6	3.31	49.65	76.13	129.09

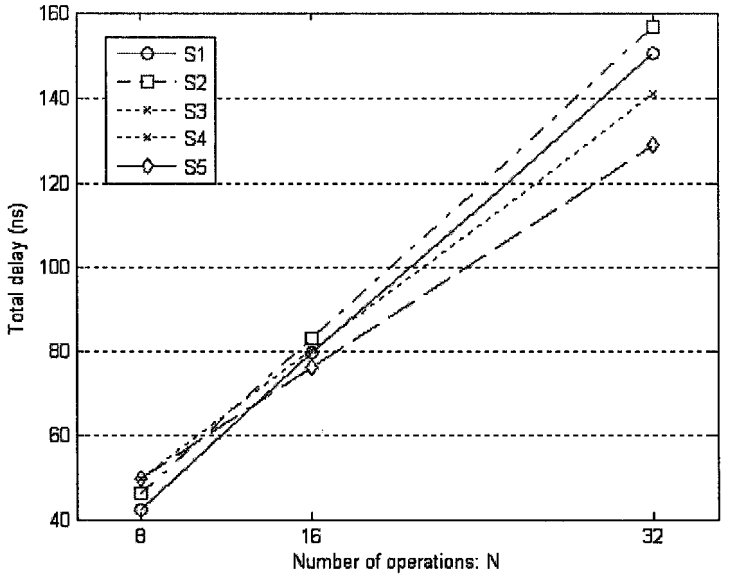


Figure 7.28 *Total delays of RNS's at their optimal clock rates versus N*

Table 7.19 *Costs of RAUs for residue channels of the RNS's*

Moduli Set	Channel	MA	MM	$N=8$	$N=16$	$N=32$
S_1	$2^{11}-1$	236	1945	17212	34660	69556
	2^{11}	179	499	4545	10669	21517
	$2^{11}+1$	314	2724	23990	48294	96902
	Total			45747	93623	187975
S_2	2^{6}-1	125	453	4499	9123	18371
	2^{6}	72	167	1840	3752	7576
	2^{6}+1	137	652	6175	12487	25111
	2^{12}+1	341	3162	27683	55707	111755
	Total			40197	81069	162813
S_3	$2^{8}-1$	162	1016	9262	18686	37534
	2^{8}	131	446	4485	9101	18333
	$2^{8}+1$	206	1548	13826	27858	55922
	$2^{7}-1$	144	801	7437	14976	30096
	Total			35010	70621	141885
S_4	$2^{8}-1$	162	1016	9262	18686	37534
	2^{8}	131	446	4485	9101	18333
	$2^{8}+1$	206	1548	13826	27858	55922
	$2^{9}-1$	187	1293	11653	23493	47173
	Total			39226	79138	158962
S_5	2^{6}-1	126	558	5346	10818	21762
	2^{6}	73	167	1847	3767	7607
	2^{6}+1	140	919	8332	16804	33748
	2^{5}-1	115	408	4069	8253	16621
	2^{7}-1	144	927	8324	16992	34128
	Total			19945	56634	113866

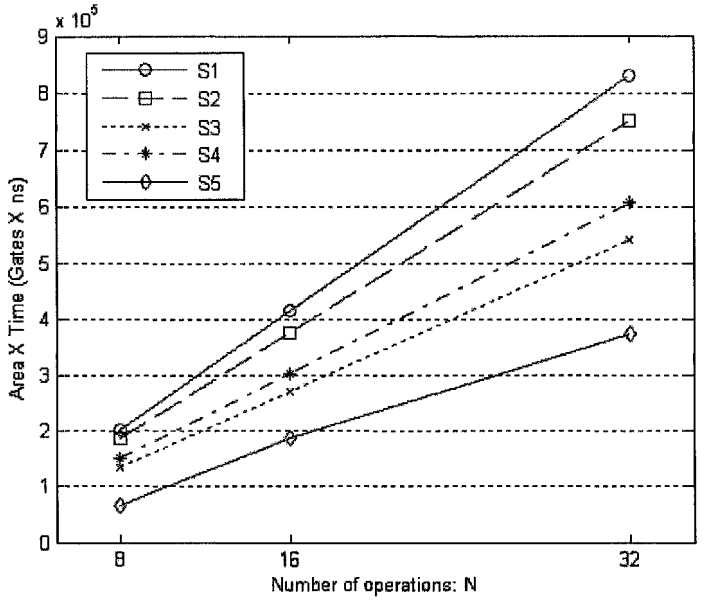


Figure 7.29 *Area-time product of different RNS's versus N*

7.5 Conclusions

In this chapter, we presented the architectures of the forward converters, and the optimization results for these forward converters which are based on triple-moduli set RNS's presented in earlier chapters. In addition, we implemented the reverse converters of these five RNS's in register transfer level. Both the forward and reverse converters have been synthesized to compare and analyze their VLSI performance in terms of area (measured in terms of the number of gates), delay and power consumption. To better understand the trade-offs of increased parallelism and associated overheads for different triple moduli based RNS's presented in this thesis, we built a RNS-based generic processor with programmable number of modulo operations that typifies the applications of RNS. Five RNS's are modeled in this general structure for different dynamic ranges and different number of modular operations. The rigorous evaluation and analysis of the VLSI performance metrics for the triple moduli based RNS's have led to the following useful guidelines and conclusions pertaining to the choice of an appropriate RNS according to the context of the application.

1. Due to the overheads in area, time and power consumption of the forward and reverse converters, RNS shall be reserved for applications that involve intensive additions, subtractions and multiplications. This is true even when the dynamic range is large and high cardinality moduli set is used.

2. In applications where RNS is useful, it is found that the modulo operations in the RAU dominate the total VLSI performance metrics of the RNS. For this reason, special moduli sets of 2^n or $2^n \pm 1$ type shall be considered as they are very efficient for the reverse converter in all aspects of VLSI metrics. Besides, the implementations of the modulo adder and multiplier with these moduli are much simpler and efficient than those for the general moduli. Of which, moduli of $2^n - 1$ type is better than that of $2^n + 1$ type for the same value of n.

3. For the triple moduli based RNS, the higher cardinality RNS usually has better performances than the lower cardinality RNS. Although the reverse converter for the higher cardinality RNS is far more inferior than that for lower cardinality RNS, the VLSI performance metrics of the reverse converter is insignificant compared to those of the RAU for applications that justify the use of RNS. The increased parallelism gained by the higher cardinality RNS outweighs the conversion overhead in those applications.

4. To achieve the same throughput rate (clock frequency), higher cardinality RNS will achieve overall better VLSI performances. This is because the optimization tools need lower effort to optimize the logic as the wordlength of each RNS channel is shorter for the same dynamic range. As the average power consumption was observed to have a direct correlation with the amount of logics used, higher cardinality RNS tends to dissipate lower power as well. If no timing constraints based on the critical channel is imposed, VLSI performance metrics of higher cardinality RNS will be higher than those of the lower cardinality RNS. However, this scenario is unlikely to happen in the context of RNS application. The primary

purpose of applying RNS over classical approach is to speedup applications through parallel processing, where high throughput rate is the premier factor.

5. To achieve efficient VLSI implementation of RNS, it is better to choose a balanced moduli set, particularly when the dynamic range is large. For example, moduli sets S_3 and S_4 are superior to S_2, because S_2 has a highly skew channel. However, due to the larger dynamic range of S_2 for the same value of n, this disadvantage of unbalance is not prominent for 32-bit dynamic range applications. If the dynamic range increases, the overall performance of an unbalanced moduli set RNS will be significantly pulled down by the critical channel.

6. Based on the simulation data and the analysis performed in Sections 7.4.2, 7.4.3 and 7.4.4, we have summarized them into a simple designer guide for choosing the five moduli sets for a RNS inner product processor with N pairs of modular multipliers and modular adders. This general rule of thumb will also be applicable to other RNS applications such as FFT, FIR and correlator as they possess similar structure as that of Figure 7.20. In Table 7.20, a score of 1 to 5 is used to rate the choices of different moduli sets, with 1 being the most preferred choice and 5 being the worst decision. A 16-tap RNS FIR filter of [Con04] is a good practical example to validate these decision rules.

Table 7.20 *Ratings of different moduli sets for RNS applications*

N	S_1	S_2	S_3	S_4	S_5
1-2	3	2	1	4	5
3-4	5	2	3	4	1
≥ 4	5	4	2	3	1

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The work presented in this thesis aimed at exploring VLSI efficient RNS through the study of special moduli sets and their underlying VLSI properties in RNS computations. Main emphasis was placed on moduli sets extended from the triple moduli sets and composed of only moduli of 2^n or $2^n \pm 1$ types. Our work focused on such supersets so defined, for reasons that the corresponding modulo arithmetic operations are generally more efficient than those involving other special formats. In addition, their forward and reverse converters also exhibit superior VLSI performances in comparison with general moduli sets or other special moduli sets. A systematic method to evaluate the VLSI metrics in terms of area, delay and power measures for the RNS processors based on such special moduli sets is proposed. Through rigorous evaluation criteria for the reverse converters, forward converters as well as modulo arithmetic amortized over the residue channels, the VLSI aspects of the RNS processor with these special moduli sets are scrutinized.

In short, the contributions of this work are summarized as follows:

We have proposed new reverse converters for two special moduli sets, $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$, with even value of n, which are extensions of the triple-moduli sets. These reverse converters provide for a choice to minimize wastage of dynamic range for a given application. Comparing to the existing reverse converters, the proposed reverse converters achieve better performance while using less hardware

and consuming less power.

New CRT has been employed to realize a new four-moduli reverse converter with a larger dynamic range of $5n$ bits. Due to the special property of this moduli set and the use of New CRT, its reverse converter has been shown to be as efficient as that for the triple-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$.

A new five-moduli set, $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, with even value of n, has been proposed, together with its area-time efficient reverse converter. The closed-form expressions for the multiplicative inverses have smaller Hamming weights, resulting in smaller and simpler CSAs with EAC, and shorter wordlength modulo $2^n - 1$ adders.

A general structure for the design of diminished-one modulo $2^n + 1$ MOMA is proposed. An analytical modeling of the diminished-one modulo $2^n + 1$ MOMA unifies its structure with that of the modulo $2^n - 1$ MOMA. By exploiting the efficient two-operand modulo $2^n + 1$ adder as the final carry propagation adder, our proposed diminished-one MOMA is as fast as that for modulo $2^n - 1$ MOMA.

Finally, a generic RNS inner product step processor with programmable number of multiplication and accumulation taps is proposed as a general application platform to simulate the VLSI performances of the overall RNS comprising the forward converter, the RAU, and the reverse converter. RNS's for all the above four supersets and the referenced triple moduli sets are synthesized and optimized by commercial EDA tools based on the 0.35μm CMOS standard cell library. The overall delay, area and power consumption of the RNS processor for each special moduli set have been reported to show that a five-moduli set RNS processor is superior to other RNS processors based on special moduli sets including the popular triple moduli set.

8.2 Recommendations for Further Work

Based on the work presented in this thesis, we suggest several interesting relevant topics for further research. Some of them are pertaining to the RNS itself, and others are pertaining to the applications of the RNS.

1. High-cardinality RNS based on the triple moduli set. In this thesis, we proposed a five-moduli superset $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$. New higher cardinality sets may be proposed as the number of moduli increases by keeping the exponents of the $n \pm k$ forms. Even if these high-cardinality sets are not co-prime for some values of n, provided that their multiplicative inverses possess a small hamming weight, their reverse converters will remain VLSI efficient. With increased parallelism, RNS's constructed on these moduli sets will be envisaged to be more efficient than the existing RNS.

2. Resource-sharing RNS processor. In a RNS processor, there are several residue channels; each performs some application-specific residue arithmetic operations. In order to reduce the hardware costs, sharing of common hardware resources among residue channels for the residue arithmetic operations is proposed in [Pali99] [Pali01]. However, those proposed structures calculate $|A \times B|_{m_i}$ instead of the desirable multiplications of $|A_i \times B_i|_{m_i}$ corresponding to various residue channels. As the characteristic of different residue channels are vastly different for general moduli set, it might be impractical to share the hardware resources among different RNS channels. For the special moduli sets of $2^n \pm 1$ types, due to the special number theoretic properties, unique analytical scheduling model can be explored to develop maximally efficient resource sharing architectures.

3. In Chapter 6, we proposed a general structure for the diminished-one MOMA. The proposed structure is similar to that of modulo $2^n - 1$ MOMA. An immediate application is that of the dual moduli multipliers for moduli $2^n - 1$ and $2^n + 1$. A configurable dual moduli MOMA has been proposed by our research group in

[Men05]. This dual moduli MOMA can be further extended to include the modulo 2^n MOMA so that a triple moduli MOMA for the triple moduli set RNS can be developed. The common primitive computations among the modulo $2^n - 1$, 2^n and $2^n + 1$ parallel prefix or carry lookahead adders can also be explored. Integrating these similarly structured MOMA and the two-operand modular adders, a dual or triple moduli multiplier configurable as either modulo $2^n - 1$, 2^n and $2^n + 1$ multiplier with very small configuration overhead will become possible for configurable modular processor.

4. RNS applications in cryptosystems. RNS has been used successfully in public key cryptosystems, RSA for high-speed implementation [Noz01] [Imb02] [Yen03]. The base transformation is a critical operation for the RNS-based Montgomery multiplication [Baja98]. Due to the moduli sets used for the RNS based RSA are not special moduli sets, the base transformation takes a long time. With the use of the special moduli sets such as the supersets of $2^n \pm 1$ types, there is a potential to improve the base transformation operation. If the triple-moduli set can be exploited to derive the base transformation for non-special moduli sets, then higher cardinality supersets may help to achieve more efficient solutions for the base transformation. The applications of RNS in Elliptic curve cryptosystem (ECC) are even more promising due to its smaller key length than RSA for similar cryptographic strength. High-cardinality special moduli sets can be used to achieve a fully parallel implementation of ECC for a very high-speed computation and possibly consuming lesser power and area than the conventional implementation with similar speed.

5. System-level methods to improve the performances of RNS, such as Quadratic RNS [Ram00], and Polynomial RNS [Abd99], can be explored and applied to the triple moduli based RNS's proposed in this thesis. With these system-level methods, the methods for designing DCT/FFT, FIR, and the cryptographic systems will be different from the way we proposed in this thesis, and their performances can be further enhanced.

6. Future performance quantifier of a VLSI system will be far more complex than what it is today. It may be expressed as a general function of speed, area, power, precision, reliability, testability, serviceability, etc.. The theoretically well structured redundancy of special moduli set RNS can help to fulfill several of those aspects simultaneously. Redundancy has been widely exploited for fault tolerant computation. Is it possible to design a scheme, leveraging on the number theoretic property of well defined high-cardinality RNS for error resilient high-speed computation? With the increased parallelism and the reduction of carry propagation path, we envisage intuitively that the power consumption will be lowered as there will be smaller number of circuit nodes switching and lower capacitive loadings. Can these complex effect be reasonably modeled for a VLSI application built based on RNS? Investigations to answer these open questions may lead to interesting and unexpected outcomes.

Appendix

A.1 EDA tools and technology libraries

1. Simulation tool: Mentor Graphics ModelSim SE 5.6.

2. Synthesis tool: Synopsys Design Compiler.

3. Platform: SunOS

4. Libraries

Technology libraries are Avant!'s Libra-Passport 0.35um V2.6 CMOS technology libraries. The cb35os142 Library is a high-performance, standard cell library in 0.35-micron CMOS process (cmpsc3514). It supports 3 and 4 metal layer processes.

The recommended operating conditions for cb35os142 library are as follows:

Parameters	Minimum	Maximum	Conditions
Power Supply	3.0V	3.6V	
V_{IL}, low level Input Voltage CMOS Input TTL Input	 -0.33V -0.33V	 0.2 x VDD 0.8V	Guaranteed Input Low voltage
V_{IH}, high level Input Voltage CMOS Input TTL Input	 0.7 x VDD 2.0V	 VDD+0.5V VDD+0.5V	Guaranteed Input high voltage
Junction Temperature	0 °C	125 °C	

The CB35IO122 library is made up of low voltage chip interface circuits powered by a voltage in the range from 3.0V to 3.6V. The library is designed to be used with the cb35os142 3.3 Volt standard cell library.

A.2 Examples of VHDL codes

Bellow is part of the VHDL codes for the forward converter for moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ when $n = 20$. The codes include the top-level code, the channels for residues $2^n - 1$, $2^n + 1$, and $2^{n-1} - 1$, respectively. The codes for the top-level, for the residue channel $2^n - 1$, and the test bench are shown as follows. Part of the test vectors are also shown after the VHDL codes.

Top level: ntyFWC_cb4m1_20.vhd

```
-- Forward converter for four-moduli set {2^n - 1, 2^n, 2^n + 1,2^(n-1) - 1},
-- n= 20
-- Top level

Library IEEE;
USE ieee.std_logic_1164.all;

ENTITY ntyFWC_cb4m1_20 IS
        GENERIC(gN : integer := 4 );
        PORT(
            pXX             : IN    STD_LOGIC_VECTOR(4*gN-2 DOWNTO 0);
            pX1, pX2        : OUT   STD_LOGIC_VECTOR(gN-1 DOWNTO 0);
            pX3             : OUT   STD_LOGIC_VECTOR(gN DOWNTO 0);
            pX4             : OUT   STD_LOGIC_VECTOR(gN-2 DOWNTO 0));
END ntyFWC_cb4m1_20;

ARCHITECTURE rtlFWC_cb4m1_20 of ntyFWC_cb4m1_20 IS

COMPONENT ntyFWC_cb4m1_X1_20
          PORT(
            pXX             : IN    STD_LOGIC_VECTOR(4*gN-2 DOWNTO 0);
            pX1             : OUT   STD_LOGIC_VECTOR(gN-1 DOWNTO 0));
END COMPONENT;

COMPONENT ntyFWC_cb4m1_X3_20
          PORT(
            pXX             : IN    STD_LOGIC_VECTOR(4*gN-2 DOWNTO 0);
            pX3             : OUT   STD_LOGIC_VECTOR(gN DOWNTO 0));
END COMPONENT;

COMPONENT ntyFWC_cb4m1_X4_20
          PORT(
            pXX             : IN    STD_LOGIC_VECTOR(4*gN-2DOWNTO 0);
            pX4             : OUT   STD_LOGIC_VECTOR(gN-2 DOWNTO 0));
END COMPONENT;

BEGIN

        lblX1   :       ntyFWC_cb4m1_X1_20 port map(pXX, pX1);

        lblX3   :       ntyFWC_cb4m1_X3_20 port map(pXX, pX3);

        lblX4   :       ntyFWC_cb4m1_X4_20 port map(pXX, pX4);

        pX2 <= pXX(gN-1 downto 0);

END rtlFWC_cb4m1_20;
```

For residue channel $2^n - 1$: ntyFWC_cb4m1_X1_20.vhd

```
-- Forward converter for four-moduli set {2^n - 1, 2^n, 2^n + 1, 2^(n-1) - 1},
-- n= 20
-- X1 module

Library IEEE;
```

```
USE ieee.std_logic_1164.all;

ENTITY ntyFWC_cb4m1_X1_20 IS
      PORT(
         pXX              : IN     STD_LOGIC_VECTOR(78 DOWNTO 0);
         pX1              : OUT    STD_LOGIC_VECTOR(19 DOWNTO 0));
END ntyFWC_cb4m1_X1_20;

ARCHITECTURE rtlFWC_cb4m1_X1_20 of ntyFWC_cb4m1_X1_20 IS

COMPONENT ntyCSAeac_N
      GENERIC(gN : INTEGER := 4);
        PORT(
            pA, pB, pC      :  in   std_logic_vector(gN-1 downto 0);
            pCout, pSum     :  out  std_logic_vector(gN-1 downto 0));
END COMPONENT;

COMPONENT ntyCLA_L2M20
        PORT(
            pA, pB          : IN    std_logic_vector(19 downto 0);
            pSum            : OUT   std_logic_vector(19 downto 0));
END component;

constant cN : integer := 20;

signal sCtp1,sStp1,sCtF1,sCtp2,sStp2    : STD_LOGIC_VECTOR(cN-1 DOWNTO 0);
signal sInput4,sCtp3,sStp3,sCtF2,sCtF3 : STD_LOGIC_VECTOR(cN-1 DOWNTO 0);


BEGIN

lblCSAeacN1:    ntyCSAeac_N generic map(cN)
                port map(pXX(cN-1 downto 0),pXX(2*cN-1 downto cN),pXX(3*cN-1 downto 2*cN),sCtp1,sStp1);
                CtF1 <= sCtp1(cN-2 downto 0) & sCtp1(cN-1);          --left circular shift 1 bit

                sInput4(cN-1) <= '0';
                sInput4(cN-2 downto 0) <= pXX(4*cN-2 downto 3*cN);

lblCSAeacN2:    ntyCSAeac_N generic map(cN)
                port map(sCtF1,sStp1,sInput4,sCtp2,sStp2);
                sCtF2 <= sCtp2(cN-2 downto 0) & sCtp2(cN-1);         --left circular shift 1 bit

lblCPAeac:      ntyCLA_L2M20 port map(sCtF2, sStp2,pX1);


END rtlFWC_cb4m1_X1_20;
```

Test bench: tbFWC_cb4m1_20.vhd

```
ENTITY tbFWC_cb4m1_20 IS
END tbFWC_cb4m1_20;

Library IEEE;
USE std.textio.all;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_textio.all;
USE work.pkgRNS_sim.all;

ARCHITECTURE arctbFWC_cb4m1_20 OF tbFWC_cb4m1_20 IS

COMPONENT ntyFWC_cb4m1_20
      GENERIC(gN : integer := 4 );
      PORT(
         pXX              : IN     STD_LOGIC_VECTOR(4*gN-2 DOWNTO 0);
         pX1, pX2         : OUT    STD_LOGIC_VECTOR(gN-1 DOWNTO 0);
         pX3              : OUT    STD_LOGIC_VECTOR(gN DOWNTO 0);
         pX4              : OUT    STD_LOGIC_VECTOR(gN-2 DOWNTO 0));
END COMPONENT;

CONSTANT cN : integer := 20;

SIGNAL sXX              : std_logic_vector(4*cN-2 downto 0);
SIGNAL sX1,sX2          : std_logic_vector(cN-1 downto 0);
SIGNAL sX3              : std_logic_vector(cN downto 0);
```

```
SIGNAL sX4              : std_logic_vector(cN-2 downto 0);

SIGNAL X1,X2            : std_logic_vector(cN-1 downto 0);
SIGNAL X3               : std_logic_vector(cN downto 0);
SIGNAL X4               : std_logic_vector(cN-2 downto 0);


FILE  fInfile: text IS IN "D:\research\project\lib_cb\ForwardConverter\TB\tv_FWC_cb4m1_20.txt";

CONSTANT cClk_cyc       : TIME := 100 ns;

BEGIN
    -- instantiate the component
    lblFWC_cb4: ntyFWC_cb4m1_20 generic map(cN)
                port map(sXX,sX1,sX2,sX3,sX4);

    lbltest: process
        variable vLBuff             : LINE;
        variable vstrX              : string(4*cN-1 downto 1);
        variable vstrX1,vstrX2      : string(cN downto 1);
        variable vstrX3             : string(cN+1 downto 1);
        variable vstrX4             : string(cN-1 downto 1);
        variable vFound_Error       : boolean := false;
        --variable vI : std_logic_vector(5 downto 0);

    BEGIN
            while (not endfile(fInfile)) loop
            readline(fInfile, vLBuff);              -- read in one line to vLBuff
            read(vLBuff, vstrX);                    -- read in X
            readline(fInfile, vLBuff);
            read(vLBuff, vstrX1);                   -- read in X1
            readline(fInfile, vLBuff);
            read(vLBuff, vstrX2);                   -- read in X2
            readline(fInfile, vLBuff);
            read(vLBuff, vstrX3);                   -- read in X3
            readline(fInfile, vLBuff);
            read(vLBuff, vstrX4);                   -- read in X4

            sXX <= fnStr2Vec(vstrX);                -- Add stimulus

            X1 <= fnStr2Vec(vstrX1);
            X2 <= fnStr2Vec(vstrX2);
            X3 <= fnStr2Vec(vstrX3);
            X4 <= fnStr2Vec(vstrX4);

            -- wait for the outputs to settle
            wait for cClk_cyc;

            -- check the results
            if (sX1 /= X1) then
            --vI := conv_std_logic_vector(i, 6);
            assert false
                    report "Result X1 is" & fnTo_string(sX1)
                            & ". Expected " & fnTo_string(X1);
            --                & ". i is" & fnTo_string(vI);
                    vFound_Error := true;
            end if;

            if (sX2 /= X2) then
            --vI := conv_std_logic_vector(i, 6);
            assert false
                    report "Result X2 is" & fnTo_string(sX2)
                            & ". Expected " & fnTo_string(X2);
            --                & ". i is" & fnTo_string(vI);
                    vFound_Error := true;
            end if;

            if (sX3 /= X3) then
            --vI := conv_std_logic_vector(i, 6);
            assert false
                    report "Result X3 is" & fnTo_string(sX3)
                            & ". Expected " & fnTo_string(X3);
            --                & ". i is" & fnTo_string(vI);
                    vFound_Error := true;
            end if;

            if (sX4 /= X4) then
            --vI := conv_std_logic_vector(i, 6);
```

```
            assert false
                    report "Result X4 is" & fnTo_string(sX4)
                            & ". Expected " & fnTo_string(X4);
            --              & ". i is" & fnTo_string(vI);
                    vFound_Error := true;
            end if;

        end loop;

       assert not vFound_Error
          report "There were ERRORS in the test."
          severity note;
        assert vFound_Error
          report "Test completed with no errors."
          severity note;
      wait;

   END process;

END arctbFWC_cb4m1_20;
```

Part of test vectors: tv_FWC_cb4m1_20.txt

```
01101111010010101011001101100110011001110011110100011001101110011101000111100000
00111111111101000111
11001110100011110000
01001001110010000000 1
01101011010000010 10
01110110011100110011100101011010011110101110000111000011110110111011000011111111
01010100011101110101
11011101100001111111
01111110001000000 0101
10001000110100001 00
01011111001011001000000000011010010000100011111010000110011110001010100100001 10
00010100111100111111
11000101010010000110
00111011101000001 0000
00000111100100101 01
01111101111100000110011001001011101001000000010000101101111011101010001011100000
11101000100000110100
01110101000101110000
00110011001100101 0000
01110100111101101 00
11001010011110110110100011000110100101100010011111001110100111011010010000010 10
10101100011110100101
11101101001000001010
01011101000110000011 0
10101111011000000 00
....
```

A.3 Scripts and report examples

1. Setup .synopsys.dc.setup

The file .synopsys_dc.setup file is a necessary file for running Design Compiler. The contents of this file stipulate the environment for running Design Compiler.

```
cell_lib_path = { \
  /avanti/passport/2000.3/cb35/v2.6/synopsys/1999.05/models/ \
  /avanti/passport/2000.3/cb35/v2.6/synopsys/1999.05/icons/ \
}

search_path = {.} + cell_lib_path

target_library =  {cb35os142_typ.db   \
                   cb35io132_typ.db   \
                   cb35io122_typ.db   \
                  };

link_library = { "*" };

symbol_library = {   \
  cb35os142.sdb   \
  cb35io132.sdb   \
  cb35io122.sdb   \
};

define_design_lib work -path ./SynWORK;


company = "NTU-CHiPES" ;
designer = "CaoBin";
view_background = "black";
```

2. Constraint file

Design rule constraints reflect technology-specific restrictions that design must meet in order to function as intended. When Design Compiler optimizes the design, it uses two types of constraints: Design rule constraints and Optimization constraints. Design rule constraints are implicit constraints; the technology library defines them. Optimization constraints are explicit constrains and represent the design's goal. One of the optimization constraint file is shown as follows.

```
set AREA_GOAL 0
set TOTAL_DELAY 0
set OP_CONDS typical
set DRIVE_CELL typical/BUFX3
set DRIVE_PIN {Y}
set CLK_TO_Q 1
set OUTPUT_LOAD [expr [load_of typical/BUFX3/A] * 4]

set_operating_conditions $OP_CONDS -lib typical
```

```
set auto_wire_load_selection false
set_wire_load_model -name " ForQA"
set_wire_load_mode top

set_load 0.1 [all_outputs]
set_max_fanout 30 [all_inputs]
set_max_fanout 30 ${TOP_DESIGN}

# setting area constraints
set_max_area $AREA_GOAL

# setting delay constraints
set_max_delay –from pXX –to pX3 $ TOTAL _DELA
```

3. Report example

The following is the synthesis report for the forward converter for the moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1\}$ *when* $n = 20$.

```
****************************************
Report : area
Design : ntyFWC_cb4m1_20_gN20
Version: 2001.08-SP1
Date   : Tue May 25 17:27:18 2004
****************************************

Library(s) Used:

    cb35os142_typ (File: /avanti/passport/2000.3/cb35/v2.6/synopsys/1999.05/models/cb35os142_typ.db)

Number of ports:              159
Number of nets:               139
Number of cells:                3
Number of references:           3

Combinational area:       2309.239746
Noncombinational area:       0.000000
Net Interconnect area:     936.772156

Total cell area:          2309.239746
Total area:               3246.011963
1
design_analyzer> Warning: In design 'ntyFWC_cb4m1_20_gN20', there are 20 feedthroughs. (LINT-30)
Warning: In design 'ntyFWC_cb4m1_X1_20', there is 1 submodule connected to power or ground. (LINT-30)
Warning: In design 'ntyFWC_cb4m1_X3_20', there are 3 submodules connected to power or ground. (LINT-30)
Warning: In design 'ntyFWC_cb4m1_X3_20', there are 2 submodules with pins connected to the same net. (LINT-30)
Warning: In design 'ntyMA_2np1_gN20', there are 3 submodules connected to power or ground. (LINT-30)
Warning: In design 'ntyCPG_2np1_gN20', there is 1 port not connected to any nets. (LINT-30)
Warning: In design 'ntyCPG_2np1_gN20', there are 2 feedthroughs. (LINT-30)
Warning: In design 'ntyCout_2np1_gN20', there are 3 ports not connected to any nets. (LINT-30)
Warning: In design 'ntyCLAS_2np1_gN20', there are 2 ports not connected to any nets. (LINT-30)
Warning: In design 'ntyFWC_cb4m1_X4_20', there is 1 submodule connected to power or ground. (LINT-30)
Warning: In design 'ntyFWC_cb4m1_X4_20', there is 1 submodule with pins connected to the same net. (LINT-30)

Information: Use the 'check_design' command for more information about warnings. (LINT-99)


****************************************
Report : power
        -analysis_effort low
Design : ntyFWC_cb4m1_20_gN20
Version: 2001.08-SP1
Date   : Tue May 25 17:27:20 2004
****************************************

Library(s) Used:

    cb35os142_typ (File: /avanti/passport/2000.3/cb35/v2.6/synopsys/1999.05/models/cb35os142_typ.db)
```

```
Global Operating Voltage = 3.3
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  18.3435 mW   (74%)
  Net Switching Power  =   6.5351 mW   (26%)
                         ---------
Total Dynamic Power    =  24.8787 mW   (100%)

Cell Leakage Power     = 119.4558 nW

1
design_analyzer>
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : ntyFWC_cb4m1_20_gN20
Version: 2001.08-SP1
Date   : Tue May 25 17:27:20 2004
****************************************

Operating Conditions: NCCOM Library: cb35os142_typ
Wire Load Model Mode: enclosed

  Startpoint: pXX[25] (input port)
  Endpoint: pX3[19] (output port)
  Path Group: default
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  ntyFWC_cb4m1_20_gN20
                     8000                  cb35os142_typ
  ntyMA_2np1_gN20    8000                  cb35os142_typ
  ntySAC_2np1_gN20   ForQA                 cb35os142_typ
  ntyFWC_cb4m1_X3_20 8000                  cb35os142_typ
  ntyCout_2np1_gN20  ForQA                 cb35os142_typ
  ntyCLAS_2np1_gN20  8000                  cb35os142_typ

  Point                                                   Incr       Path
  ------------------------------------------------------------------------
  input external delay                                    0.00       0.00 f
  pXX[25] (in)                                            0.00       0.00 f
  lblX3/pXX[25] (ntyFWC_cb4m1_X3_20)                      0.00       0.00 f
  lblX3/U175/ZN (inv0d1)                                  0.11       0.11 r
  lblX3/lblCSAeacN1/pB[5] (ntyCSAeac_N_gN20_0)            0.00       0.11 r
  lblX3/lblCSAeacN1/lblFA_1_5/pB (ntyFA_1_66)             0.00       0.11 r
  lblX3/lblCSAeacN1/lblFA_1_5/U16/Z (xr03d1)              0.63       0.74 r
  lblX3/lblCSAeacN1/lblFA_1_5/pSum (ntyFA_1_66)           0.00       0.74 r
  lblX3/lblCSAeacN1/pSum[5] (ntyCSAeac_N_gN20_0)          0.00       0.74 r
  lblX3/lblCSAeacN2/pB[5] (ntyCSAeac_N_gN20_2)            0.00       0.74 r
  lblX3/lblCSAeacN2/lblFA_1_5/pB (ntyFA_1_106)            0.00       0.74 r
  lblX3/lblCSAeacN2/lblFA_1_5/U18/Z (xr03d1)              0.64       1.37 r
  lblX3/lblCSAeacN2/lblFA_1_5/pSum (ntyFA_1_106)          0.00       1.37 r
  lblX3/lblCSAeacN2/pSum[5] (ntyCSAeac_N_gN20_2)          0.00       1.37 r
  lblX3/lblCSAeacN3/pB[5] (ntyCSAeac_N_gN20_1)            0.00       1.37 r
  lblX3/lblCSAeacN3/lblFA_1_5/pB (ntyFA_1_86)             0.00       1.37 r
  lblX3/lblCSAeacN3/lblFA_1_5/U18/Z (xr03d1)              0.44       1.82 r
  lblX3/lblCSAeacN3/lblFA_1_5/pSum (ntyFA_1_86)           0.00       1.82 r
  lblX3/lblCSAeacN3/pSum[5] (ntyCSAeac_N_gN20_1)          0.00       1.82 r
  lblX3/lblCPAeac/py[5] (ntyMA_2np1_gN20)                 0.00       1.82 r
  lblX3/lblCPAeac/lblSAC/pY[5] (ntySAC_2np1_gN20)         0.00       1.82 r
  lblX3/lblCPAeac/lblSAC/U271/Z (or02d1)                  0.23       2.05 r
  lblX3/lblCPAeac/lblSAC/U273/ZN (inv0d1)                 0.05       2.10 f
  lblX3/lblCPAeac/lblSAC/U270/Z (aor21d1)                 0.25       2.35 f
  lblX3/lblCPAeac/lblSAC/pA[5] (ntySAC_2np1_gN20)         0.00       2.35 f
  lblX3/lblCPAeac/lblCPG/pA[5] (ntyCPG_2np1_gN20)         0.00       2.35 f
  lblX3/lblCPAeac/lblCPG/U284/Z (xr02d1)                  0.46       2.81 r
  lblX3/lblCPAeac/lblCPG/pP[5] (ntyCPG_2np1_gN20)         0.00       2.81 r
  lblX3/lblCPAeac/lblCout/pP[5] (ntyCout_2np1_gN20)       0.00       2.81 r
```

lblX3/lblCPAeac/lblCout/U143/Z (ora211d1)	0.30	3.12 r
lblX3/lblCPAeac/lblCout/U142/Z (aor22d1)	0.28	3.39 r
lblX3/lblCPAeac/lblCout/U138/ZN (nr03d1)	0.07	3.47 f
lblX3/lblCPAeac/lblCout/U136/ZN (aoi322d1)	0.47	3.94 r
lblX3/lblCPAeac/lblCout/U130/ZN (nd03d2)	0.07	4.01 f
lblX3/lblCPAeac/lblCout/U131/ZN (inv0d2)	0.11	4.11 r
lblX3/lblCPAeac/lblCout/pCout (ntyCout_2np1_gN20)	0.00	4.11 r
lblX3/lblCPAeac/lblCLAS/pCo (ntyCLAS_2np1_gN20)	0.00	4.11 r
lblX3/lblCPAeac/lblCLAS/U476/ZN (inv0d2)	0.08	4.19 f
lblX3/lblCPAeac/lblCLAS/U503/ZN (aoi221d1)	0.39	4.58 r
lblX3/lblCPAeac/lblCLAS/U500/ZN (oai21d1)	0.14	4.71 f
lblX3/lblCPAeac/lblCLAS/U489/ZN (oaim31d1)	0.28	4.99 f
lblX3/lblCPAeac/lblCLAS/U483/Z (aor21d1)	0.27	5.26 f
lblX3/lblCPAeac/lblCLAS/U482/Z (aor21d1)	0.27	5.53 f
lblX3/lblCPAeac/lblCLAS/U454/Z (xr02d1)	0.36	5.89 r
lblX3/lblCPAeac/lblCLAS/pR[19] (ntyCLAS_2np1_gN20)	0.00	5.89 r
lblX3/lblCPAeac/pR[19] (ntyMA_2np1_gN20)	0.00	5.89 r
lblX3/pX3[19] (ntyFWC_cb4m1_X3_20)	0.00	5.89 r
pX3[19] (out)	0.00	5.89 r
data arrival time	5.89	
max_delay	5.90	5.90
output external delay	0.00	5.90
data required time		5.90
---	---	---
data required time		5.90
data arrival time		-5.89
---	---	---
slack (MET)		0.01

1
design_analyzer>

Author's Publications

Journal Papers

[Cao03a] 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 Trans. Circuits and Syst. I*, vol. 50, no. 10, pp. 1296-1303, October 2003. (Regular Paper).

[Cao05a] B. Cao, T. Srikanthan, and C. H. Chang, "Efficient reverse converters for the 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 Proc. Computers & Digital Techniques,* vol. 152, no. 5, pp. 687-696, September 2005. (Regular Paper).

Conference Papers

[Cao03b] B. Cao, T. Srikanthan, and C. H. Chang, "Design of a high speed reverse converter for a new 4-moduli set residue number system," *Proc. ISCAS'03*, Bangkok, Thailand, vol. 4, pp. 520-523, May 2003.

[Cao03c] B. Cao, C. H. Chang, and T. Srikanthan, "New efficient residue-to-binary converters for 4-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$," *Proc. ISCAS'03*, Bangkok, Thailand, vol. 4, pp. 536-539, May 2003.

[Cao03d] B. Cao, C. H. Chang, and T. Srikanthan, "Adder based residue to binary converters for a new balanced 4-moduli set," *Proc. 3th Int. Symp. on Image and Signal Processing and Analysis*, Rome, Italy, pp. 820-825, September 2003.

[Cao04] B. Cao, T. Srikanthan, and C. H. Chang, "Design of residue-to-binary converter for a new 5-moduli superset residue number system," *Proc. ISCAS'04*, Vancouver, Canada, vol. 2, pp. 841-844, May 2004.

[Cao05b] B. Cao, T. Srikanthan, and C. H. Chang, "A new design method to modulo $2^n - 1$ squaring," *Proc. ISCAS'05*, Kobe, Japan, vol. 1, pp. 664-667, May 2005.

[Cao05c] B. Cao, C. H. Chang and T. Srikanthan, "A new formulation of fast diminished-one multioperand modulo $2^n + 1$ adder," *Proc. ISCAS'05*, Kobe, Japan, vol. 1, pp. 656-659, May 2005.

[Men05] S. Menon, C. H. Chang, B. Cao, and T. Srikanthan, "A configurable dual moduli multi-operand modulo adder," *Proc. ISCAS'05*, Kobe, Japan, vol. 2, pp. 23-26, May 2005.

References

[Abd99] M. Abdallah and A. Skavantzos, "The multi polynomial channel polynomial residue arithmetic system," *IEEE Trans. Circuits Syst. II,* vol. 46, no. 2, pp. 165-171, February 1999.

[Alia84] G. Alia and E. Martinelli, "A VLSI algorithm for direct and reverse conversion from weighted binary number system to residue number system," *IEEE Trans. Circuits and Syst.*, vol. CAS-31, no. 12, pp. 1033-1039, December 1984.

[Alia91] G. Alia and E. Martinelli, "A VLSI modulo *m* multiplier," *IEEE Trans. Comput.,* vol. 40, no. 7, pp. 873-878, July 1991.

[Alia96] G. Alia and E. Martinelli, "Design multioperand modular adders," *Electronics Lett.*, vol. 32, no. 1, pp. 22-23, 4th January 1996.

[And88] S. Andraos and H. Ahmad, "A new efficient memoryless residue to binary converter," *IEEE Trans. Circuits Syst.*, vol. 35, no. 11, pp. 1441-1444, November 1988.

[Baja98] J. –C. Bajard, L. -S. Dider, and P. Kornerup, "An RNS Montgomery modular multiplication algorithm," *IEEE Trans. Comput.,* vol. 47, no. 7, pp. 766-776, July 1998.

[Baja01] J. –C. Bajard, L. -S. Dider, and P. Kornerup, "Modular multiplication and base extension in residue number system," *Proc. 15th IEEE symposium on Computer Arithmetic*, Vail, Colorado, pp. 56-63, June 2001.

[Ban74] D. K. Banerji, "A novel implementation method for addition and subtraction in residue number systems," *IEEE Trans. Comput.*, vol. C-23, no. 1, pp. 106-109, January 1974.

[Bar80] A. Z. Barniecka and G. A. Jullien, "Residue number system implementations of theoretic transforms in complex residue rings," *IEEE Trans. ASSP*, vol. 28, pp. 285-192, 1980.

[Bar94] F. Barsi and M. C. Pinotti, "A fully parallel algorithm for residue to binary conversion," *Inform. Proc. Lett.*, vol. 50(1), pp. 1-8, April 1994.

[Bay87] M. Bayoumi and G. Jullien, "A VLSI implementation of residue adders," *IEEE Trans. Circuits and Syst.*, vol. 34, no. 3, pp 284-288, March 1987.

[Ben88a] M. Banaissa, A. Pajayakrit, S. S. Dlay and A. G. J. Holt, "VLSI design for diminished-1 multiplication of integers modulo a Fermat number," *IEE Proc., vol. 135, Pt. E*, no. 3, pp. 161-164, May 1988.

[Ben88b] M. Banaissa, A. Bouridane, S. S. Dlay and A. G. J. Holt, "Diminished-1 multiplier for a fast convolver and correlators using the Fermat number transform," *IEE Proc., vol. 135, Pt. G*, no. 5, pp. 187-193, October 1988.

[Ber85] P. Bernardson, "Fast memoryless, over 64 bits, residue to decimal converter," *IEEE Trans. Circuits Syst.*, vol. 32, no. 3, pp. 298-300, March 1985.

[Bha98] 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 Trans. Circuits Syst. I,* vol. 45, no. 9, pp. 998-1002, September 1998.

[Bha99] M. Bhardwaj, T. Srikanthan and C. T. Clarke, "A reverse converter for the 4-moduli superset $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$," *Proc. of 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, pp. 168-175, April 1999.

[Bha99b] M. Bhardwaj, T. Srikanthan, and C. T. Clarke, "VLSI costs of arithmetic parallelism: a residue reverse conversion perspective," *Proc.of 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, pp. 176-184, April 1999.

[Bi88] G. Bi and E. V. Jones, "Fast conversion between binary and residue numbers," *Electronic Lett.*, vol. 24, no. 19, pp. 1195-1197, 15th September 1988.

[Bosi99] B. Bosi, G. Bosi and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 7, no. 3, pp. 299-308, September 1999.

[Bren82] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 260-264, March 1982.

[Capo88] R. M. Capocell, and R. Giancarlo, "Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1425-1430, 1988.

[Card98] G. C. Cardarilli, M. Re and R. Lojacono, "RNS-to-binary conversion for efficient VLSI implementation," *IEEE Trans. Circuits Syst. I*, vol. 45, no. 6, pp. 667-669, June 1998.

[Card00] G. C. Cardiralli, M. Re. R. Lojacono and G. Ferri, "A systolic architecture for high-performance scaled residue to binary conversion," *IEEE Trans. Circuits Syst. I*, vol. 47, no. 10, pp. 1523-1526, October 2000.

[Chak86] N. B. Chakraborti, J. S. Soundararajan and A. L. N. Reddy, "An implementation of mixed-radix conversion for residue number applications," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 762-764, August 1986.

[Chan85] J. J. Chang, T. K, Truong, H. M. Shao, I. S. Reed and I. -S, Hsu, "The VLSI design of a single chip for the multiplication of integers modulo a Fermat number," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, no. 6, pp. 1599-1602, December 1985.

[Clau95] E. D. D. Claudio, F. Piazza, and G. Orlandi, "Fast combinatorial RNS processors for DSP applications," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 624-633, May 1995.

[Con99] R. Conway and J. Nelson, "Fast converter for 3 moduli RNS using new property of CRT," *IEEE Trans. Comput.*, vol. 48, no.8, pp. 852-860, August 1999.

[Con03] R. Conway and J. Nelson, "New CRT-based RNS converter using restricted moduli set," *IEEE Trans. Comput.*, vol. 52, no.5, pp. 572-578, May 2003.

[Con04] R. Conway and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Trans. Circuits and Syst.-II*, vol. 51, no. 1, pp. 26-28, January 2004.

[Curi91] A. V. Curiger, H. Bonnenberg, and H. Kaeslin, "Regular VLSI architectures for multiplication modulo (2^n+1)," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, pp. 990-994, July 1991.

[Dhur98] A. Dhurkadas, "Comments on 'a high speed realization of a residue to binary number system converter'," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 3, pp. 446-447, March 1998.

[Dim03] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos, and C. Efstathiou, "A family of parallel-prefix modulo 2^n – 1 adders," *Proc. IEEE Int'l Conf. on*

Application-Specific Systems, Architectures, and Processors, Hague, Netherlands, pp. 326-336, June 2003.

[Dug94] M. Dugdale, "Residue multipliers using factored decomposition," *IEEE Trans. Circuits and Syst. I,* vol. 41, no. 9, pp. 623-627, September 1994.

[Dug92] M. Dugdale, "VLSI implementation of residue adders based on binary adders," *IEEE Trans. Circuits and Syst., II*, vol. 39. pp. 325-329, May 1992.

[Efst94] C. Efstathiou, D. Nikolos and J. Kalamatianos, "Area-time efficient modulo $2^n - 1$ adder design," *IEEE Trans. Circuits Syst. II*, vol. 41, no. 7, pp. 463-467, July 1994.

[Efst03] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Modulo $2^n \pm 1$ adder design using select-prefix blocks," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1399-1406, November 2003.

[Efst05] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos, and D. Nikolos, "Efficient diminished-1 modulo $2^n + 1$ multipliers," *IEEE Tran. Comput.*, vol. 54, no. 4, pp. 491-496, April 2005.

[Eld93] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Trans. Comput.,* vol. 42, no. 6, pp. 693-699, June 1993.

[Elle86] K. M. Elleithy, "On bit-parallel processing for modulo arithmetic," *VLSI Tech. Rep. TR86-8-1*, The Center for Advanced Computer Studies, Univ. Southwestern Louisiana, 1986.

[Elle89] K. M. Elleithy, M. A. Bayoumi and K. P. Lee, "$\theta(\log N)$ architectures for RNS arithmetic decoding," *Proc. 9th Symp. Comput. Arithmetic*, pp. 202-209, Santa Monica, CA, September 1989.

[Elle90] K. M. Elleithy and M. A. Bayoumi, "A $\theta(1)$ algorithm for modulo addition," *IEEE Trans. Circuits Syst.*, vol. 37. no. 5, pp. 628-631, May 1990.

[Elle92] K. M. Elleithy and M. A. Bayoumi, "Fast and flexible architectures for RNS arithmetic decoding," *IEEE Trans. Circuits Syst. II*, vol. 39, no.4, pp. 226-235, April 1992.

[Elle95] K. M. Elleithy and M. A. Bayoumi, "A systolic architecture for modulo multiplication," *IEEE Trans. Circuits and Syst. II*, vol. 42, no. 11, pp. 725-729, November 1995.

[Gall97] D. Gallaher, F. E. Petry, and P. Srinivasan, "The digit parallel method for fast RNS to weighted number system conversion for special moduli ($2^n - 1$, 2^n, $2^n + 1$)," *IEEE Trans. Circuits and Syst. II*, vol. 44, no. 1, pp. 53-57, January 1997.

[Gar03] A.Garg, I. Steiner, G. A. Jullien, J. W. Haslett, and G. H. McGibney, "A high speed complex adaptive filter for an asymmetric wireless LAN using a new quantized polynomial representation," in *Proc. IEEE Int. Symp. on Circuits and Syst.*, Bangkok, Thailand, vol. II, pp. 157-160, May 2003.

[Gros66] E. Grosswald, *Topics from the theory of numbers*. New York: McMillan, 1966.

[Hia92] A. Hiasat, "New memoryless, mod (2^n+1) residue multiplier," *Electronics Lett.*, vol. 28, no. 3, pp. 314-315, January 1992.

[Hia98] 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 Trans. Circuits Syst. II*, vol. 45, no. 2, pp. 204-209, February 1998.

[Hia00] A. A. Hiasat, "New efficient structure of a modular multiplier for RNS," *IEEE Trans. Comput.*, vol. 49, no. 2, pp. 170-174, February 2000.

[Hia02] A. A. Hiasat, "High-speed and reduced-area modular adder structures for RNS," *IEEE Trans. Computers*, vol. 51, no. 1, pp. 84-89, January 2002.

[Hia03] A. Hiasat, "Efficient residue to binary converter," *IEE Proc. -Comput. Digit. Tech.*, vol. 150, no. 1, pp. 11-16, January 2003.

[Huan81] C. H. Huang, D. G. Peterson, H. A. Rauch, J. W. Teague and D. F. Fraser, "Implementation of a fast digital processor using the residue number system," *IEEE Trans. Circuits Syst.*, vol. 28, pp. 32-37, January 1981.

[Huan83] C. H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications," *IEEE Trans. Comput.*, vol. C-32, no. 4, pp. 398-402, April 1983.

[Hwan79] K. Hwang, *Computer Arithmetic: Principle, Architecture and Design*. New York: Wiley, 1979.

[Ibra88] K. M. Ibrahim and S. N. Saloum, "An efficient residue to binary converter design," *IEEE Trans. Circuits Syst.*, vol. 35, no. 9, pp. 1156-1158, September 1988.

[Imb02] L. Imbert and J. -J. Barjard, "A full RNS implementation of RSA," Research Report 02068, available at http://pckaty.lirmm.fr/GEIDEFile/RR-02068.PDF?Archive=191917991919&File=RR%2D02068_PDF, May 2002.

[Imb03] L. Imbert, V. S. Dimitrov, and G. A. Jullien, "Fault-tolerant computations over replicated finite rings," *IEEE Trans. Circuits Syst.-I*, vol. 50, no. 7, pp. 858-864, July 2003.

[Jen77] W. K. Jenkins and B. J. Leon, "The use of residue number system in the design of finite impulse response filters," *IEEE Trans. Circuits Syst.*, vol. 24, pp. 191-201, April 1977.

[Jen78] W. K. Jenkins, "A highly efficient residue-combinatorial architecture for digital filters," *Proc. IEEE*, vol. 66, pp. 700-702, June 1978.

[Jull78] G. A. Jullien, "Residue number scaling and other operations using ROM arrays," *IEEE Trans. Comput.*, vol. 27, no. 4, pp. 325-336, April 1978.

[Jull80] G. A. Jullien, "Implementation of multiplication, modulo a prime number, with applications to Number Theorem Transforms," *IEEE Trans. Comput.*, vol. 29, no. 10, pp. 899-905, October 1980.

[Jull90] G. A. Jullien and W. C. Miller, "Improved cellular structures for bit-steered ROM finite ring systolic arrays," *Proc. 1990 IEEE Int. Symp. on Circuits and Syst.*, pp. 1414-1417, New Orleans, USA, May 1990.

[Kal01] K. Kaluri, W. F. Leong, K. Tan, L. Johnson and M. Soderstrand, "FPGA hardware implementation of an RNS FIR digital filter," *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1340-1344, 2001.

[Kim91] J. Y, Kim, K. H. Park and H. S. Lee, "Efficient residue to binary conversion technique with rounding error compensation," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 315-317, 1991.

[Koc90] C. K. Koc and C. Y. Hung, "Multi-operand modulo addition using carry save adders," *Electronics Lett.*, vol. 26, no. 6, pp. 361-363, 15th March 1990.

[Kri92] H. Krishna, K. Lin and J. Sun, "A coding theory approach to error control in redundant residue number systems - part I: theory and single error correction," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 8-17, January 1992.

[Lee89] K. P. Lee, M. A. Bayoumi and K. M. Elleithy, "A fast and flexible residue decoder based on the Chinese Remainder Theorem," *Proc. 1989 IEEE Int. Symp. on Circuits and Syst.*, pp. 200-203, Portland, OR, USA, May 1989.

[Leib76] L. M. Leibowitz, "A simplified binary arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 24, no. 5, pp. 356-359, October 1976.

[Li91] Z. Li, R. Krishnan and R.J. Marks, "Modulized RNS-decimal number conversion algorithm and its implementations," *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 319-322, May 1991.

[Ma98] Y. Ma, "A simplified architecture for modulo $(2^n + 1)$ multiplication," *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 333-337, March 1998.

[Man72] D. Mandelbaum, "Error correction in residue arithmetic," *IEEE Trans. Comput.*, vol. 21, no. 6, pp. 538-545, June 1972.

[Math00] J. Mathew, D. Radhakrishnan and T. Srikanthan, "Fast residue-to-binary converter architectures," *Proc. 42nd Midwest Symp. on Circuits and Systems*, Las Cruces, New Mexico, vol.2, pp. 1090-1093, August 1999.

[Mead79] C. Mead and L. Conway, *Introduction to VLSI Systems.* Addison-Wesley Longman Publishing, Boston, MA, 1979.

[Mee90] S. J. Meehan, S. D. O'Neil and J. J. Vaccaro, "An universal input and output RNS converter," *IEEE Trans. Circuits Syst.*, vol. 37, no. 6, pp. 799-803, June 1990.

[Mell96] J. D. Mellott, M. Lewis, F. Taylor and P. Coffield, "ASAP-A 2D DFT VLSI processor and architecture," *Proc. 1996 IEEE Int. Symp. on Circuits and Syst.*, Atlanta, GA, USA, vol. 2, pp. 261-264, May 1996.

[Mill84] D. D. Miller and J. N. Polky, "An implementation of the LMS algorithm in the residue number system," *IEEE Trans. Circuits Syst.*, vol. 31, pp. 452-461, May 1984.

[Mill98] D. F. Miller and W. S. McCormick, "An arithmetic free parallel mixed-radix conversion algorithm," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 1, pp. 158-162, January 1998.

[Mont85] P. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, pp. 519-522, April 1985.

[Nag96] C. Nagendra, M. J. Irwin, and R. M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 10, pp. 689-702, November 1996.

[Noz01] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery multiplication," C. Parr ed. *Cryptographic Hardware and Embedded Systems - CHES 2001*, pp. 364-376, Springer-Verlag, Berlin, Germany.

[Pali99] V. Paliouras and T. Stouraitis, "Multifunction architectures for RNS processors," *IEEE Trans Circuits Syst., II*, vol. 46, no. 8, pp. 1041-1054, August 1999.

[Pali01] V. Paliouras, K. Karagianni, and T. Stouraitis, "A low-complexity combinatorial RNS multiplier," *IEEE Trans. Circuits and Syst. II,* vol. 48, no. 7, pp. 675-683, July 2001.

[Parh94] B. Parhami and C.Y. Huang, "Optimal table lookup schemes for VLSI implementation of input/output conversions and other residue number operations," *Workshop on VLSI Signal Processing, VII*, La Jolla, CA, USA, pp. 470-182, 1994.

[Parh00] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* New York: Oxford University Press, 2000.

[Per94] S. Perumal and R. E. Siferd, "Pipelined 50 MHz CMOS ASIC for 32 bit binary to residue conversion and residue to binary conversion," *Proc. 7th IEEE Intl. ASIC Conf. and Exhibit*, Rochester, England, pp. 454-457, September 1994.

[Pie94a] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Trans. Comput.*, vol. 423, no. 1, pp. 68-77, January 1994.

[Pie94b] S. J. Piestrak, "Design of high-speed residue-to-binary number system converter based on Chinese Remainder Theorem," *Proc. Intl. Conf. Computer Design*, Boston, USA, pp. 508-511,October 1994.

[Pie95] S. J. Piestrak, "A high-speed realization of a residue to binary number system converter," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 10, pp. 661-663, October 1995.

[Pie02] S. J. Piestrak, "Design of squarers modular A with low-level pipelining," *IEEE Trans. Circuits and Syst. II,* vol. 49, no. 1, pp. 31-41, January 2002.

[Prem92] A. B. Premkumar, "An RNS to binary converter in $2n + 1$, $2n$, $2n - 1$ moduli set," *IEEE Trans. Circuits Syst. II*, vol. 39, no. 7, pp. 480-482, July 1992.

[Prem95] A. B. Premkumar, "An RNS to binary converter in a three moduli set with common factors," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 4, pp. 298-301, April 1995.

[Prem98] A. B. Premkumar, M. Bhardwaj and T. Srikanthan, "High-speed and low cost reverse converters for the $\{2n - 1, 2n, 2n + 1\}$ moduli set," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 7, pp. 903-908, July 1998.

[Prem02] A. B. Premkumar, "A formal framework for conversion from binary to residue numbers," *IEEE Trans Circuits Syst., II*, vol. 49, no. 2, pp. 135-144, February 2002.

[Pou94a] F. Pourbigharaz and H. M. Yassine, "Simple binary to residue transformation with respect to $2^m + 1$ moduli," *IEE Proc. Circuits Device Syst., -G*, vol. 141, no. 6, pp. 522-526, December 1994.

[Pou94b] F. Pourbigharaz and H. M. Yassine, "Modulo-free architecture for binary to residue transformation with respect to $\{2^m - 1, 2^m, 2^m + 1\}$ moduli set," *Proc. 1994 IEEE Int. Symp. Circuits and Syst.*, vol. 2, pp. 317-320, London, May 1994.

[Pou94c] F. Pourbigharaz and H. M. Yassine, "Intermediate signed-digit stage to perform residue to binary transformations based on CRT," *Proc. 1994 IEEE Int. Symp. Circuits and Syst.*, vol. 2, pp. 353-356, London, May 1994.

[Rad92] D. Radhakrishnan and Y. Yuan, "Novel approaches to the design of VLSI RNS multipliers," *IEEE Trans. Circuits and Syst., II*, vol. 39. no. 1, pp. 52-57, January 1992.

[Ram00a] J. Ramirez, A. Garcia, P. G. Fernandez, L. Parrilla and A. Lloris, "Analysis of RNS-FPL synergy for high throughput DSP applications: Discrete wavelet transform," *Proc. of 10th International Conference on Field-Programmable Logic and Applications (FPL 2000), Field-Programmable Logic: The Roadmap to Reconfigurable Computing*, Villach, Austria, pp. 342-351, August 2000.

[Ram00b] J. Ramirez, A. Garcia, P. G. Fernandez, L. Parrilla and A. Lloris, "A new architecture to compute the Discrete Cosine Transform using the quadratic residue number system," *Proc. 2000 IEEE Int. Symp. Circuits and Syst.*, vol.5, pp. 321-324, Geneva 2000.

[Rao94] P. B. Rao and A. Skavantzos, "ROM based methods for computing the squaring operation in modular rings," *J. VLSI Signal Processing*, vol. 7, no. 3, pp. 199-211, May 1994.

[Rej00] B. Rejeb, H. Henkelmann and W. Anheier, "Real-time implementation of fractal image encoder using residue number system," *10th Mediterranean Electrotechnical Conference*, Lemesos, Cyprus, vol. 2, pp. 612-615, May 2000.

[Rob93] N. Robbins, *Beginning Number Theory*. Dubuque, Iowa: Wm. C. Brown Publishers, 1993

[Ska89] A. Skavantzos, "Design of multioperand carry-save adders for arithmetic modulo $(2^n + 1)$," *Electronics Lett.*, 1989, 25, (17), pp. 1152-1153.

[Ska92] A. Skavantzos, "New multipliers modulo $2^N - 1$," *IEEE Trans. Comput.*, vol. 41, pp. 957-961, August 1992.

[Ska98] A. Skavantzos, "An efficient residue to weighted converter for a new residue number system," *Proc. of the 8th Great Lakes Symp. VLSI*, LA, USA, no. 9, pp. 185-191, February 1998.

[Ska99a] A. Skavantzos and M. Abdallah, "Implementation issues of the two-level residue number system with pairs of conjugate moduli," *IEEE Trans. Signal Processing*, vol. 47, no. 3, pp. 826-838, March 1999.

[Ska99b] A. Skavantzos and T. Stouraitis, "Grouped-moduli residue number systems for fast signal processing," *Proc. 1999 Int. Symp. Circuits and Syst.*, Orlando, FL, vol. 3, pp. 478-483, June 1999.

[Sod80] M. A. Soderstrand and C. Vernia, "A high-speed low-cost modulo P_i multiplier with RNS arithmetic applications," *Proc. IEEE*, vol. 68, pp. 529-532, April 1980.

[Sod83] M. Soderstrand, "A new hardware implementation of modulo adders for residue number systems," *Proc. IEEE 26th Midwest Symp. on Circuits and Systems*, Puebla, Mexico, pp. 412-415, August 1983.

[Sod86] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. J. Taylor (Eds), *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.

[Sri98] T. Srikanthan, M. Bhardwaj and C. T. Clarke, "Area-time-efficient VLSI residue-to-binary converters," *IEE Proc. -Comput. Digit. Tech.*, vol. 145, no. 3, pp. 229-235, May 1998.

[Stou93] T. Stouraitis, S. W. Kim and A. Skavantzos, "Full-adder based arithmetic units for finite integer rings," *IEEE Trans. Circuits and Syst., II*, vol. 40. pp. 740-745, November 1993.

[Sun93] S. Sunder et al., "Area-efficient diminished-1 multiplier for Fermat number-theoretic transform," *IEE Proc. -G*, vol. 135, pp. 211-215, 1993.

[Sza67] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw Hill, 1967.

[Tay81] F. J. Taylor, "Large modular multipliers for signal processing," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 731-736, July 1981.

[Tayl84] F. J. Taylor, "Residue arithmetic: a tutorial with examples," *IEEE. Comput.*, vol. 17, no. 5, pp. 50-63, May 1984.

[Tayl85] F. Taylor, "A single modulus complex ALU for signal processing," *IEEE Trans. Acoust., Speech and Signal Processing*, vol. 33, pp. 1302-1315, May 1985.

[Tayl90] F. J. Taylor, "An RNS Discrete Fourier Transform implementation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, no. 8, pp. 1386-1394, August 1990.

[Tya93] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. Comput.*, vol. 42, no.10, pp. 1162-1170, October 1993.

[Ver02] H. T. Vergos, C. Esftathiou, and D. Nikolos, "Diminished-one modulo $2^n + 1$ adder design," *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1389-1399, November 2002.

[Vin00] 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.

[Vu85] T. V. Vu, "Efficient implementations of the Chinese remainder theorems for sign detection and residue decoding," *IEEE Trans. Comput.*, vol. 34, no. 7, pp, 646-651, July 1985.

[Wal02] C. D. Walter. Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. In B. Preneel, editor, *Proceedings of Topics in Cryptology - CT-RSA 2002*, no. 2271 in Lecture Notes in Computer Science, pages 30 – 39, 2002.

[Wan95] Z. Wang, G. A. Jullien and W. C. Miller, "An algorithm for multiplication modulo $(2^n + 1)$," *Proc. 29th Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, vol. 2, pp. 956-960, November 1995.

[Wan96] Z. Wang, G. A. Jullien and W. C. Miller, "An algorithm for multiplication modulo $(2^n - 1)$," *IEEE 39th Midwest Symp. on Circuits and Systems*, vol. 3, pp. 1301-1304, August 1996.

[Wan00] Z. Wang, G. A. Jullien and W. C. Miller, "An improved residue-to-binary converter," *IEEE Trans. Circuits Syst. I*, vol. 47, no. 9, pp. 1437-1440, September 2000.

[Wang96] Y. Wang and M. Abd-El-Barr, "A new algorithm for RNS decoding," *IEEE Trans. Circuits Syst. I*, vol. 43, no. 12, pp. 998-1001, December 1996.

[Wang98] Y. Wang, "New Chinese Remainder Theorems," *Proc. 32th Asilomar Conf. Signals, Syst., Comput.*, Pacific. Grove, CA, vol. 1, pp. 165-171, 1998.

[Wang00] Y. Wang, "Residue-to-binary converters based on new Chinese Remainder Theorems," *IEEE Trans. Circuits Syst. II*, vol. 47, no. 3, pp. 197-205, March 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 Trans. Signal Processing*, vol. 50, no. 7, pp. 1772-1779, July 2002.

[Wrz93] A. Wrzyszcz and D. Milford, "A new modulo $2^a + 1$ multiplier," *Proc. 1993 Intl. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 614-617, Cambridge, MA, USA, 1993.

[Yas91] H. M. Yassine and W. R. Moore, "Improved mixed-radix conversion for residue number system architectures," *IEE Proc. -G*, vol. 138, no. 1, pp. 120-124, February 1991.

[Yen03] S. -M. Yen, S. Kim, S. Lim and S, -J. Moon, "RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis," *IEEE Trans. Comput.,* vol. 52, no.4, pp. 461-472, April 2003.

[Zimm94] R. Zimmermann *et al.*, "A 177Mb/s VLSI implementation of the international data encryption algorithm," *IEEE J. Solid-State Circuits*, vol. 29, no. 3, pp. 303-307, March 1994.

[Zimm98] R. Zimmerman, Binary adder architectures for cell-based VLSI and their synthesis, Ph.D thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hargung-Gorre Verlag, 1998.

[Zimm99] R. Zimmerman, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," *Proc. of 14th IEEE Symposium Computer Arithmetic,* Adelaide, Australia, pp. 158-167, April 1999.