New algorithms for hardware-efficient implementation of sign detection and magnitude comparison in residue number systems

Sachin Kumar

2017

Sachin Kumar. (2017). New algorithms for hardware-efficient implementation of sign detection and magnitude comparison in residue number systems. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/70712>

<https://doi.org/10.32657/10356/70712>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**New Algorithms for Hardware-efficient
Implementation of Sign Detection and
Magnitude Comparison in Residue
Number Systems**

SACHIN KUMAR

**SCHOOL OF ELECTRICAL AND ELECTRONIC
ENGINEERING**

2017

NEW ALGORITHMS FOR HARDWARE-EFFICIENT IMPLEMENTATION OF SIGN DETECTION AND
MAGNITUDE COMPARISON IN RESIDUE NUMBER SYSTEMS

SACHIN KUMAR

2017

New Algorithms for Hardware-efficient Implementation of Sign Detection and Magnitude Comparison in Residue Number Systems

SACHIN KUMAR

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

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

2017

Acknowledgment

First and foremost, I would like to express my sincere gratitude to my supervisor, Associate Professor Chang Chip Hong for his support and guidance during my Ph.D. candidature in the School of Electrical and Electronic Engineering, Nanyang Technological University. His enthusiasm and passion in research has inspired me to do best in my research area in the past four years. Without his intellectual guidance, support and attitude to strive for the best, I might not have the current research publications and completed this thesis on time. Last but not the least; I sincerely appreciate the time spent and invaluable suggestions given by him in the process of writing and reviewing the manuscripts of publications and this thesis.

I would also like to thank Mrs. Chang and Chip's Family members, Jeremy, Zhang Li, Cao Yuan, Zhang Le, Siarhei, Truan and Zhang Yue for their immense support during my candidature. I am also grateful for the support and assistance provided by the technical staffs in VIRTUS research centre and VLSI Laboratory.

I would like to express my gratitude to Professor Leonel Sousa for providing me an opportunity to do my research attachement in INESC-ID, Lisbon during my final year. I am thankful to him for funding my travel expenses and accommodation during my stay without which I would not have a chance to work in a differecnt research environment. In addition, his kind support has made my stay in Lisbon an unforgettable wonderful memory.

Table of Contents

Acknowledgment	i
Table of Contents.....	iii
Abstract	vi
List of Figures	viii
List of Tables.....	xi
List of Abbreviations	xii
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Research Objectives.....	5
1.3 Major Contributions.....	6
1.4 Organization of Thesis.....	7
Chapter 2 Background and Literature Review.....	10
2.1 Residue Number System (RNS)	10
2.1.1 Residue Representation.....	10
2.2 Residue Arithmetic	12
2.3 Residue-to-Binary Conversion	16
2.4 RNS Sign Detection.....	19
2.5 Magnitude Comparison in RNS.....	23
Chapter 3 New Algorithms for Hardware-efficient Implementation of Sign Detection in $\{2^n-1, 2^n, 2^n+1\}$ RNS	28
3.1 Introduction	28
3.2 Scaling-assisted Sign Detection Method I	29
3.2.1 Proposed Algorithm.....	29
3.2.2 Proposed Hardware Architecture.....	31
3.3 Scaling-assisted Sign Detection Method II.....	36
3.3.1 Proposed Algorithm.....	36

3.3.2	Proposed Hardware Architecture	40
3.4	Comparison and Results	45
3.4.1	Scaling-assisted Sign Detection Method I	45
3.4.2	Scaling-assisted Sign Detection Method II	46
3.5	Summary	48
Chapter 4 New Algorithm for Signed Residue Comparison in Four-moduli Supersets and Its Efficient VLSI Implementations		49
4.1	Introduction	49
4.2	Proposed Algorithm for Comparison of Signed Integers in RNS	50
4.3	Proposed Hardware Architecture for Comparison of Signed Integers in RNS	55
4.4	Evaluation and Comparison	65
4.4.1	Qualitative Comparison of General Features	65
4.4.2	Synthesis Results	66
4.5	Summary	73
Chapter 5 A Scaling-assisted Signed Integer Magnitude Comparison for Five Moduli Superset RNS $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$		74
5.1	Introduction	74
5.2	Scaling-Assisted Sign Detection Algorithm for Comparison of Signed Integers in $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ RNS	75
5.3	Proposed Hardware Architecture for Signed Integer Comparator in RNS	81
5.3.1	A_x Generator	82
5.3.2	B_x Generator	83
5.3.3	Computations of X_s and $\text{sign}(\tilde{X})$	84
5.3.4	Computations of A_z, B_z, Z_s and $\text{sign}(\tilde{Z})$	91
5.3.5	Relational Inference	96
5.4	Comparison and Discussions	97
5.4.1	Qualitative Comparison	97
5.4.2	Synthesis Results	99
5.5	Summary	103

Chapter 6 Conclusion and Future Works	104
6.1 Conclusions	104
6.2 Future Works	106
Author's Publications	109
References	110
Appendix	122

Abstract

Residue Number System (RNS), being a non-positional number system, is emerging as a promising data representation to substitute the accustomed two's complement number system for low-power and high-speed digital signal processing. Due to its carry free property at sub-word level, arithmetic operations such as addition, subtraction and multiplication can be performed at higher speed than any weighted number system. Nonetheless, the arithmetic operations such as sign detection, signed integer magnitude comparison are found to be more difficult to implement in RNS. This thesis aims to develop new algorithms for hardware-efficient implementation of these difficult operations in RNS. Specifically, the following new findings and results are reported in this thesis.

First and foremost, a new fast and area-efficient adder-based sign detector for RNS $\{2^n-1, 2^n, 2^n+1\}$ has been proposed. The circuit is greatly simplified by shrinking the dynamic range to eliminate large modulo operations with the help of the new Chinese remainder theorem (CRT)-I. Synthesis results based on 65nm CMOS standard cell library show that the proposed design outperforms all existing adder-based sign detectors reported for this moduli set in area and speed for n ranges from 5 to 25 in step of 5. Following which, a radically different quantization approach for comparing signed integers in the four-moduli supersets, $S_1 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$, is proposed. The dynamic range of the target moduli set is quantized into equal divisions, and the ranks of the divisions resided by the residue representations of the integers in comparison are identified and compared. This approach allows the sign of a residue representation to be directly extracted from the most significant bit of its rank, or with some simple additional logic function if it resides in the middle rank. Comparing with the best existing signed magnitude comparator applicable to these two moduli sets, the synthesis results based on 65 nm CMOS technology show that the proposed design is at least 22.48% smaller, 19.08% faster and 27.66% more energy-efficient for S_1 and 18.75% smaller, 16.41% faster and 23.30% more energy-efficient for S_2 . Last but not least, a scaling-assisted sign integer comparator for the balanced five-moduli set $\{2^n-1, 2^n, 2^n+1,$

$2^{n+1}-1, 2^{n-1}-1\}$ has been proposed. The signs of the operands in comparison, as well as their difference are detected after scaling them by a factor of $(2^{2n}-1)(2^{n-1}-1)$. The resulting finite series in the composite modulus channel is further factored into parallel carry-saved additions in the existing mod 2^n and mod $2^{n+1}-1$ modulus channels to reduce the sizes of modulo adders from $5n$ bits to n and $n+1$ bits. Upon detecting the signs of the operands and their difference, the relation is inferred with a small fraction of logic gates. Synthesis results in 65 nm CMOS technology show that the proposed design is 36.9% smaller, 7.6% faster and 45.5% more energy-efficient than the best CRT-II based magnitude comparator and at least 12.9% smaller, 7.3% faster and 20.8% more energy-efficient than the best reverse-conversion based implementation of signed integer comparator for the same five-moduli set.

List of Figures

Figure 2.1: Overview of RNS System.	15
Figure 2.2: The sign detection circuit for RNS of a general moduli set.	22
Figure 3.1: Reduction of addends.....	33
Figure 3.2: Generation of c_{in} signal.	34
Figure 3.3: Proposed sign detection circuit.	36
Figure 3.4: Calculation of A and B by modulo 2^8-1 CSA with EAC.	36
Figure 3.5: Mapping of the half ranges of integer X in $[0, M)$ to the half ranges of its scaled integer Y in $[0, M')$	37
Figure 3.6: Generation of carry-in signal c_{in}	42
Figure 3.7: Example of the generation of the carry signal C_1 and v for $n = 8$	42
Figure 3.8: Proposed sign detection architecture for $\{2^n-1, 2^n, 2^n+1\}$	43
Figure 3.9: Simplified prefix adder for $n = 8$	44
Figure 3.10: Computation of Y for Example 1.....	44
Figure 4.1: The quantization of M into $m_4 = 2^{n+1}-1$ for S_1 or $m_4 = 2^{n-1}-1$ for S_2 divisions.....	50
Figure 4.2: Proposed sign integer comparison algorithm in RNS S_1 or S_2	54
Figure 4.3: Architecture of the proposed signed integer comparison algorithm.....	56
Figure 4.4: Circuit architectures of (a) V generator (b) MFA.	59
Figure 4.5: Computation of W_x for $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ with $n=16$	61
Figure 4.6: Circuit architectures of W generators for (a) S_1 and (b) S_2	63
Figure 4.7: Circuit implementation of sign detector module.	64
Figure 4.8: Circuit implementation of relational inference module.....	65
Figure 4.9: Comparison of synthesized area of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$	67
Figure 4.10: Comparison of delay of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$	68

Figure 4.11: Comparison of synthesized area of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	68
Figure 4.12: Comparison of delay of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	69
Figure 4.13: Comparison of total power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	70
Figure 4.14: Comparison of leakage power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	70
Figure 4.15: Comparison of total power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	71
Figure 4.16: Comparison of leakage power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	71
Figure 4.17: Percentage reduction in energy consumption of the proposed design over the magnitude comparators of [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	72
Figure 4.18: Percentage reduction in energy consumption of the proposed design over the magnitude comparators of [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.	73
Figure 5.1: The mapping of integers in $[0, M)$ to its scaled integers in $[0, M_{R2})$.	76
Figure 5.2: Comparison of $(x_1, x_2, \dots, x_5)$ and $(y_1, y_2, \dots, y_5)$ with the help of scaling-assisted sign detection.	80
Figure 5.3: Overall architecture of the proposed RNS signed integer comparator.	81
Figure 5.4: Circuit modules required to compute (a) A_x and (b) B_x .	83
Figure 5.5: Architecture for the generation of $X_{s,2n:n-1}$, (b) sign detector of $\tilde{X}$.	90

Figure 5.6: Architecture for the generation of (a) A_z and (b) B_z	93
Figure 5.7: Architecture for the generation of $Z_{s,2n:n-1}$	96
Figure 5.8: Circuit implementation of relational inference module. The inset (right) is the subcircuit for the detection of $x_i = y_i$ for $i = 1$ to 5 except $i = 3$, where $p = \lceil \log_2 m_i \rceil - 1$	97
Figure 5.9: Percentage reduction in energy consumption of the proposed design over Unsigned and Signed CRT-II and Hybrid for S with $n = 4$ to 20.....	103

List of Tables

Table 3.1: Comparison of area, critical path delay and power of sign detectors	46
Table 3.2: Comparison of area (μm^2) and delay (ns) of sign detection circuits for $\{2^n-1, 2^n, 2^{n+1}\}$	47
Table 3.3: Comparison of average leakage (LP) in μW and total power (TP) in mW of sign detection circuits for $\{2^n-1, 2^n, 2^{n+1}\}$	47
Table 4.1: Evaluation of RNS Magnitude Comparison Algorithms	66
Table 5.1: k_3 in decimal and $\lfloor k_3/2^n \rfloor$ in binary and CSD representations.	88
Table 5.2: Evaluation of RNS Magnitude Comparison Algorithms	98
Table 5.3: Comparison of areas (μm^2) and critical path delays (ns) of RNS sign detectors for $S =$ $\{2^n-1, 2^n, 2^{n+1}, 2^{n+1}-1, 2^{n-1}-1\}$	100
Table 5.4: Comparison of Synthesized Area (μm^2) and Delay (ns) of the Proposed Design and Alternative signed and unsigned integer comparators for $S = \{2^n-1, 2^n, 2^{n+1}, 2^{n+1}-1, 2^{n-1}-1\}$	101
Table 5.5: Comparison of leakage power (LP) in μW and total power (TP) in mW of RNS sign detectors for $S = \{2^n-1, 2^n, 2^{n+1}, 2^{n+1}-1, 2^{n-1}-1\}$	102
Table 5.6: Comparison of Leakage power (LP) in μW and total power (TP) in mW of the Proposed Design and Alternative signed and unsigned integer comparators for S	102

List of Abbreviations

CRT	Chinese Remainder Theorem
CSA	Carry Save Adder
CPA	Carry Propagation Adder
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
EAC	End-Around-Carry
FA	Full Adder
FIR	Finite Impulse Response
LSB	Least Significant Bit
MFA	Modified Full Adder
MR	Mixed Radix
MRC	Mixed-Radix Conversion
MSB	Most Significant Bit
RNS	Residue Number System
ROM	Read-Only Memory
RRNS	Redundant Residue Number System
SQ	Sum of Quotients
SQT	Sum of Quotients Technique
TCS	Two's Complement Number System

Chapter 1

Introduction

1.1 Motivation

Higher speed, smaller area and lower power consumption have become the principle necessity in modern integrated circuit designs. Power dissipation, in particular, has emerged as the key issue not only for large fixed computing and memory systems in data center but also for portable and mobile electronic devices due to the increased integration density brought by the advanced semiconductor manufacturing technology. Reducing the power consumption to prolong the battery life, increasing the mobility and enhancing the user experience are now the prime competitive driver for the sales of portable electronics. Lowering the power consumption contributes to the use of smaller and cheaper battery and smaller footprint by releasing the space occupied by the bulky cooling system. Over the last few decades, unceasing growth in the semiconductor market has largely been benefited from the scaling of CMOS devices. However, lowering the cost of each transistor and improving the system performance by packing more transistors of smaller feature size into each wafer have reached their limit due to the dark silicon and reliability issues associated with high power dissipation. The benefits offered by device scaling can only be fully enjoyed provided that the power consumption by each functional block can be substantially reduced to eliminate excessive heat generation on silicon chip [Sto01], [Aka11], [Bai07], [Gar00].

Due to the demand for higher speed with reduced power consumption, many real-time digital signal processing (DSP) functions in diverse applications such as healthcare products, industrial control, biomedical field, test and measurement, military applications, manufacturing automation etc have been implemented on application specific integrated circuit (ASIC). High-speed and low-power arithmetic circuits are required to execute the complex numerical operations in these dedicated digital signal processors. The way numbers are represented in the arithmetic operations influences the power dissipation of the arithmetic operators in several

levels of design abstraction. Prevailing digital signal processors are implemented based on two's complement number system. This accustomed number system has a rooted carry propagation problem that limits the speed and power consumption improvement as it does not scale with the size and complexity of the operations. To reduce the number of devices required to realize an operator and its data activity, alternative number systems are explored, which lead to this study of a non-weighted number system called Residue Number System (RNS).

RNS has received considerable attention due to its inherent parallelism, modularity and localized carry propagation property. In RNS, a binary number with large word length is decomposed to smaller word length numbers, known as residues, with respect to the selected moduli set. Arithmetic operations such as addition, subtraction and multiplication can be performed concurrently and independently without carry propagation across the residue channels, which form the basis for the high-speed and low-power VLSI implementations of these arithmetic operations in RNS [Tay84], [Sod 86], [Sas68], [Amo07]. Many modern DSP applications, such as digital filter, discrete wavelet transform (DWT), discrete Fourier transform (DFT) [Con04], [Car05], [Ber07], [Liu04], [Tay90], [Mel96], cryptography [Sch09], [Sch14], [Baj04], digital communication [Ram02] and image processing [Vas13], [Moh12] are dominated by a large number of additions, subtractions and multiplications. These applications can directly benefit from the RNS-based computations. For examples, in [Con04], a RNS-based 16-tap filter based on moduli in the form of 2^n and $2^n \pm 1$ was implemented. When compared with two's complement number system (TCS)-based filter, the design achieves a saving of at least 35% area-delay product for the dynamic range from 20 to 40 bits. In [Car05], the synthesis results show that parallelism in RNS can reduce up to 40% of total power per tap consumed by TCS-based filter without any delay penalty. In [Vas13], a RNS-based Gaussian smoothing and Sobel's edge detector was proposed using the arithmetic-friendly three moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. For a 256×256 image, the synthesis results based on 32 nm CMOS standard cell technology show that the design occupies $112332 \mu\text{m}^2$ of area and consumes 30.23 mW of power at 1.05V and 250 MHz. The circuit is capable of processing the image at higher speed as the actual critical path delay is less than 2 ns.

However, arithmetic operations such as sign detection, magnitude comparison, scaling, division and reverse conversion (i.e., residue to binary conversion) involve inefficient inter-modulo computations. These operations are found to be more complex in RNS than in the conventional number system. Unlike addition, subtraction and multiplication, inter-modulo operations can only be computed by the collective information derived from all residue channels to produce the correct result. Due to the non-position nature of RNS, these operations require more hardware resources and incur larger delay than addition, subtraction and multiplication. Unfortunately, inter-modulo operations cannot be completely eliminated as they play important roles in broadening the applications of RNS. One of the inter-modulo operations, RNS reverse converter, is ineluctably required in the final stage of RNS-based computations if the final result of the computation needs to be interfaced to or stored in processing or storage unit operating in the normal binary number system. Over the last two decades, many researchers have proposed many efficient algorithms to the reverse conversion problem. Many of these reverse conversion techniques are based on the moduli sets containing the three most arithmetic friendly moduli [Cao03], [Moh07], [Pat14], [Mol10], [Pet13a], [Pet13b], [Ska98], [Har08], [Mol08], [Con99], [Gri98], [Vin94], and [Wan00a]. Similarly, other operations involving scaling, sign detection and magnitude comparison are also required to complete a system implementation directly in the RNS domain without the to-and-fro conversions into the two's complement number system. Since these operations play important roles in digital processing, they have become the obstacles to the full-fledged implementation of RNS-based application-specific DSPs.

Sign detection is frequently encountered in digital signal processing applications such as infinite impulse response digital filters, computational geometry and exact arithmetic operations [Bro97] and image processing [Moh12]. The importance of sign detection in RNS also stems from it being a stumbling block of other difficult signed arithmetic operations like comparison, overflow detection, scaling in RNS [Ma10], [Low13] and base transformation [Tay15]. When signed residue representations are involved in these operations, a sign detector module is required to correct the computation of unsigned arithmetic if the operand in the residue representation is negative. Similar to TCS, the positive and negative residue representations in

signed RNS are mapped symmetrically to the lower half and upper half of the dynamic range, respectively. The straightforward method to detect the sign is to decode the magnitude of a signed integer by performing full residue to binary conversion on its residue representation in RNS. Then, sign is determined by comparing the magnitude with the halfway point of its dynamic range. However, this method is inefficient as the reverse conversion itself is a complicated and difficult inter-modulo operation in RNS. For this reason, there is a dire need to develop the smaller and faster RNS sign detector circuits so that the sign detection does not impede the performance of the overall system.

Comparator is another frequently used component in many DSP algorithms and almost indispensable in sorting network [Che03], computer vision [Yu08], video coding [Mat08], image processor [She08], [Par09] and general purpose processor [Suz07], [Pon04], [Ok194]. For instances, in sorting network [Che03], huge numbers of comparators are needed to sort a large pool of data for path recirculation of switching network and multi-access memory, and for workload balancing of multi-processor system. In computer vision and video coding applications, integer comparisons are used for detecting the minimum value and computing the absolute difference in order to determine the similarity between two data sets by the sum of absolute difference. Hence, the overall performance of a system can be dominated by the efficiency of the comparator. In RNS, comparison of signed residue representation involves sign detection and magnitude comparison, and both operations are slow and expensive. The existing residue comparators are mainly developed for comparison of unsigned integers only due to hardware complexities and difficulty to unify sign detection and magnitude comparison in RNS. To address this long lasting problem, signed integer magnitude comparator in RNS is revisited as a holistic design module by exploring relevant number theoretic properties of special moduli and scaling in residue domain to reduce the size of modulo operators. The ultimate goal is to unify sign detection and magnitude comparison into one single fully combinational adder-based module without the need for a full reverse conversion process by Chinese remainder theorem (CRT) and sequential mixed radix conversion (MRC). Albeit challenging, it will be more useful if this can be achieved on higher cardinality special moduli set with larger dynamic range than the classic triple moduli set.

1.2 Research Objectives

Motivated by the merits of arithmetic-friendly moduli and the significance of sign detector and signed integer comparator in application-specific DSPs, this thesis aims to develop algorithms for hardware-efficient implementation of sign detector and signed integer magnitude comparator. The strategy is to find out the drawback and strength of existing approaches in order to develop the new solutions for the aforementioned operations based on special moduli sets. Meanwhile, theorems such as New CRT-I, CRT-II and the number theoretic properties for special moduli will also be explored to simplify the targeted inter-modulo operations. This research also aims to discover new ways to fuse different inter-modulo operations such as sign detection, magnitude comparison and scaling into a multi-functional module. This will be achieved by sharing scaled residue, which is obtained by choosing an appropriate scaling factor, to perform sign detection and sign integer comparison. This approach helps to achieve more significant area, power and speed improvement and greater versatility of operation sharing in high-level synthesis.

To fulfil the above objectives, the following research problems have been identified.

- a) To investigate the potential of quantizing the dynamic range to lower if not completely break the inter-modulo dependency of composite moduli so that inter-modulo computations can be carried out in parallel and independently in each modulus channel to a large extent.
- b) To investigate the potential of using scaling with appropriate scaling factors to reduce the dynamic range so as to simplify sign detection and signed integer comparison.
- c) To develop an efficient algorithm for signed residue comparison without requiring additional sign detection module and lookup tables.
- d) To develop an efficient algorithm for sign detection for signed integer comparison for larger cardinality.

1.3 Major Contributions

The major contributions of the research work are highlighted as follows:

- a) A pioneering concept of using an appropriate scaling factor to reduce the size of residue arithmetic operations involved in sign detection process is introduced. Two new and unique algorithms of sign detection method are presented for the three moduli set $\{2^n-1, 2^n, 2^n+1\}$. In the first algorithm, the complexity of proposed sign detection circuit is greatly reduced by leveraging the one's complement and circular left shift properties of 2^n-1 in the new modified Chinese remainder theorem and by scaling the residues by a factor of 2^n . The proposed sign detection circuit is faster and consumes smaller hardware area than the sign detection circuit designed out of the most efficient reverse conversion unit for the same moduli set. In the second algorithm, the proposed technique exploits the new CRT-I to greatly simplify the $2^{2n}-1$ scaling of residue representation into addends that can be readily obtained by circular left shifted residues at no hardware cost. The reduced dynamic range enables the sign of an integer to be computed from the most significant bit (MSB) of scaled residue with a heavily stripped down version of reverse converter. Due to its simplicity, the proposed sign detector has faster speed, smaller hardware area and requires less power than the state-of-the-art sign detectors for the same moduli set.
- b) A new design method of an efficient signed integer comparator for comparing signed integers in the four-moduli supersets, $S_1 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$, is proposed. The algorithm is developed by using a radically different quantization approach in which the dynamic range of the target moduli set is quantized into equal divisions, and the ranks of the divisions resided by the residue representations of the integers in comparison are determined and compared. Meanwhile, the sign of residue representation can be directly extracted from the MSB of its rank, or with simple logic function if it resides in the middle rank. This simplified architecture for both S_1 and S_2 can be implemented with the help of combinational logic gates without any lookup tables. The proposed signed integer comparator outperforms not only the best existing signed magnitude

comparators for these two moduli sets, but also the existing unsigned integer comparators in terms of area, speed and energy-efficiency.

- c) A simple and unique scaling-assisted approach for the comparison of signed integers in the balanced five moduli set $\{2^n-1, 2^n, 2^{n+1}, 2^{n+1}-1, 2^{n-1}-1\}$ is presented for the first time. Since this operation involves two difficult and expensive inter-modulo operations, i.e., sign detection and magnitude comparison, it's difficult to fuse even for the lower cardinality three moduli sets as their respective hardware complexity grows with the cardinality of moduli sets. Unlike the existing approaches, comparison of two signed integers is performed by computing the sign of the operands in comparison as well as their difference. Their signs are detected from a scaled residue which is obtained after scaling by a factor of $(2^{2n}-1)$ $(2^{n-1}-1)$. The size of scaled residues with respect to composite mod (2^n) $(2^{n+1}-1)$ sum for detecting the signs can be further decomposed into two parallel mod 2^n and mod $(2^{n+1}-1)$ sums. Therefore, the algorithm can be implemented with the help of modulo adders of no more than $n+1$ bits wide without any usage of lookup tables. The proposed circuit is able to run faster with smaller area and lower power consumption than the best reverse conversion based implementation of signed integer comparator for the same five-moduli set.

The above contributions have led to the publications listed in the author's publications at the end of the thesis.

1.4 Organization of Thesis

This thesis is organized into six chapters. Chapter 1 includes the motivation, the objective and the major contributions of research work.

Chapter 2 presents the fundamental of RNS including signed and unsigned residue representations, residue arithmetic operations, and residue to binary conversion in detail. In addition, an extensive review of existing works on RNS sign detector and RNS magnitude comparator is also provided.

The major original contributions of this thesis are presented in Chapters 3 to 5. Chapter 3 introduces a novel concept of scaling assisted sign detection. Based on this idea, two new algorithms related to the sign detection for the celebrated moduli set $\{2^n-1, 2^n, 2^n+1\}$ are proposed. Their translation into efficient hardware implementation is also elaborated in details. The logic synthesis results of the sign detectors designed by the proposed algorithms will also be analyzed and compared with the sign detectors designed by existing techniques for the same moduli set.

In Chapter 4, a new range quantization approach to the problem of signed integer comparison in RNS of two higher cardinality four-moduli supersets, $S_1 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$, is proposed. The concept of range quantization is introduced before the proposed algorithm for both moduli sets is described in detail. Their simplified hardware implementations are also derived and discussed. The proposed algorithms for both S_1 and S_2 are qualitatively compared against existing algorithms followed by the evaluation of their performance against compatible comparators designed based on existing techniques in terms of area, delay and energy efficiency.

A scaling-assisted sign integer comparator for an even higher cardinality balanced five moduli set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ is proposed in Chapter 5. The fundamental concept for comparing the sign integer by scaling-assisted sign detection is introduced before the presentation of the proposed algorithm and its hardware implementation details. The proposed circuit is greatly simplified by ingenious use of number theoretic properties and canonical signed digit representation for the summands of multiplicative inverses. The qualitative comparison of the proposed design with existing approaches is performed before the logic synthesis results of the proposed signed integer comparator is analysed and compared against the best possible existing techniques used for the implementation of signed and unsigned integer comparators for the same moduli set.

Chapter 6 summarizes the main contributions presented in Chapters 3 to 5, and presents several relevant and promising ideas for future exploration.

Chapter 2

Background and Literature Review

2.1 Residue Number System (RNS)

2.1.1 Residue Representation

The ancient human brought the numbers into existence in the purpose of counting and measuring the objects. A numeral system is a way to represent the numbers uniquely along with the arithmetic operations that can be carried out on them. There are two main types of number system, Weighted Number System and Non-Weighted Number System. A weighted number system can be defined by the following mathematical equation:

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

where x_i is the i -th digit and w_i is the weight of x_i .

Depending on the set of weights w_i and digits x_i , the weighted number system can be further classified into many number systems but the most famous weighted number systems are the decimal number system, hexadecimal number system, octal number system and binary number system. The set of weights in decimal number system are successive powers of ten and x_i lies in $[0, 9]$. Similarly, octal number system has a set of weights in successive powers of eight and x_i is an integer value in $[0, 7]$ whereas the binary number system has a weight in power of 2 and x_i has only two values, which is either 0 or 1. Modern processors work on two possible electrical states, i.e., either on or off, which represent the status of flow of current in the circuit. The “on” state is represented by logic value ‘1’ while the “off” state is assigned the logic ‘0’ value in the binary numeral system. Therefore, the binary number system is used in digital electronic circuits to perform all kinds of arithmetic and logical operations. The hexadecimal numeral system has a set of weights in successive powers of 16 and the possible values of x_i are from 0 to 15. This numeral system is useful to group a large string of binary bits into nibbles. On the other hand, in non-weighted number system, numbers are represented without a fixed set of

positional weights. One example of the non-positional number system is the Residue number system (RNS).

An RNS is defined by a set of N coprime integers $\{m_1, m_2, \dots, m_N\}$, where the greatest common divisor of any two moduli is unity, i.e. $\text{GCD}(m_i, m_j) = 1$ for $\forall i \neq j$ and m_i is known as the modulus. The product of all moduli, $M = \prod_{i=1}^N m_i$, defines the dynamic range of the RNS. An unsigned integer X can be represented by a set of N smaller integers $(x_1, x_2, \dots, x_N)$ in this moduli set, where each residue x_i is a least non-negative remainder obtained by dividing X by m_i , which can be expressed mathematically as $x_i = |X|_{m_i}$ for $i = 1, 2, \dots, N$. If all moduli are relative prime integers in the given moduli set, every unsigned integer within $[0, M-1]$ will have unique residue representation [Gar59].

For a given moduli set $\{m_1, m_2, \dots, m_N\}$ with dynamic range M , the residue representation of an integer $X' \in [C, M+C-1]$, where C is any integer constant, can be mapped to the residue representation of an integer $X \in [0, M-1]$ based on the following congruence [Tom08]:

$$X' \equiv |X - C|_M + C \quad (2.2)$$

For bipolar (or signed integer) representation in RNS, C can be chosen to be any integer within the range $(-M, 0)$. Equation (2.2) makes it possible to represent signed numbers in residue form while keeping the arithmetic units for the operations $\{+, -, \times\}$ compatible with those of the unsigned residues under the same RNS. C is usually set to $-\left\lfloor \frac{M}{2} \right\rfloor$, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x , to divide the range M into two symmetrical half ranges for the representation of positive and negative integers. When M is even, the range of signed integer $\hat{X}$ that can be unambiguously represented in RNS is $[-M/2, M/2-1]$. Similarly, for odd M , the range of unambiguously representable signed integers in RNS is $[-(M-1)/2, (M-1)/2]$. By substituting $X' = \hat{X}$ and $C = -\lfloor M/2 \rfloor$ into Equation (2.2), the

signed integer $\hat{X}$ can be represented using the same residue representation as an unsigned integer X for the same moduli set as follows:

$$\hat{X} \equiv \left\lfloor X + \left\lfloor \frac{M}{2} \right\rfloor \right\rfloor_M - \left\lfloor \frac{M}{2} \right\rfloor \quad (2.3)$$

When $\hat{X} \geq 0$, the residue representation of X can be mapped to that of $\hat{X}$ in the range of $[0, M/2 - 1]$ if M is even and $[0, (M-1)/2]$ if M is odd. In a similar way, when $\hat{X} < 0$, the residue representation of X can be mapped to that of $\hat{X}$ in the range of $[M/2, M-1]$ if M is even and $[(M+1)/2, M-1]$ if M is odd [Tay14], [Cha14]. Using this mapping, the relationship between $\hat{X}$ and X under the same moduli set can be formulated as:

$$\hat{X} = \begin{cases} X & \text{if } \hat{X} \geq 0 \\ X - M & \text{if } \hat{X} < 0 \end{cases} \quad (2.4)$$

$$\hat{X} = X - s \cdot M$$

where $s \in \{0, 1\}$ represents the sign of $\hat{X}$.

The sign s of $\hat{X}$ can be identified by checking whether X falls in the lower or upper half range of M [Cha14]. When M is even,

$$s = \begin{cases} 0 & \text{if } X \in [0, (M/2) - 1] \\ 1 & \text{if } X \in [M/2, M - 1] \end{cases} \quad (2.5)$$

When M is odd,

$$s = \begin{cases} 0 & \text{if } X \in [0, (M-1)/2] \\ 1 & \text{if } X \in [(M+1)/2, M - 1] \end{cases} \quad (2.6)$$

2.2 Residue Arithmetic

In RNS, the integer operands X and Y can be represented by N -tuples of residue digits, $(x_1, x_2, \dots, x_N)$ and $(y_1, y_2, \dots, y_N)$, respectively for the moduli set $\{m_1, \dots, m_N\}$. The residue representation of the results of the operation $Z = X \circ Y$ can be expressed as:

$$\begin{aligned}
Z &\equiv (z_1, z_2, \dots, z_N) \\
&\equiv (|x_1 \circ y_1|_{m_1}, |x_2 \circ y_2|_{m_2}, \dots, |x_N \circ y_N|_{m_N})
\end{aligned} \tag{2.7}$$

where $(z_1, z_2, \dots, z_N)$ is an N -tuple of residue digits of Z , with $z_i = |x_i \circ y_i|_{m_i}$ for $i \in [1, N]$ and ‘ $\circ$ ’ represents an arithmetic operation such as addition, subtraction or multiplication.

From (2.7), the resultant residue digits of $Z \equiv (z_1, z_2, \dots, z_N)$ are obtained by performing the arithmetic operation on the residue digits of X and Y in the respective modulus channels. Therefore, in each modulus channel, the residue arithmetic operation can be carried out in parallel and independently without the propagation of carry across modulus channels as shown in Figure 2.1. The carry propagation delay of residue arithmetic operations are lower than those of the arithmetic operations performed in weighted number systems due to the shorter word-lengths of residue digits. On the other hand, inter-modulo operations, such as sign detection, magnitude comparison, scaling, overflow detection and reverse conversion, are generally slower and more expensive than the same operations performed in weighed number systems. Unfortunately, these operations cannot be performed in each residue channels independently. In addition, these operations are difficult to parallelize as they require the combination of multiple residue digits to compute which makes them more difficult to be implemented efficiently. Since these inter-modulo operations may also play a vital role in RNS-based system, there is a desperate need to reduce the hardware implementation complexity of these operations.

To simplify the hardware implementation of large modulus arithmetic for these difficult inter-modulo operations in RNS, special moduli in the form of 2^r-1 , 2^r , 2^r+1 for any integer r are often explored as these moduli have good number theoretic properties. The following useful properties in modulus arithmetic that can be exploited to simplify the RNS based computations are extensively used in this thesis:

Property 2.1: $|X \pm Y|_m = ||X|_m \pm |Y|_m|_m$

Property 2.2: $|X \times Y|_m = ||X|_m \times |Y|_m|_m$

Property 2.3: $|km|_m = 0, \forall k \in \mathbb{Z}$

Property 2.4: $k|X|_m = |kX|_{mk}, \forall k \in \mathbb{Z}$

Property 2.5: $|-x|_m = |m-x|_m$

Property 2.6: $|x+km|_m = |x|_m, \forall k \in \mathbb{Z}$

Property 2.7: If m is a prime number, then $|k^m|_m = |k|_m, \forall k \in \mathbb{Z}$

Property 2.8: If $P = km, \forall k \in \mathbb{Z}$, then $|x|_P = |x|_m$

Property 2.9: Let $0 \leq x < m$ and $|x \times y|_m = 1$, then x is the multiplicative inverse of y mod m ,

denoted as $x = |y^{-1}|_m$, such that $\gcd(y, m) = 1$.

Property 2.10: The negation of an integer x modulo 2^n can be expressed as:

$$|-x|_{2^n} = |(2^n - 1 - x) + 1|_{2^n} = |\bar{x} + 1|_{2^n} \quad (2.8)$$

where $\bar{x}$ represents the one's complement of x .

Property 2.11: The modulo (2^n-1) product of the multiplication of an integer variable x and 2^r can be performed by shifting the n -bit binary representation of x circularly left by r bits. This property is known as the circular left shift property (CLS), which is expressed mathematically as [Sza67]:

$$|2^r x|_{2^n-1} = CLS_n(x, r) \quad (2.9)$$

where $CLS_n(x, r)$ represents the circular left shift operation of an n -bit binary representation of an integer y by r bit positions, and $0 \leq r < n$.

Property 2.12: If x in Property 2.11 is negated, i.e., $|2^r(-x)|_{2^n-1}$ then x needs to be one's complemented before applying the circular left shift property. i.e.,

$$|2^r(-x)|_{2^n-1} = CLS_n(\bar{x}, r) \quad (2.10)$$

where $\bar{x}$ represents the one's complement of x .

Property 2.13: The mod 2^n-1 reduction of a large r -bit ($r > n$) integer Z can be decomposed into a mod 2^n-1 sum of $(l+1)$ smaller n -bit integers [Pie94]:

$$\begin{aligned}
|Z|_{2^n-1} &= |(z_l \parallel \cdots \parallel z_2 \parallel z_1)|_{2^n-1} \\
&= |2^{l \times n} z_l + \cdots + 2^n z_2 + 2^0 z_1|_{2^n-1} \\
&= |z_l + \cdots + z_2 + z_1|_{2^n-1}
\end{aligned} \tag{2.11}$$

where $l = \lceil r/n \rceil - 1$ and z_i denotes the i -th summand of Z . $a \parallel b$ denote the concatenation of the binary representations of integers a and b .

Property 2.14: The product of an n -bit integer Z by 2 to the power of $-r$ taken modulo 2^n-1 is equivalent to the circular left shifting of the binary representation of Z by r bits [Pat14] :

$$|2^{-r} Z|_{2^n-1} = CLS_n(Z, n-r) = CRS_n(Z, r) \tag{2.12}$$

where $CRS_n(Z, r)$ represents the circular right shift operation.

Figure 2.1 shows the three main stages of a typical Residue Number System (RNS) [Cha15]. The forward converter involves the conversion of the binary input data into its residue digits with respect to each modulus m_i . This is followed by the arithmetic operation units that operate on the residue digits, which may include some non carry-free inter-modulo operations. The result of the computations in residue representation is converted back to the binary domain by the reverse converter.

Figure 2.1: Overview of RNS System.

2.3 Residue-to-Binary Conversion

Residue-to-binary conversion, also known as reverse conversion, refers to the process of decoding any integer X from their residue representation $(x_1, x_2, \dots, x_N)$. An integer X can be computed from its residues $(x_1, x_2, \dots, x_N)$ with respect to an N -tuple moduli set $\{m_1, m_2, \dots, m_N\}$ based on the Chinese remainder theorem (CRT) [Sza67], Mixed Radix Theorem (MRC) [Sza67], New CRT-1 [Wan98], CRT-II [Wan98] and alternative formulation of CRT [Sha08]. The theoretical underpinnings are described below.

Chinese Remainder Theorem (CRT):

$$X = \left\| \sum_{i=1}^n M_i \left\| M_i^{-1} x_i \right\|_{m_i} \right\|_M \quad (2.13)$$

where $M = \prod_{i=1}^N m_i$, $M_i = M/m_i$ and M_i^{-1} is the multiplicative inverse of M_i for $i \in [1, N]$.

Mixed Radix Conversion (MRC):

$$X = a_N \prod_{i=1}^{N-1} m_i + a_{N-1} \prod_{i=1}^{N-2} m_i + \dots + a_2 m_1 + a_1 \quad (2.14)$$

where $a_1, a_2, \dots, a_N$ are the MRC coefficients and $a_i \in [0, m_i)$. These coefficients can be computed as follows:

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= \left\| (x_2 - a_1) \left\| m_1^{-1} \right\|_{m_2} \right\|_{m_2} \\ a_3 &= \left\| \left((x_3 - a_1) \left\| m_1^{-1} \right\|_{m_3} - a_2 \right) \left\| m_2^{-1} \right\|_{m_3} \right\|_{m_3} \\ &\vdots \\ a_N &= \left\| \left(\left((x_N - a_1) \left\| m_1^{-1} \right\|_{m_N} - a_2 \right) \left\| m_2^{-1} \right\|_{m_N} - \dots - a_{N-1} \right) \left\| m_{N-1}^{-1} \right\|_{m_N} \right\|_{m_N} \end{aligned} \quad (2.15)$$

New Chinese Remainder Theorem (New CRT-I):

$$X = x_1 + m_1 \left\| \begin{aligned} &k_1 (x_2 - x_1) + k_2 m_2 (x_3 - x_2) \\ &+ \dots + k_{N-1} m_2 m_3 \dots m_{N-1} (x_N - x_{N-1}) \end{aligned} \right\|_{m_2 m_3 \dots m_{N-1} m_N} \quad (2.16)$$

where

$$\begin{aligned}
|k_1 m_1|_{m_2 m_3 \dots m_N} &= 1 \\
|k_2 m_1 m_2|_{m_3 \dots m_N} &= 1 \\
&\vdots \\
|k_{N-1} m_1 m_2 \dots m_{N-1}|_{m_N} &= 1
\end{aligned} \tag{2.17}$$

New Chinese Remainder Theorem –II (New CRT-II): Let (x_i, x_j) be a pair of arbitrary residues for the two-moduli set $\{m_i, m_j\}$, then

$$X_{ij} = |X|_{m_i m_j} = x_i + \left| m_i^{-1} (x_j - x_i) \right|_{m_j} \times m_i \tag{2.18}$$

This composite residue X_{ij} can be combined with another composite residue obtained from (2.18) in the next level of reduction until $X = |X|_M$ is obtained. This method is known as the New CRT (CRT-II).

This reverse conversion algorithm is best described by the following procedure:

Algorithm: translate $[(x_1, x_2, \dots, x_N), \{m_1, m_2, \dots, m_N\}, X]$ and $M = \prod_{i=1}^N m_i$

1) if $N > 2$ and N is even, then let $u = \left\lfloor \frac{N}{2} \right\rfloor$.

translate $[(x_1, x_2, \dots, x_u), \{m_1, m_2, \dots, m_u\}, X_1]$ and $M_1 = \prod_{i=1}^u m_i$

translate $[(x_{u+1}, x_{u+2}, \dots, x_N), \{m_{u+1}, m_{u+2}, \dots, m_N\}, X_2]$ and $M_2 = \prod_{i=u+1}^N m_i$

find no (X_1, X_2, M_1, M_2, X) .

2) if $N > 2$ and N is odd, then let $N = 2u+1$

translate $[(x_1, x_2, \dots, x_u), X_1]$, $M_1 = \prod_{i=1}^u m_i$

translate $[(x_{u+1}, x_{u+2}, \dots, x_N), X_2]$, $M_2 = \prod_{i=u+1}^N m_i$

find no (X_1, X_2, M_1, M_2, X) .

3) if $N = 2$, then find no $\left[(x_1, x_2), \{m_1, m_2\}, X \right]$.

4) if $N = 1$, then $X = \left| x_1 \right|_{m_1}$.

Procedure: find no (X_1, X_2, M_1, M_2, X)

1) Compute an integer k such that $\left| km_2 \right|_{m_1} = 1$, i.e., $k = \left| m_2^{-1} \right|_{m_1}$.

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

Alternative Formulation of CRT:

$$X = x_1 + m_1 \left| \sum_{i=1}^N w_i x'_i \right|_{m_2 m_3 \dots m_N} \quad (2.19)$$

When $N = 1$, $w_1 = \frac{M_1 \left| M_1^{-1} \right|_{m_1} - 1}{m_1}$, $x'_i = x_1$ and $M_1 = m_2 m_3 \dots m_N$. For $N > 1$, $w_i = \frac{M_i}{m_1}$,

$x'_i = \left| M_i^{-1} x_i \right|_{m_i}$ and $M_i = \frac{M}{m_i}$, where $i = 2, 3, \dots, N$.

CRT is the oldest reverse conversion technique among the reverse conversion techniques. In CRT, the intermediate terms $M_i \left| M_i^{-1} \right|_{m_i} x_i$ can be computed in parallel for all values of i . These terms are modulo M summed together since the value of integer X should fall in the range of $[0, M - 1]$. Hence, CRT involves a large modulo operation which has a negative impact on the cost and performance of the circuit. MRC is a sequential process. The mixed radix coefficients $(a_1, a_2, \dots, a_N)$ are computed sequentially as each coefficient has to be generated from its previous coefficient. Therefore, it has large delay, which is very inefficient for hardware implementation.

To overcome the problems of large modulo operation in CRT and sequential computation in MRC, CRT-I, CRT-II and new formulation of CRT have emerged. CRT-I is a parallel algorithm in mixed radix format to reduce the delay from $O(N)$ to $O(\log N)$. CRT-II uses a tree like structure to reduce the large modulo operation from M to $\sqrt{M}$.

An alternative formulation of CRT modifies CRT by decomposing a large modulo operation into N modulo operations. Each operation is based on one of the moduli m_i so that the reverse conversion can be carried out with faster and smaller word length parallel modulo operations.

2.4 RNS Sign Detection

The most intuitive way to detect the sign of an RNS representation is to convert its residue representation to binary form by the reverse conversion [Sza67] but the complete reverse conversion is slow and complex depending on the moduli set. Complete reverse conversion can be avoided by computing the value of the mixed radix coefficient a_N using MRC. The RNS representation is positive when $0 \leq a_N < m_N/2$ and negative otherwise. This method requires $(N-1)$ cycles to compute the value of a_N since a_N depends on the value of all other MRC coefficients. Hence, this sign detection method is still not efficient [Sza67], [Hen99]. Similar approach for sign detection was proposed in [Ulm83]. The sign is computed from the magnitude of a number through a process known as implicit-explicit conversion. The “implicit” process refers to the sign detection whereas the “explicit” process refers to the magnitude computation. In this method, $N(N+1)/2 + 2$ lookup tables are required. The sign detection method proposed in [Lin84] avoids the use of lookup tables but leads to a highly complex circuit due to its iterative nature.

A different approach for sign detection using fraction representation was proposed in [Vu85] in which the sum modulo M in the CRT conversion formula is reduced to a sum modulo 2 through the use of fractional representation and lookup tables. The modulo M reduction in the CRT equation of (2.13) can be eliminated by scaling down the integer X by $M/2$ to yield a fractional number $\tilde{X} = (2/M)X$, i.e.,

$$\tilde{X} = \left(\frac{2}{M} \right) \times X = \sum_{i=1}^N u_i \quad (2.20)$$

where $u_i = \frac{2}{m_i} |M_i^{-1} x_i|_{m_i}$ is a fractional number in the range $[0, 2)$ and N is the number of moduli.

The values of u_i are stored in a lookup table. These fractional numbers are expressed in a fixed point fractional binary representation with one binary bit for the integer part. These N fractional numbers are summed by an array of fixed point binary adders. The final result is also expressed in the same form of fractional binary representation such that $\tilde{X}$ falls in the range $[0, 2)$. Then, the sign of X can be determined by

$$\text{sign}(X) = \begin{cases} 0 & \text{if } 0 \leq \tilde{X} < 1 \\ 1 & \text{if } 1 \leq \tilde{X} < 2 \end{cases} \quad (2.21)$$

As the value of u_i is a fractional number, it may not be represented exactly in binary form with a finite number of bits. To correctly compute the sign of X , u_i needs to be represented with sufficient number of bits. Otherwise, the small truncation errors of each of these numbers may be accumulated. This method is proposed for moduli sets of any cardinality and binary adders of $\log_2 \lceil M \rceil$ bits are used to perform the summation of these fractional numbers. This method is unreliable in view of the accumulative truncation errors. Apart from any truncation errors, the performance of the resulting circuit is also poor due to the large adder tree and the size of the lookup tables required.

In [Ali98], the sign detection method was based upon CRT and base-extension method wherein the sign of an integer X in the range $[-\lfloor M/2 \rfloor, \lceil M/2 \rceil - 1]$ was computed by using a redundant modulus. The concept of this sign detection algorithm is explained as below.

Let $Y = 2X / M$. For the case when M is even, if $X = 0$, then $Y = 0$; if $X = M/2$, then $Y = 1$; if $X = M$, then $Y = 2$. Hence $\lfloor Y \rfloor$ is 0 when X is in $[0, M/2 - 1]$ and 1 when X is in $[M/2, M/2 - 1]$. Thus, $\text{sign}(X) = \lfloor 2X / M \rfloor$ for even M . For the case when M is odd, if $X = 0$, $Y = 2X / M = 0$; if $X = (M-1)/2$, then $Y = (M-1)/M < 0$; if $X = (M+1)/2$, then $Y = (M+1)/M$; if $X = M$, $Y = 2$.

Thus, for this case, $Y = 0$ when X falls in $[0, (M-1)/2]$ and $Y = 1$ when X falls in $[(M+1)/2, M-1]$. Since $2X = M \lfloor 2X/M \rfloor + |2X|_M$, $\text{sign}(X)$ can be written as

$$\text{sign}(X) = \frac{2X - |2X|_M}{M} \quad (2.22)$$

$\text{sign}(X) = 0$ indicates that X is positive and 1 indicates that X is negative.

From (2.22), $\text{sign}(X)$ can be interpreted as follows:

- ❖ If $2X < M$, then $|2X|_M = 2X$. Therefore, $\text{sign}(X) = 0$.
- ❖ If $2X \geq M$, then $2X - |2X|_M = M$. Therefore, $\text{sign}(X) = 1$.

Hence, the sign of X can be easily determined by computing the numerator of (2.22) with the help of a redundant modulus. Consider an n -tuple moduli set $\{m_1, m_2, \dots, m_n\}$ with a dynamic range of $M = \prod_{i=1}^n m_i$. Since $2X$ falls in $[0, 2M-2]$, by adding a redundant modulus m_{n+1} to the RNS, its residues can be computed in the redundant RNS (RRNS) by $|2X|_{m_i}$ for $i = 1, 2, \dots, n+1$. The subtraction of $2X - |2X|_M$ can be performed in the residue domain of RRNS by extending the n -digit representation of $|2X|_M$ to include the residue $\|2X\|_M|_{m_{n+1}}$ for the modulus channel m_{n+1} . If $2X < M$, then $|2X|_{m_{n+1}} = \|2X\|_M|_{m_{n+1}}$, which implies that X is in the positive range. Therefore, modulo m_{n+1} multiplier, modulo m_{n+1} adder and $\log_2(m_{n+1})$ bits comparator are required to detect the sign.

The residue corresponding to m_{n+1} can be computed by either MRC or CRT. MRC requires larger number of cycles than CRT for the base extension. In [She89], CRT with redundant modulus was used to compute the residue $\|2X\|_M|_{m_{n+1}}$ as shown below:

$$|2X|_{m_{n+1}} = \left\| \sum_{i=1}^n M_i \left\| \frac{2x_i}{M_i} \right\|_{m_i} - |kM|_{m_{n+1}} \right\|_{m_{n+1}} \quad (2.23)$$

where $k \leq n-1$ is computed by considering the redundant digit $x_r = |2X|_{m_r}$. With the help of (2.23), k can be expressed as:

$$k = |2X|_{m_r} = \left\lfloor \frac{1}{|M|_{m_r}} \left(\sum_{i=1}^n M_i \left\lfloor \frac{2x_i}{M_i} \right\rfloor_{m_i} - |2X|_{m_r} \right) \right\rfloor_{m_r} \quad (2.24)$$

ROMs, multioperand adders, modulo adder and binary comparator circuits are required to compute $\text{sign}(X)$, as shown in Figure 2.2. The multioperand modular adders (MOMA) are used to perform the summation required by the CRT. The $n+1$ number of ROMs in the left half input plane of Figure 2.2 are used to calculate the value of k whereas the n number of ROMs in the right half input plane are used to compute $\left\lfloor |2X|_M \right\rfloor_{m_{n+1}}$. The computation of the value of k for the base extension makes the circuit complex. As the computation time of the operations and the ROM space increases with the number of moduli, this design is not suitable for large dynamic range RNS.

Figure 2.2: The sign detection circuit for RNS of a general moduli set [She89].

The moduli set plays an important role in determining the complexity of the modular arithmetic circuit implementation. Many approaches target the moduli sets in order to exploit the simplified multiplicative inverses and modular operations for efficient hardware implementation. In [Set05], the sign can be detected with modulo subtractor and Exclusive OR gate for any three moduli set $\{m_1, m_2, m_3\}$ that fulfills the relation $|m_i|_{m_j} = 1$, where $j > i$ and

$i \in [1, N]$. As this method is based on MRC, it possesses larger sequential delay for extension to moduli sets of larger cardinality. The circuit complexity was further reduced in [Akk08] by using the modified MRC-II [Akk07] without the use of lookup tables.

The adder-based sign detection circuit based on the CRT-II in [Tom08] was proposed for the special moduli set $\{2^n-1, 2^n, 2^n+1\}$. The circuit complexity of this architecture is simplified by exploiting circular left shift property and other number theoretic properties for these modulo arithmetic friendly moduli. Therefore, the sign can be computed by performing two n -bit operations in parallel followed by an n -bit operation. Another approach for the sign determination was proposed in [Xu12] for the moduli set $\{2^n-1, 2^n, 2^n+1\}$. The algorithm was derived from a reverse converter algorithm based on the New CRT-I [Wan98]. The sign detection circuit was implemented by a two n -bit additions and a $(n+1)$ -bit binary comparator. For the same speed performance, this circuit is found to be smaller and less power-hungry than the approach given in [Tom08].

In [Xu15] and [Nir15], two recent sign detection techniques based on different approaches were proposed for the three moduli set $\{2^{n+1}-1, 2^n, 2^n-1\}$. In [Xu15], the algorithm was developed with the help of new formulation of CRT (modified CRT). Its sign detection circuit consists of a carry save adder (CSA), a comparator and a carry propagation unit. However, this method can only be applied to moduli sets that include 2^n as a modulus as it involves floor function unit in the algorithm while the sign detection algorithm [Nir15] was formulated by employing the New CRT-II. Its architecture can be implemented with the help of one n -bit carry save adder (CSA) and one $2n$ -bit parallel prefix carry generation unit.

2.5 Magnitude Comparison in RNS

Let $(x_1, x_2, \dots, x_N)$ and $(y_1, y_2, \dots, y_N)$ be the two residue representations to be compared and they can be mapped to the unsigned integers, X and Y , respectively by reverse conversion. There are two main approaches to the magnitude comparison problem in RNS. The parity check approach requires the computation of the parities of X , Y and their difference from their residue

representations, whereas the partial reverse conversion approach requires the conversion of the residue representations of X and Y into some weighted positional representations before the comparison. In what follows, the existing algorithms in these two categories are reviewed chronologically.

The first magnitude comparison algorithm for RNS was proposed in [Mil85]. This algorithm utilizes parity check approach. The parity of an integer X is determined with the help of the core function $C(X) = \sum w_i [X / m_i]$ and δ_i , where w_i and δ_i are respectively the core coefficient and parity function on the residue x_i . Therefore, an integer X is even if and only if $C(X)$ and δ_i have the same parity. The parity of $Z = 2|X - Y|_M \equiv (|2(x_1 - y_1)|_{m_1}, \dots, |2(x_N - y_N)|_{m_N})$ is also determined in order to compare the integers, X and Y . If Z is an even integer, then X is said to be greater than Y . Otherwise, X is smaller than Y . However, the case of $X = Y$ is not detectable because $Z = 0$ is considered as an even integer, and X will be assumed to be greater than Y in this case. There are two main weaknesses of this algorithm. The first weakness is the need of a redundant modulus to avoid the ambiguous cases and the second weakness is the limitation in the selection of core coefficients.

The algorithm proposed in [Lu92] also uses parity check. This algorithm computes the parities of X , Y and $Z = |X - Y|_M$. When X and Y have the same parity, X is greater than or equal to Y iff Z is an even number. Otherwise, X is less than Y . When X and Y have different parities, X is greater than or equal to Y iff Z is an odd number. Otherwise, X is less than Y . In order to detect the parity in the residue domain, a redundant modulus 2 is introduced. If a number modulo 2 is equal to zero, it is treated as an even number. Conversely, it is an odd number. In [Lu92], another algorithm for signed integer comparison in RNS was proposed. It requires an additional sign detection operation upfront. To compare two signed integers, $\tilde{X}$ and $\tilde{Y}$, in their residue form, three extra cases need to be considered. When $\tilde{X}$ and $\tilde{Y}$ have different signs, the positive integer is always larger than the negative integer. When $\tilde{X}$ and $\tilde{Y}$ are both positive, the algorithm for the comparison of unsigned integers in RNS is used. When $\tilde{X}$ and $\tilde{Y}$ are

both negative, the absolute values of $\tilde{X}$ and $\tilde{Y}$ have to be computed before the unsigned integer comparison algorithm is applied. Although the algorithm for comparing two signed integers in RNS is comprehensively presented in [Lu92], the implementation of sign detection algorithm and computations of absolute values of $\tilde{X}$ and $\tilde{Y}$ in residue domain are omitted.

In [Dim93], an algorithm called the Sum of Quotients Technique (SQT) was proposed. It is based on the partial reverse conversion approach. SQT relies on the construction of a monotonic growing diagonal function for computing the weighted representations of $(x_1, x_2, \dots, x_N)$ and $(y_1, y_2, \dots, y_N)$. The main weakness of this algorithm is the two relatively large modulo reduction operations involved in the weighted representation computation, which are known as the sum of quotients (SQ). In order to reduce the sizes of SQ modulus operation, an improved algorithm was proposed which is based on the grouping of two or more moduli from $(m_1, m_2, \dots, m_N)$ to obtain a new set of moduli known as “virtual” moduli. The residues with respect to the “virtual” moduli are computed before their weighted representations.

The magnitude comparison algorithm in [Wan99] was derived based on the New CRT II, which can be categorized as a partial reverse conversion approach. This algorithm treats two moduli as a baseline two-moduli set and decodes the residues one at a time to avoid large modulo operations. Assume that $X \equiv (x_1, x_2, x_3, x_4)$ and $Y \equiv (y_1, y_2, y_3, y_4)$ are the residue representations of two unsigned integers X and Y in a four-moduli set RNS $\{m_1, m_2, m_3, m_4\}$. Using (2.18), $X_{12} \equiv (x_1, x_2)$, $X_{34} \equiv (x_3, x_4)$, $Y_{12} \equiv (y_1, y_2)$ and $Y_{34} \equiv (y_3, y_4)$ are first computed. Then, by treating (X_{12}, X_{34}) and (Y_{12}, Y_{34}) as the residues of a two-moduli set $\{m_1m_2, m_3m_4\}$, $A_X = \left\| (m_1m_2)^{-1} \right\|_{m_3m_4} (X_{34} - X_{12}) \Big|_{m_3m_4}$ and $A_Y = \left\| (m_1m_2)^{-1} \right\|_{m_3m_4} (Y_{34} - Y_{12}) \Big|_{m_3m_4}$ are computed and compared. If $A_X > A_Y$, then $X > Y$ and vice versa. When $A_X = A_Y$, further comparisons between X_{12} and Y_{12} , and x_1 and y_1 are required.

Based on the parity check approach, an algorithm dedicated to unconventional moduli set formed by two pairs of conjugate moduli was proposed in [Sou07]. As the moduli of the target RNS $\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$ are not coprime, it has to rely on the New CRT III [Wan98]

for the computation of the parities of X , Y and $Z = |X - Y|_M$. The use of such moduli set is mainly motivated by its very fast computation of the sum-of-absolute differences in residue domain for motion estimation in video sequences. Unfortunately, the non-coprime moduli also complicate the design of reverse converter and residue arithmetic units. More severely, the dynamic range is only one third of the product of all moduli. This implies that a much larger value of n , and hence longer word lengths of residues, are required to cater for similar dynamic range as other special four-moduli sets that are built from equally arithmetic friendly but coprime moduli. Since the algorithm in [Sou07] deals only with unsigned magnitude comparison, this limitation adds further difficulty to sign detection and its extension to signed magnitude comparison.

In [Bi08], a new reverse conversion method called Mixed-Radix (MR) CRT was proposed for magnitude comparison in RNS. It possesses the advantages of the smaller modulo operations of MRC and the parallel execution of CRT. However, the weighted representations require many modular additions to compute and the numbers of additions grow with the size of the moduli set. Hence the algorithm is designed for the fundamental three-moduli set, $\{2^n-1, 2^n, 2^n+1\}$. It needs to compute three weighted digits from each residue representation before the comparison on these weighted digits can be carried out using three binary comparators.

The magnitude comparison algorithm of [Zar10] was a parity check method proposed for the moduli set $\{2^n-1, 2^n+1, 2^n+3\}$. Unlike the algorithms of [Lu92] and [Sou07], the parities of X , Y and $Z = |X - Y|_M$ are determined by checking their periodicities (odd or even) with respect to the modulus 2^n+3 and the parity of the residue of this modulus. As suggested in [Zar10], the periodicities of the integers can be pre-computed and stored in a read-only memory (ROM) module. However, ROM module is not scalable. Its size grows exponentially with n , and the data access time also increases with the size of the memory.

In [Tor16a], a magnitude comparison algorithm was proposed for the three-moduli set $\{2^n-1, 2^n, 2^n+1\}$. In this algorithm, the dynamic range is partitioned into equal subranges. Two subrange

identifiers denoted as $P_1(X)$, $P_2(X)$ for the integer X and $P_1(Y)$, $P_2(Y)$ for the integer Y are computed from the residue representations of the operands in comparison and compared. Assume that $X \equiv (x_1, x_2, x_3)$ and $Y \equiv (y_1, y_2, y_3)$ are the residue representations of two unsigned integers X and Y in the moduli set $\{2^{n-1}, 2^n, 2^{n+1}\}$. $P_2(X)$ is computed by applying the New CRT-I on (x_3, x_2) with respect to $\{2^{n+1}, 2^n\}$ as $X_{23} = x_3 + (2^n + 1) \lfloor x_2 - x_3 \rfloor_{2^n}$ where $P_2(X) = \lfloor x_2 - x_3 \rfloor_{2^n}$. In a similar manner, $P_1(X)$ is computed by applying the two-moduli New CRT-I on (X_{23}, x_1) with respect to $\{(2^n+1) \times 2^n, 2^{n-1}\}$ as $X = X_{23} + 2^n \times (2^n + 1) \lfloor 2^{n-1}(x_1 - X_{23}) \rfloor_{2^{n-1}}$, where $P_1(X) = \lfloor 2^{n-1}(x_1 - X_{23}) \rfloor_{2^{n-1}}$. Similarly, $P_1(Y)$ and $P_2(Y)$ can be computed.

If $P_1(X) > P_1(Y)$, then $X > Y$ and vice versa. When $P_1(X) = P_1(Y)$, further comparisons between $P_2(X)$ and $P_2(Y)$, and x_3 and y_3 are required. In addition, this method falls under the partial reverse conversion approaches as these subrange identifiers can be deemed as MRC coefficients. The most recent magnitude comparison algorithm based on partial reverse conversion process was proposed by the same authors [Tor16b] for the four moduli sets $\{2^{n \pm 1}, 2^n, m\}$ with $m \in [2^{n+1} \pm 1, 2^{n-1} - 1]$. Assume that $X \equiv (x_1, x_2, x_3, x_4)$ and $Y \equiv (y_1, y_2, y_3, y_4)$ are the residue representations of two unsigned integers X and Y respectively in a four-moduli set RNS $\{2^{n-1}, 2^n, 2^{n+1}, m\}$. These residue representations are represented in the weighted form by the New CRT-I as $X = x_1 + M_1 \times X_1$, where x_1 and X_1 are the residues with respect to 2^n and $M_1 = (2^n \pm 1) \times m$, respectively. When $X_1 > Y_1$, the unsigned integer X is greater than Y and vice versa. When $X_1 = Y_1$, further comparison between x_1 and y_1 is required.

Chapter 3

New Algorithms for Hardware-efficient Implementation of Sign Detection in $\{2^n-1, 2^n, 2^n+1\}$ RNS

3.1 Introduction

RNS is gaining increasing popularity in very large scale integration (VLSI) implementation of application-specific DSPs. This is in part due to its ability to accelerate and reduce the power consumptions of crucial and frequently used data path operations by subword-level parallelism and modularity, and in part due to the ease of realizing modulo operations using moduli of the forms 2^n and $2^n \pm 1$. Modular $2^n \pm 1$ arithmetic properties have been exploited with arithmetic structures like diminished-1, sparse carry chain, Kogge-Stone adder, etc., in [Mur12], [Ver12], [Ver10], [Did13] to reduce the implementation complexity of modulo addition, subtraction and multiplication for these special moduli to an extent that is comparable to their two's complement number system counterparts. These advancements have given rise to the extensive use and continual successes in exploiting the balanced three moduli set $\{2^n-1, 2^n, 2^n+1\}$ for the implementation of many new and existing DSP algorithms, including fast Fourier transform, discrete wavelet transform, finite and infinite impulse response filters and digital image processing [Con04], [Vas13], [Liu04], [Baj11]. In fact, the difficulties associated with the implementation of non-modular operations, such as scaling and reverse conversion from the residue to binary representation, have largely been resolved for this three moduli set [Tay13], [Cha11], [Wan00b].

Notwithstanding the hardware efficiency of its individual residue arithmetic operations, as well as its forward and reverse converters, some fundamental operations, such as sign detection, magnitude comparison and overflow detection, for this moduli set remain slow and expensive. These operations are difficult to parallelize as they require the combination of multiple residue values to compute. To reduce the computational complexity, lookup tables are often used to

store the pre-computed orthogonal projections of the numbers of interest. Unfortunately, memory based approaches are difficult to pipeline. The size and number of lookup tables, as well as their access time also grow with the size of the moduli. Among these operations, sign detection is an example of a less focused problem for this moduli set. Despite its importance as a pre-processing operation and an integral component of other inter-modulo operations like magnitude comparison and overflow detection [Tay13], [Bro97], only a handful of solutions are found in the literature.

In this chapter, two new scaling-assisted sign detection algorithms are proposed for the three moduli set $\{2^n-1, 2^n, 2^n+1\}$. The first algorithm is developed based on modified CRT with a scaling factor of 2^n while the second algorithm is formulated by the New CRT-I with a different scaling factor of $2^{2n}-1$. In the first algorithm, to avoid the complexity of a full reverse conversion, the segment $[0, 2^{3n-1}-2^{n-1}) \cup [2^{3n-1}, 2^{3n}-2^n)$ of the dynamic range is treated separately and processed in parallel with the rest of the dynamic range for sign detection in the residue representation. The proposed algorithm is mapped to an efficient architecture which is much faster and area efficient than using the efficient reverse converter of $\{2^n-1, 2^n, 2^n+1\}$ RNS, which has been widely adopted for sign detection. In the second algorithm, New CRT-I is exploited to greatly simplify the $2^{2n}-1$ scaling of residue representation into addends that can be readily obtained by circular left shifted residues at no logic cost. The reduced dynamic range enables the sign of an integer to be computed directly from the most significant bit (MSB) of the scaled residues with a heavily stripped-down version of reverse converter. A side benefit of both proposed sign detectors is that they can also be used as a scaler.

3.2 Scaling-assisted Sign Detection Method I

3.2.1 Proposed Algorithm

Let (x_1, x_2, x_3) be the residue representation of a signed integer X for $\{m_1, m_2, m_3\} = \{2^n-1, 2^n, 2^n+1\}$ and $\hat{X}$ be the integer obtained by the residue-to-binary conversion. As the dynamic range $M = 2^{3n}-2^n$ is even, $\text{sign}(X)$ is equal to the MSB of $\hat{X}$, i.e., $\text{sign}(X) = \text{MSB}$

($\hat{X}$) if $\hat{X} \notin [2^{3n-1} - 2^{n-1}, 2^{3n-1} - 1]$. If $\hat{X}$ falls in $[2^{3n-1} - 2^{n-1}, 2^{3n-1} - 1]$, MSB ($\hat{X}$) = 0 but this sub range lies on the upper half of the dynamic range where $\text{sign}(X) = 1$. To correctly obtain $\text{sign}(X)$ from (x_1, x_2, x_3) , the algorithm is divided into two cases:

Case (a): $\hat{X} \in [0, 2^{3n-1} - 2^{n-1}) \cup [2^{3n-1}, 2^{3n} - 2^n)$

Instead of performing a complete reverse conversion involving large modular $2^{3n} - 2^n$ operations by CRT, $\hat{X}$ can be decoded by using the new modified CRT as follows:

$$\begin{aligned}\hat{X} &= x_2 + m_2 \left| \sum_{i=1}^3 w_i x'_i \right|_{m_1 m_3} \\ &= x_2 + 2^n \times \left| w_1 x'_1 + w_2 x'_2 + w_3 x'_3 \right|_{2^{2n}-1}\end{aligned}\quad (3.1)$$

To compute $w_1, w_2, w_3, x'_1, x'_2$ and x'_3 , M_1, M_2 and M_3 as well as their multiplicative inverses modulo m_1, m_2 and m_3 need to be determined.

$$M_1 = m_2 \times m_3 = 2^n \times (2^n + 1) \text{ and } \left| M_1^{-1} \right|_{2^n-1} = 2^{n-1};$$

$$M_2 = m_1 \times m_3 = 2^{2n} - 1 \text{ and } \left| M_2^{-1} \right|_{m_2} = \left| (2^{2n} - 1)^{-1} \right|_{2^n} = -1;$$

$$M_3 = m_1 \times m_2 = 2^n \times (2^n - 1) \text{ and } \left| M_3^{-1} \right|_{2^n+1} = -2^{n-1}.$$

According to (2.19),

$$w_1 = \frac{M_1}{m_2} = \frac{2^n (2^n + 1)}{2^n} = 2^n + 1 \text{ and } x'_1 = 2^{n-1} x_1 \quad (3.2)$$

$$w_2 = \frac{M_2 \left| M_2^{-1} \right|_{m_2} - 1}{m_2} = \frac{(2^{2n} - 1)(-1) - 1}{2^n} = -2^n \text{ and } x'_2 = x_2 \quad (3.3)$$

$$w_3 = \frac{M_3}{m_2} = \frac{2^n (2^n - 1)}{2^n} = 2^n - 1 \text{ and } x'_3 = -2^{n-1} x_3 \quad (3.4)$$

By substituting (3.2), (3.3) and (3.4) into (3.1),

$$\hat{X} = x_2 + 2^n \left| (2^n + 1) 2^{n-1} x_1 - 2^n x_2 - (2^n - 1) 2^{n-1} x_3 \right|_{2^{2n}-1} \quad (3.5)$$

By scaling (3.5) by a factor of 2^n ,

$$Y = \left\lfloor \frac{\hat{X}}{2^n} \right\rfloor = \left\lfloor \frac{x_2}{2^n} \right\rfloor + \left\lfloor (2^n + 1)2^{n-1}x_1 - 2^n x_2 - (2^n - 1)2^{n-1}x_3 \right\rfloor_{2^{2n}-1} \quad (3.6)$$

Since $x_2 < 2^n$,

$$Y = \left\lfloor (2^n + 1)2^{n-1}x_1 - 2^n x_2 - (2^n - 1)2^{n-1}x_3 \right\rfloor_{2^{2n}-1} \quad (3.7)$$

Since $\hat{X}$ is $3n$ bits long, from (3.6), it can be seen that the most significant $2n$ bits of $\hat{X}$ depend only on Y . By expressing Y as:

$$\begin{aligned} Y &= \left\lfloor 2^{n-1} \left\{ -2x_2 + (2^n + 1)x_1 - (2^n - 1)x_3 \right\} \right\rfloor_{2^{2n}-1} \\ &= \left\lfloor 2^{n-1} Q \right\rfloor_{2^{2n}-1} \end{aligned} \quad (3.8)$$

As a residue of modulus $2^{2n}-1$, Q can be represented by a $2n$ -bit binary number $q_{2n-1}q_{2n-2} \cdots q_0$.

According to the circular left shift property of (2.9), Y can be implemented by $CLS_{2n}(Q, n-1) = q_n q_{n-1} \cdots q_0 q_{2n-1} \cdots q_{n+1}$. Since $\text{sign}_a(X) = \text{MSB}(\hat{X}) = \text{MSB}(Y)$, the sign of X can be detected from q_n .

Case (b): $\hat{X} \in [2^{3n-1} - 2^{n-1}, 2^{3n-1} - 1]$

When $\hat{X}$ falls in this range, it is observed that the residues of x_1 and x_3 are always equal. In addition, the difference of x_2 and x_1 is constant and equal to 2^{n-1} . For this case,

$$\text{sign}_b(X) = \begin{cases} 1 & \text{If } x_2 - x_1 = 2^{n-1} \text{ and } x_1 = x_3 \\ 0 & \text{Otherwise} \end{cases} \quad (3.9)$$

3.2.2 Proposed Hardware Architecture

The residues x_1 , x_2 and x_3 are represented in binary form as follows: $x_1 = x_{1,n-1}x_{1,n-2} \cdots x_{1,0}$,

$x_2 = x_{2,n-1}x_{2,n-2} \cdots x_{2,0}$ and $x_3 = x_{3,n}x_{3,n-1} \cdots x_{3,0}$, where $x_{i,j} \in \{0,1\}$ denotes the j -th bit of x_i .

Case (a): $\hat{X} \in [0, 2^{3n-1} - 2^{n-1}) \cup [2^{3n-1}, 2^{3n} - 2^n)$

There are four terms in the modulo $2^{2n}-1$ sum of Q in (3.8), which are labeled as u_1 , u_2 , u_3 and u_4 as follows:

$$|Q|_{2^{2n}-1} = \left| - \left(\underbrace{2x_2}_{u_1} + \underbrace{2^n x_3}_{u_2} + \underbrace{\bar{x}_3}_{u_3} \right) + \underbrace{(2^n + 1)x_1}_{u_4} \right|_{2^{2n}-1} \quad (3.10)$$

By applying the circular left shift properties of (2.9) and (2.10),

$$\begin{aligned} u_1 &= |2x_2|_{2^{2n}-1} = CLS_{2n}(x_2, 1) \\ &= \underbrace{0 \cdots 0}_{n-1} \underbrace{x_{2,n-1} \cdots x_{2,0}}_n \underbrace{0}_1 \end{aligned} \quad (3.11)$$

$$\begin{aligned} u_2 &= |2^n x_3|_{2^{2n}-1} = CLS_{2n}(x_3, n) \\ &= \underbrace{x_{3,n-1} \cdots x_{3,0}}_n \underbrace{0 \cdots 0}_{n-1} \underbrace{x_{3,n}}_1 \end{aligned} \quad (3.12)$$

$$u_3 = \underbrace{1 \cdots 1}_{n-1} \underbrace{\bar{x}_{3,n} \cdots \bar{x}_{3,0}}_{n+1} \quad (3.13)$$

$$u_4 = |2^n x_1 + x_1|_{2^{2n}-1} = \underbrace{x_{1,n-1} \cdots x_{1,0} x_{1,n-1} \cdots x_{1,0}}_{2n} \quad (3.14)$$

The three operands u_1 , u_2 and u_3 can be reduced to two operands in the modulo sum of (3.10), as illustrated in Figure 3.1. In Figure 3.1(a), the $(n-1)$ -bit string of zeros of u_2 is swapped with the $(n-1)$ -bit string $x_{2,n-2} \cdots x_{2,0}$ of u_1 to produce the new operands, u'_1 and u'_2 , in Figure 3.1(b). All but the bit $x_{2,n-1}$ in u'_1 are zeros. This leads to a sum bit of $\bar{x}_{3,n} \oplus x_{2,n-1}$ and a carry bit $t = \bar{x}_{3,n} \wedge x_{2,n-1}$ from the addition of $x_{2,n-1}$ of u'_1 and $\bar{x}_{3,n}$ of u_3 at the n -th bit position in the modulo $2^{2n}-1$ addition of u'_1 and u_3 . Since the most significant $n-1$ bits of u_3 are all ones, the carry bit t will be rippled all the ways (the addition of '1' and t yields a sum bit of $1 \oplus t = \bar{t}$ and a carry bit of t) through to become an end-around carry (EAC) to be added into the least significant bit (LSB) position of the modulo $2^{2n}-1$ sum. Since $x_{3,n}$ is zero when $t = 1$, this EAC bit t can be added by replacing the LSB $x_{3,n}$ of u'_2 by $x_{3,n} \vee t$. The final reduced operands, u'_1 and u'_2 , are shown in Figure 3.1(c).

$$\begin{array}{llll}
u_1 : & \overbrace{00 \cdots 0}^{n-1} & x_{2,n-1} & \overbrace{x_{2,n-2} x_{2,n-3} \cdots x_{2,0}}^{n-1} & 0 \\
u_2 : & x_{3,n-1} x_{3,n-2} \cdots x_{3,1} & x_{3,0} & \overbrace{0 \cdots 0}^{n-1} & x_{3,n}
\end{array}$$

(a)

$$\begin{array}{llll}
u_3 : & \overbrace{11 \cdots 11}^{n-1} & \boxed{\bar{x}_{3,n}} & \overbrace{\bar{x}_{3,n-1} \bar{x}_{3,n-2} \cdots \bar{x}_{3,1}}^{n-1} & \bar{x}_{3,0} \\
u'_1 : & 00 \cdots 0 & \boxed{x_{2,n-1}} & 00 \cdots 0 & 0 \\
u'_2 : & x_{3,n-1} x_{3,n-2} \cdots x_{3,1} & x_{3,0} & x_{2,n-2} x_{2,n-3} \cdots x_{2,0} & \boxed{x_{3,n}}
\end{array}$$

(b)

$$\begin{array}{llll}
u'_1 : & \overbrace{t \bar{t} \cdots t}^{n-1} & \bar{x}_{3,n} \oplus x_{2,n-1} & \overbrace{\bar{x}_{3,n-1} \bar{x}_{3,n-2} \cdots \bar{x}_{3,1}}^{n-1} & \bar{x}_{3,0} \\
u'_2 : & x_{3,n-1} x_{3,n-2} \cdots x_{3,1} & x_{3,0} & x_{2,n-2} x_{2,n-3} \cdots x_{2,0} & x_{3,n} \vee t
\end{array}$$

(c)

Figure 3.1: Reduction of addends

Upon reduction of operands, (3.10) becomes

$$|Q|_{2^{n+1}} = \left| \bar{u}'_1 + \bar{u}'_2 + u_4 \right|_{2^{2n}-1} \Big|_{2^{n+1}} \quad (3.15)$$

Let $A = a_{2n-1} a_{2n-2} \cdots a_0$ and $B = b_{2n-1} b_{2n-2} \cdots b_0$ be the $2n$ -bit sum and carry vectors, respectively

of a modulo $2^{2n} - 1$ carry save adder (CSA) with end around carry (EAC). Then

$$\left| \bar{u}'_1 + \bar{u}'_2 + u_4 \right|_{2^{2n}-1} = \begin{cases} A + B & \text{if } \bar{u}'_1 + \bar{u}'_2 + u_4 < 2^{2n} - 1 \\ A + B - (2^{2n} - 1) & \text{if } \bar{u}'_1 + \bar{u}'_2 + u_4 \geq 2^{2n} - 1 \end{cases} \quad (3.16)$$

Since $\left| 2^{2n} - 1 \right|_{2^{n+1}} = -1$, the sum in (3.15) can be reduced to $n+1$ bits by:

$$|Q|_{2^{n+1}} = \left| A + B - c_{in} (2^{2n} - 1) \right|_{2^{n+1}} = \left| A + B + c_{in} \right|_{2^{n+1}} \quad (3.17)$$

where $c_{in} = 1$ when $A + B \geq 2^{2n} - 1$ and 0 otherwise.

The carry-in signal c_{in} can be computed by the circuit in Figure 3.2. When $A + B \geq 2^{2n}$, c_{in} can be generated by a parallel prefix carry generator. When $A + B = 2^{2n} - 1 = \underbrace{11 \cdots 11}_{2n}$, it can be

computed by ANDing all bits of $A \oplus B$.

Figure 3.2: Generation of c_{in} signal.

Although A and B are both $2n$ bits wide, only the least significant $n + 1$ bits are required for the modulo 2^{n+1} addition in (3.17). Their modulo 2^{n+1} addition with the carry-in c_{in} can be performed by an $(n + 1)$ -th bit carry propagation adder (CPA). Since only q_n is needed to detect the sign of X , only the carry look ahead network of an n -bit CPA is required for the computation of the carry bit c_{n-1} , i.e.,

$$c_{n-1} = g_{n-1} \vee p_{n-1} \wedge (g_{n-2} \vee p_{n-2} \wedge (\cdots \vee p_1 \wedge (g_0 \vee p_0 \wedge c_{in}))) \quad (3.18)$$

where $g_i = a_i \wedge b_i$ and $p_i = a_i \vee b_i$.

Then, c_{n-1} is XORed with bits a_n and b_n of the input operands A and B , respectively to produce q_n , i.e.,

$$\text{sign}_a(X) = q_n = a_n \oplus b_n \oplus c_{n-1} \quad (3.19)$$

Case (b): $\hat{X} \in [2^{3n-1} - 2^{n-1}, 2^{3n-1} - 1]$

Since the MSB of x_1 is zero in this range, the condition $x_2 - x_1 = 2^{n-1}$ in (3.9) can be simplified

to $x_2 = x_1 + 2^{n-1} = \underbrace{0x_{1,n-2} \cdots x_{1,0}}_n + \underbrace{10 \cdots 0}_n = \underbrace{1x_{1,n-2} \cdots x_{1,0}}_n = x'_1$. This condition can be detected if

$x_2 \oplus x'_1$ is an n -bit string of all zeros, i.e., $e_1 = \bigvee_{i=0}^{n-1} (x_2 \oplus x'_1)_i = 0$, where ‘ $\vee$ ’ denotes the Boolean

OR operator. Similarly, the condition of $x_1 = x_3$ can be detected if $e_2 = \bigvee_{i=0}^{n-1} (x_1 \oplus x_3)_i = 0$. If both

conditions are fulfilled, then

$$\text{sign}_b(X) = \bar{e}_1 \wedge \bar{e}_2 \quad (3.20)$$

where ‘ $\wedge$ ’ denote the bitwise AND operation.

The overall sign detector circuit is shown in Figure 3.3, where $\text{sign}(X) = \text{sign}_a(X) \vee \text{sign}_b(X)$.

Example 3.1: For $n = 4$, the moduli set is $\{15, 16, 17\}$ and $M = 4080$. Let $\hat{X} \equiv (6, 14, 1)$. The binary representations of the residues are: $x_1 = 0110$, $x_2 = 1110$ and $x_3 = 00001$. For case (a), from (3.14), $u_4 = 01100110$. $t = \bar{x}_{3,4} \wedge x_{2,3} = 1 \wedge 1 = 1$. According to Figure 3.1(c), $u'_1 = 00001110$ and $u'_2 = 00011101$. A and B are computed by an 8-bit CSA with EAC as shown in Fig. 3.4. From Figure 3.2, the carry output of $A + B$ is 1 and $01110101 \oplus 11000101 = 10110000$. Hence, $c_{in} = 1$. From (3.18), $c_3 = g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0) \vee (p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge c_{in})$. Since $c_{in} = 1$, $g_0 = 1$, $g_1 = 0$, $g_2 = 1$, $g_3 = 0$, $p_0 = 1$, $p_1 = 0$, $p_2 = 1$ and $p_3 = 0$, $c_3 = 0 \vee (0 \wedge 1) \vee (0 \wedge 1 \wedge 0) \vee (0 \wedge 1 \wedge 0 \wedge 1) \vee (0 \wedge 1 \wedge 0 \wedge 1 \wedge 1) = 0$. From (3.19), $\text{sign}_a(X) = q_4 = a_4 \oplus b_4 \oplus c_3 = 1 \oplus 0 \oplus 0 = 1$. For case (b), since $x_2 \oplus x'_1 = 1110 \oplus 1110 = 0000$, $e_1 = 0$. Since $x_1 \oplus x_3 = 0111$, $e_2 = 1$. From (3.20), $\text{sign}_b(X) = \bar{e}_1 \wedge \bar{e}_2 = 1 \wedge 0 = 0$. Since $\text{sign}(X) = 1 \vee 0 = 1$, X is negative. Indeed, $\hat{X} = 3486 \geq 2040$.

Figure 3.3: Proposed sign detection circuit.

$\vec{u}'_1$		1	1	1	1	0	0	0	1
$\vec{u}'_2$		1	1	1	0	0	0	1	0
u_4	+	0	1	1	0	0	1	1	0
A (Sum)		0	1	1	1	0	1	0	1
B (Carry)	1	1	1	0	0	0	1	0	
									1
A (Sum)		0	1	1	1	0	1	0	1
B (Carry)		1	1	0	0	0	1	0	1

Figure 3.4: Calculation of A and B by modulo 2^8-1 CSA with EAC.

3.3 Scaling-assisted Sign Detection Method II

3.3.1 Proposed Algorithm

Let (x_1, x_2, x_3) be the residue representation of an integer X with respect to the moduli set $\{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^n + 1\}$. Since the dynamic range M of this moduli set can be factored into 2^n and $2^{2n} - 1$, the sizes of the modulo operations required for detecting the sign of $\hat{X}$ from its

equivalent residue representation of X can be substantially reduced by scaling (x_1, x_2, x_3) in the residue domain by $2^{2n} - 1$. This will map the lower half range $[0, 2^{3n-1} - 2^{n-1})$ of X to the lower half range $[0, 2^{n-1})$ of the scaled integer Y and the upper half range $[2^{3n-1} - 2^{n-1}, 2^{3n} - 2^n)$ of X to the upper half range $[2^{n-1}, 2^n)$ of Y , as shown in Figure 3.5. By shrinking the dynamic range from $M = 2^{3n} - 2^{2n}$ to $M' = 2^n$, its half range can be easily detected from the MSB of the scaled integer Y .

Figure 3.5: Mapping of the half ranges of integer X in $[0, M)$ to the half ranges of its scaled integer Y in $[0, M')$.

This new concept of sign detection in $\{2^n - 1, 2^n, 2^n + 1\}$ can be made very efficient provided that scaling by $2^{2n} - 1$ as well as the reverse conversion of the scaled residues into Y can be computed efficiently from the residues, x_1, x_2 and x_3 . As only the MSB of Y is needed for the sign detection of $\hat{X}$, a full reverse conversion from (x_1, x_2, x_3) is not required.

To simplify the scaling by $2^{2n} - 1$ in the residue domain, the New CRT [Wan98], also known as CRT-I, is used to convert X into a weighted sum of its residues modulo $2^{2n} - 1$. According to CRT-I,

$$X = x_3 + m_3 \left| k_1 (x_1 - x_3) + k_2 m_1 (x_2 - x_1) \right|_{m_1 m_2} \quad (3.21)$$

where $k_1 m_3 = 1|_{m_1 m_2}$ and $k_2 m_3 m_1 = 1|_{m_2}$.

With $m_1 = 2^n - 1$, $m_2 = 2^n$ and $m_3 = 2^n + 1$, X can be written as

$$X = x_3 + (2^n + 1) \left| k_1 (x_1 - x_3) + k_2 (2^n - 1) (x_2 - x_1) \right|_{2^n(2^n - 1)} \quad (3.22)$$

It can be proved that the multiplicative inverses of $2^n + 1|_{2^n(2^n - 1)}$ and $2^{2n} - 1|_{2^n}$ are given by $k_1 =$

$2^{2n-1} - (2^n - 1)$ and $k_2 = -1$, respectively. These closed form expressions of k_1 and k_2 are

proven below.

Proof of $k_1 = 2^{2n-1} - (2^n - 1)$:

$$\begin{aligned}
\left| k_1 (2^n + 1) \right|_{2^n(2^n-1)} &= \left| \left[2^{2n-1} - (2^n - 1) \right] (2^n + 1) \right|_{2^n(2^n-1)} \\
&= \left| 2^{2n-1} (2^n + 1) - (2^{2n} - 1) \right|_{2^n(2^n-1)} \\
&= \left| 2^{3n-1} - 2^{2n-1} + 1 \right|_{2^n(2^n-1)} \\
&= \left| 2^{2n-1} (2^n - 1) + 1 \right|_{2^n(2^n-1)} = 1
\end{aligned}$$

Q.E.D.

Proof of $k_2 = -1$:

$$\left| k_2 (2^{2n} - 1) \right|_{2^n} = \left| -1 \times (2^{2n} - 1) \right|_{2^n} = \left| -2^{2n} + 1 \right|_{2^n} = 1$$

Q.E.D.

By substituting the values of k_1 and k_2 into (3.22),

$$\begin{aligned}
X &= x_3 + (2^n + 1) \left\{ 2^{2n-1} - (2^n - 1) \right\} (x_1 - x_3) - (2^n - 1) (x_2 - x_1) \Big|_{2^n(2^n-1)} \\
&= x_3 + (2^n + 1) \left| 2^{2n-1} (x_1 - x_3) - (2^n - 1) (x_2 - x_3) \right|_{2^n(2^n-1)}
\end{aligned} \tag{3.23}$$

By scaling X by $2^{2n} - 1$, the scaled integer Y can be obtained by

$$Y = \left\lfloor \frac{X}{2^{2n} - 1} \right\rfloor = \left\lfloor \frac{x_3}{2^{2n} - 1} \right\rfloor + \left\lfloor \frac{(2^n + 1)Z}{2^{2n} - 1} \right\rfloor \tag{3.24}$$

where $Z = \left| 2^{2n-1} (x_1 - x_3) - (2^n - 1) (x_2 - x_3) \right|_{2^n(2^n-1)}$.

Since $x_3 \in [0, 2^n]$, $x_3 < 2^{2n} - 1$. Therefore, $\left\lfloor \frac{x_3}{2^{2n} - 1} \right\rfloor = 0$ and Y can be written as

$$\begin{aligned}
Y &= \left\lfloor \frac{(2^n + 1)Z}{2^{2n} - 1} \right\rfloor = \left\lfloor \frac{Z}{2^n - 1} \right\rfloor \\
&= \left\lfloor \frac{\left| 2^{2n-1} (x_1 - x_3) - (2^n - 1) (x_2 - x_3) \right|_{2^n(2^n-1)}}{(2^n - 1)} \right\rfloor
\end{aligned} \tag{3.25}$$

As $\left\lfloor \frac{|x|_{m_1 m_2}}{m_1} \right\rfloor = \left\lfloor \frac{x}{m_1} \right\rfloor_{m_2}$ from [Cha11], (3.25) can be rewritten as

$$\begin{aligned} Y &= \left\lfloor \frac{2^{2^{n-1}}(x_1 - x_3) - (2^n - 1)(x_2 - x_3)}{(2^n - 1)} \right\rfloor_{2^n} \\ &= \left\lfloor \frac{2^{2^{n-1}}(x_1 - x_3)}{2^n - 1} \right\rfloor + x_3 - x_2 \Big|_{2^n} \end{aligned} \quad (3.26)$$

Let $H = 2^{2^{n-1}}(x_1 - x_3)$. Since $H = m \left\lfloor \frac{H}{m} \right\rfloor + |H|_m$ for any integer H and m ,

$$H = (2^n - 1) \left\lfloor \frac{H}{2^n - 1} \right\rfloor + |H|_{2^n - 1} \quad (3.27)$$

Taking mod 2^n operation on both sides of (3.27),

$$|H|_{2^n} = \left| (2^n - 1) \left\lfloor \frac{H}{2^n - 1} \right\rfloor \right|_{2^n} + |H|_{2^n - 1} \Big|_{2^n} \quad (3.28)$$

Since $|H|_{2^n} = \left| 2^{2^{n-1}}(x_1 - x_3) \right|_{2^n} = 0$ and $|2^n - 1|_{2^n} = -1$,

$$\left\lfloor \frac{H}{2^n - 1} \right\rfloor_{2^n} = |H|_{2^n - 1} \Big|_{2^n} \quad (3.29)$$

By substituting (3.29) into (3.26),

$$\begin{aligned} Y &= \left| |H|_{2^n - 1} + x_3 - x_2 \right|_{2^n} \\ &= \left| 2^{2^{n-1}}(x_1 - x_3) \right|_{2^n - 1} + x_3 - x_2 \Big|_{2^n} \end{aligned} \quad (3.30)$$

If $Y \in [0, 2^{n-1})$, X falls in the lower half range of M and (x_1, x_2, x_3) represents a positive integer, i.e., $\hat{X} \geq 0$. Otherwise, if $Y \in [2^{n-1}, 2^n)$, X falls in the upper half range of M and (x_1, x_2, x_3) represents a negative integer, i.e., $\hat{X} < 0$.

3.3.2 Proposed Hardware Architecture

The residues, x_1 , x_2 and x_3 , can be represented in binary form as $x_1 = x_{1,n-1}x_{1,n-2} \cdots x_{1,0}$, $x_2 = x_{2,n-1}x_{2,n-2} \cdots x_{2,0}$ and $x_3 = x_{3,n}x_{3,n-1} \cdots x_{3,0}$, respectively, where $x_{i,j}$ denotes the j -th bit of the residue x_i . The binary vectors of x_1 and x_2 are of n bits but the binary vector of x_3 is of $n + 1$ bits. In (3.30), one of the terms in the modulo $2^n - 1$ sum involves the operation $\left| -2^{2n-1}x_3 \right|_{2^n-1}$, which cannot be directly implemented by *Property 2.12* since x_3 has $n + 1$ bits. To apply the circular left shift property on the one's complement of x_3 as in (2.10), x_3 is expressed as $x_3 = 2^n x_{3,n} + x_{3,n-1}x_{3,n-2} \cdots x_{3,0}$. Since $\left| 2^n x_{3,n} \right|_{2^n-1} = x_{3,n}$, the MSB $x_{3,n}$ of x_3 can be logically OR with $x_{3,0}$ to form an n -bit binary vector $x'_3 = \left| x_3 \right|_{2^n-1} = \left| x_{3,n-1}x_{3,n-2} \cdots x'_{3,0} \right|_{2^n-1}$, where $x'_{3,0} = x_{3,0} \vee x_{3,n}$ and ' $\vee$ ' denotes the logical OR operator.

$\left| H \right|_{2^n-1}$ in (3.30) can then be implemented using the circular left shift operations of *Properties 2.11* and *2.12* to obtain

$$Y = \left| u_1 + u_2 \right|_{2^n-1} + x_3 - x_2 \Big|_{2^n} \quad (3.31)$$

where

$$u_1 = \left| 2^{2n-1} x_1 \right|_{2^n-1} = CLS_n(x_1, 2n-1) = \underbrace{x_{1,0}}_1 \underbrace{x_{1,n-1} \cdots x_{1,1}}_{n-1} \quad (3.32)$$

$$u_2 = \left| 2^{2n-1} \bar{x}_3 \right|_{2^n-1} = CLS_n(\bar{x}_3, 2n-1) = \underbrace{\bar{x}'_{3,0}}_1 \underbrace{\bar{x}_{3,n-1} \cdots \bar{x}_{3,1}}_{n-1} \quad (3.33)$$

The term $\left| u_1 + u_2 \right|_{2^n-1}$ can be expressed as

$$\left| u_1 + u_2 \right|_{2^n-1} = \begin{cases} \left| u_1 + u_2 + 1 \right|_{2^n} & \text{if } u_1 + u_2 \geq 2^n - 1 \\ u_1 + u_2 & \text{otherwise} \end{cases} \quad (3.34)$$

Hence, $\left| u_1 + u_2 \right|_{2^n-1} \Big|_{2^n} = \left| u_1 + u_2 + c_{in} \right|_{2^n}$, where $c_{in} \in \{0, 1\}$. As $\left| -x_2 \right|_{2^n} = 2^n - x_2 = \bar{x}_2 + 1$, (3.31)

can be written as:

$$Y = \left| u_1 + u_2 + c_{in} + x_3 + \bar{x}_2 + 1 \right|_{2^n} \quad (3.35)$$

The generation of the carry-in signal c_{in} is shown in Figure 3.6. The condition $u_1 + u_2 \geq 2^n$ is detected by $C_1 = 1$ and the signal C_1 can be generated by parallel prefix operators [Ver12]. As an example, the carry signal C_1 for $n = 8$ can be generated by the circuit shown in Figure 3.7. The condition $u_1 + u_2 = 2^n - 1 = \underbrace{11 \dots 11}_n$ can be detected by $C_2 = 1$. C_2 is generated by $\bar{w} \wedge v$, where $w = \bigwedge_{i=0}^{n-1} g_i$ and $v = p_{n-1:0} = \bigwedge_{i=0}^{n-1} p_i$, where ‘ $\wedge$ ’ denotes a logical AND operator. The signals g_i and $p_{n-1:0}$ have already been generated in the computation of C_1 . Consequently, the condition $u_1 + u_2 \geq 2^n - 1$ for $c_{in} = 1$ can be detected by

$$c_{in} = C_1 \vee C_2 \quad (3.36)$$

The two addends, u_2 and x_3 , in (3.35) can be further simplified as follows:

$$\begin{aligned} |u_2 + x_3|_{2^n} &= |2u_2|_{2^n} - u_2 + |x_3|_{2^n}|_{2^n} \\ &= \left| \underbrace{\bar{x}_{3,n-1}\bar{x}_{3,n-2}\cdots\bar{x}_{3,1}}_n 0 + |x_3|_{2^n} - u_2 \right|_{2^n} \\ &= \left| \underbrace{\bar{x}_{3,n-1}\bar{x}_{3,n-2}\cdots\bar{x}_{3,1}\bar{x}_{3,0}}_n - \bar{x}_{3,0} + |x_3|_{2^n} - u_2 \right|_{2^n} \\ &= \left| \bar{x}_3|_{2^n} + |x_3|_{2^n} - u_2 - \bar{x}_{3,0} \right|_{2^n} \\ &= \left| 2^n - 1 - u_2 - \bar{x}_{3,0} \right|_{2^n} = \left| \bar{u}_2 - \bar{x}_{3,0} \right|_{2^n} \\ &= \left| x'_{3,0}x_{3,n-1}x_{3,n-2}\cdots x_{3,1} - \bar{x}_{3,0} \right|_{2^n} \end{aligned} \quad (3.37)$$

When $x_{3,n} = 0$, $x'_{3,0} = x_{3,0} \vee 0 = x_{3,0}$. Then,

$$|u_2 + x_3|_{2^n} = |x_{3,0}x_{3,n-1}x_{3,n-2}\cdots x_{3,1} - \bar{x}_{3,0}|_{2^n} \quad (3.38)$$

When $x_{3,n} = 1$, since $x_3 \in [0, 2^n]$, $x_{3,n-1}x_{3,n-2}\cdots x_{3,0} = 00\dots 0$. Hence, $x'_{3,0} = x_{3,0} \vee x_{3,n} = 1$ and

$$\begin{aligned} |u_2 + x_3|_{2^n} &= \left| \underbrace{100\dots 0}_n - 1 \right|_{2^n} = \left| \underbrace{011\dots 1}_n \right|_{2^n} \\ &= \left| \underbrace{x_{3,0}x_{3,n}x_{3,n}\cdots x_{3,n}}_n - \bar{x}_{3,0} + x_{3,n} \right|_{2^n} \\ &= \left| \underbrace{x_{3,n}00\dots 0}_n - \bar{x}_{3,0} \right|_{2^n} \end{aligned} \quad (3.39)$$

Figure 3.6: Generation of carry-in signal c_{in} .

Figure 3.7: Example of the generation of the carry signal C_1 and v for $n = 8$.

To satisfy both (3.38) and (3.39),

$$|u_2 + x_3|_{2^n} = |u_3 - \bar{x}_{3,0}|_{2^n} \quad (3.40)$$

where the n -bit binary vector u_3 is given by

$$u_3 = (x_{3,0} \vee x_{3,n}) x_{3,n-1} x_{3,n-2} \cdots x_{3,1} \quad (3.41)$$

By substituting (3.40) into (3.35),

$$Y = |u_1 + u_3 + \bar{x}_2 + c_{in} + 1 - \bar{x}_{3,0}|_{2^n} \quad (3.42)$$

If $x_{3,0} = 1$, $1 - \bar{x}_{3,0} = 1$ and if $x_{3,0} = 0$, $1 - \bar{x}_{3,0} = 0$. Hence, the term $1 - \bar{x}_{3,0}$ in (3.42) can be replaced by $x_{3,0}$ and

$$Y = \left| u_1 + u_3 + \bar{x}_2 + c_{in} + x_{3,0} \right|_{2^n} \quad (3.43)$$

The sign of $\hat{X}$ can be detected by the MSB of Y . An n -bit carry save adder (CSA) can be used to add the three n -bit operands, u_1 , u_3 , and $\bar{x}_2$ to produce an n -bit sum $A = a_{n-1}a_{n-2} \cdots a_0$ and an n -bit carry vector $B = b_{n-1}b_{n-2} \cdots b_1 0$. Due to the modulo 2^n addition, the final carry output bit b_n of the CSA needs not be generated. As $b_0 = '0'$, it can be replaced by $x_{3,0}$ of (3.43) before the MSB of Y is computed by a simplified parallel prefix adder of A and B with the input carry bit $c_{-1} = c_{in}$. The prefix adder is simplified by keeping only the carry generation network for the computation of carry signal c_{n-1} , from which the sign of $\hat{X}$ can be detected by $\text{sign}(\hat{X}) = a_{n-1} \oplus b_{n-1} \oplus c_{n-1}$. The architecture of the proposed sign detector is shown in Figure 3.8, where the circuit diagram for the simplified prefix adder is depicted in Figure 3.9 for $n = 8$.

Figure 3.8: Proposed sign detection architecture for $\{2^n-1, 2^n, 2^n+1\}$.

Figure 3.9: Simplified prefix adder for $n = 8$.

Example 3.2: for $n = 5$, $\{m_1, m_2, m_3\} = \{31, 32, 33\}$, $M = 31 \times 32 \times 33 = 32736$ and $M/2 = 16368$.

The signed integer $\hat{X} = -11161$ can be represented by the residue representation $(x_1, x_2, x_3) = (30, 7, 26)$ corresponding to the unsigned integer $X = 21575$ in the same moduli set. The binary representation of the residues are $x_1 = 11110_2$, $x_2 = 00111_2$ and $x_3 = 011010_2$. According to (3.32), (3.33) and (3.41), $u_1 = 01111_2$, $u_2 = 10010_2$ and $u_3 = 01101_2$. Also, $x_{3,0} = 0$. Since $u_1 + u_2 = 01111 + 10010 = 33 > 32$, $C_1 = 1$. Since $33 \neq 31$, $C_2 = 0$. According to (3.36), $c_{in} = C_1 \vee C_2 = 1$. The computation of Y in (3.43) is illustrated in Figure 3.10. Since MSB of $Y = '1'$, the integer $\hat{X}$ represented by $(30, 7, 26)$ is negative.

u_1		0	1	1	1	1
u_3		0	1	1	0	1
$\overline{x_2}$	+	1	1	0	0	0
<hr/>						
A		1	1	0	1	0
B		1	1	0	1	0= $x_{3,0}$
<hr/>						
A		1	1	0	1	0
B		1	1	0	1	0
c_{in}						1
<hr/>						
Y		1	0	1	0	1

Figure 3.10: Computation of Y for Example 1.

3.4 Comparison and Results

3.4.1 Scaling-assisted Sign Detection Method I

The hardware resources required for the proposed sign detector can be analyzed as follows. For the computation of $sign_a(X)$, $4n+2$ inverters, one XOR, one AND and one OR gates are needed to generate u'_1 and u'_2 from u_1 , u_2 and u_3 . $2n$ full adders (FAs) are needed to implement a $2n$ -bit CSA with EAC. The computation of c_{in} in Figure 3.2 uses approximately $2n$ XOR gates, $2n$ OR gates and $6n-2$ AND gates. The truncated CPA requires two XOR gates, n AND gates and n OR gates. As the area and delay of inverter are negligible, the estimated hardware cost is $2n$ FAs, $2n+3$ XOR gates, $7n-1$ AND gates and $3n+1$ OR gates. The delays of c_{in} , CSA and the truncated n -bit CPA (assuming a block factor of 4 for the carry lookahead network) are approximated by $t_{c_{in}} \approx t_{XOR} + 4nt_{AND}$, $t_{CSA} = t_{FA}$ and $t_{CPA} = 2t_{XOR} + (4 \lceil \log_4 n \rceil + 1)t_{AND}$, respectively, where t_{FA} , t_{XOR} and t_{AND} denote the delays of a FA, a two-input XOR gate and a two-input AND gate, respectively. For the computation of $sign_b(X)$, the hardware cost is approximated by $2n$ XOR gates, $2n-2$ OR gates, one AND gate and two inverters. The critical path delay is estimated to be $t_{XOR} + \lceil \log_2 n \rceil t_{OR} + t_{AND}$, where t_{OR} is the delay of a two-input OR gate. Combining both parts, the total hardware cost of the proposed RNS sign detector is $2n$ FAs, $4n+3$ XOR gates, $7n$ AND gates and $5n-1$ OR gates. The critical path delay of the RNS sign detector is dominated by $sign_a(X)$ path, which is approximately $t_{FA} + 3t_{XOR} + (4n + 4 \lceil \log_4 n \rceil + 1)t_{AND} + t_{OR}$.

The proposed sign detector circuit and the sign detector designed based on the efficient reverse conversion for $\{2^n, 2^n - 1, 2^n + 1\}$ [Wan02] were synthesized and mapped to STM CMOS 65nm technology standard cell library by Synopsys Design Compiler. Both designs are optimized for speed and the synthesized areas in μm^2 and critical path delays in ps are shown in left side of Table 3.1. The results show that the proposed sign detector unit is on average 24% smaller and about 45% faster compared with the reverse converter based sign detector unit. Power simulation is also performed on each design by Synopsys Prime Time PX where the clock rate for the input data is chosen based on the slowest design for each value of n . The average

leakage power (LP) and total power (TP) are compared in right side of Table 3.1. The proposed design saves 32.94% of power over the reverse converter based sign detector unit.

Table 3.1: Comparison of area, critical path delay and power of sign detectors

n	Logic Synthesis Results				Power Simulation			
	Converter based on [Wan02]		Proposed Circuit		Converter based on [Wan02]		Proposed Circuit	
	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	LP(μW)	TP(mW)	LP(μW)	TP(mW)
5	629.8	1024.40	433.5	758.16	60.47	0.99	47.83	0.69
10	1300.0	1171.04	600.0	881.40	47.00	0.84	38.02	0.54
15	1490.5	1679.60	742.0	1225.12	68.08	1.06	48.00	0.58
20	1562.0	2245.87	825.5	1788.80	90.66	1.27	75.26	1.01

3.4.2 Scaling-assisted Sign Detection Method II

The proposed architecture in Figure 3.8 consists of an n -bit CSA, a c_{in} generator and a simplified prefix adder. Based on the unit-gate model [Tya93], the area and time complexity of the proposed design is estimated to be $18n - 7$ and $2\log_2 n + 9$ units of equivalent two-input gates, respectively, which indicates that the area increases linearly while the delay increases logarithmically with the size of its input moduli n . Due to the potentially high error margin of the simplistic gate count and gate delay model, for a more accurate comparison with other designs, all designs in comparison are described in Verilog HDL and synthesized for minimum delay by Synopsys Design Compiler version E-2010.12-SP4 using the same STM CMOS 65 nm LPGA 0.9V standard cell library. The designs in comparison includes CRT-I based [Xu12], CRT-II based [Tom08], MRC-II based [Akk08], MRC based [Sou14] and the most efficient reverse converter [Wan00b] based sign detectors for $\{2^n - 1, 2^n, 2^n + 1\}$. The latter is reduced by discarding all logic gates except only those that contribute to the calculation of sign bit. The synthesized areas in μm^2 and delays in ns for $n = 5, 10, 15, 20$ and 25 were tabulated in Table 3.2. The results show that the proposed sign detector is the most area-time efficient. On average, it consumes 8.6%, 14.7%, 54.9%, 39.6% and 36.5% lesser area than [Xu12], [Tom08], [Akk08], [Sou14] and [Wan00b], respectively. It is also 7.8%, 8.6%, 52.1%, 35.3% and 32.8% faster than [Xu12], [Tom08], [Akk08], [Sou14] and [Wan00b], respectively.

Table 3.2: Comparison of area (μm^2) and delay (ns) of sign detection circuits for $\{2^n-1, 2^n, 2^n+1\}$

Design	Area/ Delay	n				
		5	10	15	20	25
[Xu12]	Area	456	738	1227	1618	1889
	Delay	0.55	0.59	0.63	0.66	0.71
[Tom08]	Area	466	863	1310	1658	2026
	Delay	0.56	0.58	0.64	0.68	0.71
[Akk08]	Area	812	1744	2239	3357	4071
	Delay	0.87	1.14	1.23	1.38	1.53
[Sou14]	Area	507	1199	2198	2438	3165
	Delay	0.63	0.83	0.91	1.04	1.17
[Wan00b]	Area	661	1051	1669	2186	3116
	Delay	0.60	0.80	0.91	1.01	1.08
This	Area	378	723	1125	1499	1731
	Delay	0.48	0.55	0.59	0.62	0.66

Monte Carlo power simulation [Bur93] is also performed on each design by Synopsys Prime Time PX, where randomly generated input vectors are applied until the error in the calculated average power is bounded below 3% with 99% confidence interval. The clock rate for the input data is chosen based on the slowest design for each value of n . The average leakage power (LP) and total power (TP) are compared in Table 3.3. The proposed design saves 5.2%, 9.7%, 58.3%, 38.3% and 29.3% of power over [Xu12], [Tom08], [Akk08], [Sou14] and [Wan00b], respectively.

Table 3.3: Comparison of average leakage (LP) in μW and total power (TP) in mW of sign detection circuits for $\{2^n-1, 2^n, 2^n+1\}$

n	Clock Frequency (GHz)	[Xu12]		[Tom08]		[Akk08]		[Sou14]		[Wan00b]		This	
		LP	TP	LP	TP	LP	TP	LP	TP	LP	TP	LP	TP
5	1.15	19.79	0.30	21.67	0.31	44.07	0.61	23.91	0.34	34.74	0.43	17.56	0.27
10	0.88	33.54	0.55	40.96	0.60	91.76	1.39	56.77	0.83	43.78	0.71	33.76	0.55
15	0.81	49.72	0.83	59.26	0.88	109.55	1.69	107.98	1.60	64.97	1.07	48.98	0.79
20	0.72	73.18	1.08	72.23	1.10	169.80	2.60	112.87	1.73	81.81	1.35	64.60	1.06
25	0.65	79.88	1.30	83.61	1.36	204.60	3.17	153.55	2.27	115.68	1.94	70.23	1.18

3.5 Summary

A novel area-efficient and high-speed adder-based sign detector is proposed for $\{2^n, 2^n-1, 2^n+1\}$ RNS. With the aid of modified CRT, and scaling and circular left shift properties of the special moduli, the size of the modular operations are greatly reduced, which results in an average area and delay reduction of more than 24% for different values of n compared with the sign detection unit designed based on the efficient reverse conversion algorithm.

By changing the scaling factor from 2^n to $2^{2n}-1$, the width of modulo operators is further reduced from $3n$ bits to n -bits, which leads to the second scaling-assisted sign detection method with a high-speed and area-efficient adder-based implementation. Experimental results show that the proposed circuit is on average 8.6%, 14.7%, 54.9%, 39.6% and 36.5% smaller, 7.8%, 8.6%, 52.1%, 35.3% and 32.8% faster, and 5.2%, 9.7%, 58.3%, 38.3% and 29.3% more power-efficient than the latest CRT-I, CRT-II, MRC-II, MRC and stripped-down reverse converter based sign detectors, respectively.

In the next chapter, a new algorithm for comparing the signed integers in the four-moduli supersets will be proposed.

Chapter 4

New Algorithm for Signed Residue Comparison in Four-moduli Supersets and Its Efficient VLSI Implementations

4.1 Introduction

In this chapter, a new algorithm is proposed for the comparison of two signed integers in the arithmetic friendly four-moduli supersets [Pat14], namely $S_1 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$, for even values of n and $k \in [0, n]$. The four-moduli supersets are formed by adding a modulus of $2^{n\pm 1}-1$ into the popular three-moduli set $\{2^n, 2^n-1, 2^n+1\}$ [Bi08], [Tay13], [Cha11] and allowing the exponent of the power-of-two modulus to be different from n . With the additional $2^{n\pm 1}-1$ modulus, the dynamic range is expanded with increased parallelism while the simplicity of modular arithmetic operations is preserved like [Cao07], [Mol10], [Pat14], and [Tay14]. With the freedom to vary the parameter k , the desired dynamic range of an application can be easily accommodated by these supersets [Cha04]. Alternatively, k can be increased to delay the originally faster arithmetic units in the modulus 2^{n+k} channel while shortening the lengths of other slower modulus channels to maintain the dynamic range [Cho09], [Cao10]. Based on these two supersets, a new range quantization concept involving the bisection of the dynamic range into distinct divisions is introduced for fast comparison of signed integers in residue representation. Complexity of full transcoding for exact magnitude evaluation is avoided by identifying only the ranks of division within which the residue representations in comparison lie. Only when their ranks are identical will partial magnitude evaluation be required. The evaluation required is simpler for the narrower range identified by their common rank. Unlike the existing methods, the sign of an integer is inherent in its rank without the need for a complicated RNS sign detector. It can be detected from the MSB of its rank except only for the middle rank. For the residue representation that falls in the middle rank, its sign can still be easily decoded with simple Boolean function without requiring any modulo

arithmetic operation. The proposed algorithm translates into efficient hardware architecture that can be implemented using only adders and primitive logic gates without any lookup table.

4.2 Proposed Algorithm for Comparison of Signed Integers in RNS

Let the four moduli be $m_1 = 2^{n+k}$, $m_2 = 2^n - 1$, $m_3 = 2^n + 1$, and $m_4 = 2^{n+1} - 1$ for S_1 and $2^{n-1} - 1$ for S_2 . The dynamic range $M = m_1 m_2 m_3 m_4$ can be quantized into m_4 equal divisions, as shown in Figure 4.1. Each division has an equal range of M_{R1} integers, where $M_{R1} = m_1 m_2 m_3 = 2^{n+k} (2^{2n} - 1)$. An integer identifier $W \in [0, m_4 - 1]$ is assigned to each division in an ascending order of its distance from zero, as shown in Figure 4.1. W is called the rank of a division as it uniquely identifies a group of contiguous integers that fall in the division based on their magnitudes. An integer that falls in a division with a higher rank will have a larger magnitude than that with a lower rank.

Figure 4.1: The quantization of M into $m_4 = 2^{n+1} - 1$ for S_1 or $m_4 = 2^{n-1} - 1$ for S_2 divisions.

By definition of the residue arithmetic, an unsigned integer X can be expressed in terms of W_x and V_x as follows:

$$\begin{aligned} X &= \left\lfloor \frac{X}{M_{R1}} \right\rfloor M_{R1} + |X|_{M_{R1}} \\ &= W_x M_{R1} + V_x \end{aligned} \quad (4.1)$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x , $W_x = \lfloor X/M_{R1} \rfloor \in [0, m_4 - 1]$, and

$$V_x = |X|_{M_{R1}}.$$

Let Y be an unsigned integer to be compared with X . Y can be similarly expressed as:

$$Y = W_y M_{R1} + V_y \quad (4.2)$$

The magnitudes of X and Y can be compared by comparing W_x with W_y . If $W_x > W_y$, it can be concluded immediately that X is larger than Y and vice versa. When $W_x = W_y$, a final comparison needs to be carried out between V_x and V_y . If $V_x > V_y$, then X is larger than Y and vice versa. Otherwise, $V_x = V_y$, which implies that $X = Y$.

As the integers X and Y are not directly known in the residue domain, W_x , and V_x , and W_y and V_y have to be computed from their respective residue representations, (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) , before the above simple steps can be applied. From the definition of V_x in (4.1), $V_x = |X|_{M_{R1}} = |X|_{m_1 m_2 m_3}$. By applying (2.13) on the residues (x_2, x_3) with respect to the moduli set $\{m_2, m_3\}$, U_x can be expressed as:

$$U_x = |X|_{m_2 m_3} = \left| m_3 \left| m_3^{-1} \right|_{m_2} x_2 + m_2 \left| m_2^{-1} \right|_{m_3} x_3 \right|_{m_2 m_3} \quad (4.3)$$

where $\left| m_3^{-1} \right|_{m_2} = \left| m_2^{-1} \right|_{m_3} = 2^{n-1}$.

Proof of $\left| (m_2)^{-1} \right|_{m_3} = \left| (2^n - 1)^{-1} \right|_{2^n + 1} = 2^{n-1}$:

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

Q.E.D.

Proof of $\left| (m_3)^{-1} \right|_{m_2} = \left| (2^n + 1)^{-1} \right|_{2^n - 1} = 2^{n-1}$:

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

Q.E.D.

Let $M_{R2} = m_2 m_3$, then $M_{R1} = m_1 M_{R2}$. By treating (U_x, x_1) as the residues of a two-moduli set $\{M_{R2}, m_1\}$, $V_x = |X|_{M_{R1}}$ can be computed from U_x and x_1 by (2.18) as follows:

$$V_x = \left| m_1^{-1} \right|_{M_{R2}} (U_x - x_1) \Big|_{M_{R2}} m_1 + x_1 \quad (4.4)$$

where $\left| m_1^{-1} \right|_{M_{R2}} = \left| (2^{n+k})^{-1} \right|_{2^{2n} - 1} = 2^{n-k}$.

Proof of $\left| \left(2^{n+k} \right)^{-1} \right|_{2^{2n-1}} = 2^{n-k} :$

$$= \left| 2^{n+k} \times 2^{n-k} \right|_{2^{2n-1}} = \left| 2^{n+k+n-k} \right|_{2^{2n-1}} = \left| 2^{2n} - 1 + 1 \right|_{2^{2n-1}} = 1$$

Q.E.D.

Similarly, by treating (V_x, x_4) as the residue representation of a two-moduli set $\{M_{R1}, m_4\}$, X can be calculated from V_x and x_4 using (2.18) as follows:

$$X = \left| M_{R1}^{-1} \right|_{m_4} (x_4 - V_x) \Big|_{m_4} \times M_{R1} + V_x \quad (4.5)$$

W_x can be obtained by comparing the two expressions of X in (4.1) and (4.5),

$$W_x = \left| M_{R1}^{-1} \right|_{m_4} (x_4 - V_x) \Big|_{m_4} \quad (4.6)$$

Since $M_{R1} = m_1 M_{R2}$ and $\left| m_1^{-1} \right|_{m_4} = 2^{-(n+k)}$, $\left| M_{R1}^{-1} \right|_{m_4} = \left| (m_1 M_{R2})^{-1} \right|_{m_4} = \left| m_1^{-1} \right|_{m_4} \left| M_{R2}^{-1} \right|_{m_4} =$

$$\left| 2^{-(n+k)} \right|_{m_4} \left| M_{R2}^{-1} \right|_{m_4},$$

$$W_x = \left| 2^{-(n+k)} \right|_{m_4} \left| M_{R2}^{-1} \right|_{m_4} (x_4 - V_x) \Big|_{m_4} \quad (4.7)$$

where $\left| M_{R2}^{-1} \right|_{m_4} = \left| (2^{2n} - 1)^{-1} \right|_{2^{n+1}-1} = \left| 8 \sum_{i=0}^{n/2-1} 2^{2i} \right|_{2^{n+1}-1}$ for S_1 and

$$\left| M_{R2}^{-1} \right|_{m_4} = \left| (2^{2n} - 1)^{-1} \right|_{2^{n-1}-1} = \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} \right|_{2^{n-1}-1} \text{ for } S_2 \text{ [Pat14].}$$

Similarly, V_y and W_y for the other operand Y can also be computed by (4.4) and (4.7). This method falls under the partial reverse conversion approach as V_x and W_x can be deemed as the MRC coefficients of a two-moduli MRC.

To apply the above technique to compare the magnitudes of two signed integers, $\tilde{X}$ and $\tilde{Y}$, in RNS, three different cases need to be considered. Case (a): if $\tilde{X}$ and $\tilde{Y}$ are both positive, the abovementioned magnitude comparison method can be applied directly to $\tilde{X}$ and $\tilde{Y}$. Case (b):

if $\tilde{X}$ and $\tilde{Y}$ have different signs, it is trivial to conclude that the positive integer is always larger than the negative integer. Therefore, no further comparison is required. Case (c): if $\tilde{X}$ and $\tilde{Y}$ are both negative, due to the congruence expressed in (2.3), the larger negative integer will have a higher W , if not higher V . Thus, the abovementioned magnitude comparison method for unsigned integers can also be applied directly to $\tilde{X}$ and $\tilde{Y}$ according to their signs. The problem is sign detection is itself a difficult operation in RNS.

Unlike all previously introduced algorithms, the information about sign of $\tilde{X}$ is readily available in W_x and V_x . There are m_4 ranks in the dynamic range M and each rank W_x is a subrange of M_{R1} integers. $W_x = \lfloor m_4/2 \rfloor$ is a watershed that divides M into an equal number of ranks before and after it. The MSB of W_x is '0' for $W_x < \lfloor m_4/2 \rfloor$ that cover the lower subrange of M where $\tilde{X}$ is positive, and the MSB of W_x is '1' for $W_x > \lfloor m_4/2 \rfloor$ that cover the upper subrange of M where $\tilde{X}$ is negative. However, when $W_x = \lfloor m_4/2 \rfloor$, half of the integers in this rank falls in the lower half range of M and the other half of them falls in the upper half range of M , as depicted in Figure 4.1. Since $V_x \in [0, M_{R1} - 1]$ indicates the exact position of the integer in W_x , the sign of $\tilde{X}$ for $W_x = \lfloor m_4/2 \rfloor$ can be detected by comparing V_x with $M_{R1}/2$. In short, the sign of $\tilde{X}$ can be computed by:

$$\text{sign}(\tilde{X}) = \begin{cases} \text{MSB}(W_x) & \text{if } W_x \neq \lfloor m_4/2 \rfloor \\ 1 & \text{if } W_x = \lfloor m_4/2 \rfloor \text{ and } V_x \geq M_{R1}/2 \\ 0 & \text{otherwise} \end{cases} \quad (4.8)$$

From (4.8), a $(3n+k)$ -bit binary magnitude comparator is required to detect the condition of $V_x \geq M_{R1}/2$. This hardware requirement can be greatly reduced to $2n+1$ basic logic gates by the technique detailed in Section 4.3.

The flowchart of the proposed quantization algorithm for comparing the magnitude of $\tilde{X}$ and $\tilde{Y}$ in the same residue domain of S_1 or S_2 is depicted in Figure 4.2. The proposed algorithm is also illustrated by the following two numerical examples.

Figure 4.2: Proposed sign integer comparison algorithm in RNS S_1 or S_2 .

Example 4.1: Consider $S_1 \equiv \{32, 15, 17, 31\}$ with $n = 4$ and $k = 1$. $M_{R1} = 32 \times 15 \times 17 = 8160$ and $M_{R2} = 15 \times 17 = 255$. Let $\tilde{X} \equiv (8, 10, 8, 1)$ and $\tilde{Y} \equiv (27, 13, 1, 20)$ be the two residue representations to be compared in S_1 . Since $|17^{-1}|_{15} = |15^{-1}|_{17} = 8$, U_x and U_y can be computed by (4.3) as $|17 \times 8 \times 10 + 15 \times 8 \times 8|_{255} = 25$ and $|17 \times 8 \times 13 + 15 \times 8 \times 1|_{255} = 103$, respectively. From (4.4), since $|m_1^{-1}|_{M_{R2}} = |32^{-1}|_{255} = 2^{4-1} = 8$, $V_x = 32 \times |8(25-8)|_{255} + 8 = 4360$ and $V_y = 32 \times |8(103 - 27)|_{255} + 27 = 3163$. From (4.7), since $|255^{-1}|_{31} = |8(1+4)|_{31} = 9$ and $|2^{-5}|_{31} = |2^{5-5}|_{31} = 1$, $W_x = |9 \times (1-4360)|_{31} = |9 \times 12|_{31} = |108|_{31} = 15 = 01111_2$. Similarly, $W_y = |9 \times (20-3163)|_{31} = |9 \times 19|_{31} = |171|_{31} = 16 = 10000_2$. Since $W_x = \lfloor m_4/2 \rfloor = \lfloor 31/2 \rfloor = 15$ and $V_x = 4360 > M_{R1}/2 = 4080$, according to (4.8), $\tilde{X}$ is negative. Since $W_y = 16 \neq \lfloor m_4/2 \rfloor$ and MSB(W_y) = '1', according to

(4.8), $\tilde{Y}$ is also negative. As $\tilde{X}$ and $\tilde{Y}$ are both negative and $W_x = 15 < W_y = 16$, it can be concluded that $\tilde{X} < \tilde{Y}$. From the reverse converted integer values of $\tilde{X}$ and $\tilde{Y}$ in the range of $[-126480, 252959]$, $\tilde{X} = -126200$ is indeed smaller than $\tilde{Y} = -119237$.

Example 4.2: Consider $S_2 \equiv \{4096, 255, 257, 127\}$ with $n = 8$ and $k = 4$. $M_{R1} = 4096 \times 255 \times 257 = 268431360$ and $M_{R2} = 255 \times 257 = 65535$. Let $\tilde{X} \equiv (1842, 49, 51, 48)$ and $\tilde{Y} \equiv (1562, 84, 135, 106)$ be the two residue representations to be compared in S_2 . Since $|257^{-1}|_{255} = |255^{-1}|_{257} = 128$, U_x and U_y can be computed by (4.3) as $|257 \times 128 \times 49 + 255 \times 128 \times 51|_{65535} = 65329$ and $|257 \times 128 \times 84 + 255 \times 128 \times 135|_{65535} = 26349$, respectively. From (4.4), since $|m_1^{-1}|_{65535} = 2^{8-4} = 16$, $V_x = 4096 \times |16(65329 - 1842)|_{65535} + 1842 = 134215474$ and $V_y = 4096 \times |16(26349 - 1562)|_{65535} + 1562 = 13854234$. From (4.7), since $|65535^{-1}|_{127} = |-2(1+4+16)|_{127} = 85$ and $|2^{-12}|_{127} = |2^{7 \times 2 - 12}|_{127} = 2^2$, $W_x = |2^2 \times 85(48 - 134215474)|_{127} = |4 \times 85 \times 79|_{127} = |26860|_{127} = 63 = 0111111_2$ and $W_y = |2^{-12} \times 85(106 - 13854234)|_{127} = |2^2 \times 85 \times 48|_{127} = 64 = 1000000_2$. Since $W_x = \lfloor m_4/2 \rfloor = \lfloor 127/2 \rfloor = 63$ and $V_x = 134215474 < M_{R1}/2 = 134215680$, according to (4.8), $\tilde{X}$ is a positive integer. Since $W_y \neq \lfloor 127/2 \rfloor = 63$ and $\text{MSB}(W_y) = '1'$, according to (4.8), $\tilde{Y}$ is negative. Since $\tilde{X}$ is positive and $\tilde{Y}$ is negative, it can directly concluded that $\tilde{X} > \tilde{Y}$. From the reverse converted integer values of $\tilde{X}$ and $\tilde{Y}$, $\tilde{X} = 17045391154$ is indeed greater than $\tilde{Y} = -16897321446$.

4.3 Proposed Hardware Architecture for Comparison of Signed Integers in RNS

Figure 4.3 depicts the proposed architecture. The ranks, W_x , V_x , W_y , and V_y , are computed from the residues of $\tilde{X}$ and $\tilde{Y}$. The internal structures of V generator, W generator and sign detector are different for S_1 and S_2 , and their implementations will be detailed later in this section. Based on the values of W_x , V_x , $\text{sign}(\tilde{X})$, W_y , V_y and $\text{sign}(\tilde{Y})$, the final state of $\tilde{X} > \tilde{Y}$, $\tilde{X} < \tilde{Y}$ and $\tilde{X} = \tilde{Y}$ are determined with two binary comparators and some logic gates.

Figure 4.3: Architecture of the proposed signed integer comparison algorithm.

The hardware implementation of U_x can be derived by substituting $|m_3^{-1}|_{m_2} = |m_2^{-1}|_{m_3} = 2^{n-1}$ into (4.3) and simplified.

$$\begin{aligned}
 U_x &= \left| (2^n + 1)(2^{n-1})x_2 + (2^n - 1)(2^{n-1})x_3 \right|_{2^{2n}-1} \\
 &= \left| 2^{2n-1}x_2 + 2^{n-1}x_2 + 2^{2n-1}x_3 - 2^{n-1}x_3 \right|_{2^{2n}-1}
 \end{aligned} \tag{4.9}$$

Let $Q_1 = \left| 2^{2n-1}x_2 + 2^{n-1}x_2 \right|_{2^{2n}-1}$ and $Q_2 = \left| 2^{2n-1}x_3 - 2^{n-1}x_3 \right|_{2^{2n}-1}$. By applying *Properties 2.11* and

2.12, Q_1 and Q_2 can be expressed as

$$\begin{aligned}
 Q_1 &= \left| CLS_{2n}(x_2, 2n-1) + CLS_{2n}(x_2, n-1) \right|_{2^{2n}-1} \\
 &= \left| x_{2,0} \underbrace{0 \cdots 0}_n x_{2,n-1} \cdots x_{2,1} + 0 x_{2,n-1} \cdots x_{2,0} \underbrace{0 \cdots 0}_{n-1} \right|_{2^{2n}-1} \\
 &= x_{2,0} \underbrace{x_{2,n-1} \cdots x_{2,0}}_n x_{2,n-1} \cdots x_{2,1}
 \end{aligned} \tag{4.10}$$

$$\begin{aligned}
 Q_2 &= \left| CLS_{2n}(x_3, 2n-1) + CLS_{2n}(\bar{x}_3, n-1) \right|_{2^{2n}-1} \\
 &= \left| x_{3,0} \underbrace{0 \cdots 0}_{n-1} x_{3,n} \cdots x_{3,1} + \bar{x}_{3,n} \cdots \bar{x}_{3,0} \underbrace{1 \cdots 1}_{n-1} \right|_{2^{2n}-1}
 \end{aligned} \tag{4.11}$$

The expression of Q_2 can be further reduced by exploiting the fact that the $(n + 1)$ -bit word $x_3 \in [0, 2^n]$. When the MSB $x_{3,n}$ is 1, its remaining bits $x_{3,n-1}x_{3,n-2} \cdots x_{3,0}$ are all zeros. Hence the two terms of (4.11) summed up to a single constant $C = \underbrace{0 \cdots 0}_n \underbrace{10 \cdots 0}_{n-1} + \underbrace{0 \underbrace{1 \cdots 1}_{2n-1}}_{2n-1} = \underbrace{10 \cdots 0}_n \underbrace{1 \cdots 1}_{n-1}$,

Depending on the value of $x_{3,n}$, U_x can be expressed as a mod $2^{2n}-1$ sum of either two or three terms as follows:

$$U_x = \begin{cases} |Q_1 + C|_{2^{2n}-1} & \text{if } x_{3,n} = '1' \\ |Q_1 + Q_3 + C|_{2^{2n}-1} & \text{if } x_{3,n} = '0' \end{cases} \quad (4.12)$$

where $Q_3 = x_{3,0} \underbrace{\bar{x}_{3,n-1} \cdots \bar{x}_{3,0}}_n \underbrace{x_{3,n-1} \cdots x_{3,1}}_{n-1}$ and $C = \underbrace{10 \cdots 0}_n \underbrace{1 \cdots 1}_{n-1}$.

Q_1 in (4.10) can be obtained without requiring any hardware logic by reordering the bits of x_2 , i.e., $Q_1 = x_{2,0} \underbrace{x_{2,n-1} \cdots x_{2,0}}_n \underbrace{x_{2,n-1} \cdots x_{2,1}}_{n-1}$. Q_3 needs only bit inversion and reshuffling. Therefore, the mod $2^{2n} - 1$ sum of U_x in (4.12) can be implemented by using n modified full adders (MFAs) [Tay15]. Each MFA is composed of two modified half adders denoted as MHA1 and MHA2, as depicted in the top circuit diagram of Figure 4.4(b). Each MHA1 is used to add two binary bits, $x_{1,i}$ and $x_{3,i}$, and a constant '1' in a column whereas each MHA2 is used to add two binary bits, $x_{1,i}$ and $(\bar{x}_{3,i} \cdot \bar{x}_{3,n})$, in a column for all $i \in [0, n-1]$, as shown in the bottom diagram of Figure 4.4(b). The final sum and carry outputs are rearranged to form two $2n$ -bit binary vectors, $A = a_{2n-1} \cdots a_1 a_0$ and $B = b_{2n-1} \cdots b_1 b_0$, respectively.

By substituting $U_x = |A + B|_{2^{2n}-1}$ and $m_1 = 2^{n+k}$ into (4.4), V_x can be expressed as:

$$\begin{aligned} V_x &= \left| 2^{n-k} (A + B - x_1) \right|_{2^{2n}-1} 2^{n+k} + x_1 \\ &= 2^{n+k} P + x_1 \end{aligned} \quad (4.13)$$

The term P in (4.13) can be implemented by one stage of $2n$ -bit carry save adder with end around carry followed by a mod $2^{2n}-1$ adder [Che09] by expanding the term $2^{n-k} (A + B - x_1)$ into three binary vectors as follows:

$$P = \left| 2^{n-k} (A + B - x_1) \right|_{2^{2n-1}} = \left| P_1 + P_2 + P_3 \right|_{2^{2n-1}} \quad (4.14)$$

where

$$\begin{aligned} P_1 &= 2^{n-k} A = CLS_{2n}(A, n-k) \\ &= \underbrace{a_{n+k-1} \cdots a_1 a_0}_{n+k} \underbrace{a_{2n-1} \cdots a_{n+k+1} a_{n+k}}_{n-k} \end{aligned} \quad (4.15)$$

$$\begin{aligned} P_2 &= 2^{n-k} B = CLS_{2n}(B, n-k) \\ &= \underbrace{b_{n+k-1} \cdots b_1 b_0}_{n+k} \underbrace{b_{2n-1} \cdots b_{n+k+1} b_{n+k}}_{n-k} \end{aligned} \quad (4.16)$$

$$\begin{aligned} P_3 &= 2^{n-k} \bar{x}_1 = CLS_{2n}(\bar{x}_1, n-k) \\ &= CLS_{2n}(\underbrace{11 \cdots 11}_{n-k} \underbrace{\bar{x}_{1,n+k-1} \cdots \bar{x}_{1,0}}_{n-k}, n-k) \\ &= \underbrace{\bar{x}_{1,n+k-1} \cdots \bar{x}_{1,0}}_{n+k} \underbrace{11 \cdots 11}_{n-k} \end{aligned} \quad (4.17)$$

Since $x_1 \in [0, 2^{n+k})$ can be expressed as a $(n+k)$ -bit binary word and multiplying the integer P by 2^{n+k} is equivalent to shifting its binary representation left by $n+k$ bits, V_x can be computed by directly concatenating the operands $x_1 = x_{1,n+k-1} \cdots x_{1,1} x_{1,0}$ and $P = p_{2n-1} \cdots p_1 p_0$ at no cost, i.e.,

$$V_x = \underbrace{p_{2n-1} \cdots p_1 p_0}_{2n} \underbrace{x_{1,n+k-1} \cdots x_{1,1} x_{1,0}}_{n+k} \quad (4.18)$$

Figure 4.4(a) and Figure 4.4(b) show the hardware implementations of V generator and MFA, respectively.

Due to the different close-form expressions of $\left| M_{R2}^{-1} \right|_{m_q}$ for S_1 and S_2 in (4.7), the hardware implementation of their W generators is also different.

Since W_x is an $(n+1)$ -bit word for S_1 and V_x has $3n+k$ bits with $k \in [0, n]$, according to *Property 2.13*, V_x can be expressed as:

$$V_x = \left| v_4 + v_3 + v_2 + v_1 \right|_{2^{n+1}-1} \quad (4.19)$$

where

$$\begin{aligned}
v_1 &= \underbrace{V_{x,n} \cdots V_{x,0}}_{n+1}, \\
v_2 &= \underbrace{V_{x,2n+1} \cdots V_{x,n+1}}_{n+1}, \\
v_3 &= \underbrace{0 \cdots 0}_{3-k} \underbrace{V_{x,3n+k-1} \cdots V_{x,3n}}_k \underbrace{V_{x,3n-1} \cdots V_{x,2n+2}}_{n-2} \text{ if } k < 4, \\
v_4 &= \underbrace{0 \cdots 0}_{n-k+4} \underbrace{V_{x,3n+k-1} \cdots V_{x,3n+3}}_{k-3} \text{ if } k \geq 4,
\end{aligned}$$

Figure 4.4: Circuit architectures of (a) V generator (b) MFA.

The fourth operand v_4 is needed only when $k \geq 4$. From (4.7), W_x can be represented as

$$W_x = \left| 2^{-(n+k)} \left| M_{R2}^{-1} \right|_{m_4} (x_4 + \bar{v}_4 + \bar{v}_3 + \bar{v}_2 + \bar{v}_1) \right|_{2^{n+1}-1} \quad (4.20)$$

The mod $2^{n+1}-1$ sum of the last product term in (4.20) can be implemented by either two or three stages of $2n$ -bit carry save adder (CSA) with end around carry (EAC), depending on whether $k < 4$ or $k \geq 4$. The final $2n$ -bit sum and carry outputs are represented by the binary vectors $W_{x,s}$ and $W_{x,c}$, respectively.

With $\left| M_{R2}^{-1} \right|_{2^{n+1}-1} = \left| 8 \sum_{i=0}^{n/2-1} 2^{2i} \right|_{2^{n+1}-1}$, and by *Property 2.14*, W_x for S_1 can be computed by:

$$\begin{aligned}
W_x &= \left| 2^{-(n+k)} \times 2^3 \sum_{i=0}^{n/2-1} 2^{2i} (W_{x,s} + W_{x,c}) \right|_{2^{n+1}-1} \\
&= \left| \sum_{i=0}^{n/2-1} 2^{2i} (2^{-(n+k+3)} W_{x,s} + 2^{-(n+k+3)} W_{x,c}) \right|_{2^{n+1}-1} \\
&= \left| \sum_{i=0}^{n/2-1} 4^i (W_{x1} + W_{x2}) \right|_{2^{n+1}-1}
\end{aligned} \tag{4.21}$$

where $W_{x1} = \text{CRS}_{n+1}(W_{x,s}, n+k+1)$ and $W_{x2} = \text{CRS}_{n+1}(W_{x,c}, n+k+3)$ and CRS denotes the circular right shift operation.

The mod $2^{n+1}-1$ sum of (4.21) has $2 \times n/2 = n$ terms, which requires $n-2$ $(n+1)$ -bit CSAs with EAC. The sum of the power-of-four integers $\sum_{i=0}^{n/2-1} 4^i = \underbrace{1010101 \dots 0101}_{n-2}$ happens to be a binary string of alternating ones and zeros of $(n-2)$ bits length. To reduce the number of operands in the mod $2^{n+1}-1$ addition, the common substring “101” can be generated first and reused by left shifting it consecutively by multiple-of-four bit positions. For the case of power-of-two integer n , i.e., $n = 2^{j+1}$ and $j \in \mathbb{Z}^+$, W_x can be computed with only $j = \log_2 n - 1$ stages of two-level CSA with EAC in the following recursion by keeping all intermediate sums in carry-saved form.

$$t_j = 2^{2^j} t_{j-1} + t_{j-1} \tag{4.22}$$

where $t_1 = (2^2 + 2^0) (W_{x1} + W_{x2}) = 2^2(W_{x1} + W_{x2}) + W_{x1} + W_{x2}$.

If n is not a power-of-two integer but a multiple-of-four integer, i.e., $n = 4j$ and $j \in \mathbb{Z}^+$, then

$$\sum_{i=0}^{n/2-1} 4^i = 0x \underbrace{5 \dots 55}_j \text{ is a sum of } j = n/4 \text{ terms of } t_1 \text{ and its multiple-of-four bit left-shifted terms.}$$

Thus, a CSA tree of one (for $j < 3$) or $j - 2$ (for $j \geq 3$) $(n+1)$ -bit CSAs with EAC is required to compute W_x in carry-save form before the final mod $2^{n+1}-1$ vector merged adder. Other cases of even integer of n can be computed by adding one additional left shifted operand corresponding to the highest weighted term to the sum of either one of the above cases. Because of the mod $2^{n+1}-1$ summation of (4.21), the left shifting of all terms is done circularly with a word length of $n + 1$ bits.

Figure 4.5 shows the computation of W_x for S_1 with $n = 16$ and $k = 0$ using the recursion of (4.22). The sum and carry vectors of $t_1 = 2^2(W_{x1} + W_{x2}) + W_{x1} + W_{x2}$ are first computed, followed by the sum and carry vectors of $t_2 = 2^4 t_1 + t_1 = (2^6 + 2^4 + 2^2 + 2^0)(W_{x1} + W_{x2})$ and the sum and carry vectors of $t_3 = 2^8 t_2 + t_2 = (2^{14} + 2^{12} + 2^{10} + 2^8 + 2^6 + 2^4 + 2^2 + 2^0)(W_{x1} + W_{x2})$ to obtain the final mod $2^{n+1}-1$ sum of $\sum_{i=0}^7 4^i (W_{x1} + W_{x2})$.

Figure 4.5: Computation of W_x for $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ with $n=16$.

Since W_x is an $(n-1)$ -bit word for S_2 , the $(3n+k)$ -bit word V_x in (4.18) can be similarly divided into no more than six $(n-1)$ -bit binary vectors as follows:

$$V_x = \left| v_6 + v_5 + v_4 + v_3 + v_2 + v_1 \right|_{2^{n-1}-1} \quad (4.23)$$

where

$$\begin{aligned} v_1 &= \underbrace{V_{x,n-2} \cdots V_{x,0}}_{n-1}, \\ v_2 &= \underbrace{V_{x,2n-3} \cdots V_{x,n-1}}_{n-1}, \\ v_3 &= \underbrace{V_{x,3n-4} \cdots V_{x,2n-2}}_{n-1}, \end{aligned}$$

$$\begin{aligned}
v_4 &= \begin{cases} V_{x,3n-1}V_{x,3n-2}V_{x,3n-3} & \text{if } n=4 \text{ and } k=0 \\ \underbrace{0 \cdots 0}_{n-k-4} \underbrace{V_{x,3n+k-1} \cdots V_{x,3n-3}}_{k+3} & \text{if } n \neq 4 \text{ and } k \leq n-4, \end{cases} \\
v_5 &= \begin{cases} \underbrace{0 \cdots 0}_{3-k} V_{x,3n+k-1} \cdots V_{x,3n} & \text{if } n=4 \text{ and } 1 \leq k \leq 3 \\ \underbrace{0 \cdots 0}_{2n-k-5} \underbrace{V_{x,3n+k-1} \cdots V_{x,4n-4}}_{k-n+4} & \text{if } n \neq 4 \text{ and } k > n-4, \end{cases} \\
v_6 &= 00V_{x,3n+k-1} \quad \text{if } n=4 \text{ and } k=4,
\end{aligned}$$

The sixth operand is needed only when $n = k = 4$, and the fifth operand is needed only when $k > n-4$. From (4.7), W_x can be expressed as follows:

$$W_x = \left| 2^{-(n+k)} \left| M_{R2}^{-1} \right|_{m_4} \left(\begin{array}{c} x_4 + \bar{v}_1 + \bar{v}_2 + \bar{v}_3 \\ + \bar{v}_4 + \bar{v}_5 + \bar{v}_6 \end{array} \right) \right|_{2^{n-1}-1} \quad (4.24)$$

The mod $2^{n-1}-1$ sum of the last product term in (4.24) can be implemented by three or four stages of $(n-1)$ -bit CSA with EAC depending on the values of n and k . The final $(n-1)$ -bit sum and carry outputs are represented by the binary vectors $W_{x,s}$ and $W_{x,c}$, respectively.

With $\left| M_{R2}^{-1} \right|_{2^{n-1}-1} = \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} \right|_{2^{n-1}-1}$, and by applying *Property 2.14*, W_x for S_2 can be computed by

$$\begin{aligned}
W_x &= \left| 2^{-(n+k)} \times (-2) \sum_{i=0}^{n/2-2} 2^{2i} (W_{x,s} + W_{x,c}) \right|_{2^{n-1}-1} \\
&= \left| \sum_{i=0}^{n/2-2} 2^{2i} \left(2^{-(n+k+1)} \overline{W_{x,s}} + 2^{-(n+k+1)} \overline{W_{x,c}} \right) \right|_{2^{n-1}-1} \\
&= \left| \sum_{i=0}^{n/2-2} 4^i (W_{x1} + W_{x2}) \right|_{2^{n-1}-1}
\end{aligned} \quad (4.25)$$

where $W_{x1} = CRS_{n-1}(\overline{W_{x,s}}, n+k+1)$ and $W_{x2} = CRS_{n-1}(\overline{W_{x,c}}, n+k+1)$.

The mod $2^{n-1}-1$ sum of (4.25) has $2(n/2-1) = n-2$ terms. The binary representation of $\sum_{i=0}^{n/2-2} 4^i =$

$\underbrace{1010101 \cdots 0101}_{n-4}$ is similar to that of $\sum_{i=0}^{n/2-1} 4^i$ for the W_x generator of S_1 with an offset of two bits.

Based on the same recursion of (4.22), for $n = 2^{j+1}+2$ where $j \in \mathbb{Z}^+$, $j = \log_2(n-2) - 1$ stages of

two-level CSA with EAC will be required to compute W_x in carry-save form. If $n = 4j+2$, then

$$\sum_{i=0}^{n/2-2} 4^i = 0x \underbrace{5 \dots 55}_j \text{ and a CSA tree of } j = (n-2)/4 \text{ (n+1)-bit CSAs with EAC is required to}$$

compute W_x in carry-save form. For other cases of even integer of n , one additional operand will be added to the sum of one of the above two cases. The left shifting of all terms is done circularly with a word length of $n - 1$ bits instead.

The general architectures of the W generators for S_1 and S_2 are shown in Figure 4.6(a) and Figure 4.6(b), respectively.

Figure 4.6: Circuit architectures of W generators for (a) S_1 and (b) S_2 .

The sign detector module of Figure 4.3 is designed according to (4.8), with the hardware required to identify the two conditions, $W_x = \lfloor m_4/2 \rfloor$ and $V_x \geq M_{R1}/2$, simplified as follows.

$$\text{For } S_1, \lfloor m_4/2 \rfloor = \lfloor (2^{n+1} - 1)/2 \rfloor = 0 \underbrace{11 \dots 11}_n \text{ whereas for } S_2, \lfloor m_4/2 \rfloor = \lfloor (2^{n-1} - 1)/2 \rfloor = 0 \underbrace{11 \dots 11}_{n-2}.$$

Thus, the condition $W_x = \lfloor m_4/2 \rfloor$ can be detected by a bitwise AND operation on W_x , i.e.,

$\overline{W_{x,n}} \wedge \underbrace{W_{x,n-1} \wedge \cdots \wedge W_{x,1} \wedge W_{x,0}}_n$ for S_1 and $\overline{W_{x,n-2}} \wedge \underbrace{W_{x,n-3} \wedge \cdots \wedge W_{x,1} \wedge W_{x,0}}_{n-2}$ for S_2 . The large AND

gate (which will be synthesized into multi-level logic structure to reduce its fan-in) for the detection of $W_x = \lfloor m_4/2 \rfloor$ is shown on the left hand side of Figure 4.7 before the two-input AND gate.

Since the least significant $n + k - 1$ bits of $M_{R1}/2 = 2^{3n+k-1} - 2^{n+k-1} = \underbrace{011\cdots1}_{2n} \underbrace{00\cdots0}_{n+k-1}$ are all zeros,

they need not be compared for $V_x \geq M_{R1}/2$ as long as the most significant $2n+1$ bits of V_x is greater than or equal to the most significant $2n+1$ bits of $M_{R1}/2$, i.e., $V_{x,3n+k-1} \cdots V_{x,n+k-1} \geq \underbrace{011\cdots1}_{2n}$.

If $V_{x,3n+k-1} = '1'$, then $V_{x,3n+k-1} \cdots V_{x,n+k-1} > \underbrace{011\cdots1}_{2n}$, and the case of $V_{x,3n+k-1} \cdots V_{x,n+k-1} = \underbrace{011\cdots1}_{2n}$

can be detected by $V_{x,3n+k-1} = '0'$ and $V_{x,i} = '1' \forall i \in [3n+k-2, n+k-1]$. Hence, the condition of

$V_x \geq M_{R1}/2$ can be detected by the logic expression

$$V_{x,3n+k-1} \vee \left(\overline{V_{x,3n+k-1}} \wedge \underbrace{V_{x,3n+k-2} \wedge \cdots \wedge V_{x,n+k} \wedge V_{x,n+k-1}}_{2n} \right) = V_{x,3n+k-1} \vee \underbrace{V_{x,3n+k-2} \wedge \cdots \wedge V_{x,n+k} \wedge V_{x,n+k-1}}_{2n},$$

as shown on the right hand side of Figure 4.7 before the two-input AND gate.

Figure 4.7: Circuit implementation of sign detector module.

Based on the output of the magnitude comparators, the three mutually exclusive binary states of the comparison between $\tilde{X}$ and $\tilde{Y}$ are produced by the final relational inference module in

Figure 4.3. The relational inference module consists of two *AND* gates, one *OR* gate, one *XNOR* gate, one *NOR* gate and one 2-to-1 multiplexer, as shown in Figure 4.8.

Figure 4.8: Circuit implementation of relational inference module.

4.4 Evaluation and Comparison

4.4.1 Qualitative Comparison of General Features

In this subsection, the proposed algorithms for both S_1 and S_2 are qualitatively compared against the existing algorithms. The features being compared in Table 4.1 include the types of moduli set, whether redundant moduli (RM) is needed and whether the comparison of signed integers (SC) is supported. The algorithms of [Mil85], [Lu92], [Dim93] and [Wan99] are designed for arbitrary moduli set, which has limited exploitable number theoretic properties for hardware complexity reduction. For the algorithms of [Mil85] and [Lu92], only odd moduli can be used and redundant moduli are required for the determination of parities. The use of redundant moduli adds further hardware overheads. The algorithm of [Sou07], on the other hand, cannot be used for signed magnitude comparison. Its useful dynamic range $(2^{2n+2}-1)(2^{2n}-1)/3$ for unambiguous residue representation cannot be matched to that of S_1 and S_2 considered in this chapter by changing the value of n . Algorithms [Bi08] and [Zar10] are designed for three-moduli sets. For high-speed applications of large dynamic ranges, the parallelism may not be sufficient. The proposed algorithm is designed for two four-moduli supersets composed of equally arithmetic friendly moduli with one additional degree of parallelism. The extra parameter k for the power-of-two residue channel helps to balance the delay and provide greater flexibility in the selection of special moduli for different dynamic range requirements. The

major advantage of the proposed algorithm is its support for comparison of signed integers in RNS without the need for a complicated RNS sign detector. Although the algorithm of [Lu92] also supports signed integer comparison, lacking of detailed description on the computations of the absolute values of negative integers and the implementation of sign detection module, it is not possible to evaluate its overall complexity for the comparison against the proposed algorithm.

Table 4.1: Evaluation of RNS Magnitude Comparison Algorithms

Method	Moduli Set	Approach	RM?	SC?
[Mil85]	Arbitrary odd moduli	Parity check	Yes	No
[Lu92]	Arbitrary odd moduli	Parity check	Yes	Yes
[Dim93]	Arbitrary	Partial reverse conversion	No	No
[Wan99]	Arbitrary	Partial reverse conversion	No	No
[Sou07]	$\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$	Parity check	No	No
[Bi08]	$\{2^n-1, 2^n, 2^n+1\}$	Partial reverse conversion	No	No
[Zar10]	$\{2^n-1, 2^n+1, 2^n+3\}$	Parity check	No	No
This (S_1)	$\{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$	quantization	No	Yes
This (S_2)	$\{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$	quantization	No	Yes

4.4.2 Synthesis Results

The complexity of inter-modulo operations, such as sign detection, magnitude comparison and scaling, is highly dependent on the moduli set and grows with the number of moduli. These operations require multiplicative inverses and modulo reductions of composite residues and moduli, which cannot be computed from independent residues. Unlike the residue-to-binary converter in the last stage of an RNS, the residues are still needed for further processing in the same moduli set after these operations. The residue representations of the same inter-modulo operation for different moduli sets are incompatible even if the moduli sets may have the same dynamic range. Since there is no existing signed and unsigned integer magnitude comparator for moduli sets S_1 and S_2 in the literature, for a fair comparison, the proposed signed integer magnitude comparator is compared against the magnitude comparator technique in [Dim93] and [Wan99]. As the algorithms of [Dim93] and [Wan99] are designed for unrestricted arbitrary moduli sets, they can be implemented for S_1 and S_2 . Since their hardware architectures are not

presented in the literature, the best possible adder-based implementations of these two algorithms are considered in the presented results below. It should be noted that the original algorithms of [Dim93] and [Wan99] support the comparison of unsigned integers only. For the comparison with the proposed signed comparators in RNS, efficient sign detection method for S_1 and S_2 [Cha14] has been incorporated to convert them into signed magnitude comparators. In addition, signed magnitude comparators designed based on the most efficient reverse converter [Pat14] for both S_1 and S_2 followed by the binary comparators are also implemented and compared with the proposed magnitude comparators for signed integers.

All magnitude comparators are described in Verilog HDL and functionally verified by ModelSim. The Verilog code for each design is synthesized by Synopsys Design Compiler version E-2010.12-SP4 using STM CMOS 65 nm technology standard cell library. Each design is optimized for minimum delay. The measured area (A) in μm^2 and critical path delay (D) in ns of each design for S_1 and S_2 are obtained for different combinations of n and k , with $n = 8, 12$ and 16 and $k = 0, n/2$ and n . The measured area and delay of all designs are plotted in Figures 4.9 and 4.10 for S_1 and Figures 4.11 and 4.12 for S_2 , where Signed [Dim93] and Signed [Wan99] refer to the sign-detector supplemented magnitude comparators of [Dim93] and [Wan99], respectively. Each bar chart is divided into three sections corresponding to $n = 8, 12$ and 16 . In each section, the three values of $k = 0, n/2$ and n are labeled on the x axis.

Figure 4.9: Comparison of synthesized area of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.10: Comparison of delay of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.11: Comparison of synthesized area of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.12: Comparison of delay of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

From Figures 4.9 and 4.10, on average across all dynamic ranges in comparison, the proposed signed magnitude comparator designed for S_1 is 56.03% and 66.51% smaller, and 24.07% and 27.98% faster than the unsigned magnitude comparators of [Dim93] and [Wan99], respectively. Comparing with the sign-detector supplemented magnitude comparators of [Dim93] and [Wan99] and the signed magnitude comparator of [Pat14] for S_1 , the proposed design for the same moduli set saves on average 71.74%, 78.26% and 22.48% of area and reduces their critical path delay by 24.85%, 29.40% and 19.08%, respectively.

From Figures 4.11 and 4.12, on average over all dynamic ranges in comparison, the proposed design for S_2 reduces the areas of unsigned magnitude comparators of [Dim93] and [Wan99] by 53.21% and 64.58% and their delays by 25.31% and 30.75%, respectively. It is also on average 70.36%, 74.91% and 18.75% smaller and 27.05%, 31.38% and 16.41% faster than the sign detector supplemented magnitude comparators of [Dim93], [Wan99] and the signed magnitude comparator of [Pat14], respectively.

Monte Carlo power simulation [Bur93] is performed on each design using Synopsys Prime Time PX, where randomly generated input vectors are applied until the error of estimated average power is bounded below 2.5% with 99% confidence interval. Since the critical path delays of all designs in comparison are different and vary with the dynamic range, to ensure

that all designs will produce the correct outputs at the same data rate, the clock rate used for the input data is chosen based on the slowest design for each combination of n and k . The total power consumption in mW and the leakage power in nW are plotted in Figures 4.13 and 4.14 for S_1 and Figures 4.15 and 4.16 S_2 , respectively.

Figure 4.13: Comparison of total power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.14: Comparison of leakage power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Since faster design consumes more dynamic power, to compare the energy consumption of each design running at its full speed, the power-delay product is computed with the power consumption measured at the clock period corresponding to the critical path delay of each design independently. The percentage reduction of energy consumption by the proposed design

over the unsigned and sign-detector supplemented magnitude comparators of [Dim93] and [Wan99] and the signed magnitude comparator of [Pat14] are plotted in Figures 4.17 and 4.18 for S_1 and S_2 , respectively. Generally the large positive savings of energy consumption are achieved over all other designs in comparison, except for the case of $n = 8$ and $k = 0$ for S_2 , where the proposed design is observed to be 10.75% less energy efficient than [Pat14]. As both designs are equally fast, this is attributed to the slightly lower power consumption of [Pat14] when the dynamic range is low.

Figure 4.15: Comparison of total power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.16: Comparison of leakage power of the proposed design with the signed and unsigned magnitude comparators based on [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

The reason for the reverse converter based signed magnitude comparator to outperform [Dim93] and [Wan99] is mainly due to the best reverse converters [Pat14] designed specifically for S_1

and S_2 have maximally exploited the number theoretic properties unique to these two moduli sets, whereas the efficiency of the best implementations of [Dim93] and [Wan99] is restricted upfront by their algorithmic generality. The hardware efficiency between the proposed design and the design based on [Pat14] can also be analyzed at component level. The proposed design requires $2 \times \text{CPA mod } 2^{2n}-1$, $2 \times \text{CPA mod } 2^{n+1}-1$ ($\text{mod } 2^{n-1}-1$ for S_2), $2 \times \text{CSA mod } 2^{2n}-1$, $2 \times \text{CSA mod } 2^{n+1}-1$ ($\text{mod } 2^{n-1}-1$ for S_2), a $(n+1)$ -bit ($(n-1)$ -bit for S_2) binary magnitude comparator, a $(3n+k)$ -bit binary magnitude comparator and $2 \times (3n+3)$ ($2 \times 3n$ for S_2) basic gates for S_1 (S_2). The design based on [Pat14] requires $2 \times \text{CPA}$ $(3n+1)$ -bit ($(3n-1)$ -bit for S_2), $2 \times \text{CPA mod } 2^{2n}-1$, $2 \times \text{CPA mod } 2^{n+1}-1$ ($\text{mod } 2^{n-1}-1$ for S_2), $2 \times \text{CSA mod } 2^{2n}-1$, $4 \times \text{CSA mod } 2^{n+1}-1$ ($\text{mod } 2^{n-1}-1$ for S_2), a $(4n+k+1)$ -bit ($(4n+k-1)$ -bit for S_2) binary magnitude comparator and $2 \times (4n+k+1)$ -bit ($(4n+k-1)$ -bit for S_2) binary constant comparators. Both designs also require a relational inference module consisting of two to three basic gates and a 2-to-1 multiplexer. The component quantity is $2 \times$ or $4 \times$ when identical components are used in parallel for both operands X and Y . For the proposed design, the basic gates used for the “sign detector” and binary comparator can be computed in parallel with the $3n+k$ -bit binary magnitude comparator, whereas for the reverse converter based design [Pat14], the binary constant comparator and the binary magnitude comparator can be computed in parallel.

Figure 4.17: Percentage reduction in energy consumption of the proposed design over the magnitude comparators of [Dim93], [Wan99] and [Pat14] for S_1 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

Figure 4.18: Percentage reduction in energy consumption of the proposed design over the magnitude comparators of [Dim93], [Wan99] and [Pat14] for S_2 with $n = 8, 12, 16$ and $k = 0, n/2, n$.

4.5 Summary

A new technique to design signed magnitude comparator for two four-moduli supersets, $S_1 \equiv \{2^{n+k}, 2^n-1, 2^{n+1}, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^{n+1}, 2^{n+1}-1\}$, for even values of n and $k \in [0, n]$, has been proposed. The proposed algorithm inherits sign detection into the magnitude comparison by quantizing the dynamic range into equal number of subdivisions, which enables the sign of an integer to be readily identified from the rank where its residue representation resides. The overall complexity required to decode the relative magnitude of two residue representations has been greatly reduced due to the zoomed in range by their ranks. Based on the synthesis results, the most competitive existing signed magnitude comparators for these two moduli sets are the ones designed based on their most efficient reverse converters reported recently in [Pat14]. Even then, the proposed design can still perform on average 19.1% and 16.4% faster than [Pat14], and saves about 22.5% and 18.8% of their areas and conserves around 26.2% and 23.4% of their power consumptions for S_1 and S_2 , respectively.

The next chapter will present the first attempt to overcome the challenge of reducing the implementation complexity of signed residue comparator for an even higher cardinality moduli set with five balanced and arithmetic friendly moduli by leveraging on the scaling-assisted sign detection in a different manner.

Chapter 5

A Scaling-assisted Signed Integer Magnitude Comparison for Five Moduli Superset RNS $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$

5.1 Introduction

The workhorse of RNS is the residue arithmetic. Owing to their rich inherent number theoretic properties, special moduli of the form 2^i and $2^i \pm 1$ for $i \in \mathbb{Z}^+$ are preferred. That gives the triple moduli set $\{2^n-1, 2^n, 2^n+1\}$ presented earlier in Chapter 3 its unprecedented popularity in RNS-based computations [Cha15], [Kal00], [Efs04], [Cha11], [Mur12]. As the legitimate dynamic range of RNS is governed by the product of all its coprime moduli, the extra power/performance efficiency for high dynamic range computations needs to be harnessed by increasing the number of moduli without expanding the bitwidth of each residue. This is the motivation for the development of hardware-efficient magnitude comparison for the two balanced four-moduli supersets discussed in Chapter 4.

Balanced high-cardinality moduli sets with more than four modular arithmetic friendly moduli are hard to come by. Existing high-cardinality special moduli sets either contain some moduli that are less arithmetic friendly than 2^i and $2^i \pm 1$ or are imbalanced (i.e., the word lengths of their moduli are different by more than one bit) [Mol10], [Pat14], [Pet13b]. Moduli sets expanded from the balanced triple moduli set by including pairs of conjugate moduli [Ska99] to simplify their residue-to-binary converter design suffer from the reduced legitimate dynamic range due to the presence of non-coprime moduli. Thus the moduli set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ for even n remains the most useful coprime five-moduli set reported thus far, with simple hardware for all residue channels of RNS processor and efficient reverse converter [Cao07]. Even with high-cardinality special moduli set, the advantages of RNS may still be limited by operations that are trivial in weighted number system but difficult in RNS. Such

operations include scaling, sign detection and magnitude comparison. The inefficiency arises from the unordered residue digits, complicated multiplicative inverses and large composite modulo operations as discussed earlier. Unfortunately, increasing the cardinality of the moduli set further aggravates the complexity of these operations in RNS.

In this Chapter, a new algorithm is proposed for the comparison of two signed integers represented in $S \equiv \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ for even n . The approach is fundamentally different from all existing approaches in that it compares the difference of the two signed integers against the mid-point of the dynamic range. The inefficient sign detection problem that has prevented this intuitive method from being used in RNS is resolved by prescaling. By reducing the dynamic range of S to $2^n(2^{n+1}-1)$, the size of the resulting composite mod $2^n(2^{n+1}-1)$ sum for detecting the sign of the difference between two residue representations can be further decomposed into two parallel mod 2^n and mod $2^{n+1}-1$ sums. Hence, without using any lookup table, the algorithm can be implemented mainly by modular adders of no more than $n+1$ bit wide. These modulo operators can be reused for arithmetic processing of the residues in existing 2^n and $2^{n+1}-1$ modulus channels.

5.2 Scaling-Assisted Sign Detection Algorithm for Comparison of Signed

Integers in $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ RNS

In this section, a new algorithm is derived for the comparison of the residue representation of two signed integers, $\tilde{X}$ and $\tilde{Y}$, in the five moduli set, $S \equiv \{m_1, m_2, m_3, m_4, m_5\} = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ with even values of n . The comparison is performed by merely detecting the signs of the integers $\tilde{X}$, $\tilde{Y}$ and their difference, $\tilde{X} - \tilde{Y}$, in the residue domain followed by some simple relational logic. This is because the values of two unequal signed integers can be compared by the sign of their difference if they have the same sign and by the sign of one of the two integers if they are of opposite sign. The advantage of this intuitive approach is that the signs of $\tilde{X}$, $\tilde{Y}$ and $\tilde{X} - \tilde{Y}$ can be detected in parallel with three identical sign detectors. While sign detection is a trivial operation in two's complement number system, it is no less difficult

than magnitude comparison in RNS. Its complexity increases with the dynamic range. Hence, this simplistic idea requires a highly efficient implementation of sign detection in RNS, which is accomplished here by a unique but simple scaling-assisted sign detection method.

The idea is to shrink the dynamic range M of S such that sign detection can be performed at ease in the remapped residue domain by reducing the size of the modulo operations with better exploitation of unique number theoretic properties of the resulting composite moduli. To accomplish this, M of S is factored into two integers $M_{R1} \times M_{R2}$, where $M_{R1} = (2^{2n}-1)(2^{n-1}-1)$ and $M_{R2} = 2^n(2^{n+1}-1)$. By scaling all integers X in the range $[0, M)$ by a factor of M_{R1} , every residue representation of X in the lower half range $[0, M/2-1]$ is mapped to a residue representation of X_s in the range $[0, M_{R2}/2-1]$ and every residue representation of X in the upper half range $[M/2, M)$ is mapped to a residue representation of X_s in the range $[M_{R2}/2, M_{R2})$ of M_{R2} , as depicted in Figure 5.1. By shrinking the dynamic range from $[0, M)$ to $[0, M_{R2})$, the sign of $\tilde{X}$ can be detected by checking its scaled integer X_s against the halfway point of the reduced dynamic range of $M_{R2}/2 = 2^{n-1}(2^{n+1}-1)$.

Figure 5.1: The mapping of integers in $[0, M)$ to its scaled integers in $[0, M_{R2})$.

To obtain the scaled integer X_s from the residue representation of $\tilde{X}$, the integer X_{13} is first computed from the residues (x_1, x_3) with respect to $\{2^{n-1}, 2^n+1\}$ by CRT-II as follows:

$$\begin{aligned} X_{13} &= x_3 + \left| k_1 (x_1 - x_3) \right|_{2^{n-1}} (2^n + 1) \\ &= x_3 + A_x (2^n + 1) \end{aligned} \quad (5.1)$$

where $k_1 = \left| (2^n + 1)^{-1} \right|_{2^{n-1}} = 2^{n-1}$ and $A_x = \left| k_1 (x_1 - x_3) \right|_{2^{n-1}} = \left| 2^{n-1} (x_1 - x_3) \right|_{2^{n-1}}$.

By treating X_{13} as a residue of modulus $2^{2n}-1$, the integer $X_{135} = |X|_{(2^{2n}-1)(2^{n-1}-1)}$ can be computed by applying (2.18) on the residue (x_5, X_{13}) with respect to the moduli set $\{2^{n-1}-1, 2^{2n}-1\}$ as follows:

$$\begin{aligned} X_{135} &= X_{13} + |k_2 (x_5 - X_{13})|_{2^{n-1}-1} (2^{2n}-1) \\ &= X_{13} + B_x (2^{2n}-1) \end{aligned} \quad (5.2)$$

where $k_2 = \left| (2^{2n}-1)^{-1} \right|_{2^{n-1}-1} = \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} \right|_{2^{n-1}-1}$ as derived in [Pat14] and

$$B_x = |k_2 (x_5 - X_{13})|_{2^{n-1}-1}.$$

Equation (2.13) can be applied to a two moduli set $\{m_2, m_4\}$ to obtain an integer X_{24} from the residues (x_2, x_4) . With the multiplicative inverses, $|m_4^{-1}|_{m_2} = \left| (2^{n+1}-1)^{-1} \right|_{2^n} = -1$ and $|m_2^{-1}|_{m_4} = \left| (2^n)^{-1} \right|_{2^{n+1}-1} = 2$,

$$\begin{aligned} X_{24} &= |X|_{m_2 m_4} = |m_4 |m_4^{-1}|_{m_2} x_2 + m_2 |m_2^{-1}|_{m_4} x_4|_{m_2 m_4} \\ &= \left| (2^{n+1}-1) \times -1 \times x_2 + 2^n \times 2 \times x_4 \right|_{2^n (2^{n+1}-1)} \\ &= \left| -2^{n+1} x_2 + x_2 + 2^{n+1} x_4 \right|_{2^n (2^{n+1}-1)} \end{aligned} \quad (5.3)$$

Since, X_{135} and X_{24} can be treated as the residues of the moduli, $M_{R1} = (2^{2n}-1)(2^{n-1}-1)$ and $M_{R2} = 2^n(2^{n+1}-1)$, respectively, X can be expressed in a weighted form according to (2.18), i.e.,

$$X = X_{135} + M_{R1} |k_3 (X_{24} - X_{135})|_{M_{R2}} \quad (5.4)$$

where $k_3 = |M_{R1}^{-1}|_{M_{R2}} = \left| \left[(2^{2n}-1)(2^{n-1}-1) \right]^{-1} \right|_{2^n(2^{n+1}-1)}$.

By scaling X in (5.4) by M_{R1} , the scaled integer X_s is given by:

$$X_s = \left\lfloor \frac{X_{135}}{M_{R1}} \right\rfloor + \left\lfloor \frac{M_{R1} |k_3 (X_{24} - X_{135})|_{M_{R2}}}{M_{R1}} \right\rfloor \quad (5.5)$$

Since $X_{135} < M_{R1}$, $\lfloor X_{135}/M_{R1} \rfloor = 0$. X_s can be expressed as:

$$X_s = \left| k_3 (X_{24} - X_{135}) \right|_{2^n(2^{n+1}-1)} \quad (5.6)$$

By substituting the expression of X_{135} from (5.2) into (5.6),

$$X_s = \left| k_3 (X_{24} - X_{13} - B_x (2^{2n} - 1)) \right|_{2^n(2^{n+1}-1)} \quad (5.7)$$

Furthermore, by substituting the expressions of X_{13} and X_{24} from (5.1) and (5.3), respectively into (5.7), the final scaled integer $X_s \in \{0, M_{R2}\}$ can be fully evaluated from the residues $(x_1, x_2, x_3, x_4, x_5)$ as follows:

$$X_s = \left| k_3 \begin{pmatrix} -2^{n+1}x_2 + x_2 + 2^{n+1}x_4 - x_3 \\ -A_x(2^n + 1) - 2^{2n}B_x + B_x \end{pmatrix} \right|_{M_{R2}} \quad (5.8)$$

where $k_3 = \left[\left((2^{2n} - 1)(2^{n+1} - 1) \right)^{-1} \right]_{2^n(2^{n+1}-1)}$ is a constant that can be precomputed. A_x and B_x can

be computed from the residues x_1, x_3 and x_5 . Simplification of their hardware implementation will be discussed in the next section.

If $X_s \in [0, M_{R2}/2)$, X falls in the lower half range of M and $(x_1, x_2, x_3, x_4, x_5)$ represents a positive integer, i.e., $\tilde{X} \geq 0$. Otherwise, if $X_s \in [M_{R2}/2, M_{R2})$, X falls in the upper half range of M and $(x_1, x_2, x_3, x_4, x_5)$ represents a negative integer, i.e., $\tilde{X} < 0$. The conditions can be expressed as:

$$\text{sign}(\tilde{X}) = \begin{cases} 1 & X_s \geq M_{R2}/2 \\ 0 & \text{Otherwise} \end{cases} \quad (5.9)$$

The sign of the other integer $\tilde{Y}$ in comparison can be detected likewise by computing its scaled integer Y_s from its residue representation $(y_1, y_2, y_3, y_4, y_5)$ as in (5.8) followed by the comparison of Y_s with $M_{R2}/2$ as in (5.9).

Let $\tilde{Z} = \tilde{X} - \tilde{Y}$. The sign of $\tilde{Z}$ can be similarly detected by (5.9) from Z_s . The residues $z_i = |x_i - y_i|_{m_i}$ for $i = 1, 2, \dots, 5$. By replacing x_i by z_i in (5.8), the scaled integer Z_s can be calculated directly from the residues of $\tilde{X}$ and $\tilde{Y}$ as follows:

$$Z_s = \left\| k_3 \begin{pmatrix} -2^{n+1}x_2 + x_2 + 2^{n+1}x_4 + 2^{n+1}y_2 - y_2 \\ -2^{n+1}y_4 - |x_3 - y_3|_{m_3} - A_z(2^n + 1) \\ -2^{2n}B_z + B_z \end{pmatrix} \right\|_{M_{R2}} \quad (5.10)$$

where $A_z = |k_1(x_1 - y_1 - |x_3 - y_3|_{2^n+1})|_{2^n-1}$ and $B_z = |k_2(x_5 - y_5 - Z_{13})|_{2^{n-1}-1}$ with $Z_{13} = |x_3 - y_3|_{2^n+1} + A_z(2^n + 1)$. The constants k_1, k_2 and k_3 have the same expressions as before.

The flowchart in Figure 5.2 shows how the equality and strict inequality relations can be established from the residue representations $(x_1, x_2, x_3, x_4, x_5)$ and $(y_1, y_2, y_3, y_4, y_5)$ of two signed integers $\tilde{X}$ and $\tilde{Y}$, respectively with $\text{sign}(\tilde{X})$, $\text{sign}(\tilde{Y})$ and $\text{sign}(\tilde{X} - \tilde{Y})$ determined by the abovementioned scaling-assisted sign detection algorithm. Only two cases need to be considered for strict inequality. If $\tilde{X}$ and $\tilde{Y}$ have different signs, it is trivial to conclude that the positive integer is always larger than the negative integer. Otherwise, if $\tilde{X}$ and $\tilde{Y}$ are both positive or both negative, the sign of their difference, $\text{sign}(\tilde{X} - \tilde{Y})$, is considered. If $\tilde{X} - \tilde{Y}$ is positive, then $\tilde{X}$ is greater than $\tilde{Y}$. Otherwise, $\tilde{X}$ is less than $\tilde{Y}$. The equality of $\tilde{X}$ and $\tilde{Y}$ is evaluated exclusively by checking if all five pairs of residues, $\{x_i, y_i\}$ are equal. As long as there is one unequal pair, it will branch out to the detection of strict inequality relation.

Figure 5.2: Comparison of $(x_1, x_2, \dots, x_5)$ and $(y_1, y_2, \dots, y_5)$ with the help of scaling-assisted sign detection.

Example 5.1: For $n = 4$, $S \equiv \{15, 16, 17, 31, 7\}$. $M_{R1} = 15 \times 17 \times 7 = 1785$ and $M_{R2} = 16 \times 31 = 496$ and $M = 885360$. Let $\tilde{X} \equiv (11, 13, 1, 15, 4)$ and $\tilde{Y} \equiv (8, 4, 16, 23, 1)$ be the two residue representations to be compared in S . Since $k_1 = 2^{4-1} = 8$, A_x , A_y and A_z can be computed by (5.1) as $A_x = |8(11-1)|_{15} = 5$, $A_y = |8(8-16)|_{15} = |-64|_{15} = 11$ and $A_z = |8(11-8-|1-16|_{17})|_{15} = 8$ respectively. X_{13} , Y_{13} and Z_{13} are $1 + 5 \times 17 = 86$, $16 + 11 \times 17 = 203$ and $|1-16|_{17} + 8 \times 17 = 2 + 8 \times 17 = 138$, respectively. From (5.2), since $k_2 = |255^{-1}|_7 = 5$, $B_x = |5(4-86)|_7 = 3$, $B_y = |5(1-203)|_7 = 5$ and $B_z = |5(|4-1|_7 - 203)|_7 = 1$. From (5.8), since $k_3 = |1785^{-1}|_{496} = 329$, $X_s = |329 \times (-32 \times 13 + 13 + 32 \times 15 - 1 - 17 \times 5 - 256 \times 3 + 3)|_{496} = |329 \times -774|_{496} = |-254646|_{496} = 298$. Similarly, $Y_s = |329 \times (-32 \times 4 + 4 + 32 \times 23 - 16 - 17 \times 11 - 256 \times 5 + 5)|_{496} = |329 \times -866|_{496} = |-284914|_{496} = 286$. Similarly, from (5.10), $Z_s = |329 \times (77 - 612 - 2 - (17 \times 8) - 256 \times 1)|_{496} = |329 \times -928|_{496} = |-305312|_{496} = 224$. Since $M_{R2}/2 = 248$, $X_s, Y_s > M_{R2}/2$ and $Z_s < M_{R2}/2$. According to (5.9), $\text{sign}(\tilde{X}) = \text{sign}(\tilde{Y}) = 1$ and $\text{sign}(\tilde{X} - \tilde{Y}) = 0$. Since $x_1 = 11 \neq y_1 = 8$, from the flow chart of Figure 5.2, $\tilde{X} > \tilde{Y}$. Using (2.13), the reverse converted integer values of $(11, 13, 1, 15, 4)$ and $(8, 4, 16, 23, 1)$ are $X = 532781$ and $Y = 511988$. According to (2.3), $\tilde{X} =$

$|532781 + 442680|_{885360} - 442680 = -352579$ and $\tilde{Y} = |511988 + 442680|_{885360} - 442680 = -373372$, which verifies that $\tilde{X}$ is indeed greater than $\tilde{Y}$.

5.3 Proposed Hardware Architecture for Signed Integer Comparator in RNS

Figure 5.3 shows the overall architecture for the implementation of the algorithm proposed in Section 5.2, where the components and signals in the critical path are highlighted in red bold print. The inputs to the signed integer comparator are the residue representations, $(x_1, x_2, x_3, x_4, x_5)$ and $(y_1, y_2, y_3, y_4, y_5)$, of the two signed integers, $\tilde{X}$ and $\tilde{Y}$, in comparison. The main building blocks are the A_j and B_j generators for $j \in \{x, y, z\}$ and sign detector modules for $\tilde{X}$, $\tilde{Y}$ and $\tilde{Z}$. While the sign detectors for $\tilde{X}$, $\tilde{Y}$ and $\tilde{Z}$ are implemented in the same way, the A_z and B_z generators are implemented in a slightly different way from the A_j and B_j generators for $j \in \{x, y\}$. Also, X_s and Y_s share the same architecture that is slightly different from Z_s . The final status of comparison, $\tilde{X} > \tilde{Y}$, $\tilde{X} < \tilde{Y}$ or $\tilde{X} = \tilde{Y}$, is determined by the “Relational Inference” block. As the condition of $x_3 = y_3$ has been detected upfront by the condition $y_3 \geq x_3$ in an $(n+1)$ -bit binary comparator, the residues, x_3 and y_3 , need not be compared again in the “Relational Inference” block.

Figure 5.3: Overall architecture of the proposed RNS signed integer comparator.

In what follows, the circuit implementation of these building blocks is further elucidated. For notational convenience, let $x_{i,j}$ denotes the j -th bit of the binary representation of a residue x_i and $x_{i,j:k}$ denotes a concatenation of bits j to k of x_i . Thus, the five residues of $\tilde{X}$ can be represented as: $x_1 = x_{1,n-1:0} = x_{1,n-1} \cdots x_{1,1} x_{1,0}$, $x_2 = x_{2,n-1:0} = x_{2,n-1} \cdots x_{2,1} x_{2,0}$, $x_3 = x_{3,n:0} = x_{3,n} \cdots x_{3,1} x_{3,0}$, $x_4 = x_{4,n:0} = x_{4,n} \cdots x_{4,1} x_{4,0}$ and $x_5 = x_{5,n-2:0} = x_{5,n-2} \cdots x_{5,1} x_{5,0}$.

5.3.1 A_x Generator

From (5.1), $A_x = \left| 2^{n-1} (x_1 - x_3) \right|_{2^n-1}$. Since $\left| 2^n \right|_{2^n-1} = 1$, $\left| x_3 \right|_{2^n-1} = \left| 2^n x_{3,n} \right|_{2^n-1} + \left| \sum_{i=0}^{n-1} 2^i x_{3,n-1} \right|_{2^n-1}$
 $= \left| x_{3,n} + x_{3,n-1:0} \right|_{2^n-1}$. As $0 \leq x_3 \leq 2^n$, the only case for which $x_{3,n} = 1$ is when $x_3 = 2^n = \underbrace{10000}_n$.

When $x_{3,n} = 1$, $x_{3,0} = 0$. Therefore, $\left| x_3 \right|_{2^n-1}$ can be expressed as an equivalent n -bit binary string

$x'_3 = x_{3,n-1} \cdots x_{3,1} x_{3,0 \vee n}$ where binary bit $x_{3,0 \vee n} = x_{3,0} \vee x_{3,n}$ and $\vee$ denotes a logic OR operator.

By applying the circular left shift *Properties 2.11* and *2.12*, A_x can be expressed as:

$$\begin{aligned} A_x &= \left| 2^{n-1} x_1 - 2^{n-1} x_3 \right|_{2^n-1} \\ &= \left| CLS_n(x_1, n-1) + CLS_n(\bar{x}'_3, n-1) \right|_{2^n-1} \\ &= \left| x_{1,0} x_{1,n-1} \cdots x_{1,1} + \bar{x}_{3,0 \vee n} \bar{x}_{3,n-1} \cdots \bar{x}_{3,1} \right|_{2^n-1} \end{aligned} \quad (5.11)$$

Hence, A_x can be computed with only n inverters, a two-input NOR gate and a mod 2^n-1 adder as shown in Figure 5.4(a). The circular left shift operation is accomplished by re-routing the input signals at no logic cost.

Figure 5.4: Circuit modules required to compute (a) A_x and (b) B_x .

5.3.2 B_x Generator

Once A_x has been computed, B_x in (5.2) can be computed by

$$\begin{aligned}
 B_x &= \left| k_2 (x_5 - x_3 - A_x (2^n + 1)) \right|_{2^{n-1}-1} \\
 &= \left| k_2 (x_5 - x_3 - U) \right|_{2^{n-1}-1}
 \end{aligned} \tag{5.12}$$

where $k_2 = \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} \right|_{2^{n-1}-1}$ and $U = 2^n A_x + A_x = A_x \| A_x$.

The modulo reduction of operands larger than $n-1$ bits before the modulo summation of (5.12) can be avoided altogether by applying the concatenation *Property 2.13* of mod $2^{n-1}-1$ reduction. According to (2.11), the $(n+1)$ -bit residue x_3 can be represented as a mod $2^{n-1}-1$ sum of $x_3^{(0)} = x_{3,n-2} \cdots x_{3,1} x_{3,0}$ and $x_3^{(1)} = \underbrace{0 \cdots 0}_{n-3} x_{3,n} x_{3,n-1}$. Similarly, the $2n$ -bit integer $U = u_{2n-1} \dots u_1 u_0$ can also be represented as a mod $2^{n-1}-1$ sum of $U^{(0)} = u_{n-2} \cdots u_1 u_0$, $U^{(1)} = u_{2n-3} \cdots u_n u_{n-1}$ and $U^{(2)} = \underbrace{0 \cdots 0}_{n-3} u_{2n-1} u_{2n-2}$. Hence,

$$\begin{aligned}
 B_x &= \left| k_2 (x_5 + \overline{x_3^{(1)}} + \overline{x_3^{(0)}} + \overline{U^{(2)}} + \overline{U^{(1)}} + \overline{U^{(0)}}) \right|_{2^{n-1}-1} \\
 &= \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} (B_{x,s} + B_{x,c}) \right|_{2^{n-1}-1}
 \end{aligned} \tag{5.13}$$

In (5.13), the six $(n-1)$ -bit addends are summed by three stages of $(n-1)$ -bit carry-save adder with end-around carry (EAC-CSA). B_x can be further simplified by applying *Property 2.12* to the final $(n-1)$ -bit sum $B_{x,s}$ and carry $B_{x,c}$ outputs of the EAC-CSA tree as follows:

$$\begin{aligned}
B_x &= \left| \sum_{i=0}^{n/2-2} 2^{2i} (-2) (B_{x,s} + B_{x,c}) \right|_{2^{n-1}-1} \\
&= \left| \sum_{i=0}^{n/2-2} 2^{2i} \left(CLS_{n-1}(\overline{B_{x,s}}, 1) + CLS_{n-1}(\overline{B_{x,c}}, 1) \right) \right|_{2^{n-1}-1} \\
&= \left| \sum_{i=0}^{n/2-2} 4^i B'_x \right|_{2^{n-1}-1}
\end{aligned} \tag{5.14}$$

where $B'_x = CLS_{n-1}(\overline{B_{x,s}}, 1) + CLS_{n-1}(\overline{B_{x,c}}, 1)$.

It is noted that the constant $\sum_{i=0}^{n/2-2} 4^i$ is an $(n-4)$ -bit binary string of alternating ones and zeros, i.e., $\underbrace{101\cdots 0101}_{n-4}$. Thus, the number of addends in (5.14) can be greatly reduced by first

computing the common subexpression $2^2 B'_x + B'_x = CLS_{n-1}(\overline{B_{x,s}}, 3) + CLS_{n-1}(\overline{B_{x,c}}, 3) + CLS_{n-1}(\overline{B_{x,s}}, 1) + CLS_{n-1}(\overline{B_{x,c}}, 1)$ by two levels of $(n-1)$ -bit EAC-CSA, and then circularly left shifting it by consecutive multiple-of-four bit positions. Using the recursion of $B_j = 2^{2^j} B_{j-1} + B_{j-1}$, where $B_1 = 2^2 B'_x + B'_x$, these addends can be summed in logarithmic time complexity with j ($j \in \mathbb{Z}^+$ and $j > 1$) stages of two-level $(n-1)$ -bit EAC-CSA. For $n = 2^{j+1} + 2$, $j = \log_2(n-2) - 1$ and for $n = 4j + 2$, $j = (n-2)/4$. For other even integer values of n , one additional operand will be added to the sum of one of the above two cases. Finally, the sum and carry vectors are merged using a mod $2^{n-1}-1$ adder [Che09]. The complete circuit implementation of B_x is shown in Figure 5.4(b).

5.3.3 Computations of X_s and $\text{sign}(\tilde{X})$

Since $x_3 = |x_{3,n} + x_{3,n-1:0}|_{2^n-1}$, from (5.8), X_s can be expressed as:

$$\begin{aligned}
X_s &= \left| k_3 \begin{pmatrix} -2^{n+1}x_2 + x_2 + 2^{n+1}x_4 - 2^n x_{3,n} \\ -x_{3,n-1:0} - A_x(2^n + 1) - 2^{2n}B_x + B_x \end{pmatrix} \right|_{2^n(2^{n+1}-1)} \\
&= |k_3 V_x|_{2^n(2^{n+1}-1)}
\end{aligned} \tag{5.15}$$

According to *Property 2.10*, $-x_2 = |-x_2|_{2^n} = \bar{x}_2 + 1 - 2^n$, $-A_x = |-A_x|_{2^n} = \bar{A}_x + 1 - 2^n$, $-B_x = |-B_x|_{2^{n-1}} = \bar{B}_x + 1 - 2^{n-1}$, $-2^n x_{3,n} = 2^n |-x_{3,n}|_2 = 2^n (\bar{x}_{3,n} + 1 - 2) = 2^n (\bar{x}_{3,n} - 1)$ and $-x_{3,n-1:0} = |-x_{3,n-1:0}|_{2^n} = \bar{x}_{3,n-1:0} + 1 - 2^n$. V_x can be represented as:

$$V_x = \left| \begin{aligned} &2^{n+1}(\bar{x}_2 + 1 - 2^n) + x_2 + 2^{n+1}x_4 + 2^n(\bar{x}_{3,n} - 1) \\ &+ (\bar{x}_{3,n-1:0} + 1 - 2^n) + (2^n + 1)(\bar{A}_x + 1 - 2^n) \\ &+ 2^{2n}(\bar{B}_x + 1 - 2^{n-1}) + B_x \end{aligned} \right|_{2^n(2^{n+1}-1)} \tag{5.16}$$

The constant terms in (5.16) add up to $2 - 2^{2n+1} - 2^{3n-1}$. Since $2^{2n+1} + 2^{3n-1} = (2^{n+1} - 1)(2^n + 2^n \times 2^{n-2}) + 2^n + 2^n \times 2^{n-2}$, $|2^{2n+1} + 2^{3n-1}|_{2^n(2^{n+1}-1)} = |2^n + 2^n \times 2^{n-2}|_{2^n(2^{n+1}-1)}$. Hence,

$$V_x = \left| \begin{aligned} &2^{n+1}\bar{x}_2 + x_2 + 2^{n+1}x_4 + 2^n\bar{x}_{3,n} + \bar{x}_{3,n-1:0} \\ &+ (2^n + 1)\bar{A}_x + 2^{2n}\bar{B}_x + B_x \\ &+ 2 - 2^n - 2^n \times 2^{n-2} \end{aligned} \right|_{2^n(2^{n+1}-1)} \tag{5.17}$$

To simplify the mod $2^n(2^{n+1} - 1)$ sum of V_x , the following Lemma is defined.

Lemma 5.1: Let $(S, C) = (S_{2n:0}, C_{2n:0})$ be the stored carry representation of a mod $2^n(2^{n+1} - 1)$

sum of k integer variables such that $|S + C|_{2^n(2^{n+1}-1)} = \left| \sum_{i=1}^k t_i \right|_{2^n(2^{n+1}-1)}$. Then, (S, C) can be split into

two subvectors in stored carry form, $(S_{n-1:0}, C_{n-1:0})$ and $(S_{2n:n}, C_{2n:n})$, and computed individually by:

$$\begin{aligned}
(S_{n-1:0}, C_{n-1:0}) &= \left| \sum_{i=1}^k t_i \right|_{2^n} \\
(S_{2n:n}, C_{2n:n}) &= \left\lfloor \frac{\sum_{i=1}^k t_i}{2^n} \right\rfloor + \sum_{i=1}^{k-2} c_i \right|_{2^{n+1}-1}
\end{aligned} \tag{5.18}$$

where $c_i \in \{0,1\} \forall i = 1, 2, \dots, k-2$. c_i is the carries out of the mod 2^n operation.

Proof: Based on the remainder theorem, each $(2n+1)$ -bit integer variable t_i can be expressed as:

$$t_i = 2^n \left\lfloor \frac{t_i}{2^n} \right\rfloor + |t_i|_{2^n}. \text{ Hence, } \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n(2^{n+1}-1)} = \left\lfloor 2^n \sum_{i=1}^k \left\lfloor \frac{t_i}{2^n} \right\rfloor + \sum_{i=1}^k |t_i|_{2^n} \right\rfloor_{2^n(2^{n+1}-1)}.$$

The sum $\sum_{i=1}^k |t_i|_{2^n}$ can be computed by a binary CSA tree of $k-2$ units of n -bit CSAs. Each

CSA generates a carry, $c_i \in \{0, 1\}$ of weight 2^n . The carries generated out of the CSA tree add

up to $2^n \sum_{i=1}^{k-2} c_i$. Hence, $\sum_{i=1}^k |t_i|_{2^n} = \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n} + 2^n \sum_{i=1}^{k-2} c_i$ and $\left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n(2^{n+1}-1)} = \left\lfloor 2^n \left(\sum_{i=1}^k \left\lfloor \frac{t_i}{2^n} \right\rfloor + \sum_{i=1}^{k-2} c_i \right) \right\rfloor_{2^n(2^{n+1}-1)}$

$+ \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n}$. Based on the axiom of $|an_1|_{n_1 n_2} = n_1 |a|_{n_2}$ for $a, n_1, n_2 \in \mathbb{Z}^+$,

$$\begin{aligned} \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n(2^{n+1}-1)} &= 2^n \left\lfloor \left(\sum_{i=1}^k \left\lfloor \frac{t_i}{2^n} \right\rfloor + \sum_{i=1}^{k-2} c_i \right) \right\rfloor_{2^{n+1}-1} + \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n} \\ &= \left\lfloor \frac{\sum_{i=1}^k t_i}{2^n} \right\rfloor_{2^{n+1}-1} + \sum_{i=1}^{k-2} c_i \left\lfloor \left\lfloor \sum_{i=1}^k t_i \right\rfloor_{2^n} \right\rfloor_{2^{n+1}-1} \end{aligned}$$

Q.E.D.

According to *Lemma 5.1*, (5.18) can be decomposed into two smaller modular carry-save additions. The lower subwords of n LSBs of all integers are summed by a mod 2^n CSA tree while the upper subwords from bit n to the MSB are summed by a mod $2^{n+1}-1$ EAC-CSA tree as follows:

$$V_x = \left\lfloor \begin{array}{l} \bar{x}_{2,n-1:0} 0 + x_{4,n:0} 0 + x_{3,n} + \bar{A}_x \\ + 2^n \bar{B}_x - 1 - 2^{n-2} + c_1 + c_2 \end{array} \right\rfloor_{2^{n+1}-1} \parallel \left\lfloor \begin{array}{l} x_2 + \bar{x}_{3,n-1:0} \\ + \bar{A}_x + B_x + 2 \end{array} \right\rfloor_{2^n} \quad (5.19)$$

In (5.19), the terms inside the mod 2^n sum are added by two stages of n -bit binary CSA and the constant '2' is added by feeding a carry input of '1' into both stages of CSA to produce the sum and carry vectors $(S_{n-1:0}, C_{n-1:0})$. The carry outputs, c_1 and c_2 , of the two CSAs are fed into the mod $2^{n+1}-1$ EAC-CSA tree.

Using *Property 2.11*, $\left|2^n \bar{B}_x\right|_{2^{n+1}-1}$ in the mod $2^{n+1}-1$ sum can be expressed as an $(n+1)$ -bit binary string, $CLS_{n+1}(\bar{B}_x, n) = \bar{B}_{x,0} 00 \bar{B}_{x,n-2:1}$. The constant term $\left|-1-2^{n-2}\right|_{2^{n+1}-1}$ can also be represented as an $(n+1)$ -bit binary string, i.e., $\underbrace{1101}_{3} \cdots \underbrace{10}_{n-3} \underbrace{1}_{1}$. The terms within the mod $2^{n+1}-1$ sum can be compacted into the following five $(n+1)$ -bit terms by replacing the zero bits in some terms by the binary variables from other terms in the same bit positions.

$$\begin{aligned}
t_1 &= \underbrace{\bar{x}_{2,n-1} \cdots \bar{x}_{2,1} \bar{x}_{2,0}}_n \underbrace{x_{3,n}}_1 \\
t_2 &= \underbrace{x_{4,n} 00}_{3} \underbrace{\bar{B}_{x,n-2} \cdots \bar{B}_{x,2} \bar{B}_{x,1}}_{n-2} \\
t_3 &= \underbrace{\bar{B}_{x,0}}_1 \underbrace{\bar{A}_{x,n-1} \cdots \bar{A}_{x,1} \bar{A}_{x,0}}_n \\
t_4 &= \underbrace{x_{4,n-1} \cdots x_{4,0}}_n \underbrace{C_1}_1 \\
t_5 &= \underbrace{1101}_{3} \cdots \underbrace{1}_{n-3} \underbrace{C_2}_1
\end{aligned} \tag{5.20}$$

The mod $2^{n+1}-1$ sum of these five operands can be computed by three stages of $(n+1)$ -bit EAC-CSA to produce the sum and carry vectors $(S_{2n:n}, C_{2n:n})$. Finally, the sum and carry vectors (S, C) of V_x are obtained by concatenation at no logic cost, i.e., $(S, C) = (S_{2n:n} \| S_{n-1:0}, C_{2n:n} \| C_{n-1:0})$.

From (5.15), X_s can be expressed in stored carry form as

$$X_s = \left| k_3 (S, C) \right|_{2^n(2^{n+1}-1)} \equiv \left| k_3 S + k_3 C \right|_{2^n(2^{n+1}-1)} \tag{5.21}$$

Although a closed-form expression for k_3 is not available, k_3 for any even integer n can be computed by the extended Euclidean algorithm. For a given n , the constant k_3 can be expressed as a sum of power-of-two terms as follows:

$$k_3 = \sum_{i=0}^{2n} 2^i k_{3,i} \tag{5.22}$$

where $k_{3,i} \in \{0,1\}$ for binary representation and $k_{3,i} \in \{0,1,-1\}$ for binary signed digit representation.

Consequently, $|k_3 S|_{2^n(2^{n+1}-1)} + |k_3 C|_{2^n(2^{n+1}-1)}$ can be collectively expressed as a single sum

$\left| \sum_{j=1}^l t_j \right|_{2^n(2^{n+1}-1)}$, where l is the number of nonzero bits of k_3 and t_j consists of a left-shifted sum

vector S and a left-shifted carry vector C of V_x by i bits for every nonzero bit of k_3 . Hence, X_s can be calculated by *Lemma 5.1* as follows:

$$\begin{aligned} (S'_{n-1:0}, C'_{n-1:0}) &= \left\lfloor \sum_{k_{3,i} \neq 0, i \in [n-1,0]} 2^i (C + S) \right\rfloor_{2^n} \\ (S'_{2n:n}, C'_{2n:n}) &= \left\lfloor \sum_{k_{3,i} \neq 0, i \in [2n,0]} 2^{i-n} (C + S) \right\rfloor + \sum_{i=1}^{k-2} c_i \Big|_{2^{n+1}-1} \end{aligned} \quad (5.23)$$

where $(S', C') = (S'_{2n:n} \| S'_{n-1:0}, C'_{2n:n} \| C'_{n-1:0})$ are the sum and carry vectors of X_s , and $c_i \in \{0,1\}$

$\forall i = 1, 2, \dots, k-2$ are the carries generated from the k CSAs for the mod 2^n sum.

To reduce the number of terms in (5.23), k_3 is represented by canonical signed digit representation (CSD) [Hew00], which has the minimum number of nonzero digits among the binary signed digit representations.

Table 5.1: k_3 in decimal and $\lfloor k_3/2^n \rfloor$ in binary and CSD representations.

n	k_3 (Dec.)	$\lfloor k_3/2^n \rfloor$ (Bin.)	$\lfloor k_3/2^n \rfloor$ (CSD, $\underline{1} \equiv -1$)
2	15	11	10 <u>1</u>
4	329	10100	10100
6	7777	1111001	1000 <u>1</u> 001
8	36737	10001111	1001000 <u>1</u>
10	1282561	10011100100	10100 <u>1</u> 00100
12	31692801	1111000111001	1000 <u>1</u> 00100 <u>1</u> 001
14	149151745	10001110001111	100100 <u>1</u> 001000 <u>1</u>
16	5249466369	10011100011100100	10100 <u>1</u> 00100 <u>1</u> 00100
18	129803616257	1111000111000111001	1000 <u>1</u> 00100 <u>1</u> 00100 <u>1</u> 001
20	610841133057	10001110001110001111	100100 <u>1</u> 00100 <u>1</u> 001000 <u>1</u>

Table 5.1 shows the decimal representation of k_3 for $n \leq 20$ and its binary and CSD representations for $\lfloor k_3/2^n \rfloor$, where the signed digit -1 is denoted by $\underline{1}$ in CSD. The least significant n bits of k_3 are not shown because it is equal to $1\underbrace{0\cdots 0}_{n-2}1$ for all n .

As $|k_3|_{2^n} = 2^{n-1} + 1$, $|k_3|_{2^n} (C + S) = S_{2n:0} \underbrace{0\cdots 0}_{n-1} + C_{2n:0} \underbrace{0\cdots 0}_{n-1} + S_{2n:0} + C_{2n:0}$. From (5.23),

$$(S'_{n-1:0}, C'_{n-1:0}) = \left\lfloor \frac{S_0 \underbrace{0\cdots 0}_1 + C_0 \underbrace{0\cdots 0}_1 + S_{n-1:0} + C_{n-1:0}}{2^n} \right\rfloor \quad \text{and} \quad (S'_{2n:n}, C'_{2n:n}) =$$

$$\left\lfloor \frac{\sum_{t_{3,j} \neq 0, i \in [2n,n]} 2^{i-n} (C + S)}{2^{n+1}-1} + S_{2n:1} + C_{2n:1} + S_{2n:n} + C_{2n:n} + \sum_{i=1}^{k-2} C_i \right\rfloor_{2^{n+1}-1}. \quad \text{Since only the MSB of the first two terms in the mod } 2^n$$

sum is nonzero, $(S'_{n-1:0}, C'_{n-1:0})$ can be computed by using only one n -bit CSA and a half adder.

The half adder produces a sum bit $s = S_0 \oplus C_0$ to merge the first two terms and a carry bit $c_1 =$

$S_0 \wedge C_0$ into the mod $2^{n+1}-1$ sum. In addition, according to *Property 2.13*, $S_{2n:1}$ and $C_{2n:1}$

are divided into $|00S_{2n:n+2} + S_{n+1:1}|_{2^{n+1}-1}$ and $|00C_{2n:n+2} + C_{n+1:1}|_{2^{n+1}-1}$. Similarly, S and C in the

mod $2^{n+1}-1$ sum of (5.23) are also decomposed into $(n+1)$ -bit terms, $0S_{2n:n+1}$, $S_{n:0}$, $0C_{2n:n+1}$ and

$C_{n:0}$, before they are circularly left shifted by $i-n$ bits according to *Property 2.12* for all $k_{3,i} \neq 0$.

Hence,

$$\begin{aligned} (S'_{n-1:0}, C'_{n-1:0}) &= \left\lfloor \frac{S_0 \oplus C_0 \underbrace{0\cdots 0}_{n-1} + S_{n-1:0} + C_{n-1:0}}{2^n} \right\rfloor \\ (S'_{2n:n}, C'_{2n:n}) &= \left\lfloor \frac{\sum_{k_{3,i} \neq 0, i \in [2n,n]} t_i + 00S_{2n:n+2} + S_{n+1:1} + 00C_{2n:n+2} + C_{n+1:1} + S_{2n:n} + C_{2n:n} + S_0 \wedge C_0 + c_2}{2^{n+1}-1} \right\rfloor \end{aligned} \quad (5.24)$$

where $t_i = CLS_{n+1}(0S_{2n:n+1}, i-n) + CLS_{n+1}(S_{n:0}, i-n) + CLS_{n+1}(0C_{2n:n+1}, i-n) +$

$CLS_{n+1}(C_{n:0}, i-n)$ and c_2 is the carry generated by the n -bit CSA of the mod 2^n sum.

Finally, (S', C') can be merged by a mod $2^{n+1}-1$ adder and an n -bit binary adder to obtain X_s as follows:

$$X_s = \left| S'_{2n:n} + C'_{2n:n} + c_3 \right|_{2^{n+1}-1} \parallel \left| S'_{n-1:0} + C'_{n-1:0} \right|_{2^n} \quad (5.25)$$

where c_3 is the carry output of the n -bit binary adder. Since $(S'_{n-1:0}, C'_{n-1:0})$ can be computed much faster than $(S'_{2n:n}, C'_{2n:n})$ in (5.24), c_3 , c_2 and $S_0 \wedge C_0$ can be combined by a full adder and fed as an operand into the last stage of EAC-CSA of the mod $2^{n+1}-1$ adder. Thus, the critical path delay of the final vector merged adder (VMA) of (5.25) is determined solely by the mod $2^{n+1}-1$ adder, which is significantly shorter than using a mod $2^n(2^{n+1}-1)$ adder to sum S' and C' directly.

Figure 5.5: Architecture for the generation of $X_{s,2n:n-1}$, (b) sign detector of $\tilde{X}$.

From (5.9), $\text{sign}(\tilde{X})$ can be detected by comparing X_s against $M_{R2}/2 = 2^{n-1}(2^{n+1}-1) = 0\underbrace{11\dots1}_{n+1}\underbrace{00\dots0}_{n-1}$. Since the least significant $n-1$ bits of $M_{R2}/2$ are all zeros, only the most significant $n+2$ bits of X_s need to be compared, i.e., if $X_{s,2n} \dots X_{s,n-1} \geq \underbrace{011\dots1}_{n+1}$, then $\text{sign}(\tilde{X}) = 1$. Otherwise, $\text{sign}(\tilde{X}) = 0$. $X_{s,2n} \dots X_{s,n-1} > \underbrace{011\dots1}_{n+1}$ as long as $X_{s,2n} = '1'$ and $X_{s,2n} \dots X_{s,n-1}$

$= 0 \underbrace{11 \cdots 1}_{2n}$ iff $X_{s,2n} = '0'$ and $X_{s,i} = '1' \forall i \in [2n-1, n-1]$. Hence, the condition of $X_s \geq M_{R2}/2$

can be determined by the logic function $X_{s,2n} \vee \left(\overline{X_{s,2n}} \wedge \left(\bigwedge_{i=n-1}^{2n-1} X_{s,i} \right) \right) = X_{s,2n} \vee \left(\bigwedge_{i=n-1}^{2n-1} X_{s,i} \right)$. As

the complete n -bit VMA for $(S'_{n-1:0}, C'_{n-1:0})$ is not required, substantial hardware can be saved by keeping only the carry generation network for c_3 and the subcircuit for generating only the MSB $X_{s,n-1}$ of the sum. Figure 5.5(a) shows the circuit architecture for the generation of $X_{s,2n:n-1}$. The circuit for the detection of $\text{sign}(\tilde{X})$ is shown in Figure 5.5(b).

5.3.4 Computations of A_z, B_z, Z_s and $\text{sign}(\tilde{Z})$

$|x_3 - y_3|_{2^{n+1}}$ is a common term in A_z, B_z and Z_s . If $x_3 < y_3$, $|x_3 - y_3|_{2^{n+1}} = x_3 - y_3 + 2^n + 1$; otherwise, $|x_3 - y_3|_{2^{n+1}} = x_3 - y_3$. Therefore, $|x_3 - y_3|_{2^{n+1}}$ can be replaced by $x_3 - y_3 + c(2^n + 1)$, where $c \in \{0, 1\}$ with $c = 1$ if $x_3 < y_3$ and 0 otherwise. Once c has been computed by an $(n+1)$ -bit binary comparator of x_3 and y_3 , the hardware architectures of A_z, B_z and Z_s can be derived in a similar way as A_x, B_x and Z_x .

Since $k_1 = 2^{n-1}$, A_z can be expressed as:

$$\begin{aligned} A_z &= \left| 2^{n-1} (x_1 - y_1 - x_3 + y_3 - c(2^n + 1)) \right|_{2^{n-1}} \\ &= \left| 2^{n-1} (x_1 - y_1 - x_3 + y_3 - 2c) \right|_{2^{n-1}} \end{aligned} \quad (5.26)$$

The constant term $|-2c|_{2^{n-1}}$ can be expressed as an n -bit binary string, $\underbrace{c \cdots c}_{n-2} 0c$. Following the

same modulo $2^n - 1$ reduction of $(n+1)$ -bit residues as in the derivation of (5.11), $|x_3|_{2^{n-1}} = x'_3 =$

$x_{3,n-1} \cdots x_{3,1} x_{3,0 \vee n}$ and $|y_3|_{2^{n-1}} = y'_3 = y_{3,n-1} \cdots y_{3,1} y_{3,0 \vee n}$, where $x_{3,0 \vee n} = x_{3,0} \vee x_{3,n}$ and

$y_{3,0 \vee n} = y_{3,0} \vee y_{3,n}$. Therefore, A_z can be rewritten as:

$$\begin{aligned} A_z &= \left| 2^{n-1} x_1 + 2^{n-1} \bar{y}_1 + 2^{n-1} \bar{x}_3 + 2^{n-1} y_3 + 2^{n-1} \underbrace{c \cdots c}_{n-2} 0c \right|_{2^{n-1}} \\ &= |a_1 + a_2 + a_3 + a_4 + a_5|_{2^{n-1}} \end{aligned} \quad (5.27)$$

where

$$\begin{aligned}
a_1 &= \left| 2^{n-1} x_1 \right|_{2^n-1} = CLS_n(x_1, n-1) = x_{1,0} x_{1,n-1} \cdots x_{1,1} , \\
a_2 &= \left| 2^{n-1} \bar{y}_1 \right|_{2^n-1} = CLS_n(\bar{y}_1, n-1) = \bar{y}_{1,0} \bar{y}_{1,n-1} \cdots \bar{y}_{1,1} , \\
a_3 &= \left| 2^{n-1} \bar{x}_3 \right|_{2^n-1} = CLS_n(\bar{x}_3, n-1) = \bar{x}_{3,0 \vee n} \bar{x}_{3,n-1} \cdots \bar{x}_{3,1} , \\
a_4 &= \left| 2^{n-1} y_3 \right|_{2^n-1} = CLS_n(y'_3, n-1) = y_{3,0 \vee n} y_{3,n-1} \cdots y_{3,1} , \text{ and} \\
a_5 &= \left| 2^{n-1} \underbrace{c \cdots c}_{n-2} 0 c \right|_{2^n-1} = CLS_n\left(\underbrace{c \cdots c}_{n-2} 0 c, n-1\right) = \underbrace{c \cdots c}_{n-1} 0 .
\end{aligned}$$

Figure 5.6(a) shows the implementation of A_z , where the operands a_1 , a_2 , a_3 , a_4 and a_5 are added by three stages of n -bit EAC-CSA followed by a mod 2^n-1 adder.

$$B_z = \left| k_2 (x_5 - y_5 - Z_{13}) \right|_{2^{n-1}-1} . \text{ By substituting } Z_{13} = \left| x_3 - y_3 \right|_{2^n+1} + A_z (2^n + 1) ,$$

$$\begin{aligned}
B_z &= \left| k_2 (x_5 - y_5 - \left| x_3 - y_3 \right|_{2^n+1} - A_z (2^n + 1)) \right|_{2^{n-1}-1} \\
&= \left| k_2 (x_5 - y_5 - (x_3 - y_3 + (2^n + 1)c) - V) \right|_{2^{n-1}-1} \\
&= \left| k_2 (x_5 - y_5 - x_3 + y_3 - 3c - V) \right|_{2^{n-1}-1}
\end{aligned} \tag{5.28}$$

where $V = A_z \parallel A_z$.

Following the same derivation of (5.13), $x_3 = \left| x_3^{(1)} + x_3^{(0)} \right|_{2^{n-1}-1}$, $y_3 = \left| y_3^{(1)} + y_3^{(0)} \right|_{2^{n-1}-1}$ and $U =$

$\left| U^{(2)} + U^{(1)} + U^{(0)} \right|_{2^{n-1}-1}$. Furthermore, $y_3^{(1)} = \underbrace{0 \cdots 0}_{n-3} y_{3,n} y_{3,n-1}$ and $\left| -3c \right|_{2^{n-1}-1} = \underbrace{c \cdots c}_{n-3} 00$ can be

merged into one term $y_3^{(1)'} = \underbrace{c \cdots c}_{n-3} y_{3,n} y_{3,n-1}$. Therefore, B_z can be expressed as:

$$\begin{aligned}
B_z &= \left| k_2 \left(\begin{array}{l} x_5 + \bar{y}_5 + \overline{x_3^{(1)}} + \overline{x_3^{(0)}} + y_3^{(1)'} \\ + y_3^{(0)} + \overline{U^{(2)}} + \overline{U^{(1)}} + \overline{U^{(0)}} \end{array} \right) \right|_{2^{n-1}-1} \\
&= \left| -2 \sum_{i=0}^{n/2-2} 2^{2i} (B_{z,s} + B_{z,c}) \right|_{2^{n-1}-1} \\
&= \left| \sum_{i=0}^{n/2-2} 4^i (CLS_{n-1}(\overline{B_{z,s}}, 1) + CLS_{n-1}(\overline{B_{z,c}}, 1)) \right|_{2^{n-1}-1}
\end{aligned} \tag{5.29}$$

The nine $(n-1)$ -bit operands are summed in a four-stage $(n-1)$ -bit EAC-CSA tree to produce the sum and carry vectors, $B_{z,s}$ and $B_{z,c}$. B_z can be computed from $B_{z,s}$ and $B_{z,c}$ by the same

circuit used for the evaluation of $\sum_{i=0}^{n/2-2} 4^i B'_x$ in (5.14). The circuit architecture of B_z is shown in

Figure 5.6(b).

Figure 5.6: Architecture for the generation of (a) A_z and (b) B_z .

From (5.10), Z_s can be expressed as:

$$\begin{aligned}
 Z_s &= \left| k_3 \begin{pmatrix} -2^{n+1}x_2 + x_2 + 2^{n+1}x_4 + 2^{n+1}y_2 - y_2 \\ -2^{n+1}y_4 - x_3 + y_3 - c(2^n + 1) \\ -A_z(2^n + 1) - 2^{2n}B_z + B_z \end{pmatrix} \right|_{2^n(2^{n+1}-1)} \\
 &= |k_3 V_z|_{2^n(2^{n+1}-1)}
 \end{aligned} \tag{5.30}$$

Comparing V_z of (5.30) with V_x of (5.15), the extra terms in V_z are $2^{n+1}y_2$, $-y_2$, $-2^{n+1}y_4$, y_3 and $-c(2^n+1)$. With A_x replaced by A_z and B_x replaced by B_z , the common terms between V_z and V_x can be added in a similar manner as V_x . The resulting sum vector S and carry vector C are then added with the extra terms as follows:

$$V_z = \left| \begin{pmatrix} S + C + 2^{n+1}y_2 - y_2 \\ -2^{n+1}y_4 + y_3 - c(2^n + 1) \end{pmatrix} \right|_{2^n(2^{n+1}-1)} \tag{5.31}$$

According to *Property 2.10*, $-y_2 = \bar{y}_2 + 1 - 2^n$ and $-y_4 = \bar{y}_4 - (2^{n+1} - 1)$. By substituting them into (5.31) and rearranging the terms, V_z can be written as:

$$V_z = \left| \begin{array}{c} S + C + 2^{n+1} y_2 + \bar{y}_2 + 2^{n+1} \bar{y}_4 + y_3 \\ -2^{n+1}(2^{n+1} - 1) + 1 - 2^n - c(2^n + 1) \end{array} \right|_{2^n(2^{n+1}-1)} \quad (5.32)$$

Since $\left| 2^{n+1}(2^{n+1} - 1) \right|_{2^n(2^{n+1}-1)} = 0$ and $\left| 1 - 2^n - c(2^n + 1) \right|_{2^n(2^{n+1}-1)} = \left| -2^n(1+c) + 1 - c \right|_{2^n(2^{n+1}-1)} = \underbrace{1 \dots 1}_{n-1} \bar{c} \underbrace{0 \dots 0}_{n-1} \bar{c}$, V_z can be rewritten as:

$$V_z = \left| \begin{array}{c} S + C + 2^{n+1} y_2 + \bar{y}_2 + 2^{n+1} \bar{y}_4 \\ + y_3 + \underbrace{1 \dots 1}_{n-1} \bar{c} \underbrace{0 \dots 0}_{n-1} \bar{c} \end{array} \right|_{2^n(2^{n+1}-1)} \quad (5.33)$$

By *Lemma 5.1*, V_z can be decomposed into two smaller modular sums with $y_3 = 2^n y_{3,n} + y_{3,n-1:0}$.

$$V_z = \left| \begin{array}{c} S_{2n:n} + C_{2n:n} + y_{2,n-1:0} 0 \\ + y_{3,n} + 2\bar{y}_4 + \underbrace{1 \dots 1}_{n-1} \bar{c} \bar{c} \\ + c_1 + c_2 \end{array} \right|_{2^{n+1}-1} \quad \parallel \quad \left| \begin{array}{c} \bar{y}_2 + y_{3,n-1:0} + S_{n-1:0} \\ + C_{n-1:0} + \bar{c} \end{array} \right|_{2^n} \quad (5.34)$$

The four n -bit operands in the mod 2^n sum can be added by two stages of n -bit binary CSA, with $\bar{c}$ fed into the carry input of one of the CSAs, to produce an n -bit sum $S_{z,n-1:0}$ and an n -bit carry $C_{z,n-1:0}$. The carry outputs, c_1 and c_2 , of the two CSAs are forwarded to the mod $2^{n+1} - 1$ sum.

By applying *Property 2.11*, $\left| 2\bar{y}_4 \right|_{2^{n+1}-1} = CLS_{n+1}(\bar{y}_4, 1) = \bar{y}_{4,n-1:0} \bar{y}_{4,n}$. Also, $y_{2,n-1:0} 0 + y_{3,n}$ can be merged into one term $y_{2,n-1:0} y_{3,n}$. Therefore, the sum and carry vectors, $S_{z,2n:n}$ and $C_{z,2n:n}$ of the mod $2^{n+1} - 1$ sum of V_z can be simplified to:

$$(S_{z,2n:n}, C_{z,2n:n}) = \left| \begin{array}{c} S_{2n:n} + C_{2n:n} + y_{2,n-1:0} y_{3,n} \\ + \bar{y}_{4,n-1:0} \bar{y}_{4,n} + \underbrace{1 \dots 1}_{n-1} \bar{c} \bar{c} + c_1 + c_2 \end{array} \right|_{2^{n+1}-1} \quad (5.35)$$

The first five $(n+1)$ -bit terms are added by three stages of $(n+1)$ -bit EAC-CSA followed by a simplified $(n+1)$ -bit EAC-CSA with one of its input equal to $\underbrace{0 \cdots 0}_{n-1} \underbrace{c_1 \wedge c_2}_1 \underbrace{c_1 \oplus c_2}_1$. Eventually, the sum and carry vectors of V_z are given by $(S_z, C_z) = (S_{z,2n:n} \parallel S_{z,n-1:0}, C_{z,2n:n} \parallel C_{z,n-1:0})$.

From (5.30), Z_s can be expressed as

$$Z_s = \left| k_3 (S_z, C_z) \right|_{2^n(2^{n+1}-1)} \equiv \left| k_3 (S_z + C_z) \right|_{2^n(2^{n+1}-1)} \quad (5.36)$$

Subsequent derivation towards the implementation of $(S'_{z,n-1:0}, C'_{z,n-1:0})$ and $(S'_{z,2n:n}, C'_{z,2n:n})$ of Z_s follows exactly the same process as (5.21) to (5.25) of X_s . Similar to (5.25), these vectors are merged by a modulo $2^{n+1}-1$ adder and an n -bit binary adder to obtain the final Z_s as follows:

$$Z_s = \left| S'_{z,2n:n} + C'_{z,2n:n} + c_3 \right|_{2^{n+1}-1} \parallel \left| S'_{z,n-1:0} + C'_{z,n-1:0} \right|_{2^n} \quad (5.37)$$

Again, as the condition of $Z_s \geq M_{R2}/2$ for $\text{sign}(\tilde{Z}) = 1$ can be detected by the logic function

$Z_{s,2n} \vee \left(\bigwedge_{i=n-1}^{2n-1} Z_{s,i} \right)$, only part of the n -bit VMA required to generate $Z_{s,2n} \cdots Z_{s,n-1}$ of $S'_{z,n-1:0} + C'_{z,n-1:0}$ need to be implemented, as depicted in Figure 5.7.

Figure 5.7: Architecture for the generation of $Z_{s,2n:n-1}$.

5.3.5 Relational Inference

Based on the outputs of $\text{sign}(\tilde{X})$, $\text{sign}(\tilde{Y})$ and $\text{sign}(\tilde{X} - \tilde{Y})$, the three mutually exclusive binary states of the comparison between $\tilde{X}$ and $\tilde{Y}$ are determined by a relational inference module. As shown in Figure 5.8, it consists of two *AND* gates, one *XNOR* gate, one *NOR* gate, one 2-to-1 multiplexer, $4n$ two-input *XNOR* gates and four wide input *AND* gates (i.e., two with n inputs, one with $n+1$ inputs and one with $n-1$ inputs). The latter are used for the detection of $x_i = y_i$ for $i = 1$ to 5 except $i = 3$ as the condition $x_3 = y_3$ has already been detected upfront before the computation of X_5 .

Figure 5.8: Circuit implementation of relational inference module. The inset (right) is the subcircuit for the detection of $x_i = y_i$ for $i = 1$ to 5 except $i = 3$, where $p = \lceil \log_2 m_i \rceil - 1$.

5.4 Comparison and Discussions

It is important to note that the complexity of an RNS comparator is highly dependent on the moduli set and grows with the number of moduli. Unlike the residue-to-binary converter that is used in the last stage of an RNS, the residues in comparison are usually needed for other operations in the same moduli set after the comparison. Therefore, magnitude comparators designed for special moduli sets with different number of equally arithmetic friendly moduli are incomparable even if they can be made to possess the same dynamic range. Unfortunately, algorithms that are not designed for arbitrary moduli set are generally inextensible to a different moduli set. For these reasons, a qualitative comparison among existing RNS comparators for different moduli sets is first made before the experimental results of the proposed design are evaluated against two alternative implementations adapted from CRT-II-based magnitude comparator for arbitrary moduli set and the hybrid solution based on the most efficient residue-to-binary converter for the same moduli set.

5.4.1 Qualitative Comparison

In Table 5.2, the proposed algorithm is compared qualitatively against the existing algorithms for the design of RNS comparator in terms of their target moduli sets, whether redundant moduli (RM) are required and whether the algorithm is applicable to comparison of residue representations of sign integers (SC) with negligible additional hardware cost. The algorithms of [Mil85], [Lu92], [Dim93] and [Wan99] are designed for arbitrary moduli set. Due to the limited number theoretic properties of generic moduli, lookup tables are required for their

hardware implementation, which are less area-time efficient than fully adder-based implementations. For the algorithms in [Mil85] and [Lu92], the designs are applicable to moduli sets with odd dynamic range. Redundant moduli are required in these methods to determine the parities, which incur substantial hardware overheads. The algorithm in [Sou07] is also based on parity check approach, and is designed for the four-moduli set $\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$ that contains two pairs of conjugate moduli. The non-coprime moduli complicate the design of reverse converter and other inter-modulo operations. More severely, its legitimate dynamic range is reduced to only one third of the product of all moduli, which makes this moduli set less useful. Besides, it is dedicated to comparison of unsigned integers and cannot be easily extended to compare residue representations of signed integers due to the complexity of sign detection for this moduli set. The algorithms in [Zar10], [Tor16a] and [Bi08] are designed for three-moduli sets. The speedup of integer arithmetic in these moduli sets is restricted by the smaller degree of parallelism for large dynamic range whereas the algorithm in [Tor16b] is applicable to comparison of only unsigned integers. The proposed algorithm is designed for a five-moduli set composed of equally arithmetic friendly moduli but with 40% increase in parallelism. One major advantage of the proposed design is its ability to perform comparison of signed integers in residue domain by using only mod 2^n and mod $2^{n+1}-1$ adders. The bit lengths of these adders are only around one-fifth of the dynamic range, i.e., $\sim n$ bits as opposed to $\sim 5n$ bits.

Table 5.2: Evaluation of RNS Magnitude Comparison Algorithms

Method	Moduli Set	Approach	RM?	SC?
[Mil85]	Arbitrary odd moduli	Parity check	Yes	No
[Lu92]	Arbitrary odd moduli	Parity check	Yes	Yes
[Sou07]	$\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$	Parity check	No	No
[Zar10]	$\{2^n-1, 2^n+1, 2^n+3\}$	Parity check	No	No
[Dim93]	Arbitrary moduli	Partial reverse conversion	No	No
[Wan99]	Arbitrary moduli	Partial reverse conversion	No	No
[Tor16a]	$\{2^n-1, 2^n, 2^n+1\}$	Partial reverse conversion	No	No
[Bi08]	$\{2^n-1, 2^n, 2^n+1\}$	Partial reverse conversion	No	No
[Tor16b]	$\{2^n-1, 2^n, 2^n+1, m\}$ $m \in \{2^{n+1}+1, 2^{n+1}-1, 2^{n-1}-1\}$	Partial reverse conversion	No	No
This	$\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$	Sign detection based	No	Yes

5.4.2 Synthesis Results

To the best of my knowledge, no RNS comparator has been proposed for moduli sets with five or more modular arithmetic friendly moduli of the forms 2^k and $2^k \pm 1$ with $k \in \mathbb{Z}^+$. In the absence of compatible RNS signed integer comparator for $S = \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, the CRT-II based technique [Wan99] for the magnitude comparator of unsigned integers represented in arbitrary moduli sets is adapted to realize an efficient adder-based RNS comparator for S . For a fair comparison, its implementation is extensively optimized by the number theoretic properties associated with the special moduli of S whenever applicable. As the original algorithms [Wan99] do not support the comparison of signed integers, the proposed sign detector (consisting of A_x and B_x generators, X_s computation and sign detector in Figure 5.3) is also incorporated into the devised CRT-II based magnitude comparator to parallelly detect the polarity of input operands. Another well known approach to compare the residue representations of two signed integers is to convert them into binary domain using the most efficient reverse converter for S [Cao07] followed by the binary comparators. Unfortunately, no RNS sign detector has been reported for this five-moduli set. To corroborate the efficiency of the proposed scaling assisted sign detector, it will be compared against the only alternative way to detect the sign of a residue representation in S by its best-known residue-to-binary converter [Cao07] followed by a $5n$ -bit binary comparator. All the abovementioned designs are described in Verilog HDL and functionally verified by Mentor Graphics ModelSim. Each design is implemented for five different dynamic ranges of S by varying n from 4 to 20 in step of 4. Each design is synthesized for minimum delay by Synopsys Design Compiler version E-2010.12-SP4 using STM CMOS 65 nm technology standard cell library.

Table 5.3 shows the synthesized area (A) in μm^2 and critical path delay (D) in ns of the proposed scaling-assisted and the reverse-converter based sign detectors. The results show that the proposed sign detector is 26.5% more area-efficient and 26.4% faster than the sign detector designed based on the best reverse converter for S [Cao07].

Table 5.3: Comparison of areas (μm^2) and critical path delays (ns) of RNS sign detectors for $S = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$.

n	Reverse converter based sign detector		This		Percentage reduction	
	$D (ns)$	$A (\mu m^2)$	$D (ns)$	$A (\mu m^2)$	$\Delta D (\%)$	$\Delta A (\%)$
4	2.91	4815.20	2.06	3768.96	29.21	21.73
8	4.24	12207.52	2.92	9548.76	31.13	21.78
12	4.84	20436.00	3.70	15450.76	23.55	24.39
16	5.37	24945.44	4.19	17307.6	21.97	30.62
20	6.10	30958.20	4.51	20385.04	26.07	34.15

Table 5.4 shows the synthesis results for the proposed comparator with the magnitude comparator based on CRT-II [Wan99] (denoted as unsigned CRT-II), signed integer comparator based on CRT-II with proposed sign detector (denoted as signed CRT-II) and signed integer comparator based on reverse converter [Cao07] succeeded by binary comparator (denoted as Hybrid). On average, the proposed comparator is 36.9% smaller and 7.6 % faster than unsigned CRT-II, 50.2% smaller and 9.4% faster than signed CRT-II, and 12.9% smaller and 7.3% faster than Hybrid. As indicated in Figure 5.3, the critical path delay of the proposed design is dominated by the scaling-assisted sign detector for the difference $\tilde{Z}$. Even though this delay is slightly larger than the sign detector for the input operand $\tilde{X}$, it is still shorter than the delay of Hybrid and unsigned CRT-II for all n . In fact, because of the significantly faster scaling-assisted sign detector, as evinced by the results of Table 5.3, the polarities of the input operands are detected well before the magnitude comparator of CRT-II. Thus the small delay difference between unsigned and signed CRT-II is due to the extra polarity multiplexing logic for the relational inference. Overall, the area savings of the proposed comparator are notably higher than its delay savings, especially for large n .

Table 5.4: Comparison of Synthesized Area (μm^2) and Delay (ns) of the Proposed Design and Alternative signed and unsigned integer comparators for $S = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$.

method		n				
		4	8	12	16	20
This	A	10177.96	21981.44	37295.96	46883.72	54984.28
	D	2.87	3.92	4.70	5.19	5.99
Unsigned CRT-II	A	14894.88 (31.67%)	26739.44 (17.79%)	66669.20 (44.06%)	92182.48 (49.14%)	94484 (41.81%)
	D	3.19 (10.03%)	4.02 (2.49%)	4.97 (5.43%)	6.11 (15.06%)	6.31 (5.07%)
Signed CRT-II	A	17998.76 (43.45%)	37814.40 (41.87%)	82047.16 (54.54%)	106461.16 (55.96%)	123174.48 (55.36%)
	D	3.26 (11.96%)	4.02 (2.49%)	5.12 (8.20%)	6.30 (17.62%)	6.41 (6.55%)
Hybrid	A	10581.48 (3.81%)	26096.72 (15.77%)	38764.96 (3.79%)	54699.84 (14.29%)	74909.12 (26.60%)
	D	3.13 (8.31%)	4.39 (10.71%)	5.12 (8.20%)	5.51 (5.81%)	6.20 (3.39%)

Power consumption is also compared by running Monte Carlo power simulation [Bur93] on each design using Synopsys Prime Time PX, where randomly generated input vectors are applied until the error of estimated average power is bounded below 2.5% with 99% confidence interval. Since the critical path delays of all designs in comparison are different and vary with the dynamic range, to ensure that all designs will produce the correct outputs at the same data rate, the clock rate used for the input data is chosen based on the slowest design for each n . The total power consumption in mW and the leakage power in μW are compared in Table 5.5 for the proposed scaling assisted and reverse converter based sign detectors, and in Table 5.6 for all comparators in comparison. From Table 5.5, the proposed sign detector reduces the total power consumption of the reverse converter based sign detector by 31.5% on average. From Table 5.6, on average, the proposed comparator saves 45.4%, 52.7% and 20.7% of total power consumption over unsigned CRT-II, signed CRT-II and Hybrid, respectively. It is noted from Tables 5.3 and 5.5 that the area, delay and power consumption of the proposed sign detector increase only gently with n . Thanks to the sparsity of CSD representation for k_3 , the growth in the number of product terms with n due to this multiplicative inverse has slowed down tremendously. The size of the modular adders have also been reduced to about one-fifth from $5n$ bits to n or $n+1$ bits, leading to the substantial area and power savings for all n . The differences in the margin of reduction of the proposed design over other comparators can be

ascribed to the relative ability of their base algorithms ([Cao07] for Hybrid and [Wan99] for CRT-II based comparators) to harness the number theoretic properties unique to S .

Table 5.5: Comparison of leakage power (LP) in μW and total power (TP) in mW of RNS sign detectors for $S = \{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$.

n	Reverse converter based sign detector		This		Percentage reduction	
	$LP (\mu W)$	$TP (mW)$	$LP (\mu W)$	$TP (mW)$	$\Delta LP (\%)$	$\Delta TP (\%)$
4	303.69	6.66	235.44	4.73	22.47	28.98
8	748.71	18.40	606.96	12.35	18.93	32.88
12	1257.54	29.58	958.30	23.06	23.80	22.04
16	1489.94	36.21	979.11	24.04	34.29	33.61
20	1745.32	41.52	1119.21	24.95	35.87	39.91

Table 5.6: Comparison of Leakage power (LP) in μW and total power (TP) in mW of the Proposed Design and Alternative signed and unsigned integer comparators for S .

method		n				
		4	8	12	16	20
This	LP	568.13	1192.37	2105.20	2314	2422.09
	TP	13.22	29.22	56.32	48.63	56.28
Unsigned CRT-II	LP	983.20 (42.22%)	1653.10 (27.87%)	4263.40 (50.62%)	5540.61 (58.24%)	5389.35 (55.06%)
	TP	20.66 (36.01%)	39.35 (25.74%)	101.58 (44.56%)	128.27 (62.09%)	135.91 (58.59%)
Signed CRT-II	LP	1055.04 (46.15%)	2170.69 (45.07%)	4941.95 (57.40%)	5765.16 (59.86%)	6471.58 (62.57%)
	TP	22.32 (40.77%)	48.27 (39.47%)	120.29 (53.18%)	142.21 (65.80%)	157.73 (64.32%)
Hybrid	LP	663.40 (14.36%)	1612.11 (26.04%)	23463.32 (10.28%)	3318.64 (30.27%)	4523.25 (46.45%)
	TP	13.79 (4.13%)	37.12 (21.28%)	57.28 (1.68%)	69.26 (29.79%)	105.70 (46.75%)

Faster design generally consumes more dynamic power. To compare the energy consumption of each design running at full speed, power-delay product is computed for each design with the power consumption measured at its maximum allowable clock frequency. The percentage reduction of the energy consumption by the proposed design over CRT-II and Hybrid are plotted in Figure 5.9 for different n . Noteworthy double-digit percentage savings are achieved except for the percentage savings over Hybrid for $n = 4$ and 12, which are only 4.3% and 1.2%, respectively. For these two cases, the proposed design is only slightly faster and modestly smaller than Hybrid.

Figure 5.9: Percentage reduction in energy consumption of the proposed design over Unsigned and Signed CRT-II and Hybrid for S with $n = 4$ to 20.

5.5 Summary

A signed integer comparator for the balanced five-moduli set $\{2^n-1, 2^n, 2^{n+1}, 2^{n+1}-1, 2^{n-1}-1\}$ with even values of n has been proposed for the first time. With carefully selected composite moduli as scaling factor, scaling is performed efficiently to ease the sign detection of input operands and their difference, which in turn simplifies their relational inference. The bottleneck sign detector of scaled integer is overcome by factoring the large composite modulo sum into two simple modulo sums in the 2^n and $2^{n+1}-1$ modulus channels. The number of terms in the linear combination of sum and carry vectors are further reduced by representing the multiplicative inverse in CSD. The proposed architecture can be deemed as an ingenious unification of three difficult RNS operations, i.e., scaling, sign detection and magnitude comparison. As all arithmetic units involved are reduced to simple modular adders of existing moduli, the resources can be shared with regular arithmetic operations besides using for scaling, sign detection and comparison.

Chapter 6

Conclusion and Future Works

6.1 Conclusions

This research aims to develop efficient algorithms for efficient hardware architectures of sign detection and signed integer magnitude comparator without resorting to full reverse conversion to binary domain. The ambiguity in magnitude and range of the residue representation make these operations difficult to be implemented efficiently. Unfortunately, the complexity of these operations further grows with the increasing cardinality of the moduli set. Thus, the inefficiency of these operations has become the main hurdle to full-fledged implementation of general-purpose processor in RNS even though the use of special moduli sets with good number theoretic properties have significantly improved the basic arithmetic operations such as addition and multiplication in RNS. The three new proposals made in Chapters 3 to 5 of this thesis, to a great extent, have overcome these difficult sign detection and magnitude comparison problems for special moduli sets with three to five balanced special moduli of the form 2^n , $2^n \pm 1$ where n is a positive integer.

In Chapter 3, two new algorithms for sign detection in the celebrated three-moduli set $\{2^n-1, 2^n, 2^n+1\}$ have been presented. The first algorithm is leverage on the new formulation of CRT by scaling the residues by a factor of 2^n to reduce the size of modulo adder involved in the process of sign detection. Compared with sign detection friendly reverse converter, the resulting architecture due to this technique is found to be smaller and faster. The complexity of sign detection is further reduced by the second algorithm which is formulated based on CRT-II and scaling with a different factor of $2^{2n}-1$. The selected scaling factor not only reduces the size of modulo operator from $3n$ bits to n -bits, but also aligns the dynamic range of the scaled residue representation to that of an n -bit binary representation. Hence, the sign can be detected easily from the MSB of its scaled residue without reverse conversion. Compared with existing sign

detectors for the same moduli set, notable improvements in terms of the delay, area and power consumption have been demonstrated by the logic synthesis experiments.

Since sign detection and magnitude comparison are both difficult operations in RNS, existing algorithms mainly tackle either problem independently. As a consequence, comparisons of signed integers in residue domain are usually supplemented with a separately designed RNS sign detector. This adds significant overheads to the already complex problem of both operations in higher cardinality moduli set. Herein, a new technique to design signed magnitude comparator for two four-moduli supersets, $S_1 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$, for even values of n and $k \in [0, n]$, has been proposed in Chapter 4. The dynamic range is divided into equal number of sub-divisions. The rank of subdivision instead of the exact magnitude is identified from the operands in comparison and compared with their residue representations. The sign of an operand can be easily detected from the rank of the subdivision where its residue representation lies without the need of a separate sign detector module. Compared with best-possible existing signed and unsigned integer magnitude comparators for the same moduli sets, the proposed signed integer magnitude comparator outperforms them in terms of the area, delay and energy efficiency.

Last but not least, a scaling-assisted signed integer comparator has been proposed for the balanced five moduli set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ with the even values of n . The algorithm presented in Chapter 5 solves the hardest sign magnitude comparison problem for the highest cardinality special moduli set ever attempted in the literature. It is developed with the help of two-moduli CRT-II, CRT and an appropriate scaling factor. Scaling is first performed to reduce the word-length of operators involved in the sign detection of the operands and their difference. The latter helps to simplify the final relation inference module substantially but its simplification is contingent on the ease of sign detection made possible by scaling. The complexity of the scaled integer is further reduced by decomposing the large modulo sum into two small modulo sums in the 2^n and $2^{n+1}-1$ modulus channels and representing the multiplicative inverses in CSD form. Compared with the existing unsigned and signed integer

comparators, the proposed circuit is faster, smaller and less power-hungry as proven by the logic synthesis results.

6.2 Future Works

Based on the accomplished research works presented in this thesis and the extensive survey on related works on this subject, the following relevant topics are identified for future exploration.

- a) The nascent concept of exploiting base transformation for dynamic range reduction opens out new avenues of research in the interoperability of multi-base RNS [Tay15]. In [Tay15], an elegant algorithm for remapping the residue digits of a signed integer from the moduli set $\{2^n-1, 2^{n+k}, 2^n+1\}$ to another set $\{2^{n-a}-1, 2^{n+k-a}, 2^{n-a}+1\}$ of lower dynamic range was proposed, where $k \in [0, n]$ and $a \in [1, n-2]$. The reverse mapping of residues from moduli set S_1 to other moduli set S_2 with larger dynamic range are equally important in certain applications which require temporary expansion of dynamic range to avoid overflow. A preliminary work has been done to develop a new algorithm using the New CRT-II to map the residue representation of a signed integer from $\{2^{n-a}-1, 2^{n+k-a}, 2^{n-a}+1\}$ to $\{2^n-1, 2^{n+k}, 2^n+1\}$. Comparing with the best possible improvised solution, the preliminary results showed that this method produces design that is faster but consumes more area. It is believed that the method can be further refined and improved to reduce the area and power consumption so as to complete the full interoperability between S_1 and S_2 without resorting to any residue-to-binary conversion.
- b) Inter-modulo operations are not limited to reverse conversion and the sign detection, scaling and magnitude comparison considered in this thesis. These inefficient inter-modulo operations limit the usefulness of RNS-based computation to specific digital signal processing applications. Full-fledge or full-scale implementation usually involves a hybrid binary-RNS implementation, which explains why the humongous amount of new reverse converter designs in the literature. However, hybrid RNS design is not an elegant and effective means of advancing RNS applications as some operations such as division are

equally difficult in binary domain as in RNS. Moreover, the forward and reverse conversion between two distinctively different domains reduces the opportunity to fuse and reuse resources common to heterogenous arithmetic operations within the same domain. In fact, this thesis has for the first time demonstrated the possibility of using one intermodulo operation in RNS to simplify another intermodulo operation in the same moduli set. Scaling in RNS has been shown to complement sign detection, which in turn helps to simplify and accelerate sign-magnitude comparison if these operations are all performed within the residue domain. The hardware resources in heterogenous inter-modulo computations beyond scaling, sign detection and magnitude comparison such as leading and trailing zero detection, over- and underflow detection, majority voting, etc. can be fused or time-shared extensively in an application to significantly lower the area, delay and power consumption and facilitates voltage-frequency scaling. Efficient unified architectures for heterogenous residue operations for special moduli sets as well as for general moduli sets are promising game changers to widen RNS applications.

- c) Quadratic residue number systems (QRNS) possesses similar advantages as standard RNS. The choice of moduli is of prime importance to QRNS as it is to RNS. Many useful DSP problems are based fundamentally on complex mathematics such as fast Fourier transform and z-transform for representation of periodical signals. The imaginary parts of the complex variables in these transforms are often ignored in order to simplify computations but there are situations where the complete magnitude and phase information of the complex variables need to be preserved. QRNS is widely known for its efficient implementation of complex number multiplications as it has eliminated the need of additional hardware resources to manage the cross product terms. However, the form of moduli in QRNS is highly restrictive to fulfil the algebraic structure of complex number. The moduli must be in the form of $4k+1$ in order to compute the quadratic residue for unit imaginary number. However, moduli in the form of $4k+1$ are highly imbalanced, which results in high variation in the computational speed among residue channels. One possible solution is to decompose the big modulus into a number of smaller moduli to approximately equalize the computational speed of all moduli channels. Simultaneously, the balancing of the moduli

should also meet the fundamental criterion of quadratic residue. Besides, inter-modulo operations such as sign detection, reverse conversion, magnitude comparison, etc. are all very new problems that have not been explored well in QRNS. Given the increasing number of new applications that demand complex transforms in wireless communications and video signal processing, it is believed that QRNS may offer more competitive advantages than standard RNS in their hardware implementation.

Author's Publications

Journal Papers

[Kum16a] **S. Kumar** and C. H. Chang, “A new fast and area-efficient adder-based sign detector for RNS $\{2^n-1, 2^n, 2^n+1\}$,” *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.*, vol. 24, no. 7, pp. 2608-2612, Jul. 2016.

[Kum16b] **S. Kumar**, C. H. Chang and T. F. Tay, “New algorithm for signed residue comparison in four-moduli supersets and its efficient VLSI implementations,” *IEEE Trans. on Cir. Syst. I, Regular Papers*, pp. 1-13, Jul. 2016.

[Kum16c] **S. Kumar** and C. H. Chang, “Scaling-assisted sign integer comparator for the balanced five-moduli set RNS $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$,” submitted to *IEEE Transactions on VLSI*.

Conference Papers

[Cha14] C.H. Chang and **S. Kumar**, “Area-efficient and fast sign detection for four-moduli set RNS $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$,” in *Proc. 2014 Int. Symp. on Circuits and Syst. (ISCAS)*, Melbourne, Australia, June 2014, pp. 1540-1543.

[Kum15] **S. Kumar** and C. H. Chang, “A high-speed and area-efficient sign detector for three moduli set RNS $\{2^n, 2^n-1, 2^n+1\}$,” in *Proc. 2015 IEEE 11th Int. Conf. ASIC (ASICON)*, Chengdu, China, Nov. 2015, pp. 1-4.

[Kum16d] **S. Kumar** and C. H. Chang, “A VLSI-efficient signed magnitude comparator for $\{2^n-1, 2^n, 2^{n+1}-1\}$,” in *Proc. 2016 Int. Symp. on Circuits and Syst. (ISCAS16)*, Montreal, Canada, June 2016, pp. 1966-1969.

References

- [Aka11] Y. Akasaka *et al.* (2011). *Process integration, devices, and structures* [Online]. Available: <http://www.itrs.net/Links/2011ITRS/Home2011.htm>
- [Amo07] R. O. Amos and P. Benjamin, *Residue Number Systems: Theory and Implementation*. London: Imperial College Press 2007.
- [Ali98] G. Alia and E. Martinelli, "Sign detection in residue arithmetic units," *Jour. of sys. Arch.*, Vol.45, pp. 251-258, Nov. 1998.
- [Akk08] M.Akkal, and P.Siy, "Optimum RNS sign detection algorithm using MRC-II with special moduli set," *Jour. Syst. Arch.*, vol. 54, pp. 911-918, Oct. 2008
- [Akk07] M. Akkal, and P.Siy, "A new mixed radix conversion algorithm MRC-II," *Jour. of sys. Archi*, Vol. 53, pp. 577-586, Sept. 2007.
- [Bai07] S. Bailey, G. Chidolue and A. Crone, "Low power design and verification techniques," *Mentor graphics, white paper*, 2007. [Online]. Available: http://low-powerdesign.com/Low_Power_WP_9-13-07.pdf
- [Bro97] H. Bronnimann *et al.*, "Computing exact geometric predicates using modular arithmetic with single precision," in *Proc. 13th Annu. Symp. Geom.*, Nice, France, Jun. 1997, pp. 174-182.
- [Ber07] G. L. Bernocchi, G. C. Cardarilli, A. Del Re, A. Nannarelli and M. Re, "Low power adaptive filter based on RNS components," in *Proc. 2007 IEEE Int. Symp. on Cir. and Syst.*, New Orlean, USA, May 2007, pp. 3211-3214.
- [Baj04] J.C. Bajard and L.Imbert, "A full RNS implementation of RSA," *IEEE Trans. on Comp.*, Vol. 53, p.p. 769-774, Jun. 2004.

- [Bi08] S. Bi and W. J. Gross, "The mixed-radix Chinese remainder theorem and its applications to residue comparison," *IEEE Trans. Comput.*, vol.57, no. 12, pp. 1624-1632, Dec. 2008.
- [Bur93] R. Burch, F. N. Najm, P. Yang, and T. N. Trick, "A Monte Carlo approach for power estimation," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 63-71, Mar. 1993.
- [Baj11] J. C. Bajard, L. S. Didier, and T. Hilair, " ρ -direct form transposed and residue number systems for filter implementations," in *Proc. of IEEE Midwest Symp. on Cir. and Syst.*, Seoul, Korea, Aug. 2011, pp. 1-4.
- [Con04] R. Conway and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Trans. Cir. Syst. -II*, vol. 51, no.1, 26-28, Jan. 2004.
- [Car05] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Low power and low leakage implementation of RNS FIR filters," in *Thirty-Ninth Asilo. Conf. on Sig. Syst. and Comp.*, Pacific Grove, California, USA, Nov. 2005, pp. 1620-1624.
- [Cao03] B. Cao, C. H. Chang and T. Srikanthan, "An efficient reverse converter for the 4-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{2n}-1\}$ based on the new Chinese Remainder Theorem," *IEEE Trans. Cir. Syst. I*, Vol. 50, no. 7, pp. 1296-1303, Jul. 2003.
- [Con99] R. Conway and J. Nelson, "Fast converter for three moduli RNS using new property of CRT," *IEEE Trans. on Comp.*, vol. 48, pp. 852-860, Aug. 1999.
- [Che03] S. W. Cheng, "A high-speed magnitude comparator with small transistor count," in *Proc. 2003 IEEE Int. Conf. Electron. Circuits Syst.*, Sharjah, UAE, Dec. 2003, pp. 1168-1171.
- [Cha15] C. H. Chang, Amir, A.S. Molahosseini, A.A.E. Zarandi and T. F. Tay, "Residue Number Systems: a new paradigm to datapath optimization for low-power and high-

performance digital signal processing applications,” *IEEE Circuits and Systems Magazine*, vol. 15, no. 4, pp. 26-44, Nov. 2015.

[Cha11] C. H. Chang and J. Y. S. Low, “Simple, fast and exact RNS scaler for three moduli $\{2^n-1, 2^n, 2^n+1\}$ set,” *IEEE Trans. Cir. Syst.-I*, vol.58, no.11, pp. 2686-2697, Nov. 2011.

[Cao07] B. Cao, C. H. Chang, and T. Srikanthan, “A residue-to-binary converter for a new five-moduli set,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 5, pp. 1041–1049, May 2007.

[Cha04] R. Chaves and L. Sousa, “ $\{2^n+1, 2^{n+k}, 2^n-1\}$: A new RNS moduli set extension,” in *Proc. 2004 EUROMICRO Syst. Digit. Syst. Des. (DSD'04)*, Rennes, France, Aug. 2004, pp. 210-217.

[Cho09] R Chokshi, K. S. Berezowski, A. Shrivastava, and S. J. Piestrak, “Exploiting residue number system for power-efficient digital signal processing in embedded processors,” in *Proc. 2009 ACM Int. Conf. on Compilers, Architecture, and Synthesis of Embedded Syst.*, (CASES '09), New York, USA, Oct. 2009, pp. 19-28.

[Cao10] B. Cao, Y. S. Low, C. H. Chang and T. Srikanthan, “Performance analysis of different special moduli sets for RNS-based inner product step processor,” in *Proc. 2010 Int. Conf. on Green Circuits and Syst. (ICGCS-2010)*, Shanghai, China, Jun. 2010, pp. 236-241.

[Che09] J. Chen and J. Stine, “Parallel prefix Ling structures for modulo 2^n-1 addition,” in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Arch., & Processors*, Boston, MA, USA, Jul. 2009, pp. 16-23.

[Dim93] G. Dimauro, S. Impedovo, and G. Pirlo, “A new technique for fast number comparison in Residue Number System,” *IEEE Trans. Comput.*, vol. 42, No. 5, pp. 608-612, May 1993.

[Did13] L. S. Didier and L. Jaulmes, "Fast modulo 2^n-1 and 2^n+1 adder using carry-chain on FPGA," in *Proc. 2013 Asilom. Conf. Signals, Syst. & Comput.*, Pacific Grove, California, USA, Nov. 2013, pp. 1155-1159.

[Efs04] C. Efstathiou, H. T. Vergos and D. Nikolos, "Modified booth modulo 2^n-1 multipliers," *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 370-374, Mar. 2004.

[Gar 00] P. Gargini, "The International Technology Roadmap for Semiconductors (ITRS): Past, present and future," in *Proc. 2000 IEEE Symp. 22nd Annual Gall. Ars. Integ. Cir. (GaAs IC)*, Seattle, Washington, Nov. 2000, pp. 3-5.

[Gri98] M. Griffin, F. Taylor, and M. Sousa, "New scaling algorithms for the Chinese remainder theorem," in *Twenty-Second Asilomar Conf. on Sig., Syst. and Comp.*, Pacific Grove, CA, Oct.1998, pp. 375-378.

[Gar59] H. L. Garner, "The residue number system, " *IRE Trans. Electronic Computers*, vol. EC-8, pp. 140-147, June 1959.

[Har08] A. Hariri, K. Navi, and R. Rastegar, "A new high dynamic range set with efficient reverse converter," *Jour. Comp. Math. Appl.*, vol. 55, pp. 660-668, Feb. 2008.

[Hen99] H. Henkelmann and W. Anheier, "Implementation of sign detection in RNS using mixed radix representation," in *Proc. Int. Conf. Electron., Circuits and Syst., IEEE*, Vol. 1, Paphos, Cyprus, Sep. 1999, pp. 323-326.

[Hew00] R. M. Hewlitt and E. S. Swartzlantler, "Canonical signed digit representation for FIR digital filters," in *Proc. IEEE Workshop Signalb Process. Syst.*, Lafayette, LA ,USA, Oct. 2000, pp. 416-426.

- [Kal00] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos and J. Kalamatianos, "High-speed parallel prefix adder module 2^n-1 adder," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 673-680, Jul.2000.
- [Liu04] Y. Liu and E M. K. Lai, "Design and implmentaiton of an RNS-based 2-D DWT processor," *IEEE Trans. Consumer Electronics*, vol. 50, no. 1, pp.376-385, Feb. 2004.
- [Low13] J. Y. S. Low, T. F. Tay, and C. H. Chang, "A signed integer programmable power-of-two scaler for $\{2^n-1, 2^n, 2^n+1\}$ RNS," in *2013 Int. Symp. on Circuits and Syst.*, Beijing, China, May 2013, pp. 2211-2214.
- [Lin84] M.L.Lin, E. Leiss, and B. McInnis, "Division and sign detection algorithms for residue number systems," in *Comp. & Maths. with Appls.*, Vol.10, pp.331-342,1984.
- [Lu92] M. Lu and J. Chiang, "A novel division algorithm for the residue number system," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 1026-1032, Aug. 1992.
- [Moh12] S.Moharrami and D.K.Taleshmekaeil, "The application of the residue number system in digital image processing: propose a scheme of filtering in spatial domain," *Research jour. of appl. Sci.*, Vol.7, pp.286-292, 2012.
- [Mel96] J. D. Mellott, M. Lewis, F. Taylor, and P. Coffield, "ASAP-A 2-D DFT VLSI processor and architecture," in *Proc. 1996 IEEE Int. Symp. Cir. and Syst.*, Atlanta, USA, May 1996, pp. 261-264.
- [Moh07] P. V. A. Mohan and A. B. Premkumar, "RNS-to-binary converter for two four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$," *IEEE Trans. on Cir. and Syst. I*, Vol. 54, no. 6, pp. 1245-1254, Jun. 2007.

[Mol10] A. S. Molahosseini et al., “Efficient reverse converter designs for the new 4-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ and $\{2^n-1, 2^{2n}, 2^n+1, 2^{2n}-1\}$ based on New CRTs,” *IEEE Trans. Cir. Syst. I*, vol. 57, no. 4, pp. 823-835, Apr. 2010.

[Mur12] R. Muralidharan and C. H. Chang, “Area-power efficient modulo 2^n-1 and modulo 2^n+1 multipliers for $\{2^n-1, 2^n, 2^n+1\}$ based RNS,” *IEEE Trans. Cir. Syst.-I*, vol. 59, no. 10, pp. 2263-2274, Oct. 2012.

[Ma10] S. Ma, J. Hu, Y. Ye, L. Zhang, and X. Ling, “A 2^n scaling scheme for signed RNS integers and its VLSI implementation,” *Sci. China, Inf. Sci.*, Vol. 53, pp. 203-212, Jan. 2010.

[Mol08] A. S. Molahosseini and K. Navi, “An efficient architecture for designing reverse converters based on a general three-moduli set,” *Jour. Sys. Arch.*, vol. 54, pp. 929-934, Oct. 2008.

[Mat08] P. M. Matutino and L. Sousa, “An RNS based specific processor for computing the minimum sum-of-absolute-differences,” in *Proc. 2008 11th Euromicro Conf. on Digital Syst. Design (DSD)*, Parma, Italy, Sept. 2008, pp. 768-775.

[Mil85] D. D. Miller, R. E. Altschul, J. R. King and J. N. Polky, “Analysis of the residue class core function of Akushskii, Burcev and Pak,” in *Residue Number System Arithmetic, Modern Applications in Digital Signal Processing*, M. A. Soderstrand, W. C. Jenkins, G. A. Jullien, and F. J. Taylor, eds. New York, IEEE Press, paper 7-2, pp. 390-401, 1985.

[Nir15] C. V. Niras, and Y. Kong, “Fast sign detection algorithm for the RNS moduli Set $\{2^{n+1}-1, 2^n-1, 2^n\}$,” *J. IET comput. and digital Tech.*, vol. 10, pp. 54-58, Mar. 2016.

[Ok194] V. G. Oklobdzija, “An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 1, pp. 124-128, Mar. 1994.

[Pat14] P. Patronik and S.J. Piestrak, "Design of Reverse Converters for General RNS Moduli Sets $\{2^k, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $\{2^k, 2^n-1, 2^n+1, 2^{n+1}+1\}$ (n even)," *IEEE Trans. Cir. Syst. I*, vol. 61, no. 6, pp. 1687-1700, Jun. 2014.

[Pet13a] H. Pettenghi, R. Chaves, and L. Sousa, "Method to design general rns reverse converters for extended moduli sets," *IEEE Trans. on Cir. and Sys. II*, Vol. 60, pp. 877-881, Dec. 2013.

[Pet13b] H. Pettenghi, R. Chaves and L. Sousa, "RNS reverse converters for moduli sets with dynamic range upto $(8n+1)$ -bit," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 6, pp. 1487-1500, Jun. 2013.

[Par09] B. Parhami, "Efficient hamming weight comparators for binary vectors based on accumulative and up/down parallel counters," *IEEE Trans. Circuits Syst.*, vol. 56, no. 2, pp. 167-171, Feb. 2009.

[Pie94] S. Piestrak, "Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders" *IEEE Trans. Comput.*, vol. 423, no. 1, pp. 68-77, Jan. 1994.

[Pon04] D. V. Ponomarev, G. Kucuk, O. Ergin, and K. Ghose, "Energy efficient comparators for superscalar datapaths," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 892-904, Jul. 2004.

[Ram02] J. Ramirez, A. Garcia, U. Meyer-Baese, and A. Lloris, "Fast RNS-FPL based communications receiver design and implementation," in *Proc. Int. Conf. Field Prog. Logic*, vol. 2438, Montpellier, France, Sept. 2002, pp. 472-481.

[Sto01] T. Stouraitis and V. Paliouras, "Considering the alternatives in low-power design," *IEEE Cir. And Device Mag.*, Vol. 17, pp. 22-29, Jul. 2001.

- [Sod86] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number Arithmetic: Modern Applications in Digital Signal Processing*. Piscataway, NY: IEEE Press, 1986.
- [Sas68] A. Sasaki, "The basis for implementation of additive operation in the residue number system," *IEEE Trans. Comp.*, vol. C-17, pp.1066-1073, Nov. 1968.
- [Ska98] A. Skavantzios, "An efficient residue to weighted converter for a new residue number system," in *Proc. 8th Great Lakes Symp. On VLSI*, Lafayette, Louisiana, Feb. 1998 pp. 185-191.
- [She08] Y. Sheng and W. Wang, "Design and implementation of compression algorithm comparator for digital image processing on component," in *Proc. 9th Int. Conf. Young Comput. Sci.*, Hunan, China, Nov. 2008, pp. 1337-1341.
- [Sch09] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A.P. Kakarountas, and T. Stouraitis, "An RNS implementation of an elliptic curve point multiplier," *IEEE Trans. on Cir. and Sys.*, Vol.56, p.p. 1202-1213 , Jun. 2009.
- [Suz07] H. Suzuki, C. H. Kim, and K. Roy, "Fast tag comparator using diode partitioned domino for 64-bit microprocessor," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 2, pp. 322–328, Feb. 2007.
- [Sch14] D. Schinianakis and T. Stouraitis, "Multifunction residue architectures for cryptography," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1156-1169, Apr. 2014.
- [Sza67] N. S. Szabo and R. I. Tanka, *Residue Arithmetic and Its Applications to Computer Technology*. New York: Mc Graw-Hill, 1967.

- [Sha08] B. Shaoqiang and W. J. Gross, "The mixed-radix Chinese Remainder Theorem and its applications to residue comparison," *IEEE Trans. on Comp.*, Vol. 57, pp. 1624-1632, Dec. 2008.
- [She89] A.P.Shenoy and R.Kumaresan, "Fast base extension using a redundant modulus in RNS," *IEEE Trans. on Comp.*, Vol.38, pp.292-297, Feb.1989.
- [Set05] E. Setiaarif and P. Siy, "A new moduli set selection technique to improve sign detection and number comparison in residue number system (RNS)," in *Annual Meeting of the North American Fuz. Inf. Process. Soc.*, NAFIPS 2005, Detroit, MI, USA, Jun. 2005, pp. 766-768.
- [Sou07] L. Sousa, "Efficient method for magnitude Comparison in RNS based on two pairs of conjugate moduli," in *Proc.2007 18th IEEE Symp. Comput. Arithmetic*, Montpellier, France, Jun. 2007, pp. 240-250.
- [Ska99] A. Skavantzios and T. Stouraitis, "Grouped-moduli residue number systems for fast signal processing," in *Proc. 1999 Int. Symp. Circuits Syst.*, Orlando, FL, vol. 3, Jun. 1999, pp. 478-483.
- [Sou14] L.Sousa and P. Martins, "Efficient sign identification engines for integers represented in RNS extended 3-moduli set $\{2^n-1, 2^{n+k}, 2^n+1\}$," *Electronics Letters*, vol.50, no.16, pp.1138-1139, Jul. 2014.
- [Tay84] F. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Trans. Comp.*, vol. C-32, pp. 50-62, May 1984.
- [Tay90] F. J. Taylor, "An RNS discrete Fourier transform implementation," *IEEE Trans. Acoust., Speech, Sig.I Process.*, Vol. 38, no. 8, pp.1386-1394, Aug. 1990.
- [Tya93] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. Comp.*, vol. 42, no. 10, pp. 1163-1170, Oct. 1993.

[Tay15] T. F. Tay, C. H. Chang, and L. Sousa, "Base transformation with injective residue mapping for dynamic range reduction in RNS," *IEEE Trans. Circuits and Syst. I*, vol. 62, no. 9, pp. 2248-2259, Sept. 2015.

[Tom08] Tadeusz Tomczak, "Fast sign detection for RNS $\{2^n-1, 2^n, 2^n+1\}$," *IEEE Trans. Cir. Syst. I*, vol. 55, no. 6, pp. 1502-1511, Jul. 2008.

[Tor16a] Z. Torabi and G. Jaberipur, "Low-power/cost RNS comparison via partitioning the dynamic range," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 5, pp. 1849-1857, Apr. 2016.

[Tor16b] Z. Torabi and G. Jaberipur, "Fast low energy RNS comparators for 4-moduli sets $\{2^n \pm 1, 2^n, m\}$ with $m \in [2^{n+1} \pm 1, 2^{n-1} - 1]$," *Integ. The VLSI journal*, vol. 55, pp. 155-161, Sept. 2016.

[Tay14] T. F. Tay and C. H. Chang, "New algorithm for signed integer comparison in four-moduli superset $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$," in *2014 Asia Pacific Conf. Circuits and Syst.*, Okinawa, Japan, Nov. 2014, pp. 519-522.

[Tay13] T. F. Tay, C. H. Chang, and J. Y. S. Low, "Efficient VLSI implementation of $2n$ scaling of signed integer in $\{2^n, 2^n-1, 2^n+1\}$," *IEEE Trans. VLSI Syst.*, vol. 21, no. 10, pp. 1936-1940, Oct. 2013.

[Ulm83] Z. D. Ulman, "Sign detection and implicit-explicit conversion of numbers in residue arithmetic," *IEEE Trans. Comp.*, Vol. C-32, no.6, pp. 590-594, Jun. 1983

[Vas13] E. Vassalos, D. Bakalis, and H.T. Vergos, "RNS assisted image filtering and edge detection," in *Proc. 2013 IEEE Int. Symp. on Digital Signal Proces.*, Santorini, Greece, July 2013, pp. 1-6.

- [Vin94] B. Vinnakota and V. V. Bapeswara Rao, "Fast conversion techniques for binary-residue number systems," *IEEE Trans. on Cir. and Syst I: Fundamental Theory and Applications*, vol. 41, pp. 927-929, Dec. 1994.
- [Vu85] T. V. Vu, "Efficient implementations of Chinese Remainder Theorem for sign detection and residue decoding," *IEEE Trans. Comp.*, vol. C-34, no. 7, pp. 646-651, Jul. 1985.
- [Ver12] H. T. Vergos and G. Dimitrakopoulos, "On modulo 2^n+1 adder design," *IEEE Trans. Comput.*, vol. 61, no. 2, pp. 173-186, Feb. 2012.
- [Ver10] H.T. Vergos, "A family of area-time efficient modulo 2^n+1 adders," in *Proc. 2010 IEEE annu. Symp. on Comput. Society.*, Lixouri, Kefalonia, July 2010, pp. 442-443.
- [Wan00a] Y. Wang, "Residue-to-binary converters based on new Chinese remainder theorems," *IEEE Trans. Cir. Syst. II, Analog Digit. Sig. Process.*, vol. 47, no. 3, pp. 192-206, Mar. 2000.
- [Wan00b] Z. Wang, G. A. Jullien and W. C. Miller, "An improved residue-to-binary converter," *IEEE Trans. Cir. Syst.-I*, vol. 47, no. 9, pp. 1437-1440, Sept. 2000.
- [Wan98] Y. Wang, "New Chinese remainder theorem," in *Proc. 32th Asilom. Conf. Signals, Syst. & Comput.*, Pacific Grove, California, USA, Nov. 1998, pp. 165-171.
- [Wan99] Y. Wang, X. Song, and M. Aboulhamid, "A new algorithm for RNS magnitude comparison based on new Chinese Remainder Theorem II," in *Proc. 1999 9th Great Lakes Symp. VLSI*, Michigan, USA, Mar. 1999, pp. 362-365.
- [Wan02] Y. Wang, X. Song, M. Aboulhamid and H. Shen, "Adder based residue to binary converters for $\{2^n-1, 2^n, 2^n+1\}$," *IEEE Trans. Signal Process.*, vol. 5, no. 7, pp. 1772-1779, Jul. 2002.

- [Yu08] J. Yu, J. Amores, N. Sebe, P. Radeva and Q.Tian “Distance learning for similarity estimation,” *IEEE Trans. on Comput.*, vol. 30, No. 3, pp. 451-462, May 2008.
- [Xu12] M. Xu, R. Yao, and F. Luo “Low-complexity sign detection algorithm for RNS $\{2^n-1, 2^n, 2^{n+1}\}$,” *IEICE Trans. Electronics*, vol. E95.C, pp. 1552-1556, Sept. 2012.
- [Xu15] M. Xu, Z. Bian, and R. Yao, “Fast sign detection algortihm for residue number system moduli Set $\{2^{n+1}-1, 2^n-1, 2^n\}$,” *IEEE Trans. VLSI Syst.*, vol. 23, pp. 379-383, Feb. 2015.
- [Zar10] B. Zarei, M. Askarzadeh, N. Derakhshanfard, and M. Hosseinzadeh, “A high-speed residue number comparator for the 3-Moduli Set $\{2^n-1, 2^n+1, 2^n+3\}$,” in *proc. 2010 Int. Symp. Signals Syst. and Electronics*, Nanjing, China, Sept. 2010, pp. 1-4.

Appendix

For Chapter 4, the hardware components of the proposed signed integer magnitude comparators and signed integer magnitude comparators designed based on the reverse converter [Pat14] for S_1 and S_2 are listed in Table 1 below;

Table 1: Breakdown of hardware components used in the proposed signed magnitude comparators and those based on reverse converter [Pat14] for S_1 and S_2 , respectively.

$S_1 \equiv \{2^{n+k}, 2^n-1, 2^{n+1}, 2^{n+1}-1\}$		$S_2 \equiv \{2^{n+k}, 2^n-1, 2^n+1, 2^{n-1}-1\}$	
Proposed Design	Signed [Pat14]	Proposed Design	Signed [Pat14]
$2 \times \text{CPA mod } 2^{2n}-1$	$2 \times \text{CPA } (3n+1)\text{-bit}$	$2 \times \text{CPA mod } 2^{2n}-1$	$2 \times \text{CPA } (3n-1)\text{-bit}$
$2 \times \text{CPA mod } 2^{n+1}-1$	$2 \times \text{CPA mod } 2^{2n}-1$	$2 \times \text{CPA mod } 2^{n-1}-1$	$2 \times \text{CPA mod } 2^{2n}-1$
$2 \times \text{CSA mod } 2^{2n}-1$	$2 \times \text{CPA mod } 2^{n+1}-1$	$2 \times \text{CSA mod } 2^{2n}-1$	$2 \times \text{CPA mod } 2^{n-1}-1$
$2 \times \text{CSA mod } 2^{n+1}-1$	$2 \times \text{CSA mod } 2^{2n}-1$	$2 \times \text{CSA mod } 2^{n-1}-1$	$2 \times \text{CSA mod } 2^{2n}-1$
$(n+1)\text{-bit binary magnitude comparator}$	$2 \times \text{CSA mod } 2^{n+1}-1$ for computing L .	$(n-1)\text{-bit binary magnitude comparator}$	$2 \times \text{CSA mod } 2^{n-1}-1$ for computing L
$(3n+k)\text{-bit binary magnitude comparator}$	$2 \times \text{CSA mod } 2^{n+1}-1$ for computing R .	$(3n+k)\text{-bit binary magnitude comparator}$	$2 \times \text{CSA mod } 2^{n-1}-1$ for computing R .
$2 \times (3n+1 \text{ AND and } 2 \text{ OR gates})$ for sign detection of integers X and Y	$(4n+k+1)\text{-bit binary magnitude comparator for comparing } X \text{ with } Y$	$2 \times (3n-2 \text{ AND and } 2 \text{ OR gates})$ for sign detection of integers X and Y	$(4n+k-1)\text{-bit binary magnitude comparator for comparing } X \text{ with } Y$
	$2 \times (4n+k+1)\text{-bit binary constant comparators for detecting the signs of integers } X \text{ and } Y.$		$2 \times (4n+k-1)\text{-bit binary constant comparators for detecting the signs of integers } X \text{ and } Y.$
Two AND gates, one OR gate, one XNOR gate, one NOR gate and one 2-to-1 multiplexer for the relational inference module	One AND gate, one XNOR gate, one NOR gate and one 2-to-1 multiplexer for the relational inference module	Two AND gates, one OR gate, one XNOR gate, one NOR gate and one 2-to-1 multiplexer for the relational inference module	One AND gate, one XNOR gate, one NOR gate and one 2-to-1 multiplexer for the relational inference module

Note: $2 \times A$ for the same component A used for operand X and operand Y .

Logic Synthesis Script

```
set DESIGN_LIST [list ]

set TOP_DESIGN simc_s1_n8_k0_reg

set CLOCK_PERIOD 2.262

set                                     PROCESS_LIB

"/net/wildar/software2/local/library/STM/cmos065_LP GP _522/CORE65GPSVT_5.1"

#-----

# Set search path

set search_path [list ./ $PROCESS_LIB/libs ]

#$env(SYNOPSYS)/libraries/syn/

#-----

# Define technology library

#read_lib buffer_memory_fast@0C_syn.lib

set target_library [list CORE65GPSVT_wc_0.90V_125C.db]

set      link_library      [list      *      CORE65GPSVT_wc_0.90V_125C.db

CORE65GPSVT_bc_1.10V_m40C.db]

set symbol_library [list CORE65GPSVT.sdb]

#-----

# Read in design in Verilog HDL

foreach module "$DESIGN_LIST" {analyze -lib WORK -format verilog ./ $module.v}

elaborate $TOP_DESIGN

current_design $TOP_DESIGN

uniquify

check_design

#-----

# Set design environments

set_min_library      "CORE65GPSVT_wc_0.90V_125C.db"      -min_version

"CORE65GPSVT_bc_1.10V_m40C.db"

# set_operating_conditions -min tt1p2v25c -max ss1p08v125c

set auto_wire_load_selection true
```

```

# set_wire_load_model -name area_4Kto5K rns_scaler_5bit_w_reg

# set_wire_load_mode enclosed

# set_driving_cell -cell ????? -pin ????? [all_inputs] # Not properly defined

# set_load ????? [all_outputs] # Not properly defined

# set_max_transition ????? current_design # Not properly defined

# set_max_fanout ????? [all_outputs] # Not properly defined

# set_max_capacitance ????? [all_outputs] # Not properly defined

#-----

# Set operating conditions

create_clock in_clock -period $CLOCK_PERIOD -waveform [list 0 [expr
$CLOCK_PERIOD/2]]

# create_generated_clock -name ????? -source ????? -divide_by ????? # Not in use

# set_clock_latency 1.0 [get_clocks clk]

# This value will not be reflected in SDF for pre-layout dynamic simulation

# set_clock_uncertainty 0.05 -setup [get_clocks clk]

# set_clock_uncertainty 0.01 -hold [get_clocks clk]

# set_clock_transition 0.25 [get_clocks clk]

# set_propagated_clock [get_clocks ?????] # Not in use for prelayout

set_dont_touch_network {in_clock}

set_drive 0 {in_clock}

#set_input_delay -max 0 -clock clk [all_inputs]

#set_input_delay -min 0 -clock clk [get_ports i_]

# set_output_delay -max 20 -clock i_clk[0] [get_ports o_mod_*] # Output delay for research
based cannot be determined easily

# set_false_path ????? # Not in use

# group_path -from [list ?????] -to [list ??????] -name ????? # Not in use

#-----

# Optimize design

set_max_area 0

```

```

#current_design cal_y2

ungroup -flatten -all

#current_design $TOP_DESIGN

#set_flatten true -design [list rns_scaler_5bit] -effort medium -phase true

#set_structure true -design [list rns_scaler_5bit] -boolean true -timing true

#set compile_new_boolean_structure true

# ungroup # Depends on design

#-----

# Compile design

compile -map_effort high

#-----

# Write the database

#write -xg_force_db -hierarchy -output ./synthesis_output/$TOP_DESIGN.db

write -format verilog -hierarchy -output ./synthesis_output/$TOP_DESIGN.sv

# set_disable_timing ????? -from ????? -to ????? # Not in use

# set_annotated_check 0 -setup -hold -from ????? -to ????? # Not in use

write_sdf -version 2.1 ./synthesis_output/pre_${TOP_DESIGN}.sd

write_sdc ./synthesis_output/pre_${TOP_DESIGN}.sdc

write_parasitics -output ./synthesis_output/pre_${TOP_DESIGN}.spef

#-----

# Create reports

report_constraint -significant_digits 4 -verbose > ./reports/pre_${TOP_DESIGN}.constraint.txt

report_area > ./reports/pre_${TOP_DESIGN}.area.txt

report_power > ./reports/pre_${TOP_DESIGN}.power.txt

#report_timing      -from      [all_inputs]      -to      [all_registers      -data_pins]      -delay
max > ./Synthesis/pre_${TOP_DESIGN}.timingmax

#report_timing              -from              [all_inputs]              -to
[all_outputs] > ./Synthesis/pre_${TOP_DESIGN}.timingmax

```

```
#report_timing    -from    [all_inputs]    -to    [all_registers    -data_pins]    -delay
min > ./Synthesis/pre_$TOP_DESIGN.timingmin

# report_timing -from ????? -to ????? -delay_type ????? -nets -capacitance -transition_time
pre_$TOP_DESIGN.timingdetail

exit

#-----The end-----
```