An FPGA implementation of redundant residue number system for low-cost fast speed fault-tolerant computations

Zhang, Yufei

2018

Zhang, Y. (2018). An FPGA implementation of redundant residue number system for low-cost fast speed fault-tolerant computations. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/89387>

<https://doi.org/10.32657/10220/47113>

**AN FPGA IMPLEMENTATION OF REDUNDANT
RESIDUE NUMBER SYSTEM FOR LOW-COST
FAST SPEED FAULT-TOLERANT
COMPUTATIONS**

ZHANG YUFEI

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Master of Engineering

2018

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, A/P Chang Chip Hong for his guidance during my M.Eng experience in the School of Electrical and Electronic Engineering, Nanyang Technological University. His enthusiasm and passion in research really motivates me throughout my research experience. Besides teaching me how should a good researcher be like, he also shared with me a lot of deep philosophy of life, which is valuable and continues inspiring me in all aspects of life.

I would like to thank my seniors Dr. Sachin and Dr. Thian Fatt for giving me guidance on RNS and sharing very useful research experiences with me. I would also like to thank my other team mates, Sergey, Zheng Yue, Liu Chaoqun, Truan, Zhong Huishu, Wang Si and Liu Wenye for inspiring and accompanying me during my two years' research life. Without them, I could not make it all the way till where I am today.

I would like to acknowledge the assistance and technical support from the staff in VIRTUS, the IC Design Center of Excellence. I would also like to thank Silicon Labs Singapore Pte. Ltd's financial support for my M.Eng program. Silicon Labs also provided me much support in terms of digital design, verification, and microcontroller knowledge, which equipped me with essential skills to be an digital IC designer. I also appreciate kind support from Ms. Seow-Guee Geok Lian and Mr. David Robert Neubronner. They have helped a lot in lab experiment equipment and software setup. Without them, I could not complete my experiment smoothly.

Last but not the least, I also want to express my gratitude to my family, who have been very supportive of my study. They encouraged me a lot throughout my M.Eng experience.

Table of Contents

Summary	vi
List of Figures	viii
List of Tables	x
Chapter 1 Introduction.....	1
1.1 Background and Motivation.....	1
1.2 Objectives.....	4
1.3 Main Contributions	5
1.4 Thesis Organization	6
Chapter 2 Literature Review	8
2.1 Residue Number System	8
2.1.1 Definition	8
2.1.2 RNS operations and properties	10
2.1.3 Design of Modulo Arithmetic Circuits	11
2.1.4 Residue-to-Binary (R2B) Conversion	18
2.2 Redundant Residue Number System (RRNS).....	20
2.2.1 Overview.....	20
2.2.2 Error Detection and Correction in RRNS	20
2.2.3 Overflow Detection in RRNS	22
2.2.4 Applications of RRNS in Fault-tolerant Computing	22
2.3 Error Detection and Correction Algorithms based on RRNS	23

2.3.1	Overview	23
2.3.2	Multi-Digit Error Detection and Correction Algorithms	25
2.3.3	A Non-Iterative Multi-Digit Residue Error Detection and Correction Algorithm.....	33
Chapter 3	Proposed Hardware Architectures for Double and Multiple Residue Digit Error Correction Algorithms	41
3.1	Overview	41
3.2	A CRT-based Syndrome Generator with Shared Logic.....	42
3.2.1	Syndrome Case 1: $\delta_1 = \left \tilde{X}_U - \tilde{X}_K \right _{M_U}$	43
3.2.2	Syndrome Case 2: $\delta_2 = \left \tilde{X}_V - \tilde{X}_K \right _{M_V}$	45
3.2.3	Syndrome Case 3: $\delta_3 = \left \tilde{X}_V - \tilde{X}_U \right _{M_V}$	48
3.3	An Arithmetic Error Retrieval Circuit for Double Residue Digit Error Correction	49
3.3.1	Error Retrieval Case 1:	50
3.3.2	Error Retrieval Case 2:	55
3.3.3	Error Retrieval Case 3:	60
3.4	A Multiple Residue Digit Error Correction Scheme by Adaptive Information Channel Partitioning	63
3.4.1	Concept and Implementation	63
3.4.2	Advantages and Limitations	71
3.4.3	Error Correction Examples	73

Chapter 4	Results and Discussion	76
4.1	FPGA Synthesis Results and Analysis.....	76
4.1.1	Proposed Double Residue Digit Error Correction Circuit vs. RRNS-based Double Residue Digit Error Correction Circuits in Literature	77
4.1.2	Comparison of Proposed Adaptive Input Partitioned Multiple Residue Digit Error Correction Circuit and Other RRNS-based Multiple Residue Digit Error Correction Circuits	81
4.2	Reliability Analysis	85
4.2.1	System Reliability	85
4.2.2	Percentile Life	91
4.2.3	Mean Time to Failure (MTTF)	92
Chapter 5	Conclusions and Future Works.....	95
5.1	Conclusions	95
5.2	Future Works.....	96
References		101

Summary

Shrinking of the device feature size allows high complexity systems to be designed and integrated with into a single chip but also causes potential issues on system reliability. Existing coding techniques can only detect and correct transmission and storage errors but not errors occurred in arithmetic operations. Redundant Residue Number System (RRNS) is a number representation that offers a more versatile fault-tolerant capability that allows error corrections in arithmetic operation with lower complexity than majority vote triple modular redundancy for fault-tolerant computing in two's complement number system. The main issue for existing RRNS-based multiple residue digit error correction algorithms is that they sacrifice a lot on either hardware cost by using large error correction lookup tables or speed due to inherent iterations of the algorithm, resulting in their inefficient hardware implementations.

This thesis presents an innovative hardware-efficient FPGA implementation of RRNS without iteration for multiple residue digit error detection and correction. The implementation is based on a syndrome-based table-lookup algorithm with new architectural design to address reduce the hardware cost and increase its speed. It adaptively partitions all the information channels such that each partitioned block contains no more than two information channels. The partitioned blocks handle the smaller scale error correction task independently and their outputs are combined to provide the corrected value. Each of the partitioned block is implemented by a modified double-error correction circuit by appropriately reorder the error vectors for table-lookup and applying modulo

arithmetic properties to replace lookup tables (LUTs) with logic and adder-based circuits. The syndrome generation module is realized by base-extension operation. An array of binary comparators is used for modulo reduction factor computation and common reverse conversion circuitries among modulo channels are reused to lower hardware cost. Hardware cost due to large modulo reduction operation is further reduced by multi-level table lookup technique. Virtex-UltraScale xcvu190-flgc2104-2-e FPGA is used to evaluate the hardware cost and the circuit speed of the proposed implementation and direct implementations of other RRNS-based multiple error correction algorithms. It has in total of 1074240 LUT slices and 2148480 flip-flops available. FPGA synthesis results show that the proposed work has the lowest hardware cost, critical path delay and throughput compared with other implementations. Using small arbitrary moduli set given that the size of each information channel no more than 8-bit, with eight information channels and eight correctable residue digit errors, the proposed circuit can be implemented with 84148 LUT slices (7.8% of the total available LUT slices) and critical path delay of 19.3 ns. This achieves 93% of hardware saving and 52.5% of speed improvement over implementations of the other recent RRNS-based algorithm. Meanwhile, the system reliability of the proposed multiple residue digit correction scheme is high enough compared with existing fault-tolerant methods for mission-critical applications.

List of Figures

Fig. 2.1 RNS building blocks.....	9
Fig. 2.2. Modulo 2^n+1 parallel-prefix adder [33].....	13
Fig. 2.3. Structure of modulo-m adder proposed in [34]	14
Fig. 2.4. Structure of the unified modular adder/subtractor for $m = 11$ and $n = 4$ [15].....	16
Fig. 2.5. Structure of modulo-m multiplier proposed in [39]	17
Fig. 2.6. Flowchart of non-iterative multi-digit error correction algorithm proposed in [31]	38
Fig. 3.1. Computation of λ_{12}	45
Fig. 3.2. Generation of I_{12}	46
Fig. 3.3. Computations of δ_{1_u} and δ_{2_v}	46
Fig. 3.4. Implementation of modulo M_K reduction.....	48
Fig. 3.5. Modified R2B conversion for Part I and Part III of Equation (3.15). 53	
Fig. 3.6. Generation of r_1	54
Fig. 3.7. Overall circuit architecture for implementation of Equation (3.15)... 54	
Fig. 3.8. Error retrieval circuits for EL1 - EL3	57
Fig. 3.9. Error retrieval circuit for EL4	58
Fig. 3.10. Error retrieval circuit for EL5	59
Fig. 3.11. Generation of $\left\ \delta_1 \right\ _{M_V} \Big _{m_i}$ for $i = k + t + 1, \dots, k + 2t$	60
Fig. 3.12. Error retrieval circuit for EL6	61
Fig. 3.13. Top level design of error retrieval circuit ($t > 2$).	62

Fig. 3.14. Proposed top-level circuit for double residue digit error correction ($t = 2$).	63
Fig. 3.15 Adaptive partitioning of information residues and allocation of their respective redundant residues.	66
Fig. 3.16. SEL and BEX circuitry for partitioned information block 1.	68
Fig. 3.17. Adaptive input partitioning scheme for multiple residue digit error correction ($\left\lfloor \frac{r}{4} \right\rfloor \geq 3$ and $\left\lfloor \frac{r}{4} \right\rfloor$ is odd).	68
Fig. 3.18. Randomization and allocation of redundant residue groups of the i^{th} partitioned information block in adaptive partitioning scheme ($\left\lfloor \frac{r}{4} \right\rfloor \geq 3$ and $\left\lfloor \frac{r}{4} \right\rfloor$ is even).	70
Fig. 3.19. Simplified adaptive input partitioning scheme for multiple residue digit error correction ($\left\lfloor \frac{r}{4} \right\rfloor < 3$).	71
Fig. 4.1 Improvement in LUT usage, critical path delay and throughput (proposed design vs. other benchmark designs).	81
Fig. 4.2 Reliability curves for the RNS under different schemes ($k = 10$).	89
Fig. 4.3 Percentile life for different designs	91

List of Tables

Table 2.1 Comparison of Multiple Residue Digit Error Detection and Correction Algorithms	32
Table 2.2. Possible Syndrome Expressions for Error Location Cases EL1-EL7	37
Table 3.1 Re-organized possible error vectors and their respective error correction implementation strategies	50
Table 3.2. Error types for different number of correctable errors (t).....	64
Table 3.3. Retrieved error digits for all partitioned information blocks and their majority voting.....	74
Table 3.4. Retrieved error digits for all partitioned information blocks and their majority voting.....	75
Table 4.1 FPGA information.	77
Table 4.2 Comparison of LUT usage for double residue digit error correction circuits.....	78
Table 4.3 Comparison of critical path delay for double residue digit error correction circuits.....	79
Table 4.4 Comparison of latency and throughput for double residue digit error correction circuits.....	79
Table 4.5 Comparison of LUT usage for multiple residue digit error correction circuits.....	83
Table 4.6 Comparison of critical path delay for multiple residue digit error correction circuits.....	83

Table 4.7 Comparison of latency and throughput for multiple residue digit error correction circuits.....	84
Table 4.8 Derivation of probability functions for three cases.	88
Table 4.9 Percentile life at different failure levels ($F = 0.1$ and $F = 0.05$).	92
Table 4.10 Summary of system reliability (R) for different designs.	93
Table 4.11 MTTF of different designs ($k = 10$).....	93

Chapter 1 Introduction

1.1 Background and Motivation

With the development of CMOS technology, device feature size has been aggressively scaled down to nano-scale level to fulfil consumer's demand for more versatile and sophisticated electronic products with higher integration density, faster speed, smaller size and lower power consumption [1]. The time has come that digital systems become one of the necessities in every aspect of our daily life. Nowadays, they are designed to accomplish jobs that require high precision and huge workload or complex tasks dealing with life-threatening problems [2]. Those applications include military, medical, power control system, commercial missions such as air traffic control, banking, communications and so on. Most of them require high reliability of the whole system to function correctly, or it will cause economic damage, severe social confusion or even endanger human life [3] [4].

Shrinking of the device feature size causes reliability issues. Since the smaller device becomes more sensitive to parameter variability, failure mechanisms due to parameter uncertainty become more severe and increase the unpredictability of the system [4]. The reliability of a system decreases drastically when it is highly integrated [4]. Consequently, high-quality components and good design techniques are required to guarantee high system reliability. For example, if the defect rate of a “reliable” device (transistor) is 1 ppm ($10^{-4}\%$), then for a VLSI system containing 1 million non-redundant components, the reliability R of the system would become

$$R = (1 - 10^{-4}\%)^{1000000} \approx 37\% \quad (1.1)$$

In addition, as slower pace in scaling down the supply voltage causes higher electric field applied on gate oxide, the probability of device breakdown will be elevated [5]. When going down to nano-meter scale, transistors are more likely to be affected by noise, infrared, EM radiation, aging and other dynamic conditions such as malicious fault injection and non-permanent faults like transient fault and intermittent fault when operating at low supply voltage [6]. Therefore, even if a system is defect-free upon production, it can still be perturbed by unexpected environmental factors beyond the control of the designers. Since it is impossible to build a “perfect” system free of errors, and to avoid huge yield loss and product failures, fault tolerant design is imperative to enhance the system operational reliability [4].

Fault-tolerance in widely used two’s complement number system (TCS) is typically achieved by incorporating certain amount of redundancy [7]. Techniques such as using majority voting with replicated hardware components, self-checking logic, error correction code and circuit reconfiguration have been used [2]. However, these methods are non-scalable and expensive [8]. Some of the checking and reconfiguration approaches to counter soft errors require large memory with high reliability or enormous amount of time on testing and reconfiguration [5]. Except for majority voting, none of these technique can propagate its error correction capacity down to the arithmetic operations. Unfortunately, for majority voting technique, even the simplest form of triple modular redundancy will require three times more hardware resources. The cost

of extra resources escalates as many more modules with the same functionality need to be replicated at finer granularity to achieve a high level of reliability.

Redundant Residue Number System (RRNS) makes a good candidate for fault-tolerant system design. For arithmetic operations in Residue Number System (RNS), computation errors in each residue channel can be isolated to avoid error from propagating from one channel to another. Therefore, errors can be removed by simply shutting down the faulty modulus channel as long as the surviving channels have adequate dynamic range. Such unique computational error isolation property is especially useful in security and biomedical applications with zero error-tolerance [8]. Based on this principle, by adding redundant residue channels, RRNS can be used for error detection and correction. Different from coding techniques in TCS, algorithms based on RRNS are not only able to detect and correct transmission errors caused by noise, but also detect and correct arithmetical processing errors [9] due to component defects, device processing variation, voltage and temperature variations and deliberately injected faults [8]. They require substantially less hardware overheads compared with majority voting triple modular redundancy method in TCS to achieve similar reliability.

Generally, for single digit error detection and correction [10] [11] [12] [13] [14], it is relatively easier to decode error locations with only two redundant moduli channels and the circuit structures are less complex. However, detecting and correcting multiple residue digit errors remain theoretically appealing but practically challenging. This is mainly because of the pairwise prime criteria of moduli in RNS has limited the number of special moduli of comparable word

length with good number theoretic properties from being used to simplify the error decoding for RRNS with high error correction capability. The identification of multiple digit errors requires either iterations, recursive computations and comparisons [15] [16] [17] [18] [19] or large error lookup table [8] [20] for parallel hardware implementation. Due to the iterative structures [17], most research works stop at algorithmic level for general purpose processor without advancing into architectural simplification or optimization for application-specific integrated circuit (ASIC) or field programmable gate array (FPGA) implementation [8] [20].

1.2 Objectives

Motivated by the attractive property of error localization and error correction capability for arithmetic operations, the challenges on hardware realization of multiple digit error detection and correction algorithms in RRNS are addressed in this research. The aim is to develop a new and general architecture for RRNS which will scale well to different dynamic range and error correction requirement for efficient hardware implementation. The implementation targets FPGA platform due to its advantages of reconfigurability, flexibility and programmability over the ASIC platform. Small arbitrary moduli are chosen to reduce the area-time complexity of arithmetic operations in the information channel and error decoding. Existing RRNS error correction algorithms are analysed for their limitations in error correction capability and hardware implementation complexity bottlenecks. Key research problems to be overcome include the inevitable dependencies among operations in iterative algorithms,

non-deterministic time bound in error decoding and excessive hardware requirements of large lookup-tables (LUTs) and complex modular arithmetic operations with limited exploitable number theoretic properties from arbitrary moduli as the dynamic range of information or the number of correctable errors increases.

1.3 Main Contributions

This thesis presents a novel design of a pipelined circuit for multiple residue digit error correction in general arbitrary moduli set based RRNS for efficient FPGA implementation. Although the basic error correction algorithm has been adapted from [8], several important improvements have been made to reduce the hardware implementation cost. First, the growth of LUT sizes with input dynamic range, number of information channels and error correction capability has been significantly suppressed by partitioning the input moduli channels into subblocks. Each subblock of smaller number of moduli is capable of double-error correction and can be executed in parallel with reduced computational complexity. Secondly, an arithmetic-based double residue digit error correction circuit has been proposed. Error lookup tables have been eliminated by reordering the error vectors of lookup table to simplify certain modulo arithmetic computation for efficiently implementation by logic and adder-based circuits. Thirdly, the elimination of large error lookup tables enables more operations without data dependency to be parallelized. The critical path delay in each pipelined stage has been significantly reduced to increase the throughput rate. Finally, the complexity of using Chinese Remainder Theorem (CRT) for syndrome

generation has been substantially reduced by reusing common circuitries among different moduli channels.

1.4 Thesis Organization

The thesis is organized into five Chapters.

Chapter 1 introduces the research motivation, objective and summarizes the major contributions of this work.

Chapter 2 reviews the theory of RNS, RRNS and the design of modulo arithmetic circuits in RNS such as modulo adders and modulo multipliers. Existing RRNS error correction algorithms are reviewed, and their pros and cons are analyzed. A non-iterative multiple residue digit error correction algorithm is singled out for detailed analysis. Its flowchart, uniqueness conditions and advantages over other algorithms are presented. Numerical examples are used to illustrate the error correction procedure. Design challenges for the hardware implementation of non-iterative algorithm are highlighted.

Chapter 3 presents the proposed hardware architectures for the implementations of double and multiple residue digit error correction circuits based on the non-iterative residue digit error correction algorithm. Design challenges for the existing algorithms are discussed. Three major circuit improvements, namely, adaptive partition of input channels, pipelined arithmetic-based error retrieval

double-error correction circuitry and hardware reuse for syndrome generation are discussed in detail.

In Chapter 4, FPGA synthesis results, as well as MATLAB simulation results for the proposed circuit implementations and direct implementation of existing RRNS-based algorithms are presented and analyzed. Comparisons are made based on FPGA device XC7z020 synthesis results with recent RRNS-based multiple error digit error detection and correction algorithms to show the efficiency of our proposed design in terms of LUT resource utilization rate, critical path delay, latency and throughput. System reliability parameters for each design in terms of reliability, percentile life and mean time to failure (MTTF) are also evaluated and compared.

Chapter 5 concludes the thesis and suggests possible works based on the current findings, which includes the possible improvements on the proposed multiple residue digit error correction scheme, the implementation of proposed design using modular arithmetic friendly special moduli set for specific applications such as hybrid memory and the application of the proposed design as a countermeasure to the differential fault analysis (DFA) attack in cryptography.

Chapter 2 Literature Review

2.1 Residue Number System

2.1.1 Definition

Residue Number System (RNS) is a non-weighted number system invented by Harvey Garner in 1959 [21]. In RNS, an integer X can be represented by a k -tuple $(x_1, x_2, \dots, x_k)$ with respect to a set of positive coprime integers $(m_1, m_2, \dots, m_k)$. The magnitude of X is bounded by the range $[0, M_K - 1]$, where $M_K = \sum_{j=1}^k m_j$. Therefore, a large integer can be represented by several smaller residue digits. Each residue digit x_i is the least non-negative remainder of X divided by m_i , i.e., $x_i = X \bmod m_i$ (or $|X|_{m_i}$) [22]. Given the residue representation $(x_1, x_2, \dots, x_k)$, the integer X can be reversely computed from residue to integer representation by Mixed-Radix Conversion (MRC) or Chinese Remainder Theorem (CRT) [23]. Fig. 2.1 shows the typical building blocks used for an RNS system. The operands in two's complement representation are passed through forward converter for residue digit computation. Each converted residue digit will then be processed in its respective modulo channel. Upon processing of all the data from residue channels, reverse converter is used to retrieve the integer output from its residue representation [24].

In the conventional binary number system, the long carry-propagation chain strictly limits the circuit speed and becomes the main bottleneck of fast arithmetic operation [24]. On the contrary, in RNS arithmetic, modular arithmetic operations are independently carried out with respect to each residue channel

without inter-carry propagation. Instead of performing big word length operations using two's complement number system, several modulo arithmetic operations with shorter word length can be carried out in parallel for faster circuit speed using RNS [24].

Fig. 2.1 RNS building blocks.

The hardware implementation of an RNS based application largely relies on the selection of moduli set. In general, the moduli set is classified into two types. The first type of moduli set is arbitrary moduli set [25] [26] [27]. There is no restriction on the set of integers except that they must be coprime. The flexibility makes it easy to find a high cardinality moduli set with balanced word length among the residue channels. The second type of moduli set is special moduli set, where the integers moduli possess strong mathematical properties that can help to simplify modulo arithmetic operations. Special moduli in the forms of 2^n and $2^n \pm 1$ [28] [29] [30] has hardware friendly arithmetic operations which help to produce more cost-effective designs [24]. The drawback of special moduli set is

that high cardinality moduli set are unbalanced. The system's throughput is limited by the largest modulus.

2.1.2 RNS operations and properties

In RNS, operations on residue digits such as addition, subtraction and multiplication can be performed individually on each residue channel without affecting other channels. The operations are expressed as follows [22]:

$$\begin{aligned} X \pm Y &\equiv \langle |x_1 \pm y_1|_{m_1}, |x_2 \pm y_2|_{m_2}, \dots, |x_n \pm y_n|_{m_n} \rangle \\ X \times Y &\equiv \langle |x_1 \times y_1|_{m_1}, |x_2 \times y_2|_{m_2}, \dots, |x_n \times y_n|_{m_n} \rangle \end{aligned} \quad (2.1)$$

where m_i for $i = 1, 2, \dots, n$ are the n co-prime moduli, $x_i = |X|_{m_i}$ and $y_i = |Y|_{m_i}$

are the residues of the integer operands, X and Y , respectively.

For example, let $(m_1, m_2, m_3) = (3, 5, 7)$ be the moduli set of the RNS. The dynamic range, $M = 3 \times 5 \times 7 = 105$. Consider two integers operands, $X = 40$ and $Y = 30$. Their residue representations after forward conversion are $\langle x_1, x_2, x_3 \rangle = \langle 1, 0, 5 \rangle$ and $\langle y_1, y_2, y_3 \rangle = \langle 0, 0, 2 \rangle$. Equation (2.2) shows the operations of $X + Y$, $X - Y$ and XY performed in the residue domain.

$$\begin{aligned} \langle x_1, x_2, x_3 \rangle + \langle y_1, y_2, y_3 \rangle &= \langle |1+0|_3, |0+0|_5, |5+2|_7 \rangle = \langle 1, 0, 0 \rangle \\ \langle x_1, x_2, x_3 \rangle - \langle y_1, y_2, y_3 \rangle &= \langle |1-0|_3, |0-0|_5, |5-2|_7 \rangle = \langle 1, 0, 3 \rangle \\ \langle x_1, x_2, x_3 \rangle \times \langle y_1, y_2, y_3 \rangle &= \langle |1 \times 0|_3, |0 \times 0|_5, |5 \times 2|_7 \rangle = \langle 0, 0, 3 \rangle \end{aligned} \quad (2.2)$$

The following mathematical properties of modulo operation are useful for simplifying complex expressions for circuit reduction [22], where x is an integer variable, and k and m are integer constants.

Property 1: $|xm|_m = 0$

Property 2: $|x \pm km|_m = |x|_m$

Property 3: $|kx|_{km} = k|x|_m$

Property 4: $|-x|_m = |m - x|_m$

Property 5: $x - |x|_m = \left\lfloor \frac{x}{m} \right\rfloor \cdot m$

Property 6: The multiplicative inverse of residue digit $|x|_m$ is denoted by $|x^{-1}|_m$

and it can be represented as follows:

$$|x \times |x^{-1}||_m = 1 \quad (2.3)$$

Property 7 [31]: given two moduli a and b , if b divides a , then $||x|_a|_b = |x|_b$.

2.1.3 Design of Modulo Arithmetic Circuits

2.1.3.1. Modulo Adders/Subtractors

Modulo adder is one of the basic building blocks in RNS computations. Since subtraction can be implemented as addition with the negated subtrahend [22], a modulo subtractor can be easily implemented using a modulo adder with a minor modification. A modulo adder can be implemented with mainly lookup tables (LUTs)/ROMs, full combinational logic, or a combination of the two. Based on

modulus, adders are divided into two categories: (1) adders with hardware-friendly special moduli of power of two with zero or plus/minus 1 offset, and (2) adders with small arbitrary moduli.

For the first category of adders, it is relatively easy to implement since operations with 2^n can be realized by simple shift or truncation without extra logic cost. To add two numbers A and B modulo m (where $0 \leq A, B < m$), a straightforward way to design a modulo $2^n - 1$ adder is to use two parallel prefix adders, one for computing $A + B$ and the other for computing $A + B + 1$. Then a multiplexer is used to select one of the two result based on intermediate sum value. For the addition based on modulo $2^n + 1$, diminished-one representation can be used [32]. A main drawback of this method is that conversions are required for the operands and the result. A better approach [33] is to subtract $2^n + 1$ at the beginning and then add it back if the result is negative. Let $X = A + B - (2^n + 1)$, then the modulo addition of A and B can be expressed as

$$|A + B|_{2^n + 1} = \begin{cases} X & \text{if } X \geq 0 \\ 2^n + |X + 1|_{2^n} & \text{if } X = -1 \\ |X + 1|_{2^n} & \text{otherwise} \end{cases} \quad (2.4)$$

Equation (2.4) can be implemented by adding a carry-saved adder (CSA) stage to the modulo $2^n - 1$ adder for the subtraction and making use of the carry out and propagate signals of the last stage of the prefix tree to generate an end-around-carry, as shown in Fig. 2.2.

Fig. 2.2. Modulo 2^n+1 parallel-prefix adder [33].

For the second category of adders, the expression for the modulo m addition of two integers A and B , $0 \leq A, B < m$ is given by:

$$|A+B|_m = \begin{cases} A+B & \text{if } A+B < m \\ A+B-m & \text{otherwise} \end{cases} \quad (2.5)$$

If the modulus m is sufficiently small, the addition can be realized by a table lookup process. The table takes in the concatenation of A and B as an input address and the output returns the precomputed result of $|A+B|_m$. If faster addition is required, it can be implemented using a combination of CSAs and Carry Propagation Adders (CPA).

Fig. 2.3. Structure of modulo-m adder proposed in [34]

In [34], a modulo adder architecture was proposed based on CSA and CPA. It uses multiplexers (MUXs) to select the carry generate and propagate signals before driving them into the carry computation unit. Given two n -bit input integers X and Y , their binary representations are $x_{n-1} \dots x_1 x_0$ and $y_{n-1} \dots y_1 y_0$. In the design, half-adder like (HAL) cells are used to generate the intermediate signals. The circuit structure is shown in Fig. 2.3, where SAC is the sum-and-carry unit, CPG is the carry propagate generate unit, and CLA is the carry look ahead unit for computing C_{out} . a and b are the sum and carry outputs from the SAC unit and they can be expressed by

$$a_i = \begin{cases} A_i & \text{if } z_i = 0 \\ \hat{A}_i & \text{if } z_i = 1 \end{cases} \quad (2.6)$$

$$b_i = \begin{cases} B_i & \text{if } z_{i-1} = 0, i \neq 0 \\ \hat{B}_i & \text{if } z_{i-1} = 1, i \neq 0 \\ 0 & \text{if } i = 0 \end{cases} \quad (2.7)$$

where $A_i = x_i \oplus y_i$, $B_i = x_i y_i$, $\hat{A}_i = \overline{x_i \oplus y_i}$ and $\hat{B}_i = x_i + y_i$ for $i = 0, 1, \dots, n-1$.

$Z \equiv z_{n-1} \dots z_1 z_0$ is the correction factor and is defined as $Z = 2^n - m$. P , G , p and g are the propagate and generate signals from the CPG unit, where $P_i = A_i \oplus B_i$, $G_i = A_i B_i$, $p_i = a_i \oplus b_i$ and $g_i = a_i b_i$. A MUX is used for the selection of P , G or p , g into the CLA summation unit (CLAS). The CLAS unit is used for the computation of carries.

A unified modular adder/subtractor for arbitrary moduli is proposed in 2014 [35]. The unified structure is achieved by changing the range selection criteria of modular subtraction and identifying the common computations of $|x + y|_m$ and $|x - y|_m$. A binary signal s can be used to switch the function of the circuit between modulo addition and modulo subtraction. In this way, modular addition and subtraction for any modulus can be merged. The reported work can be implemented by a regular 3-stage architecture, as shown in Fig. 2.4. The first stage can be implemented using HAL cells proposed in [34] to generate the sum and carry terms a_i^j and b_{i+1}^j for $0 \leq i \leq n-1$ and $j = 0, 1$. The second stage can be implemented by an array of 2-to-1 input MUXs and parallel prefix adders (PPAs) to obtain the intermediate result of modulo addition and subtraction of w and v . A selection signal s is used in the final stage to select the desired output. The final stage consists of simple logic gates and n 2-to-1 input MUXs.

Fig. 2.4. Structure of the unified modular adder/subtractor for $m = 11$ and $n = 4$ [15]

2.1.3.2. Modulo Multipliers

Similar to modulo adders, circuit architectures for modulo multiplication can be divided into two categories: (1) multipliers with commonly used special moduli, and (2) multipliers with small arbitrary moduli. For the multipliers based on special moduli set, researchers have been devoting effort into more hardware-efficient designs over the years. Many designs such as [30] [36] [37] [38] have been proposed for better circuit performance and smaller circuit area.

For arbitrary moduli set, a modulo multiplier was proposed in [39] by using conventional multiplier to obtain $z = xy$ followed by a modulo reduction to obtain the result of modulo multiplication, as shown in Fig. 2.5. Given that $a = 2^n - m$ and assume that k bits are used to represent a . The $2n$ -bit product Z can be written as

$$Z = D2^{2^{n-k-1}} + C2^n + B2^{n-1} + A \quad (2.8)$$

where A , B , C and D are $(n-1)$ -, 1-, $(n-1-k)$ - and $(k+1)$ -bit words. It can be proved that $(C2^n) \bmod m = Ca$, which can be represented by $n-1$ bits. The resulting $D2^{2^{n-k-1}}$, $C2^n$ and $B2^{n-1}$ are then added together using an n -bit CSA followed by a mod m adder for the final output.

Fig. 2.5. Structure of modulo- m multiplier proposed in [39]

2.1.4 *Residue-to-Binary (R2B) Conversion*

Unlike two's complement number system, due to its non-weighted property of RNS, the magnitude of a set of residue digits cannot be computed as straightforwardly as in binary two's complement number system. The magnitude of an integer X can be evaluated from its residue representation $(x_1, x_2, \dots, x_k)$ by MRC or CRT as mentioned earlier. The mixed radix (MR) [23] representation can be expressed as follows:

$$X = \sum_{i=1}^k a_i \prod_{j=1}^{i-1} m_j + a_1 \quad (2.9)$$

where a_i is the mixed-radix digit and it can be computed from $(x_1, x_2, \dots, x_k)$ by

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= \left| (x_2 - a_1) \right|_{m_1}^{-1} \Big|_{m_2} \\ a_3 &= \left| \left((x_3 - a_1) \right|_{m_1}^{-1} - a_2 \right) \right|_{m_2}^{-1} \Big|_{m_3} \\ &\dots \\ a_k &= \left| \left(\left((x_k - a_1) \right|_{m_1}^{-1} - a_2 \right) \right|_{m_2}^{-1} - \dots - a_{k-1} \right) \right|_{m_{k-1}}^{-1} \Big|_{m_k} \end{aligned} \quad (2.10)$$

The MR system obtained by MRC is a weighted number system. Although the magnitude of an MR number can be evaluated by simply summing up all the products of mixed-radix digits and moduli, computations of a_i is dependent on a_{i-1} . This means that MRC needs to be carried out with sequential computations and the number of sequential steps increases linearly with the number of moduli. Among all the mixed-radix digits, the computations of a_k has the longest delay, which severely limits the overall circuit performance. When the number of

moduli in a selected moduli set is large, MRC is no longer suitable for magnitude evaluation as the computation time becomes too long to evaluate one integer.

The other way of R2B conversion is to use CRT and it can be evaluated as follows:

$$X = \left| \sum_{j=1}^k M_j \left| M_j^{-1} \right|_{m_j} x_j \right|_{M_K} = \sum_{j=1}^k M_j \left| M_j^{-1} \right|_{m_j} x_j \Big|_{m_j} - \lambda M_K \quad (2.11)$$

where $M_j = \frac{M_K}{m_j}$, $\left| M_j^{-1} \right|_{m_j}$ is the multiplicative inverse of $\left| M_j \right|_{m_j}$, and λ is an

integer smaller than k . For example, consider the moduli set $\{m_1, m_2, m_3\} = \{7, 8, 9\}$. The dynamic range of the moduli set can be calculated as

$M_K = 7 \times 8 \times 9 = 504$. Since $M_1 = 504/7 = 72$, $M_2 = 504/8 = 63$, and

$M_3 = 504/9 = 56$, the corresponding multiplicative inverses are $\left| M_1^{-1} \right|_{m_1} = 4$,

$\left| M_2^{-1} \right|_{m_2} = 7$ and $\left| M_3^{-1} \right|_{m_3} = 5$. Therefore, the integer can be represented by residue

format as $X \equiv (x_1, x_2, x_3) \equiv (0, 5, 2)$. X can be computed by CRT as follows:

$$\begin{aligned} X &= \left| M_1 \left| M_1^{-1} \right|_{m_1} x_1 + M_2 \left| M_2^{-1} \right|_{m_2} x_2 + M_3 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{M_K} \\ &= \left| (72 \times 4 \times 0) + (63 \times 7 \times 5) + (56 \times 5 \times 2) \right|_{504} \\ &= 245 \end{aligned} \quad (2.12)$$

In CRT, the computation of each partial sum $M_j \left| M_j^{-1} \right|_{m_j} x_j \Big|_{m_j}$ is independent of

each other. Therefore, all the computations can be parallelized and operated at a faster speed compared to the sequential computations of MRC. However, after the summation of partial sum, modulo reduction by M_K must be performed to the summation result to ensure that the result X is in the range $[0, M_K - 1]$. Since M_K

is usually a large number, the CRT reverse converter is complicated and incurs high implementation cost.

2.2 Redundant Residue Number System (RRNS)

2.2.1 Overview

By adding redundant moduli into the existing moduli set of an RNS, Redundant Residue Number System (RRNS) can be formed to detect and correct residue digit errors without altering the original informational dynamic range. An RRNS has a base of $n = k + r$ coprime moduli $\{m_1, m_2, \dots, m_k, m_{k+1}, \dots, m_n\}$ and a dynamic range of $M_N = \prod_{i=1}^n m_i$. The additional moduli, $m_{k+1}, m_{k+2}, \dots, m_{k+r}$, are called the redundant moduli as opposed to the information moduli, $m_1, m_2, \dots, m_k$, of the original base. The interval $[0, M_K - 1]$ of the k information residue digits of the n -tuple is called the legitimate range and it represents the useful computational range of the number system. The interval $[M_K, M_N - 1]$ due to the additional r redundant residue digits is called the illegitimate range and it is used for error and overflow detection [31].

2.2.2 Error Detection and Correction in RRNS

The basis for RRNS error detection and correction is that the residue form of an error-free integer is uniquely mapped into the interval $[0, M_K - 1]$. With additional r residue digit channels added together with the information moduli, an (n, k) maximum distance separable (MDS) RRNS code can be formed [40].

Based on CRT, errors with total magnitude E can be linearly added with integer X within the legitimate range. It can be expressed as $\tilde{X} = X + E$, where $\tilde{X}$ stands for the integer in error. It is observed that $\tilde{X}$ always falls into the illegitimate range if $m_{k+1}, m_{k+2}, \dots, m_{k+r} > m_i, \forall i \leq k$ and the number of erroneous residues is less than or equal to r [17]. The corrected residues can be recovered from the magnitude $\tilde{X}$. The number of detectable and correctable residue digit errors are controlled by the number of redundant moduli. Based on the coding theory for RRNS in [40], an RRNS with r redundant modulo channels has the minimum non-zero Hamming weight $wt_{\min} \geq r+1$ and minimum distance $d_{\min} \geq r+1$ [41]. It can detect up to r residue digit errors and correct up to $t = \lfloor r/2 \rfloor$ residue digit errors [14].

Let l_i be the positions of the erroneous residue digits and e_{l_i} be the magnitude of error injected into the corresponding residue digit in the modulus channel m_{l_i} for $1 \leq i \leq t$. Assume that t residue digit errors are introduced into $X = (x_1, x_2, \dots, x_n)$, the error magnitude E_l can be represented by an n -tuple $(e_{l_1}, \dots, 0, e_{l_2}, \dots, e_{l_t})$ in RRNS, where $n = k + r$ and $r = 2t$. The residue representation of $\tilde{X}$ with t residue digit errors can be expressed as:

$$\begin{aligned} \tilde{X} &= X + E_l \\ (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n) &= (x_1, x_2, \dots, x_n) + (e_{l_1}, \dots, 0, e_{l_2}, \dots, e_{l_t}) \\ &= \left(\left| x_{l_1} + e_{l_1} \right|_{m_{l_1}}, \dots, \left| x_{l_t} + e_{l_t} \right|_{m_{l_t}} \right) \end{aligned} \quad (2.13)$$

For convenience, the residue representation of E_l is referred to as the error vector.

Based on CRT, E_l can also be computed from its error vector $(e_{l_1}, \dots, 0, e_{l_2}, \dots, e_{l_t})$:

$$E_l = \left| \sum_{i=1}^t \left| M_{l_i} \right|_{m_{l_i}} \left| M_{l_i}^{-1} \right|_{m_{l_i}} e_{l_i} \right|_{m_{l_i}} \right|_{M_N} \quad (2.14)$$

where $\left| M_{l_i} \right|_{m_{l_i}} = \left| M_N / m_{l_i} \right|_{m_{l_i}}$ and $\left| M_{l_i}^{-1} \right|_{m_{l_i}}$ is the multiplicative inverse of $\left| M_{l_i} \right|_{m_{l_i}}$.

2.2.3 *Overflow Detection in RRNS*

Overflow occurs when the computation results exceed the defined dynamic range [24]. In digital filters, it is necessary to detect the occurrence of overflow to suppress overflow oscillations and discard erroneous outputs [42]. According to the theory in [42], by adding an additional redundant modulo channel in addition to the k information modulus channels, overflow can be detected by determining whether the received digits fall in the legitimate range $[0, M_K - 1]$. Including a redundant modulus channel for the sole purpose of overflow detection is expensive in comparison with using guard bits in a conventional two's complement number system. However, if redundancy is already provided in the circuit for error detection and correction, then overflow can be accomplished easily.

2.2.4 *Applications of RRNS in Fault-tolerant Computing*

One of the applications of RRNS in error detection and correction is hybrid memory [43]. An error correction code based on six-moduli RRNS was implemented on hybrid memories. It provides larger data storage compared with

Reed-Solomon codes of similar error correction capability due to shorter codeword length

RRNS was also used in FIR filters to mitigate errors. Comparing with triple modular redundancy (TMR), the RRNS-based approach consumed less hardware area for the same error correction capability [44]. It can eliminate soft errors produced by single-event upset (SEU) in FIR filters with zero fault missing rate and less hardware area than traditional SEU mitigation schemes [45].

Multicarrier modulation scheme has also adopted RRNS concept to curb the frequency-selective fading effects. The scheme was adaptively coded in RRNS with three information and six redundant residues with each residue channel length of 8 bits. Compared with convolutional constituent code, it has better bit per symbol throughput when the channel signal-to-noise ratio (SNR) is above 15 dB [46].

2.3 Error Detection and Correction Algorithms based on RRNS

2.3.1 *Overview*

Typically, error detection and correction in RRNS can be performed in three sequential steps: 1) error existence check, 2) identification of erroneous residue digits and 3) error correction. For the first step, one basic approach is to check the magnitude of the residue representation of an RRNS. If its magnitude falls outside the legitimate range, then the representation is declared to be in error. The most straightforward method to compute the integer magnitude is to perform

full R2B conversion using either MRC or CRT. Since MRC is a sequential process, its delay is linearly related to the number of residue channels. Therefore, it usually requires very long computation time for each round of error checking. As for CRT, although its operations can be parallelized, its computation complexity due to full magnitude evaluation on all received residue digits will increase when the information dynamic range becomes larger.

After the existence of the residue digit error is confirmed, the next step is to locate and find the magnitude of the erroneous digits. This is the most difficult and complicated step among all three steps mentioned above. For single residue digit error cases, only two redundant moduli and limited error locations are involved. Therefore, the identification of the erroneous digit is substantially less complex. The common techniques used for error locating can be summarized into three categories: 1) consistency checking [40], 2) iterative modulus-projection computation [12], and 3) syndrome checking [14] [11]. However, when it comes to locating multiple residue digit errors, the identification process becomes much more complicated. It usually requires longer time and more operations to look for errors among the enormous number of different residue digit error locations and to retrieve error magnitudes.

Upon identifying the error digits locations and magnitudes, these erroneous digits can be independently corrected by simply subtracting the error residue digits from the received erroneous residue representation. The corrected output can either be represented in integer form or residue form.

2.3.2 Multi-Digit Error Detection and Correction Algorithms

As for multiple digit error detection and correction, in general, erroneous residue digits can be located by three different approaches. They are 1) error recovery by CRT, 2) modulus projection and 3) consistency check using syndrome. Algorithms based on these three approaches in literature will be discussed in this section.

The first algorithm for multiple residue digit error detection and correction was proposed in 1973 [15]. It can correct up to t residue digit single-burst errors with $2t$ redundant moduli by base-extension approach. The syndrome values $|\Delta|_{m_i}$ are obtained by taking the difference between the received residue digits x_i and the based-extended residue digits $|X_K|_{m_i}$, as shown in (2.13):

$$|\Delta|_{m_i} = \left| |X_K|_{m_i} - x_i \right|_{m_i} \quad (2.15)$$

where $X_K = (x_1, x_2, \dots, x_k)$ represents the magnitude of the received information residue digits and $i = 1, 2, \dots, k + 2t$. Upon computation of syndrome digits $|\Delta|_{m_i}$, they are grouped into $\lceil (k+t-1)/(t+1) + 1 \rceil$ groups. Each group consists of $2t$ syndrome digits as shown below:

$$\begin{aligned} S_1 &= \left\{ |\Delta|_{m_{k+1}}, |\Delta|_{m_{k+2}}, \dots, |\Delta|_{m_{k+2t}} \right\} \\ S_2 &= \left\{ |\Delta|_{m_{k-t}}, |\Delta|_{m_{k-t+1}}, \dots, |\Delta|_{m_{k+t-1}} \right\} \\ &\dots \\ S_{\lceil (k+t-1)/(t+1) + 1 \rceil} &= \left\{ |\Delta|_{m_1}, |\Delta|_{m_2}, \dots, |\Delta|_{m_{2t}} \right\} \end{aligned} \quad (2.16)$$

The erroneous residue digits can thus be determined by inspecting the elements of S_i for $i = 1, 2, \dots, \lceil (k+t-1)/(t+1) + 1 \rceil$. The erroneous residue digits can be

located at modulus channels m_i if the corresponding S_i have no more than t non-zero elements. The correct residue representation is then obtained by replacing the erroneous residue digits with the corresponding base-extended residue digits. The complexity of this algorithm is dependent on the number of sets to be examined and the number of syndromes to be computed for each set.

In 1976, an error decoding algorithm making use of continued fractions was proposed [18]. In the algorithm, the erroneous residue digits are not retrieved directly. Instead, they are recursively computed using Euclid's algorithm. The correct error fraction is then used to compute the magnitude of the error-free residue representation. However, this algorithm is inefficient for hardware implementation due to two reasons. Firstly, more iterations are required for larger number of moduli and residue digit errors, restricting the circuit to run at very low speed. Secondly, since the number of iterative steps cannot be determined in the design phase, it is very hard for designers to allocate resources and optimize the design. As suggested in [18], the algorithm was recommended to be executed on a general-purpose computer without specifying its maximum number of iterations.

Sun and Krishna [16] developed a coding theory approach for error control in RRNS in 1992. They introduced the concepts of Hamming weight, minimum distance, weight distribution and error detection and correction capabilities in RRNS. In [16], four algorithms were proposed based on consistency checking, double residue digit error correction and multiple residue digit error detection, single-burst residue error correction and extended double residue digit error

correction. Similar to the algorithm in [15], the first three algorithms use syndrome consistency check for error locating except that only the redundant residue digits are involved in syndrome generation. For double residue digit error correction, it requires four redundant residue channels and the base extension method to compute the consistency quantities $|\Delta|_{m_{k+r}}$ for $r=1,2,3,4$. These quantities are derived as follows:

$$|\Delta|_{m_{k+r}} \equiv \left| e_{ij}^{(1,r)} \frac{M_N}{m_i m_j} \right|_{m_{k+r}} \quad (2.17)$$

$$|\Delta|_{m_{k+r}} \equiv \left| \left(e_{ij}^{(2,r)} - m_i m_j \right) \frac{M_N}{m_i m_j} \right|_{m_{k+r}} \quad (2.18)$$

where $e_{ij}^{(1,r)}$ and $e_{ij}^{(2,r)}$ for $r=1,2,3,4$ are the error digits introduced into the information residue channels m_i and m_j . By substituting possible m_i and m_j , $e_{ij}^{(1,r)}$ and $e_{ij}^{(2,r)}$ can be computed. For the case that errors only occur in information residues, based on $\{e_{ij}^{(1,1)}, e_{ij}^{(1,2)}\}$ and $\{e_{ij}^{(2,1)}, e_{ij}^{(2,2)}\}$, the error magnitude $e_{ij}^{(1)}$ and $e_{ij}^{(2)}$ can be computed by CRT:

$$\begin{aligned} e_{ij}^{(1)} &\equiv \left| \sum_{r=1}^2 \left| e_{ij}^{(1,r)} \right|_{m_{k+r}} \right|_{m_{k+1} m_{k+2}} \\ e_{ij}^{(2)} &\equiv \left| \sum_{r=1}^2 \left| e_{ij}^{(2,r)} \right|_{m_{k+r}} \right|_{m_{k+1} m_{k+2}} \end{aligned} \quad (2.19)$$

The location of the erroneous residue digits can be determined by checking two conditions on the error magnitude. Firstly, if $e_{ij}^{(1)} < m_i m_j$ and Equation (2.17) holds by substituting $e_{ij}^{(1,r)}$ for $e_{ij}^{(1)}$, then the consistent solution is $e_{ij}^{(1)}$. Secondly, if $e_{ij}^{(2)} < m_i m_j$ and Equation (2.18) holds by substituting $e_{ij}^{(2,r)}$ for $e_{ij}^{(2)}$, then the consistent solution is $e_{ij}^{(2)}$. The corresponding information residue error digits can

be retrieved from the consistent solution of $e_{ij}^{(1)}$ or $e_{ij}^{(2)}$. For the case that one residue digit error occurs in the information residues one residue digit error occurs in the redundant residues, it requires to form error magnitude based on $\{e_{ij}^{(1,1)}, e_{ij}^{(1,2)}\}, \{e_{ij}^{(2,1)}, e_{ij}^{(2,2)}\}, \{e_{ij}^{(1,3)}, e_{ij}^{(1,4)}\}$ and $\{e_{ij}^{(2,3)}, e_{ij}^{(2,4)}\}$. The information residue error can still be retrieved by the consistency checking as in the previous case. The redundant residue error is determined by firstly finding the error magnitude $e_{ij}^{(1)}$ or $e_{ij}^{(2)}$ such that exactly $n - k - 1$ solutions are consistent. The error digit can then be retrieved by the syndrome value from the residue channel that has inconsistency solution. The extended double residue digit error correction algorithm in [16] is based on the concept of modulus projection. The erroneous residue digits can be successfully located if the magnitude of the residue representation after excluding the two erroneous residue digits falls within the legitimate range. The magnitude can be obtained by MRC on the residue representation of each projection. All four algorithms require iterative evaluation of every possible combination of residue digit error locations.

In [19], two error decoding algorithms based on CRT were proposed. They are Unique-Decode and List-Decode algorithms. The Unique-Decode algorithm is used for detecting and correcting a small number of errors. It tries to find two unknown integers y and z , such that $|y \cdot X|_M \equiv z$, where X is the magnitude of the received residue representation and M is the product of all moduli of the RRNS. The computation of y and z involves integer programming with a fixed number of variables [47] and must be carried out recursively until z/y becomes an integer. The List-Decode algorithm is for large number of errors and it is a generalization

of the Unique-Decode algorithm. The List-Decode algorithm searches for a list of integers, $c_0, \dots, c_l$, such that $\left| \sum_{i=0}^l c_i r^i \right|_M = 0$ and outputs all the roots of the integer polynomial $C(x) = \sum_{i=0}^l c_i x^i$, where l represents the total number of moduli. With $2t$ redundant moduli channels, the List-Decode algorithm can correct more than t residue digit errors.

In [17], a multiple residue digit error correction algorithm is reported based on modulo projection concept. Similar to the extended algorithm in [16], this algorithm computes the magnitude of the received residue digits by ignoring t residue channels at each iteration, where t stands for the number of correctable residue digit errors. Instead of using MRC, CRT is adopted for error magnitude evaluation. If the computed magnitude falls within the legitimate range during the iteration process, then the ignored residue channel can be declared as erroneous. The erroneous digit can be retrieved by reverse computing the error values using the modulo projection of that iteration. The maximum number of iterations required in this algorithm is C_n^t , where n is the total number of moduli.

The algorithm in [17] has been extended to reduce the number of iterations by maximum likelihood decoding (MLD) technique. The extended scheme computes the magnitude of t residue digits, where t is the number of correctable residue digits. When the computed magnitude is within the legitimate range, the remaining residue digits are computed by using the base extension method and then compared with the received residue digits. If the Hamming distance between the base-extended residue digits and the received residue digits is less than or

equal to t , the base-extended residue digits are declared to be the error-free residue digits. The number of iterations for this extended scheme is $\lceil C'_n/C_n \rceil$, which is significantly lower than the first method proposed in [17], especially when the number of moduli is large.

In [20], a pseudo-syndrome based double-error correction in RRNS is proposed. The algorithm is a modification of [16] but it does not need iterative steps for error retrieval. In the modified algorithm, CRT is performed on the received residue digits to obtain their base extended residue digits in redundant residue channels. Pseudo syndromes $\tilde{\Delta}_{r,i}$ are then constructed using the received residue digit $\bar{x}_{k+r}$ and base extended residue digit $\tilde{x}_{k+r}$.

$$\tilde{\Delta}_{r,i} = |\tilde{x}_{k+r} - \bar{x}_{k+r} - iM_K|_{m_{k+r}} \quad (2.20)$$

where $i = \{2a, \dots\} \cup \{k-1\}, a = 0, 1, \dots, \left\lfloor \frac{k-1}{2} \right\rfloor$, M_K is the product of all information moduli and $r = 1, 2, 3$ and 4 . For each i value, the obtained pseudo syndromes are grouped into a pseudo syndrome set $(\tilde{\Delta}_{1,i}, \dots, \tilde{\Delta}_{r,i})$. A mapping unit can be constructed based on each pseudo syndrome set to locate possible errors and find their magnitudes. The mapping unit can be implemented as a LUT. It takes the value of each pseudo syndrome set as address and outputs the corresponding double error locations (l_p, l_q) as well as magnitudes (e_p, e_q) , where p and q stand for two possible residue digit error indexes among the information residues. The residue errors can be retrieved by comparing the output from all the mapping units and looking for common non-zero value.

A non-iterative multi-digit residue error detection and correction algorithm was proposed in [31]. In the algorithm, the received residue digits are divided into three groups. Based on these three groups, seven error location categories are defined to cover all the combinations of residue digit errors. The error magnitudes are abstracted into three syndrome values. These syndrome values are then used as addresses to six error LUTs to identify the error location category and retrieve the exact erroneous digits. The error retrieval process can be achieved within a fixed number of computations and table lookup operations regardless of the moduli set and number of correctable residue digit errors.

Table 2.1 summarizes all the above-mentioned multiple residue digit error detection and correction algorithms in terms of the number of correctable residue digit errors, the methods used for detecting and locating errors, whether it has fixed number of computation cycles and whether it requires error LUTs.

Table 2.1 Comparison of Multiple Residue Digit Error Detection and Correction Algorithms

Algorithm	No. of errors correctable	Method		Fixed Latency?	Error LUTs?
		Detect	Locate		
[15]	Single Burst $\leq t$	Syndrome	Syndrome Sets	No	No
[18]	$\leq t$	CRT	Continued Fraction	No	No
[16] #1	1	Syndrome	Consistency Check	No	No
[16] #2	2	Syndrome	Consistency Check	No	No
[16] #3	Single Burst $\leq t$	Syndrome	Consistency Check	No	No
[16] #4	2	MRC	Modulus Projection	No	No
[19] #1	$\frac{(n-k)\log m_1}{\log m_1 + \log m_n}$	CRT	Unique-Decode	No	No
[19] #2	$\leq n - \sqrt{nk}$	CRT	List-Decode	No	No
[17] #1	$\leq t$	CRT	Modulus Projection	No	No
[17] #2	$\leq t$	CRT	Improved Modulus Projection	No	No
[36]	2	Pseudo Syndrome	Syndrome Check	Yes	Yes
[31]	$\leq t$	Syndrome	Syndrome Check	Yes	Yes

From Table 2.1, Algorithms [15] and [16] #3 are only able to correct single burst errors. Random residue digit errors are not detectable or correctable by using these two algorithms. Algorithm [16] #2, [16] #4 and [36] are only able to correct up to two residue digit errors. Therefore they are not suitable for high fault-tolerant applications. The number of correctable residue errors in [19] #1 is dependent on the choice of the smallest and largest moduli m_1 and m_n . Error correction capability specification can be satisfied by careful selection of moduli set. Algorithm [19] #2 can correct more than t residue digit errors with $2t$ redundant moduli. However, both [19] #1 and [19] #2 are not suitable for efficient implementation due to two reasons. Firstly, they both require high number of iterative steps which slows down the circuit performance. Secondly, searching

for unknown integers as required by the algorithms needs intensive arithmetic computations and cannot be easily implemented by simple logic gates or LUTs. Algorithm [18], [17] and [31] are able to correct t residue digit errors with $2t$ redundant residues. However, Algorithm [18] is not suitable for efficient hardware implementation. As mentioned earlier, its massive iteration process restricts the circuit speed and makes it very hard to allocate resources and optimize the design. [17] #1 and [17] #2 are algorithms based on modulus projection concept. They also need iterative computations to obtain the error free output. Both algorithms involve modulo reduction over very large integers. Algorithm [31] stands out among other existing algorithms in terms of efficient implementation. It has high parallelism and requires a fixed number of computations and table lookup operations regardless of the moduli set and error correction capability. The size of syndrome computations is reduced to moderate size and can be further decomposed into smaller size within each modulo channel. Therefore, it can be used as a base of development for an efficient hardware implementation in RRNS fault-tolerant computing. Algorithm [31] will be analyzed in detail in terms of its algorithmic structure, uniqueness conditions and limitations for hardware implementation in the next section.

2.3.3 A Non-Iterative Multi-Digit Residue Error Detection and Correction Algorithm

2.3.3.1. Algorithm Description

The algorithm for multiple digit error correction proposed in [31] uses $2t$ number of redundant moduli to detect up to $2t$ and correct up to t residue digit errors.

After confirming that a received residue representation is in error, the most straightforward method to locate the erroneous residue digit is to perform a full R2B conversion on the received residue representation and use the converted magnitude as the address to a residue digit error LUT. However, the conventional method requires complex hardware implementation and long computation time. The algorithm described in [31] avoids these problems by dividing the received residue digits into three groups: information residue digit group (IR), redundant residue digit group 1 (RR1), and redundant residue digit group 2 (RR2). Their expressions are shown as follows:

Moduli Set: $(m_1, m_2, \dots, m_k, m_{k+1}, \dots, m_{k+t}, m_{k+t+1}, \dots, m_{k+2t})$

Received Residue Digits: $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k, \tilde{x}_{k+1}, \dots, \tilde{x}_{k+t}, \tilde{x}_{k+t+1}, \dots, \tilde{x}_{k+2t})$

IR: $\tilde{X}_K \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k)$

RR1: $\tilde{X}_U \equiv (\tilde{x}_{k+1}, \tilde{x}_{k+2}, \dots, \tilde{x}_{k+t})$

RR2: $\tilde{X}_V \equiv (\tilde{x}_{k+t+1}, \tilde{x}_{k+t+2}, \dots, \tilde{x}_{k+2t})$

The signal dynamic range is further divided into three subranges, M_K , M_U and M_V , where $M_K = \sum_{j=1}^k m_j$ is the product of information moduli, $M_U = \sum_{j=k+1}^{k+t} m_j$ is the product of the first t redundant moduli and $M_V = \sum_{j=k+t+1}^{k+2t} m_j$ is the product of the remaining t redundant moduli. Three syndromes can be constructed based on the following equations:

$$\delta_1 = \left| \tilde{X}_U - \tilde{X}_K \right|_{M_U} \quad (2.21)$$

$$\delta_2 = \left| \tilde{X}_V - \tilde{X}_K \right|_{M_V} \quad (2.22)$$

$$\delta_3 = \left| \tilde{X}_V - \tilde{X}_U \right|_{M_V} \quad (2.23)$$

where $\tilde{X}_K = \left| \tilde{X} \right|_{M_K}$, $\tilde{X}_U = \left| \tilde{X} \right|_{M_U}$, $\tilde{X}_V = \left| \tilde{X} \right|_{M_V}$ and $\tilde{X}$ is the received RRNS representation. $\tilde{X}$ falls in the range $[0, M_N - 1]$, where $M_N = M_K M_U M_V$. Based on the three syndromes, 6 LUTs are constructed to cover all the error combinations in M_K , M_U and M_V . In total, 7 error locations are formed, namely **EL1** ~ **EL7**. By grouping the received residue digits into three distinct groups, any combination of residue digit errors can be categorically identified by one of the seven error location categories defined as follows:

- EL1:** Erroneous residue digit(s) in IR only.
- EL2:** Erroneous residue digit(s) in RR1 only.
- EL3:** Erroneous residue digit(s) in RR2 only.
- EL4:** Erroneous residue digit(s) in IR and RR1.
- EL5:** Erroneous residue digit(s) in IR and RR2.
- EL6:** Erroneous residue digit(s) in RR1 and RR2.
- EL7:** Erroneous residue digit(s) in IR, RR1 and RR2.

Table 2.2 summarized the content of all 6 LUTs, respective look-up syndrome and error locations covered. The error correction procedure can be summarized as follows and the corresponding flowchart is shown in Fig. 2.6.

1. Decode $\tilde{X}_K$, $\tilde{X}_U$ and $\tilde{X}_V$ from the received residue digits.
2. Compute δ_1 , δ_2 and δ_3 by (7), (8) and (9).
3. If $\delta_1 = \delta_2 = \delta_3 = 0$, there is no error. Goto Step 8.

4. If only one of δ_1 , δ_2 and δ_3 is non-zero, declare that there are more than t residue digit errors. Goto Step 8.
5. If $\delta_1 \neq 0, \delta_2 \neq 0$ and $\delta_3 = 0$, the erroneous residue digit(s) is(are) in **EL1**. If $\delta_1 \neq 0, \delta_2 = 0$ and $\delta_3 \neq 0$, the erroneous residue digit(s) is(are) in **EL2**. If $\delta_1 = 0, \delta_2 \neq 0$ and $\delta_3 \neq 0$, the erroneous residue digit(s) is(are) in **EL3**. Retrieve the respective error digit(s) and Goto Step 8.
6. If $\delta_1 \neq 0, \delta_2 \neq 0$ and $\delta_3 \neq 0$, the erroneous residue digit(s) could be in **EL4**, **EL5**, **EL6** or **EL7**. Retrieve the respective error digit(s) and Goto Step 8.
7. If there is none or more than one error location category found in Step 6, declare that there are more than t residue digit errors. Goto Step 8. Otherwise, subtract the error vector of E from the received residue digit(s) to correct the erroneous residue digit(s).
8. End procedure.

Table 2.2. Possible Syndrome Expressions for Error Location Cases **EL1-EL7**

δ	Matched values	Error vectors			Categories
		E_K	E_U	E_V	
δ_1	$v_1 = \left E_U \right _{M_U} - \left E_K \right _{M_K} + \alpha_K M_K \Big _{M_U}$	✓	✓		EL4, EL7
	$v_2 = \left \alpha_K M_K - \left E_K \right _{M_K} \right _{M_U}$	✓			EL1, EL5
	$v_3 = \left E_U \right _{M_U}$		✓		EL2, EL6
δ_2	$v_4 = \left \alpha_K M_K - \left E_K \right _{M_K} \right _{M_V}$	✓			EL1, EL4
	$v_5 = \left E_V \right _{M_V} - \left E_K \right _{M_K} + \alpha_K M_K \Big _{M_V}$	✓		✓	EL5, EL7
	$v_6 = \left E_V \right _{M_V}$			✓	EL3, EL6
δ_3	$v_7 = \left \alpha_U M_U - \left E_U \right _{M_U} \right _{M_V}$		✓		EL2, EL4
	$v_8 = \left E_V \right _{M_V}$			✓	EL3, EL5
	$v_9 = \left E_V \right _{M_V} - \left E_U \right _{M_U} + \alpha_U M_U \Big _{M_V}$		✓	✓	EL6, EL7

To avoid ambiguity, the moduli set should fulfil the following two conditions:

(1) $M_U > M_K$

(2) $M_V > 2S_{EL1}$ where S_{EL1} stands for the total number of error combinations in

case **EL1**.

Fig. 2.6. Flowchart of non-iterative multi-digit error correction algorithm proposed in [31]

2.3.3.2. Error Correction Example

Consider the moduli set $\{7, 9, 11, 13, 17, 19, 23, 29, 31\}$. According to the algorithm, the moduli set is divided into 3 sub-regions: IR $\{7, 9, 11\}$, RR1 $\{13, 17, 19\}$ and RR2 $\{23, 29, 31\}$. Therefore, $M_K = 693$, $M_U = 4199$ and $M_V = 20677$. It can be verified that $M_K < M_U$ and $S_{EL1} = 692$, therefore both conditions are fulfilled. This RRNS system can correct up to $t = 3$ errors with 6 redundant moduli channels. Assume an integer $X = 100 \equiv (2, 1, 1, 9, 15, 5, 8, 13, 7)$ with an injected error vector $E \equiv (1, 0, 0, 1, 0, 0, 0, 2, 0) = 15298532271$. $\tilde{X}_K \equiv (3, 1, 1) = 199$, $\tilde{X}_U \equiv (10, 15, 5) = 746$ and $\tilde{X}_V \equiv (8, 15, 7) = 12934$. The

syndrome values can be computed using Equation (2.21) - (2.23):

$$\delta_2 = |12934 - 199|_{20677} = 12735, \quad \delta_3 = |12934 - 746|_{20677} = 12188.$$

Since all three syndromes are non-zero value, the error location is either **EL4**, **EL5**, **EL6** or **EL7**.

The error magnitude with respect to each region can be computed as: $|E_K|_{M_K} = 99$,

$|E_U|_{M_U} = 646$ and $|E_V|_{M_V} = 12834$. The possible syndrome values are:

$$v_1 = |646 - 99|_{4199} = 547 \text{ for } \alpha_K = 0 \text{ and } |547 + 693|_{4199} = 1240 \text{ for } \alpha_K = 1;$$

$$v_2 = |-99|_{4199} = 4100 \text{ for } \alpha_K = 0 \text{ and } |693 - 99|_{4199} = 594 \text{ for } \alpha_K = 1; \quad v_3 = 646;$$

$$v_4 = |-99|_{20677} = 20578 \text{ for } \alpha_K = 0 \text{ and } |693 - 99|_{20677} = 594 \text{ for } \alpha_K = 1;$$

$$v_5 = |12384 - 99|_{20677} = 12285 \text{ for } \alpha_K = 0 \text{ and } |12285 + 693|_{20677} = 12978 \text{ for}$$

$$\alpha_K = 1; \quad v_6 = 12384; \quad v_7 = |-646|_{20677} = 20031 \text{ for } \alpha_U = 0 \text{ and}$$

$$|4199 - 646|_{20677} = 3553 \text{ for } \alpha_U = 1; \quad v_8 = 12384; \quad v_9 = |12384 - 646|_{20677} = 11738$$

$$\text{for } \alpha_U = 0 \text{ and } |11738 + 4199|_{20677} = 15937 \text{ for } \alpha_U = 1. \text{ These values are the}$$

entries in the constructed LUTs with the address given by the syndrome values.

From the LUTs, we can find the matched entry values to the computed syndrome values using the LUT for **EL7** cases. The retrieved error vector is

$$E = E_{EL7} \equiv (1, 0, 0, 1, 0, 0, 0, 0, 0) \cup (1, 0, 0, 0, 0, 0, 0, 2, 0) = (1, 0, 0, 1, 0, 0, 0, 2, 0).$$

Hence the corrected output is

$$\tilde{X} - E = (3, 1, 1, 10, 15, 5, 8, 15, 7) - (1, 0, 0, 1, 0, 0, 0, 2, 0) =$$

$$(2, 1, 1, 9, 15, 5, 8, 13, 7) \equiv X.$$

2.3.3.3. Algorithm Limitations

Although Algorithm [31] provides appears to be a more promising candidate for hardware implementation, it cannot achieve good hardware-efficiency by direct mapping to FPGA. The limitations for this algorithm can be seen from two aspects. Firstly, the size of syndrome computation requires modulo reduction with moderate size. It increases with information dynamic range M_K . These reduction operations, if implemented directly, would take up large amount of circuit resource and make the circuit runs slower. Second, the retrieving error digit(s) using LUT need to contain all the errors to be corrected, whose size could increase exponentially with the number of correctable errors of the design. Hardware cost will increase proportionally when the information size or number of correctable errors increases.

In the next chapter, a new circuit implementation based on Algorithm [31] will be proposed. The proposed implementation deals with the first limitation by decomposing the moderate-size computations into smaller moduli channels and carefully sharing the common circuitries in syndrome generation as well as error retrieval phase. The second limitation is relaxed by applying adaptive partitioning concept to the information residues. By dividing them into blocks with only two information channels in each block, the large LUTs for error retrieval can be eliminated.

Chapter 3 Proposed Hardware Architectures

for Double and Multiple Residue Digit Error

Correction Algorithms

3.1 Overview

Based on the algorithm reported in [31], the process of locating erroneous residue digits and finding their magnitudes can be implemented merely by LUTs. When the input size is small, such design can be implemented without consuming too much circuit area. However, when the input size increases, the sizes of LUTs increase exponentially and the overall hardware cost becomes very large. In this chapter, a new multiple residue digit error correction scheme for efficient hardware implementation is proposed. The proposed scheme splits the received residue information channels into several information blocks where each block contains only two information residue channels. Then a modified double residue digit error correction circuit developed from [31] is used as the basic error correction circuit to detect and correct residue digit errors for each information block. In the modified double residue digit error correction circuit, large error LUTs are replaced by combinational logic arithmetic circuits. This largely reduces the hardware cost. The syndrome generator has been modified to reduce the hardware cost by replacing the modulo reduction LUTs with binary comparators and sharing common logic blocks among all three syndrome generators. The scheme is implemented in a pipeline structure for higher circuit speed and throughput. The main new contributions of this proposed circuit architecture are: (1) a CRT-based syndrome generator with shared logic, (2) an

arithmetic error retrieval circuit for double residue digit error correction and (3) a multiple residue digit error correction scheme using adaptive information channel partitioning.

3.2 A CRT-based Syndrome Generator with Shared Logic

Consider a received residue representation

$\tilde{X} \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k, \tilde{x}_{k+1}, \dots, \tilde{x}_{k+t}, \tilde{x}_{k+t+1}, \dots, \tilde{x}_{k+r})$ of the moduli set

$(m_1, m_2, \dots, m_k, m_{k+1}, \dots, m_{k+t}, m_{k+t+1}, \dots, m_{k+r})$, where $r = 2t$. The residue

representation can be divided into three groups: IR, RR1 and RR2 of dynamic

ranges $M_K = \prod_{i=1}^k m_i$, $M_U = \prod_{i=k+1}^{k+t} m_i$ and $M_V = \prod_{i=k+t+1}^{k+2t} m_i$, respectively. According

to Equations (2.21), (2.22) and (2.23), the syndrome values can be generated

using CRT followed by modulo reduction with respect to M_K , M_U and M_V . By

taking the modulo m_i operation on each syndrome, the size of the modulo

operations for syndrome computations can be reduced from M_U and M_V to m_i ,

where $i = k+1, k+2, \dots, k+r$. After the modulo reduction, the resulting

syndromes can be expressed as $\delta_j = \left(|\delta_j|_{m_i}, |\delta_j|_{m_{i+1}}, \dots, |\delta_j|_{m_{i+t-1}} \right)$. Base-extension

(BEX) operations reported in [48] are then performed after the modulo reduction

to obtain the generated syndrome values with respect to each modulus channel.

There are three cases of syndrome computation according to Equations (2.21),

(2.22) and (2.23).

3.2.1 Syndrome Case 1: $\delta_1 = \left| \tilde{X}_U - \tilde{X}_K \right|_{M_U}$

The residue representation of the received integer $\tilde{X}$ can be reduced to the information dynamic range M_K by:

$$\begin{aligned} \left| \tilde{X} \right|_{M_K} &= \left| \left| \tilde{X} \right|_M \right|_{M_K} \\ &= \left| \left| \sum_{i=1}^n M_i \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_M \right|_{M_K} \end{aligned} \quad (3.1)$$

Due to **Property 7**, since M_K divides M , the equation can be written as:

$$\left| \tilde{X} \right|_{M_K} = \left| \sum_{i=1}^n M_i \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} \quad (3.2)$$

where $M_i = \prod_{j=1}^n m_j$ ($j \neq i$) and M_i^{-1} is its multiplicative inverse with respect to m_i . Note that for $i > k$, M_i is a multiple of M_K , therefore $\left| M_i \right|_{M_K} = 0$ ($i > k$).

Due to **Property 3**, this equation can be written as

$$\begin{aligned} \left| \tilde{X} \right|_{M_K} &= \left| \sum_{i=1}^k \left| M_{Ki} M_U M_V \right|_{M_K} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} \\ &= \left| \sum_{i=1}^k M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} \\ &= \sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \end{aligned} \quad (3.3)$$

where $M_{Ki} = \prod_{j=1}^k m_j$ ($j \neq i$) and λ_{12} is the modulo reduction factor.

Hence the syndrome δ_1 can be generated by:

$$\begin{aligned} \delta_{1_u} &= \left| \tilde{X}_U - \left(\sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \right) \right|_{m_{k+u}} \\ &= \left| \tilde{x}_{k+u} - \left| \sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \right|_{m_{k+u}} \right|_{m_{k+u}} \end{aligned} \quad (3.4)$$

where $u = 1, 2, \dots, t$.

The implementation of Equation (3.4) can be realized by performing modulo multiplication and addition operations on the received residues $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k)$ to obtain $M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}$. These product terms are then modulo reduced with respect to M_K to obtain the intermediate sum term $\sum_{i=1}^k |M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}|_{M_K}$. The modulo reduction factor λ_{12} ($\lambda_{12} < k$) can be determined by

$$\lambda_{12} = \left\lfloor \frac{\sum_{i=1}^k |M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}|_{M_K}}{M_K} \right\rfloor \quad (3.5)$$

Since the sum term $\sum_{i=1}^k |M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}|_{M_K} < kM_K$, it will be fed into $k - 1$ binary comparators with respect to $M_K, 2M_K, \dots, (k-1) \cdot M_K$ to determine its range. λ_{12} can be determined based on the range of the sum term. For example, if $M_K \leq \sum_{j=1}^k |M_{Kj} |M_U M_V|_{m_j} |M_j^{-1} \tilde{x}_j|_{m_j}|_{M_K} < 2M_K$, then λ_{12} should be 1 so that the result after subtraction of $\lambda_{12} M_K$ falls into the range $[0, M_K - 1]$. The outputs of all the comparators are then combined to form a “thermometer code”, which can be converted into integer format to represent λ_{12} by a simple LUT. The process of obtaining λ_{12} is shown in Fig. 3.1.

Fig. 3.1. Computation of λ_{12} .

3.2.2 Syndrome Case 2: $\delta_2 = \left| \tilde{X}_v - \tilde{X}_K \right|_{M_v}$

The generation of δ_2 is similar to δ_1 except that $\tilde{X}_u$ has to be replaced by $\tilde{X}_v$ and $\text{mod } m_{k+t+v}$ reduction is performed at the end. It can be expressed as

$$\begin{aligned} \delta_{2_v} &= \left| \tilde{X}_v - \left(\sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \right) \right|_{m_{k+t+v}} \\ &= \left| \tilde{x}_{k+t+v} - \left| \sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \right|_{m_{k+t+v}} \right|_{m_{k+t+v}} \end{aligned} \quad (3.6)$$

where $v = 1, 2, \dots, t$. Define $I_{12} = \sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K$,

Equations (3.4) and (3.6) can be rewritten as

$$\delta_{1_u} = \left| \tilde{x}_{k+u} - |I_{12}|_{m_{k+u}} \right|_{m_{k+u}} \quad (3.7)$$

$$\delta_{2_v} = \left| \tilde{x}_{k+t+v} - |I_{12}|_{m_{k+t+v}} \right|_{m_{k+t+v}} \quad (3.8)$$

where $u = 1, 2, \dots, t$ and $v = 1, 2, \dots, t$.

Since I_{12} is the common part for the computations of both δ_{1_u} and δ_{2_v} , in the actual implementation, Equations (3.7) and (3.8) are merged to share the hardware resources. Fig. 3.2 shows the circuitry of I_{12} generation and Fig. 3.3 shows the computation of δ_{1_u} and δ_{2_v} .

Fig. 3.2. Generation of I_{12} .

Fig. 3.3. Computations of δ_{1_u} and δ_{2_v} .

The most expensive parts of the circuit in Fig. 3.2 are the circuit blocks for modulo M_K reduction, which are usually implemented by LUTs. The size of each LUT can be evaluated by $W(\text{width}) \times D(\text{depth})$, where width stands for the word length of the output and depth stands for the number of entries in the LUT. Based on the term $M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}$, the size of each LUT does not exceed $\lceil \log_2 M_K \rceil \times M_K m_k$, where m_k is largest modulus among $(m_1, m_2, \dots, m_k)$. Therefore, in total, k LUTs of size $\lceil \log_2 M_K \rceil \times M_K m_k$ are needed. A closer look at the content of each LUT reveals that for the first M_K entries, the outputs of the LUT are the same as their input addresses. This means that the logic for these entries can be implemented by direct wiring without table lookup. Multi-level table lookup technique suggested in [49] is used to avoid the usage of large LUTs. To minimize the number of modulo addition levels, a partition size of $n_{\text{partition}} = \lceil \log_2 M_K \rceil - 1$ bits is used. In this way, the total length of the term $M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}$ can be divided into two segments: $n_{\text{LSB}} = n_{\text{partition}}$ (LSB) and $n_{\text{MSB}} = \lceil \log_2 (M_K m_k) \rceil - n_{\text{LSB}}$. Since $M_K \square m_k$ when there is more than one residue in the information channels, n_{MSB} is much smaller than n_{LSB} . Modulo reduction is performed on the n_{MSB} partitioned bits of the term $M_{Ki} |M_U M_V|_{m_i} |M_i^{-1} \tilde{x}_i|_{m_i}$ using h LUTs. The size of each LUT is $\lceil \log_2 M_K \rceil \times 2^{\frac{n_{\text{MSB}}}{h}}$ for $h = 1, \dots, n_{\text{MSB}}$. The remaining n_{LSB} bits are implemented by direct wiring, as shown in Fig. 3.4. Note that there is a trade-off between hardware cost and circuit delay with different h values and an optimal h value can be determined for the design according to the primary constraint between area and timing.

3.2.3 Syndrome Case 3: $\delta_3 = \left| \tilde{X}_v - \tilde{X}_U \right|_{M_v}$

The generation of δ_3 can be expressed as

$$\delta_{3_v} = \left| \tilde{x}_{k+t+v} - \left| \sum_{i=k+1}^{k+t} \left| M_{Ui} \left| M_K M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_U} - \lambda_3 M_U \right|_{m_v} \right|_{m_v} \quad (3.9)$$

where $i = k+1, \dots, k+t$, $v = 1, 2, \dots, t$, $M_{Ui} = \prod_{j=k+1}^{k+t} m_j$ ($j \neq i$) and λ_3 is the modulo reduction factor.

It uses the similar circuit structure as in Syndrome Cases 1 and 2 by changing

$$I_{12} = \sum_{i=1}^k \left| M_{Ki} \left| M_U M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_K} - \lambda_{12} M_K \quad \text{to}$$

$$I_3 = \sum_{i=k+1}^{k+t} \left| M_{Ui} \left| M_K M_V \right|_{m_i} \left| M_i^{-1} \tilde{x}_i \right|_{m_i} \right|_{M_U} - \lambda_3 M_U. \text{ Since there is no common part}$$

between Syndrome Case 3 and Syndrome Cases 1 and 2, Syndrome Case 3 needs to be implemented individually without logic sharing.

Fig. 3.4. Implementation of modulo M_K reduction.

3.3 An Arithmetic Error Retrieval Circuit for Double Residue Digit Error Correction

The error retrieval process in Algorithm [31] can be implemented solely by error LUTs. However, the hardware cost will increase exponentially with the increase in the number of information channels k , the size of each information residue channel L and the number of correctable errors t .

From Table 2.2 it is observed that all the error vectors, except for v_1 , v_5 and v_9 , have a one-to-one mapping between the syndromes and the corresponding error values. Therefore, these error digits can be retrieved by either directly mapping from the syndrome values or their arithmetic operations. In Table 3.1, those error vectors have been re-organized for clearer understanding of the implementation

strategies. Note that $\alpha_k = \begin{cases} 0 & \text{if } X + |E_k|_{M_k} < M_k \\ 1 & \text{otherwise} \end{cases}$ and

$\alpha_u = \begin{cases} 0 & \text{if } X + |E_u|_{M_u} < M_u \\ 1 & \text{otherwise} \end{cases}$. Based on the strategies indicated in Table

3.1Error! Reference source not found., the error retrieval circuit can be divided into three cases.

Table 3.1 Re-organized possible error vectors and their respective error correction implementation strategies

Error Vectors	δ	Error Region	Strategy
$v_2 = \left \alpha_K M_K - \left E_K \right _{M_K} \right _{M_U}$	δ_1	E_K	Arithmetic Computation
$v_4 = \left \alpha_K M_K - \left E_K \right _{M_K} \right _{M_V}$	δ_2	E_K	Arithmetic Computation
$v_7 = \left \alpha_U M_U - \left E_U \right _{M_U} \right _{M_V}$	δ_3	E_U	Arithmetic Computation
$v_3 = \left E_U \right _{M_U}$	δ_1	E_U	Direct Mapping
$v_6 = \left E_V \right _{M_V}$	δ_2	E_V	Direct Mapping
$v_8 = \left E_V \right _{M_V}$	δ_3	E_V	Direct Mapping
$v_1 = \left \left E_U \right _{M_U} - \left E_K \right _{M_K} + \alpha_K M_K \right _{M_U}$	δ_1	E_K, E_U	LUT/ROM
$v_5 = \left \left E_V \right _{M_V} - \left E_K \right _{M_K} + \alpha_K M_K \right _{M_V}$	δ_2	E_K, E_V	LUT/ROM
$v_9 = \left \left E_V \right _{M_V} - \left E_U \right _{M_U} + \alpha_U M_U \right _{M_V}$	δ_3	E_U, E_V	LUT/ROM

3.3.1 Error Retrieval Case 1:

For v_2 , v_4 and v_7 , the errors can be retrieved by the similar arithmetic computation strategy. For $\delta_1 = v_2 = \left| \alpha_K M_K - \left| E_K \right|_{M_K} \right|_{M_U}$, since $M_U > M_K$ and they are coprime to each other, we can simplify the computation of the error term by the following calculation.

$$\begin{aligned}
 \left| \left| E_K \right|_{M_K} \right|_{M_U} &= \left| E_K \right|_{M_K} \\
 &= \left| \alpha_K M_K - \delta_1 \right|_{M_U}
 \end{aligned} \tag{3.10}$$

Taking modulo operation with respect to m_i ($i = 1, 2, \dots, k, k+t+1, \dots, k+2t$) on both sides of the equation, we have

$$\begin{aligned}
|E_K|_{m_i} &= \left| \left| \alpha_K M_K - \delta_1 \right|_{M_U} \right|_{M_K|_{m_i}} \\
&= \left| \left| M_U + \alpha_K M_K - \delta_1 \right|_{M_U} \right|_{M_K|_{m_i}} \\
&= \begin{cases} \left| \left| M_U - \delta_1 \right|_{M_K} \right|_{m_i} & \text{if } \alpha_K = 0 \\ \left| \left| M_U + M_K - \delta_1 \right|_{M_U} \right|_{M_K|_{m_i}} & \text{if } \alpha_K = 1 \end{cases} \quad (3.11) \\
&= \begin{cases} \left| \left| M_U - \delta_1 \right|_{M_K} \right|_{m_i} & \text{if } \alpha_K = 0 \\ \left| \left| (1-r_1)M_U + M_K - \delta_1 \right|_{M_K} \right|_{m_i} & \text{if } \alpha_K = 1 \end{cases} \\
&= \left| \left| (1-\alpha_K r_1)M_U - \delta_1 \right|_{M_K} \right|_{m_i}
\end{aligned}$$

where r_1 is the modulo reduction factor of the term $M_U + M_K - \delta_1$ with respect to the modulus M_U .

Since $0 < M_U + M_K - \delta_1 < (M_U + M_K) < 2M_U$, r_1 can have either value 0 or 1, depending on the value of δ_1 . The condition is given as follows:

$$r_1 = \begin{cases} 0 & \text{if } \delta_1 > M_K \\ 1 & \text{otherwise} \end{cases} \quad (3.12)$$

Equation (3.11) can be further simplified according to the range of i as follows:

$$\begin{aligned}
|E_K|_{m_i} &= \begin{cases} \left| (1-\alpha_K r_1) \left| M_U \right|_{m_i} - \left| \delta_1 \right|_{m_i} \right|_{m_i} & \text{for } i = 1, \dots, k \\ \left| (1-\alpha_K r_1) M_U - \delta_1 \right|_{M_K|_{m_i}} & \text{for } i = k+t+1, \dots, k+2t \end{cases} \\
&= \begin{cases} \left| (1-\alpha_K r_1) \left| M_U \right|_{m_i} - \left| \delta_1 \right|_{m_i} \right|_{m_i} & \text{for } i = 1, \dots, k \\ \left| (1-\alpha_K r_1) \left| M_U \right|_{M_K} - \left| \delta_1 \right|_{M_K} \right|_{m_i} & \text{for } i = k+t+1, \dots, k+2t \end{cases} \quad (3.13)
\end{aligned}$$

By applying CRT on the previously computed syndrome terms, we have

$$\delta_1 = \sum_{j=1}^t M_{Uj} \left| M_{Uj}^{-1} \delta_{1j} \right|_{m_{k+j}} - \gamma_1 M_U \quad (3.14)$$

where $M_{Uj} = \frac{M_U}{m_{k+j}}$, M_{Uj}^{-1} is the multiplicative inverse of M_{Uj} modulo m_{k+j} and

γ_1 is the modulo reduction factor of $A_1 = \sum_{j=1}^t M_{Uj} \left| M_{Uj}^{-1} \delta_{1j} \right|_{m_{k+j}}$ with respect to the

modulus M_U . It can be obtained by comparing $\sum_{j=1}^t M_{Uj} \left| M_{Uj}^{-1} \delta_{1j} \right|_{m_{k+j}}$ with

$M_U, 2M_U, \dots, (t-1)M_U$.

The error term can be finally expressed by the equation below:

$$|E_K|_{m_i} = \begin{cases} \left| \underbrace{(1 - \alpha_K r_1) |M_U|_{m_i}}_{\text{II}} - \underbrace{|A_1|_{m_i} - |\gamma_1 M_U|_{m_i}}_{\text{I}} \right|_{m_i} & i = 1, 2, \dots, k \\ \left| \underbrace{(1 - \alpha_K r_1) |M_U|_{M_K}}_{\text{IV}} - \underbrace{|A_1|_{M_K} - |\gamma_1 M_U|_{M_K}}_{\text{III}} \right|_{M_K} \Big|_{m_i} & i = k + t + 1, \dots, k + 2t \end{cases} \quad (3.15)$$

In addition, the error magnitude E_K can be converted into residue representation for the moduli set $\{m_{k+t+1}, m_{k+t+2}, \dots, m_{k+2t}\}$. This is used for verifying the correctness of the computed error digits later.

The implementation of Equation (3.15) can be divided into four parts. Part I and Part III can be realized by a modified R2B conversion for the previously generated δ_{1j} values based on Equation (3.14). Its circuit structure is shown in Fig. 3.5. In Fig. 3.5, the mod M_K block can be implemented using similar structure shown in Fig. 3.4. γ_1 computation block is implemented by the same

structure as demonstrated in Fig. 3.1. The hardware cost for this block can be optimized by carefully selecting the moduli set such that the difference between M_U and M_K is minimized while fulfilling the condition $M_U > M_K$. The term $|\gamma_1|_{M_U}|_{m_i}|_{m_j}$ can be generated by a modulo m multiplication on the operands γ_1 and $|M_U|_m$, where $m \in \{m_1, \dots, m_k, M_K\}$. As shown in Fig. 3.6, r_1 can be generated once γ_1 is available. The subtractor can be implemented using the parallel prefix tree structure for fast operation.

Fig. 3.5. Modified R2B conversion for Part I and Part III of Equation (3.15).

Fig. 3.6. Generation of r_1 .

Part II and Part IV can be implemented using simple MUXs and the overall circuit structure of Equation (3.15) is shown in Fig. 3.7.

Fig. 3.7. Overall circuit architecture for implementation of Equation (3.15).

For $\delta_2 = v_4 = \left| \alpha_K M_K - |E_K|_{M_K} \right|_{M_V}$ and $\delta_3 = v_7 = \left| \alpha_U M_U - |E_U|_{M_U} \right|_{M_V}$, since the second uniqueness criterion $M_V > 2S_{EL1}$ cannot guarantee that the term $|E_K|_{M_K}$ or $|E_U|_{M_U}$ can be implemented without requiring complex circuitry, one more condition of $M_V > M_U > M_K$ is added. The final conditions for the simplified

error correction circuit can be summarized as: (1) $M_V > M_U > M_K$ and (2)

$M_V > 2S_{EL1}$. Under these conditions, the error terms can be similarly retrieved by:

$$|E_K|_{m_i} = \begin{cases} \left| (1 - \alpha_K r_2) |M_V|_{m_i} - |A_2|_{m_i} - |\gamma_2 M_V|_{m_i} \right|_{m_i} & i = 1, 2, \dots, k \\ \left| (1 - \alpha_K r_2) |M_V|_{M_K} - |A_2|_{M_K} - |\gamma_2 M_V|_{M_K} \right|_{M_K} & i = k+1, \dots, k+t \end{cases} \quad (3.16)$$

$$|E_U|_{m_i} = \left| (1 - \alpha_U r_3) |M_V|_{m_i} - |A_3|_{m_i} - |\gamma_3 M_V|_{m_i} \right|_{m_i} \quad i = k+1, k+2, \dots, k+t \quad (3.17)$$

where γ_2 and γ_3 are the modulo reduction factors for the terms

$$A_2 = \sum_{j=1}^t M_{Vj} |M_{Vj}^{-1} \delta_{2j}|_{m_{k+t+j}} \quad \text{and} \quad A_3 = \sum_{j=1}^t M_{Vj} |M_{Vj}^{-1} \delta_{3j}|_{m_{k+t+j}}, \quad \text{respectively.}$$

$M_{Vj} = \frac{M_V}{m_{k+t+j}}$ and M_{Vj}^{-1} is the multiplicative inverse of M_{Vj} with respect to

$$m_{k+t+j} \quad . \quad \alpha_K = \begin{cases} 0 & \text{if } X + |E_K|_{M_K} < M_K \\ 1 & \text{otherwise} \end{cases}, \quad \alpha_U = \begin{cases} 0 & \text{if } X + |E_U|_{M_U} < M_U \\ 1 & \text{otherwise} \end{cases},$$

$$r_2 = \begin{cases} 0 & \text{if } \delta_2 > M_K \\ 1 & \text{otherwise} \end{cases} \quad \text{and} \quad r_3 = \begin{cases} 0 & \text{if } \delta_3 > M_U \\ 1 & \text{otherwise} \end{cases}.$$

The circuit architectures for Equation (3.16) and (3.17) are the same as that of Equation (3.15), which needs no further elaboration.

3.3.2 Error Retrieval Case 2:

For v_3 , v_6 and v_8 , since their error vectors are the same as the corresponding syndromes, the error digits based on these vectors can be retrieved from the direct mapping to the syndromes for each modulus channel using

$$|E_U|_{m_i} = |\delta_1|_{m_i} = \delta_{1_i} \quad \text{for } i = k+1, k+2, \dots, k+t \quad (3.18)$$

$$|E_V|_{m_i} = |\delta_2|_{m_i} = \delta_{2_i} \quad \text{for } i = k+t+1, k+t+2, \dots, k+2t \quad (3.19)$$

$$|E_V|_{m_i} = |\delta_3|_{m_i} = \delta_{3_i} \quad \text{for } i = k+t+1, k+t+2, \dots, k+2t \quad (3.20)$$

Based on Equations (3.15) to (3.20), error retrieval circuits of location categories **EL1** to **EL6** and their respective verification circuits can be constructed as follows.

From Table 3.1, error digit $|E_K|_{m_i}$ for **EL1** can be obtained from either $\delta_1 = v_2 = |\alpha_K M_K - |E_K|_{M_K}|_{M_U}$ or $\delta_2 = v_4 = |\alpha_K M_K - |E_K|_{M_K}|_{M_V}$. The outputs from Equation (3.15) are used as retrieved error digits. The results obtained from Equation (3.16) can be used to verify their correctness. As for **EL2**, error digit can be computed by $\delta_3 = v_7 = |\alpha_U M_U - |E_U|_{M_U}|_{M_V}$ or $\delta_1 = v_3 = |E_U|_{M_U}$. The results from Equation (3.17) are used as retrieved error digits and the results of $|\delta_1|_{m_i}$ for $i = k+1, \dots, k+t$ are used as checkers. For **EL3**, the errors can be retrieved by $|\delta_2|_{m_i}$ for $i = k+t+1, \dots, k+2t$ or $|\delta_3|_{m_i}$ for $i = k+t+1, \dots, k+2t$. Therefore, $|\delta_2|_{m_i}$ is used for the error retrieval and $|\delta_3|_{m_i}$ for checking of the correctness.

A ‘‘Comparison Unit’’ is used for each error location category to check whether two retrieved results match. The matched result is output. Since the error retrieval circuits may correct more than t residue errors at a time, a Hamming distance check mentioned in [17] is required after each of the error retrieval block. The Hamming distance check module can be implemented by a very small LUT,

which only consists of a few entries. A “valid” signal will be generated after Hamming check to indicate whether the output from this retrieval circuit can be treated as the final retrieved error digits. The final implementation of **EL1 – EL3** is illustrated in Fig. 3.8.

Fig. 3.8. Error retrieval circuits for **EL1 - EL3**.

In location category **EL4**, residue digit errors occur in both IR and RR1 groups.

Based on Table 3.1, error digits in IR and RR1 groups can be computed from

$$\delta_2 = v_4 = \left| \alpha_K M_K - |E_K|_{M_K} \right|_{M_V} \quad \text{and} \quad \delta_3 = v_7 = \left| \alpha_U M_U - |E_U|_{M_U} \right|_{M_V}, \quad \text{respectively.}$$

As for the correctness check, $\delta_1 = v_1 = \left| |E_U|_{M_U} - |E_K|_{M_K} + \alpha_K M_K \right|_{M_U}$ is used by

substituting $|E_U|_{M_U}$ and $|E_K|_{M_K}$ with the retrieved error digits. Note that since the

output of δ_1 is represented by $|\delta_1|_{m_i}$ for $i = k+1, \dots, k+t$, $|E_K|_{M_K}$ need to be

represented by $\left| |E_K|_{M_K} \right|_{m_i}$ for $i = k+1, \dots, k+t$ as well. This can be realized by

performing BEX operations on $\delta_2 = v_4 = \left| \alpha_K M_K - |E_K|_{M_K} \right|_{M_V}$. The process of

obtaining $\left|E_K\right|_{M_K}\left|_{m_i}\right|$ for $i = k + 1, \dots, k + t$ is shown in Fig. 3.7. The retrieval circuit for **EL4** is shown in Fig. 3.9.

Fig. 3.9. Error retrieval circuit for **EL4**.

In error location category **EL5**, the error digits in IR and RR2 groups can be retrieved from $\delta_1 = v_2 = \left|\alpha_K M_K - \left|E_K\right|_{M_K}\right|_{M_U}$ and $\delta_3 = v_8 = \left|E_V\right|_{M_V}$, respectively, as shown in Fig. 3.10. The correctness of the retrieved outputs can be checked using $\delta_2 = v_5 = \left|\left|E_V\right|_{M_V} - \left|E_K\right|_{M_K} + \alpha_K M_K\right|_{M_V}$ by substituting $\left|E_V\right|_{M_V}$ and $\left|E_K\right|_{M_K}$ with the outputs from $\delta_1 = v_2 = \left|\alpha_K M_K - \left|E_K\right|_{M_K}\right|_{M_U}$ and $\delta_3 = v_8 = \left|E_V\right|_{M_V}$. Similar to **EL4**, in **EL5**, BEX operations are also required for $\delta_1 = v_2 = \left|\alpha_K M_K - \left|E_K\right|_{M_K}\right|_{M_U}$ to obtain $\left|E_K\right|_{M_K}\left|_{m_i}\right|$ for $i = k + t + 1, \dots, k + 2t$.

Fig. 3.10. Error retrieval circuit for **EL5**.

As for the error location category **EL6**, the error digits in RR1 and RR2 groups can be obtained using $\delta_1 = v_3 = |E_U|_{M_U}$ and $\delta_2 = v_6 = |E_V|_{M_V}$. The checking process can be realized based on $\delta_3 = v_9 = ||E_V|_{M_V} - |E_U|_{M_U} + \alpha_U M_U|_{M_V}$. $\delta_1 = v_3 = |E_U|_{M_U}$ needs to be represented by $|\delta_1|_{M_V}|_{m_i}$ for $i = k+t+1, \dots, k+2t$. Based on the condition $M_V > M_U > M_K$, $|\delta_1|_{M_V}|_{m_i}$ can be rewritten as $|\delta_1|_{m_i}$ for $i = k+t+1, \dots, k+2t$. The syndromes are computed by Equations (2.19), (2.20) and (2.21). By subtracting Equation (2.20) by Equation (2.21), we have

$$\begin{aligned}
 |\delta_2 - \delta_3|_{M_V} &= |\tilde{X}_V - \tilde{X}_K - (\tilde{X}_V - \tilde{X}_U)|_{M_V} \\
 &= |\tilde{X}_U - \tilde{X}_K|_{M_V}
 \end{aligned} \tag{3.21}$$

Since $|\tilde{X}_U - \tilde{X}_K|_{M_U} < M_U < M_V$, (2.19) can be rewritten as

$$\begin{aligned}
|\delta_1|_{M_V} &= \left| \tilde{X}_U - \tilde{X}_K \right|_{M_U} \Big|_{M_V} \\
&= \left| \tilde{X}_U - \tilde{X}_K + \eta_1 M_U \right|_{M_V}
\end{aligned} \tag{3.22}$$

where $\eta_1 = \begin{cases} 1 & \tilde{X}_U < \tilde{X}_K \\ 0 & \text{otherwise} \end{cases}$, which can be obtained by comparing the previously

computed results I_{12} and I_3 in the syndrome generators. Taking modulo m_i on both sides of the equation for $i = k + t + 1, \dots, k + 2t$, we have

$$\left| \delta_1 \right|_{M_V} \Big|_{m_i} = \left| \delta_2 \right|_{m_i} - \left| \delta_3 \right|_{m_i} + \eta_1 \left| M_U \right|_{m_i} \Big|_{m_i} \quad \text{for } i = k + t + 1, \dots, k + 2t \tag{3.23}$$

With the previously computed values of $|\delta_2|_{m_i}$ and $|\delta_3|_{m_i}$, Equation (3.23) can be implemented by two modulo m_i adders as shown in Fig. 3.11. The implementation of **EL6** is shown in Fig. 3.12.

Fig. 3.11. Generation of $\left| \delta_1 \right|_{M_V} \Big|_{m_i}$ for $i = k + t + 1, \dots, k + 2t$.

3.3.3 Error Retrieval Case 3:

For v_1 , v_5 and v_9 , since each error vector contains error across two different regions, it is hard to use memoryless arithmetic circuit or direct mapping to retrieve the error. Therefore, the error retrieval circuits of these three vectors are

implemented by LUTs. Since Error Retrieval Cases 1 and 2 have already taken care of all the error combinations in **EL1** to **EL6**, the LUTs only contains error combinations for **EL7**, which reduces the size of these LUTs. The top level of the modified error correction circuit is depicted in Fig. 3.13.

Fig. 3.12. Error retrieval circuit for **EL6**.

The multiplexing between Error Retrieval Cases 1 and 2 (PCOM) and Case 3 (PLUT) is determined by a “valid signal” (**valid_{PLUT}**) from the LUT block, which determines if the error matches the LUT entry. An output signal named “too_many_error” is used to indicate if there are more than t residue digit errors detected, which is generated based on **valid_{PCOM}** and **valid_{PLUT}** signals. The final retrieved error digits $(E_K, E_U, E_V)_{OUTPUT}$ are modulo subtracted from the original input residue digits $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k, \tilde{x}_{k+t}, \dots, \tilde{x}_{k+2t})$ with respect to the moduli set $(m_1, m_2, \dots, m_k, m_{k+t}, \dots, m_{k+2t})$ to obtain the final corrected outputs.

The top level of the proposed error correction circuit can be implemented in a pipelined architecture by adding appropriately clocked pipeline registers between

the syndrome generators and error retrieval circuits. In this way, the circuit can be operated at a faster speed and higher throughput, as shown in Fig. 3.13.

Fig. 3.13. Top level design of error retrieval circuit ($t > 2$).

As for double residue digit error correction, it can be treated as a special case of Fig. 3.13. When the number of correctable errors, $t = 2$, the PLUT part for the retrieval of **EL7** error digits can be removed from the circuit since there are only two errors to be corrected and it is impossible for two residue digit errors to appear in all three groups. Fig. 3.14 shows the modified circuit for this special case.

Fig. 3.14. Proposed top-level circuit for double residue digit error correction ($t = 2$).

3.4 A Multiple Residue Digit Error Correction Scheme by Adaptive Information Channel Partitioning

3.4.1 Concept and Implementation

The proposed arithmetic error retrieval circuit in Fig. 3.13 replaces LUTs for **EL1 – EL6** with arithmetic logic. However, the size of the remaining LUTs for **EL7** still grows significantly with the increase in the number of information channels k , size of each information channel L and number of correctable errors t . With the increase of t , the number of combinations of errors to be covered in the modified LUT becomes impractically large. Table 3.2 gives a brief overview on the type of error combinations needed in the LUT for t ranges from 2 to 5. In this table, the number of errors in each group (IR, RR1 and RR2) is expressed as

a 3-digit integer. For example, if there is one error in IR group, 1 error in RR1 group and 2 errors in RR2 group, the integer value will be “112”.

Table 3.2. Error types for different number of correctable errors (t)

t	# of error types in LUT	Error types should be included in LUT (# of errors occur in residue group IR, RR1 and RR2)
2	0	None
3	1	111
4	4	111, 211, 121, 112
5	10	111, 211, 121, 112, 311, 131, 113, 221, 122, 212

From Table 3.2, when $t = 2$, there is no error case to be included in the PLUT block. This is because all the combinations of errors can be retrieved by the arithmetic operations or direct mapping from the three generated syndromes using the proposed arithmetic error retrieval circuit (PCOM block). Therefore, if only double residue digit error correction circuits in Fig. 3.14 are used in a multiple residue digit error correction system, the LUTs for **EL7** error retrieval can be eliminated to greatly reduce the hardware cost and improve the circuit performance.

Let the received residue digits be $\tilde{X} \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k, \tilde{x}_{k+1}, \tilde{x}_{k+2}, \dots, \tilde{x}_{k+r})$ with k information residue channels and r redundant residue channels. The number of correctable errors is $t = \left\lfloor \frac{r}{2} \right\rfloor$. The residue representation for the information residue digits is $\tilde{X}_K \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k)$. Before choosing the redundant moduli, an adaptive partition can be adopted to divide k information channels into $p = \left\lfloor \frac{k}{2} \right\rfloor$ information blocks, where each block only consists of two information channels. Each piece of data containing a smaller number of information channels will then

be treated as a “new” information set for error correction. Every combination of the partitioned information moduli set and its corresponding allocated redundant moduli set must fulfil the first and second uniqueness criteria: 1) $M_{V(i,j)} > M_{U(i,j)} > M_{K(i,j)}$ and 2) $M_{V(i,j)} > 2S_{EL1(i,j)}$, where $M_{V(i,j)}$, $M_{U(i,j)}$, $M_{K(i,j)}$ and $S_{EL1(i,j)}$ denote the j^{th} ($j=1,2,...,t-1$ or $1,2,...,t$) combination of M_V , M_U , M_K and S_{EL1} , respectively from the i^{th} ($i=1,2,...,p$) partitioned information block. The partition of the information residue channels and allocation of the redundant residue channels are demonstrated in Fig. 3.15. For example, when k is even, the information residue channels are partitioned into $p = \frac{k}{2}$ blocks. The residues of the first partitioned information block is $(\tilde{x}_1, \tilde{x}_2)$ with respective moduli set $\{m_1, m_2\}$. For this information block, redundant residues $(\tilde{x}_{k+1}, \tilde{x}_{k+2}, \tilde{x}_{k+3}, \tilde{x}_{k+4})$, $(\tilde{x}_{k+5}, \tilde{x}_{k+6}, \tilde{x}_{k+7}, \tilde{x}_{k+8})$, ..., and $(\tilde{x}_{k+r-3}, \tilde{x}_{k+r-2}, \tilde{x}_{k+r-1}, \tilde{x}_{k+r})$ for the moduli sets $\{m_{k+1}, m_{k+2}, m_{k+3}, m_{k+4}\}$, $\{m_{k+5}, m_{k+6}, m_{k+7}, m_{k+8}\}$, ..., and $\{m_{k+r-3}, m_{k+r-2}, m_{k+r-1}, m_{k+r}\}$ are used for designing the double residue digit error correction circuits. All the combinations $\{m_1, m_2, m_{k+1}, m_{k+2}, m_{k+3}, m_{k+4}\}$, $\{m_1, m_2, m_{k+5}, m_{k+6}, m_{k+7}, m_{k+8}\}$, ..., and $\{m_1, m_2, m_{k+r-3}, m_{k+r-2}, m_{k+r-1}, m_{k+r}\}$ must fulfil both uniqueness criteria 1) $M_{V(1,j)} > M_{U(1,j)} > M_{K(1,j)}$ and 2) $M_{V(1,j)} > 2S_{EL1(1,j)}$, where $M_{V(1,j)}$, $M_{U(1,j)}$, $M_{K(1,j)}$ and $S_{EL1(1,j)}$ denote for the j^{th} $\left(j=1,2,...,\left\lfloor \frac{r}{4} \right\rfloor\right)$ combination of M_V , M_U , M_K and S_{EL1} , respectively, from the first partitioned information block.

Fig. 3.15 Adaptive partitioning of information residues and allocation of their respective redundant residues.

The implementation of the proposed scheme can be divided into two main categories: (1) $k > 4$ and (2) $k \leq 4$. The main difference between these two cases is majority voting is adopted for higher fault recovery coverage in (1) whereas no voting mechanism is used in (2) due to insufficient number of double residue digit error correction circuits.

3.4.1.1. Category 1: $\left\lfloor \frac{r}{4} \right\rfloor \geq 3$

If $\left\lfloor \frac{r}{4} \right\rfloor \geq 3$, there will be $p = \left\lceil \frac{k}{2} \right\rceil$ information blocks. Each partitioned information block needs p' double residue digit error correction circuit blocks and a voter to retrieve its residue digit errors. The implementation can be further divided into two sub-cases depending on whether $\left\lfloor \frac{r}{4} \right\rfloor$ is odd or even:

Sub-case 1: $\left\lfloor \frac{r}{4} \right\rfloor$ is odd

If $\left\lfloor \frac{r}{4} \right\rfloor$ is odd, then $p' = \left\lfloor \frac{r}{4} \right\rfloor$. Each partitioned information block needs $\left\lfloor \frac{r}{4} \right\rfloor$ double residue digit error correction circuits for error retrieval. A Selection Logic block (SEL) is used to select the final retrieved information residue error digits through majority voting. The circuitry of SEL is shown in Fig. 3.16. Pipeline registers are added between the double residue digit error correction blocks and SELs for high throughput and fast speed. The result $E = (e_1, e_2, \dots, e_k)$ is obtained by concatenating the outputs from each SEL. Upon obtaining the corrected information residue digits $\bar{X}_K \equiv (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_k)$, the corrected redundant residue digits can be obtained through BEX operations on the corrected information residues $(\bar{x}_1, \bar{x}_2)$, $(\bar{x}_3, \bar{x}_4)$, ..., $(\bar{x}_{k-1}, \bar{x}_k)$. Each BEX operation can be implemented by the modified R2B module as shown in Fig. 3.5. The hardware cost of the overall system can be significantly reduced by controlling the maximum allowable word length L of each information residue channel. Fig.

3.17 shows how this adaptive input partitioning can be performed for this sub-case.

Fig. 3.16. SEL and BEX circuitry for partitioned information block 1.

Fig. 3.17. Adaptive input partitioning scheme for multiple residue digit error

correction ($\left\lfloor \frac{r}{4} \right\rfloor \geq 3$ and $\left\lfloor \frac{r}{4} \right\rfloor$ is odd).

Sub-case 2: $\left\lfloor \frac{r}{4} \right\rfloor$ is even

If $\left\lfloor \frac{r}{4} \right\rfloor$ is even, then $p' = \left\lfloor \frac{r}{4} \right\rfloor - 1$. The circuitry of this sub-case is the same as that of Fig. 3.17 except that the number of double residue digit error correction circuits for each partitioned information block becomes $\left\lfloor \frac{r}{4} \right\rfloor - 1$ instead of $\left\lfloor \frac{r}{4} \right\rfloor$. Since there are altogether $\left\lfloor \frac{r}{4} \right\rfloor$ redundant residue groups, for each partitioned information block, only $\left\lfloor \frac{r}{4} \right\rfloor - 1$ redundant residue groups are used. In this case, $\left\lfloor \frac{r}{4} \right\rfloor - 1$ groups are randomly selected from $\left\lfloor \frac{r}{4} \right\rfloor$ redundant residue groups for each partitioned information block. For example, given that the original order of the redundant residue groups is 1, 2, 3, 4, 5, 6, the randomized order of the redundant residue groups is denoted by $s_1, s_2, \dots, s_{p'}$. Then for the i^{th} information block, the first $\left\lfloor \frac{r}{4} \right\rfloor - 1$ redundant residue groups from the randomized groups (e.g., if $s_1, s_2, \dots, s_{p'} = 2, 3, 5, 1, 4, 6$, then the first 5 redundant residue groups have indices 2, 3, 5, 1 and 4) are used for designing the respective double residue digit error correction circuits, as shown in Fig. 3.18. The randomization process is needed for every partitioned information block for determining its corresponding redundant residue groups.

Fig. 3.18. Randomization and allocation of redundant residue groups of the i^{th}

partitioned information block in adaptive partitioning scheme ($\left\lfloor \frac{r}{4} \right\rfloor \geq 3$ and

$\left\lfloor \frac{r}{4} \right\rfloor$ is even).

3.4.1.2. Category 2: $\left\lfloor \frac{r}{4} \right\rfloor < 3$

If $\left\lfloor \frac{r}{4} \right\rfloor < 3$, SELs are no longer needed. For each partitioned information block,

only one double residue digit error correction circuit is used for the error retrieval.

Therefore, there will be only $p = \left\lceil \frac{k}{2} \right\rceil$ information blocks and p double residue

digit error correction circuits. The circuit structure in Fig. 3. 17 can be simplified

by removing the SELs, BEX blocks and the pipeline registers between double

residue digit error correction circuits and SELs. The simplified circuit is shown

in Fig. 3.19.

Fig. 3.19. Simplified adaptive input partitioning scheme for multiple residue digit error correction ($\left\lfloor \frac{r}{4} \right\rfloor < 3$)

3.4.2 Advantages and Limitations

Three main advantages can be derived from this adaptive partitioning scheme. Firstly, by dividing multiple residue digit error ($t > 2$) correction problem into double residue digit error subproblems, LUTs for error retrieval can be eliminated to drastically reduce the hardware cost. Secondly, the partitioning further reduces the redundant space because the divided blocks are independent of each other. The redundant moduli only depend on the partitioned input channels. For example, with five information moduli $\{m_1, m_2, m_3, m_4, m_5\}$, without adaptive partitioning, the redundant moduli, M_U and M_V , must fulfil conditions (1) $M_U > M_K$ and (2) $M_V > 2S_{ELI}$, where $M_K = \prod_{i=1}^5 m_i$. With adaptive partitioning, the modified M_K will be reduced to $M_{K(1,j)} = m_1 m_2$,

$M_{K(2,j)} = m_3 m_4$ and $M_{K(3,j)} = m_5 m_1$. It is evident that the resulting $M_{U(i,j)}$ and

$M_{V_{(i,j)}}$ have been reduced significantly compared with the original M_U and M_V . Although the additional circuit blocks such as BEX blocks and SELs are added to the system, the circuit complexity will not increase much since each BEX operation is based on a much smaller dynamic range like $M_{K_{(1,j)}}$, $M_{K_{(2,j)}}$ and $M_{K_{(3,j)}}$ instead of M_K . The circuit complexity for the forward conversion as well as the residue arithmetic operations outside the error correction module (ECM) can also be reduced due to smaller redundant residue channels. More reduction can be gained when the information channels are made up of many arbitrary moduli. Lastly, since the computations can be fully implemented with combinational logic, the proposed circuit can also take advantage of the hardware-friendly special moduli set such as $\{2^n, 2^n + 1, 2^n - 1\}$. Designs that rely on large error retrieval LUTs do not have such advantage since the size of the LUT will grow with the total number of error combinations. This is very useful for applications such as DSP and cryptography since they usually make use of fewer special moduli than higher number of smaller arbitrary moduli for large information size.

As for the limitation, the proposed scheme may not always be able to correct up to t residue digit errors. There are a few error cases that the scheme is unable to detect or correct. This is because the system is composed from double residue digit error correction circuits. When there are ≥ 5 erroneous residues occur in any of these circuits, these residue digit errors are undetectable. For the same reason, if there are ≥ 3 erroneous residues occur in any of these circuits, these erroneous residue digits are uncorrectable. This limitation can be relaxed by applying the

majority voting on the retrieved information residue error digits. The chance of successfully detecting and correcting these errors will increase. This will be further analyzed in the next chapter.

3.4.3 Error Correction Examples

Let the information moduli set be $\{m_1, m_2, m_3, m_4, m_5, m_6\} = \{13, 17, 19, 23, 29, 31\}$.

Based on the proposed scheme, it will be divided into three information blocks:

(1) $\{m_1, m_2\} = \{13, 17\}$, (2) $\{m_3, m_4\} = \{19, 23\}$ and (3) $\{m_5, m_6\} = \{29, 31\}$. The

dynamic range of each partitioned information moduli block is computed as

$M_{K_{(1,j)}} = 13 \times 17 = 221$, $M_{K_{(2,j)}} = 19 \times 23 = 437$ and $M_{K_{(3,j)}} = 29 \times 31 = 899$. All the

redundant moduli to be allocated to each partitioned information block should be

chosen to fulfil the uniqueness conditions in Section 3.2.3. Therefore, the

redundant moduli can be selected as $\{47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,$

$101\}$. The original integer is assumed to be

$$X = 10000 \equiv \begin{pmatrix} x_1, x_2, x_3, x_4, x_5, x_6, \\ x_{r1}, x_{r2}, x_{r3}, x_{r4}, x_{r5}, x_{r6}, \\ x_{r7}, x_{r8}, x_{r9}, x_{r10}, x_{r11}, x_{r12} \end{pmatrix} \equiv \begin{pmatrix} 3, 4, 6, 18, 24, 18, \\ 36, 36, 29, 57, 17, 71, \\ 72, 46, 40, 32, 9, 1 \end{pmatrix}.$$

Suppose there is an injected error vector $E_1 \equiv \begin{pmatrix} e_1, e_2, e_3, e_4, e_5, e_6, \\ e_{r1}, e_{r2}, e_{r3}, e_{r4}, e_{r5}, e_{r6}, \\ e_{r7}, e_{r8}, e_{r9}, e_{r10}, e_{r11}, e_{r12} \end{pmatrix}$

$\equiv \begin{pmatrix} 0, 1, 2, 0, 6, 0, \\ 3, 0, 0, 0, 0, 0, \\ 0, 0, 0, 2, 1, 0 \end{pmatrix}$. The received residue representation is

$$\tilde{X} \equiv \begin{pmatrix} \tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6, \\ \tilde{x}_{r1}, \tilde{x}_{r2}, \tilde{x}_{r3}, \tilde{x}_{r4}, \tilde{x}_{r5}, \tilde{x}_{r6}, \\ \tilde{x}_{r7}, \tilde{x}_{r8}, \tilde{x}_{r9}, \tilde{x}_{r10}, \tilde{x}_{r11}, \tilde{x}_{r12} \end{pmatrix} \equiv \begin{pmatrix} 3, 5, 8, 18, 1, 18, \\ 39, 36, 29, 57, 17, 71, \\ 72, 46, 40, 34, 10, 1 \end{pmatrix}. \text{ According to the}$$

proposed scheme, the received information residue digits are partitioned into three blocks: (1) $(\tilde{x}_1, \tilde{x}_2) = (3, 5)$, (2) $(\tilde{x}_3, \tilde{x}_4) = (8, 18)$ and (3) $(\tilde{x}_5, \tilde{x}_6) = (1, 18)$.

These three blocks are then treated as new input information channels for the proposed double residue digit error correction circuits. For each partitioned information block, its residue error digits are retrieved by a majority voting result

based on the retrieved error digits from $p' = \left\lfloor \frac{r}{4} \right\rfloor = 3$ double residue digit error

correction circuits with redundant moduli $\{47, 53, 59, 61\}$, $\{67, 71, 73, 79\}$ and $\{83, 89, 97, 101\}$, respectively. The retrieved error digits for all partitioned information blocks can be summarized in Table 3.3.

Table 3.3. Retrieved error digits for all partitioned information blocks and their majority voting results (correctable case).

Partitioned Information Block	Double Residue Digit Error Correction 1	Double Residue Digit Error Correction 2	Double Residue Digit Error Correction 3	Majority Voting Result
$(\tilde{x}_1, \tilde{x}_2) = (3, 5)$	$(e_1, e_2, e_{r1}, e_{r2}, e_{r3}, e_{r4})$ $\equiv (0, 1, 3, 0, 0, 0)$	$(e_1, e_2, e_{r5}, e_{r6}, e_{r7}, e_{r8})$ $\equiv (0, 1, 0, 0, 0, 0)$	too_many_error = 1	$(e_1, e_2) \equiv (0, 1)$
$(\tilde{x}_3, \tilde{x}_4) = (8, 18)$	$(e_3, e_4, e_{r1}, e_{r2}, e_{r3}, e_{r4})$ $\equiv (2, 0, 3, 0, 0, 0)$	$(e_3, e_4, e_{r5}, e_{r6}, e_{r7}, e_{r8})$ $\equiv (2, 0, 0, 0, 0, 0)$	too_many_error = 1	$(e_3, e_4) \equiv (2, 0)$
$(\tilde{x}_5, \tilde{x}_6) = (1, 18)$	$(e_5, e_6, e_{r1}, e_{r2}, e_{r3}, e_{r4})$ $\equiv (6, 0, 3, 0, 0, 0)$	$(e_5, e_6, e_{r5}, e_{r6}, e_{r7}, e_{r8})$ $\equiv (6, 0, 0, 0, 0, 0)$	too_many_error = 1	$(e_5, e_6) \equiv (6, 0)$

By concatenating the retrieved information residue digit errors and applying the BEX operations to the corrected information residue digits, we can obtain the corrected redundant residue digits as

$X_{correct} \equiv (3, 4, 6, 18, 24, 18, 36, 36, 29, 57, 17, 71, 72, 46, 40, 32, 9, 1) = 10000$, which is the original value of X .

Consider another error vector $E_2 \equiv \begin{pmatrix} 1, 1, 2, 0, 6, 0, \\ 3, 0, 0, 0, 1, 0, \\ 0, 0, 0, 0, 1, 0 \end{pmatrix}$ injected into the same integer

input residue representation of X . Similarly, the retrieved error digits are summarized in Table 3.4. In this case, errors occurred in the information block $(\tilde{x}_1, \tilde{x}_2) = (4, 5)$ cannot be successfully retrieved due to more than three residue digit errors occur in all three double residue digit error correction circuits. Therefore, the corrected information residue digits cannot be obtained.

Table 3.4. Retrieved error digits for all partitioned information blocks and their majority voting results (uncorrectable case).

Partitioned Information Block	Double Residue Digit Error Correction 1	Double Residue Digit Error Correction 2	Double Residue Digit Error Correction 3	Majority Voting Result
$(\tilde{x}_1, \tilde{x}_2) = (4, 5)$	too_many_error = 1	too_many_error = 1	too_many_error = 1	Invalid
$(\tilde{x}_3, \tilde{x}_4) = (8, 18)$	$(e_3, e_4, e_{r1}, e_{r2}, e_{r3}, e_{r4}) \equiv (2, 0, 3, 0, 0, 0)$	$(e_3, e_4, e_{r5}, e_{r6}, e_{r7}, e_{r8}) \equiv (2, 0, 1, 0, 0, 0)$	$(e_3, e_4, e_{r9}, e_{r10}, e_{r11}, e_{r12}) \equiv (2, 0, 0, 0, 1, 0)$	$(e_3, e_4) \equiv (2, 0)$
$(\tilde{x}_5, \tilde{x}_6) = (1, 18)$	$(e_5, e_6, e_{r1}, e_{r2}, e_{r3}, e_{r4}) \equiv (6, 0, 3, 0, 0, 0)$	$(e_5, e_6, e_{r5}, e_{r6}, e_{r7}, e_{r8}) \equiv (6, 0, 1, 0, 0, 0)$	$(e_5, e_6, e_{r9}, e_{r10}, e_{r11}, e_{r12}) \equiv (6, 0, 0, 0, 1, 0)$	$(e_5, e_6) \equiv (6, 0)$

Chapter 4 Results and Discussion

4.1 FPGA Synthesis Results and Analysis

Field-programmable gate array (FPGA) is a popular implementation platform for RNS applications. Implementation of RRNS-based fault-tolerant circuits on FPGA has advantages of reconfigurability, flexibility and programmability [24] that ASIC designs cannot match. It is easy for designers to reconfigure the overall system by reconfiguring the logic inside FPGA when different moduli sets are used for different applications. The high recurring engineering cost to redesign, debug and tape out the whole system has been eliminated.

The experiments are carried out in two aspects. Firstly, the proposed double residue digit error correction circuit is compared with two implementations of the recently proposed double residue digit error correction algorithms [20] and [17]. Since algorithm [17]#1 requires more iterative steps and more complex circuitry for modulo reduction than algorithm [17]#2, only [17]#2 is compared in this experiment. Secondly, the proposed adaptive partition method using the proposed double error correction circuit is compared with direct implementation of the recent multiple residue digit error correction algorithm [17]#2. All the designs are described in Verilog HDL, verified by Xilinx Vivado 2016.2 design environment and synthesized on Virtex-UltraScale xcvu190-flgc104-2-e FPGA. The details of the FPGA device are summarized in Table 4.1. The real-time Soft Error Rate (SER) as reported in Xilinx Reliability Report [50] is 30 FIT/Mb = 3×10^{-5} FIT/bit with 90% confidence level.

Table 4.1 FPGA information.

Part Name	Virtex-UltraScale xcvu190-flgc104-2-e
I/O Pin Count	2104
Total Number of Flip-Flops	2148480
Total Number of LUTs	1047240
Technology Node	20 nm
Soft Error Rate (SER)	3×10^{-5} FIT/bit

For fair comparison, all the designs make use of the same type of modulo adders proposed in [35] and modulo multipliers in [39]. Modulo M reduction of the proposed designs, [20] and [17]#2 have been implemented by multi-level lookup tables as suggested in [49], where the partition size of each LUT has been carefully selected to optimize the hardware cost as well as delay. The synthesized number of LUTs, critical path delay, latency and input data rate are compared. None of the designs use DSP and BRAM blocks of the FPGA, therefore these dedicated resources are not shown in the comparison.

4.1.1 *Proposed Double Residue Digit Error Correction Circuit vs. RRNS-based Double Residue Digit Error Correction Circuits in Literature*

FPGA synthesis results for the proposed implementation and direct implementations of other recent RRNS-based double residue digit error correction algorithms are shown in Table 4.2 to Table 4.4. Five information moduli sets are selected for comparison: $\{63, 67\}$, $\{257, 263\}$, $\{17, 19, 23\}$, $\{63, 67, 71\}$ and $\{63, 67, 71, 73\}$. Note that algorithms [20], [17]#2 have different redundant moduli selection criteria from the proposed circuit. To fully optimize the designs, we choose the arbitrary moduli set for each design such that the size of each redundant moduli is minimized while meeting the uniqueness criteria. For example, with information moduli set $\{63, 67\}$, the redundant moduli set for

algorithm [20], [17]#2 and the proposed design should be {97, 101, 103, 107}, {71, 73, 79, 83} and {71, 73, 101, 103}, respectively. For the ease of comparison, the percentage saving in #LUTs and critical path delay reduction as well as throughput improvement for the proposed double residue digit error correction circuit over other benchmark designs are shown in Fig. 4.1.

Table 4.2 Comparison of LUT usage for double residue digit error correction circuits.

Information moduli set	#LUTs (% of available)		
	[20]	[17]#2	Proposed
{63, 67}	72327 (6.7)	4574 (0.4)	3963 (0.4)
{257, 263}	919861 (85.6)	10404 (1.0)	7876 (0.7)
{17, 19, 23}	32549 (3.0)	6240 (0.6)	4242 (0.4)
{63, 67, 71}	72423 (6.7)	12919 (1.2)	10080 (0.9)
{63, 67, 71, 73}	1572307* (146.4)	25807 (2.4)	22242 (2.1)

*estimated value(s) of the #LUTs from synthesis tool

Due to its pipelined structure, the proposed design requires slightly more flip-flops than the two benchmark designs. However, the number of additional flip-flops is very small compared with the number of total available flip-flops (2148480). For hardware cost comparison, #LUTs is more important than #flip-flops. Based on the synthesis results, the proposed double residue digit error correction circuits under all selected moduli sets have much less LUT usage. In algorithm [20], each mapping unit needs to contain all the error combinations based on four pseudo-syndrome values. Therefore, the cost on #LUTs is huge. As for [17]#2, its hardware cost mainly depends on the number of iterations and the complexity of modulo reduction operation $|Y|_{Z_c}$, where the modulus $Z_c \geq M_K$. For $t = 2$, its implementation consists of three $|Y|_{Z_c}$ blocks and each

$|Y|_{Z_C}$ block needs many multi-level LUTs as well as modulo Z_C adders to realize.

The error retrieval process for the proposed double-error correction circuit only involves three low complexity base-extension operations and a few binary comparators. It does not need complicated lookup tables as well as iterative computation steps. By dividing the received residue digits into three groups, the complexity of modulo operations such as modulo reduction and CRT has been largely reduced. Based on the five sets of design cases shown in Table 4.2 and Fig. 4.1(a), our proposed design reduces #LUT of [20] and [17]#2 by 93.1% and 21.1%, respectively.

Table 4.3 Comparison of critical path delay for double residue digit error correction circuits.

Information moduli set	Critical Path Delay (ns)		
	[20]	[17]#2	Proposed
{63, 67}	25.2	21.8	16.1
{257, 263}	28.1	27.0	19.4
{17, 19, 23}	24.3	22.6	16.5
{63, 67, 71}	24.0	30.5	21.1
{63, 67, 71, 73}	24.9	36.8*	27.1

*estimated value(s) of the critical path delay from synthesis tool

Table 4.4 Comparison of latency and throughput for double residue digit error correction circuits.

Information moduli set	Latency (#cycles) and Throughput (Mbps)					
	[20]		[17]#2		Proposed	
	Latency (#cycles)	Throughput (Mbps)	Latency (#cycles)	Throughput (Mbps)	Latency (#cycles)	Throughput (Mbps)
{63, 67}	0	476.2	3	183.5	1	740.7
{257, 263}	0	569.4	3	197.5	1	824.7
{17, 19, 23}	0	617.3	4	165.9	1	909.1
{63, 67, 71}	0	833.3	4	163.9	1	947.9
{63, 67, 71, 73}	0	1084.3	5	146.7*	1	996.3

*throughput value(s) based on the estimated critical path delay value(s)

Table 4.3 and Fig. 4.1(a) shows that the proposed work is on average 20.5% and 27.7% faster than [20] and [17]#2, respectively. With the pipelined structure, the critical path delay of the proposed circuit occurs at the generation of error digits in PCOM block. The complexity of modulo reduction operations are much smaller compared with those in [17]#2. Compared with [20], the proposed double residue digit error correction circuit has much lower delay for all selected moduli sets except for {63, 67, 71, 73}. For the information moduli set {63, 67, 71, 73}, the proposed circuit consumes 98.6% less LUT resources with slightly worse delay (2.2 ns) compared with [20]. The design of [17]#2 cannot benefit much from pipelining due to its iterative structure with computational depth dependent on the number of residue channels. Its minimum waiting time between data transmission becomes $(\text{latency} \times \text{critical path delay})$, which results in very low input data rate. On the other hand, there is no waiting time in [20] and the proposed design. Therefore, input data rate depends solely on the critical path delay of the circuit. Our proposed design has achieved on average 1.28 and 5.3 times higher input data rate compared with [20] and [17]#2, as shown in Table 4.4 and Fig. 4.1(b).

Fig. 4.1 Improvement in LUT usage, critical path delay and throughput (proposed design vs. other benchmark designs).

4.1.2 Comparison of Proposed Adaptive Input Partitioned Multiple Residue Digit Error Correction Circuit and Other RRNS-based Multiple Residue Digit Error Correction Circuits

For multiple residue digit error correction, experiments have been carried out for [17]#2 and the proposed adaptive partitioning scheme with different k and n , as shown in Table 4.5. From the results, three observations can be made: (1) for $k = 8$ and t varies from 6 to 8, #LUTs for [17]#2 grows from 573350 to 1204103, which is almost twice as much. As for the proposed scheme, its number of used LUTs increases from 64338 to 84148, which is only around 30.7% growth in #LUTs. On average, it has achieved 90.8% saving on LUT usage. (2) for $t = 6$ and $k = 6, 7, 8$ and 10, #LUTs increases with k for both [17]#2 and the proposed scheme. However, the growth of LUTs for the proposed scheme is much smaller than that for [17]#2 and the LUT usage for the proposed scheme is much smaller than that for [17]#2 for each combination of k and n . The proposed scheme achieves an average of 85.2% saving on LUT usage. (3) for $k = t = 2, 4, 6$ and 8,

the synthesized numbers of LUTs for [17]#2 are 4574, 37714, 243648 and 1204103, respectively. A significant growth of #LUTs with the increase of k and t is observed. On the other hand, for the proposed scheme, its synthesized numbers of LUTs are 3963, 8178, 48048 and 84148, respectively. From all three cases, it is obvious that the growth in #LUTs of the proposed design is much slower than [17]#2. This is due to the rapid increase in the number and complexity of the modulo reduction units for $|Y|_{z_c}$. When $t = 8$ and $k = 8$, [17]#2 uses up to 1204103 LUTs and when $t = 6$ and $k = 10$, it uses up to 1367983 LUTs. The LUT usages for these two cases are more than the total number of available LUTs (1047240) as shown in Table 4.1. Although Xilinx Vivado tool can still synthesize and estimate the LUT usage number, the designs cannot be fully implemented with the resources available on the actual FPGA device. For the proposed design, since it only consists of double residue digit error correction blocks, the growth of #LUT is bounded by $O(k^2)$. Compare with [17]#2, an average of 70.8% LUT usage saving has been achieved for different values of k and t .

Table 4.5 Comparison of LUT usage for multiple residue digit error correction circuits.

Size of Each Information Residue Channel (L)	(k, n)	t	#LUTs (% of available)		Saving (%)
			[17]#2	Proposed Scheme	
6-bit to 8-bit	(2, 6)	2	4574 (0.4)	3963 (0.4)	13.4
	(3, 9)	3	14929 (1.4)	8727 (0.8)	41.5
	(4, 12)	4	37714 (3.5)	8178 (0.8)	78.3
	(6, 18)	6	243648 (22.7)	48048 (4.5)	80.3
	(7, 19)	6	284675 (26.5)	63627 (5.9)	77.6
	(8, 20)	6	573350 (53.4)	64338 (6.0)	88.7
	(8, 24)	8	1204103* (112.1)	84148 (7.8)	93.0
	(10, 22)	6	1367983* (127.3)	79500 (7.4)	94.1

*estimated value(s) of the #LUTs from synthesis tool

Table 4.6 Comparison of critical path delay for multiple residue digit error correction circuits.

L	(k, n)	t	Critical Path Delay (ns)		Delay Reduction (%)
			[17]#2	Proposed Scheme	
6-bit to 8-bit	(2, 6)	2	21.1	16.1	23.7
	(3, 9)	3	27.5	15.5	43.6
	(4, 12)	4	30.1	16.2	46.2
	(6, 18)	6	35.1	18.9	46.1
	(7, 19)	6	25.4	19.3	24.0
	(8, 20)	6	40.5	18.7	53.8
	(8, 24)	8	40.6*	19.3	52.5
	(10, 22)	6	47.4*	19.4	59.1

*estimated value(s) of the critical path delay from synthesis tool

The synthesized results for critical path delay are summarized in Table 4.6. The critical path delay of [17]#2 is much larger than that of the proposed scheme. This is because for multiple digit error correction, the proposed scheme divides the information channels into pairs and use double residue digit error correction circuits to process them in parallel. The critical path delay T_{crit} of the whole system is equal to the critical path delay of the slowest double residue digit error

correction block (T_p), where T_p is much shorter than the critical path delay of [17]#2.

$$T_{crit} = \max \{T_1, T_2, \dots, T_p\} \quad (4.1)$$

From Table 4.6, the proposed scheme has achieved an average critical path delay of 17.9 ns. On average, it has reduced the critical path delay by 37.9% compared with [17]#2.

Table 4.7 Comparison of latency and throughput for multiple residue digit error correction circuits.

L	(k, n)	t	Latency (#cycles) and Throughput (Mbps)				Throughput Improvement (times)
			[17]#2		Proposed Scheme		
			Latency (#cycles)	Throughput (Mbps)	Latency (#cycles)	Throughput (Mbps)	
6-bit to 8-bit	(2, 6)	2	3	205.4	1	740.7	3.6
	(3, 9)	3	5	145.4	1	1290.3	8.8
	(4, 12)	4	8	112.1	1	1667.7	14.8
	(6, 18)	6	21	55.6	2	2169.3	39.0
	(7, 19)	6	30	62.9	2	2487.0	39.5
	(8, 20)	6	42	32.3	2	2941.2	91.0
	(8, 24)	8	58	23.3	2	2849.7	122.3
	(10, 22)	6	81	17.9	2	3556.7	198.7

*throughput value(s) based on the estimated critical path delay value(s)

The proposed work is implemented with a pipelined structure for higher input data rate. Input data can be transmitted during each cycle without additional waiting time. On the contrary, in [17], since new input data is not allowed to be transmitted before the previous set of data has been successfully processed, the duration between each set of input data transmission must be no less than $(T_{crit} \times Latency)$, which increases whenever k or t increases. When k increases from 2 to 10 and t increases from 2 to 8, [17]#2 requires more iteration steps to

process and correct one input data set. Therefore, its input data rate drops from 205.4 Mbps to 17.9 Mbps. For the proposed design, its input data rate increases from 740.7 Mbps to 3556.7 Mbps for $k = 2$ to 10 and $t = 2$ to 8. The proposed scheme achieves from 3.6 to 198.7 times higher throughput rate than that of [17]#2 for different k and t values.

4.2 Reliability Analysis

Generally, there are three basic metrics to measure system reliability as mentioned in [51]: (1) system reliability, (2) percentile life, and (3) average life / mean time to failure (MTTF). In this section, these three metrics will be analyzed for the proposed scheme and other benchmark designs.

4.2.1 System Reliability

System reliability is the probability of a system surviving a specified mission time [51]. In RRNS based circuits, the datapath can be modelled by a binary symmetric channel (BSC) where the probability of one random bit to be in error is independent of the actual bit value [52]. For easier analysis, the lengths of all residue channels are assumed to be the same (L bits). Therefore, the probability for a random residue channel to be free of error at some point of time T (years) can be modelled by $x(T)$ ($0 < x(T) < 1$). s processing steps are needed before data reaches the error correction module (ECM). According to the military standard for electronic reliability design (MIL-HDBK-338 [53]), $x(T)$ has an exponential relationship with time T and can be expressed by Equation (4.2):

$$x(T) = e^{-s\lambda LT} \quad (4.2)$$

where λ is the average device failure rate in $[\#/(bit \cdot year)]$ and the value of $s = 10^6$ is taken from the largest experimental value suggested in [52].

For an RNS system with k information residue channels, without any error correction module (NoECM), its reliability R_{NoECM} at time T can be expressed as

$$\begin{aligned} R_{NoECM} &= x(T)^k \\ &= e^{-s\lambda kLT} \end{aligned} \quad (4.3)$$

If such RNS system is hardened by an N -Modular Redundancy (N -MR) system, its system reliability R_{NMR} at time T becomes

$$R_{NMR} = \sum_{i=N+1}^{2N+1} \binom{2N+1}{i} R_{NoECM}^i (1 - R_{NoECM})^{2N+1-i} \quad (4.4)$$

In [17]#2, an (n, k) RRNS with k information residue channels and $r = (n - k)$

redundant residue channels can correct up to $t = \left\lfloor \frac{r}{2} \right\rfloor$ residue digit errors. The

error correction scheme proposed in [17]#2 with t number of correctable errors is defined as RRNS(t) in this thesis for the ease of analysis. The system reliability for RRNS(t) at time T can be expressed by

$$R_{RRNS(t)} = x(T)^n + \binom{n}{1} (1 - x(T))^1 x(T)^{n-1} + \dots + \binom{n}{t} (1 - x(T))^t x(T)^{n-t} \quad (4.5)$$

For the proposed double residue digit error correction circuit, it has two information residue channels and four redundant residue channels. At some point of time T , the probability of the proposed double residue error correction circuit to be free of errors R_d can be expressed as

$$\begin{aligned}
R_D &= x(T)^6 + \binom{6}{1}(1-x(T))^1 x(T)^5 + \binom{6}{2}(1-x(T))^2 x(T)^4 \\
&= 10x(T)^6 - 24x(T)^5 + 15x(T)^4 \\
&= 10e^{-6s\lambda LT} - 24e^{-5s\lambda LT} + 15e^{-4s\lambda LT}
\end{aligned} \tag{4.6}$$

In the proposed adaptive partitioning scheme for multiple residue digit error correction, when $\left\lfloor \frac{r}{4} \right\rfloor < 3$, majority voting scheme is not adopted. Therefore, for each partitioned information channel pair, only one double-error correction circuit is used to retrieve residue digit errors. Since all these double-error correction circuits do not share common residue channels, they are independent from each other and the system reliability $R_{\text{proposed}\left(\left\lfloor \frac{r}{4} \right\rfloor < 3\right)}$ can be expressed as

$$R_{\text{proposed}\left(\left\lfloor \frac{r}{4} \right\rfloor < 3\right)} = R_D^p \tag{4.7}$$

When $\left\lfloor \frac{r}{4} \right\rfloor \geq 3$, for each pair of partitioned information channels, p' double-error correction circuits are used for detecting and correcting their residue digit errors. Therefore, two information channels are shared by p' double-error correction circuits. The reliability for each partitioned information block depends on the error location. The system reliability can be derived based on conditional probability concept as mentioned in [4]. In a random pair of partitioned information channels, there are only three cases for residue digit errors: (1) no error in information channels, (2) one error occurs among two information channels and (3) two residue digit errors occur in two information channels. Recall that in RRNS, residue channels are modelled by BSC and the probability

is identical for a random error occurs in a random residue channel. The probability for the three cases can be summarized in Table 4.8.

Table 4.8 Derivation of probability functions for three cases.

	Probability
Case 1 (no error)	$x(T)^2$
Case 2 (one residue digit error)	$2x(T) - 2x(T)^2$
Case 3 (two residue digit errors)	$(1 - x(T))^2$

In all three cases, due to the advantage of N -modular majority voting scheme, the system is error free as long as more than half of the double-error correction circuits can successfully retrieve the correct error residue digits. For **Case 1** in Table 4.8, the reliability can be expressed as

$$R_{Case1} = \sum_{i=\zeta+1}^{2\zeta+1} \binom{2\zeta+1}{i} R_1^i (1-R_1)^{2\zeta+1-i} \quad (4.8)$$

where $i = p' = 2\zeta + 1 = 3, 5, 7, \dots$ and

$$R_1 = x(T)^4 + \binom{4}{1} (1-x(T))x(T)^3 + \binom{4}{2} (1-x(T))^2 x(T)^2 \quad (4.9)$$

Similarly, for **Case 2** and **Case 3**, we have

$$R_{Case2} = \sum_{i=\zeta+1}^{2\zeta+1} \binom{2\zeta+1}{i} R_2^i (1-R_2)^{2\zeta+1-i} \quad (4.10)$$

$$R_{Case3} = \sum_{i=\zeta+1}^{2\zeta+1} \binom{2\zeta+1}{i} R_3^i (1-R_3)^{2\zeta+1-i} \quad (4.11)$$

where $i = p' = 2\zeta + 1 = 3, 5, 7, \dots$ and

$$R_2 = x(T)^4 + \binom{4}{1} (1-x(T))x(T)^3 \quad (4.12)$$

$$R_3 = x(T)^4 \quad (4.13)$$

Therefore, the overall system reliability of the proposed scheme $R_{proposed\left(\left\lfloor \frac{r}{4} \right\rfloor \geq 3\right)}$

can be expressed as:

$$R_{proposed\left(\left\lfloor \frac{r}{4} \right\rfloor \geq 3\right)} = \left[x(T)^2 R_{case1} + \left[2x(T) - 2x(T)^2 \right] R_{case2} + (1 - x(T))^2 R_{case3} \right]^p \quad (4.14)$$

Based on the above analysis, the system reliability (R) vs. time (T) can be plotted for the following designs: (1) no error correction module (NoECM), (2) N -MR ($N = 3, 5, 7$), (3) RRNS(t) and (4) proposed adaptive partitioning multiple error correction scheme. Fig. 4.2 shows the reliability curves for an RNS with k information residue channels under the following cases: (1) without any error correction module (NoECM), (2) with N -MR ($N = 3, 5, 7$), (3) with RRNS(t) for $k = 10, t = 6$ and $k = 10, t = 10$, and (4) with the proposed scheme for $k = 10, t = 6$ and $k = 10, t = 10$.

Fig. 4.2 Reliability curves for the RNS under different schemes ($k = 10$)

From Fig. 4.2, several observations can be made: Firstly, N -MR circuits have better reliability than the RNS without error correction modules (NoECM) when $R > 0.5$ and their reliability become worse than NoECM when $R < 0.5$. The point of $R = 0.5$ occurs at $T = 0.69MTTF_{NoECM} = 34$ (years), where $MTTF_{NoECM}$ is the MTTF of RNS without ECM and it can be computed by

$$MTTF_{NoECM} = \frac{1}{k\lambda L} \quad (4.15)$$

Secondly, the proposed scheme has achieved much higher reliability than NoECM and N -MR systems with different k values at all time. Thirdly, there is almost no difference of the system reliability between the proposed scheme and RRNS(t) when $T < 0.69MTTF_{NoECM} = 34$ (years). When T increases above 34 years, each residue channel has higher chance to be affected by errors. This results in a higher probability of occurrence of uncorrectable error cases, and therefore its reliability starts to become worse than RRNS(t) when $T > 0.69MTTF_{NoECM} = 34$ (years).

When the number of correctable errors t is increases from 6 to 10, the system reliability levels of the proposed scheme and RRNS(t) stay at nearly 100% and do not drop when $T < 0.69MTTF_{NoECM} = 34$ (years). When $T > 0.69MTTF_{NoECM} = 34$ (years), the reliability levels of both schemes start to drop. Based on Fig. 4.2, with higher number of correctable errors t , the overall reliability increases for both RRNS(t) and the proposed scheme. With longer service span, errors are more likely to occur in each of the residue channels. The

overall system reliability becomes worse due to more uncorrectable residue digit error cases, resulting in its poorer reliability than $RRNS(t)$ for lifespan longer than 34 years.

4.2.2 Percentile Life

Percentile life stands for the time at which the system reliability meets a specific value [51]. Based on system failure level, percentile life profile is presented in Fig. 4.3.

Fig. 4.3 Percentile life for different designs

As system failure level becomes higher, percentile life of the proposed scheme becomes worse than that of $RRNS(t)$. The difference in the percentile life between the proposed and ideal $RRNS$ becomes larger as failure level increases. For example, at failure level $F = 1 - R = 0.1$, the proposed scheme with $t = 6$ has a percentile life of 80 years, whilst $RRNS(t)$ has a slightly better percentile life of 100 years with the same t . Compared with NoECM and N -MR systems, the

proposed scheme has achieved much higher percentile life for almost all failure levels except for the case that failure level is very close to 1. However, such scenario only happens when the components used in the system are of very low quality. For the applications that need high system reliability, poor quality components should be avoided in the design phase. Table 4.9 shows the percentile life values of different designs at low failure levels of 0.05 and 0.1. Based on these results, the proposed scheme ($t = 10$) has achieved 72.4 and 87.1 years of percentile life at $F = 0.05$ and 0.1, respectively. Compared with RNS without ECM (NoECM), the percentile life of the proposed scheme is 30.2 and 17.4 times higher at $F = 0.05$ and 0.1, respectively.

Table 4.9 Percentile life at different failure levels ($F = 0.1$ and $F = 0.05$).

Design	Percentile Life (years) ($F = 0.05$)	Percentile Life (years) ($F = 0.1$)
NoECM	2.4	5.0
TMR	6.8	10.1
5-MR	9.5	13.5
7-MR	12.2	15.5
RRNS(t) ($t = 6$)	82.9	98.1
RRNS(t) ($t = 10$)	118.8	135.2
Proposed Scheme ($t = 6$)	34.2	45.0
Proposed Scheme ($t = 10$)	72.4	87.1

4.2.3 Mean Time to Failure (MTTF)

Mean time to failure measures the average lifetime that the system can operate without failure [54]. Based on Equation (4.2) to Equation (4.14), the system reliability of the RNS system without ECM, proposed scheme, RRNS(t) as well as N -MR systems can be summarized in Table 4.10.

Table 4.10 Summary of system reliability (R) for different designs.

Design	System Reliability (R)
NoECM	$R_{NoECM} = x(T)^k$
N -MR	$R_{N-MR} = \sum_{i=N+1}^{2N+1} \binom{2N+1}{i} R_{NoECM}^i (1 - R_{NoECM})^{2N+1-i}$
RRNS(t) ([17]#2)	$R_{RRNS(t)} = x(T)^n + \binom{n}{1} (1 - x(T))^1 x(T)^{n-1} + \dots + \binom{n}{t} (1 - x(T))^t x(T)^{n-t}$
Proposed Scheme	$R_{proposed\left(\left\lfloor \frac{r}{4} \right\rfloor < 3\right)} = R_D^p$ $R_{proposed\left(\left\lfloor \frac{r}{4} \right\rfloor \geq 3\right)} = \left[x(T)^2 R_{case1} + \left[2x(T) - 2x(T)^2 \right] R_{case2} + (1 - x(T))^2 R_{case3} \right]^p$

Given the reliability function R with respect to time (T), the mean-time-to-failure ($MTTF$) can be expressed as

$$MTTF = \int_0^{\infty} R(T) dT \quad (4.16)$$

With $k = 10$, $MTTF$ of each design can be summarized in Table 4.11.

Table 4.11 $MTTF$ of different designs ($k = 10$).

Design	$MTTF$ (years)
NoECM	47.5
TMR	39.6
5-MR	37.2
7-MR	36.0
RRNS(t) ([17]#2)	177.2 ($t = 6$) 212.7 ($t = 10$)
Proposed Scheme	56.5 ($t = 6$) 55.9 ($t = 10$)

From Table 4.11, it can be observed that the proposed scheme has achieved an average of 56.2 years of $MTTF$ with $k = 10$ for $t = 6$ and 10. It has on average 40.2% better $MTTF$ than NoECM, TMR, 5-MR and 7-MR designs. In addition, the $MTTF$ of N -MR designs are worse than that of NoECM. When the usage time reaches $0.69MTTF_{NoECM}$, the component reliability drops below 0.5, and the

system reliability of N -MR becomes worse than that of NoECM. This indicates that N -MR systems are suitable for applications whose shelf life is shorter than 0.69 of MTTF without ECM. Normally, MTTF of a system should be much larger than its useful life. This means that arithmetic systems incorporating the proposed scheme can achieve longer useful life than those without ECM or with N -MR. As expected, RRNS(t) has the highest MTTF values (177.2 years when $t = 6$ and 212.7 years when $t = 10$) among all the designs since it can cover all the residue digit errors cases within its maximum number of correctable errors. In the proposed scheme, there will be some error cases that the system cannot correct as mentioned in the previous chapter.

In summary, the proposed scheme has achieved much better system reliability, percentile life profile as well as MTTF compared with those designs without ECM or with N -MR techniques. Although the system reliability of the proposed scheme is not better than [17]#2, it remains acceptable for applications that require a high level of fault-tolerance because its system reliability stays at very high level before $0.69MTTF_{NoECM}$ years. The system reliability of the proposed scheme only becomes worse than that of [17]#2 after $0.69MTTF_{NoECM}$ years. Usually, $0.69MTTF_{NoECM}$ is a relatively long time and it is longer than the specified system useful life unless the components used in the system are of poor quality.

Chapter 5 Conclusions and Future Works

5.1 Conclusions

This research aims to develop a scalable hardware-efficient implementation scheme to achieve multiple residue digit error detection and correction on FPGA platform (Virtex-UltraScale xcvu190-flgc2104-2-e FPGA, with a total of 1074240 LUTs available). For RRNS-based multiple residue digit error correction algorithms, so far there is no hardware efficient implementation reported in literature. The main reason is that existing RRNS-based algorithms rely on either large error lookup tables or iterative computation which are not suitable for efficient hardware implementation as the number of correctable error increases. Direct implementations of them either require very large circuit area or can only be operated at very slow speed.

In this thesis, firstly, an arithmetic residue digit error retrieval circuit structure for double residue digit error correction has been proposed. The implementation is based on a syndrome-based reduced table-lookup algorithm that led to a new architectural design for lower hardware cost and higher circuit speed. The syndrome generator has been designed to reduce the hardware cost by replacing the modulo reduction LUTs with binary comparators, sharing common logic and adopting multi-level table lookup technique with adaptive partitioned operand sizes. The error retrieval process is realized by appropriately reorder the error vectors for table-lookup and applying modulo arithmetic properties to replace lookup tables (LUTs) with logic and adder-based circuits. The circuit has been implemented with a pipelined structure for faster speed and higher throughput.

Synthesis results show that the proposed double residue digit error correction circuit achieves the lowest hardware cost (#LUTs), lowest critical path delay and much larger throughput rate compared with the direct implementations of two recent RRNS-based double residue error correction algorithms.

Secondly, an adaptive partitioning scheme was proposed for multiple residue digit error correction. The proposed scheme adaptively partitions all the information channels into several two-channel blocks to handle smaller scale error correction task independently. Each of the partitioned block is implemented by a double residue digit error correction circuit and they can be executed in parallel with reduced computational complexity. The growth of LUT sizes with input dynamic range, number of information channels and error correction capability has been significantly suppressed. With different moduli sets from $k = 2$ to 10, the proposed scheme for multiple residue digit error correction achieves on average saving of 70.8% LUT usage, 37.9% less critical path delay and much higher throughput (3.6x ~ 198.7x) than the direct implementation of the most recently proposed algorithm [17]#2. The drop on system reliability for the proposed scheme is within the acceptable range for applications that require high level of fault-tolerance.

5.2 Future Works

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

a) The authors of [16]#2 have proposed a double residue digit error correction algorithm. The algorithm is based on consistency check and it requires

$\binom{k}{2} = \frac{k(k-1)}{2}$ iterative steps to accomplish one error correction process, where

k stands for the number of information residue channels. The selection condition of redundant moduli set is expressed in Equation (5.1):

$$\min\{m_{r_1}, m_{r_2}\} > \max\{2m_{i_1}m_{i_2} - m_{i_1} - m_{i_2}\} \quad (5.1)$$

where $k < r_1, r_2 \leq n$, n stands for the total number of moduli channels and

$1 \leq i_1, i_2 \leq k$. This condition leads to a more relaxed redundant space which may

help reduce hardware cost for double residue digit error correction circuit design.

On the other hand, the consistency checking based algorithm [16]#2 can be implemented with only a few modulo adders, modulo multipliers and binary comparators given that the number of information residue channels is very small.

When $k = 2$, it is possible to replace the iterative circuit structure with combinational logic structure. This may lead to better hardware efficiency if such implementation of double residue digit error correction circuit is integrated into the proposed multiple residue digit error correction scheme. However, it remains to be seen whether there is an effective way to implement such iterative structure into combinational circuit structure and how much more efficiency it can achieve compared with using the existing implementations.

b) Hybrid CMOS/non-CMOS memories (hybrid memories) suffer from high degree of cluster faults. RRNS encoders and decoders can be added into a hybrid memory with respect to each modulo channel. In [55], two modified RRNS based error correcting codes were proposed for hybrid memories: (1) Three Non-

Redundant Moduli RRNS (3NRM-RRNS) and (2) Two Non-Redundant Moduli RRNS (2NRM-RRNS). As for 2NRM-RRNS, since it only consists of two information residue channels, to cover those information residue channels, a double residue digit error correction circuit is needed. Special moduli set $\{2^f + 1, 2^f\}_{f=8}$ is usually adopted for such application. Compare with existing RRNS-based error correction algorithms, the proposed arithmetic error retrieval double residue error correction circuit may benefit more from the selection of moduli set in terms of hardware cost, critical path delay as well as throughput. Therefore, a double residue digit error correction circuit dedicated to hybrid memories using special moduli set $\{2^f + 1, 2^f\}_{f=8}$ can be designed for better hardware efficiency.

c) Cryptographic algorithms are employed in many consumer products for high security requirements. These algorithms are designed in a way that they are difficult to break mathematically and there is no known methodology to reduce the secret key search space. However, the secret key information may leak through side channels. Differential fault analysis (DFA) attack is one of the very effective side-channel attacks. It can be realized through the injection of deliberate faults into a cryptographic device and the observation of the corresponding erroneous outputs. Fault detection and correction circuits based on RRNS provide good candidates to counter this kind of attack at lower cost. Injected channel faults can be corrected before final outputs are retrieved. Therefore, a hardware-efficient RRNS-based error correction circuit implementation will be very attractive as a countermeasure for DFA attacks and

dedicate RRNS circuit can be designed to enhance the security of cryptographic systems.

Author's Publications

Journal Paper(s):

Yufei Zhang and Chip-Hong Chang, “An FPGA Implementation of Redundant Residue Number System for Low-cost Fast Speed Fault-Tolerant Computations with Arbitrary Moduli Set”, IEEE Transactions on Circuits and Systems I: Regular Papers (to be submitted in December 2018)

References

- [1] H. Iwai, “Roadmap for 22nm and beyond (Invited Paper),” *Microelectronic Engineering*, vol. 86, no. 7-9, pp. 1520-1528, 2009.
- [2] V. P. Nelson, “Fault-Tolerant Computing: Fundamental Concepts,” *Computer*, vol. 23, no. 7, pp. 19-25, 1990.
- [3] S. Osaki, “Performance/Reliability Measures for Fault-Tolerant Computing Systems,” *IEEE Transactions on Reliability*, Vols. R-33, no. 4, pp. 268-271, 1984.
- [4] E. Dubrova, *Fault-Tolerant Design*, New York: Springer-Verlag New York, 2013.
- [5] A.S. Molahosseini, L. Sousa, C. H. Chang, *Embedded Systems Design with Special Arithmetic and Number Systems*, Springer, 2017.
- [6] M. Berg and I. Koren, “On Switching Policies for Modular Redundancy Fault-Tolerant Computing Systems,” *IEEE Transactions on Computers*, Vols. C-36, no. 9, pp. 1052-1062, 1987.
- [7] J. P. Hayes, “A Graph Model for Fault-Tolerant Computing Systems,” *IEEE Transactions on Computers*, Vols. C-25, no. 9, pp. 875-884, 1976.
- [8] T. F. Tay and C. H. Chang, “A non-iterative multiple residue digit error detection and correction algorithm in RRNS,” *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 396-408, February 2016.

- [9] E. D. D. Claudio, G. Orlandi, F. Piazza, "A systolic redundant residue arithmetic error correction circuit," *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 427-432, 1993.
- [10] W. K. Jenkins and E. J. Altman, "Self-checking properties of residue number error checkers based on mixed radix conversion," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 2, pp. 159-167, 1988.
- [11] P. Yin and L. Li, "A new algorithm for single error correction In RRNS," in *International Conference on Communications, Circuits and Systems (ICCCAS)*, Chengdu, 2013.
- [12] Vik Tor Goh, M. Tinauli and M. U. Siddiqi, "A novel error correction scheme based on the Chinese remainder theorem," in *The Ninth International Conference on Communications Systems*, Singapore, China, 2004.
- [13] Yangyang Tang, E. Boutillon, C. Jégo and M. Jézéquel, "A new single-error correction scheme based on self-diagnosis residue number arithmetic," in *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Edinburgh, 2010.
- [14] T. F. Tay and C. H. Chang, "A new algorithm for single residue digit error correction in Redundant Residue Number System," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Melbourne, 2014.
- [15] S. S. S. Yau and Yu-Cheng Liu, "Error Correction in Redundant Residue Number Systems," *IEEE Transactions on Computers*, Vols. C-22, no. 1, pp. 5-11, 1973.

- [16] J. D. Sun and H. Krishna, "A coding theory approach to error control in redundant residue number systems. II. Multiple error detection and correction," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 1, pp. 18-34, 1992.
- [17] V. T. Goh and M. U. Siddiqi, "Multiple error detection and correction based on redundant residue number systems," *IEEE Transactions on Communications*, vol. 56, no. 3, pp. 325-330, 2008.
- [18] D. Mandelbaum, "On a class of arithmetic codes and a decoding algorithm," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 85-88, 1976.
- [19] O. Goldreich, D. Ron and M. Sudan, "Chinese remaindering with errors," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1330-1338, 2000.
- [20] GangQiang Liu , Lei Li, "A Modified Algorithm for Double Errors Correction in RRNS," in *IEEE 13th International Conference on Signal Processing (ICSP)*, 2016.
- [21] H. L. Garner, "The Residue Number System," *IRE Transactions on Electronic Computers*, Vols. EC-8, no. 2, pp. 140-147, 1959.
- [22] A. Omondi and B. Premkumar, *Residue Number Systems: Theory and Implementations*, London: Imperial College Press, 2007.
- [23] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, McGraw-Hill, 1967.
- [24] C. H. Chang, 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, 2015.
- [25] A.B. Premkumar, E.L. Ang, and E.M. Lai, “Improved memoryless RNS forward converter based on the periodicity of residues,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 2, pp. 133-157, 2006.
- [26] J. Y. S. Low and C. H. Chang, “A New Approach to the Design of Efficient Residue Generators for Arbitrary Moduli,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 9, pp. 2366-2374, 2013.
- [27] R. A. Patel, M. Benaissa, N. Powell, and S. Boussakta, “Novel power-delay-area-efficient approach to generic modular addition,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 6, pp. 1279-1292, 2007.
- [28] R.A. Patel, M. Benaissa, and S. Boussakta, “Fast parallel-prefix architectures for modulo 2^n-1 addition with a single representation of zero,” *IEEE Transactions on Computers*, vol. 56, no. 11, pp. 1484-1492, 2007.
- [29] H.T. Vergos and G. Dimitrakopoulos, “On modulo 2^{n+1} adder design,” *IEEE Transactions on Computers*, vol. 61, no. 2, pp. 173-186, 2012.
- [30] R. Muralidharan and C. H. Chang, “Area-Power Efficient Modulo 2^n-1 and Modulo 2^n+1 Multipliers for $\{2^n-1, 2^n, 2^n+1\}$ Based RNS,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2263-2274, 2012.

- [31] T. Fatt Tay and C. H. Chang, "A non-iterative multiple residue digit error detection and correction algorithm in RRNS," *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 396-408, February 2016.
- [32] H. T. Vergos, C. Efstathiou and D. Nikolos, "Diminished-one modulo $2n+1$ adder design," *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1389-1399, 2002.
- [33] C. Efstathiou, H. T. Vergos, and D. Nikolos., "Fast Parallel-Prefix Modulo $2n + 1$ Adders," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1211-1216, 2004.
- [34] A. A. Hiasat, "High-speed and reduced-area modular adder structures for RNS," *IEEE Transactions on Computers*, vol. 51, no. 1, pp. 84-89, 2002.
- [35] T. F. Tay and C. H. Chang, "A new unified modular adder/subtractor for arbitrary moduli," in *International Symposium on Circuits and Systems (ISCAS)*, Lisbon, 2015.
- [36] R. Muralidharan and C. H. Chang, "Radix-4 and Radix-8 Booth Encoded Multi-Modulus Multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 11, pp. 2940-2952, 2013.
- [37] R. Zimmermann, "Efficient VLSI implementation of modulo $(2n\pm 1)$ addition and multiplication," in *Proceedings 14th IEEE Symposium on Computer Arithmetic*, Adelaide, SA, 1999.
- [38] L. Sousa and R. Chaves, "A universal architecture for designing efficient modulo $2n+1$ multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 6, pp. 1166-1178, 2005.

- [39] A. A. Hiasat, "New efficient structure for a modular multiplier for RNS," *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 170-174, 2000.
- [40] H. Krishna, K. Y. Lin and J. D. Sun, "A coding theory approach to error control in redundant residue number systems. I. Theory and single error correction," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 1, pp. 8-17, 1992.
- [41] C. Ding, D. Pei and A. Salomaa, Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography, Singapore: World Scientific Publishing, 1996.
- [42] M. Etzel and W. Jenkins, "Redundant residue number systems for error detection and correction in digital filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 5, pp. 538-545, 1980.
- [43] N. Z. Haron and S. Hamdioui, "Redundant residue number system code for fault-tolerant hybrid memories," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 7, no. 1, 2011.
- [44] S. Pontarelli, G. C. Cardarilli, M. Re and A. Salsano, "Totally Fault Tolerant RNS Based FIR Filters," in *14th IEEE International On-Line Testing Symposium*, Rhodes, 2008.
- [45] Z. Gao et al., "Efficient Arithmetic-Residue-Based SEU-Tolerant FIR Filter Design," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 8, pp. 497-501, 2013.
- [46] T. Keller, T.-H. Liew, and L. Hanzo, "Adaptive redundant residue number system coded multicarrier modulation," *IEEE Journal on Selected Areas in Communication*, vol. 18, no. 11, pp. 2292-2301, 2000.

- [47] H. Lenstra, "Integer programming with a fixed number of variables," *Mathematics of Operations Research*, vol. 8, no. 4, pp. 538-548, 1983.
- [48] G. X. Yao, J. Fan, R. C. C. Cheung and I. Verbauwhede, "Novel RNS Parameter Selection for Fast Modular Multiplication," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 2099-2105, 2014.
- [49] B. Parhami, "Modular reduction by multi-level table lookup," in *Proceedings of 40th Midwest Symposium on Circuits and Systems*, Sacramento, 1997.
- [50] Xilinx, "Device Reliability Report (UG116 v10.8.2)," Xilinx, 2018.
- [51] Kyungmee O.K. and Way K., "Percentile Life and Reliability As Performance Measures in Optimal System Design," *IIE Transactions*, vol. 35, no. 12, pp. 1133-1142, 2003.
- [52] L. Li and J. Hu, "Redundant Residue Number Systems Based Radiation Hardening for Datapath," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 2332-2343, 2010.
- [53] U. D. o. Defense, MIL-HDBK-338: Electronic reliability design handbook, 1998.
- [54] Kalina K. G. and Olivier L. W., "Economic case for the retirement of geosynchronous communication satellites via space tugs," *Acta Astronautica*, vol. 58, no. 9, pp. 485-498, 2006.
- [55] N. Z. Haron and S. Hamdioui, "Using RRNS Codes for Cluster Faults Tolerance in Hybrid Memories," in *24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Chicago, 2009.