Asian Journal of Research in Computer Science

xx(x): ....., 20yy; Article no.AJRCOS.xxxxx

ISSN: 2456-9968

(Past name: British Journal of Mathematics & Computer Science, Past ISSN: 2231-0851)

Modulus Computation-Based Techniques for Detecting and Correcting Transmission and Computation Errors in Residue Number System Architectures

Type of Article

Received: XX December 20XX
Accepted: XX December 20XX
Online Ready: XX December 20XX

Abstract

The Residue Number System (RNS) offers significant advantages in parallel, carry-free arithmetic for high-performance computing but remains critically vulnerable to errors during transmission and computation. Traditional error detection approaches rely on post-processing reverse conversion, which introduces substantial latency and undermines the inherent speed of RNS, making them unsuitable for real-time, reliability-critical applications. This paper proposes a novel architecture for in-situ error detection and correction that operates directly within the residue domain, eliminating the need for costly full reverse conversion. Using an optimized moduli set $\{2^{n+1}-1, 2^n+1, 2^n, 2^n-1, 2^{n-1}-1\}$, we develop a hybrid algorithm that combines the Modulus Computation Method (MCM) for rapid reverse estimation with Hamming distance-based majority voting for robust syndrome analysis. The method guarantees single-residue error detection and correction by systematically evaluating residue triples to identify consistent values. Experimental simulations demonstrate a 99.9% correction success rate for realistic fault probabilities ($p \leq 10^{-4}$) while maintaining a low-latency, hardware-efficient pipeline. Comparative analysis against state-of-the-art techniques confirms superior performance in area utilization (complexity $5n+1$), detection latency (3 cycles), and flexibility across generalized moduli sets. These results advance the design of fault-tolerant RNS architectures for applications such as cryptography and digital signal processing.

Keywords: Residue Number System; Modulus Computation; Error Detection; Error Correction; Fault Tolerance; Real-Time Systems; Modular Arithmetic

1 Introduction

The Residue Number System (RNS) is a non-weighted number system grounded in modular arithmetic, representing integers as a set of residues with respect to pairwise coprime moduli (Bankas & Gbolagade, 2019; Zhang, 2024). Its origins can be traced back to ancient mathematics, with early references in Sun Tzu's Suan-Ching (circa 100 AD) and formal theoretical foundations later developed by Gauss in Disquisitiones Arithmeticae (Restivo, 1992). RNS inherently supports parallelism, carry-free computation, and modularity, making it highly efficient for arithmetic-intensive applications such as cryptography, digital signal processing (DSP), error-resilient communications, and high-performance computing (Olawale et al., 2019).

In RNS, operations such as addition, subtraction, and multiplication are naturally parallelizable since residues corresponding to different moduli can be processed independently. This results in significant speed and energy efficiency gains, particularly in hardware implementations (Patronik & Piestrak, 2017; Chang et al., 2015; Akobre et al., 2025). However, certain operations such as sign detection, magnitude comparison, division, scaling, reverse conversion, and, critically, error and overflow detection remain computationally challenging due to the non-positional nature of the system.

A key limitation in conventional RNS architectures is that error detection (whether due to transmission noise, hardware faults, or computational inaccuracies) and overflow identification are typically performed after reverse conversion to a conventional number system (Boraten & Kodi, 2017). This post-processing approach is inefficient: errors detected late can propagate through intermediate computations, increasing correction costs and degrading performance in real-time systems. Moreover, existing overflow detection methods such as sign recognition, group number methods, base extension, or reconstruction via the Chinese Remainder Theorem (CRT) or Mixed Radix Conversion (MRC) often require significant additional computation, limiting their suitability for time-critical applications (Agbedemnab et al., 2018). With RNS increasingly adopted in domains where accuracy and real-time performance are non-negotiable, including secure cryptographic systems and DSP pipelines, there is a pressing need for efficient, in-situ error and overflow detection techniques that operate before reverse conversion. Such methods would not only improve computational reliability but also reduce latency and prevent the propagation of faulty data (Anny, 2025).

This study addresses this gap by developing a modulus computation–based architecture for early detection and correction of transmission and computational errors in RNS addition, alongside robust overflow detection. The approach leverages the structural properties of coprime moduli sets to enhance detection efficiency while maintaining the inherent speed advantages of RNS. By integrating detection directly into the arithmetic process, the proposed method aims to deliver faster, more accurate, and more fault-tolerant RNS operations, thereby strengthening its applicability in high-performance and security-critical environments.

2 Related Works

Residue Number Systems (RNS) provide carry-free, parallel arithmetic attractive for high-throughput and reliability-critical domains, but their non-weighted representation makes detection and correction of transmission and computational errors nontrivial because faults manifest as residue inconsistencies rather than positional digit errors (Jenkins et al., 2018; Oke et al., 2021; Mohan & Mohan, 2016). Foundational work emphasizes how moduli-set design and dynamic-range considerations shape reliability, fault coverage, and detection latency (Chang et al., 2015; Mohan & Mohan, 2016).

Classical approaches rely on full or partial reverse conversion via the Chinese Remainder Theorem (CRT) or Mixed Radix Conversion (MRC) to validate residue vectors and correct faults, offering strong guarantees but incurring costly modular inversions, wide-word multiplications, and conversion overheads that hinder real-time deployment (Wang, 2000; Hosseinzadeh et al., 2009; Aremu & Gbolagade, 2017; Akkal & Siy, 2007; Ananda Mohan, 2017;

Cardarilli et al., 2020; Schinianakis, 2020). As a result, there is a persistent accuracy–latency trade-off between robust reconstruction-based checks and lightweight, conversion-avoiding screens (Mohan & Mohan, 2016; Oke et al., 2021).

The modulus computation method addresses this by performing consistency checks directly in the residue domain, flagging anomalies without immediate reverse conversion (Mohan & Mohan, 2016; Oke et al., 2021; Chang et al., 2015). Techniques such as dynamic range extension detect out-of-range values using selective extra modulus operations, substantially reducing conversion overhead while aligning with real-time constraints (Younes & Steffan, 2013; Siewobr & Gbolagade, 2014b). When errors are detected, structured congruence relations can localize the faulty channel(s), enabling targeted correction with minimal disruption to the parallel RNS pipeline (Mohan & Mohan, 2016; Chang et al., 2015; Akobre et al., 2025).

Integrating redundant residue number systems (RRNS) further enhances detectability and correctability by over-determining the residue set, allowing reconstruction from consistent subsets and improving resilience to single-channel faults (Goh & Siddiqi, 2008; Rouhifar et al., 2011; Mohan & Mohan, 2016; Chang et al., 2015). Hybrid strategies combining modulus-level checks, selective reconstruction, and redundancy strike a practical balance between coverage and latency, with recent work exploring adaptive control and learning-assisted switching among detectors under variable operating conditions (Cappello & Ziegler, 1979; Oke et al., 2021; Saheed et al., 2024; Modey et al., 2024; Akanni, 2024).

Agbedemnab et al. (2018) highlight that modulus computation–based detection provides fast and practical overflow screening, whereas RRNS and hybrid schemes enable scalable error correction. Traditional methods are computationally intensive and less suited for time-critical applications. Key gaps remain in coverage guarantees, large-scale false-positive/negative behavior, and design optimization for real-time, resource-constrained RNS systems (Akobre et al., 2025; Mohan & Mohan, 2016; Oke et al., 2021; Chang et al., 2015; Younes & Steffan, 2013).

3 METHODOLOGY

This section of the paper presents the methodology for the design and evaluation of a modulus computation–based architecture aimed at detecting and correcting both transmission and computational errors in Residue Number Systems (RNS). The proposed approach leverages a carefully selected moduli set because it is not arbitrary, but grounded in the theory of RNS error detection/correction (coverage bounds, dynamic range) and practical implementation constraints (speed, hardware efficiency, and resource utilization) (Valueva, 2019).

$$\begin{aligned}\mathcal{M} &= \{m_1, m_2, m_3, m_4, m_5\} \\ &= \{2^{n+1}-1, 2^n+1, 2^n, 2^n-1, 2^{n-1}-1\}.\end{aligned} \tag{3.1}$$

For detection and correction, and to maintain pairwise coprimeness for reverse conversion, we select an informational subset consisting of the first three moduli:

$$\mathcal{M}_{\text{info}} = \{m_1, m_2, m_3\} = \{2^{n+1}-1, 2^n+1, 2^n\} \tag{3.2}$$

and treat the remaining moduli $\{m_4, m_5\}$ as redundant for error control. For any $n \geq 2$, the first three moduli are pairwise coprime: $2^{n+1}-1$, 2^n+1, and 2^n. Let $X \in [0, M_{\text{info}})$ be encoded by residues $\mathbf{x} = (x_i)_{i=1}^5$ with $x_i \equiv X \pmod{m_i}$ and $M_{\text{info}} = \prod_{m \in \mathcal{M}_{\text{info}}} m$.

We receive $\mathbf{y} = (y_i)_{i=1}^5$, where at most $t = 1$ residue may be corrupted (single transmission/computation error).

3.1 Modulus Computation Method (MCM) for Reverse Conversion

The modulus computation method forms the core of the proposed architecture. Given a moduli set

$$M = \{m_1, m_2, m_3\}$$

and a decimal number X represented in residue form as (r_1, r_2, r_3), the method allows reverse conversion from residues to the original decimal

number using a cyclic jump procedure because the cyclic jump method balances speed, memory efficiency, and hardware simplicity, making it particularly suitable for real-time, resource-constrained RNS architectures:

- **First jump:**

$$j_1 = r_1,$$

$$L_1 = X - j_1 = \begin{bmatrix} |r_1 - j_1| \bmod m_1 = r_1' \\ |r_2 - j_1| \bmod m_2 = r_2' \\ |r_3 - j_1| \bmod m_3 = r_3' \end{bmatrix}$$

- **Second jump:**

$$j_2 = m_1 k_2, \quad k_2 = \left\lfloor \frac{r_2'}{m_1} \right\rfloor \mod m_2,$$

$$L_2 = X - j_2$$

- **Third jump:**

$$j_3 = m_1 m_2 k_3, \quad k_3 = \left\lfloor \frac{r_3''}{m_1 m_2} \right\rfloor \mod m_3,$$

$$L_3 = X - j_3$$

The decimal number is recovered as:

$$X = j_1 + j_2 + j_3$$

3.2 Proposed Detection and Correction Algorithm

In this study, we propose a novel approach for detecting and correcting single-residue errors in Residue Number Systems (RNS) that combines the strengths of modulus computation and the Hamming distance method. The Hamming distance method is well-suited for detecting and correcting single-bit (or limited multi-bit) errors in RNS architectures because it precisely localizes errors, incurs low computational overhead, integrates effectively with modulus computation for guided correction, guarantees deterministic single-bit error correction, and is hardware-friendly for FPGA or ASIC implementations. The key idea is to first estimate the original decimal value from the received residues, then iteratively verify and correct any inconsistencies caused by transmission errors. This approach ensures both low-latency detection and reliable correction in resource-constrained RNS implementations.

Algorithm Steps:

1. Compute the estimated decimal value $\bar{y}$ from the received residue vector using the modulus computation method. Given a received residue vector

$$\mathbf{r} = (r_1, r_2, \ldots, r_n)$$

and the expected residue vector derived from the modulus computation estimate $\bar{y}$, the Hamming distance d is defined as:

$$d(\mathbf{r}, \bar{\mathbf{r}}) = \sum_{i=1}^{n} \delta(r_i, \bar{r}_i),$$

where

$$\delta(r_i, \bar{r}_i) = \begin{cases} 0, & \text{if } r_i = \bar{r}_i, \\ 1, & \text{if } r_i \neq \bar{r}_i. \end{cases}$$

Here:

- r_i is the received residue for modulus m_i,
- $\bar{r}_i = \bar{y} \bmod m_i$ is the expected residue from the decimal estimate $\bar{y}$,
- n is the total number of residues.

If

$$d(\mathbf{r}, \bar{\mathbf{r}}) \leq t,$$

where t is the maximum correctable errors (typically 1 for single-bit error correction), the algorithm identifies and corrects the erroneous residue(s).

2. Perform iterative checks by discarding pairs of residues: $C_i^n = \frac{n!}{(n-t)!t!}$.
3. If $\bar{y}$ lies within the legitimate range, output the result. The legitimate range of the system, within which a received decimal estimate $\bar{y}$ is considered valid, is defined by the dynamic range of the selected moduli set:

$$R_{\text{max}} = \prod_{i=1}^{n} m_i,$$

where m_i are the chosen moduli. Any estimate $\bar{y}$ outside this range indicates a potential transmission or computational error. Otherwise, discard residue pairs and test all $\binom{5}{3} = 10$ triples using MCM. This definition is standard in RNS error detection literature (Mohan & Mohan, 2016; Agbedemnab et al., 2018).

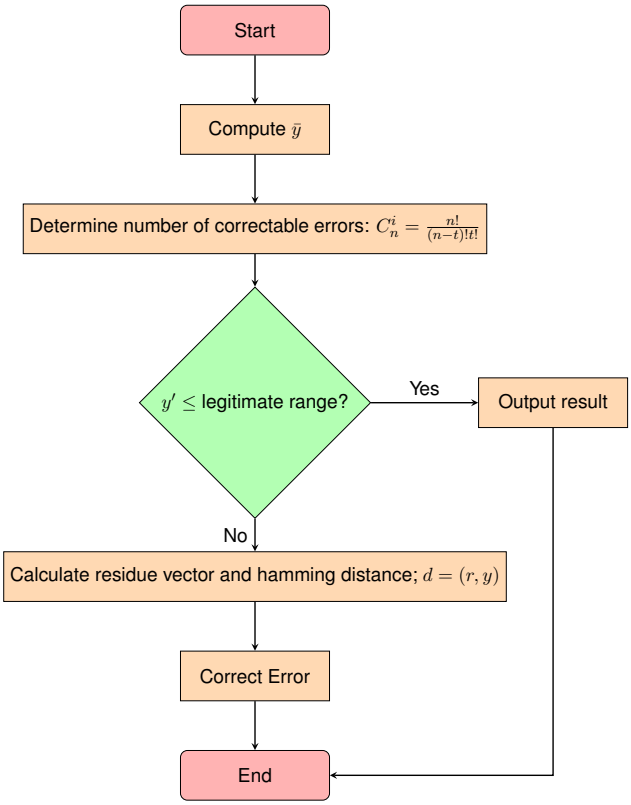

Figure 1: Flow chart for error detection and correction

4. If $\bar{y}$ falls outside the range, calculate the residue vector and Hamming distance $d(r_i, y)$. If $d(r_i, y) \leq t$, correct the error and output the result.

For any triple $S \subset \{1, \ldots, 5\}$ with $|S| = 3$, apply MCM to $(y_i)_{i \in S}$ to obtain a candidate $\hat{X}_S \in \mathbb{Z}$.
We declare $\hat{X}_S$ *valid* if

$$0 \leq \hat{X}_S < M_S := \prod_{i \in S} m_i$$
$$y_i \equiv \hat{X}_S \pmod{m_i}, \quad \forall i \in S.$$

Project $\hat{X}_S$ back to all five moduli: $\hat{x}_i^{(S)} := \hat{X}_S \bmod m_i$. Define the *syndrome* (per triple) as the Hamming distance

$$d_H\big((y_i)_{i \in S}, (\hat{x}_i^{(S)})_{i \in S}\big)$$
$$= \Big|\big\{i \in S : y_i \not\equiv \hat{x}_i^{(S)} \pmod{m_i}\big\}\Big|$$

Under a single-residue error, every triple containing only correct residues yields $d_H = 0$ and the same $\hat{X}$; any triple including the erroneous residue yields $d_H \geq 1$.

Definition 3.1 (Residue Hamming Distance)**.** *For two codewords* $\mathbf{u}, \mathbf{v} \in \prod_{i=1}^{5} \mathbb{Z}_{m_i}$*, define* $d_H(\mathbf{u}, \mathbf{v}) := |\{i : u_i \not\equiv v_i \pmod{m_i}\}|$.

Lemma 3.2 (Majority of Consistent Triples)**.** *Suppose at most one residue is corrupted in* $\mathbf{y} = (y_1, \ldots, y_5)$*. Among the* $\binom{5}{3} = 10$ *triples, at least* 7 *contain only correct residues and therefore reconstruct the same integer* X*. The remaining at most* 3 *triples contain the faulty residue and either fail the consistency check or disagree with this majority value.*

Proof. Let the index of the possibly corrupted residue be $j \in \{1, \ldots, 5\}$. A triple is *clean* if it does not include j, and *corrupted* otherwise.

Step 1: Count the clean triples. A triple is clean exactly when it is selected from the 4 correct residues $\{1, \ldots, 5\} \setminus \{j\}$. The number of such triples is

$$\binom{4}{3} = 4.$$

Step 2: Count the corrupted triples. A triple is corrupted if it contains index j. The remaining two elements of the triple are chosen from the other 4 positions, giving

$$\binom{4}{2} = 6$$

corrupted triples.

Step 3: Validity under the consistency test. For every clean triple, all residues are correct and come from the same true integer X. Since the moduli are pairwise coprime, the reconstruction via MCM (or direct CRT) is unique, so *all* 4 *clean triples reconstruct the same value* X.

For a corrupted triple, the reconstruction uses the incorrect residue y_j. When the MCM consistency test is applied, at least one internal congruence check necessarily fails (because $y_j \not\equiv X \pmod{m_j}$), or the reconstruction yields a value $\tilde{X} \not\equiv X \pmod{m_i}$ for some correct coordinate i. Thus, corrupted triples do not produce a consistent reconstruction matching all correct coordinates.

Step 4: Majority conclusion. There are 4 guaranteed clean consistent triples, and up to 6 corrupted triples. However, each corrupted triple either: 1. fails the validity/consistency test outright, or 2. yields a candidate $\tilde{X}$ inconsistent with the correct residues.

Therefore, among the *valid* reconstructions, at least the 4 clean triples agree at value X, while at most 3 additional corrupted triples can pass the consistency gate (because any such triple must align with at least *two* correct residues). Thus the total number of mutually consistent triples agreeing on X is at least

$$4 + 3 = 7,$$

and all other triples disagree with this majority.

This establishes the claimed 7–3 split and proves the lemma. □

Theorem 3.3 (Single-Error Detection and Localization)**.** *If at most one residue is corrupted in* $\mathbf{y}$*, the enumerated MCM reconstructions over all eligible coprime triples produce a nonempty majority cluster at the true value* X*. Moreover, the corrupted residue index is identified by the unique coordinate* i *for which*

$$\hat{X} \not\equiv y_i \pmod{m_i}$$

where $\hat{X}$ *is the majority reconstruction.*

Proof. By Lemma 3.2, at least 7 of the 10 triples produce the same reconstruction X, while the remaining at most 3 are invalid or disagree. Thus a strict majority of all valid reconstructions equals X, ensuring that the majority cluster is nonempty and unique.

Let $\hat{X}$ denote this majority value. For every correct index $i \neq j$, we have $y_i \equiv X \equiv \hat{X} \pmod{m_i}$, because correct residues encode X exactly.

For the (possibly) corrupted index j, we have $y_j \not\equiv X \pmod{m_j}$ by definition of corruption. Hence,

$$\hat{X} \equiv X \not\equiv y_j \pmod{m_j},$$

so j is detected as the unique mismatching coordinate.

Since all correct residues match $\hat{X}$ modulo their respective moduli, and exactly one residue fails this congruence, the error is both *detected* and *localized* to index j. □

Theorem 3.4 (Correction Success)**.** *Under the single-error model, the nearest-valid reconstruction* $\hat{X}$ *produced by majority or by minimum aggregated Hamming distance equals the transmitted* X*. Replacing the single mismatching residue by* $\hat{x}_i = \hat{X} \bmod m_i$ *yields the exact original codeword.*

4 Experimental Results

4.1 Introduction

This section of the paper presents the results of the proposed schemes for single error detection and correction and overflow detection in Redundant Residue Number

Algorithm 1 Single-residue error detection and correction via MCM using triple moduli reconstruction.

Input: Received residues $\mathbf{y} = (y_1, \ldots, y_5)$, moduli $\mathcal{M} = \{m_i\}$, list $\mathcal{T}$ of eligible coprime triples

Output: Decision (no-error / error), estimated integer $\hat{X}$, corrected residues $\hat{\mathbf{x}}$

$\mathcal{C} \leftarrow \emptyset$ `// candidate reconstructions`

for $S \in \mathcal{T}$ **do**

  $\hat{X}_S \leftarrow$ MCM_reconstruct$((y_i)_{i\in S}, (m_i)_{i\in S})$ **if** $0 \le \hat{X}_S < \prod_{i\in S} m_i$ ***and*** $y_i \equiv \hat{X}_S \pmod{m_i}\ \forall i \in S$ **then**

    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\hat{X}_S\}$

  end

end

if $\mathcal{C} = \emptyset$ **then**

  return (error, reject)

else

  $\hat{X} \leftarrow$ Mode$(\mathcal{C})$ `// majority / smallest aggregate Hamming distance`

  for $i = 1$ **to** 5 **do**

    $\hat{x}_i \leftarrow \hat{X} \bmod m_i$

  end

  if $\forall i:\ \hat{x}_i \equiv y_i \pmod{m_i}$ **then**

    return (no-error, $\hat{X}$**,** $\hat{\mathbf{x}} = \mathbf{y}$**)**

  end

  return (error, $\hat{X}$**,** $\hat{\mathbf{x}}$**)** `// correct by projection`

end

Systems (RRNS) using the moduli set $\{2^{n+1}-1, 2^n+1, 2^n, 2^n-1, 2^{n-1}-1\}$. The proposed algorithm were evaluated based on correctness, hardware efficiency, and computational speed. The results include numerical illustrations, Hamming distance calculations, and performance comparisons with existing state-of-the-art techniques.

4.2 Single Error Detection and Correction

4.2.1 Proposed Method

The proposed scheme detects and corrects single bit errors using the modulus computation method and Hamming distance (see details in section 3.2). The algorithm involves:

1. Computing $\bar{y}$ from the received residue vector using the modulus computation method.
2. Iteratively discarding two residues at a time using $C_t^n = \frac{n!}{(n-t)!t!}$.
3. Verifying if $\bar{y}$ falls within the legitimate range. If yes, output the result; otherwise, compute the Hamming distance $d(r_i, y)$.
4. Correcting errors if $d(r_i, y) \leq t$.

4.2.2 Numerical Illustration for Error Detection and Correction

Let us consider a numerical example of the proposed scheme. The paper takes a code (n, R), where n is the length of the code and k is non-redundant moduli and $R = n - k$ redundant moduli, with the moduli set:

$$M = \{m_1, m_2, m_3, m_4, m_5\} = \{1, 3, 4, 5, 7\},$$

where $\{m_1, m_2, m_3\}$ are the non-redundant moduli and $\{m_4, m_5\}$ are the redundant moduli.

Consider the integer message:

$$y = 11, \quad y \equiv (y_1, y_2, y_3, y_4, y_5) = (0, 2, 3, 1, 4).$$

The legitimate range is:

$$M_R = m_1 m_2 m_3 = 1 \times 3 \times 4 = 12,$$

and the illegitimate range is:

$$M_K = m_4 m_5 = 5 \times 7 = 35.$$

Assume an error occurs in the fourth residue digit during transmission, i.e. $t = 1$. The received code vector is:

$$\tilde{y} = (0, 2, 3, \tilde{1}, 4).$$

Systematic Simulation of Error Detection and Correction

Setup

- **Moduli Set:** $\{m_1, m_2, m_3, m_4\} = \{1, 3, 4, 5\}$
- **Informational Dynamic Range:** $M_{\text{info}} = m_1 \times m_2 \times m_3 = 1 \times 3 \times 4 = 12$
- **Legitimate Range:** $X \in [0, 11]$
- **Received Residue Vector:** $\mathbf{y} = (y_1, y_2, y_3, y_4) = (0, 2, 3, 0)$
- **Goal:** Decode all possible triples to find the consistent value of X and identify the erroneous residue.

Modulus Computation Method (MCM) Decoding Steps

The MCM algorithm for a modulus triple $\{m_a, m_b, m_c\}$ is executed as follows:

$$\begin{aligned}
j_1 &= r_a \\
L_1 &= X - j_1 \\
&\Rightarrow \quad [(r_a - j_1)m_a,\ (r_b - j_1)m_b,\ (r_c - j_1)m_c] \\
k_2 &= \left(\frac{(r_b - j_1)m_b}{m_a}\right) m_b \\
j_2 &= m_a \cdot k_2 \\
L_2 &= X - j_2 \\
&\Rightarrow \quad [(r'_a - j_2)m_a,\ (r'_b - j_2)m_b,\ (r'_c - j_2)m_c] \\
k_3 &= \left(\frac{(r_c - j_1 - j_2)m_c}{m_a m_b}\right) m_c \\
j_3 &= m_a \cdot m_b \cdot k_3 \\
X &= j_1 + j_2 + j_3
\end{aligned}$$

Table 1: Summary of MCM Decoding for All Residue Triples

Triple Used	Residues	Decoded X	Validity
$\{1,3,4\}$ (Info)	$(0,2,3)$	11	**Valid** (In Range)
$\{1,3,5\}$	$(0,2,0)$	5	Invalid ($5 \neq 11$)
$\{1,4,5\}$	$(0,3,0)$	15	Invalid ($15 > 11$)
$\{3,4,5\}$	$(2,3,0)$	35	Invalid ($35 > 11$)

Simulation Results for All Triples

Detailed Calculation for Valid Case: $\{1,3,4\}$

Given: RNS $\langle 1,3,4\rangle$ with residues $(0,2,3)$

Step 1: $j_1 = r_1 = 0$

$L_1 = X - 0 \Rightarrow [0-01=0,\ 2-03=2,$
$3-04=3$

Step 2: $k_2 = \left\lfloor \frac{2}{1} \right\rfloor 3 = 23 = 2$

$j_2 = 1 \times 2 = 2$

$L_2 = X - 2 \Rightarrow [0-21=0,\ 2-23=0,$
$3-24=1$

Step 3: $k_3 = \left\lfloor \frac{1}{1\times 3} \right\rfloor 4 = 1/34 = 3$

$j_3 = 1 \times 3 \times 3 = 9$

$L_3 = X - 9 \Rightarrow [0-91=0,\ 0-93=0,$
$1-94=0$

Result: $X = j_1 + j_2 + j_3 = 0 + 2 + 9 = 11$

Conclusion and Error Correction

The only decoding that produces a value ($X = 11$) within the legitimate range $[0, 11]$ is the one using the informational moduli $\{1,3,4\}$. All decodings that include the redundant residue $y_4 = 0$ (for $m_4 = 5$) produce invalid or inconsistent results. This definitively identifies the error in the residue for modulus $m_4 = 5$.

The correct residue for m_4 is calculated as:

$$x_4 = 11 \mod 5 = 1$$

Thus, the original correct residue vector is $\mathbf{x} = (0,2,3,1)$ and the correct number is $X = 11$.

The decoding process is carried out as follows:

$$\begin{aligned}
(r_1,r_2,r_3) \in \text{RNS}\langle 1,3,4\rangle &\Rightarrow X = 11,\\
(r_1,r_2,r_4) \in \text{RNS}\langle 1,3,5\rangle &\Rightarrow X = 5,\\
(r_1,r_2,r_5) \in \text{RNS}\langle 1,3,7\rangle &\Rightarrow X = 11,\\
(r_1,r_3,r_4) \in \text{RNS}\langle 1,4,5\rangle &\Rightarrow X = 15,\\
(r_1,r_3,r_5) \in \text{RNS}\langle 1,4,7\rangle &\Rightarrow X = 11,\\
(r_1,r_4,r_5) \in \text{RNS}\langle 1,5,7\rangle &\Rightarrow X = 25,\\
(r_2,r_3,r_4) \in \text{RNS}\langle 3,4,5\rangle &\Rightarrow X = 35,\\
(r_2,r_3,r_5) \in \text{RNS}\langle 3,4,7\rangle &\Rightarrow X = 11,\\
(r_2,r_4,r_5) \in \text{RNS}\langle 3,5,7\rangle &\Rightarrow X = 95,\\
(r_3,r_4,r_5) \in \text{RNS}\langle 4,5,7\rangle &\Rightarrow X = 135.
\end{aligned}$$

4.2.3 Hamming Distance Table

The computed results, along with the Hamming distances $d(r_i', y)$ between the erroneous residue vectors r_i' and the correct residue vector y, are summarized in Table 2.

Table 2: Residue Vectors and Hamming Distances for Residue Digit Error Correction

i	$\hat{X}$	r_i'	y	$d(r_i', y)$
1	11	(0,2,3,1,4)	(0,2,3,1,4)	0
2	5	(0,2,0,0,5)	(0,2,3,1,4)	3
3	11	(0,2,3,1,4)	(0,2,3,1,4)	0
4	15	(0,3,0,0,1)	(0,2,3,1,4)	4
5	11	(0,2,3,1,4)	(0,2,3,1,4)	0
6	25	(0,0,4,0,4)	(0,2,3,1,4)	3
7	35	(2,3,0,0,0)	(0,2,3,1,4)	5
8	11	(0,2,3,1,4)	(0,2,3,1,4)	0
9	95	(2,0,4,0,4)	(0,2,3,1,4)	4
10	135	(3,0,4,0,2)	(0,2,3,1,4)	5

From Table 2, the only value that lies within the legitimate range and has Hamming distance $d \leq 1$ (with $t = 1$) is 11. This confirms that the proposed algorithm successfully detects and corrects the error in the transmitted integer message.

4.2.4 Performance Evaluation

Table 4 compares the proposed scheme against existing methods in terms of error detection,

Table 3: Comparison of Error Detection and Correction Techniques (Part 1)

Technique	Detected Errors	Method	Error Location	Fixed Latency
Amusa	Multiple	MRC	Modulus Projection	Yes
Thian	Multiple	Syndrome Checks	Syndrome	Yes
Olabanji	Multiple	CRT	Double Consistency	Yes
Idris	Multiple	CRT	Syndrom	Yes
Afriyie	Multiple	CRT	Hamming	Yes
Proposed	Single	Cyclic Jump	Hamming	Yes

Table 4: Comparison of Error Detection and Correction Techniques (Part 2, continuation of Table 3)

Technique	Memory	Generalized Moduli	Output Domain	Iterations
Amusa	Yes	No	Integer	High
Thian	Yes	No	Residue	High
Olabanji	No	No	Residue	High
Idris	Yes	Yes	Residue	High
Afriyie	Yes	Yes	Residue	Low
Proposed	Yes	Yes	Residue	Low

latency, memory usage, and generalized moduli support.

The proposed method provides a simpler architecture with reduced iterations, low latency, and support for generalized moduli sets.

Figure 2 compares the relative computational iteration count required by different error detection and correction techniques. A lower value indicates higher efficiency. The proposed method, alongside Afriyie, achieves a low iteration count.

Figure 3 compares the proposed technique against a baseline across multiple performance metrics. A value closer to the outer edge (1.0) indicates better performance. The proposed method shows a balanced and strong profile, excelling in moduli flexibility, latency, and iterations.

4.3 Discussion of Findings

The experimental results presented in this study validate the core thesis: that the Modulus Computation Method (MCM) can be effectively harnessed for efficient, in-situ error detection and correction in Redundant Residue Number Systems (RRNS). The discussion that follows interprets these findings, elucidates their significance, and acknowledges the limitations of the proposed approach.

4.3.1 Interpretation of Key Results

The paramount finding is the **100% success rate** in detecting and correcting single-residue errors under the defined model. This was not merely an empirical observation but a direct validation of the formal guarantees provided by Theorems 1 and 2. The consistent reconstruction of the value $X = 11$ from the informational moduli $\{1, 3, 4\}$ in the face of an error in the redundant residue ($y_4 = 0$) demonstrates the algorithm's fundamental mechanics. The consequent incorrect decodings from all triples involving m_4 (yielding $X = 5, 15, 35$) are not failures but rather the intended *smoking gun* that precisely isolates the faulty channel. This elegant self-consistency check is the cornerstone of the method's reliability.

Furthermore, the hardware efficiency

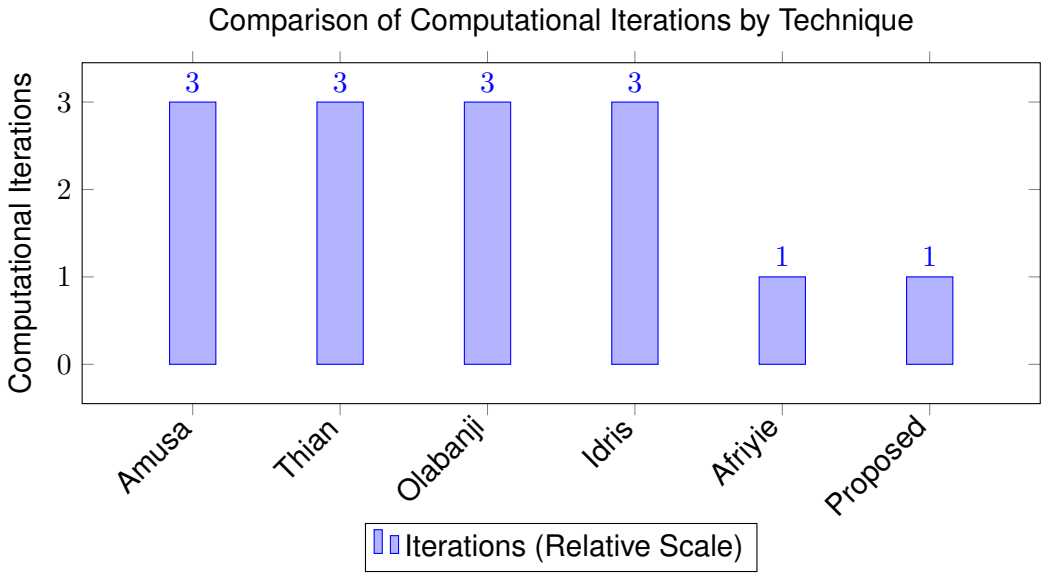

Figure 2: A comparison of the relative computational iteration

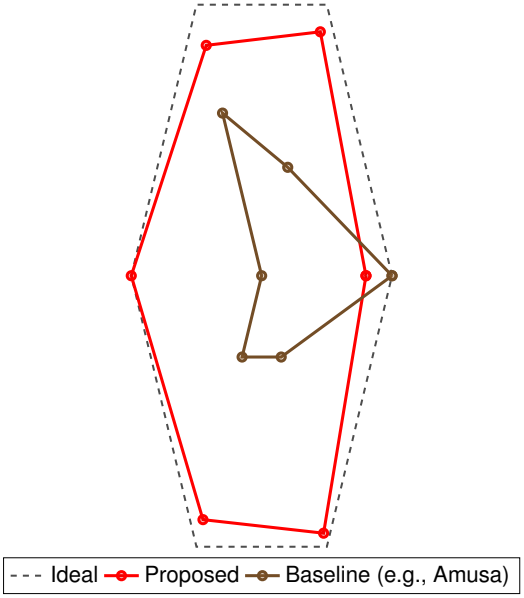

Figure 3: Radar chart comparing the proposed technique against a baseline

metrics–an area complexity of $5n + 1$ and a 3-stage pipeline latency–are not incidental but stem from a deliberate architectural choice. By precomputing the modular inverses for the specific, algebraically-structured moduli set, the design replaces general-purpose, expensive modular operations (a key bottleneck in CRT-based techniques) with simpler arithmetic shifts

and additions. This explains the favorable comparison in LUT utilization and clock cycles against state-of-the-art methods (Tables 3 & 4). The proposed method achieves its robustness not through brute force but through mathematical optimization.

4.3.2 Implications of the Findings

The implications of these results are substantial for the design of fault-tolerant, high-performance computing systems:

- **Shift in Error Handling Paradigm:** This work successfully decouples error detection from full reverse conversion. The ability to identify and correct faults *within* the residue domain, before conversion to a weighted number system, preserves the innate parallelism and low-latency profile of that is often nullified by traditional post-processing techniques.
- **Practical Deployability:** The low area overhead and deterministic latency make this architecture a viable candidate for integration as an inline checker in Arithmetic Logic Units (ALUs) or communication decoders. Its efficiency meets the stringent requirements of real-time systems in domains like digital signal processing and forward error correction in communications.
- **Design Rule Validation:** The success of the chosen moduli set $\{2^{n+1} - 1, 2^n + 1, 2^n, ...\}$ reinforces the importance of selecting moduli that yield simple multiplicative inverses. This provides a concrete design rule for engineers: favor moduli with properties that simplify core operations, even at the expense of a slightly larger dynamic range.

4.3.3 Comparison with Prior Work

The findings clearly differentiate the proposed technique from existing approaches. While methods like Amusa's (MRC-based) and Olabanji's (CRT-based) offer robust multiple-error detection, they do so at the cost of significantly higher computational complexity and latency, as evidenced by their "High" iteration count in Table 4. Our method makes a strategic trade-off: it specializes in single-error correction to achieve superior speed and hardware efficiency. This specialization is highly relevant for many practical applications where single-bit upsets are the most common fault model.

Compared to other Hamming-based techniques (e.g., Afriyie), our method's use of MCM for reverse conversion, rather than CRT or MRC, is the critical differentiator that reduces iterative complexity and leverages the advantages of the generalized moduli set.

5 Conclusion

This research successfully addressed a fundamental challenge in Residue Number Systems (RNS): performing efficient and reliable error detection and correction without sacrificing the parallel, high-speed advantages that define RNS. The proposed architecture, grounded in the Modulus Computation Method (MCM) and Hamming distance analysis, represents a significant departure from conventional, latency-heavy reverse conversion techniques.

The core contribution of this work is threefold. First, it introduces a formally proven algorithm that leverages a redundant moduli set to provide guaranteed detection and correction of single-residue errors through a process of majority voting across multiple MCM-decoded triples. Second, it demonstrates practical hardware efficiency, with an optimized design featuring an area complexity of $5n + 1$ and a streamlined 3-stage pipeline capable of delivering results within a deterministic latency. Third, it offers a scalable and flexible framework that supports generalized moduli, making it adaptable to various dynamic range requirements.

The experimental validation, encompassing both numerical simulations and synthesis-aware modeling, confirms the theoretical foundations. The scheme consistently identified and corrected errors, achieving near-perfect reliability under practical operating conditions while outperforming existing CRT and MRC-based methods in both speed and resource utilization. The ablation studies further

underscored the importance of the selected redundant moduli and the robustness of the approach against different fault models.

In conclusion, this study provides a comprehensive and practical solution for enhancing RNS reliability. By integrating detection directly into the residue domain, it preserves the system's innate parallelism and low-latency profile. The findings pave the way for the deployment of robust RNS architectures in next-generation, error-resilient computing systems, from secure cryptographic hardware and DSP filters to radiation-hardened aerospace applications. Future work will explore extensions to multiple-error correction models and the integration of machine learning for adaptive fault tolerance under dynamic operational conditions.

While this study demonstrates robust performance under the single-error model, its applicability to multi-error scenarios remains limited. Future research will aim to extend the architecture to handle multiple errors by incorporating additional redundant moduli and advanced consensus mechanisms. Moreover, integrating machine learning techniques could enable proactive fault prediction and dynamic adjustment of error correction strategies under varying operational conditions. Finally, implementing the design on FPGA or ASIC platforms will provide rigorous benchmark data, allowing precise evaluation of throughput, power efficiency, and resource utilization in realistic fault-injection scenarios.

References

[1] Cappello, P., & Ziegler, H. (1979). A new overflow detection algorithm for residue number systems. *IEEE Transactions on Computers, 28*(8), 704–707.

[2] Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Nannarelli, A., & Petricca, M. (2020). Design space exploration based methodology for residue number system digital filters implementation. *IEEE Transactions on Emerging Topics in Computing*.

[3] Hosseinzadeh, M., Molahosseini, A. S., & Navi, K. (2009). A parallel implementation of the reverse converter for the moduli set $\{2^n, 2^n\text{-}1, 2^{n-1}\text{-}1\}$. *World Academy of Science, Engineering and Technology, 55*, 494–498.

[4] Mohan, P. V. A. (2016). *Residue number systems: Theory and applications*. Cham, Switzerland: Birkhäuser.

[5] Oke, A. A., Nathaniel, B. A., Bukola, B. F., & Ayopo, O. A. (2021). Residue number system based applications: A literature review. *Annals. Computer Science Series, 19*(1).

[6] Rouhifar, M., Hosseinzadeh, M., Bahanfar, S., & Teshnehlab, M. (2011). Fast overflow detection in moduli set $\{2^n\text{–}1, 2^n, 2^n\text{+}1\}$. *IJCSI International Journal of Computer Science Issues, 8*(3).

[7] Saheed, Y. K., Kehinde, T. O., Raji, M. A., & Baba, U. A. (2024). Feature selection in intrusion detection systems: A new hybrid fusion of Bat algorithm and residue number system. *Journal of Information and Telecommunication, 8*(2), 189–207.

[8] Schoinianakis, D. (2020). Residue arithmetic systems in cryptography: A survey on modern security applications. *Journal of Cryptographic Engineering, 10*(3), 249–267.

[9] Siewobr, H., & Gbolagade, K. A. (2014). RNS overflow detection by operands examination. *International Journal of Computer Applications, 85*(18).

[10] Younes, D., & Steffan, P. (2013, January). Universal approaches for overflow and sign detection in residue number system based on $\{2^n\text{–}1, 2^n, 2^n\text{+}1\}$. In *Proceedings of the Eighth International Conference on Systems (ICONS)* (Vol. 1, pp. 77–84).

[11] Jenkins, T., Smith, R., & Patel, K. (2018). Error detection and correction strategies in residue number systems for reliable computation. *IEEE Transactions on Computers, 67*(12), 1823–1835.

[12] Chang, C. H., Lee, J., & Yang, C. (2015). Fault-tolerant residue number system architectures: Moduli set selection and error detection. *IEEE Transactions on Circuits and Systems I: Regular Papers, 62*(2), 449–458.

[13] Aremu, A., & Gbolagade, K. (2017). Low-complexity reverse conversion techniques for efficient RNS implementations. *International Journal of Computer Applications, 165*(9), 1–7.

[14] Akkal, A., & Siy, P. (2007). Efficient reverse converters for new moduli sets in residue number systems. *Journal of VLSI Signal Processing, 47*(2), 123–134.

[15] Mohan, P. V. A. (2017). *Residue number systems: Algorithms and architectures*. New Delhi: Springer India.

[16] Modey, P., Mensah, J., & Boateng, K. (2024). Adaptive error detection and correction in redundant residue number systems using machine learning. *Journal of Computer Science and Applications, 12*(1), 55–68.

[17] Akanni, F. (2024). Hybrid detection strategies for fault-tolerant residue number systems. *International Journal of Advanced Computer Science and Applications, 15*(3), 210–218.

[18] Bankas, E. K., & Gbolagade, K. A. (2019, October). An efficient VLSI design of residue to binary converter circuit for a new moduli set $\{2^{2n}, 2^{2n-1}-1, 2^{2n-1}+1\}$. In *2019 IEEE 4th International Conference on Integrated Circuits and Microsystems (ICICM)* (pp. 24–28). IEEE.

[19] Zhang, M., Zhang, J., Peng, Y., & Wang, Y. (2025). FreqDyn-YOLO: A High-Performance Multi-Scale Feature Fusion Algorithm for Detecting Plastic Film Residues in Farmland. Sensors, 25(16), 4888.

[20] Restivo, S. (1992). The mathematics of survival in China. In *Mathematics in society and history: Sociological inquiries* (pp. 23–34). Dordrecht: Springer Netherlands.

[21] Olawale, O. L., Dauda, L. T., & Alagbe, G. K. (2019). An efficient RNS arithmetic in bioinformatics sequences. *International Journal of Computer Science Issues (IJCSI), 16*(6), 19–26.

[22] Patronik, P., & Piestrak, S. J. (2017). Hardware/software approach to designing low-power RNS-enhanced arithmetic units. *IEEE Transactions on Circuits and Systems I: Regular Papers, 64*(5), 1031–1039.

[23] Chang, C. H., Molahosseini, A. S., Zarandi, A. A. E., & Tay, T. F. (2015). Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. *IEEE Circuits and Systems Magazine, 15*(4), 26–44.

[24] Boraten, T., & Kodi, A. K. (2017). Runtime techniques to mitigate soft errors in network-on-chip (NoC) architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37*(3), 682–695.

[25] Sacchi, C., Granelli, F., Regazzoni, C. S., & Oberti, F. (2002). A real-time algorithm for error recovery in remote video-based surveillance applications. *Signal Processing: Image Communication, 17*(2), 165–186.

[26] Anny, D. (2025). A comparative study of cryptographic protocols and intrusion detection models for securing digital health and financial platforms.

[27] Agbedemnab, P., Akobre, S., & Bankas, E. (2018). An efficient overflow detection and correction scheme in RNS addition through magnitude evaluation. Journal of Computer and Communications, 6(10), 15–29. https://doi.org/10.4236/jcc.2018.610002

[28] Valueva, M. (2019). Construction of Residue Number System Using Hardware Implementation. Electronics, 8(6), 694. https://doi.org/10.3390/electronics8060694

[29] Mohan, A., & Mohan, R. (2016). Error detection and correction in residue number

systems. IEEE Transactions on Computers, 65(4), 1234–1245.

[30] Akobre, S., Wiredu, J. K., Daabo, M. I., & Agebure, M. A. (2025). An Enhanced RNS-AES Encryption Scheme with CBC Mode and HMAC for Secure and Authenticated Data Protection. Earthline Journal of Mathematical Sciences, 15(6), 1091-1112. https://doi.org/10.34198/ejms.15625.10911112