Journal of Computer and Communications, 2018, 6, 15-29
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# An Efficient Overflow Detection and Correction Scheme in RNS Addition through Magnitude Evaluation

**Peter Awon-natemi Agbedemnab, Stephen Akobre, Edem Kwedzo Bankas**

Department of Computer Science, University for Development Studies, Navrongo, Ghana
Email: nabpiero@gmail.com, pagbedemnab@uds.edu.gh, sakobre@uds.edu.gh, eb433007@ohio.edu

**How to cite this paper:** Agbedemnab, P.A., Akobre, S. and Bankas, E.K. (2018) An Efficient Overflow Detection and Correction Scheme in RNS Addition through Magnitude Evaluation. *Journal of Computer and Communications*, **6**, 15-29.
https://doi.org/10.4236/jcc.2018.610002

**Received:** August 24, 2018
**Accepted:** October 22, 2018
**Published:** October 25, 2018

## Abstract

Number Systems are media for representing numbers; the popular ones being the Weighted Number Systems (WNS), which sometimes propagate carries during arithmetic computations. The other category, Un-Weighted Number Systems, of which the Residue Number System (RNS) belongs, do not carry weights but have not yet found widespread usage in general purpose computing as a result of some challenges; one of the main challenges of RNS is overflow detection and correction. The presence of errors in calculated values due to such factors as overflow means that systems built on this number system will continue to fail until serious steps are taken to resolve the issue. In this paper, a scheme for detecting and correcting overflow during RNS addition is presented. The proposed scheme used mixed radix digits to evaluate the magnitude of the addends in order to detect the occurrence of overflow in their sum. The scheme also demonstrated a simplified technique of correcting the overflow in the event that it occurs. An analysis of the hardware requirements and speed limitations of the scheme showed that it performs considerably better in relation to similar state of art schemes.

## Keywords

Number Systems, Weighted Number Systems (WNS), Residue Number System (RNS), Overflow Detection, Overflow Correction, Faults, Mixed Radix Digits (MRDs)

## 1. Introduction

The Residue Number System (RNS) has gained prominence in recent years due to its seemingly inherent features such as parallelism and carry-propagation free arithmetic computations. Notwithstanding the fact that, RNS is currently being applied in Digital Signal Processing (DSP) intensive computations like digital filtering, convolutions, correlations, Discrete Fourier Transform (DFT) computations, Fast Fourier Transform (FFT) computations and Direct Digital Frequency synthesis [1] [2] [3]; researchers in the area are still working hard around the clock in order that the RNS becomes a general purpose processor. These efforts have not completely come to fruition because of challenges, including conversion to and from RNS and decimal/binary number systems, the moduli sets to use, overflow detection and correction, magnitude evaluation, and scaling.

An RNS number *X*, is represented as $x_i = \left|X\right|_{m_i}$, where $m_i = \{m_1, m_2, \cdots, m_n\}$, a set of pairwise relatively prime integers such that $m_1 \neq m_2 \neq \cdots \neq m_n$ and $\gcd(m_1, m_2), \cdots, \gcd(m_{n-1}, m_1) = 1$. The residue set $x_i = [x_1, x_2, \cdots, x_n]$ is uniquely represented provided $X$ lies within the legitimate range $[0, M-1]$ where $M = \prod_{i=1}^{n} m_i$ is the Dynamic Range (DR) for the chosen moduli set. Let *X* and *Y* be two different integers within the DR, if $X \odot Y$, ($\odot$ are the arithmetic operations $+, -, \times, \div$), results in a value that is outside the legitimate range, then overflow is said to have occurred.

Overflow in general computing occurs if a calculated value is greater than its intended storage location in memory [4] [5]; this relates to the DR in RNS which situation usually arises during addition and multiplication operations and failure to detect it will lead to improper or wrong representation of numbers and calculated results. Thus detecting overflow is one of the fundamental issues in the design of efficient RNS systems [6].

The conversion of an RNS number into its decimal/binary equivalent number (*a process called reverse conversion*) has long been mainly based on the Chinese Remainder Theorem (CRT) and the Mixed Radix Conversion (MRC) techniques with few modifications being their variants of recent times. Whiles the former deals with the modulo-*M* operation, the later does not but computes sequentially which tends to reduce the complexity of the architecture. Computations can be done using the MRC as follows:

$$X = e_1 + e_2 m_1 + e_3 m_1 m_2 + \cdots + e_n m_1 m_2 \cdots m_{n-1} \tag{1}$$

where $e_i, i = 1, 2, \cdots, n$ are the Mixed Radix Digits (MRDs) and computed as follows:

$$\begin{aligned} e_1 &= x_1 \\ e_2 &= \left| (x_2 - e_1) \left| m_1^{-1} \right|_{m_2} \right|_{m_2} \\ e_3 &= \left| \left( (x_3 - e_1) \left| m_1^{-1} \right|_{m_3} - e_2 \right) \left| m_2^{-1} \right|_{m_3} \right|_{m_3} \\ &\vdots \\ e_n &= \left| \left( \cdots \left( (x_3 - e_1) \left| m_1^{-1} \right|_{m_n} - e_2 \right) \left| m_2^{-1} \right|_{m_n} - \cdots - e_{n-1} \right) \left| m_{n-1}^{-1} \right|_{m_n} \right|_{m_n} \end{aligned} \tag{2}$$

The MRDs $e_i$ are within the range $0 \le e_i \le m_i$, and a positive number, *X*, in the interval $[0, M]$ can be uniquely represented. The magnitude of a number can become crucial in the determination overflow in RNS. The sign of an RNS number is determined by partitioning *M* into two parts: $0 \le X < \lfloor M/2 \rfloor$ (*for positive integers*) and $\lfloor M/2 \rfloor \le X < M$ (*for negative integers*).

Recently, some techniques have been developed to detect overflow without necessarily completing the reverse conversion process; in [7], an algorithm to detect overflow in the moduli set $(2^n - 3, 2^n - 1, 2^n, 2^n + 1, 2^n + 3)$ by adding a redundant modulus 2 to this moduli set and making use of ROM and XOR gates was proposed. In [8], a method for detecting overflow in the moduli set $(2^n - 1, 2^n, 2^n + 1)$ based on group of numbers is presented where numbers within $[0, M - 1]$ are distributed among several groups. Then, by using the groupings, the scheme is able to diagnose in the process of addition of two numbers, whether overflow has occurred or not. The scheme in [3] evaluated the sign of the sum of two numbers *X* and *Y* and used it to detect overflow but adopted a residue-to-binary converter proposed by [9]. The scheme in [10] presented a scheme by an Operands Examination Method for overflow detection for the moduli set $(2^n - 1, 2^n, 2^n + 1)$ during RNS addition. All these schemes either relied on complete reverse conversion process as in the case of [3], or other costly and time consuming procedures such as base extension, group number and sign detection as in [8] and [10].

In this paper, a new technique for detecting and correcting overflow during the addition of two RNS numbers for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is presented; the technique evaluates the sign of an RNS number by performing a partial reverse conversion using the mixed radix conversion method. The sign of the addends is evaluated using only the MRDs, which is then used to detect the occurrence of overflow during RNS addition. The rest of the paper is organized as follows: Section 2 presents the proposed method, an anticipated hardware implementation (albeit theoretical) is presented in Section 3 with its realization in Section 4. Numerical illustrations are shown in Section 5 whiles the performance of the proposed scheme is evaluated in Section 6. The final part of this paper is the conclusion in Section 7.

## 2. Proposed Method

Given the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, where $m_1 = 2^n + 1$, $m_2 = 2^n$ and $m_3 = 2^n - 1$, then

$$M = 2^n (2^n + 1)(2^n - 1) \tag{3}$$

This implies

$$M/2 = 2^{n-1}(2^n + 1)(2^n - 1) = (2^n + 1)(2^{2n-1} - 2^{n-1}) \tag{4}$$

**Lemma 1:** Given the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, where $m_1 = 2^n + 1$, $m_2 = 2^n$ and $m_3 = 2^n - 1$ for every integer $n > 1$, the following hold true [10]:

$$\left|m_1^{-1}\right|_{m_2} = 1 \tag{5}$$

$$\left|m_2^{-1}\right|_{m_3} = 1 \tag{6}$$

$$\left|m_1^{-1}\right|_{m_3} = 2^{n-1} \tag{7}$$

Therefore, we can re-write (2) as;

$$\begin{aligned} e_1 &= x_1 \\ e_2 &= \left|(x_2 - e_1)1\right|_{2^n} = \left|x_2 - x_1\right|_{2^n} \\ e_3 &= \left|\left((x_3 - e_1)2^{n-1} - e_2\right)1\right|_{2^n-1} = \left|(x_3 - e_1)2^{n-1} - e_2\right|_{2^n-1} \end{aligned} \tag{8}$$

***Theorem* 1:** For the given moduli set, any integer $X \ge M/2$ if and only if

$$e_3 = 2^n - 2^{n-1} \tag{9}$$

or

$$e_3 = 2^n - 2^{n-1} - 1 \text{ AND } e_2 = 2^n - 2^{n-1} \tag{10}$$

for any $n > 1$.

***Proof:*** If it can be shown that by substituting (9) and (10) into Equation (4) that, $X \ge (2^n + 1)(2^{2n-1} - 2^{n-1})$ then, it implies $X \ge M/2$.

Assume (9) is true, then

$$\begin{aligned} X &= e_1 + (2^n + 1)e_2 + (2^n - 2^{n-1})2^n(2^n + 1) \\ &= e_1 + (2^n + 1)\left[e_2 + 2^{2n} - 2^{2n-1}\right] \\ &> (2^n + 1)(2^{2n-1} - 2^{n-1}), \forall n > 1 \end{aligned}$$

Also, assume (10) is true, then

$$\begin{aligned} X &\ge e_1 + (2^n - 2^{n-1})(2^n + 1) + (2^n - 2^{n-1} - 1)(2^{2n} + 2^n) \\ &= e_1 + (2^n + 1)\left[2^{2n} - 2^{2n-1} - 2^{n-1}\right] \\ &\ge (2^n + 1)(2^{2n-1} - 2^{n-1}), \forall n > 1 \end{aligned}$$ ■

Thus, from (9) and (10), it is possible to determine the sign of an RNS number *X*; whether $X \ge M/2$ (for a negative number) or $X < M/2$ (for a positive number).

The proposed method uses comparison by computing the MRDs of each of the addends to determine which half of the RNS range it belongs rather than performing a full reverse conversion. To detect overflow during addition of two addends $X$ and $Y$ based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, a single bit that indicates the sign of that addend is defined. Now, based on this bit, three cases will then be considered:

1) Overflow will definitely occur if both of the addends are equal to or greater than half of the dynamic range (*M*/2).

2) Overflow will not occur if both of the addends are less than *M*/2.

3) Overflow may or may not occur if only one of the addends is equal or greater than *M*/2 and will require further processing to determine whether over-

flow will occur or not.

Let the magnitude evaluation of the addends $(X, Y)$ be represented by $\beta$, such that if $\beta = 1$ or $\beta = 0$ represents a positive number or a negative number respectively as shown in Equation (11). The evaluation of the undetermined case in (3) is also represented by a single bit $\lambda$ in (12).

$$\beta = \begin{cases} 1; e_3 = 2^n - 2^{n-1} \\ 1; e_3 = 2^n - 2^{n-1} - 1 \text{ AND } e_2 = 2^n - 2^{n-1} \\ 0; \text{otherwise} \end{cases} \tag{11}$$

and,

$$\lambda = \begin{cases} 1; x_1 + y_1 \geq 2^n + 1 \\ 0; \text{otherwise} \end{cases} \tag{12}$$

The proposed method will then detect overflow as follows:

$$overflow = \begin{cases} 0; \beta_X + \beta_Y = 0 \\ 1; \beta_X \cdot \beta_Y = 1 \\ \lambda; \beta_X \oplus \beta_Y = 1 \end{cases} \tag{13}$$

where $(+, \cdot, \oplus)$ refer to the logical operations (OR, AND, XOR), respectively. For clarity, "1" means overflow occurs whilst "0" means no overflow.

### Correction Unit

Let $Z$ be the sum of the two addends. By substituting the individual MRDs for both addends ($X$ and $Y$), $Z$ can be obtained as follows;

$$\begin{aligned} Z &= X + Y \\ &= \left[e_1(X) + e_2(X)m_1 + e_3(X)m_1m_2\right] + \left[e_1(Y) + e_2(Y)m_1 + e_3(Y)m_1m_2\right] \\ &= \left(e_1(X) + e_1(Y)\right) + \left(e_2(X) + e_2(Y)\right)m_1 + \left(e_3(X) + e_3(Y)\right)m_1m_2 \end{aligned}$$

by letting $\psi_i = e_i(X) + e_i(Y)$, we shall have

$$Z = \psi_1 + \psi_2 m_1 + \psi m_1 m_2 \tag{14}$$

Thus by adding the individual MRDs of the two addends, we obtain the sum $Z$ according to (1) without having to compute separately for its MRDs. The value of $Z$ obtained from (14) is the correct result of the addition whether overflow occurs or not. In case of overflow occurrence, the redundant modulus is employed by shifting $M$ one bit to the left in order to accommodate the value.

## 3. Hardware Implementation

From Equation (8), the MRD's $e_1$, $e_2$ and $e_3$ can be represented in binary as;

$$e_1 = \underbrace{e_{1,n}e_{1,n-1}\cdots e_{1,1}e_{1,0}}_{n+1} \tag{15}$$

$$e_2 = \underbrace{e_{2,n-1}e_{2,n-2}\cdots e_{2,1}e_{2,0}}_{n} \tag{16}$$

$$e_3 = \underbrace{e_{3,n-1}e_{3,n-2}\cdots e_{3,1}e_{3,0}}_{n} \tag{17}$$

Equations (15) to (17) can further be simplified as follows;

$$e_1 = x_1 = \underbrace{x_{1,n}x_{1,n-1}\cdots x_{1,1}x_{1,0}}_{n+1} \tag{18}$$

$$\begin{aligned} e_2 &= \left|x_2 - x_1\right|_{2^n} = \left|x_2 + t_1\right|_{2^n} \\ &= \left|\underbrace{x_{2,n-1}x_{2,n-2}\cdots x_{2,1}x_{2,0}}_{n} + \underbrace{t_{1,n-1}t_{1,n-2}\cdots t_{1,1}t_{1,0}}_{n}\right|_{2^n} \\ &= \underbrace{e_{2,n-1}e_{2,n-2}\cdots e_{2,1}e_{2,0}}_{\substack{n \\ n}} \end{aligned} \tag{19}$$

where,

$$\begin{aligned} t_1 &= \left|-x_1\right|_{2^n} = \left|-\left(\underbrace{x_{1,n}x_{1,n-1}\cdots x_{1,1}x_{1,0}}_{n+1}\right)\right|_{2^n} = \left|-\left(\underbrace{x_{1,n-1}x_{1,n-2}\cdots x_{1,1}x_{1,0}}_{n}\right)\right|_{2^n} \\ &= \left|\underbrace{\overline{x}_{1,n-1}\overline{x}_{1,n-2}\cdots \overline{x}_{1,1}\overline{x}_{1,0}}_{n}\right|_{2^n} \end{aligned} \tag{20}$$

and

$$\begin{aligned} e_3 &= \left|2^{n-1}x_3 - 2^{n-1}e_1 - e_2\right|_{2^n-1} = \left|t_2 + t_3 + t_4\right|_{2^n-1} \\ &= \left|\underbrace{t_{2,n-1}\cdots t_{2,1}t_{2,0}}_{n} + \underbrace{t_{3,n-1}\cdots t_{3,1}t_{3,0}}_{n} + \underbrace{t_{4,n-1}\cdots t_{4,1}t_{4,0}}_{n}\right|_{2^n-1} \\ &= \underbrace{e_{3,n-1}e_{3,n-2}\cdots e_{3,1}e_{3,0}}_{n} \end{aligned} \tag{21}$$

where

$$t_2 = \left|2^{n-1}x_3\right|_{2^n-1} = \left|2^{n-1}\left(\underbrace{x_{3,n-1}x_{3,n-2}\cdots x_{3,1}x_{3,0}}_{n}\right)\right|_{2^n-1} = \underbrace{x_{3,0}x_{3,n-1}\cdots x_{3,2}x_{3,1}}_{n} \tag{22}$$

$$\begin{aligned} t_3 &= \left|-2^{n-1}x_1\right|_{2^n-1} = \left|-2^{n-1}\left(\underbrace{x_{1,n}x_{1,n-1}\cdots x_{1,1}x_{1,0}}_{n+1}\right)\right|_{2^n-1} \\ &= \left|-2^{n-1}\left(x_{1,n}\times 2^n + \underbrace{x_{1,n-1}\cdots x_{1,1}x_{1,0}}_{n}\right)\right|_{2^n-1} \end{aligned} \tag{23}$$

Since, $x_1$ is a number that is smaller than $2^n+1$, two cases are considered for $x_1$. First, when $x_1$ is smaller than $2^n$, and second, when $x_1$ is equal to $2^n$ [11]. If $x_{1,n}=0$, we have

$$t_{31} = \left|-2^{n-1}\left(\underbrace{x_{1,n-1}x_{1,n-2}\cdots x_{1,1}x_{1,0}}_{n}\right)\right|_{2^n-1} = \underbrace{\overline{x}_{1,0}\overline{x}_{1,n-1}\cdots \overline{x}_{1,2}\overline{x}_{1,1}}_{n} \tag{24}$$

Else if $x_{1,n}=1$, the following binary vector can be obtained as

$$t_{32} = \left|-2^{n-1}\times 2^n\left(\underbrace{00\cdots 0}_{n-1}x_{1,n}\right)\right|_{2^n-1} = 0\underbrace{11\cdots 1}_{n-1} \tag{25}$$

Therefore, $t_3$ is calculated as

$$t_3 = \begin{cases} t_{31}, & \text{if } x_{1,n} = 0 \\ t_{32}, & \text{if } x_{1,n} = 1 \end{cases} \tag{26}$$

And finally,

$$t_4 = \left|-e_2\right|_{2^n-1} = \left|-\underbrace{\left(e_{2,n-1}e_{2,n-2}\cdots e_{2,1}e_{2,0}\right)}_{n}\right|_{2^n-1} = \underbrace{\bar{e}_{2,n-1}\bar{e}_{2,n-2}\cdots\bar{e}_{2,1}\bar{e}_{2,0}}_{n} \tag{27}$$

Let $\gamma$ and $\omega$ represent the MRDs of the two integers $X$ and $Y$ respectively. Thus from equations (19) to (21), we have

$$\psi_i = \gamma_i + \omega_i \tag{28}$$

which implies

$$\psi_1 = \gamma_1 + \omega_1 = \underbrace{\gamma_{1,n}\gamma_{1,n-1}\cdots\gamma_{1,0}}_{n+1} + \overbrace{\omega_{1,n}\omega_{1,n-1}\cdots\omega_{1,0}}^{n+1} = \psi_{1,n}\psi_{1,n-1}\cdots\psi_{1,0} \tag{29}$$

$$\psi_2 = \gamma_2 + \omega_2 = \underbrace{\gamma_{2,n-1}\gamma_{2,n-2}\cdots\gamma_{2,0}}_{n+1} + \overbrace{\omega_{2,n-1}\cdots\omega_{2,0}}^{n+1} = \psi_{2,n-1}\psi_{2,n-2}\cdots\psi_{2,0} \tag{30}$$

finally,

$$\psi_3 = \gamma_3 + \omega_3 = \underbrace{\gamma_{3,n-1}\gamma_{3,n-2}\cdots\gamma_{3,0}}_{n+1} + \overbrace{\omega_{3,n-1}\cdots\omega_{3,0}}^{n+1} = \psi_{3,n-1}\psi_{3,n-2}\cdots\psi_{3,0} \tag{31}$$

and so, $Z$ is implemented as;

$$Z = z_1 + z_3 + z_4 = \underbrace{\overbrace{0\cdots0}^{2n}\underbrace{z_{1,n}\cdots z_{1,0}}_{n+1} + 0\underbrace{z_{3,3n-1}\cdots z_{3,0}}_{3n} + \overbrace{0\cdots0}^{n+1}\underbrace{z_{4,2n-1}\cdots z_{4,0}}_{2n}}_{3n+1} \tag{32}$$

where,

$$z_1 = \psi_1 \tag{33}$$

$$\begin{aligned} z_3 &= z_2 + 2^{2n}\psi_3 = \underbrace{\psi_{3,n-1}\cdots\psi_{3,0}}_{n}\overbrace{00\cdots0}^{2n} \bowtie \underbrace{z_{2,2n-1}\cdots z_{2,0}}_{2n} \\ &= z_{3,3n-1}z_{3,3n-2}\cdots z_{3,1}z_{3,0} \end{aligned} \tag{34}$$

$$\begin{aligned} z_2 &= 2^n\psi_2 + \psi_2 = \underbrace{\psi_{2,n-1}\cdots\psi_{2,0}}_{n}\overbrace{00\cdots0}^{n} \bowtie \underbrace{\psi_{2,n-1}\cdots\psi_{2,0}}_{n} \\ &= z_{2,2n-1}z_{2,2n-2}\cdots z_{2,1}z_{2,0} \end{aligned} \tag{35}$$

and,

$$z_4 = 2^n\psi_3 = \underbrace{\psi_{3,n-1}\cdots\psi_{3,1}\psi_{3,0}}_{n}\overbrace{00\cdots0}^{n} = z_{4,2n-1}z_{4,2n-2}\cdots z_{4,1}z_{4,0} \tag{36}$$

## 4. Hardware Realization

The hardware realization of the proposed scheme is divided into four parts as shown in **Figures 1-4**. First is the Partial Conversion Part (PCP) shown in **Figure 1**, which evaluates the MRDs based on (18), (19) and (21) with their parameters clearly defined according to (20) and (21) - (27). The PCP begins with an Operands Preparation Unit (OPU) which prepares the operands in (20), (22) and (26) by simply manipulating the routing of the bits of the residues. Also, an $n$-bit 2:1 Multiplexer (MUX) is used for obtaining (26). ADD1 is an $n$-bit Carry Propagate Adder (CPA) and is used to compute (19), meanwhile (21) is obtained by using an $(n-1)$-bit CPA as ADD2 whose save ($s_1$) and carry ($c_1$) are then added using ADD3 which is also an $(n-1)$-bit CPA. These MRDs are used to determine the sign of the RNS number in **Figure 2**. Thus, the critical path for the PCP unit is made up of one $(2^n)$ modulo adder and two $(2^n-1)$ modulo adders.

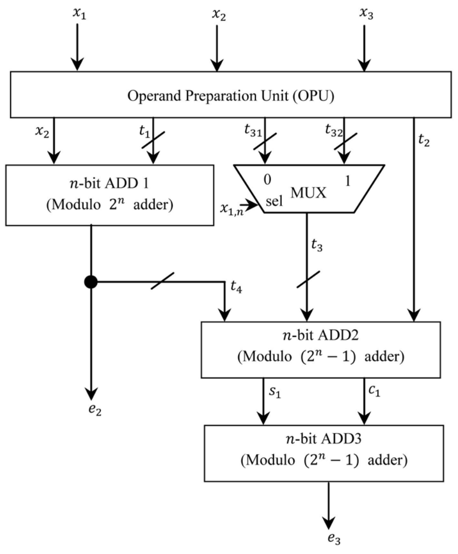

**Figure 1.** Partial conversion part (PCP).

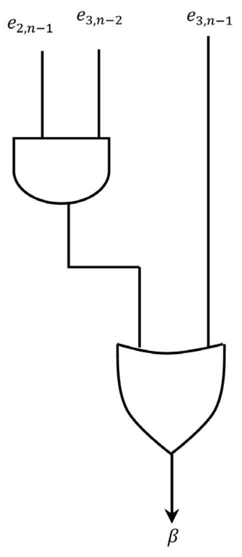

**Figure 2.** Magnitude evaluation part (MEP).

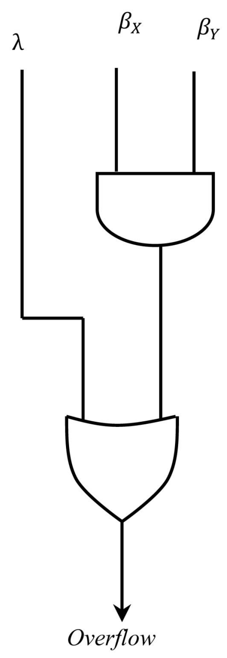

Figure 3. Overflow detection part (ODP).

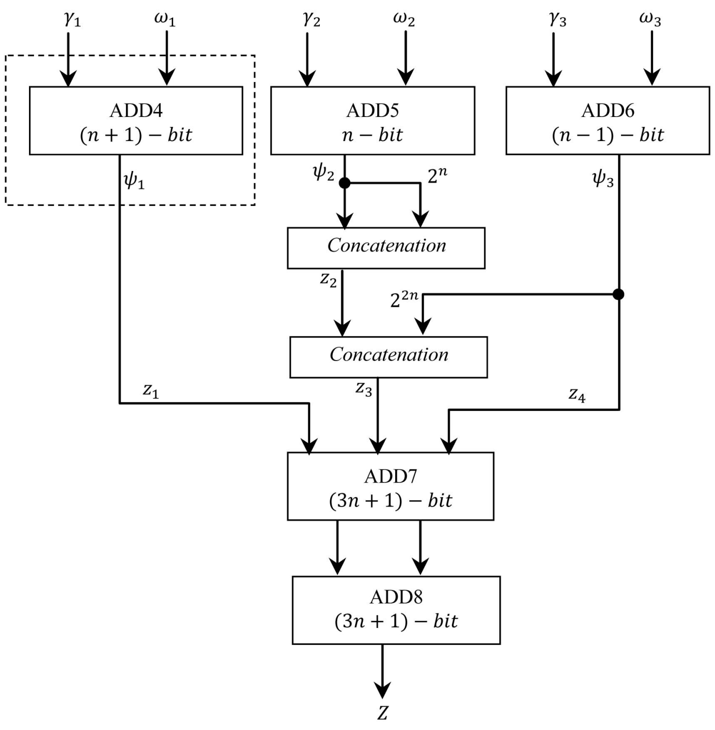

Figure 4. Overflow correction part (OCP).

Second, is the Magnitude Evaluation Part (MEP) shown in Figure 2, which evaluates whether an RNS number is positive or negative according to Equation (11). The MEP uses one AND gate and an OR gate. These are both two input monotonic gates. Next, is the Overflow Detection Part (ODP) which compares the sign bits of the two addends by using an AND gate according to (13) which is then ORed with the evaluated bit of the undetermined case in (12) as shown in **Figure 3**. This is where the scheme detects the occurrence of overflow during the addition of two numbers.

Lastly, in **Figure 4** is the Overflow Correction Part (OCP). The OCP evaluates the individual MRDs of the two addends separately to achieve the sum $Z$ in (14). This is done using five adders; four regular CPAs and one carry save adder (CSA). This is computed according to (29) - (36). ADD4, ADD5 and ADD6 add separately the MRDs $e_1, e_2$ and $e_3$ respectively for the two addends. The result of ADD4 is of importance because it is used in evaluating the undetermined case in (12). $z_2$ is a result of concatenation as well as $z_3$ which do not require any hardware. ADD7 is a CSA which computes the result of $z_1, z_2$ and $z_3$ whose save ($s_2$) and carry ($c_2$) are added using ADD 8 which is a CPA in order to get accurate sum $Z$ whether overflow occurs or not. The schematic diagrams for the proposed scheme are presented in **Figures 1-4**.

The area ($A$) and time ($D$) requirements of the proposed scheme are estimated based on the unit-gate model as used in [12] and [13] for fair comparison. In this model, each two-input monotonic gate such as AND, OR, NAND, NOR has area $A=1$ and delay $D=1$, each two-input gate XOR/XNOR has $A=D=2$, The area and delay of an inverter is a negligible fraction of a unit, and it is thus assumed to require zero units of area and delay [14]. A 2:1 multiplexer has an area $A=3$ and delay $D=2$; A full adder has an area of seven gates and a delay of four gates but a CSA has a constant delay. Also, the adder requirements based on this model as presented in [14] is adopted for the comparison since the adopted adders are similar to the adders used for the proposed scheme. The results state that an estimation modulo;

$$\left(2^n\right): A = 5n + \left(\frac{3}{2}\right) n \log_2 n,$$

$$D = 2\log_2 n + 3$$

$$\left(2^n - 1\right): A = 12n + 3n\left(\log_2 n - 1\right),$$

$$D = 2\log_2 n + 3$$

Therefore, the hardware requirements of the scheme are as follows:

$$A_{PCP} = A_{ADD1} + A_{MUX} + A_{ADD2} + A_{ADD3} = 23n + \left(\frac{15}{2}\right)\log_2 n + 3$$

$$A_{MEP} = 2\left(A_{AND}\right) = 2$$

$$A_{ODP} = 2\left(A_{AND}\right) = 2$$

$$A_{OCP} = A_{ADD4} + A_{ADD5} + A_{ADD6} = 63n + 14$$

The estimated delay of the scheme will be as follows:

$$D_{PCP} = D_{ADD1} + D_{ADD2} + D_{ADD3} = 4\log_2 n + 5$$

$$A_{MEP} = 2\left(A_{AND}\right) = 2$$

$$A_{ODP} = 2\left(A_{AND}\right) = 2$$

$$D_{OCP} = D_{ADD4} + A_{ADD7} + A_{ADD8} = 9\,gates$$

Now, in order to make an effective comparison, the proposed scheme is divided into two: Proposed Scheme I for when the OCP is not included in the comparison and Proposed Scheme II for the OCP being included in the comparison. The delay of the OCP overrides the delay of the delays of the MEP and the ODP if Proposed Scheme II is consider since they will all be computed in parallel and the critical path in that case will be dictated by the OCP. The area for the PCP and the MEP is double for two numbers $X$ and $Y$ but this is not the case for the delay of the two numbers since they are computed in parallel. Thus, the total area and delay of the proposed schemes are:

$$A_{TOTAL(I)} = 2A_{PCP} + 2A_{MEP} + A_{ODP} = 46n + 15\log_2 n + 10$$

$$D_{TOTAL(I)} = D_{PCP} + D_{MEP} = 4\log_2 n + 7$$

and,

$$A_{TOTAL(II)} = 2A_{PCP} + 2A_{MEP} + A_{ODP} + A_{OCP} = 109n + 15\log_2 n + 24$$

$$D_{TOTAL(II)} = D_{PCP} + D_{OCP} = 4\log_2 n + 16$$

## 5. Numerical Illustrations

This subsection presents numerical illustrations of the proposed scheme.

*Checking overflow in the sum of 49 and 21 using RNS moduli set {3, 4, 5}*

$$X = 49 = (4,1,1)_{RNS\langle 5|4|3\rangle} = (100,01,01)_{RNS\langle 101|100|11\rangle}$$

$$Y = 21 = (1,1,0)_{RNS\langle 5|4|3\rangle} = (001,01,00)_{RNS\langle 101|100|11\rangle}$$

$$Z = \left((100,01,01) + (001,01,00)\right)_{RNS\langle 101|100|11\rangle} = (000,10,01)_{RNS\langle 101|100|11\rangle}$$

A reverse conversion of $(000,10,01)_{RNS\langle 101|100|11\rangle}$ will result in the decimal number 10. Whilst the sum of the decimal numbers 49 and 21 is 70 which is obvious of overflow occurring.

*Checking for RNS overflow using the proposed technique*

$$e_2(49) = |01 - 100|_{100} = 001$$

$$e_3(49) = |(01-01)(10) - 01|_{11} = |-01|_{11} = 10 = 2^2 - 2$$

which implies, $\beta(49) = 1$ from (11).

Also,

$$e_2(21) = |01 + |-001|_{100}|_{100} = |01 + 11|_{100} = 000$$

$$e_3(21) = |(11-01)(10) - 11|_{11} = |100 - 11|_{11} = 01 = 2^2 - 3$$

But $e_2(21) = 000$, which implies $\beta(21) = 0$ from (11).

Therefore, from (13) $overflow = \lambda$ and needs further processing.

Since $e_3(49) + e_3(21) = 10 + 01 = 11 > 2^2 - 2$, the scheme detects overflow occurring after processing.

***Correction unit***

$$Z = (100 + 001) + (01 + 00)(101) + (10 + 01)(101)(100)$$
$$= 1000110 = (70)_{decimal}$$

*Checking overflow in the sum of* 10 *and* 11 *using RNS moduli set* {3, 4, 5}

$$X = 49 = (4,1,1)_{RNS\langle 5|4|3\rangle} = (100,01,01)_{RNS\langle 101|100|11\rangle}$$

$$Y = 11 = (1,3,2)_{RNS\langle 5|4|3\rangle} = (001,11,10)_{RNS\langle 101|100|11\rangle}$$

$$Z = ((000,10,01) + (001,11,10))_{RNS\langle 101|100|11\rangle} = (001,01,00)_{RNS\langle 101|100|11\rangle}$$

RNS to decimal conversion of $(001,11,10)_{RNS\langle 101|100|11\rangle}$ will result in the decimal number 21, which is correct result of 10 + 11.

*Checking for RNS overflow using the proposed algorithm*

$$e_2(10) = |10 - 000|_{100} = 10 = 2^2 - 2$$

$$e_3(10) = |(01 - 000)(10) - 10|_{11} = |10 - 10|_{11} = 00 < 2^2 - 3$$

which implies, $\beta(10) = 0$ since $e_3(10) = 00 < 2^2 - 3$, from (11).

Also,

$$e_2(11) = |11 + |-001|_{100}|_{100} = |11 + 11|_{100} = 10 = 2^2 - 2$$

$$e_3(11) = |(10 - 001)(10) - 10|_{11} = |10 - 10|_{11} = 00 < 2^2 - 3$$

this implies, $\beta(11) = 0$ since $e_3(10) = 00 < 2^2 - 3$, from (11).

Thus, from (13) $overflow = 0$, which implies no overflow has occurred according to the proposed scheme after processing.

***Correction unit***

$$Z = (000 + 001) + (10 + 10)(101) + (00 + 00)(101)(100)$$
$$= 010101 = (21)_{decimal}$$

## 6. Performance Evaluation

The performance of the proposed scheme is compared to schemes in [3] and [8]; the scheme in [8] does not contain a correction unit; the scheme by [3] has a correction unit but is not included in the comparison. And so both schemes do not have the correction component in the comparison. Table 1 shows the analysis of the proposed scheme with that of similar state-of-the art schemes.

As shown in Table 1, the proposed scheme for detecting overflow (Proposed Scheme I) in the given moduli set is very cheap in terms of hardware resources

Table 1. Area, delay comparison.

| *Scheme* | *Area* | *Delay* |
|---|---|---|
| [8] | $76n + (33/2)n\log_2 n$ | $6\log_2 n + 23$ |
| [3] | $37n + 18$ | $16n + \log_2 n + 13$ |
| Proposed I | $46n + 15\log_2 n + 10$ | $4\log_2 n + 7$ |
| Proposed II | $109n + 15\log_2 n + 24$ | $4\log_2 n + 16$ |

and faster than the scheme by [8] but requires a little hardware resources than the scheme by [3] albeit slower than the Proposed Scheme I. However, the complete proposed scheme (Proposed Scheme II) for detecting and correcting overflow requires more hardware resources than the other compared schemes but faster than both schemes by [3] and [8].

Clearly, Proposed Scheme I completely outperforms the similar state-of-the-art scheme by [8] for detecting overflow, but the trust of this work is to detect and correct overflow anytime it occurs; in so doing it has made tremendous gains in speed as shown in **Table 1**.

**Table 2** shows a detailed analysis of the complexities and delay of the proposed scheme with that of the similar state-of-the-art schemes.

**Table 2** reveals interesting results theoretically, from the analysis it is clear that the Proposed Scheme I requires less resources than what is required by [8]. From the table, smaller values of $n$ shows that Proposed Scheme II requires more resources than that by [8] but drastically improves upon this requirements up to over 51% better than [8] for higher values of $n$ (*i.e.* $n > 4$), this is clearly shown in the graph in **Figure 5**. The analysis from the table also shows that whilst for smaller values of $n$ (say $n = 1$), the Proposed Scheme I is better than the scheme by [3] in terms of hardware resources, it tends to require up to about 18% resources more than that by [3].

From **Figure 5**, the scheme by [8] sharply increases for higher values of $n$ followed by the Proposed Scheme II whilst the scheme by [3] requires the lesser resources. Regarding the delay, the proposed schemes (Proposed I and Proposed II) completely outperforms both schemes by up to over 35% than the scheme by [8] and over 90% faster than the scheme by [3] as shown in **Table 2** and in **Figure 5**. It is worth noting that whiles the scheme by [3] performs better in terms of hardware resources, it tends to be the worst performer for speed and the

**Table 2.** Area, delay analysis for various values of $n$.

| $n$ | AREA | | | | DELAY | | | |
|---|---|---|---|---|---|---|---|---|
| | [8] | [3] | *Proposed* I | *Proposed* II | [8] | [3] | *Proposed* I | *Proposed* II |
| 1 | 76 | 57 | 56 | 133 | 23 | 29 | 7 | 16 |
| 2 | 185 | 96 | 117 | 257 | 29 | 46 | 11 | 20 |
| 4 | 436 | 174 | 224 | 490 | 35 | 79 | 15 | 24 |
| 8 | 1004 | 330 | 423 | 941 | 41 | 144 | 19 | 28 |
| 16 | 2272 | 642 | 806 | 1828 | 47 | 273 | 23 | 32 |
| 32 | 5072 | 1266 | 1557 | 3587 | 53 | 530 | 27 | 36 |
| 64 | 11,200 | 2514 | 3044 | 7090 | 59 | 1043 | 31 | 40 |
| 128 | 24,512 | 5010 | 6003 | 14,081 | 65 | 2068 | 35 | 44 |
| 256 | 53,248 | 10,002 | 11,906 | 28,048 | 71 | 4117 | 39 | 48 |
| 512 | 114,944 | 19,986 | 23,697 | 55,967 | 77 | 8214 | 43 | 52 |
| **Total** | **212,949** | **40,077** | **47,833** | **112,422** | **500** | **16,543** | **250** | **340** |

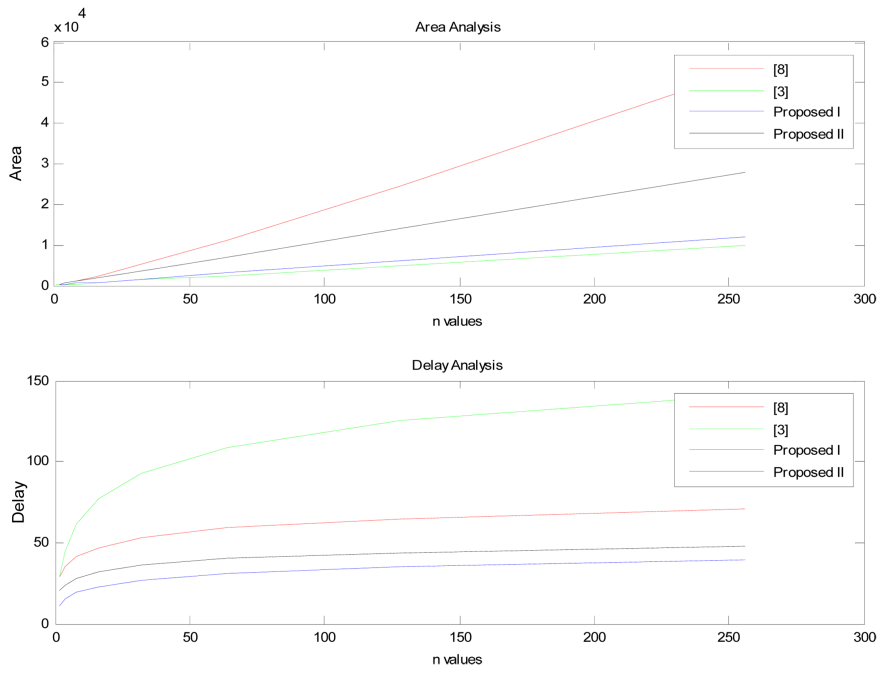

**Figure 5.** Graphs of area and delay analysis of the various compared schemes.

percentage difference shows that Proposed I is more efficient. It is clear from the graphs that in terms of delay, the scheme in [3] sharply increases with increasing values of $n$ whiles the marginal increase of the rest of the schemes are minimal.

## 7. Conclusion

Detecting overflow in RNS arithmetic computations is very important but can be difficult, more so, if it has to be corrected. In this paper, an ingenious technique of detecting overflow by use of the MRC method through magnitude evaluation as well correcting the overflow when it occurs was presented. This technique did not require full reverse conversion but used the MRDs to evaluate the sign of a number to detect the occurrence of overflow. With this technique, the correct value of the sum of two numbers is guaranteed whether overflow occurred or not. The scheme has been demonstrated theoretically to be very fast than similar-state-of-the-art scheme but required a little more hardware resources. However, the Proposed Scheme I, which is the one without the correction component completely outperformed the scheme in [8] in terms of both area and delay requirements. Also, results from Table 2 and Figure 5 showed that for higher values of $n$, the Proposed Scheme II also outperformed the scheme by [8]. Future works will focus on simulating the theoretical results and implementing it on FPGA boards.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] Omondi, A. and Premkumar, B. (2007) Residue Number Systems: Theory and Implementation, vol. 2. Imperial College Press, London. https://doi.org/10.1142/p523

[2] Agbedemnab, P.A. and Bankas, E.K. (2015) A Novel RNS Overflow Detection and Correction Algorithm for the Moduli Set {2^n−1,2^n,2^n+1}. *International Journal of Computer Applications in Technology*, **110**, 30-34. https://doi.org/10.5120/19403-0925

[3] Younes, D. and Steffan, P. (2013) Universal Approaches for Overflow and Sign Detection in Residue Number System Based on {2n − 1, 2n, 2n + 1}. *The Eighth International Conference on Systems*, Seville, 27 January-1 February 2013, 77-81.

[4] Daabo, M.I. and Gbolagade, K.A. (2012) RNS Overflow Detection Scheme for the Moduli set {M − 1, M}. *Journal of Computing*, **4**, 39-44.

[5] Daabo, M.I. (2015) Overflow Detection Schemes for Residue Number System Architecture. PhD Thesis, University for Development Studies, Tamale, Ghana.

[6] Debnath, R.C. and Pucknell, D.A. (1978) On Multiplicative Overflow Detection in Residue Number System. *Electronics Letters*, **14**, 129-130. https://doi.org/10.1049/el:19780088

[7] Askarzadeh, M., Hosseinzadeh, M. and Navi, K. (2009) A New Approach to Overflow Detection in Moduli Set 2n−3, 2n−1, 2n+1, 2n+3. 2009 *Second International Conference on Computer and Electrical Engineering*, **1**, 439-442. https://doi.org/10.1109/ICCEE.2009.197

[8] Rouhifar, M., Hosseinzadeh, M., Bahanfar, S. and Teshnehlab, M. (2011) Fast Overflow Detection in Moduli set {2^n-1,2^n,2^n+1}. *International Journal of Computer Science Issues*, **8**, 407-414.

[9] Piestrak, S.J. (1995) A High-Speed Realization of a Residue to Binary Number System Converter. *IEEE Transactions on Circuits and Systems. Part II: Analog and Digital Signal Processing*, **42**, 661-663. https://doi.org/10.1109/82.471401

[10] Siewobr, H. and Gbolagade, K.A. (2014) RNS Overflow Detection by Operands Examination. *International Journal of Computer Applications in Technology*, **85**, 1-5. https://doi.org/10.5120/14938-2906

[11] Molahosseini, A.S., Navi, K., Dadkhah, C., Kavehei, O. and Timarchi, S. (2010) Efficient Reverse Converter Designs for the New 4-Moduli Sets {2^n−1, 2^n, 2^n+1, 2^(2n+1)−1} and {2^n−1, 2^n+1, 2^n, 2^(2n)+1} Based on New CRTs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, **57**, 823-835. https://doi.org/10.1109/TCSI.2009.2026681

[12] Zimmermann, R. (1999) Efficient VLSI Implementation of Modulo (2^n±1) Addition and Multiplication. Proceedings 14*th IEEE Symposium on Computer Arithmetic*, Adelaide, 14-16 April 1999, 158-167.

[13] Chang, C.H., Low, J. and Yung, S. (2011) Simple, Fast, and Exact RNS Scaler for the Three-Moduli Set {2 ^n−1, 2^n, 2^n+1}. *IEEE Transactions on Circuits and Systems I: Regular Papers*, **58**, 2686-2697. https://doi.org/10.1109/TCSI.2011.2142950

[14] Sousa, L. (2015) 2^n RNS Scalers for Extended 4-Moduli Sets. *IEEE Transactions on Computers*, **99**, 1-14.