www.ijecs.in

International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 7 Issue 2 February 2018, Page No. 23578-23584
Index Copernicus Value (2015): 58.10, 76.25 (2016) DOI: 10.18535/ijecs/v7i2.09

A Fault Tolerant Scheme for Detecting Overflow in Residue Number Microprocessors

Agbedemnab P. A.[1], Agebure M. A.[2] & Akobre S.[3]
[1,2,3]Department of Computer Science, University for Development Studies
P. O. Box 24, Navrongo Campus, Ghana

Abstract: The decomposition of larger numbers into smaller ones termed as residues is the main operation behind the concept of Residue Number System (RNS); it possesses inherent features such as parallelism and independent digit arithmetic computations. These features of the RNS has made it desirable for applications that require intensive computations such as Digital Signal Processing (DSP), Digital Filtering and Convolutions. Overflow detection is one of the major challenges that confront the efficient implementation of RNS in general purpose computer processors. Overflow occurs in RNS when an illegitimate value is represented within legitimate range – Dynamic Range (DR) as if it is legitimate value. This misrepresentation of results, which usually arises during addition operations ultimately affects systems built on this Number System. It is therefore imperative that steps are taken not to only detect but correct the occurrence of overflow whenever it occurs. In this paper, an additive overflow detection and correction scheme for the moduli set $\{\mathbf{2^{2n+1}-1, 2^n+1, 2^n-1}\}$ is presented. The scheme uses a redundant modulus to extend the DR of the moduli set. The proposed scheme is demonstrated theoretically to be an efficient scheme by comparing it to previous similar works.

Keywords: Residue Number System (RNS), Digital Signal Processing (DSP), Overflow Detection, Number System, Dynamic Range.

1. Introduction

A riddle by Sun Tzu, a Chinese scholar in a book authored in the first century was the first documented manifestation of Residue Number System (RNS) representation [1] and [2]. Summarily, the riddle sought to find out a number that will yield the remainders 2, 3, and 2 when divided by 3, 5, and 7, respectively; thus in modern terminology, 2, 3, and 2 are the residues, and 3, 5, and 7, are the moduli that make the RNS. This number system was later rediscovered by computer scientists who found it necessary in the implementation of fast arithmetic and fault-tolerant computing [1]. Three properties of RNS make it well suited for these: the first is absence of carry-propagation in addition and multiplication, carry-propagation being the most significant speed-limiting factor in these operations. The second is that because the residue representations carry no weight-information, an error in any digit-position in a given representation does not affect other digit-positions. The third is that there is no significance-ordering of digits in an RNS representation, which means that faulty digit-positions may be discarded with no effect other than a reduction in dynamic range. This renewed interest is even more prominent now because a great deal of computing now takes place in embedded processors, such as those found in mobile devices, and for these, high speed and low-power consumption are critical; the absence of carry-propagation facilitates the realization of high-speed, low-power arithmetic. Also, computer chips are now getting to be so dense that full testing will no longer be possible and therefore, makes fault-tolerance and the general area of computational integrity essential. Lastly, there has been progress in the implementation of some difficult arithmetic operations such as division, number/data conversion, scaling, overflow and magnitude detections [3],[4].

An RNS number X, is represented as $X \equiv x_i \, mod \, m_i$, where $m_i = \{m_1, m_2 \dots m_n\}$, a set of pairwise relatively prime integers such that $\gcd(m_i, m_j)$; $m_i \neq m_j$, $i, j = 1, 2 \dots n$. The residue set $x_i = [x_1, x_2, \dots, x_n]$ 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 $+, -, \times, \div$), [5], results in a value that is outside the legitimate range, then overflow is said to have occurred.

Overflow is an error in computing as a result of misrepresentation of illegitimate values in a given memory space [6], [4]; this relates to the DR in RNS which situation usually arises during addition and multiplication operations. Thus, detecting overflow is one of the fundamental issues in the design of efficient RNS microprocessors [7].

The conversion of an RNS number into its decimal/binary equivalent number has long been mainly based on the Chinese Remainder Theorem (CRT) and the Mixed Radix Conversion (MRC) techniques with few modifications being their variants in 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.

The MRC, as used in this paper is famously computed as follows:

$$X = e_1 + e_2 m_1 + e_3 m_1 m_2 + \cdots + e_n m_1 m_2 \dots m_{n-1} \qquad (1)$$

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

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

The MRDs e_i are within the range $0 \leq e_i \leq m_i$, and a positive number, X, in the interval $[0, M]$ can be uniquely represented.

Recently, some techniques have been developed to detect overflow without necessarily completing the reverse conversion process; [8] proposed an algorithm to detect overflow in the moduli set $(2^n-3, 2^n-1, 2^n, 2^n+1, 2^n+3)$ using a parity checking technique but used ROMs for implementation. In [9], 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 [5] 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 [10]. The scheme in [11] 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 [5], or other costly and time consuming procedures such as base extension, group number and sign detection as in [9] and [11]. In this paper, an additive overflow detection and correction scheme for the moduli set $\{2^{2n+1}-1, 2^n+1, 2^n-1\}$, which has odd dynamic range is presented. The scheme uses a redundant modulus – 2 by extending the dynamic range of the moduli set. This redundant modulus is then used to detect overflow during addition whenever it occurs by XORing the sum of the residues corresponding to the redundant modulus and the LSB of the result of summing the residues corresponding to two numbers in the original moduli set.

2. Proposed Method

Given the moduli set $\{2^{2n+1}-1, 2^n+1, 2^n-1\}$, [12], [13], let $m_1 = 2^{2n+1}-1$, $m_2 = 2^n+1$ and $m_3 = 2^n-1$. Let $m_4 = 2$ be a redundant modulus by extending the original moduli set. In order to detect overflow in the given moduli set, a redundant modulus 2 is added so that the new set becomes $\{2^{2n+1}-1, 2^n+1, 2^n-1, 2\}$; but the dynamic range $M = (2^{2n+1}-1)(2^n+1)(2^n-1)$.

Theorem 1: Given the moduli set $\{2^{2n+1}-1, 2^n+1, 2^n-1\}$, where $m_1 = 2^{2n+1}-1$, $m_2 = 2^n+1$ and $m_3 = 2^n-1$, we have;

$$|m_1^{-1}|_{m_2} = 1 \quad (3)$$

$$|m_1^{-1}|_{m_3} = 1 \quad (4)$$

$$|m_2^{-1}|_{m_3} = 2^{n-1} \quad (5)$$

Proof: If it can be shown that $|a \times b|_{m_i} = 1$, then $|b|_{m_i} = |a^{-1}|_{m_i}$ is the multiplicative inverse of a. Thus, for (3)

$$\begin{aligned}|(2^{2n+1}-1)\times 1|_{2^n+1} &= |(2(2^n)(2^n)-1)|_{2^n+1}\\ &= |(2(-1)(-1)-1)|_{2^n+1}\\ &= |1|_{2^n+1} = 1\end{aligned}$$

Hence 1 is the multiplicative inverse of m_1 with respect to m_2.
Similarly, for (4)

$$\begin{aligned}|(2^{2n+1}-1)\times 1|_{2^n-1} &= |(2(2^n)(2^n)-1)|_{2^n-1}\\ &= |1|_{2^n-1} = 1\end{aligned}$$

Hence 1 is the multiplicative inverse of m_1 with respect to m_3.
Also, for (5)

$$\begin{aligned}|(2^n+1)\times 2^{n-1}|_{2^n-1} &= |(2)\times 2^{n-1}|_{2^n-1}\\ &= |2^n|_{2^n-1}\\ &= |1|_{2^n-1} = 1\end{aligned}$$

Hence 2^{n-1} is the multiplicative inverse of m_2 with respect to m_3.

Given the RNS numbers $X = (x_1, x_2, x_3, x_4)$ and $Y = (y_1, y_2, y_3, y_4)$.
Let the sum $Z = (z_1, z_2, z_3, z_4) = (x_i + y_i)$, $i = 1, 2, 3, 4$. Then two scenerios arise;

(i) If both addends have the same parity then $Z = (z_1, z_2, z_3)$ *is even and* $z_4 = 0$

(ii) *If the addends have different parity then* $Z = (z_1, z_2, z_3)$ *is odd and* $z_4 = 1$

Therefore, overflow occurs whenever $Z = (z_1, z_2, z_3)$ is odd and $z_4 = 0$ or $Z = (z_1, z_2, z_3)$ is even and $z_4 = 1$. Thus, the proposed method detects overflow as follows;

$$Overflow = \begin{cases} 1; & z_4\ XOR\ LSB(Z) = 1 \\ \\ 0, & Otherwise \end{cases} \quad (6)$$

Where $LSB(Z)$ is the least significant bit of the sum Z.

Next, a partial residue-binary conversion of the addends is done by computing their respective MRDs. The MRDs of one addend (say X) is done by substituting (3) – (5) into equation (2) to simplify as follows;

$$\begin{aligned} e_1 &= x_1 \\ e_2 &= |(x_2 - e_1)1|_{2^n+1} \\ &= |x_2 - x_1|_{2^n+1} \\ e_3 &= |((x_3 - e_1) - e_2)2^{n-1}|_{2^n-1} \\ &= |2^{n-1}(x_3 - x_1) - 2^{n-1}e_2|_{2^n-1} \end{aligned} \quad (7)$$

Therefore, Z is obtained by adding the individual MRDs of the two addends. In case of an occurrence of overflow, the dynamic range should be shifted one bit to the left thus including the modulus 2 in order to legitimize the value of Z. The value of Z computed this way is the correct result whether overflow occurs or not.

3. Hardware Implementation

From equation (7), the MRDs can be represented in binary as;

$$e_1 = \underbrace{e_{1,2n}e_{1,2n-1}\dots e_{1,1}e_{1,0}}_{2n+1} \quad (8)$$

$$e_2 = \underbrace{e_{2,n}e_{2,n-1}\dots e_{2,1}e_{2,0}}_{n+1} \quad (9)$$

$$e_3 = \underbrace{e_{3,n-1}e_{3,n-2}\dots e_{3,1}e_{3,0}}_{n} \quad (10)$$

Equations (8) – (10) can further be simplified as follows;

$$e_1 = x_1 = \underbrace{x_{1,2n}x_{1,2n-1}\dots x_{1,1}x_{1,0}}_{2n+1} \quad (11)$$

$$\begin{aligned} e_2 &= |x_2 + A|_{2^n+1} \\ &= \left|\underbrace{x_{2,n}x_{2,n-1}\dots x_{2,1}x_{2,0}}_{n+1} + \underbrace{A_nA_{n-1}\dots A_1A_0}_{n+1}\right|_{2^n+1} \\ &= \underbrace{e_{2,n}e_{2,n-1}\dots e_{2,1}e_{2,0}}_{n+1} \end{aligned} \quad (12)$$

where,

$$A = \left|-\underbrace{(x_{1,2n}x_{1,2n-1}\dots x_{1,1}x_{1,0})}_{2n+1}\right|_{2^n+1}$$

$$= \left|\underbrace{11\dots 1\bar{x}_{1,2n}}_{n} + \underbrace{\bar{x}_{1,2n-1}\dots\bar{x}_{1,n}}_{n} + \underbrace{\bar{x}_{1,n-1}\dots\bar{x}_{1,0}}_{n}\right|_{2^n+1} \quad (13)$$

Let $A_1 = \underbrace{11\dots 1\bar{x}_{1,2n}}_{n}$, $A_2 = \underbrace{\bar{x}_{1,2n-1}\bar{x}_{1,2n-2}\dots\bar{x}_{1,n}}_{n}$ and
$A_3 = \underbrace{\bar{x}_{1,2n-1}\bar{x}_{1,2n-2}\dots\bar{x}_{1,n}}_{n}$ (14)

Also,

$$\begin{aligned} e_3 &= |2^{n-1}(x_3 - x_1) - 2^{n-1}e_2|_{2^n-1} \\ &= \left|\underbrace{x_{3,0}\dots x_{3,1}}_{n} + \underbrace{B_{n-1}\dots B_1B_0}_{n} + \underbrace{C_{n-1}\dots C_1C_0}_{n}\right|_{2^n-1} \\ &= \underbrace{e_{3,n-1}e_{3,n-2}\dots e_{3,1}e_{3,0}}_{n} \end{aligned} \quad (15)$$

where,

$$B = |-2^{n-1}x_1|_{2^n-1}$$
$$= \left|-2^{n-1}\underbrace{\left(x_{1,2n}x_{1,2n-1}\dots x_{1,1}x_{1,0}\right)}_{2n+1}\right|_{2^n-1}$$
$$= \left|-2^{n-1}\underbrace{\left(00\dots 0x_{1,2n}\right)}_{n} - 2^{n-1}\underbrace{\left(x_{1,2n-1}\dots x_{1,n}\right)}_{n} - 2^{n-1}\underbrace{\left(x_{1,n-1}\dots x_{1,0}\right)}_{n}\right|_{2^n-1}$$
$$= \left|\underbrace{\bar{x}_{1,2n}11\dots 1}_{n} + \underbrace{\bar{x}_{1,n}\bar{x}_{1,2n-1}\dots \bar{x}_{1,n+1}}_{n} + \underbrace{\bar{x}_{1,0}\bar{x}_{1,n-1}\dots \bar{x}_{1,1}}_{n}\right|_{2^n-1} \quad (16)$$

Let

$B_1 = \underbrace{x_{3,0}x_{3,n-1}\dots x_{3,1}}_{n}$, $B_2 = \underbrace{\bar{x}_{1,2n}11\dots 1}_{n}$, $B_3 = \underbrace{\bar{x}_{1,n}\dots \bar{x}_{1,n+1}}_{n}$

and $B_3 = \underbrace{\bar{x}_{1,0}\bar{x}_{1,n-1}\dots \bar{x}_{1,1}}_{n}$ (17)

Finally,

$$C = |-2^{n-1}e_2|_{2^n-1}$$
$$= \left|-2^{n-1}\underbrace{\left(e_{2,n}e_{2,n-1}\dots e_{2,1}e_{1,0}\right)}_{n+1}\right|_{2^n-1}$$
$$= \left|-2^{n-1}\left(e_{2,n}\times 2^n + \underbrace{e_{2,n-1}\dots e_{2,1}e_{1,0}}_{n}\right)\right|_{2^n-1} \quad (18)$$

Since, e_2 is a number that is smaller than $2^n + 1$, two cases can be considered. First, when e_2 is smaller than 2^n, and second, when e_2 is equal to 2^n [13].

If $e_{2,n} = 0$, we have

$$C_1 = \left|-2^{n-1}\underbrace{\left(e_{2,n-1}e_{2,n-2}\dots e_{2,1}e_{1,0}\right)}_{n}\right|_{2^n-1}$$
$$= \underbrace{\bar{e}_{2,0}\bar{e}_{2,n-1}\dots \bar{e}_{2,2}\bar{e}_{2,1}}_{n} \quad (19)$$

Else if $e_{2,n} = 1$, the following binary vector can be obtained as

$$C_2 = \left|-2^{n-1}\times 2^n(\underbrace{00\dots 0}_{n-1}\,e_{2,n})\right|_{2^n-1} = 0\,\underbrace{11\dots 1}_{n-1} \quad (20)$$

Therefore, t_3 is calculated as

$$C = \begin{cases} C_1, & \text{if } e_{2,n} = 0 \\ C_2, & \text{if } e_{2,n} = 1 \end{cases} \quad (21)$$

Let γ and ω represent the MRDs of the two integers X and Y respectively. Then from equations (11), (12) and (15), we have

$$\psi_i = \gamma_i + \omega_i \quad (22)$$

which implies

$$\psi_1 = \gamma_1 + \omega_1$$
$$= \underbrace{\gamma_{1,n}\gamma_{1,n-1}\dots\gamma_{1,1}\gamma_{1,0}}_{2n+1} + \overbrace{\omega_{1,n}\omega_{1,n-1}\dots\omega_{2,1}\omega_{1,0}}^{2n+1}$$
$$= \psi_{1,2n}\psi_{1,n-1}\dots\psi_{1,1}\psi_{1,0} \quad (23)$$

$$\psi_2 = \gamma_2 + \omega_2$$
$$= \underbrace{\gamma_{2,n}\gamma_{2,n-1}\dots\gamma_{2,1}\gamma_{2,0}}_{n+1} + \overbrace{\omega_{2,n}\omega_{2,n-1}\dots\omega_{2,1}\omega_{2,0}}^{n+1}$$
$$= \psi_{2,n}\psi_{2,n-1}\dots\psi_{2,1}\psi_{2,0} \quad (24)$$

finally,

$$\psi_3 = \gamma_3 + \omega_3$$
$$= \underbrace{\gamma_{3,n-1}\gamma_{3,n-2}\dots\gamma_{3,1}\gamma_{3,0}}_{n} + \overbrace{\omega_{3,n-1}\omega_{3,n-2}\dots\omega_{3,1}\omega_{3,0}}^{n}$$
$$= \psi_{3,n-1}\psi_{3,n-2}\dots\psi_{3,1}\psi_{3,0} \quad (25)$$

and so, Z is implemented as;

$$Z = t_2 + t_3 + t_4 + t_5 + t_6$$
$$= \underbrace{\underbrace{t_{2,4n}\dots t_{1,0}}_{4n+1} + \overbrace{0\dots 0}^{n-1}\underbrace{t_{3,2n+1}\dots t_{3,0}}_{3n+2} + \overbrace{0\dots 0}^{3n+1}\underbrace{t_{4,n-1}\dots z_{4,1}z_{4,0}}_{n} + \overbrace{0\dots 0}^{3n}\underbrace{t_{5,n}\dots t_{5,0}}_{n+1} + \overbrace{0\dots 0}^{2n+1}\underbrace{t_{6,n}\dots t_{6,0}}_{2n}}_{4n+1} \quad (26)$$

where,

$$t_2 = 2^{3n+1}\psi_3 + t_1$$
$$= \underbrace{\psi_{3,n-1}\psi_{3,n-2}\dots\psi_{3,1}\psi_{3,0}}_{n}\overbrace{00\dots 0}^{3n+1} \bowtie \underbrace{t_{1,3n}\dots t_{1,1}t_{1,0}}_{3n+1}$$
$$= t_{2,4n}\dots t_{2,1}t_{2,0} \quad (27)$$

and,

$$t_1 = 2^{2n+1}\psi_3 + \psi_1$$
$$= \underbrace{\psi_{3,n-1}\dots\psi_{3,1}\psi_{3,0}}_{n}\overbrace{00\dots 0}^{2n+1} \bowtie \underbrace{\psi_{1,2n}\dots\psi_{1,1}\psi_{1,0}}_{2n+1}$$
$$= t_{1,3n}t_{1,3n-1}\dots t_{1,1}t_{1,0} \quad (28)$$

$$t_3 = 2^{2n+1}\psi_2$$
$$= \underbrace{\psi_{2,n}\psi_{2,n-1}\dots\psi_{2,1}\psi_{2,0}}_{n+1}\overbrace{00\dots 0}^{2n+1}$$
$$= t_{3,3n+1}t_{3,3n}\dots t_{3,1}t_{3,0} \quad (29)$$

$$t_4 = -\psi_3$$
$$= \underbrace{\bar{\psi}_{3,n-1}\bar{\psi}_{3,n-2}\dots\bar{\psi}_{3,1}\bar{\psi}_{3,0}}_{n}$$
$$= t_{4,n-1}t_{4,n-2}\dots t_{4,1}t_{4,0} \quad (30)$$

$$t_5 = -\psi_3$$
$$= \underbrace{\bar{\psi}_{2,n}\bar{\psi}_{2,n-1}\dots\bar{\psi}_{2,1}\bar{\psi}_{2,0}}_{n}$$
$$= t_{5,n}t_{5,n-1}\dots t_{5,1}t_{5,0} \quad (31)$$

finally,

$$t_6 = -2^n\psi_3$$
$$= \underbrace{\bar{\psi}_{3,n-1}\bar{\psi}_{3,n-2}\dots\bar{\psi}_{3,1}\bar{\psi}_{3,0}}_{n}\overbrace{11\dots 1}^{n}$$
$$= t_{6,2n-1}t_{6,2n-1}\dots t_{6,1}t_{6,0} \quad (32)$$

3.1 Hardware Realisation

The hardware architecture of the proposed scheme is first realised by computing the MRDs of the two addends X and Y according to (12) and (15) which parameters are defined in (13), (14), (16), (17) and (21). These MRDs are e_2 and e_3, and e_1 in (4) which is equivalent to x_1. Figure 1 shows the unit for computing the MRDs of one addend X and repeated for the other addend Y. Figure 1 consists of a two level Carry Save Adder (CSA) tree for computing e_2 and another three level CSA tree for computing e_3 whose sum and carry are added using two separate CPAs each. This unit is called here Partial Reverse Converter (PRC) as a component of the proposed scheme. The PRC starts with an Operands Preparation Unit (OPU 1), which prepares the operands in (14) and (17) by simply manipulating the routing of the bits of the residues. The operands in (14) are added with CSA 1 at a first level and at a second level includes x_2 in CSA 3 which sum and carry are added using CPA 1 to get e_2. A multiplexer is used to determine (21) by either choosing (19) or (20) depending on the MSB of e_2. The value from (21) and the operands in (17) are then added using the three level CSA tree in CSA 2, CSA 4 and CSA 5 and finally propagated with CPA 2 in order to get e_3. These MRDs are useful in computing the sum of the addends Z by the Reverse Converter (RC) in Figure 2. The respective MRDs of the addends are summed according to (23) – (25) and computed according to CPA 3, CPA 4 and CPA 5. These and other four adders make up the architecture in the reverse converter for the sum Z in Figure 2. After an operand preparation (26) is computed by a three level carry save tree in CSA 2, CSA 3 and CSA 4 in a cascading manner whose sum and carry are then added using CPA 6 in order to get Z which is the correct result of the addition operation whether overflow occurs or not. Finally, overflow is detected by XORing the LSB(Z) with $|Z|_2 = z_4$ according to (6) and shown in Figure 3.

The hardware complexities and delay (time required for processing) of the proposed scheme are estimated as follows;

The area (A) and delay (D) of the PRC are:

$$A_{PRC} = A_{CSA1} + A_{CSA2} + A_{CSA3} + A_{CSA4} + A_{CSA5} + A_{CPA1} + A_{CPA2}$$

$$= (n+1)\Delta_{FA} + n\Delta_{FA} + (n+1)\Delta_{FA} + n\Delta_{FA} + n\Delta_{FA} + n\Delta_{FA} + n\Delta_{FA}$$

$$= (7n+2)\Delta_{FA}$$

$$D_{PRC} = D_{CSA1} + D_{CSA3} + D_{CPA1} + D_{CSA5} + D_{CPA1} + D_{CPA2}$$

$$= D_{FA} + D_{FA} + 2nD_{FA} + D_{FA} + 2nD_{FA}$$

$$= (4n+3)D_{FA}$$

The area requirement and delay imposed by the RC are:

$$A_{RC} = A_{CPA3} + A_{CPA4} + A_{CPA5} + A_{CSA2} + A_{CSA3} + A_{CSA4} + A_{CPA6}$$

$$= (2n+1)\Delta_{FA} + (n+1)\Delta_{FA} + (n-1)\Delta_{FA}$$

$$+3(4n+1)\Delta_{FA} + (4n+1)\Delta_{FA}$$

$$= (20n+5)\Delta_{FA}$$

$$D_{PRC} = D_{CPA3} + D_{CSA2} + D_{CSA3} + D_{CSA4} + D_{CPA6}$$

$$= (4n+2)D_{FA} + 3D_{FA} + (8n+2)D_{FA}$$

$$= (12n+7)D_{FA}$$

The ODU is a two input XOR gate and requires a unit of gate each for the area and delay. Equations (27) and (28) are realised by merely joining (concatenating) bits since the sum of a and $2^n b$ is computed as b concatenation a if a is an n-bit number [14], hence does not require any hardware or impose a delay. Also, the area for two addends will be double in the case of the PRC but the same delay.

Therefore, the total area requirements and delay of the proposed scheme are:

$$A_{TOTAL} = 2A_{PRC} + A_{RC} + A_{ODU}$$

$$= (14n+4)\Delta_{FA} + (20n+5)\Delta_{FA} + \Delta_{FA}$$

$$= (34n+10)\Delta_{FA}$$

$$D_{TOTAL} = D_{PRC} + D_{RC} + D_{ODU}$$

$$= (4n+3)D_{FA} + (12n+7)D_{FA} + D_{FA}$$

$$= (16n+11)D_{FA}$$

The schematic diagrams of the proposed scheme are shown in Figures 1, 2 and 3.

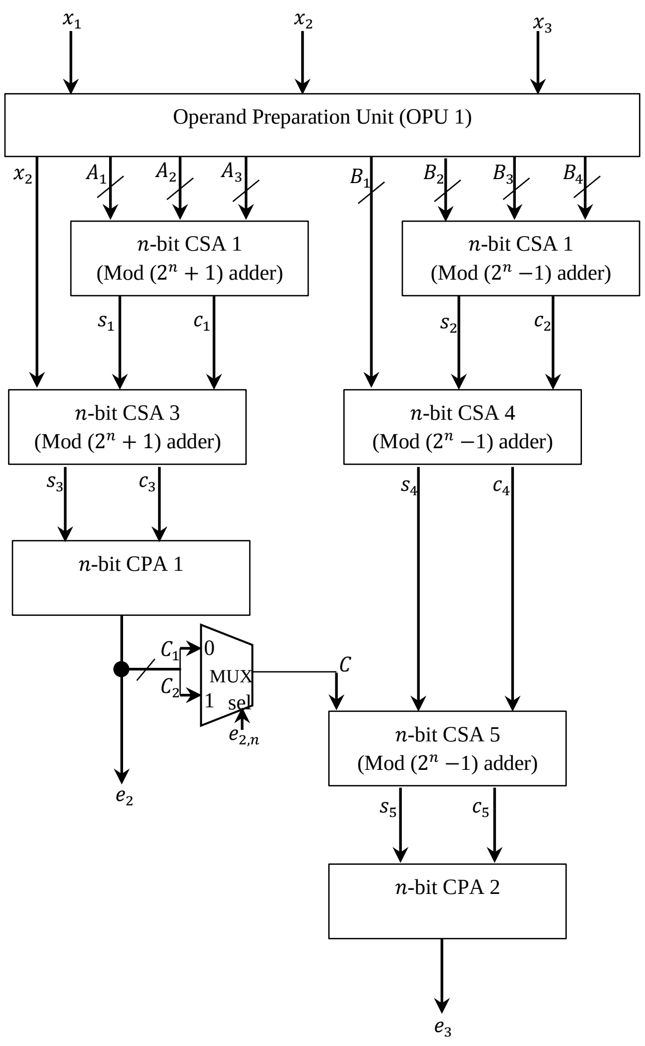

x_1
x_2
x_3
Operand Preparation Unit (OPU 1)
x_2
A_1
A_2
A_3
B_1
B_2
B_3
B_4
n-bit CSA 1 (Mod ($2^n + 1$) adder)
n-bit CSA 1 (Mod ($2^n - 1$) adder)
s_1
c_1
s_2
c_2
n-bit CSA 3 (Mod ($2^n + 1$) adder)
n-bit CSA 4 (Mod ($2^n - 1$) adder)
s_3
c_3
s_4
c_4
n-bit CPA 1
C_1
C_2
0
1
MUX
sel
$e_{2,n}$
C
e_2
n-bit CSA 5 (Mod ($2^n - 1$) adder)
s_5
c_5
n-bit CPA 2
e_3

Figure 1: Partial Reverse Converter (PRC)

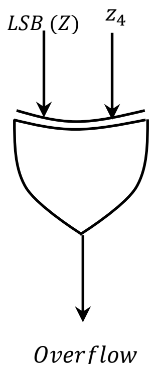


Figure 2: Overflow Detection Unit (ODU)

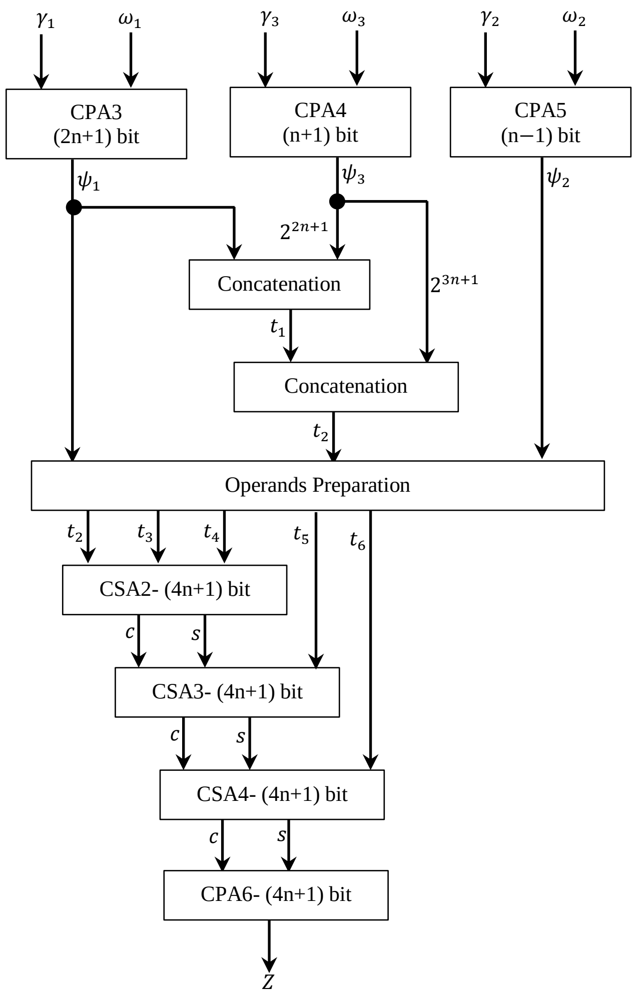


Figure 3: *Reverse Converter (RC)*

3.2 Numerical Illustrations

This section presents numerical illustrations of the proposed scheme.

Checking overflow in the sum of 225 and 275 using RNS moduli set $\{31,5,3,2\}$.

Legitimate range (DR) = 465. Let;

$X = 225 = (8,0,0,\mathbf{1})_{RNS(31|5|3|\mathbf{2})} = (01000,000,00,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

$Y = 275 = (27,0,2,\mathbf{1})_{RNS(31|5|3|\mathbf{2})} = (11011,000,10,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

$Z = \left((01000,000,00,\mathbf{1}) + (11011,000,10,\mathbf{1})\right)_{RNS(31|5|3|\mathbf{2})}$

$= (00100,000,10,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$

RNS to decimal conversion of $(00100,000,10)_{RNS(31|5|3)}$ results in decimal number 35. Meanwhile the sum of 225 and 275 is 500, a clear case of overflow occurrence.

Checking for RNS overflow using the proposed method

$Z = (00100,000,10,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$ implies $\mathbf{z_4} = \mathbf{0}$ and

$Z = (00100,000,10)_{RNS(31|5|3)} = 100011_{BINARY}$ which implies $LSB(Z) = 1$

Therefore, $LSB(Z) XOR\ \mathbf{z_4} = 1\ XOR\ \mathbf{0} = \mathbf{1}$. Thus overflow has occurred according to the proposed method since both numbers have the same parity.

Correction part

The correct value of Z is RNS to decimal conversion of $(00100,000,10,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$ which results in decimal number 500.

Checking overflow in the sum of 225 and 322 using RNS moduli set $\{31,5,3,2\}$.

Legitimate range (DR) = 465. Let;

$X = 225 = (8,0,0,\mathbf{1})_{RNS(31|5|3|\mathbf{2})} = (01000,000,00,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

$Y = 322 = (12,2,1,\mathbf{0})_{RNS(31|5|3|\mathbf{2})} = (01100,010,01,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$

$Z = \left((01000,000,00,\mathbf{1}) + (01100,010,01,\mathbf{0})\right)_{RNS(31|5|3|\mathbf{2})}$

$= (10100,010,01,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

RNS to decimal conversion of $(10100,010,01)_{RNS(31|5|3)}$ results in decimal number 82. Meanwhile the sum of 225 and 322 is 547, a clear case of overflow occurrence.

Checking for RNS overflow using the proposed method

$Z = (10100,010,01,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$ implies $\mathbf{z_4} = \mathbf{1}$ and

$Z = (10100,010,01)_{RNS(31|5|3)} = 1010010_{BINARY}$ which implies $LSB(Z) = 0$

Therefore, $LSB(Z) XOR\ \mathbf{z_4} = 0\ XOR\ \mathbf{1} = \mathbf{1}$. Thus overflow has occurred according to the proposed method since both numbers have different parity.

Correction part

The correct value of Z is RNS to decimal conversion of $(10100,010,01,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$ which results in decimal number 547.

Checking overflow in the sum of 225 and 35 using RNS moduli set $\{31,5,3,2\}$.

Legitimate range (DR) = 465. Let;

$X = 225 = (8,0,0,\mathbf{1})_{RNS(31|5|3|\mathbf{2})} = (01000,000,00,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

$Y = 35 = (4,1,1,\mathbf{1})_{RNS(31|5|3|\mathbf{2})} = (00100,001,01,\mathbf{1})_{RNS(31|5|3|\mathbf{2})}$

$Z = \left((01000,000,00,\mathbf{1}) + (00100,001,01,\mathbf{1})\right)_{RNS(31|5|3|\mathbf{2})}$

$= (001100,001,01,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$

RNS to decimal conversion of $(001100,001,01)_{RNS(31|5|3)}$ results in decimal number 260 which is the correct result of summing 225 and 35. In this case overflow has not occurred.

Checking for RNS overflow using the proposed method

$Z = (001100,001,01,\mathbf{0})_{RNS(31|5|3|\mathbf{2})}$ implies $\mathbf{z_4} = \mathbf{0}$ and

$Z = (001100,001,01)_{RNS(31|5|3)} = 100000100_{BINARY}$ which implies $LSB(Z) = 0$

Therefore, $LSB(Z) XOR\ \mathbf{z_4} = 0\ XOR\ \mathbf{0} = \mathbf{0}$. Thus overflow has not occurred according to the proposed method.

Since overflow has not occurred, there will not be any need for the correction unit.

4. Performance Evaluation

The performance of the proposed scheme is compared to similar schemes of equal dynamic range reverse converter as well as the scheme by [8] that have odd dynamic range. The complexities that are considered here for the analysis are a Full Adder (FA), a Half Adder (converted to FA) and a two input XOR gate. It is also worth noting that the complexities (area) as presented in [15] are for a single number (say X) and so would have to be doubled in order to take care of two numbers (say X and Y) for the reverse conversion process. Table 1 presents the complexities and delay by the various schemes for the purpose of comparison.

Table 1: Area and Delay analysis of proposed scheme with similar schemes of equal DR

Scheme	**AREA**	**DELAY**
[15]	$(28n + (5n^2/2) + 12)\Delta_{FA}$	$(18n + 23)D_{FA}$
[8]	$(48n + 21)\Delta_{FA}$	$(16n + 15)D_{FA}$
Proposed Scheme	$(34n + 10)\Delta_{FA}$	$(16n + 11)D_{FA}$

From Table 1, it is obvious that the proposed scheme is better than [8] in terms of the area complexities even though the delay is almost the same, but the proposed scheme has a correction component. Also, the proposed scheme performs better than the scheme by [15] for higher values of n, in both area and delay. A detailed analysis is presented in Table 2 taking some values of n.

Table 2: Area, Delay analysis for various values of n for

	AREA			**DELAY**		
N	***[15]***	***[8]***	***Proposed***	***[15])***	***[8]***	***Proposed***
1	42.5	69	44	41	31	27
2	78	117	78	59	47	43
4	164	213	146	95	79	75
8	396	405	282	167	143	139
16	1100	789	554	311	271	267
32	3468	1557	1098	599	527	523
64	12044	3093	2186	1175	1039	1035
128	44556	6165	4362	2327	2063	2059
256	171020	12309	8714	4631	4111	4107
512	669708	24597	17418	9239	8207	8203
Total	**902577**	**49314**	**34882**	**18644**	**16518**	**16478**

scheme

Table 2 shows detailed analysis of the area and delay comparison of scheme3 for various values of n with similar-state of the art schemes. The results from Table 2 are used to plot the graphs in Figure 4 and Figure 5; Figure 4 is a graph of area comparison of the various schemes. It shows that the proposed scheme requires the lesser area than the other schemes. Figure 5 also presents the graph of the delay comparison of the compared schemes which shows however that the proposed scheme and the scheme by [8] have almost the same speed but performs better than the scheme by [15].

Figure 4: Graph of area analysis of proposed scheme3 with other schemes

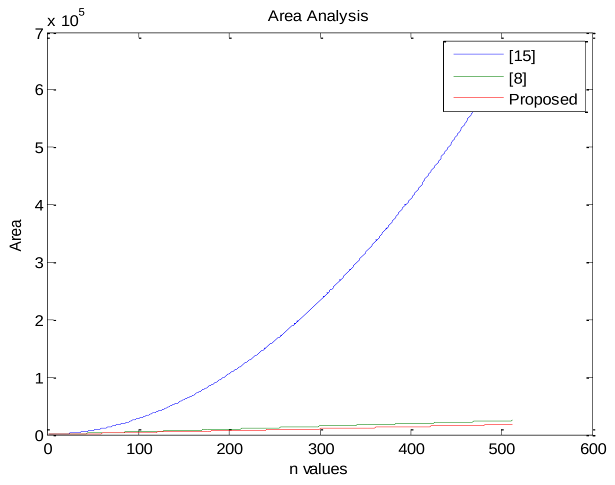
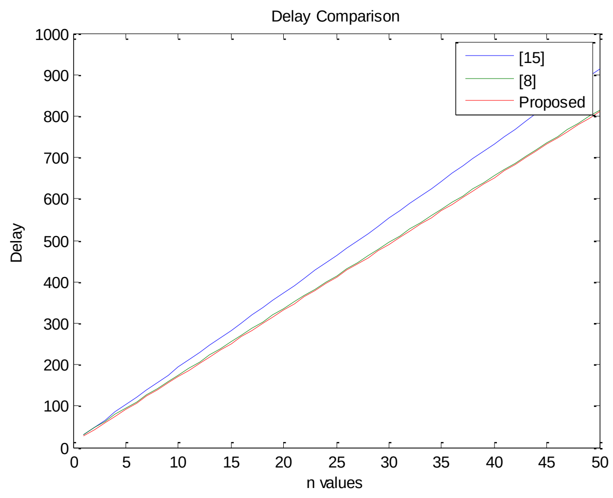


Figure 5: Graph of delay analysis of proposed scheme3 with other schemes



5. Conclusion

In this paper, an additive overflow detection and correction scheme for the moduli set $\{2^{2n+1} - 1, 2^n + 1, 2^n - 1\}$ was presented. The scheme used a redundant modulus 2 to extend the dynamic range of the moduli set. Overflow was then detected during addition whenever it occurred by XORing the residue corresponding to the redundant modulus and the LSB of the result by summing the residues corresponding to two numbers in the original moduli set. The proposed scheme was demonstrated theoretically to be an efficient scheme by comparing it to previous similar works. Practical implementation of the proposed scheme using Field Programmable Gate Arrays (FPGAs) will actualise the real gains as desired by the researchers since this was a limitation on the work due to the unavailability of such practical tools. Therefore, any future works would be focused on the practical implementation of the proposed scheme.

References

[1] A. Omondi and B. Premkumar, *Residue Number Systems: Theory and Implementation*, vol. 2. Published by imperial college press and distributed by world scientific publishing co., 2007.

[2] N. Singh, "An overview of Residue Number System," presented at the National Seminar on Devices, Circuits & Communication, Mesra, Ranchi, 2008.

[3] P. A. Agbedemnab and E. K. Bankas, "A Novel RNS Overflow Detection and Correction Algorithm for the Moduli Set {2^n-1,2^n,2^n+1}," *Int. J. Comput. Appl.*, vol. 110, no. 16, pp. 30–34, Jan. 2015.

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

[5] D. Younes and P. Steffan, "Universal Approaches for Overflow and Sign Detection in Residue Number System Based on {2n – 1, 2n, 2n + 1}," presented at the ICONS 2013, The Eighth International Conference on Systems, 2013, pp. 77–81.

[6] M. I. Daabo and K. A. Gbolagade, "RNS Overflow Detection Scheme for the Moduli set {M − 1, M}," *J. Comput.*, vol. 4, no. 8, pp. 39–44, 2012.

[7] R. C. Debnath and D. A. Pucknell, "On multiplicative overflow detection in residue number system," *Electron. Lett.*, vol. 14, no. 5, pp. 129–130, Mar. 1978.

[8] M. Askarzadeh, M. Hosseinzadeh, and K. Navi, "A New Approach to Overflow Detection in Moduli Set 2^n-3,

2^n-1, 2^n+1, 2^n+3," in *Second International Conference on Computer and Electrical Engineering, 2009. ICCEE '09,* 2009, vol. 1, pp. 439–442.

[9] M. Rouhifar, M. Hosseinzadeh, S. Bahanfar, and M. Teshnehlab, "Fast Overflow Detection in Moduli set {2^n-1, 2^n, 2^n+1}," *Int. J. Comput. Sci. Issues*, vol. (8/3), pp. 407–414, May 2011.

[10] S. J. Piestrak, "A high-speed realization of a residue to binary number system converter," *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, pp. 661–663, Oct. 1995.

[11] H. Siewobr and K. A. Gbolagade, "RNS Overflow Detection by Operands Examination," *Int. J. Comput. Appl.*, vol. 85, no. 18, pp. 1–5, Jan. 2014.

[12] E. K. Bankas and K. A. Gbolagade, "A New Efficient RNS Reverse Converter for the 4-Moduli Set {2^n, 2^n+1, 2^n-1, 2^{2n+1}-1}," *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 8, no. 2, pp. 318–322, 2014.

[13] A. S. Molahosseini, K. Navi, C. Dadkhah, O. Kavehei, and S. Timarchi, "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 Trans. Circuits Syst. Regul. Pap.*, vol. 57, no. 4, pp. 823–835, Apr. 2010.

[14] E. K. Bankas and K. A. Gbolagade, "A New Efficient FPGA Design of Residue-To-Binary Converter," *Int. J. VLSI Des. Commun. Syst. VLSICS*, vol. 4, no. 6, Dec. 2013.

[15] P. V. A. Mohan, "New reverse converters for the moduli set {2^n-3, 2^n-1, 2^n+1, 2^n+3}," *Int. J. Electron. Commun.*, vol. 62, pp. 643–658, 2008.

Peter A. Agbedemnab has a BSc in Computer Science and MSc with Research in Computational Mathematics from the University for Development Studies (UDS), Ghana in 2008 and 2015 respectively. He is a professional teacher with over ten years of experience in teaching and research work. He is currently an Assistant Lecturer in the Department of Computer Science, UDS. His research areas include but not limited to Computer Arithmetic, Residue Number System, Information Security and Genetic Algorithm.

Moses A. Agebure received his BSc degree in computer science from the University for Development Studies, Tamale, Ghana in 2008 and master of philosophy degree in Computer Engineering from the University of Ghana, Accra, Ghana in 2014. After his first degree, he worked as a senior research assistant at the Department of Computer Science of the University for Development Studies and is currently an assistant lecturer in the same Department of the University for Development Studies. He has co-authored papers published in journals. His research interests include, data mining, machine learning, software engineering and mobile computing systems.

Stephen Akobre received his Bsc. degree in Computer Science in 2006 and Msc. degree in Telecommunications Engineering in 2011 from the Kwame Nkrumah University of Science and Technology, Kumasi-Ghana. In 2007 he joined the University for Development studies as a research assistant. He is now a lecturer at the Department of Computer Science, University for Development Studies, Navrongo Campus. His research interest include effect of propagation impairments on satellite communications systems, data mining, big data and machine learning.