Overflow Detection in Residue Number System, Moduli Set $\{2^n\text{-}1,2^n,2^n+1\}$

Babak Tavakoli[1], Mehdi Hosseinzadeh[2], Somayeh Jassbi[3]

Received (2015-11-23)
Accepted (2016-2-1)

***Abstract* — Residue Number System (RNS) is a non-weighted number system for integer number arithmetic, which is based on the residues of a number to a certain set of numbers called module set. The main characteristics and advantage of residue number system is reducing carry propagation in calculations. The elimination of carry propagation leads to the possibility of maximizing parallel processing and reducing the delay. Residue number system is mostly fitted for calculations involving addition and multiplication. But some calculations and operations such as division, comparison between numbers, sign determination and overflow detection is complicated. In this paper a method for overflow detection is proposed for the special moduli set {2^n-1,2^n,2^n+1}. This moduli set is favorable because of the ease of calculations in forward and reverse conversions. The proposed method is based on grouping the dynamic range into 2^{2n}-2^n groups by using the New Chinese Theorem and exploiting the properties of residue differences. Each operand of addition is mapped into a group, then the sum of these groups is compared with the indicator and the overflow is detected. The proposed method can detect overflow with less delay comparing to previous methods.**

***Index Terms* — Computer Arithmetic, Moduli set , Overflow Detection, Residue Number System.**

1- Department of Computer Engineering, Islamic Azad University, Science and Research Branch (babak_tavakoli@yahoo.com)
2- Department of Computer Engineering, Islamic Azad University, Science and Research Branch (mehdi.hoza@gmail.com)
3- Department of Computer Engineering, Islamic Azad University, Science and Research Branch (sjassbi@gmail.com)

I. INTRODUCTION

RNS has recently attracted attention. The need for high speed and low power computation with the emergence of new mobile and embedded applications are the driving force behind this attraction. RNS is an answer to those new challenges; the carry free arithmetic which can be used in RNS will result in development of parallel fast computation methods[1]. It has proven its advantages in the achievement of high performance parallel computing. In RNS a number is divided into several much smaller numbers which can be manipulated concurrently and independently. The system is mostly beneficial in the applications where addition and multiplication are dominant; such applications are found in digital signal and image processing and cryptography[1]. RNS is also useful in fault tolerant applications; due to the non-weighted nature of this system, a single error in a digit will not affect the whole number so the faulty digit may be discarded with no effect other than a small reduction in dynamic range. However RNS has also its own disadvantages and limitations which restrict its application as a primary choice for designing an all-purpose CPU. Some operations and calculations are complicated in RNS[2]; division, magnitude comparison, sign determination and overflow detection are among those difficult operations in RNS which overshadow the full utilization of the RNS in a general purpose computer system.

II. DEFINITIONS AND NOTATIONS

An RNS is distinguished by its moduli set; a set of N pairwise prime integers such as $\{m_N, m_{N-1}, \dots m_1\}$. Each member of moduli set is called a modulus. An integer number *X* is represented as a set of *N* residues according to the equation (1) with respect to the moduli set [3]. The remainder on division of *X* by the modulus m_i is defined as x_i and represented by equation(2).

$$X = x_N, x_{N-1}, \ldots, x_1 \quad (1)$$

$$x_i = |X|_{m_i} \quad (2)$$

The product of the moduli is defined as *dynamic range M* and it is the upper bound of representable numbers in a RNS according to the equation (3).

$$M = \prod_{i=1}^{N} m_i \;\; and \;\; 0 \le X < M \quad (3)$$

Addition, subtraction and multiplication are carried out by individually modular adding, subtracting or multiplying corresponding residues relative to the modulus for their position [3].

$$\begin{aligned} X \pm Y &= |x_N \pm y_N|_{m_N}, \ldots, |x_1 \pm y_1|_{m_1} \\ X \times Y &= |x_N \times y_N|_{m_N}, \ldots, |x_1 \times y_1|_{m_1} \end{aligned} \quad (4)$$

Forward Conversion is the process in which a number from ordinary (binary) representation is converted to RNS representation. The process is finding the remainder of the number upon division by every modulus. Forward conversion is almost always carried out by any technique, other than division which is complex and time consuming.

Reverse conversion is the process in which a number in RNS representation is converted back to an ordinary (binary) representation. There are two different approaches to the reverse conversion; the first method is based of *Chinese Remainder* (Equation (5)) and the second method is called Mixed Radix Conversion which is a sequential process (Equation (6)).

$$X = \left| \sum_{i=1}^{N} w_i x_i \right|_M , w_i = M \times \left| M_i^{-1} \right|_{m_i} , M_i = \frac{M}{m_i} \quad (5)$$

$$X = z_1 + z_2 m_1 + z_3 m_1 m_2 + \ldots + z_N m_1 \ldots m_{N-1} \quad (6)$$

III. BACKGROUND

A. Problem Statement

Overflow occurs whenever the result of an arithmetic operation could not be represented correctly by hardware. i.e. the result falls beyond the representable range. Overflow can be easily detected in a weighted number system by comparing methods; the sum of two unsigned integers is expected to be greater than each addend or the sum of two positive/negative integers should not yield a negative/positive result. Overflow can also be detected by examining the equality between carry-in and carry-out of the most significant bit. RNS is non-weighted i.e. the digit position carries no weight information so the overflow detection is complicated and it is almost as complex as magnitude comparison. Although most RNS applications avoid complicated operations, the detection of overflow is inevitable to ensure the correctness and integrity of arithmetic operations. There are also some other methods based on redundancy [12].

B. Overflow Detection Methods

There are three main approaches to overflow detection. The first approach relies on the parity characteristics[6][8], the second is based on grouping the dynamic range[7][9] and the third is by comparison.

1- Adding an extra modulus 2 to the moduli set which has no even modulus, expands the dynamic range and adds the parity bit to each number. The sum of two even/odd numbers naturally results in an even number, the sum of an even number and odd number will produce an odd number. Any violation of the above mentioned rule will indicate an overflow.

2- Overflow can also be detected by Grouping the dynamic range into different partitions and defining comparing rules on groups. The advantage of this method is reducing the comparison complexity; the comparison between groups is performed in the binary number system with less bits.

3- The other method for overflow detection is based on comparison. A mechanism for comparing numbers is hired which is similar to reverse conversion. The numbers are partially converted to a weighted number system in which the comparison can be performed. The efficiency of these methods is fully dependent on the comparison and reverses conversion mechanisms. Some methods such as "Core function"[4] are used to evaluate an RNS number without full reverse conversion.

IV. PROPOSED METHOD

Our proposed method can detect overflow in adding two unsigned integers. Each operand and the sum are mapped to a group by using the grouping function. The sum of groups is calculated and compared with the indicator 2^{2n} - 2^{2-1} . If the sum of groups is greater than indicator the over flow signal will be set *(ov=1)*, if it is less than indicator the over flow signal will be cleared *(ov=0)* and if it is equal to indicator the group associated with the sum of operands will be compared with zero. If the group is equal to zero then over flow signal will be set and otherwise it will be cleared according to equations (7). The method is illustrated in figure (1).

$$
\begin{aligned}
& Z = X + Y \\
& \text{if } G(X) + G(Y) > 2^{2n} - 2^2 - 1 \rightarrow ov = 1 \\
& \text{if } G(X) + G(Y) < 2^{2n} - 2^2 - 1 \rightarrow ov = 0 \\
& \begin{cases} \text{if } G(X) + G(Y) = 2^{2n} - 2^2 - 1 & \\ \quad \text{and} \quad G(Z) = 0 & \rightarrow ov = 1 \\ \quad \text{otherwise} \quad ov = 0 & \end{cases}
\end{aligned} \tag{7}
$$

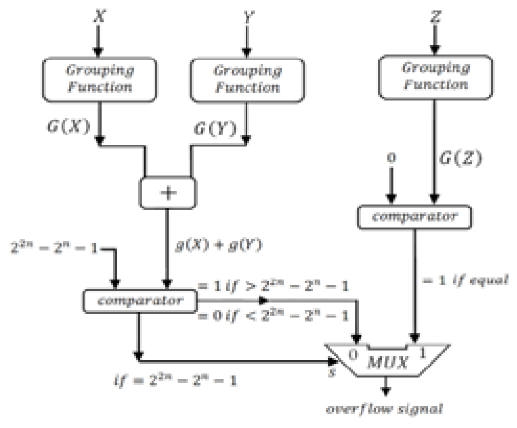

Figure 1. Schematic of Proposed Method

A. *Grouping Function*

Grouping function maps every number within the dynamic range into a group. The dynamic range is divided into 2^{2n}-2^2 groups. Each group has 2^n+1 members. The grouping is based on difference of residues. The difference between x_3 and x_2 modulo m_3 is defined as A' and A'' is one bit right rotate of A' [9]. The difference between x_2 and x_1 modulo m_2 is defined as B'. The grouping function is defined in equation (8) and it is depicted in figure (2).

$$
\begin{aligned}
& X = x_3, x_2, x_2 \quad m_3 = 2^n - 1, m_2 = 2^n, m_1 = 2^n + 1 \\
& A' \square \left|x_3 - x_1\right|_{m_3}, B' \square \left|x_2 - x_1\right|_{m_2} \\
& A'' \square \text{ one bit right rotate } (A') \\
& G(X) = B' + \left|A'' - B'\right|_{m_3} . m_2
\end{aligned} \tag{8}
$$

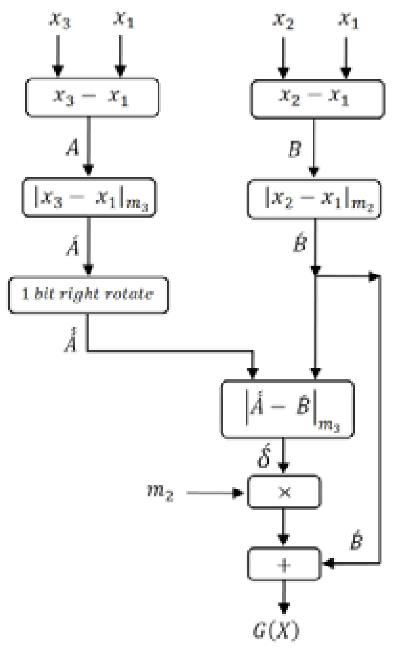

Figure 2. Schematic of Grouping Function

B. *Mathematical Proof*

In this section we present two mathematical proofs for our proposed method. The first proof assumes the overflow has occurred and shows the sum of groups exceeds the indictor. The second (reverse) proof assumes the sum of the groups is greater than indicator as the hypotheses and concludes the sum of two numbers is greater than or equal to dynamic range. In both proofs we assume that Z is the sum of two unsigned integers *X* and *Y* (*Z=X+Y*) and *X,Y,Z* are mapped to *G(X),G(Y),G(Z)* by grouping function respectively.

Before proving the proposed method, two useful lemmas are presented.

Lemma 1: for $a,b \in \mathbb{Z}$ if $a \le b$ then $a < b + 1$

Lemma 2: for $a,b \in \mathbb{Z}$ if $a > b$ then $a \ge b + 1$

$$X + Y \geq 2^{3n} - 2^{n} \Rightarrow G(X) + G(Y) \geq 2^{2n} - 2^{n} - 1$$

$$G(X)\left(2^{n}+1\right) \leq X \leq G(X)\left(2^{n}+1\right) + 2^{n}$$

$$G(Y)\left(2^{n}+1\right) \leq Y \leq G(Y)\left(2^{n}+1\right) + 2^{n}$$

$$X < G(X)\left(2^{n}+1\right) + 2^{n} + 1 \ (lemma\,1)$$

$$Y < G(Y)\left(2^{n}+1\right) + 2^{n} + 1 \ (lemma\,1)$$

$$X + Y < G(X)\left(2^{n}+1\right) + G(Y)\left(2^{n}+1\right) + 2\left(2^{n}+1\right)$$

$$X + Y < \left(2^{n}+1\right)\left(G(X) + G(Y) + 2\right) \ (I)$$

$$X + Y \geq 2^{3n} - 2^{n} \rightarrow X + Y \geq \left(2^{2n} - 2^{n}\right)\left(2^{n}+1\right) \ (II)$$

$$(I),(II):$$

$$\left(2^{n}+1\right)\left(G(X) + G(Y) + 2\right) > \left(2^{2n} - 2^{n}\right)\left(2^{n}+1\right)$$

$$G(X) + G(Y) > 2^{2n} - 2^{n} - 2 \geq 2^{2n} - 2^{n} - 1 \ (lemma\,2)$$

$$G(X) + G(Y) \geq 2^{2n} - 2^{n} - 1$$

$$G(X) + G(Y) > 2^{2n} - 2^{n} - 1 \Rightarrow X + Y \geq 2^{3n} - 2^{n}$$

$$G(X) + G(Y) \geq 2^{2n} - 2^{n} \quad (I) \ (lemma\,2)$$

$$2^{n}\left(G(X) + G(Y)\right) \geq 2^{3n} - 2^{2n} \ (II)$$

$$(I) + (II) \rightarrow G(X) + G(Y) + 2^{n}G(X) + 2^{n}G(Y) \geq 2^{2n} - 2^{n} + 2^{3n} - 2^{2n}$$

$$G(X) + 2^{n}G(X) + G(Y) + 2^{n}G(Y) \geq 2^{3n} - 2^{n}$$

$$G(X)\left(2^{n}+1\right) + G(Y)\left(2^{n}+1\right) \geq 2^{3n} - 2^{n} \quad (III)$$

$$G(X)\left(2^{n}+1\right) \leq X \leq G(X)\left(2^{n}+1\right) + 2^{n},$$

$$G(Y)\left(2^{n}+1\right) \leq Y \leq G(Y)\left(2^{n}+1\right) + 2^{n}$$

$$G(X)\left(2^{n}+1\right) \leq X \ , G(Y)\left(2^{n}+1\right) \leq Y$$

$$G(X)\left(2^{n}+1\right) + G(Y)\left(2^{n}+1\right) \leq X + Y \quad (IV)$$

$$(III),(IV) \rightarrow X + Y \geq 2^{3n} - 2^{n}$$

V. HARDWARE IMPLEMENTATION

In this section the logical design of main components of the proposed method, are being discussed. The grouping unit and the comparators are the building blocks of the proposed method. The grouping unit and the comparators are consist of different parts, each of them are being explained in the following sub-sections.

A. Grouping Function

The grouping operation is started by calculating the differences between the residues according to the following equations: $A = x_3 - x_1$, $B = x_2 - x_1$.

These operations are carried out by using two Parallel Prefix Adders (PPA). Then the remainder of A when dividing by $2^n - 1$ and the remainder of B when dividing by 2^n are calculated according to equation (9).

$$A' \square \left|x_3 - x_1\right|_{2^n - 1}, B' \square \left|x_2 - x_1\right|_{2^n} \qquad (9)$$

The calculation of B' is straightforward; the remainder of any number when dividing by 2^n is the n least significant bits of the dividend. In order to calculate A', a special adder is being used which is illustrated in figure (3).

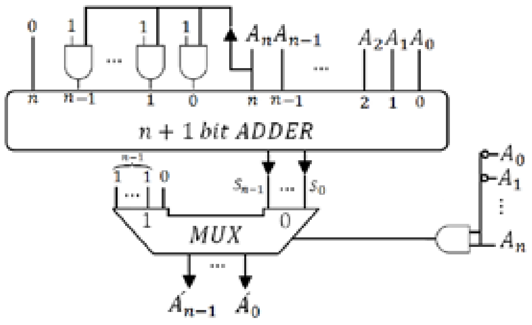

Figure 3. Calculation unit for

As shown in figure (3), "A" is added to a constant offset. The MSB of "A" will determine this constant; when MSB of "A" is zero the offset will be zero otherwise it will be $01\ldots1 = 2^n - 1$ there is also an exception to this rule; when the input is $1\ldots10$, the output should be $10\ldots0$. This situation is handled by using a 2 input multiplexer which is controlled by the exceptional condition. The values of A' are not ordered; i.e. the even values come first and the odd values come after even values. To make these values ordered we should rotate the A' one bit to the right [9]. A'' is the result of one bit right rotation of A'. The remainder of difference between A'' and B' when dividing by $m_3 = 2^n - 1$ is δ' and it is calculated by using the same mechanism as A'. The exceptional condition does not occur and the extra hardware which handles the exceptions can be eliminated figure (4). When δ' is calculated, it should be multiplied by $m_2 = 2^n$ and added to B'. The multiplication of any number by 2^n is just an “n bit” left shift or right padding the number with n zeros. While B' is an “n bit” variable, adding B' to the $2^n \times \delta'$ is carried out by placing the B' bits on the right side of δ' replacing the n zeros without using any hardware.

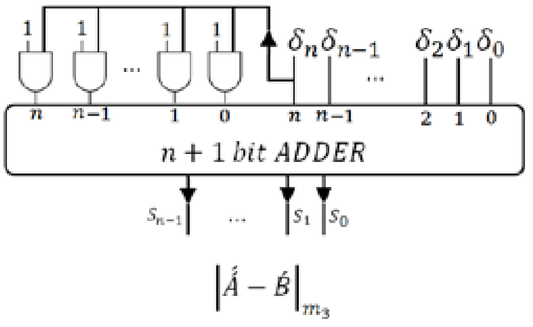

Figure 4. Calculation unit for δ'

B. Group Addition

When the groups for each operand are calculated by grouping module, the groups should be added to form the “sum of groups”. The result is compared with the indicator to detect the overflow. While groups are 2n bit variables, the adder should be a 2n bit adder. In order to avoid using bigger than n bit component, we use three n bit adder instead of one 2n bit adder. These method uses up more area while reducing the delay. The group adder is depicted in figure (5).

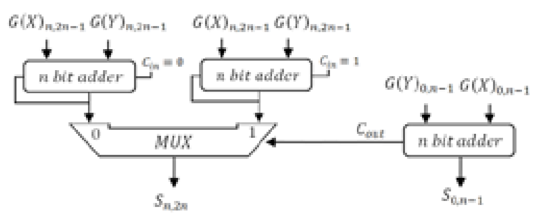

Figure 5. Group adder

The left n bits of *G(X)* and *G(Y)* are added in two n bit adder with and without carry-in. The right n bits are added with third n bit adder. By using a multiplexer, the carry-out of the right adder determines which result will be selected for the high n bits of the sum.

C. Comparing with Indicator

The sum of groups should be compared with the indicator. The indicator is one unit less than, the number of groups,i.e. $2^{2n} - 2^n - 1$. Generally the comparison is carried out by subtraction and testing the sign bit of difference. We would like to avoid subtraction so we offer a special method to predict the sign bit of the difference between sum of groups and indicator without subtraction. The method is completely dependent on the characteristics of the indicator. The two’s complement (negative form) of indicator bit pattern is $11\underbrace{0..0}_{n-1}1\underbrace{0..0}_{n-1}1$. It is composed of two “1 bits” on the left followed by two (n bit) blocks of $\underbrace{0..0}_{n-1}1$ on the right. The sum of groups should be added to two”s complement of indicator then, the sign bit of result will be tested to determine whether sum of groups is greater than indicator or less than indicator. Groups are 2n bit variables so the sum of groups is 2n+1 variable, the indicator is also 2n bit variable and it should be considered that, the indicator will be compared with a 2n+1 bit variable (the sum of groups) so one extra bit is added to the left side of indicator. The complement operation also adds another bit to extend the range and make room for negative values and the final length of minus-indicator will be 2n+2. One zero bit is added to the left side of “sum of groups” to match the 2n+2 bit length of minus-indicator. As illustrated in figure (6), the minus indicator is consist of two n bit blocks followed by two “1” at the left. The sum of groups is shown above the minus indicator.

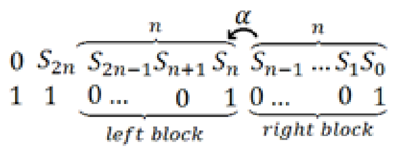

Figure 6. The minus-indicator in two blocks

We define the variable as the carry bit coming out of right block. If all bits of $S_{n-1} \cdots S_1 S_0$ equal to 1, the carry will be generated i.e. α equals to 1, otherwise α equals to 0. If $\alpha = 0$ the final carry-out of left block is dependent on the $S_{2n-1} \cdots S_{n+1} S_n$ if all bits of $S_{2n-1} \cdots S_{n+1} S_n$ equal to 1, then the carry-out of left block equals to 1. If $\alpha = 0$ the final carry-out is dependent on the $S_{2n-1} \cdots S_{n+1}$ similar to the previous situation, if all bits of $S_{2n-1} \cdots S_{n+1}$ equal to 1, then the carry-out of left block equals to 1. The carry bit coming out of left block combining with S_{2n} will determine the final sign bit as depicted in figure (7).

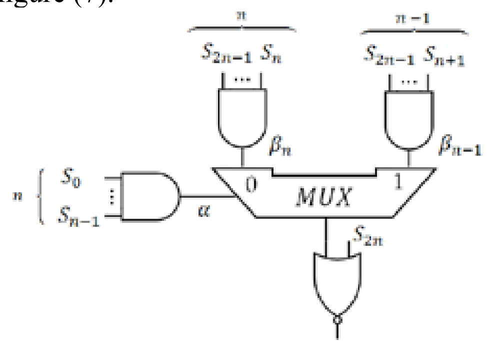

Figure 7. Schematic of comparator

The comparator can be implemented by using one "n input AND gate", one "n-1 input AND" gate, three "2 input AND" gates and one "2 input NOR" gate as illustrated in figure (8).

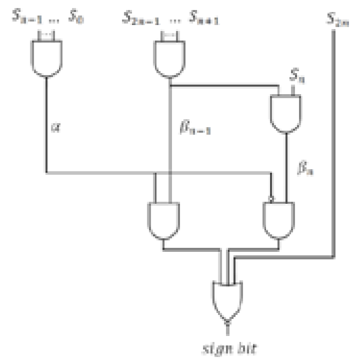

Figure 8. Implementation of comparator

The comparator descried above can determine whether the sum of groups is greater than indicator or less than it. It cannot tell if the sum of groups is equal to indicator. To have a complete comparator we should use another unit for "equality comparison", this unit can be made of "XOR" gates combining with one "n input AND gate" figure (9).

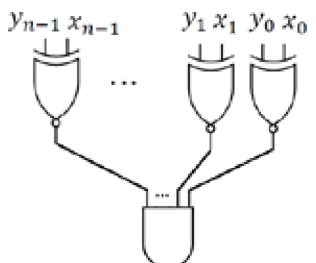

Figure 9. Equality comparison between X and Y

The "n input AND" gate is a hypothetical gate which cannot be implemented directly similar to a regular "2 input AND" gate. The "fan-in" is a parameter of synthetic technology which limits the inputs of any gate. Regarding to the restrictions imposed by VLSI technology and fabrication methods, the number of inputs for any gate is limited. In our proposed method we consider the worst case; we suppose that, we cannot use any gate which has more than two inputs. So implementing the "N input AND" gate is accomplished by cascading, n-1 "2 input AND" gate in a tree like design. An example of a 16 input AND gate is illustrated in Figure (10).

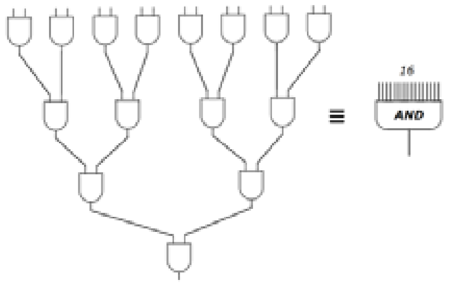

Figure 10. Implementation of 16 Input AND gate

VI. COMPLEXITY AND DELAY CALCULATION

In this section the area and delay of proposed method is calculated. The procedure for calculating the area and delay is based on uni-gate [5] method. In this method each basic gate has 1 unit delay and 1 unit area. The basic gates are "AND","OR","NOR" and "NAND". The compound gates such as "XOR" and "XNOR" have 2 units of delay and 3 units of area. Our proposed method is consist of 3 grouping units two comparators (one for comparing with indicator and other for comparing with zero) and a multiplexer. We also suppose all of adders are "Parallel Prefix" which has the delay of $T_{PPA} = 2\log_2 n + 4$, and the area of $A_{PPA} = 5n + \frac{3}{2} n \log_2 n$ [5], the area and delay of grouping unit can be calculated as follows:

$$A_{Grouping} = 10n + 7 + \frac{3}{2}(n+1)\log_2(n+1)$$

$$+6(n+1) + -(n+1)\log_2(n+1) \approx 16n + 3n\log_2(n+1) + 7$$

$$T_{Grouping} = 2\log_2(n+1) + 5 + 3\log_2(n+1)$$

$$+9 = 5\log_2(n+1) + 14$$

If we add all the area and delay of components considering the parallel and serial parts the total delay and area can be calculated as follows:

$$A_{Total} = 3A_{Grouping} + A_{Group\ Adder} + \underbrace{A_{Comparator} + A_{is\ equal}}_{comparator}$$

$$+A_{is\ zero} + A_{MUX\ 2\times1\ (1bit}$$

$$A_{Total} = 3\left(16n + 3n\log_2(n+1) + 7\right) + \left(17n + \frac{9}{2}n\log_2 n\right)$$

$$+\underbrace{(2n+2) + (3n-1)}_{comparator} + (2n-1) + 3$$

$$A_{Total} = 72n + \frac{27}{2}n\log_2(n+1) + 10$$

$$T_{Total} = T_{Grouping} + T_{Group\ Adder} + T_{Comparator} + A_{MUX\ 2\times1\ (1bit)}$$

$$T_{Total} = \left(5\log_2(n+1) + 14\right) + \left(6\log_2 n + 2\right) + \left(\log_2 n + 3\right) + 2$$

$$T_{Total} = 12\log_2(n+1) + 21$$

We substitute all $\log_2 n$ with $\log_2(n+1)$ to form a uniform equation. This substitution will not violate the uni-gate rules; because the upper bound of delay (the worst case) is used.

VII. COMPARISON

There are few methods which directly and specifically dealt with overflow detection in moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. Some of the fast and efficient (in terms of complexity and area) are compared with our proposed method in table 1.

TABLE I. COMPARISON BETWEEN OVERFLOW DETECTION METHODS

Method	Area	Delay
[11]	$115n + 186$	$4n + \log_2 n + 36$
[10]	$96n + n\log_2 n + 16$	$8n + \log_2 n + 12$
$[13] - CI$	$56n + 22 + A_{BC}$	$16n + 4 + \tau_{BC}$
$[13] - CII$	$96n + 24 + A_{BC}$	$4n + 4 + \tau_{BC}$
$[13] - CIII$	$80n + 18 + A_{BC}$	$4n + 4 + \tau_{BC}$
[9]	$76n + \left(\frac{33}{2}\right)n\log_2 n$	$6\log_2 n + 23$
Proposed Method	$72n + \frac{27}{2}n\log_2(n+1) + 10$	$12\log_2(n+1) + 21$

VIII. CONCLUSIONS

Overflow detection is an essential part of all arithmetic operations. Undoubtedly any ALU needs a mechanism for overflow detection to insure that, results are correct and to provide a guarantee against possible out of range, incorrect calculation results. Overflow detection is considered as a complicated operation with the complexity almost equal to magnitude comparison and sign detection. In this paper we proposed an overflow detection mechanism based on grouping. Our method shows less delay and area, comparing to previous methods and all the operations are n bit long, avoiding complexity of processing 2n bit and 3n bit numbers which are common among previous algorithms.

REFERENCES

[1] Omondi A., Premkumar B., RESIDUE NUMBER SYSTEMS, Theory and Implementation, ImperialCollege Press, 2007.

[2] N. S. Szabo and R. I. Tanaka, Residue Arithmetic and Its Application to Computer Technology, New York: McGraw- Hill, 1967.

[3] Parhami B., Computer arithmetic Algorithms and hardware designs, Oxford University Press, 2000.

[4] Miller D.D., Analysis of the residue class core function of Akushskii, Burcev, and Park. In: G. Jullien, Ed., RNS Arithmetic: Modern Applications in Digital Signal Processing. IEEE Press 1986.

[5] Zimmermann R., Efficient VLSI implementation of modulo 2n+-1 addition and multiplication. Proc. 14th IEEE Symp. Computer Arithm. , Adelaide, Australia, 1999, pp. 158–167

[6] Zarei B., Askarzadeh M., Derakhshanfard N.,Hosseinzadeh M., A High-speed Residue Number Comparator For the 3-Moduli Set {2n-1, 2n+1, 2n+3}. Proceedings of International Symposium on Signals,

Systems and Electronics (ISSSE2010),2010

[7] DEBNATH R.C., PUCKNELL D.A.,ON MULTIPLICATIVE OVERFLOW DETECTION IN RESIDUE NUMBER SYSTEM,Department of Electrical Engineering University of Adelaide, 1977

[8] Askarzadeh M.,Hosseinzadeh M., Keivan Navi K.,A New approach to Overflow detection in moduli set {2n-3, 2n-1, 2n+1, 2n+3},Second International Conference on Computer and Electrical Engineering,2009

[9] Rouhifar M.,Hosseinzadeh M.,Bahanfar S.,Teshnehlab M.,Fast Overflow Detection in Moduli Set {2n – 1, 2n, 2n + 1},IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011

[10] Shaoqiang B., Groos W.J.,Efficient Residue Comparison Algorithm for General Moduli Sets,IEEE International Circuits and Systems, 2005, (2): 1601-1604.

[11] Gholami E., Farshidi R., Hosseinzadeh M., Navi K.,High speed residue number system comparison for the moduli set {2n-1, 2n, 2n+1},Journal of Communication and Computer, ISSN 1548-7709, USA,2009.

[12] TAI L.C.,CHEN C.F., Overflow Detection In a Redundant Residue Number System,IEE PROCEEDINGS, Vol. 131, Pi. E, No. 3, MAY 1984

[13] Wang Y., Song X., Abdolhamid M., Shen H., Adder based residue to binary converters for {2n-1, 2n, 2n+1}, 2002, pp. 1772-1779