Design and Implementation of An RNS Division Algorithm

Ahmad A. Hiasat
Elect. Eng. Dept.
Princess Sumaya University
PO Box 1438, Amman 11941, JORDAN
Email: aahiasat@rss.gov.jo

Hoda S. Abdel-Aty-Zohdy
Elect. & Sys. Eng. Dept.
Oakland University
Rochester, MI 48309, USA
Email: zohdyhsa@oakland.edu

Keywords: Division, Residue Number System, Fractional Representation, Hardware Implementation.

Abstract: In a recent publication [1], we introduced the main outlines of a new algorithm for division in Residue Number System, which can be applied to *any* moduli set. Simulation results proved that the algorithm was many times faster than most competitive published work [2]. Determining the position of the most significant nonzero bit of any residue number in that algorithm is the major speed limiting factor. In this paper, we customize the same algorithm to serve two specific moduli sets: $(2^k, 2^k-1, 2^{k-1}-1)$ and $(2^k+1, 2^k, 2^k-1)$, and thus, eliminate that speed limiting factor. Based on this work, hardware needed to determine most significant bit position has been reduced to a single adder. Therefore, computation time and hardware requirements are substantially improved. This would enable RNS to be a stronger force in building general purpose computers.

1 Introduction

A basic feature of Residue Number System (RNS) (redundant or nonredundant) is being an unweighted (nonpositional) numbering system [3]-[4]. Thus, digits in a RNS have no ordering significance. Furthermore, in addition, subtraction and multiplication, any particular digit of the resultant depends solely on the corresponding digits of its suboperation. This parallel property of RNS makes it capable of performing carry-free addition and multiplication operations, and borrow-free subtraction. Moreover, multiplication is executed in a single step without partial products. Therefore, RNS usage has been limited to some digital signal processing computations, where addition, subtraction and multiplication are the only needed operation. This is due to the known difficulty of residue operations like division, sign and overflow detection.
The moduli sets $(2^k, 2^k-1, 2^{k-1}-1)$ and $(2^k+1, 2^k, 2^k-1)$ are particularly importan in applications which require high degree of precision [4]. The properties of this set become more apparent in hardware considerations because most moduli are powers or diminished powers of 2, so residue addition is the carry-add type, and multiplication by a power of 2 is equivalent to left rotation. Moreover, residues are encoded with, almost, full efficiency. Therefore, they can play an increased role in implementing a Residue Number System (RNS) arithmetic unit for computers.

2 Notation

The following notational convention has been adopted for this paper:

- $\{m_1, m_2, \ldots\ldots, m_N\}$, moduli set of N relatively prime positive integers.
- $M = \prod_{i=1}^{N} m_i$, dynamic range.
- For any integer $X \in [0, M)$, residue representation of X is:
  $X \overset{RNS}{\rightarrow} (r_1, r_2, \ldots\ldots, r_N)$
- $r_i = |X|_{m_i}$, least positive remainder when dividing X by m_i.
- $\hat{m}_i = \frac{M}{m_i}$.
- $|\frac{1}{\hat{m}_i}|_{m_i}$, multiplicative inverse of $\hat{m}_i$ (i.e. $\left|\hat{m}_i \left|\frac{1}{\hat{m}_i}\right|_{m_i}\right|_{m_i} = 1$)
- Define a function h(I) such that:

$$h(I) = \begin{cases} 1 + \lfloor log_2\, I \rfloor & \text{, if I is an integer} > 0 \\ 0 & \text{, if I} = 0 \\ \lfloor log_2\, I \rfloor & \text{, if } 0 < I < 1 \end{cases}$$

3 Residue To Binary Conversion

In order to decode a residue number system into a weighted number system, there are two approaches [3]:

Chinese Remainder Theorem (CRT) and Mixed-Radix Conversion (MRC). CRT will be introduced here, since new implementation of this theorem [5]-[6] makes it faster for conversion purposes than other conversion techniques which are based on mixed radix system [4]. For an N moduli set, CRT utilizes one level of N RAMs and one level of adder(s) [5]-[6].

Chinese Remainder Theorem: Many techniques can be found in literature which decode RNS into an equivalent Binary system. The basic theorem which governs this decoding scheme is given by CRT, expressed as [3]:

$$X = \left|\sum_{i=1}^{N} \hat{m}_i \left|\frac{x_i}{\hat{m}_i}\right|_{m_i}\right|_M \tag{1}$$

In 1985, Van Vu [6] developed two conversion techniques based on the CRT. The first technique adopts the fractional representation of X/M, which is useful in applications when the sign and general magnitude information about X is needed. This technique was earlier proposed by Soderstrand [5] and then modified by Van Vu[6]. The modification was regarding the number of bits for accurate representation of the fractional value. The other technique, introduced by Van Vu, is useful where the representation of X as an integer value is required. To have this paper self-contained, the first technique will be introduced briefly.

Fractional Representation Technique: Eq.(1) can be rewritten as

$$X = \sum_{i=1}^{N} \frac{M}{m_i} \left|\frac{r_i}{\hat{m}_i}\right|_{m_i} - Mp \tag{2}$$

where p is a non-negative integer. Dividing both sides of eq.(2) by M or by $M/2$ for unsigned or signed conversion respectively, yields:

$$\frac{X}{M} = \sum_{i=1}^{N} \frac{1}{m_i} \left|\frac{r_i}{\hat{m}_i}\right|_{m_i} - p \tag{3}$$

$$\frac{X}{M} = \sum_{i=1}^{N} \frac{2}{m_i} \left|\frac{r_i}{\hat{m}_i}\right|_{m_i} - 2p \tag{4}$$

Unsigned integers are considered in this paper, however, the same concepts still apply for signed integers, using the proper equation of fractional representation. Hence, the value of $\frac{X}{M}$ can be obtained by evaluating the right hand side of eq.(3). This is basically done by letting each x_i addresses a table which stores $\frac{1}{m_i}|\frac{x_i}{\hat{m}_i}|_{m_i}$. The output of these N tables can be added using multioperand adder . Any integer overflow resulting from this adder is disregarded since it represents the value p (multiplies of M). The fractional value stored in tables should be expressed using t bits where t $\geq \lceil log_2 MN \rceil$ if M is odd and t $\geq \lceil log_2 MN \rceil - 1$ otherwise [6].

4 Division Algorithm:

Assume that X, Y and Q are non-negative integers such that $Q = \lfloor \frac{X}{Y} \rfloor, Y \neq 0$, then the following steps introduce the basic idea for division in RNS [1]:

1. Set Quotient Q to zero; Q = 0.
2. Find the position of the most-significant non-zero bit in the divisor Y , say k, that is $k = h(Y)$.
3. Find the position of the most-significant non-zero bit in the dividend X , say j, that is $j = h(X)$.
4. If $j > k$, then:
   $Q' = Q + 2^{j-k-1}$
   $X' = X - 2^{j-k-1} * Y$
   $Q = Q'$, $X = X'$.
   Go To Step 3.
5. If $j = k$, then:
   $X' = X - Y$
   $j' = h(X')$, so if $j' < j$ then $Q = Q + 1$
   Otherwise, Q is unchanged. End procedure.
6. If $j < k$, then Q is unchanged. End procedure.

This basic algorithm can be used efficiently and effectively with RNS division arithmetic. An important feature of this algorithm is the selection of the quotient to be 2^{j-k-1}, hence, the quantity $X-(2^{j-k-1}*Y)$ is guaranteed to be non-negative as long as $X > Y$. Unlike other algorithms, this will eliminate all hardware and execution time needed for difficult residue operations.

5 Realization Of The Algorithm In RNS

The realization of this new algorithm is still based upon converting the residue representation of the dividend and the divisor to a weighted code in order to derive some information regarding the position of the most-significant non-zero bit contained in a residue number. In this paper, two different realizations are introduced for the new algorithm. The first realization, henceforth called Realization I, is most useful for small dynamic ranges. The other realization, called Realization II, is based on Soderstrand and Van Vu Fractional Representation Technique. This scheme is suitable for large dynamic ranges.

5.1 Realization I:

This realization is quite useful for small and medium dynamic ranges where all the bits of residue digits can be applied to a single RAM in order to evaluate $h(I)$. Moduli sets like (7,11,13,15), (31,32,33), and (11,13,15,16) are among many possible candidates for such a realization.
Realization I can be described by the following steps:

1. Set Quotient Q to zero; Q = 0.

2. Apply Y to a RAM (referred to as RAM 1 in Fig.(1)) to obtain k = h(Y), where k is expressed in r bits, $r = \lceil log_2(log_2 M)\rceil$.

3. Apply X to RAM 1 to obtain j = h(X), j is also expressed in r bits.

4. Apply j and k (or the difference j-k) to a RAM (referred to as RAM2 in Fig.(1)) or a PAL that produces Q_i, where Q_i is the residue representation of 2^{j-k-1} if $j > k$. However, for the case $j = k$, RAM2 will produce the residue representation of 1. Otherwise, the procedure is stopped. The size of this RAM is 2^{2r} or 2^r(depending on whether j and k or their non-negative difference is used). It is worth mentioning that RAM2 can be replaced by a PAL or a simple combinational circuit.

5. A residue multiplier will then multiply Q_i by Y.

6. The output of the multiplier , i.e. Q_iY, is subtracted from X to produce a new remainder. In parallel with the multiplier or the subtractor, a residue adder will accumulate the outputs of RAM2, i.e $Q_{i's}$ to produce the quotient Q.

7. Whenever $j = k$, the output of RAM2 (PAL) for this case is 1. The new remainder would be $X' = X - Y$. The residue adder in this case may need to be incremented by 1 depending on the following: the value of X' is applied to RAM1, if the output of this RAM produces j' where $j' \geq j$, then this indicates that an overflow has taken place, and the residue adder should not be incremented. However, if $j' < j$, then the residue adder should be incremented by 1 (see Theorem 1). After either case, the iterating procedure is stopped. The output of the adder is Q.

8. For the case $j < k$, then $X < Y$. Procedure is stopped.

This realization requires $log_2 Q$ iterations . Each iteration consists of two consecutive memory cycles followed by two consecutive residue operations. The realization is very attractive for many digital signal processing applications which utilizes small and medium dynamic ranges.

Proof Of Correctness Of The Algorithm (Realization I):
Before proving the algorithm, the following theorem has to be introduced:

Theorem 1 *For any residue integers $X, Y \in [0, M)$, for which $j = h(X)$, $k = h(Y)$ and $j = k \neq 0$ then $X \geq Y$ if $j > j'$, and $X < Y$ if $j' \geq j$, where $j' = h(X - Y)$.*

proof:
Since $j = k$, then X and Y can be expressed as:
$X = 2^{j-1} + a$, and $Y = 2^{j-1} + b$
where: $0 \leq a, b \leq 2^{j-1} - 1$
For the case $X \geq Y$ (i.e. $a \geq b$):
$|X - Y|_M = X - Y \geq 0$, then $X - Y = a - b$, but since $0 \leq a - b \leq 2^{j-1} - 1$,
then $j' = h(X - Y) = h(a - b) < j$.
For the case $X < Y$ (i.e. $a < b$):
$|X - Y|_M = M + X - Y$
$j' = h(M + X - Y) = h(M + a - b)$.
The minimum value of j' happens when $a = 0$, $b = (2^{j-1} - 1)$ and $M = Y + 1$. Upon substituting these values: $j' \geq h(2^{j-1} + 1) = j$, or equivalently: $j' \geq j$.
The proof of the algorithm as given in Realization I is as follows:

- For the case $j > k$, and since $Y < 2^{k+1}$ and $X \geq 2^j$ then $\frac{X}{Y} > 2^{j-k-1} = Q_i$ (i.e Q_i is the ith partial quotient). Hence the estimate of the quotient in each iteration is guaranteed to produce a positive remainder. Assume there are v iterations which satisfy the condition $j > k$, then the total partial quotients resulting from this case is Q, where: $Q = \sum_{i=1}^{v} Q_i$.

- For the case j = k, (i.e. $(v + 1)th$ iteration), two possibilities are expected:

  1. $X < Y$: This case is detected according to Theorem 1 by evaluating $j' = h(X - Y)$. Hence, if $j' \geq j$ then the quotient should not be updated. $Q_{v+1} = 0$. Procedure is then stopped.
  2. $X \geq Y$: This case is detected according to Theorem 1 by evaluating $j' = h(X - Y)$. Hence, if $j' < j$ then the quotient should be incremented by 1. $Q_{v+1} = 1$. Procedure is then stopped.

- For the case $j < k$, it is obvious that $X < Y$, consequently, Q is unchanged and procedure has to be stopped.

Therefore, the Quotient would be: $Q = \lfloor \frac{X}{Y} \rfloor = \sum_{i=1}^{v} Q_i + Q_{v+1}$, where:

$$Q_{v+1} = \begin{cases} 1 & \text{, if } j_{v+1} > j' \\ 0 & \text{, otherwise} \end{cases}$$

$j_{v+1} = h(X)$, in the $(v+1)th$ iteration (i.e. when j=k).

5.2 Realization II:

This realization is concerned with large dynamic ranges which cannot be decoded using a single table. Hence, the fractional representation mentioned earlier will be adopted here to evaluate $h(Y)$ once, and then to evaluate $h(X)$ iteratively.

This realization can be described as follows:

1. Apply the residue digits of the divisor Y to the fractional representation circuit to obtain Y/M. This requires a memory cycle followed by a multi-operand addition (see Fig.(2)).

2. Obtain k using a proper combinational circuit like a priority encoder (not shown in Fig.(2)). $k = h(Y/M)$

3. Apply the residue digits of the dividend (remainder) to the fractional representation circuit to obtain X/M. A memory access cycle and a multi-operand addition are also needed here. Obtain j, where $j = h(X/M)$

4. Apply j and k (a total of $2r$ bits), or their difference, to a RAM or PAL that will produce Q_i. Q_i is the residue representation of X_e/Y_e, where: X_e is an estimated value of X given by: $X_e = 2^j M$. Similarly, Y_e is an estimated value of Y given by: $Y_e = 2^{k+1}M$. Or equivalently $Q_i = 2^{j-k-1}$.

5. The output of the table, Q_i, is applied to a residue multiplier to compute Q_iY.

6. The output of the multiplier (Q_iY) is subtracted from X using a residue subtractor.

7. The output of the residue subtractor is applied again to the fractional representation circuit. (i.e. step 3 above) as long as $j > k$.

8. Whenever $j = k$, the output of the RAM (PAL) for this case is 1. The new remainder would be $X' = X - Y$. The residue adder will or will not be incremented depending on the following: the value of X' is applied to the fractional representation circuit to produce j', where $j' = h(\frac{X'}{M})$. If $j' = -1$, then an overflow has taken place, and the residue adder should not be incremented. However, if $j' \neq -1$, the residue adder should be incremented by 1. After either case, the iterative procedure is stopped. The output of the adder is Q.

9. For the case $j < k$, then this implies that $X < Y$.

Fig.(2) shows the main hardware components of this realization.

Proof Of Correctness Of The Algorithm (Realization II):

Before introducing the proof of the algorithm, the following theorem has, also, to be proved:

Theorem 2 *In RNS, for any fractional representations $\frac{X}{M}, \frac{Y}{M}$ where $X, Y \in [0, M)$, $j = h(\frac{X}{M})$, $k = h(\frac{Y}{M})$ and $j = k$ then $X \geq Y$ if $j' \neq -1$, and $X < Y$ if $j' = -1$, where $j' = h(\frac{X-Y}{M})$.*

proof:

Since $\frac{X}{M}$ and $\frac{Y}{M}$ are of the same order (i.e j = k) that is:

$2^j \leq \frac{X}{M} < 2^{j+1}$, and similarly $2^j \leq \frac{Y}{M} < 2^{j+1}$

For the case $X \geq Y$: $|X - Y|_M = X - Y \geq 0$, then:

$0 \leq \frac{X-Y}{M} < 2^j$. Hence, $j' = h(\frac{X-Y}{M}) < j$, or equivalently; $j' < j$.

Since the highest value of j is -1, then: $j' \leq -2$

Note that for the special case, $\frac{X-Y}{M} = 0$, then by definition $h(0) = 0$.

Consequently, if $X \geq Y$ then $j' \neq -1$.

For the case $X < Y$:

$|X - Y|_M = M - (Y - X)$, then $j' = h(\frac{M+X-Y}{M})$. or $j' = h(1 - \frac{Y-X}{M})$.

but since $X < Y$, then: $0 \leq \frac{Y-X}{M} < 2^j$, or $j' \geq h(1 - 2^j)$.

Since $X, Y \leq M$, then the maximum value of j is -1. Therefore, $0 > j' \geq h(\frac{1}{2})$. Or: $j' = -1$.

The proof of the algorithm as given by Realization II is the following:

- For the case $j > k$:

$$\frac{X}{M} = \sum_{i=-n}^{j} b_i 2^i \qquad (5)$$

where $b_{i's}$ are the binary bits of the variable, and $n = \lceil log_2 M \rceil$. Hence the estimated value X_e of X given by $X_e = 2^j M$ is guaranteed to be less or equal to the actual dividend (remainder) X. Similarly:

$$\frac{Y}{M} = \sum_{i=-n}^{k} b_i 2^i \qquad (6)$$

where the estimated value Y_e of Y given by $Y_e = 2^{k+1}M$ is guaranteed to be greater or equal to the actual divisor Y.
Thus, $\frac{X}{Y} \geq \lfloor\frac{X_e}{Y_e}\rfloor$. In other words, the remainder is guaranteed to be non-negative.

- For the case j=k, two possibilities are expected:
  1. $X < Y$. This case is detected according to Theorem 2 by evaluating j'. Hence, if $j' = -1$ then the quotient should not be incremented. Procedure is then stopped.
  2. $X \geq Y$. If $j' \neq -1$ then the quotient is incremented by 1. Procedure is then stopped.
- For the case $j < k$, it is obvious that $X < Y$. Procedure is stopped.

Therefore, (following the same approach given in the proof of Realization I):

$$Q = \lfloor\frac{X}{Y}\rfloor = \sum_{i=1}^{v} Q_i + Q_{v+1}$$

$$Q_{v+1} = \begin{cases} 1 & \text{, if } j' \neq -1 \\ 0 & \text{, otherwise} \end{cases}$$

6 Evaluation

The most recent work was introduced by Lu in 1992 [2]. It does NOT use MRC, however, it utilizes the idea of fractional representation of X/M to detect the parity of a residue number, and hence to check if an overflow has taken place. Lu's algorithm requires $2log_2\ Q$ steps, where Q is the quotient. Each step consists of several residue additions and subtractions, one residue multiplication, two memory access cycles and one multioperand addition (in fact, in parts II and IV of Lu's algorithm, more than one multioperand additions might be needed). Realization II of the new algorithm requires $log_2\ Q$ steps where each step consists of one residue multiplication (Q_iY) , one residue subtraction $(X-Q_iY)$ that is performed in parallel with one residue addition $(Q = Q+Q_i)$, one multioperand addition and two memory access cycles: one to get the fractional representations of residue digits, while the other is to obtain Q_i.
Following the literature prevalent method of measuring execution time for residue arithmetic algorithms, the mean of the basic residue arithmetic operations needed by each algorithm is computed, while memory accesses are not counted. The basic residue operations are: addition, subtraction and multiplication.
To compare performances of this algorithm and Lu's algorithm, computer programs simulating both algorithms have been developed to calculate the Mean Of Residue Operations (MORO). The Mean Of Multioperand Additions (MOMA) has been calculated and compared, where it applies. For simulation purposes, three moduli sets were selected to serve different dynamic ranges. These sets are: $M1 = (7,11,13,15)$, $M2 = (11,13,15,19,23,29,31)$, and $M3 = (29,31,43,47,53,55,59,61,63)$.

Table 1: Simulation Results

Moduli	MORO		MOMA	
Set	Our's	Lu's	Our's	Lu's
M1	2.68	10.34	0	0
M2	2.67	10.36	3.041	5.496
M3	2.72	10.39	3.089	5.504

For instance, moduli set M1 was selected to have a small dynamic range ($M < 2^{14}$). Moduli set M2 was selected to have a large dynamic range ($M < 2^{30}$). While dynamic range of M3 was chosen to be very large ($M < 2^{51}$). None of the moduli was chosen to be even, since that is not consistent with Lu's algorithm. However, no such condition is imposed by the new algorithm. The results are listed in Table 1. For moduli set M1, the results are exact. All possible combinations of dividends and divisors were simulated. However, for M2 and M3, a sample of 200 million randomly generated numbers within the dynamic range defined by each moduli set were simulated. The simulation of the new algorithm is based on Fig.(1) for M1, and on Fig.(2) for M2 and M3. Simulation of Lu's algorithm is based on the flowchart given in [2]. Since signed numbers had to be converted first to positive numbers, the simulation was conducted assuming unsigned integers.

For the moduli set M1, Table 1 indicates that the new algorithm is four times faster than Lu's algorithm. This conclusion applies to every moduli set where all the bits of residue digits can be applied simultaneously to a single RAM.
For other moduli sets like M2 and M3 which has very large dynamic ranges, the new algorithm is still four times faster regarding the number of basic residue operations. Moreover, the average number of multioperand additions needed is almost half that needed by the other algorithm.

7 Speed and Hardware Considerations

The implementation of either Realization I or II is relatively a simple operation. The speed of Realization II is determined by evaluating, the bottleneck function, $h(X)$. However, as densities and speed of RAMs increase, the use of larger look-up tables would be advantageous. Since, no implementation of a RNS divider has been published yet; the proposed ones (Fig.(1) and Fig. (2)), would allow for feasible, practical and beneficial RNS dividers. Improved memory density, will further, enhance the speed and expand the range of an implemented divider. For instance, a RAM of 1Mb can be used to implement the function $h(X)$ for moduli sets with dynamic ranges of 20 bits in the form of Realization I. This dynamic range would be very satisfactory for many applications. Moduli sets of 32 bits or less would require two 64k RAMs, along with a binary adder of two operands.

A moduli set with a dynamic range of 51 bits is very suitable for computer applications. The hardware requirements to build a RNS divider for such a range would mainly consist of: three memory tables each of 128k, one three-operand binary adder, a 4k RAM (or a table of 51 locations only if $j-k$ is used), a residue multiplier, subtractor and adder. However, if smaller memory sizes are to be used, then a larger multioperand adder should be considered.

8 Evaluating $h(I)$ for the moduli set $(2^k, 2^k-1, 2^{k-1}-1)$:

In this paper, we are customizing this fractional representation approach to serve a particular moduli set, namely, $(2^k, 2^k-1, 2^{k-1}-1)$. This customization results in reduced hardware requirements, and thus, decoding time.

8.1 Decoding Analysis:

Defining $m_1 = 2^k$, $m_2 = 2^k - 1$, $m_3 = 2^{k-1} - 1$. Therefore, $\hat{m}_1 = (2^k-1)(2^{k-1}-1)$, $\hat{m}_2 = 2^k(2^{k-1}-1)$, $\hat{m}_3 = 2^k(2^k-1)$. The residue representation of X is (r_1, r_2, r_3). Three new multiplicative inverses for $\hat{m}_1$, $\hat{m}_2$, and $\hat{m}_3$ are introduced here; these are: $2^{k-1}+1$, $2^k - 3$, and 2^{k-2} respectively. For a three moduli set, the CRT is given in the form [3]:

$$X = \sum_{i=1}^{3} \hat{m}_i \left| \frac{r_i}{\hat{m}_i} \right|_{m_i} - Mp \tag{7}$$

Substituting the corresponding values of $\hat{m}_i$ and their multiplicative inverses in (7), and dividing both sides by M, then:

$$\begin{aligned} \frac{X}{M} = \frac{1}{2^k} &\left|(2^{k-1}+1)r_1\right|_{2^k} + \\ &\frac{1}{2^k-1}\left|(2^k-3)r_2\right|_{2^k-1} \\ &+\frac{1}{2^{k-1}-1}\left|2^{k-2}r_3\right|_{2^{k-1}-1} - p \end{aligned} \tag{8}$$

Since $X < M$, then $\frac{X}{M} < 1$. Hence:

$$\begin{aligned} \frac{X}{M} = FRAC(\frac{1}{2^k} &\left|(2^{k-1}+1)r_1\right|_{2^k} + \\ &\frac{1}{2^k-1}\left|(2^k-3)r_2\right|_{2^k-1} \\ &+\frac{1}{2^{k-1}-1}\left|2^{k-2}r_3\right|_{2^{k-1}-1}) \end{aligned} \tag{9}$$

where FRAC($\cdots$) denotes the fractional part of the operand. In the following analysis, a significant property [3] will be used. It states that modulo $(2^p - 1)$ multiplication of an integer by 2^n, where p and n are positive integers, is equivalent to n-bits circular left-shift. (e.g. $\left|2^3(27)\right|_{31} \overset{binary}{\rightarrow} \overset{3-bits}{\overleftarrow{11011}} = 11110 \overset{decimal}{\rightarrow} 30$).

Therefore, to simplify the terms on the right hand Side (RHS) of (9), we proceed as follows:

- Evaluating $\frac{1}{2^k}\left|(2^{k-1}+1)r_1\right|_{2^k}$:
  Assuming that the binary form of r_1 is given by: $b_{1(k-1)}b_{1(k-2)}\cdots b_{11}b_{10}$, then,
  $\left|(2^{k-1}+1)r_1\right|_{2^k} = \left|2^{k-1}r_1 + r_1\right|_{2^k} \overset{binary}{\rightarrow}$
  $|b_{1(k-1)}b_{1(k-1)}\cdots b_{11}b_{10}\overbrace{00\cdots000}^{(k-1)zeros} +$
  $b_{1(k-1)}b_{1(k-1)}\cdots b_2 b_1 b_0|_{2^k}$
  Recalling that for *mod* 2^k, only the Least Significant (LS) k bits are significant, then $\left|(2^{k-1}+1)r_1\right|_{2^k} = b_x b_{1(k-2)}\cdots b_{11}b_{10}$, where $b_x = (b_{10} \text{ XOR } b_{1(k-1)})$.
  Now, let R_1 represents the binary form of $\frac{1}{2^k}\left|(2^{k-1}+1)r_1\right|_{2^k}$, then R_1 is obtained by multiplying the binary form of $\frac{1}{2^k}$ by that of $\left|(2^{k-1}+1)r_1\right|_{2^k}$. That is:
  $$R_1 = 0.b_x b_{1(k-2)}\cdots b_{11}b_{10} \tag{10}$$

- Evaluating $\frac{1}{2^k-1}\left|(2^k-3)r_2\right|_{2^k-1}$:
  Assuming that the binary form of r_2 is given by: $b_{2(k-1)}b_{2(k-2)}\cdots b_{21}b_{20}$, then, $\left|(2^k-3)r_2\right|_{2^k-1} = \left|2^k r_2 - 3r_2\right|_{2^k-1}$. But since $\left|2^k r_2\right|_{2^k-1} = |r_2|_{2^k-1}$, then, $\left|(2^k-3)r_2\right|_{2^k-1} = |-(2r_2)|_{2^k-1}$. Based on the circular left-shift property: $|-(2r_2)|_{2^k-1} \overset{binary}{\rightarrow}$ $\left|-(b_{2(k-2)}b_{2(k-3)}\cdots b_{21}b_{20}b_{2(k-1)})\right|_{2^k-1}$.

Or equivalently,

$$\left|-(b_{2(k-2)}b_{2(k-3)}\cdots b_{21}b_{20}b_{2(k-1)})\right|_{2^k-1} = \left|\overbrace{(11\cdots 111)}^{k-ones(=2^k-1)} -(b_{2(k-2)}b_{2(k-3)}\cdots b_{21}b_{20}b_{2(k-1)})\right|_{2^k-1}.$$

Assuming $r_2 \neq 0$, then:

$$\left|(2^k-3)r_2\right|_{2^k-1} \overset{binary}{\rightarrow} \bar{b}_{2(k-2)}\bar{b}_{2(k-3)}\cdots\bar{b}_{21}\bar{b}_{20}\bar{b}_{2(k-1)}$$

where $\bar{b}$ denotes the complement of the bit b.
However if $r_2 = 0$, then $\left|(2^k-3)r_2\right|_{2^k-1} = 0$.
On the other hand, the term $\frac{1}{2^k-1}$ can be written as $\frac{2^{-k}}{1-2^{-k}}$. Recalling that any fraction in the form $\frac{q}{1-q}$, where $|q| < 1$, can be expanded in a power series form as: $\frac{q}{1-q} = \sum_{i=1}^{i=\infty} q^i$. Therefore: $\frac{2^{-k}}{1-2^{-k}} = 2^{-k} + 2^{-2k} + 2^{-3k} + 2^{-4k} + \cdots\cdots$. Based on error analysis introduced in [4], then the MS $(3k+1)$ bits are the only significant bits in our computations. Let R_2 represents the binary form of $\frac{1}{2^k-1}\left|(2^k-3)r_2\right|_{2^k-1}$, then R_2 is obtained by considering the MS $(3k+1)$ bits of multiplying $\frac{1}{2^k-1}$ by $\left|(2^k-3)r_2\right|_{2^k-1}$. That is:

$$\begin{aligned} R_2 = 0.\overbrace{\bar{b}_{2(k-2)}\bar{b}_{2(k-3)}\cdots\bar{b}_{21}\bar{b}_{20}\bar{b}_{2(k-1)}}^{k\ bits} * \\ \overbrace{\bar{b}_{2(k-2)}\bar{b}_{2(k-3)}\cdots\bar{b}_{21}\bar{b}_{20}\bar{b}_{2(k-1)}}^{k\ bits} * \\ \overbrace{\bar{b}_{2(k-2)}\bar{b}_{2(k-3)}\cdots\bar{b}_{21}\bar{b}_{20}\bar{b}_{2(k-1)}}^{k\ bits}\bar{b}_{2(k-2)} \end{aligned} \quad (11)$$

where $*$ implies that terms are concatenated.

- Evaluating $\frac{1}{2^{k-1}-1}\left|2^{k-2}r_3\right|_{2^{k-1}-1}$:
Assuming that the binary form of r_3 is given in $(k-1)$ bits by: $b_{3(k-2)}b_{3(k-3)}\cdots b_{31}b_{30}$, then based on the circular left-shift property: $\left|2^{k-2}r_3\right|_{2^{k-1}-1} \overset{binary}{\rightarrow} b_{30}b_{3(k-2)}b_{3(k-3)}\cdots b_{31}$.
On the other hand, the term $\frac{1}{2^{k-1}-1}$ can be written as $\frac{2^{-(k-1)}}{1-2^{-(k-1)}}$. Thus, it can be expanded in a power series form as: $\frac{2^{-(k-1)}}{1-2^{-(k-1)}} = 2^{-(k-1)} + 2^{-2(k-1)} + 2^{-3(k-1)} + 2^{-4(k-1)} + \cdots\cdots$. Now, let R_3 be the binary form of $\frac{1}{2^{k-1}-1}\left|2^{k-2}r_3\right|_{2^{k-1}-1}$, then R_3 is obtained by considering the MS $(3k+1)$ bits of multiplying $\frac{1}{2^{k-1}-1}$ by $\left|2^{k-2}r_3\right|_{2^{k-1}-1}$. That is:

$$\begin{aligned} R_3 = 0.\overbrace{b_{30}b_{3(k-2)}\cdots b_{32}b_{31}}^{(k-1)\ bits}\overbrace{b_{30}b_{3(k-2)}\cdots b_{32}b_{31}}^{(k-1)\ bits} * \\ \overbrace{b_{30}b_{3(k-2)}\cdots b_{32}b_{31}}^{(k-1)\ bits}\overbrace{b_{30}b_{3(k-2)}b_{3(k-3)}b_{3(k-4)}}^{4\ bits} \end{aligned} \quad (12)$$

Therefore, (9) can be rewritten as:

$$\frac{X}{M} = FRAC(R_1 + R_2 + R_3) \quad (13)$$

Thus $h(\frac{X}{M})$ is nothing but the position of the MS nonzero bit of $\frac{X}{M}$.

Example: Consider the mouli set {16, 15, 7}. To find $h(\frac{X}{M})$ where $X = (8,12,4)$, (i.e. $X = 312$ and $M = 1680$), then:
$r_1 = 8 \overset{binary}{\rightarrow} 1000$, so $R_1 = 0.1000$
$r_2 = 12 \overset{binary}{\rightarrow} 1100$, so $R_2 = 0.0110\ 0110\ 0110\ 0$
$r_3 = 4 \overset{binary}{\rightarrow} 100$, so $R_3 = 0.010\ 010\ 010\ 010\ 0$
Therefore, $\frac{X}{M} = FRAC(R_1 + R_2 + R_3) = .00\underline{1}011111000$
Since the underlined MS nonzero bit is in the third location, then $h(\frac{X}{M})$= -3.

8.2 Hardware Implementation:

To implement (13), one three-operand binary adder is needed. A Carry-Save Adder (CSA) followed by a carry propagate adder (CPA) can realize the addition of the three operands. On the other hand, if $h(\frac{X}{M})$ is to be evaluated using the implementation proposed in [3]-[4] and [1], then two ROMs of size $(2^k \times 3k)$, another ROM of size $(2^{k-1} \times 3k)$ and one three-operand adder are needed. In our new implementation, the decoding time is also reduced by the memory access cycle time; which is very significant compared with the delay time of an adder.

9 Evaluating $h(I)$ for the moduli set $(2^k+1, 2^k, 2^k-1)$:

In this section, we are customizing the same RNS division approach to serve a particular moduli set, namely, $(2^k+1, 2^k, 2^k-1)$. This customization will reduce the complexity of evaluating $h(I)$.

9.1 Decoding Analysis:

The residue decoder introduced by Sweidan and Hiasat [7] has the advantages of reduced hardware requirements, and extremely wide fixed-point dynamic ranges since its upper bound is not limited by a memory size. Moreover, it requires a total of *only* four $2k$ bit binary adders, which makes it very attractive compared to other published decoders [8]-[9]. In this paper, we propose a hardware layout that can decode residue digits of the moduli set $(2^k+1, 2^k, 2^k-1)$ into binary equivalent. The new layout is an improvement of that presented in [7]. In this new contribution, we are reducing the number of adders needed for the decoding operation from four $2k$ bit binary adders into one $2k$ bit three-operand binary adder.

To have this paper self-contained, we briefly restate the analysis in [7]. Based on that, the contribution of our modified algorithm is introduced.

Using the basic definition of a residue system [3], and for any $X \epsilon [0, M)$:

$$X = \lfloor \frac{X}{2^k} \rfloor 2^k + r_2 \tag{14}$$

However, it has been proved in [7] that:

$$\lfloor \frac{X}{2^k} \rfloor = |A + B + C - r_1|_{2^{2k}-1} \tag{15}$$

where:
$A = \left|(2^{2k-1} + 2^{k-1})r_3\right|_{2^{2k}-1}$

$B = \left|(2^{2k} - 2^k - 1)r_2\right|_{2^{2k}-1}$

$C = \left|(2^{2k-1} + 2^{k-1})r_1\right|_{2^{2k}-1}$

That is, if r_1, r_2 and r_3 has the following binary format:

$r_1 = b_{1k}b_{1(k-1)} \cdots b_{11}b_{10}$

$r_2 = b_{2(k-1)}b_{2(k-2)} \cdots b_{21}b_{20}$

$r_3 = b_{3(k-1)}b_{3(k-2)} \cdots b_{31}b_{30}$

Then using circular left-shift property; A , B and C can be expressed as in [7]:

$A = b_{30}b_{3(k-1)} \cdots b_{32}b_{31}b_{30}b_{3(k-1)} \cdots b_{32}b_{31}$

$B = \bar{b}_{2(k-1)}\bar{b}_{2(k-2)} \cdots \bar{b}_{21}\bar{b}_{20}\underbrace{11\cdots1}_{k\ ones}$

$C = b_{1x}b_{1(k-1)} \cdots b_{12}b_{11}b_{1x}b_{1(k-1)} \cdots b_{12}b_{11}$

where: $b_{1x} = b_{10}$ OR b_{1k}.

By redefining $R'_1 = A$,
$R'_2 = B - r_1$
and
$R'_3 = C$, then (15) can be rewritten as:

$$\lfloor \frac{X}{2^k} \rfloor = |R'_1 + R'_2 + R'_3|_{2^{2k}-1} \tag{16}$$

- Case I: Since $R'_1 = A$, then the binary representation is the same:

$$R'_1 = b_{30}b_{3(k-1)}b_{3(k-2)} \cdots b_{32}b_{31} * \\ b_{30}b_{3(k-1)}b_{3(k-2)} \cdots b_{32}b_{31} \tag{17}$$

- Case II: Since $R'_2 = B - r_1$, then for the case $r_1 < 2^k$, and using the 2's Complement notation, $R'_2 = B + (1's\ Complement\ of\ r_1) + 1$. Noting that the LS k bits of B are all ones, then the LS k bits of the result of the subtraction is simply the 1's Complement of r_1 and an overflow of 1 at the $(k+1)^{th}$ bit. Based on 2's Complement, this overflow indicates that the result of subtraction is positive, hence it can be disregarded. However, when $|r_1|_{2^k} = r_2 = 0$, then $R'_2 = \left|2^{2k} - 1\right|_{2^{2k}-1} = 0$. Therefore, R'_2 can be expressed in binary format as:

$$R'_2 = \begin{cases} 0 & \text{, if } |r_1|_{2^k} = r_2 = 0 \\ \bar{b}_{2(k-1)}\bar{b}_{2(k-2)} \cdots \bar{b}_{21}\bar{b}_{20} * \bar{b}_{1(k-1)}\bar{b}_{1(k-2)} \cdots \bar{b}_{11}\bar{b}_{10} & \text{, otherwise} \end{cases} \tag{18}$$

- Case III: $r_1 = 2^k$. In this case, the $(k+1)^{th}$ bit of r_1 is 1, thus, the values R'_2 and B are the same because in the computation of R'_2 we used the LS $(k-1)$ bits of r_1, which are all zeros in this case. Therefore, the format of R'_2 is not changed. However, to take care of this nonzero $(k+1)^{th}$ bit of r_1, it has to be subtracted from R'_3. Therefore

$$R'_3 = \begin{cases} b_{1x}b_{1(k-1)} \cdots b_{12}b_{11}b_{1x}b_{1(k-1)} \cdots b_{12}b_{11} & \text{, if } r_1 \neq 2^k \\ \bar{b}_{1x}\bar{b}_{1(k-1)} \cdots \bar{b}_{12}\bar{b}_{11}b_{1x}b_{1(k-1)} \cdots b_{12}b_{11} & \text{, } r_1 = 2^k \end{cases} \tag{19}$$

9.2 Hardware Implementation

Realizing (16) is simply accomplished by adding R'_1, R'_2 and R'_3. The output should then be incremented by any overflow-carry [7]. Nevertheless, any overflow resulting from this adder can be neglected as long as the output does not have the value $(2^n - 1)$, where $0 \leq n \leq 2k$. This can be justified by the fact that $h(I) = h(I+1)$ if $I \neq (2^n - 1)$. However, if $I = 2^n - 1$, the overflow would be significant and the priority encoder proposed in implementing the residue divider can take care of this special case.
A single carry-save adder can add these three operands. Few logic gates are also needed to detect the $(k+1)^{th}$

bit of r_1 and to select the proper format of R'_3.

Moreover, if the same moduli set is to be decoded using the implementation proposed in [3]-[4] and [1], then two ROMs of size $(2^k \times 2k)$, a ROM of size $(2^{k+1} \times 2k)$, and one three-operand adder are needed.

Example: Consider the mouli set {17, 16, 15}. To find $h(X)$ where $X = (11, 11, 2)$, (i.e. $X = 827$), then:
$r_1 = 11 \overset{binary}{\rightarrow} 01011$, so $R'_3 = 1101\ 1101$
$r_2 = 11 \overset{binary}{\rightarrow} 1011$, so $R'_2 = 0100\ 0100$
$r_3 = 2 \overset{binary}{\rightarrow} 0010$, so $R'_1 = 0001\ 0001$

Therefore, Adder Output $= (R'_1 + R'_2 + R'_3) =$ 0011 0010, where the overflow has been neglected.
Based on (14), $h(X) = h(0011\ 0010\ 1011) = 10$ (i.e. 10^{th} position).

10 Hardware and Speed Considerations

Figure (3) shows the new proposed hardware realization of a RNS divider for the moduli sets $(2^k, 2^k - 1, 2^{k-1} - 1)$ and $(2^k + 1, 2^k, 2^k - 1)$. The operation of this divider is self-explanatory. The propagation delay in Fig.(3) , as compared with that in Fig.(2), has been reduced by a memory access cycle per iteration. Recalling that memory access cycle is very significant compared with the delay of other components, and that division is an iterative procedure, then this reduction will, eventually, be more and more significant as the number of iterations per division problem is increased. This implies that the new proposed realization is much faster for these particular moduli sets. Moreover, the reduction in hardware requirements is another substantial improvement.

11 Conclusions

This paper introduced a new general division algorithm customized to serve two moduli sets: $(2^k, 2^k - 1, 2^{k-1} - 1)$ and $(2^k + 1, 2^k, 2^k - 1)$. The proposed realization of the algorithm requires less hardware and processing time. A RNS divider would then require a binary adder, priority encoder, ROM, residue adder, residue subtractor, and residue multiplier only. The reduced hardware requirements and processing speed qualify the new realization to be very practical for many computing applications, and therefore, enable RNS to play an increased role in designing arithmetic logic units for general purpose computers.

References

[1] A. Hiasat, and H. Abdel-Aty-Zohdy, "High-Speed division algorithm for residue number system ," *in the Proceedings of 1995 IEEE International Symposium on Circuits and Systems, vol. 3, pp 1996-1999, May 1995.*

[2] M. Lu, and J. Chiang, "A novel division algorithm for residue number system,". *IEEE Trans. Compu., C-41, pp 1026-1032, Aug. 1992.*

[3] N. Szabo, and R. Tanaka, *"Residue Arithmetic and Its Applications to Computer Technology,"*. McGraw Hill, New York, 1967.

[4] M. Soderstrand, M. A., W. Jenkins, G. Jullien, F. Taylor, Eds. *"Residue Number System Arithmetic: Modern Applications in Digital Signal Processing,"*. IEEE Press, New York, 1986.

[5] M. Soderstrand, C. Vernia, and J. Chang, "An improved residue system digital-to-analog converter,". *IEEE Trans Circ. and Sys., CAS-30, pp 903-907, Dec. 1983.*

[6] T. Van Vu, "Efficient implementations of chinese remainder theorem for sign detection and residue decoding,". *IEEE Trans Compu., C-34, pp 646-651, July 1985.*

[7] A. Sweidan, and A. Hiasat, "New efficient memoryless, residue to binary converter,". *IEEE Trans. on Circuits and Systems, CAS-35, pp 1441-1444, Nov. 1988.*

[8] B. Bernardson, "Fast memoryless, over 64 bits, residue to decimal converter,". *IEEE Trans. on Circuits and Systems, CAS-32, pp 298-300, March 1985.*

[9] K. Ibrahim, and S. Saloum, "An efficient residue to binary converter design,". *IEEE Trans. on Circuits and Systems, CAS-35, pp 1156-1158, Sep. 1988.*

[10] A. Hiasat, and H. Abdel-Aty-Zohdy, "Design and implementation of a fast and compact residue-based semi-custom VLSI arithmetic chip,". *in the Proceedings of 1994 IEEE MidWest Symposium on Circuits and Systems, vol. 1, pp 428-431, August 1994.*

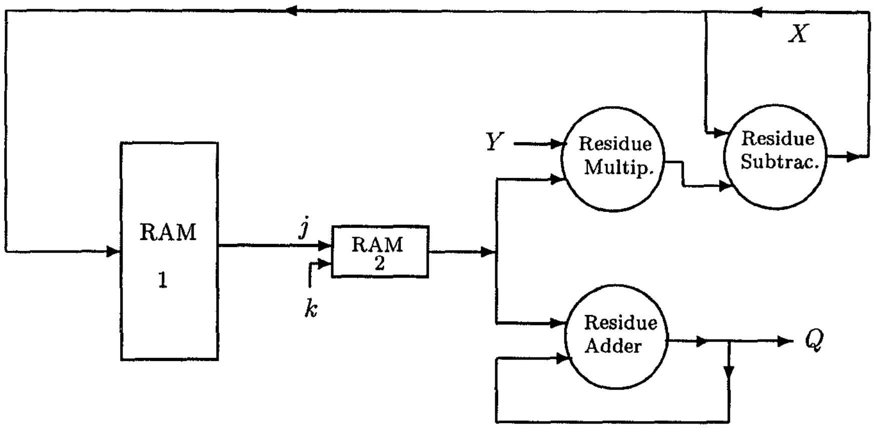

Fig. (1): Proposed Implementation of Realization I.

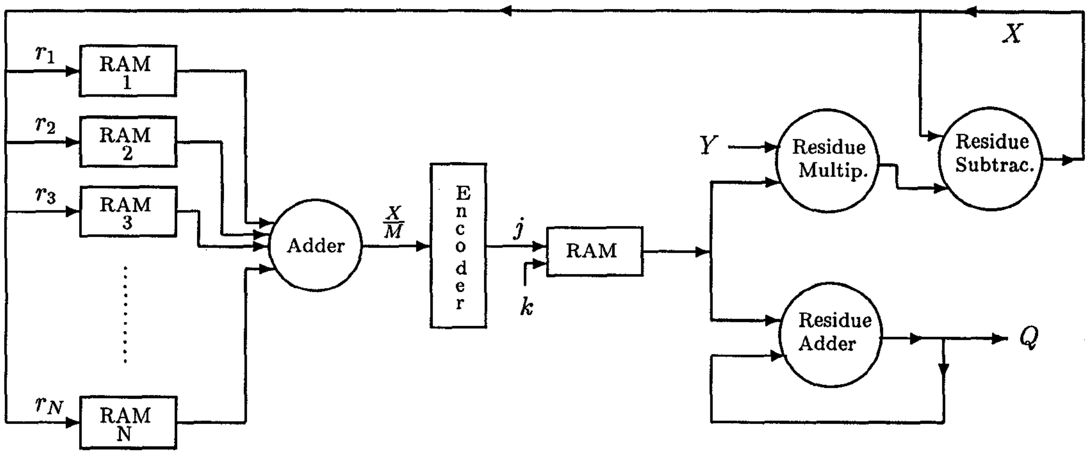

Fig. (2): Proposed Implementation of Realization II.

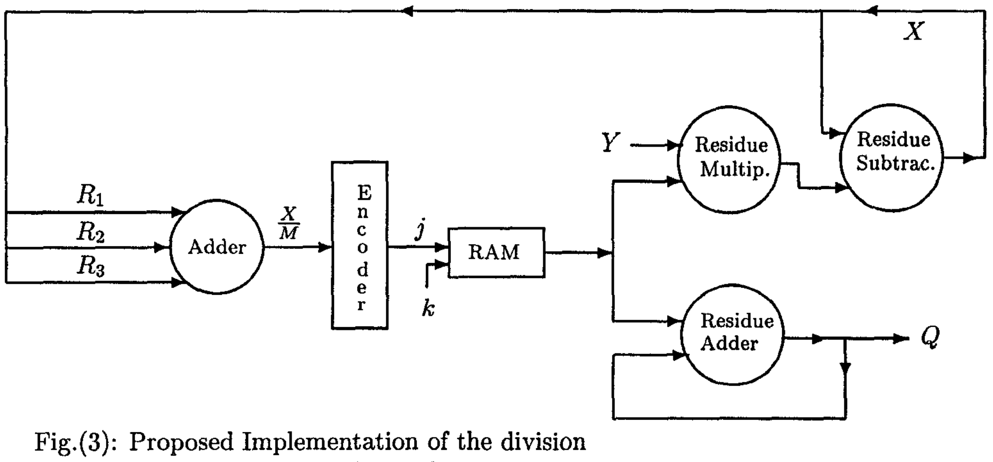

Fig.(3): Proposed Implementation of the division algorithm customized for moduli sets: $(2^k, 2^k - 1, 2^{k-1} - 1)$ and $(2^k + 1, 2^k, 2^k - 1)$.