$\theta(logN)$ ARCHITECTURES FOR RNS ARITHMETIC DECODING*

K. M. Elleithy, M. A. Bayoumi, and K. P. Lee

The Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504, U.S.A.

ABSTRACT

Decoding in Residue Number System (RNS) based architectures can be a bottleneck. A high speed and flexible modulo decoder is an essential computational element to maintain the advantages of RNS. In this paper, a fast and flexible modulo decoder, based on the Chinese Remainder Theorem (CRT), is presented. It decodes a set of residues into its equivalent representation in either unsigned magnitude or 2's complement binary number system. Two different architectures are analyzed; the first one is based on using Carry Save Adders(CSA), while, the other is based on utilizing a modified structure of Carry Save Adders(MCSA). Both architectures are modular and are based on simple cells which leads to efficient VLSI implementation. The proposed decoder is fast, it has a time complexity of $\theta(logN)$.

1. Introduction

Recently, RNS has received increased attention due to its ability to support high-speed concurrent arithmetic [1-3]. Applications such as fast fourier transform, digital filtering, and image processing utilize the efficiencies of RNS arithmetics in addition and multiplication, they do not require the difficult RNS operations such as division and magnitude comparison. RNS has been employed efficiently in the implementation of several special purpose processors such as digital signal processors[4].

Since special purpose processors are associated with general purpose computers, binary-to-residue and residue-to-binary conversions become inherently important and the conversion process should not offset the speed gain in RNS operations. While the binary-to-residue conversion does not pose a serious threat to the speed gain in RNS operations, the residue-to-binary conversion can be a bottleneck. It is mainly carried out employing the Chinese Remainder Theorem (CRT) [5,6]. Several implementations of the residue decoder have been reported [7-12]. In [12], the proposed residue decoders are basically based on biased addition, and take advantage of the fast addition speed of CSA[13]. But, the conversion output is not in 2's complement form. The implementation in [11] requires that one of the moduli must be a power of two; therefore, it may be limited in application. The residue decoders in [7,8] are based on using three moduli in the form $(2^n-1, 2^n, 2^n+1)$. Due to the limitation imposed on the number of moduli and the choice of them, it is limited in application. In [10], the residue decoder is based on the base extension technique, it uses modular look-up tables in its implementation. Since two moduli are fed into a look-up table, the choice of moduli must not be large for the implementation to be feasible. In addition, it does not support residue to 2's complement binary number system conversion. Although look-up tables are used in this scheme, its time complexity is $\theta(N^2)$. In [14,15], the scheme used has a time complexity of $\theta((logN)^2)$. In [9], a scheme of $\theta(logNP)$ (where P is the number of bits) is used to support only unsigned magnitude binary numbers.

*This work was supported in part by NSF grant No. MIP-8809811.

In this paper, a $\theta(logn)$ residue decoder capable of decoding a set of residues to its equivalent representation in unsigned magnitude or 2's complement binary number system is introduced. Two different architectures using CSAs based on[16] and MCSA[17] are implemented. In the following section, the RNS theory is reviewed. Section 3 discusses how this fast and flexible residue decoder can be implemented. Section 4 evaluates the speed performance of this residue decoder.

2. Residue Number System

In RNS, an integer , X, can be represented by N-tuple of residue digits,

$$X = (r_1, r_2, \ldots\ldots\ldots\ldots, r_N)$$

where $r_i = \left|X\right|_{m_i}$, with respect to a set of N moduli $\{m_1, m_2, \ldots\ldots\ldots\ldots, m_N\}$. In order to have a unique residue representation, the moduli must be pairwise relatively prime, that is,

$$GCD(m_i, m_j) = 1, \qquad \text{for } i \neq j$$

then it is shown that there is a unique representation for each number in the range of $0 \leq X < \prod_{i=1}^{N} m_i = M$ where N is the number of moduli.

The arithmetic operation on two integers A and B is equivalent to the arithmetic operation on its residue representation, that is,

$$\left|A \cdot B\right|_M = \left(\left|\left|A\right|_{m_1} \cdot \left|B\right|_{m_1}\right|_{m_1}, \left|\left|A\right|_{m_2} \cdot \left|B\right|_{m_2}\right|_{m_2}, \ldots, \left|\left|A\right|_{m_N} \cdot \left|B\right|_{m_N}\right|_{m_N} \right)$$

where '$\cdot$' can be addition, subtraction, or multiplication. Therefore, it is desired to convert binary arithmetic on large integers to residue arithmetic on small residue digits in which the operations can be parallelly executed, and there is no carry chain between residue digits.

For applications in digital signal processing, it is helpful to define a dynamic range for the RNS with positive and negative integers. The dynamic range is defined as $\left[-\frac{M-1}{2}, \frac{M-1}{2}\right]$ for M odd and as $\left[-\frac{M}{2}, \frac{M}{2}-1\right]$ for M even, or more specifically, for M odd,

$$X = \begin{cases} Z & \text{if } Z \leq \frac{M-1}{2} \\ Z-M & \text{if } Z > \frac{M-1}{2} \end{cases}$$

and for M even,

$$X = \begin{cases} Z & \text{if } Z < \frac{M}{2} \\ Z-M & \text{if } Z \geq \frac{M}{2} \end{cases}$$

where Z is an integer within the legitimate range, $0 \leq Z < M$. Any integer, X, within the dynamic range can be represented by N residue digits.

The conversion from RNS to weighted binary number system is done by using the CRT, which states that

$$\left|X\right|_M = \left|\sum_{j=1}^{N} \hat{m}_j \left|\frac{r_j}{m_j}\right|_{m_j}\right|_M \qquad (1)$$

where

$$M = \prod_{j=1}^{N} m_j, \quad \hat{m}_j = \frac{M}{m_j}$$

Although the CRT provides a direct, fast, and simple conversion formula, the lack of large and fast modulo M adder has held back this approach.

3. The Residue Decoder

The residue decoder based on the CRT can be implemented by a modulo M adder tree. The modulo M adders at each level are used to correct the partial sum so that it is within the legitimate range. Since modulo M adder is very slow, the possible implementation may pose an overhead to the overall speed performance of an RNS processor. In addition, the CRT only converts residues to its binary representation in the legitimate range but not in the dynamic range. Therefore, conversion to 2's complement binary number system requires a final correction.

In order to implement a high speed residue decoder that can perform conversion to both unsigned magnitude and 2's complement binary number system, the following solutions are proposed:

1) The number of modulo M adders or binary adders should be reduced to a minimum.
2) CSAs or MCSAs can be used wherever multi-operand addition is required due to its high addition speed.
3) Correction can be performed only at the last stage, and it supports conversion to both unsigned magnitude and 2's complement binary number system.

For ease of residue decoder design, it is partitioned into 4 stages as shown in Figure 1. The input to the residue decoder are the residues and a control line, C, which determines the output to be in unsigned magnitude or 2's complement number system.

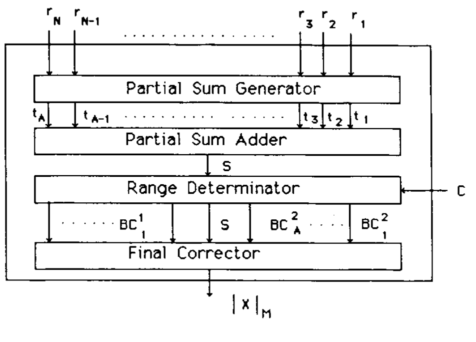

Figure 1. Block Diagram of the Residue Decoder

3.1. Partial Sum Generator

The inputs to this stage are the N residues. The main function of this stage is to compute partial sums, t_i's, where

$$t_i = \left|\hat{m}_i \left|\frac{r_i}{m_i}\right|_{m_i}\right|_M$$

Since m_i is usually small, the value of t_i can be obtained by accessing a lookup table with a small address space. Hence, r_i will serve as ROM address input, and t_i will be obtained from ROM output.

In most cases, it is better to reduce the number of partial sums, (t_i's), in order to reduce the complexity at lower stages and hence increase the residue decoder's speed as a whole. Since a modulus m_j can be represented by $\lceil \log_2 m_j \rceil$-bit binary number, the jth residue,

$$r_j = \sum_{k=0}^{\lceil \log_2 m_j \rceil - 1} 2^k b_k^j$$

where $b_k^j \in \{0, 1\}$. By substituting r_j in eq. (1), we can rewrite the CRT as follows:

$$\left|X\right|_M = \left|\sum_{j=1}^{N}\left|\sum_{k=0}^{\lceil \log_2 m_j \rceil - 1} \hat{m}_j \left|\frac{1}{\hat{m}_j}\right|_{m_j} 2^k b_k^j\right|_M\right|_M \qquad (2)$$

Hence, if we have a set of 8 moduli {2,3,5,7,11,13,17,23} with residues $\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8\}$, respectively, only 4 ROMs with 7-bit address input are needed to implement this level, and modulus summation of 4 operands instead of 8 is needed, where

$$t_1 = \left|\frac{M}{2}\left|\frac{r_1}{2}\right|_2 + \frac{M}{3}\left|\frac{r_2}{3}\right|_3 + \frac{M}{5}\left|\frac{r_3}{5}\right|_5\right|_M$$

$$t_2 = \left|\frac{M}{7}\left|\frac{r_4}{7}\right|_7 + \frac{M}{11}\left|\frac{r_5}{11}\right|_{11}\right|_M$$

$$t_3 = \left|\frac{M}{13}\left|\frac{r_6}{13}\right|_{13} + \sum_{k=0}^{2}\frac{M}{17}\left|\frac{1}{17}\right|_{17} 2^k b_k^7\right|_M$$

$$t_4 = \left|\sum_{k=3}^{4}\frac{M}{17}\left|\frac{1}{17}\right|_{17} 2^k b_k^7 + \frac{M}{23}\left|\frac{r_8}{23}\right|_{23}\right|_M$$

3.2. Partial Sum Adder

By far the modulo M summation of partial sums, (t_i's,) poses the biggest challenge to the implementation of the residue decoder due to the slow computational speed of the modulo M adder. This stage can be implemented using two different approaches.

§3.2.1. Implementation using CSA

A multilevel CSA tree consists of N-2 CSAs and a carry propagate adder, CPA[13], are used to reduce A partial sums, t's, to a sum, S. Let l be the number of levels on a CSA tree, and $\theta(l)$ be the maximum number of operands that can be processed with a l-level CSA tree. We can compute θ by the recursive formula provided by Avizienis[18],

$$\theta(l) = \left\lfloor \frac{\theta(l-1)}{2} \right\rfloor * 3 + (\theta(l-1)) \bmod 2$$

$$\text{for } l = 2, 3, \ldots\ldots, \text{ and } \textit{initially } \theta(1) = 3 \qquad (3)$$

A CSA tree for adding 6 operands is shown in Figure 2. CPA is a (m-1)-bit two-level carry lookhead adder, CLA[13] where:

$$m = \lceil \log_2(MA) \rceil$$

Hence, the output S is an m-bit number that is passed to the next stage. The complexity of the scheme is determined by Theorem 1.

Theorem 1: *The addition of N numbers using CSAs can be performed in $\theta(\log N)$ steps.*

Proof: The number of levels in a CSA tree is determined by:

$$\theta(l) = \left\lfloor \frac{\theta(l-1)}{2} \right\rfloor * 3 + \theta(l-1) \bmod 2$$

To determine the number of levels required to add N numbers let us consider the following two cases:

(i) $\theta(l-1)$ is even , then:

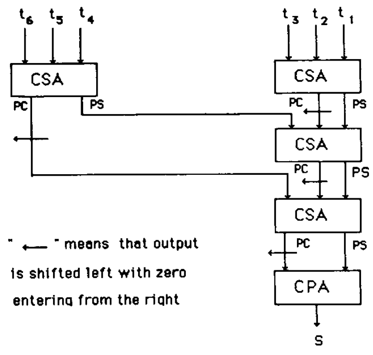
t6 t5 t4
t3 t2 t1
CSA
PC
PS
CSA
CSA
CSA
CPA
S
" ⟵ " means that output is shifted left with zero entering from the right

Figure 2. An Example for Partial Sum Adder for A = 6.

$$\left\lfloor \frac{\theta(l-1)}{2} \right\rfloor = \frac{1}{2}\theta(l-1) \tag{4}$$

$$\theta(l-1)\ mod2 = 0 \tag{5}$$

Substituting in (3) using (4) and (5), we have:

$$\theta(l) = \frac{3}{2}\theta(l-1) \tag{6}$$

Since $\theta(1) = 3$, we can substitute in (6) to get successive values for $\theta(l)$ as follows:

$$\theta(2) = \frac{3}{2}*3$$
$$\theta(3) = (\frac{3}{2})^2*3$$
$$\theta(4) = (\frac{3}{2})^3*3$$
$$\theta(5) = (\frac{3}{2})^4*3$$

.

.

$$\theta(l) = (\frac{3}{2})^{l-1}*3$$
$$= (\frac{3}{2})^{l}*2$$

$\theta(l)$ represents the number of operands that can be added using a CSA tree that has l levels. Suppose that the number of operands is N then:

$$N = (\frac{3}{2})^{l}*2$$

Taking the logarithm of both sides we have:

$$\log N = l*\log\frac{3}{2}$$

Then: $l = \frac{1}{\log\frac{3}{2}}*\log N$

We can find constants $C_1 > 0$, $C_2 > 0$, and $N_0 \geq 0$, such that for all $N \geq N_0$ the following is true:

$$C_1 \log N \leq \frac{1}{\log\frac{3}{2}}*\log N \leq C_2 \log N \tag{7}$$

Then

$$C_1 \log N \leq l \leq C_2 \log N \quad \forall N \geq N_0 \tag{8}$$

Possible values for C_1 , C_2 and N_0 are 1,2,1. Equation (8) means that $l = \theta(logN)$.

(ii) $\theta(l-1)$ is odd , then:

$$3 * \left\lfloor \frac{\theta(l-1)}{2} \right\rfloor = \frac{3}{2}\theta(l-1) - 1.5 \tag{9}$$

$$\theta(l-1)\ mod2 = 1 \tag{10}$$

Substituting in (3) using (9) and (10), we get:

$$\theta(l) = \frac{3}{2}\theta(l-1) - \frac{1}{2} \tag{11}$$

Since $\theta(1) = 3$, we can substitute in (11) to get successive values for $\theta(l)$ as follow:

$$\theta(2) = \frac{3}{2}*3-\frac{1}{2}$$
$$\theta(3) = (\frac{3}{2})^2*3-(\frac{3}{2})^1*3-\frac{1}{2}$$
$$\theta(4) = (\frac{3}{2})^3*3-(\frac{3}{2})^2*3-(\frac{3}{2})^1*3-\frac{1}{2}$$

.

.

$$\theta(l) = (\frac{3}{2})^{l-1}*3-((\frac{3}{2})^{l-2}+(\frac{3}{2})^{l-3}+...+1)*0.5$$
$$= (\frac{3}{2})^{l}*2-\frac{(\frac{3}{2})^{l-1}-1}{\frac{3}{2}-1}*0.5$$
$$= \frac{4}{3}(\frac{3}{2})^{l}+1$$

Suppose that the number of operands is N then:

$$N = \frac{4}{3}(\frac{3}{2})^{l}+1$$

Using the same analytical method used for the case of even $\theta(l-1)$ we can find constants C_1, C_2, and $N_0 \geq 0$, such that for all $N \geq N_0$ the following is true:

$$C_1 \log N \leq \frac{1}{\log\frac{3}{2}}*\log N \leq C_2 \log N$$

From the previous analysis in both cases *i* and *ii*, N numbers can be added using CSAs in $\theta(logN)$. □

§3.2.2. Implementation using MCSA

The MCSA is based on the idea of representing a number as a *Carry* and a *Sum* similar to CSA. It can be used in the modulo addition of two numbers to obtain a scheme that has a constant speed which does not depend on the number of bits. Basically CSA depends on the idea of not completing the addition process at a certain stage, but postpone it to the final stage. In the intermediate stages numbers are represented as *Sum* and *Carry* to avoid the complete addition process. The MCSA is used to add two numbers A and B in modulo m. Figure 3.*a* shows that A is represented as a pair of numbers $(A_S\ , A_C)$, B is represented as $(B_S\ , B_C)$, and the output C is represented as $(C_S\ , C_C)$. Each number is represented as a group of *Sum* bits and *Carry* bits. There is no unique representation for A_S and A_C. The condition that need to be satisfied is:

$$\left| A_S + A_C \right|_m = \left| A \right|_m$$

One possible representation is:

$$A_S = \left| A \right|_m \qquad A_C = 0$$

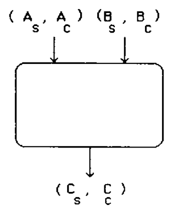

Figure 3.a. A Modified CSA (MCSA).

We need to add four numbers (A_S , A_C ,B_S ,B_C), which needs two steps of CSA. After the addition process we need to detect if $-M$ or $2*(-M)$ is required to adjust the result. The adjusting process takes at most three steps. The proposed algorithm for modulo m addition of two numbers can be described as follow:

Algorithm modulo_add (A , B , Result)

Input: Two variables A and B in modulo m, A is represented as A_S and A_C. B is represented as B_S and B_C. All variables are n bit numbers.

Output: Variable *Result* represented as $Result_C$ and $Result_S$. The relation between A, B, and *Result* is: $Result = \left| A + B \right|_m$.

Procedure:

```
begin
  Do in parallel
    begin
      Call Sum($temp_1$ , $A_S$ , $A_C$ , $B_S$)
      Call Carry($temp_2$ , $A_S$ , $A_C$ , $B_S$)
    end
  Do in parallel
    begin
      Call Carry($temp_3$ , $temp_1$ , $temp_2$ , $B_C$)
      Call Carry($temp_4$ , $temp_1$ , $temp_2$ , $B_C$)
    end
  Case ( temp sub 2 [n+1] + temp sub 4 [n+1] ) of
    0: Do in parallel
         begin
           $Result_S := temp_3$
           $Result_C := temp_4$
         end
         exit
     1: Do in parallel
         begin
           Call Sum($temp_5$ , $temp_3$ , $temp_4$ , $(2^n-m)$)
           Call Carry($temp_6$ , $temp_3$ , $temp_4$ , $(2^n-m)$)
         end
    2: Do in parallel
         begin
           Call Sum($temp_5$ , $temp_3$ , $temp_4$ , $2*(2^n-m)$)
           Call Carry($temp_6$ , $temp_3$ , $temp_4$ , $2*(2^n-m)$)
         end
  end case
  Case ( temp sub 6 [n+1] ) of
    0: Do in parallel
         begin
           $Result_S := temp_5$
           $Result_C := temp_6$
         end
         exit
     1: Do in parallel
         begin
           Call Sum($temp_7$ , $temp_5$ , $temp_6$ , $(2^n-m)$)
           Call Carry($temp_8$ , $temp_5$ , $temp_6$ , $(2^n-m)$)
         end
  end case
  Case ( temp sub 8 [n+1] ) of
    0: Do in parallel
         begin
           $Result_S := temp_7$
           $Result_C := temp_8$
         end
     1: Do in parallel
         begin
           Call Sum($temp_9$ , $temp_7$ , $temp_8$ , $(2^n-M)$)
           Call Carry($temp_{10}$ , $temp_7$ , $temp_8$ , $(2^n-M)$)
         end
       Do in parallel
         begin
           $Result_S := temp_9$
           $Result_C := temp_{10}$
         end
  end case
end.
```

```
Sum ($A$ , $B$ , $C$ , $D$)
begin
  Do in parallel ($1 \le i \le n$)
    $A[i] := (B[i] \wedge C[i]) \vee (B[i] \wedge D[i]) \vee (C[i] \wedge D[i])$
end
```

```
Carry ($A$ , $B$ , $C$ , $D$)
begin
  $A[1] := 0$
  Do in parallel ($1 \le i \le n$)
    $A[i+1] := B[i] \oplus C[i] \oplus D[i]$
end
```

An implementation of the algorithm is shown in Figure 3.*b*.

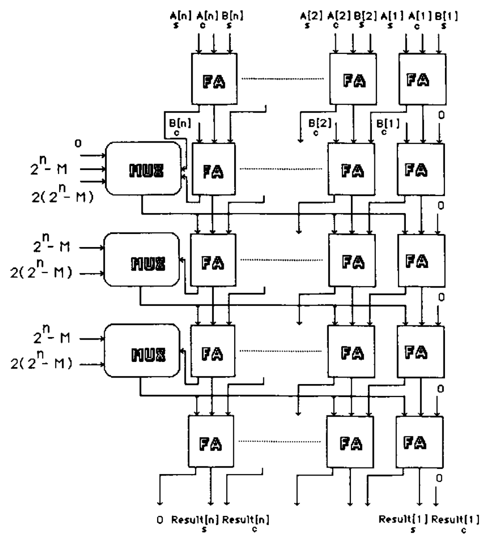

Figure 3.b. Different Stages of the MCSA.

Theorem 2: *The modulo adder scheme for adding two n-bit numbers in modulo m has an asymptotic time complexity $\theta(1)$.*

Proof: To prove that the number of steps is constant (five) we need to prove that the last carry is equal to zero in five or less steps. Induction is used to prove the correctness of the theorem on the number of bits n.

[1] Basis step: for $n = 0$, means that we do not add any numbers and in this case the required number of steps is zero.

[2] Induction hypothesis: assume for a fixed arbitrary $n \geq 0$ that that the maximum number of steps is five.

[3] Induction step: for numbers with $n+1$ bits let:
$\eta = temp_2[n+1] + temp_4[n+2]$.
Then we have the following cases:
(a) $\eta=0$: then the carry propagation stopped at bit n, and it ends after five steps at most according to the induction hypothesis.
(b) $\eta=1$: then the correction is $2^{n+1}-m$ in step 3. Since $m > 2^n$ then $2^{n+1}-m < 2^n$, which means that $(2^{n+1}-m)[n] = 0$. The worst case we get to have $temp_3[n+1]$ and $temp_4[n+2]$ to be

equal one. This means that $temp_5[n+1]=0$ and $temp_6[n+2]=1$, then $temp_8[n+2]=0$. In this case the correction is done in two steps (step3 and step 4).

(c) η=2: then the correction is $2*(2^{n+1}-m)$ in step 3. The worst case we get to have $temp_3[n+1]$, $temp_4[n+2]$, and $2*(2^{n+1}-m)$ to be equal one. Then $temp_5[n+1]=1$, $temp_6[n+1]=1$ and $2^{n+1}-M=0$. At step4 $temp_7[n+1]=0$ and $temp_8[n+2]=1$. At step5 $temp_9[n+1]=1$ and $temp_{10}[n+2]=0$. In this case the correction is done in three steps (steps3—5). □

Since the adder has a fixed number of stages which does not depend on the operands' length, it can be used in the implementation of a pipelined multi-operand modulo addition scheme[19].

Example: As an example, the modulo addition of $A = 1272$ and $B = 450$ for $m = 2050$ is shown in Figure 3.c. There is no unique representation for A and B. One valid representation is shown in Figure 3.c. Figure3.c shows the detailed modulo addition operation for this example. In step1 we get $temp_2[13] = 1$, and in step2 we get $temp_4[13] = 1$, which means that at step3 we have to add $2(2^n-M)$. At step3 we get $temp_6[13] = 1$, which means that at step4 we have to add 2^n-M. At step4 we get $temp_8[13] = 0$, which means that the addition process stops at step4. The result of step4 is the final result. □

Initial: A_s= 101111110111
A_c= 110011101101
B_s= 111100010101
B_c= 101010110011
M = 2050 , N = 12

Step 1. A_s= 101111110111
A_c= 110011101101
B_s= 111100010101
--------------
$temp_1$= 100000001111
$temp_2$=$\boxed{1}$111111101010

Step 2. $temp_1$= 100000001111
$temp_2$= 111111101010
B_s= 111100010101
--------------
$temp_3$= 110101010110
$temp_4$=$\boxed{1}$010101010110

Step 3. $temp_3$= 110101010110
$temp_4$= 010101010110
$2(2^n - M)$ = 111111111100
--------------
$temp_5$= 011111111100
$temp_6$=$\boxed{1}$101010101100

Step 4. $temp_5$= 011111111100
$temp_6$= 101010101100
$2^n - M$ = 011111111110
--------------
$temp_7$= 101010101110
$temp_8$=$\boxed{0}$111111111000

$Result_s$= 101010101110
$Result_c$= 111111111000

Figure 3.c. A detailed Example for the Modulo Addition.

Theorem 3: ***Adding n numbers $(y_1, y_2, \cdots, y_n)$ in modulo M is equivalent to :***

[1] Adding (y_1, y_2) modulo M ,....,(y_i, y_{i+1}), . . . , and (y_{n-1}, y_n) gives $y_{12}, \ldots, y_{(n-1)n}$.

[2] Step [1] is repeated on (y_{12}, y_{34}) ,..., $(y_{(n-3)(n-2)}, y_{(n-1)n})$.

[3] Step [2] is repeated for $\lceil \log N \rceil - 2$ times to obtain one final output represented as a sum and carry.

Proof: To add two numbers a and b in modulo M we have the following cases:

(i) $a < M$ and $b < M$ then $a = \left| a \right|_M$ and $b = \left| b \right|_M$, then:
$$\left| a+b \right|_M = \left| a_M + b_M \right|_M \tag{12}$$

(ii) $a > M$ and $b < M$ then $b = \left| b \right|_M$ and $a = M + x$, then:
$$\left| a+b \right|_M = \left| M+x+b \right|_M = \left| x+b \right|_M \tag{13}$$
Since $x < M$ and $b < M$, then from (12) and (13):
$$\left| a + b \right|_M = \left| a_M + b_M \right|_M$$

(iii) $a > M$ and $b < M$ like case (ii).

(iv) $a > M$ and $b > M$ then $a = M+x$ and $b = M+y$, then:
$$\left| a+b \right|_M = \left| M+x+M+y \right|_M = \left| x+y \right|_M = \left| a_M + b_M \right|_M$$

From the previous four cases:
$$\left| a+b \right|_M = \left| a_M + b_M \right|_M \tag{14}$$

Since addition is associative then:
$$\left| y_1 + y_2 + \ldots + y_n \right|_M = \left| (y_1 + \ldots + y_{\frac{n}{2}}) + (y_{\frac{n}{2}+1} + \ldots y_n) \right|_M$$
$$= \left| \left| y_1 + \ldots + y_{\frac{n}{2}} \right|_M + \left| y_{\frac{n}{2}}+1 + \ldots + y_n \right|_M \right|_M \quad \text{(using 14).}$$

We can further expand this expression using the same method to get the addition process in the right hand side in terms of only two operands added in modulo M. □

Theorem 3 means that adding n numbers in modulo M can be performed using a binary tree consists of units that are capable of adding only two numbers in modulo M. MCSAs are used as those building blocks to perform the addition process. Since MCSA requires that inputs be represented in the form of sum and carry, then this form should be enforced at all levels. The form will be enforced automatically for levels ≥ 2, because the outputs of the previous levels are in the correct form. For first level we have the following:

$$Y_{iS} = y_i \;,\; Y_{iC} = 0 \quad \forall\, 1 \leq i \leq n$$

For the last stage the output is in the form of sum and carry which is exactly the same form as CSAs. Figure 3.d. shows the binary tree required to add n numbers in modulo M.

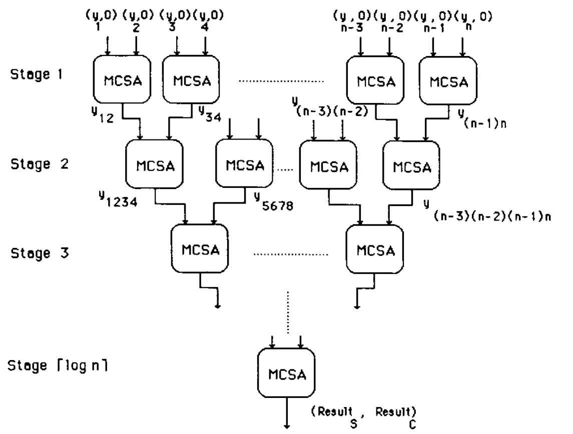

Figure 3.d. Partial Sum Adder Implementation Using MCSAs.

3.3. Range Determinator

This stage consists of three levels namely ROM, Magnitude Comparator(MC), and Bit Corrector(BC). The major function of this stage is to determine S range so that appropriate value can be subtracted from S to obtain the desired result. In order to accomplish this, 2 sets of values as shown in Table *I* have to be compared. For simplicity, we explain the first set then the second set.

Since the input to this stage, S, is a large binary number, it is partitioned into groups of adjacent bits. For example, if S is a 24-bit number, we can partition S into 3 8-bit groups G_1, G_2, and G_3, where

$$G_1 = S_{7..0},\ G_2 = S_{15..8},\ \text{and}\ G_3 = S_{23..16}$$

Since each group is fed into a ROM module as an address input, the number of bits in each group should be small so that small ROMs that are fast and occupy small silicon area are used to implement this level. However, the number of groups, g, should be kept small as possible since the complexity of MC cells is a function of the number of ROM modules, g. Hence, there are tradeoffs in choosing g and the number of bits in each group.

As shown in Figure 4, the input to ith ROM module of the the first set of ROMs is G_i, and the outputs are B_j^i's and C_k^i's. The function of this ROM module is depicted as follows:

$$B_j^i = \begin{cases} 0 & \text{if } G_i \le \left(jM-1\right)_i \\ 1 & \text{if } G_i > \left(jM-1\right)_i \end{cases} \qquad \text{for } j = 1..A-1$$

<table>
<tr><th rowspan="3">A</th><th colspan="3">Magnitude Compared</th></tr>
<tr><th rowspan="2">First Set</th><th colspan="2">Second Set</th></tr>
<tr><th>if M Odd</th><th>if M Even</th></tr>
<tr><td>1</td><td>$M-1$</td><td>$\frac{M-1}{2}$</td><td>$\frac{M}{2}-1$</td></tr>
<tr><td>2</td><td>$2M-1$</td><td>$\frac{3M-1}{2}$</td><td>$\frac{3M}{2}-1$</td></tr>
<tr><td></td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>$n-1$</td><td>$(n-1)M-1$</td><td>$\frac{(2n-3)M-1}{2}$</td><td>$\frac{(2n-3)M}{2}-1$</td></tr>
</table>

Table I Values Compared by Multi-magnitude Comparators

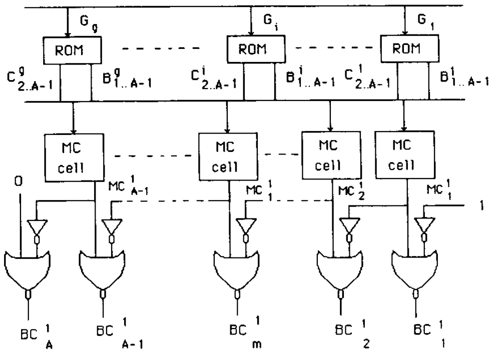

Figure 4. Implementation of the first part of Range Determinator Stage

$$C_k^i = \begin{cases} 0 & \text{if } G_i \ne \left(kM-1\right)_i \\ 1 & \text{if } G_i = \left(kM-1\right)_i \end{cases} \qquad \text{for } k = 2..A-1$$

$$\text{for } i = 1, 2, \ldots\ldots, g$$

Clearly, these ROM modules serve as a partial multi-magnitude comparator that compares the input pattern S to the first set of values as shown in Table *I* and produce g*(2A-3) outputs that are to be fed into the MC level.

The MC level consists of (A-1) MC cells. This level takes the input from ROM level and does further comparison so that a 2-level multi-magnitude comparator is formed. The complexity of a MC cell is a function of the number of ROM modules. If we have g ROM modules, then the Boolean equation for the lth MC cell is as follows:

$$\begin{aligned} MC_l^1 = {} & B_l^g + B_l^{g-1}C_l^g + B_l^{g-2}C_l^gC_l^{g-1} + \ldots\ldots \\ & + B_l^2C_l^gC_l^{g-1}C_l^{g-2}\ldots\ldots C_l^3 \\ & + B_l^1C_l^gC_l^{g-1}\ldots\ldots C_l^3C_l^2 \end{aligned} \tag{15}$$

Hence we have,

$$MC_l^1 = \begin{cases} 0 & \text{if } S < lM \\ 1 & \text{if } S \ge lM \end{cases} \qquad \text{for } l = 1, 2, \ldots\ldots, A-1 \tag{16}$$

Since S may be larger than several values compared, the outputs of several MC cells may be set to 1; therefore, the BC level is used to ensure that only one of the outputs of the MC^1 cells is set to one and also to indicate the appropriate range. In order to do so, A identical BC^1 cells are needed, and their common Boolean equation is as follows:

$$BC_m^1 = \overline{MC_m^1 + \overline{MC_{m-1}^1}} \qquad \text{for } m = 1, 2, \ldots\ldots, A$$

where,

$$MC_0^1 = 1 \text{ and } MC_A^1 = 0 \tag{17}$$

Hence, the range of S is determined to be $(m-1)M \le S < mM$ if $BC_m^1 = 1$. Figure 4 shows the implementation of the multi-magnitude comparator that compares S with the first set of values shown in Table *I* and its BC level.

Thus far, the range determination enables the S modulus M operation to be performed by $S-(m-1)M$ if BC_m^1 is set to one. Therefore, only residue to unsigned magnitude number system conversion is possible. However, for residue to 2's complement number system conversion, the second set of values, as shown in Table *I*, has to be compared with S by another multi-magnitude comparator which is done in the same way as previously explained. Figure 5 shows the input to the ith ROM module of the second set of ROMs is G_i, and the outputs are D_j^i's and E_k^i's. The function of this ROM module is clearly depicted as follows:
For M odd,

$$D_j^i = \begin{cases} 0 & \text{if } G_i \le \left(\frac{(2j-1)M-1}{2}\right)_i \\ 1 & \text{if } G_i > \left(\frac{(2j-1)M-1}{2}\right)_i \end{cases} \qquad \text{for } j = 1..A-1 \tag{18}$$

$$E_k^i = \begin{cases} 0 & \text{if } G_i \ne \left(\frac{(2k-1)M-1}{2}\right)_i \\ 1 & \text{if } G_i = \left(\frac{(2k-1)M-1}{2}\right)_i \end{cases} \qquad \text{for } k = 2..A-1 \tag{19}$$

$$\text{for } i = 1, 2, \ldots\ldots, g$$

and for M even,

$$D_j^i = \begin{cases} 0 & \text{if } G_i \leq \left\lfloor \frac{(2j-1)M}{2} - 1 \right\rfloor_i \\ 1 & \text{if } G_i > \left\lfloor \frac{(2j-1)M}{2} - 1 \right\rfloor_i \end{cases} \quad \text{for } j = 1..A-1 \quad (20)$$

$$E_k^i = \begin{cases} 0 & \text{if } G_i \neq \left\lfloor \frac{(2k-1)M}{2} - 1 \right\rfloor_i \\ 1 & \text{if } G_i = \left\lfloor \frac{(2k-1)M}{2} - 1 \right\rfloor_i \end{cases} \quad \text{for } k = 2..A-1 \quad (21)$$

$$\text{for } i = 1, 2, \ldots\ldots, g$$

The MC level of this part is exactly the same as previously proposed, that is, it consists of A MC^2 cells, and each MC^2 cell has the same Boolean equation as follows:

$$MC_l^2 = D_l^g + D_l^{g-1}E_l^g + D_l^{g-2}E_l^gE_l^{g-1} + \ldots\ldots$$
$$+ D_l^2E_l^gE_l^{g-1}E_l^{g-2}\ldots\ldots E_l^3$$
$$+ D_l^1E_l^gE_l^{g-1}\ldots\ldots E_l^3E_l^2$$

Since different set of values is compared with S, we have for M odd,

$$MC_l^2 = \begin{cases} 0 & \text{if } S \leq \frac{(2l-1)M-1}{2} \\ 1 & \text{if } S > \frac{(2l-1)M-1}{2} \end{cases}$$

and for M even,

$$MC_l^2 = \begin{cases} 0 & \text{if } S < \frac{(2l-1)M}{2} - 1 \\ 1 & \text{if } S \geq \frac{(2l-1)M}{2} - 1 \end{cases}$$

$$\text{for} = 1, 2, \ldots\ldots\ldots, A-1$$

The BC level for this part of the design consists of A BC^2 cells. Each of these cells has a control line C. If C is equal to zero, then all the output lines of BC level will be equal to one and residue to unsigned magnitude number system conversion will be performed; otherwise, only one of the BC level output lines will be equal to one, and thus residue to 2's complement number system conversion will be performed. The Boolean equation for a BC cell is as follows:

$$BC_m^2 = \overline{C \cdot \overline{\overline{MC_m^2} \cdot MC_{m-1}^2}} \quad \text{for } m = 1, 2, \ldots\ldots\ldots, A$$

where,

$$MC_0^2 = 1 \text{ and } MC_A^2 = 0$$

Therefore, the range of S is determined to be $\frac{(2m-3)M+1}{2} \leq S < \frac{(2m-1)M+1}{2}$ for M odd, and $\frac{(2m-3)M}{2} \leq S < \frac{(2m-1)M}{2}$ for M even if $BC_m^2 = 1$ (Note that the lower bound is equal to zero when m=1). Figure 5 shows the implementation of this part of the design.

3.4. Final Corrector

This stage consists of A tristate multiplexers and a carry look-head adder. The BC^1 input lines will be used to enable one of the tristate multiplexers while BC^2 input lines will be used as the selectors of the multiplexers. If BC_i^1 is set, then $(i-1)M \leq S < iM$. The lower bound $(i-1)M$ will be subtracted from S if conversion to unsigned magnitude number system is desired, or S is less than $\frac{(2i-1)M+1}{2}$ for M odd or $\frac{(2i-1)M}{2}$ for M even; otherwise, the upper bound, iM, will be subtracted from S. The implementation of this stage is shown in Figure 6. The CLA is used to add the 2's complement of the value to be subtracted to S and output the desired result.

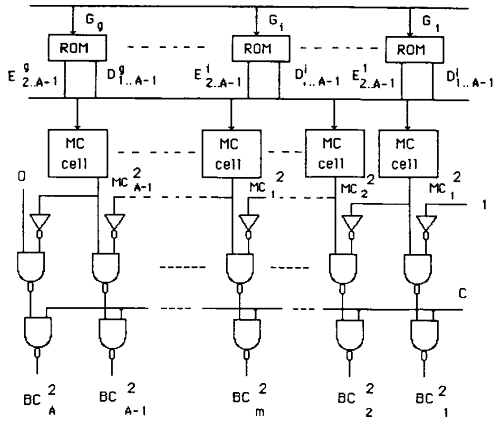

Figure 5. Implementation of the Second part of Range Determinator Stage

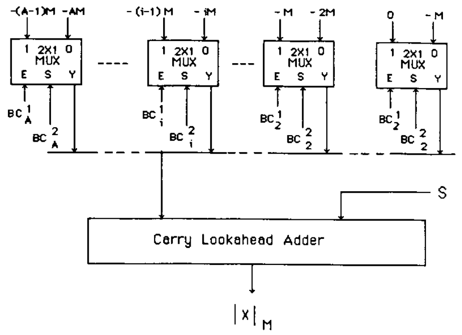

Figure 6. Implementation of Final Corrector Stage

4. Performance Evaluation

[1] The partial sum generator is implemented using small ROMs, If the number of residues is N and each residue is represented in P bits, then it is required to use N ROMs. Each ROM is storing values bounded by M, then the size of each ROM is $2^P * \lceil \log M \rceil$. The total area required for this stage is: $N * 2^P * \lceil \log M \rceil$. Since ROMs have a constant time delay (P is a small number) which does not depend on N, then the delay of this stage is $\theta(1)$.

[2] The partial sum adder is implemented in two different ways:
(a) Using CSAs: The complexity of the scheme is determined by Theorem 1. Since each CSA has a constant time delay, then the total time required to add N numbers in modulo M is $\theta(logN)$.
(b) Using MCSAs: The number of levels required to perform the addition of N numbers using a binary tree of MCSAs is $\lceil \log N \rceil$ as it is shown in Theorem 3. Since at each level the required time is constant (MCSA has a constant time), then the total time required for this step using MCSAs is $\theta(logn)$.

[3] The range determinator consists of three different levels(Figure 4). The first level consists of g ROMs. The second level is the MC cells, which are combinational circuits that can

be represented with a two level switching function. Finally the last level is a two stage combinational circuit. The Three levels have a constant time delay that does not depend on N. The previous analysis shows that the range determinator has a time delay of $\theta(1)$.

[4] The Final corrector consists of two stages. In the first stage we have A tristate multiplexers which have a constant delay equivalent to two serial NAND gates. The second stage is a CLA which has a constant delay and for number of bits less than 64 the delay is equivalent to the delay of 12 serial NAND gates as shown in[13]. For number of bits larger than this we can still obtain a constant delay CLA. Then the final corrector has a delay of $\theta(1)$.

From cases [1]-[4] we see that all stages except the partial sum adder has a constant time delay which does not depend on the number of residues N. Only the second stage requires $\theta(logN)$ steps.

5. Conclusions

The residue decoder introduced in this paper has a total delay of $\lceil \log N \rceil$. In addition, it has several advantages as listed below:

1) The design is quite modular and consists of simple cells such as small ROMs and MC cells. This makes the implementation of the whole residue decoder in a single chip is possible.
2) It doesn't have any limitation on the moduli used.
3) It is flexible since it can convert residues to either unsigned magnitude or 2's complement number system, and it is controlled by only a control line, C. This means that it can be applied to wider area.
4) It is fast compared with most schemes proposed before since it has a time complexity of $\theta(\log N)$.
5) It can be easily pipelined without any modifications.

Acknowledgement

The authors wish to thank the reviewers for thier valuable comments.

References

[1] M. A. Bayoumi, "Digital Filter VLSI Systolic Arrays over Finite Fields for DSP Applications," Proc. of the 6th IEEE Annual Phoenix Conference on Computers and Communications, pp. 194-199, Feb. 1987.

[2] M. A. Bayoumi, G. A. Jullien, W. C. Miller, "A Look-up Table VLSI Design Methodology for RNS Structures Used in DSP Applications," IEEE Trans. on Circuits and Systems, pp. 604-616, Vol. 34, No. 6, June 1987.

[3] F. J. Taylor, "Residue Arithmetic: A Tutorial with Examples," IEEE Computer Magazine, pp. 50-62,May 1984.

[4] M. A. Bayoumi, "A High Speed VLSI Complex Digital Signal Processor Based on Quadratic Residue Number System," VLSI Signal Processing II, pp. 200-211, IEEE Press, 1986.

[5] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, New York: McGraw-Hill, 1967.

[6] W. K. Jenkins, "Techniques for Residue to Analog Conversion for Residue Encoded Digital Filters," IEEE Trans. Circuits Syst., vol. CAS-25, pp. 553-562, July 1978.

[7] S. Andraos and H. Ahmed, "A New Efficient Memoryless Residue to Binary Converter," IEEE Trans. Circuits Syst., vol. 35, Nov. 1988, pp. 1441-1444.

[8] K. M. Ibrahim and S. N. Saloum, "An Efficient Residue to Binary Converter Design," IEEE Trans. Circuits Syst., vol. 35, pp 1156-1158, September 1988.

[9] S. Bandyopadhyay, G. A. Jullien, and A. Sengupta, "A Systolic Array for Fault-Tolerant Digital Signal Processing Using A Residue Number System Approach," Proc. of Intl. Conf. on Systolic Arrays, pp. 577-586, 1988.

[10] A. P. Shenoy and R. Kumaresan, "Residue to Binary Conversion for RNS Arithmetic Using Only Modular Look-up Tables," IEEE Trans. circuits Syst., vol. 35, pp. 1158-1162, September 1988.

[11] T. V. Vu, "Efficient Implementations of the CRT for Sign Detection and Residue Decoding," IEEE Trans. Comp., vol. C-34, pp. 646-651, July 1985.

[12] C. N. Zhang, B. Shirazi, and D. Y. Y. Yun, "Parallel Designs for Chinese Remainder Conversion," Proc. IEEE 16 th Annual Conf. on Parallel Processing, Aug. 1987.

[13] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*. New York: Wiley, 1978.

[14] R. M. Capocelli and R. Giancarlo, "Efficient VLSI Networks for Converting an Integer from Binary System and Vice Versa," IEEE Trans. Circuits Syst., vol. 35, Nov. 1988, pp. 1425-1430.

[15] G. Alia and E. Martinelli, "A VLSI Algorithm for Direct and Reverse Conversion from Weighted Binary Number System to Residue Number System," IEEE Trans. Circuits Syst., vol. CAS-31, 1984, pp. 1033-1039.

[16] K. P. Lee, M. A. Bayoumi and K. M. Elleithy, "A Fast and Flexible Residue Decoder Based on The Chinese Remainder Theorem," The 1989 International Symposium on Circuits and Systems.

[17] K. M. Elleithy, "On Bit-Parallel Processing for Modulo Arithmetic," VLSI Technical Report TR86-8-1, The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1986.

[18] A. Avizienis, "A Study of Redundant Number Representations for Parallel Digital Computers," Ph.D Thesis, Univ. of Illinois, Urbana, Illinois, May 1960.

[19] K. M. Elleithy, "On the Bit-Parallel Implementation for the Chinese Remainder Theorem," VLSI Technical Report TR87-8-1, The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1987.