

Modulized RNS-Decimal Number Conversion Algorithm And Its Implementations

Zhi Li1, Ramasamy Krishnan1,2, Robert J. Marks II1

1. Interactive Systems Design Lab
Department of Electrical Engineering, FT - 10
University of Washington
Seattle, WA 98195, USA
Tel: (206) 543 - 6990

2. Boeing High Tech Center
Mail Stop 73 - 24
P.O. Box 3000
Seattle, WA 98124, USA
Tel: (206) 865 - 3038

Abstract

In this paper, we investigate residue number system (RNS) to decimal number system conversion algorithms and their implementations. We first, for the general Q-tuple RNS decoding, propose a modularized decoding algorithm in which each module decodes only a 2-tuple RNS code, and discuss its computational complexity at the algorithm level. The second algorithm is for decoding a 2-tuple RNS code and can be called a partial table-lookup method. By memorizing only partial decimal numbers in the dynamic range of any RNS, this method can generate the correct decimal number with at most three additions. We discuss hardware implementation methods and compare our algorithm with the conventional decoding algorithm in terms of the arithmetic operation complexity, dynamic range requirements and hardware implementation complexity.

I. Introduction

The Residue Number System (RNS) has attracted a great deal of attention recently in application in ultra-speed, dedicated, real-time systems that support parallel processing of integer-valued data, because of its two attractive features [1, 2, 6, 9, 12, 14]. Most importantly, no carry mechanism is needed in residue arithmetic. The second feature is that RNS decomposes a computation into subcalculations of smaller computational complexity. However, a much higher accuracy is achieved after the results of these low-accuracy subcalculations are recombined.

In this paper, we advocate a new RNS to decimal conversion algorithm and discuss its implementations. We first propose a modularized decoding algorithm in which each module decodes only a 2-tuple RNS code. The second algorithm is for decoding a 2-tuple RNS code and can be called a partial table-lookup method. By memorizing only partial decimal numbers in the dynamic range of any RNS, this method is able to generate the correct decimal number for any given code with at most three additions. We discuss implementation methods at both the algorithm and hardware level and compare our algorithm with the conventional decoding algorithm, namely, the Chinese Remainder Theorem (CRT) and the Mixed Radix Representation Algorithm (MRC), in terms of the arithmetic operation complexity, dynamic range requirements and the hardware implementation complexity.

This paper is organized as follows: Section II is a brief overview of RNS and its conventional decoding algorithms. Section III introduces two procedures for the conversion and corresponding implementations. Section IV is the conclusion.

II. An Overview of the RNS

The RNS is based on Q fixed and relatively prime (i.e., containing no common factors, except 1 ) integers m1, m2, ..., mQ, which are called moduli. The residue of any integer x with respect to a particular moduli mi is denoted

$$R_i = (x \mod (m_i))$$

which is the least positive integer remainder of the division of x by Ri [3]. The Q-tuple of residues

$$(R_1, R_2, \cdots, R_Q)$$

with respect to the Q different moduli provides a unique representation of any integer x in the range 0 to N - 1, where the N is the product of all the moduli

$$N = \prod_{i=1}^{Q} m_i \ldots (1)$$

The N is called the legitimate dynamic range of the RNS.

In the RNS applications, conversion of the RNS to a more conventional numerical form is a singularly important operation. Until now, the most commonly used decoding methods of RNS to decimal system are the CRT and MRC [2,3].

The basic formula for CRT is

$$x = \left( \sum_{i=1}^{Q} \hat{m}_i \left[ \frac{R_i}{x_{m_i}} \right] \mod (N) \right) / \hat{m}_i$$

(2)

where, $\hat{m}_i = \frac{N}{m_i}$ and $\frac{N}{m_i}$ is the multiplicative inverse of the $\hat{m}_i$.

From a computational point of view, the CRT is extremely costly, for it requires a number of conventional multiplications and additions [2,9]. Furthermore, it needs a large dynamic range at the final mod(N) operation.

The MRC transforms the Q-tuple RNS code into Q coefficients of a mixed radix number representation of

$$x = a_1 + a_2m_1 + a_3m_1m_2 + \cdots + a_Qm_1m_2\cdots m_{Q-1}$$

(3)

To obtain the coefficients, we first divide the x by the $m_1$, the remainder will be $a_1$ (which is $R_1$). Then, we subtract the $a_1$ from the x and then divide the difference by $m_2$, we get the $a_2$. This process continues until we acquire all the coefficients.

The disadvantage of the MRC is the decoding time is linearly proportional to the number of the moduli of the RNS. The computation load is still relatively heavy, with the computational complexity of $O(Q^2)$ [11].

III. Modulized RNS Decoding Algorithms

3.1 Modulized RNS Decoding Procedure and Partial Table-Lookup Method:
In this section, we discuss two procedures for RNS decoding.

3.1.1 Modulized RNS Decoding Procedure

Procedure-1:

Any Q-tuple RNS code can be decoded by a modular structure of (Q-1) 2-tuple RNS decoders arranged in \([\log_{2} Q]\) levels. Each module is arranged as a binary tree structure. (Here, \([x]\) is the ceiling function of \(x\).)

For each decoder:

1. The two moduli are the dynamic ranges of the 2 decodes in the immediate previous level which the decoder is connected. Those in the first level are directly from the chosen RNS;
2. A 2-tuple RNS code is the two decoded outputs form the immediate previous level with which the decoder is connected. That in the first level is directly from the code to be decoded;
3. Each module decodes the 2-tuple RNS code into a decimal number which is unique within the dynamic range of the decoder. The structure continues until we get only one final decoded decimal number.

To prove this procedure works, we only need to notice the uniqueness of the RNS code with respect to its decimal number and the duality between the encoding and decoding process, and then apply a simple mathematical induction. The proof is omitted here.

Figure 1 shows the decoder structure. It is very suitable for the VLSI implementation because of its modularity and local duality.

![Fig. 1 Modulized Q-tuple RNS Decoder Structure](image)

As an example for the modulized decoder structure of Fig.1, we use the 2-tuple MRC to implement each box. Here, the computation involved at each box is:

\[
d(R_1, R_2) = R_1 + m_1 \left\lfloor \frac{R_2 - R_1}{m_2} \right\rfloor_{m_2} \mod (m_2) \tag{4}\]

Let us have a numerical example: assuming a RNS has moduli: \((m_1, m_2, m_3, m_4) = (5, 7, 9, 4)\), the code to be decoded is \((R_1, R_2, R_3, R_4) = (1, 2, 3, 0)\). At the top level, there are 2 decoders, and the bottom level is 1 decoder. At the top level, the left side decoder decodes the \((R_1, R_2) = (1, 2)\) with respect to the \((m_1, m_2) = (5, 7)\). From the MRC, we have the output as the \(R_{15} = 16\) which is unique in the range of \(m_1 m_2 = 5 \times 7 = 35\). Similarly, the right side decoder decodes \((R_3, R_4) = (3, 0)\) into \(R_{34} = 12\) with respect to the \(m_3 m_4 = 9 \times 4 = 36\). The bottom decoder take the \((R_{13}, R_{34})\) as inputs with respect to the moduli \((35, 36)\) respectively. Note that 35 is relatively prime to 36. The decoded number \(R_{134} = 156\).

Comparing the modulized MRC and conventional MRC, here we see the modulized MRC algorithm improves the performance over the conventional MRC in: (1). If a single processing element (PE) is used to implement both algorithms, then, at the algorithm level, our modulized MRC reduced the complexity from \(O(Q^2)\) to \(O(Q)\). (2). The total decoding time is reduced if each module is implemented by a individual PE. Time complexity is proportional to \([\log_{2} Q]\). However, that for the conventional MRC is \(O(Q)\). (3). About the dynamic range requirement for the Modulized MRC, generally speaking, the closer to the final stage a module is, the wider the required dynamic range. Nevertheless, none of them exceeds the \(N\).

3.1.2 Partial Table-Lookup Method

Motivated by Procedure 1, we propose a partial table-lookup method only for the 2-tuple RNS decoding.

First, let us put all the decimal numbers within the dynamic range of a 2-tuple RNS into a \(m_1\) by \(m_2\) matrix \(D\), with the first element with the index of \((0, 0)\) and the last element with \((m_1 - 1, m_2 - 1)\).

\[
D = \{d(i, j)\} = \{d(R_1, R_2)\} = \begin{pmatrix} D_0 & D_1 & \cdots \ & D_{m_1 - 1} \\
\end{pmatrix} \tag{5}
\]

where, the \(d(R_1, R_2)\) is the decimal number which has the RNS code as \((R_1, R_2)\). We will refer \(k\)th row of the \(D\) matrix as the \(D_k, k \in [0, m_1 - 1]\).

For example, for \(m_1 = 3, m_2 = 2\), the matrix \(D\) is:

\[
D = \begin{pmatrix} 0 & 3 \\
4 & 1 \\
2 & 5 \\
\end{pmatrix} \tag{6}
\]

In the RNS, the order of the moduli can be any order desired. In the following discussion, we will refer only the row vectors of \(D\). But any conclusions we have for the rows can be applied to the columns.

Procedure-2:

In the matrix \(D\), any element \(d(R_1, R_2)\) can be obtained by a mapping from one element in the first row with a relation summarized as:

\[
d(0, R_2) + R_1 = d(R_1, (R_1 + R_2) \mod (m_2)) \tag{7}
\]

This equation tells us two properties of the \(D\):

- **Property 1.** The set of all the elements in \(k\)th vector \(D_k, k \in [1, m_1 - 1]\) is equal to the set of the elements in vector \(D_{k+1}\) added with the row index \(k\).

- **Property 2.** In the vector \(D_k\), for the element \(d(0, R_2)\), after the set mapping of Property 1, the resultant decimal number has the row index of \(R_1\) and the column index of \((R_1 + R_2) \mod (m_2)\).

A formal proof of this procedure is in the appendix. A quick check with the example in (6) shows this truth.
3.2 Implementation of Partial Table-Lookup

We can use the Procedure 2 to decode any \( R_1, R_2 \) in the \( D \) and save the computations encountered in the traditional 2-tuple CRT and MRC once we know the \( m_2 \) decimal numbers in the \( D_0 \). In the following, we will present three ways to implement it.

3.2.1 Linear Array Method

Once the \( D_0 \) vector which is derivable from the Eq.(4) are stored, and put them into an linear array data structure\textsuperscript{10} according to the the index of \( R_1 \) and \( R_2 \). (To be distinctive, in this part, we use the hat sign to represent the known RNS code to be decoded.) From Eq. (7), let

\[
d(\hat{R}_1, \hat{R}_2) = d[R_1, (R_1 + R_2) \mod (m_2)]
\]

then, we have

\[
\begin{align*}
\hat{R}_1 &= R_1 \\
\hat{R}_2 &= (R_1 + R_2) \mod (m_2)
\end{align*}
\]

which will give us the value:

\[
R_2 = [\hat{R}_2 - \hat{R}_1] \mod (m_2)
\]  
(9)

Since the following bounds exist:

\[
-m_1 < \hat{R}_2 - \hat{R}_1 < m_2
\]  
(10)

the \( \mod (m_2) \) operation is seen as a way to make the value of \( R_2 \) positive. Since for any \( x \)

\[
(-x) \mod (m_2) = (m_2 - x) \mod (m_2).
\]

In order to let the \( R_1 \) in Eq. (9) has a positive value with the smallest number of arithmetic operations, we should choose the order of the moduli such that

\[
m_1 < m_2
\]  
(11)

When (11) is satisfied, the algorithm is summarized as follows.

Algorithm-1

1. Compute the index \( p \):

\[
p = \hat{R}_2 - \hat{R}_1
\]

2. Test if

\[
p \geq 0?
\]

If so, goto next step; If not, add \( m_2 \) to \( p \);

3. Select the corresponding decimal number with the \( R_2 \) index equal to the \( p \) from the step \( 2 \) in \( D_0 \) vector;

4. Add the \( R_1 \) to that decimal number.

The advantage for Algorithm-1 is that it can decode any \( \hat{R}_1, \hat{R}_2 \) with at most 3 additions and one logic comparison. The dynamic range is always less than that of the chosen RNS.

3.2.2 Circular Shift Register Decoder

Consider the decoder shown in Fig. 2 with circularly shift register bank of length \( m_2 \), a multiplexer and an adder. The register bank initially contains the decimal number of \( D_0 \) from left-to-right according to the index of \( R_2 \). \( R_1 \) controls the number of shift. With one shift from the initial state of the register, we get the \( D_1 = I \) vector. (Here the \( \hat{n} \) is a constant vector of the same length of \( D_0 \) as all elements as n.) With 2 shifts from the initial state, we have the \( D_2 = 2 \) vector. \( \hat{R}_2 \) selects a number from the \( D_0 \) vector; finally, the value of \( R_1 \) is added to that number.

With respect to Fig. 2, to decode \( \hat{R}_1, \hat{R}_2 \), the detailed operation is:

Operation-1

1. Let the register shift for \( R_1 \) times, then, the register states are hold.

2. \( R_2 \) selects the output of the \( R_1 \)th register.

3. The value of \( R_1 \) is added to the decimal number from the step \( 2 \), then we have the decoded decimal number;

4. Refreshing all the register states to the initial statues. The decoder is ready to accept a new code of \( (R_1, R_2) \).

The advantage of the simple decoder Fig.2 is that it only involves \( R_1 \) shift operations and one addition. The required dynamic region is always less than \( N \). The limitation is that the decoding delay is linearly proportional to \( R_1 \). In the worst case, it needs \( m_1 \) shifts. More specifically, if \( T_1 \) is the time needed for one shift and \( T_2 \) is for the addition, then response time for the decoder in Fig. 2 is \( m_1 T_1 + T_2 \).

3.2.3 A Constant-Time Decoder

In order to overcome the limitation of the Fig. 2, we replace the addressing mechanism of shifting by a direct address computation. Figure 3 shows a hardware implementation of Algorithm-1 from which a constant response time is expected.

In Figure 3, we still assume the condition of Eq. (11) true. Then, the conditional addition of the \( m_2 \) with the result of \( R_2 - R_1 \) is controlled directly from the sign bit for the each bit fed to the second adder. For the response time, only \( 3T_1 \) is needed to decode a 2-tuple RNS code.
VI. Conclusion

In this paper, we proposed a modularized RNS decoding algorithm and a partial table lookup algorithm. Compared with the CRT and the conventional MRC, our algorithms have the following advantages: 1. Fast decoding with less arithmetic operations: if a single PE is used, then the computational load is reduced from $O(Q^2)$ in the conventional MRC to $O(Q)$; if it is implemented by $(Q-1)$ PEs, then the modularized decoder reduces the decoding time of the conventional MRC from $O(Q)$ to $O(\log_2 Q)$. Furthermore, pipelining can be used; 2. The dynamic range can be constrained not to exceed that of the RNS; 3. Existence of a trade off between the decoding speed and the memorized data; 4. In compared with the conventional Table Lookup method, our partial table lookup algorithm can be implemented with much less ROM space.

Appendix

Equation 7 can be proved from Eq. (4) by a substitution.

$$d(R_1, R_2 \mod (m_2))$$
$$= R_1 + m_1 \lfloor \frac{[R_1 + R_2 \mod (m_2) - R_1]^{-1}}{m_1} \rfloor \mod (m_2)$$
$$= R_1 + m_1 \lfloor \frac{[R_2]^{-1}}{m_1} \rfloor \mod (m_1)$$
$$= R_1 + d(0, R_2).$$