Choosing the System Moduli of RNS Arithmetic Processors

Khaled M. Elleithy

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Abstract

Designing an optimal Residue Number System (RNS) processor in terms of area and speed depends on the choice of the system moduli. In this paper an optimal algorithm for choosing the system moduli is presented. The algorithm takes into consideration several constraints imposed by the problem definition. The problem is formalized as an integer programming problem to optimize an area/time objective function.

1: Introduction

Residue Number System (RNS) has received increased attention due to its ability to support high-speed concurrent arithmetic [1-5]. Although, Digital Signal Processing (DSP) applications utilize the efficiencies of RNS arithmetic 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 DSP processors.

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 high speed RNS operations, the residue-to-binary conversion can be a bottleneck. Chinese Remainder Theorem (CRT) [6] is considered the main algorithm for the conversion process. Several implementations of the residue decoder have been reported [2,5,7-10].

Designing an optimal RNS processor in terms of area and speed depends on the choice of the system moduli. Most of the reported implementations in the literature for choosing the system moduli are based on using a special moduli The residue decoders in [7,8] are based on using three moduli in the form $(2^n - 1, 2^n, 2^n + 1)$, where n is the number of bits. Due to the limitation imposed on the number of moduli and the choice of them, it is limited in application. In [9], the residue decoder is based on the base extension technique, it uses only modular look-up tables in its implementation. Since look-up tables are used, 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. The implementation in [10] requires that one of the moduli must be a power of two; therefore, it may be limited in application. Due to the constraints imposed on the chosen moduli set, such approaches have limited applications and the final design is not optimal in area and time.

In this paper an optimal algorithm for choosing the system moduli is presented. The algorithm takes into consideration several constraints imposed by the problem definition. The problem is formalized as an integer programming problem to optimize an area/time objective function.

2: Residue Number System

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

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

where $r_i = |X|_{m_i}$ with respect to a set of N moduli set $(r_1, r_2, \ldots, r_n)$. In order to have a unique residue representation, the moduli must be pairwise relatively prime, that is,

$$GCD(r_i, r_j) \neq 1 \qquad \forall i \neq j$$

then it is shown that there is a unique representation for each number in the range of:

$$0 \le X \le (\prod_{i=1}^{n} m_i = M)$$

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

1058-6393/97 $10.00 © 1997 IEEE

$$|A \bullet B|_m = \left(\left| |A|_{m1} \bullet |B|_{m_1} \right|_{m_1}, \ \ldots , \ \left| |A|_{m_n} \bullet |B|_{m_n} \right|_{m_n} \right)$$

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

The conversion from RNS to weighted binary number system is done using the Chinese Remainder Theorem (CRT), which states that:

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

where

$$M = \prod_{j=1}^{N} m_j \quad , \quad \hat{\mathrm{m}}_\mathrm{j} = \frac{M}{m_j} \quad , \quad GCD(m_j, m_k) = 1 \ \ \forall \mathrm{j} \neq \mathrm{k}$$

3: ROM Requirements

A typical architecture for RNS-based processor is shown in Figure 1. The architecture consists of four stages. In the first stage a variable X is converted to its RNS representation. In the second stage an arithmetic operation is performed. The third and fourth stages perform the conversion from RNS representation to weighted number system representation. The third stage is responsible for the computation of *t's* according to the following definition:

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

The fourth stage is a modulo adder that adds the *t's* inputs and gives the final weighted value.

The ROM required for the RNS based system can be divided into the following:

1- ROM required to implement processor *i* mod m_i , $l \leq i \leq n$. The processor implementation is shown in Figure 2. The ROM required for each processor is given by:

$$2^{\lceil \log m_i \rceil} * 2^{\lceil \log m_i \rceil} * \lceil \log m_i \rceil$$

The total memory requirements for *i* processors is given by:

$$\sum_{i=1}^{n} 2^{2 \lceil \log m_i \rceil} * \lceil \log m_i \rceil \qquad (1)$$

2- ROM required to implement *t's*. The ROM size is given by:

$$\sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i) , \quad l_i = \left\lceil \log \left\{ \frac{M}{m_i} \left| \frac{r_i}{\hat{m}_i} \right|_{m_i} \right\} \right\rceil \qquad (2)$$

3- If the summation unit is implemented using ROM, as shown in Figure 3, the memory required is:

$$\{ 2^{\lceil \log m_1 \rceil} * 2^{\lceil \log m_2 \rceil} * \ldots * 2^{\lceil \log m_n \rceil} \} * l = \qquad (3)$$

$$l * 2^{\sum_{i=1}^{n} \lceil \log m_i \rceil} \quad \text{, where} \quad l = 2^{\lceil \log M \rceil}$$

The ROM implementation of the summation unit is impossible since it needs huge memory even for a small size system.

4- The implementation of processors as well as *t's* calculation can be implemented using ROMs as given in equations 1 and 2. The total size of ROM is:

$$\sum_{i=1}^{n} 2^{2 \lceil \log m_i \rceil} * \lceil \log m_i \rceil + \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i) \qquad (4)$$

4: Implementation of the Chinese Remainder Theorem

Chinese Remainder Theorem (CRT) is considered the main algorithm for the conversion process. Several implementations of CRT have been reported[2,5,7-10]. In [10] Vu introduced an algorithm for computing the CRT based on manipulating the form the summands stored in ROMs. The main idea in the algorithm is representing the summands in a different form doesn't require any special real-time processing as they are stored in ROMs. The summands are stored in a form that can be efficiently used in evaluation of the final summation modulo M. The following derivation is used:

$$X = \left| \sum_{i=1}^{n} S_i \right|_M$$

$$= \left| \sum_{i=1}^{n} \left(q_i \frac{M}{m_j} + r_i \right) \right|_M$$

$$= \left| \left| \frac{M}{m_j} \sum_{i=1}^{n} (q_i) \right|_M + \left| \sum_{i=1}^{n} (r_i) \right|_M \right|_M$$

$$= \left| \left| \frac{M}{m_j} \sum_{i=1}^{n} (q_i) \right|_{m_j} + \left| \sum_{i=1}^{n} (r_i) \right|_M \right|_M$$

Computing the first part of the sum is easy because modules m_j is small and can be implemented using ROMs. However the second part is obtained most easily when m_j equal 2^k or 2^k-1. Since it is not unusual to have a power of 2 included in most practical systems of moduli, $m_j = 2^k$ is assumed. The following equation is valid:

$$0 \le \sum_{i=1}^{n} r_i \prec \frac{nM}{m_j} \le M, \text{ if } n \le (m_j = 2^k)$$

4.1 Implementation details

One of the moduli must be equal to 2^k. A hardware implementation is given in Figure 4. There are four levels in implementing the modulo M adder as follows:

1. At level 1 a Mod 2^k adder is used. The speed depends on the speed of the multi-operand adder. An optimal adder of $\Theta(\log n)$ time complexity[3] can be used in this stage.
2. Level 2 is a small size ROM.
3. Levels 3 and 4 are 2-operand adders.

The size of buses and ROMs used are as follows:

1. q_i requires $\log m_j$ bits $(0 \le q_i \le m_j)$
2. r_i requires $\left\lceil \log \frac{M}{m_j} \right\rceil$ bits $(0 \le r_i \le \frac{M}{m_j})$
3. q requires $\lceil \log m_j \rceil$ bits $(q = \left| \sum_{i=1}^{n} q_i \right|_{m_j})$
4. r requires $\lfloor \log M \rfloor$ bits $(r = \left| \sum_{i=1}^{n} r_i \right|_{M})$
5. Y requires $\lfloor \log M \rfloor$ bits $(Y = q(\frac{M}{2^k} + 2^c - M)$
6. $\tilde{Z}$ requires $\lfloor \log M \rfloor$ bits

4.2 Choosing the System moduli

4.2.1 Choosing the system moduli to minimize the ROM size: The following two cases are considered:

(1) ROM is used to obtain ti's, then the problem is modeled as follows:

$$\begin{aligned} &\textit{minimize} \ \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i), \quad l_i = \left\lceil \log \left\{ \frac{M}{m_i} \left| \frac{r_i}{\hat{m}_i} \right|_{m_i} \right\} \right\rceil \\ &\textit{such that} \quad \begin{array}{l} m_1 * m_2 * \ldots\ldots * m_n \ge 2^l - 1 \\ m_1, m_2, \ldots\ldots, m_n \ \textit{are relatively prime} \end{array} \end{aligned}$$

(2) ROM is used to obtain ti's and implement the processors, then the problem is

$$\begin{aligned} &\textit{minimize} \ \sum_{i=1}^{n} 2^{2\lceil \log m_i \rceil} * \lceil \log m_i \rceil + \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i) \\ &\textit{such that} \quad \begin{array}{l} m_1 * m_2 * \ldots\ldots * m_n \ge 2^l - 1 \\ m_1, m_2, \ldots\ldots, m_n \ \textit{are relatively prime} \end{array} \end{aligned}$$

As mentioned earlier the problem can be solved using integer programming. An extra constraint that the moduli are relatively prime. It is not an integer programming problem, it needs some modifications, it can be called Prime Integer Programming (PIP).

4.2.2 Special Case of $M = 2^l$: In this section a special case is considered. The required condition for M is that $M \ge 2^l - 1$. If $M = 2^l$ the difficult problem of designing a modulo M will be reduced to the design of multi-operand adder. The integer programming problem is represented as:

$$\begin{aligned} &\textit{minimize} \ \sum_{i=1}^{n} 2^{2\lceil \log m_i \rceil} * \lceil \log m_i \rceil + \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i) \\ &\textit{such that} \quad \begin{array}{l} m_1 * m_2 * \ldots\ldots * m_n = 2^l - 1 \\ m_1, m_2, \ldots\ldots, m_n \ \textit{are relatively prime} \end{array} \end{aligned}$$

Solving this special case is as difficult as the general case. There may not be a feasible solution. If there is no feasible solution the general problem is solved instead of the special case problem.. If there is a feasible solution for the special case, it is likely that the memory requirements may be larger than the general case. In this case we have to compare between larger memory and high speed in realizing the summation network, on the other side, less memory and complicated summation network.

4.2.3 Choosing one of the moduli equal to 2^k: The implementation discussed in section 4.1 requires one of system moduli to be equal to 2^k. In this case the problem is defined as follows:

$$\begin{aligned} &\textit{minimize} \ \sum_{i=1}^{n} 2^{2\lceil \log m_i \rceil} * \lceil \log m_i \rceil + \\ &\qquad \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i) \\ &\textit{such that} \quad \begin{array}{l} m_1 * m_2 * \ldots\ldots * m_n \le 2^l - 1 \\ m_1, m_2, \ldots\ldots, m_n \ \textit{are relatively prime} \\ m_j = 2^k \ \textit{for any } j \textit{ and } k \end{array} \end{aligned}$$

As in the previous case, there may not be a feasible solution. If there is no feasible solution the general problem is solved instead of the special case problem.. If there is a feasible solution for this special case, it is likely that the memory requirements may be larger than the general case. In this case we have to compare between larger memory and high speed in realizing the summation network, on the other side, less memory and complicated summation network.

4.3 New Algorithm for choosing the system moduli

From our previous analysis in section 4.2 we see that the problem of choosing the system moduli is presented as an integer programming problem with the following objective function:

$$\sum_{i=1}^{n} 2^{2\lceil \log m_i \rceil} * \lceil \log m_i \rceil + \sum_{i=1}^{n} (2^{\lceil \log m_i \rceil} * l_i)$$

4.3.1 Algorithm

Step #1. Solve objective function such that:

$m_1 * m_2 * \ldots\ldots * m_n \leq 2^l - 1$

$m_1, m_2, \ldots\ldots, m_n$ *are relatively prime*

$m_j = 2^k$ *for any j and k*

Step #2. Solve objective function such that:

$m_1 * m_2 * \ldots\ldots * m_n \leq 2^l - 1$

$m_1, m_2, \ldots\ldots, m_n$ *are relatively prime*

$m_j = 2^k$ *for any j and k*

Step #3. Solve objective function such that:

$m_1 * m_2 * \ldots\ldots * m_n \leq 2^l - 1$

$m_1, m_2, \ldots\ldots, m_n$ *are relatively prime*

Step #4. Choosing solution

(a) Speed

Sol 1 (if exists) is the fastest

Sol 2 (if exists) is slower than Sol 1

Sol 3 is the slowest

(b) Memory requirements

Sol 3 needs the least memory

Sol 2 needs memory equal or greater than Sol 3

Sol 1 needs memory similar to Sol 3

(c) Implementation

Sol 1: using n-operands adder.

Sol 2: is implementation-A[Figure 4].

Sol 3: is Elleithy's implementation[2].

4.3.2 Results: The problem for choosing the system moduli has been solved using the integer programming approach with extra constraint of having relatively prime moduli set. Table 1 shows the results starting from 7-bit output to 16 bits. The approach is general and can be used to obtain results for larger bits. For each bit size a number of solutions are obtained. In Table 1 only the solutions with the minimum ROM size are shown.

5: Conclusions

Conversion from Binary Representation into RNS Representation is a time consuming process. In this paper the (CRT) is used as the main algorithm for the conversion process. An algorithm to design optimal RNS processor in terms of area and speed depends on the choice of the system moduli. In this paper an optimal algorithm for choosing the system moduli is presented. The algorithm takes into consideration several constraints imposed by the problem definition. The problem is formalized as an integer programming problem to optimize an area/time objective function.

Acknowledgments

The author would like to acknowledge King Fahd University of Petroleum and Minerals for support provided for this work.

References

1- K. M. Elleithy, and M. A. Bayoumi, *"A Systolic Architecture for Modulo Multiplication,"* IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 42, no. 11, pp. 725-729, Nov. 1995.

2- K. M. Elleithy and M. A. Bayoumi, *"Fast and Flexible Architectures for RNS Arithmetic Decoding,"* IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 39, no. 4, pp. 226-235, April 1992.

3- K. M. Elleithy and M. A. Bayoumi, *"A $\Theta(1)$ Algorithm for modulo Addition,"* IEEE Transactions on Circuits and Systems, vol. 37, no. 5, pp. 628-631, May 1990.

4- K. M. Elleithy and M. A. Bayoumi "A theta(1) Algorithm for Modulo Multiplication," Proc. of the 32nd Midwest Symposium on Circuits and Systems, Aug. 1989.

5- K. M. Elleithy, M. A. Bayoumi, and K. P. Lee, "$\Theta(\log n)$ Architectures for RNS Arithmetic Decoding," Proc. of the 9th Symposium on Computer Arithmetic, pp. 202-209, Sep. 1989.

6- S. Szabo and R. I. Tanaka, "Residue Arithmetic and its Applications to Computer Technology," New York: McGraw-Hill, 1967.

7- Andraos and H. Ahmed, "A New Efficient Memoryless Residue to Binary Converter," IEEE Trans. Circuits Syst., vol. 35, Nov. 1988, pp. 1441-1444.
8- 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- 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.
10- V. Vu, "Efficient Implementations of the CRT for Sign Detection and Residue Decoding," IEEE Trans. Comp., vol. C-34, pp. 646-651, July 1985.
11- 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.

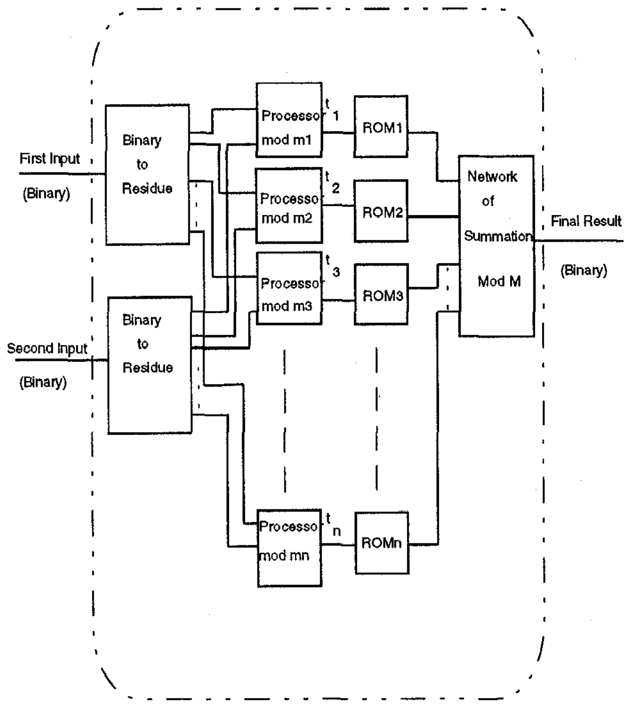

Figure 1. RNS-Based Architecture.

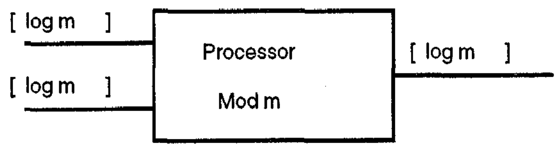

Figure 2. Processor i mod m_i Implementation.

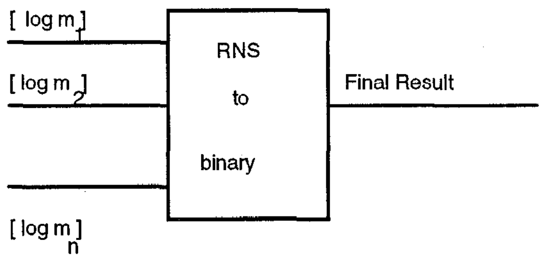

Figure 3. Implementation of the CRT Decoder.

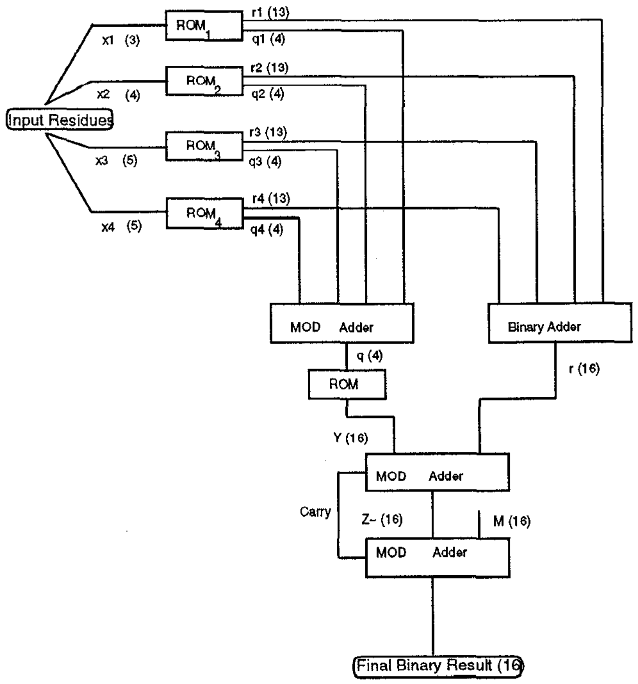

Figure 4. Implementation of the CRT decoder with one of the moduli equal 2^k

	7	8	9	10	11	12	13	14	15	16
Sol1		480	512	1024	1088	1152		2300	5885	6600
Sol2	342	810	878	1468	2277	2409	2541	2673	6488	7648
Sol3	448	480	512	1024	1088	1152	2541	2300	5885	6600

Table 1: Optimal ROM size for RNS processors where the bus size varies between 7 and 16 bits.