# APPLICATION OF THE RESIDUE NUMBER SYSTEM TO COMPUTER PROCESSING OF DIGITAL SIGNALS

G. A. Jullien and W. C. Miller

Department of Electrical Engineering
University of Windsor
Windsor, Ontario, Canada, N9B 3P4

## ABSTRACT

The residue number system offers parallel processing, digital hardware, implementations for the binary operations of addition, subtraction and multiplication. This paper discusses the use of the residue number system in implementing digital signal processing functions, in which these binary operations abound. The paper covers implementations using arrays of read only memories, and briefly discusses the use of parallel microprocessor structures. ROM array implementations of scaling operations are also presented.

## INTRODUCTION

Nearly all digital signal processing structures consist of interconnections of the two basic arithmetic operations: addition and multiplication. The residue number system (RNS) has unique advantages when used to implement such structures. These advantages stem from the fact that parallel independent paths can be used to perform the arithmetic operations. Each path can be constructed from a moderate IC package count, and be made to operate at very high data rates.

The theoretical basis for the RNS has its ground roots in Fermat and Gauss, though the modern, computer oriented work was done in the 50's and 60's. In particular, the work of Svoboda[1] and Szabo and Tanaka[2] are important primary references.

In order to introduce the notation, and to point out the special properties of the RNS, the following introduction is presented.

The RNS representation of a number, X, takes the form of an N-tuple $X=(x_0, x_1, .., x_{N-1})$

Each digit, or residue, is found from:

$$x_i = X \text{ Modulo } m_i \quad \text{or} \quad x_i = |X|_{m_i}$$

The $\{m_i\}$ are relatively prime and yield a number range $M = \prod_{i=0}^{N-1} m_i$. A signed integer system can be developed using the additive inverse, or modulo complement notation, where the complement of X, $\bar{X}$, is found from $\bar{X} = M - X$. The complement operation on individual residues is $\bar{x}_i = m_i - x_i$. Thus $\bar{X} = (\bar{x}_0, \bar{x}_1, ...., \bar{x}_{N-1})$. Using this convention for signed integers we find the following remarkable property. Binary operations of addition (subtraction) or multiplication between two variables represented in the RNS, can be performed by independent operations on the respective residues. i.e., $Z = X \circ Y$ implies $z_i = |x_i \circ y_i|_{m_i}$ where $\circ \in (+, \cdot)$. We have entirely eliminated the inter digit carry required by weighted magnitude systems. The RNS is not without disadvantages however. In particular there is no well defined order to the RNS digits, and this makes sign and relative magnitude determination difficult. In fact, what is easy to perform in a weighted number system is difficult to perform in the RNS, and vice versa.

Fortunately, most digital signal processing structures have an abundance of easy operations as far as the RNS is concerned. Probably the only difficult RNS operation to be performed, is that of scaling to prevent overflow[3].

Two examples of typical digital signal processing structures are shown in Figs 1 and 2. Fig 1 shows a second order canonic recursive section which can be repeated, or multiplexed, to provide a required filter transfer function[4].

Here, 5 multiplications and 4 additions are performed for the price of one scaling operation. In Fig 2, a radix 4 FFT complex calculation is shown[5]. Assuming that we require to scale after every other application of the structure, then we perform 24 multiplications and 44 additions for a total of 8 scaling operations. In terms of each scaling operation we are performing 3 multiplications and 5.5 additions. There are some digital filtering structures that require no scaling, since they use exact arithmetic. These are the many classes of Number Theoretic Transform (NTT) that now exist. Certain speed and hardware advantages can be obtained by implementing NTTs with the RNS.

## LOOK UP TABLE IMPLEMENTATION

If we restrict the moduli set to a maximum modulus size of 32, then the binary operations can be carried out by storing all possible outcomes in a look up table of maximum size 5K bits. Thus, commercially available 8K bit single packages can be used. This is certainly a cost effective solution for multiplication. In terms of addition, fairly straightforward logic circuitry with small capacity ROMs can be used[11], though this consumes more 'PCB real estate' than the single package ROM. One big advantage of the single chip ROM approach is illustrated in Fig 3. Pre and post binary operations with constants can be calculated and stored in the ROM along with the results of the particular binary operation. Table I shows values of M for monotonically descending, relatively prime moduli $m_i \leqslant 32$. Moduli $m_i < 16$ can be implemented in 1K ROMs. For most digital signal processing applications, the number ranges are more than adequate for a system with 5 or more moduli.

The operation of scaling is more cumbersome, though ROM arrays can be used to effect two different scaling operations[3]. The first one is the technique of exact division. We assume that the scaling factor is the product of the first S moduli $K = \prod_{i=0}^{S-1} m_i$ . The scaling operation $Y \simeq X/K$ is performed by the following recursive relationship.

$$|\Phi^{(k+1)}|_{m_i} = \left| |\Phi^{(k)} - \phi_k|_{m_i} \cdot \left|\frac{1}{m_k}\right|_{m_i} \right|_{m_i}$$

where $\Phi^{(0)} = X, \Phi^{(s)} = Y$, and $\phi_k = |\Phi^{(k)}|_{m_k}$ with the bounds $0 \leq k \leq S-1; k \leq i \leq N-1$.

This is referred to as the original scaling algorithm. The second technique is that of summing scaled metric vectors[1], and this is performed as below:

$$|Y'|_{m_n} = \left| \sum_{i=0}^{N-1} \left| \left[ \frac{\left(\prod_{j=S}^{N-1} m_j\right) \left|\frac{X_i}{\hat{m}_i}\right|_{m_i}}{m_i} + \frac{1}{2} \right] \right|_{m_n} \right|_{m_n}$$

This is referred to as the estimate algorithm. In both cases we assume that the scaling factor is formed by multiplying the first S moduli. $[\cdot]$ indicates the integer value of a function. In both algorithms we have to reconstruct the first S residues of the scaled number. This is referred to as base extension[2], and employs a partial conversion to a mixed radix weighted magnitude representation. Figs 4 and 5 show the two scaling algorithms, plus base extension stages, implemented as ROM arrays. The pre calculated tables T and e can be found in reference [3] . The Figs are shown for S=3, N=6.

## HARDWARE FEATURES

One of the advantages of structures using ROM arrays, is the ease of pipelining for high speed throughput. Fig 6 illustrates the approach for the function:

$$|e|_{m_i} = |(a.b) + (c.d)|_{m_i}$$

A ripple latch pulse stores the output of each ROM, this becomes part of the address for the next ROM. The buffers are used to provide delay and power gain. Providing that we can keep the pipeline full, the throughput rate is equal to the inverse access time and latch settling time. A conservative figure for 8K ROMs is in excess of 10 MHz.

As an indication of the hardware requirements for the examples given in the introduction, we consider implementing the following two RNS systems.

i) Digital Filter 2nd Order Canonic Section

$m_i \in \{29,31,25,27,32\}$ S=2 ; $M= 1.16 \times 2^{24}$

$K = 1.76 \times 2^9$. We chose an input data length of $\sim 2^{12}$ and a coefficient range of $\sim 2^9$. Assuming that the filter has a gain (including scaling) of $< 1$, we will never overflow the number range. The hardware requirement for estimate scaling and filter implementation is 33 8K ROMs with latches. This assumes that we already know the coefficients of the filter, and that the section is not to be multiplexed. If this is not the case, we will require an extra 20 ROMs assuming that the multiplier coefficient, A, is included in the scaling array. If we allow the section to be multiplexed in a parallel structure, we can keep the pipeline full (and hence achieve a 10 MHz data rate for the section) by arranging as many parallel stages as there are latch cycles in the section. This particular pipeline problem will only occur in recursive networks.

ii) FFT Radix 4 Complex Calculation

Using a comprehensive error analysis[6] we are able to optimise RNS system designs based on a prescribed normalised RMS error value. The following system yields acceptable errors as far as using the FFT to filter audio signals is concerned. $m_i \in \{31,11,29,16,15,13\}$ S = 2 ; $M = 1.84 \times 2^{24}$ $K = 1.29 \times 2^7$. The input data to the FFT is 11 bits wide, and the 'twiddle factor' coefficients are approximately 7 bits wide. The theoretical maximum normalised error is 1.8% and this has been verified experimentally. Results are very good for wide band filtering of speech or music signals. The ROM requirement is a mixture of 8K, 4K, 2K, and 1K packages since the moduli set spans both 4 bit and 5 bit resolution. The total package count for the radix 4 complex calculation (8 real inputs) and 8 scaling arrays is:

76 x 8K; 32 x 4K; 64 x 2K; 200 x 1K.

The package count seems large, but the performance is fairly impressive. Assuming that we adopt the pipelining principles outlined above, then the computational element accepts 8 real inputs in less than 100 nS. The serial input data rate is therefore in excess of 80 MHz. If we choose an FFT of 1024 complex samples, then we require to multiplex the system through 5 cascaded stages, and so the serial input data rate for a complete FFT calculation is in excess of 15 MHz. It may be necessary to pack the pipeline with extra latch cycles in order to be able to control the multiplexing of data into the computational element more effectively.

## NTT IMPLEMENTATION

The NTT is a transform that is structurally equivalent to the Discrete Fourier Transform, with the exception that it is defined over a finite ring of integers. As such, the computation is exact. The transform domain does not seem to have any practical value (unlike the Fourier Transform domain), however the cyclic convolution property of the DFT is retained in the NTT. The transform pair is defined as:

$$X_m = \left| \sum_{n=0}^{N-1} x_n \alpha^{nm} \right|_M ; \; x_n = \left| N^{-1} \sum_{m=0}^{N-1} X_m \alpha^{-nm} \right|_M$$

$0 \leq m,n \leq N-1$. $\alpha$ is a primitive root of order N in the ring. i.e. $\alpha^N \equiv 1$. A complete description of the implementation of NTTs in finite integer rings can be found in reference [7]. In order to make the NTT an efficient convolution operator within the binary number system, $\alpha$ is normally chosen to be 2 and M chosen to be of the form $2^k \pm 1$. These restrictions can be lifted if we use the RNS. In doing this, we require M to be composite and we have to ensure that the NTT exists in each residue ring modulo $m_i$. The flexibility can be increased by defining the NTT within complex residue rings[8]. General rings of quadratic integers are also currently being investigated. In using ROM arrays for NTT implementation, we require all the moduli to be prime and this, in general, requires some of the moduli to be greater than 32. We can still use 8K ROMs, of course. As an example of a radix 2 complex butterfly, we can use moduli $m_i \in \{31,47,79,97\}$ which gives a dynamic range of over 23 bits and a cyclic convolution length of 32. 170 8K ROMs are required for the implementation. This will process 64 real input data in excess of a 13 MHz serial input data rate.

## PARALLEL MICROPROCESSOR IMPLEMENTATION

Because of the independence between operations on residue digits, the RNS is ideally suited to parallel processing. Rather than store just the results of residue operations (as in the ROM array approach) we can consider using single chip microprocessors to store complete algorithms within each modulus. The residue operations will now be computed instead of storing all possible outcomes. Using 8 bit microprocessors, we can compute modulo $m_i$ addition for $m_i \leqslant 256$. If we choose the $m_i$ to be prime, then primitive roots exist within each ring. A table of indices can be constructed, allowing multiplication to be carried out using modulo $m_i-1$ addition and index look ups. If we restrict the moduli to $m_i \leqslant 127$, then addition tables with built in inverse index look ups can be stored in one page of memory (256 bytes). If we also perform addition operations using table look up, then the total memory requirement for 2 addition tables and one index table is a maximum of 632 bytes. If we choose, for example, an Intel 8048 series chip, then we have 392 bytes left for the algorithm. With 4 chips, we can generate a dynamic range in excess of 27 bits. For scaling operations, an 8 bit I/O port can be bussed between all the chips for inter digit transfers. This will not be a requirement for the basic NTT, but will be necessary if scaling and/or residue to binary conversion are required[9],[10] on the final output.

## CONCLUSIONS

In this paper we have presented a brief summary of the use of the residue number system in implementing digital signal processing functions. The use of ROM arrays in both high speed computational elements, and associated scaling operations, has been discussed. In particular, we have used examples of recursive filter sections, and FFT and NTT computational elements. Alternate implementations, using parallel microprocessors, has also been presented.

## REFERENCES

[1] A. Svoboda, 'Rational numerical system of residual classes', Stroje Na Zpracovani Informaci, pp 1-29, Sbornik V, 1957.

[2] N. S. Szabo and R. I. Tanaka, 'Residue arithmetic and its applications to computer technology', New York, McGraw-Hill 1967.

[3] G. A. Jullien, 'Residue number scaling and other operations using ROM arrays', IEEE Transactions on Computers, Vol C-27,No. 4, pp 325-336, April 1978.

[4] L. R. Rabiner and B. Gold, 'Theory and application of digital signal processing', Englewood Cliffs, N.J., Prentice-Hall, 1975

[5] G. A. Jullien et al, 'Hardware realisation of digital signal processing elements using the residue number system', Proceedings of ICASSP, pp 506-510, May 1977.

[6] B. D. Tseng et al, 'An error analysis of a FFT implementation using the residue number system', Proceedings of ICASSP, pp 800-803, April 1978.

[7] R. C. Agarwal and C. S. Burrus, 'Fast convolution using Fermat number transforms with applications to digital filtering', IEEE Trans. on ASSP, Vol ASSP-26, No. 1 pp 87-97, April 1974.

[8] M. C. Vanwormhoudt, 'Structural properties of complex residue rings applied to number theoretic transforms', IEEE Trans. on ASSP, Vol ASSP-26, No. 1, pp 99-103, Feb. 1978.

[9] H. K. Jenkins and B. J. leon, 'The use of re sidue number systems in the design of finite impulse response digital filters', IEEE Trans. on Circuits and Systems, Vol CAS-24, pp 191-201, April 1977.

[10] A. Baraniecka and G. A. Jullien, 'On decoding techniques for residue number system realisations of digital signal processing hardware', IEEE Trans. on Circuits and Systems, Vol CAS-25, No. 10, October 1978.

[11] D. K. Banerji, 'A novel implementation for addition and subtraction in residue number systems', IEEE Trans. on Computers, Vol C-23, No. 1, pp 106-109, January 1974.

## ACKNOWLEDGEMENTS

The authors wish to acknowledge the financial assistance of a grant from the National Research Council of Canada.

## FIGURES

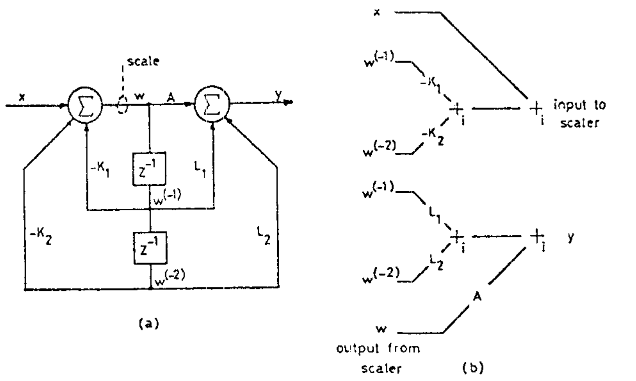

Fig. 1 (a) Canonic section (b) ROM implementation

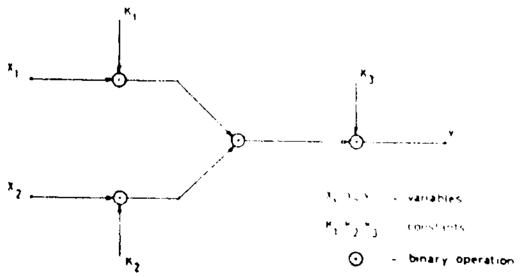

Fig. 3 Binary operations that can be combined into one look-up table.

TABLE 1

| $i$ | $m_i$ | $\prod_{j=0}^{i} m_j$ |
|---|---|---|
| 0 | 32 | $1.00 \times 2^5$ |
| 1 | 31 | $1.94 \times 2^9$ |
| 2 | 29 | $1.76 \times 2^{14}$ |
| 3 | 27 | $1.48 \times 2^{19}$ |
| 4 | 25 | $1.16 \times 2^{24}$ |
| 5 | 23 | $1.66 \times 2^{28}$ |
| 6 | 19 | $1.98 \times 2^{32}$ |
| 7 | 17 | $1.05 \times 2^{37}$ |
| 8 | 13 | $1.71 \times 2^{40}$ |
| 9 | 11 | $1.17 \times 2^{44}$ |
| 10 | 7 | $1.03 \times 2^{47}$ |

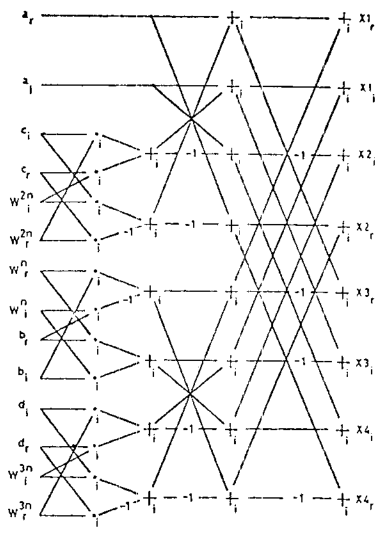

Fig.2 - Radix 4 Complex Calculation

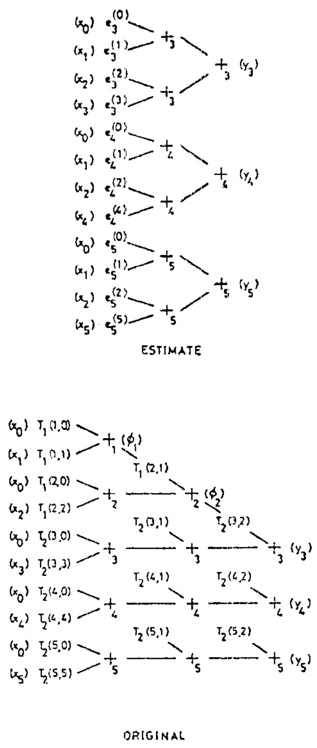

Fig. 4 - Scaling Arrays

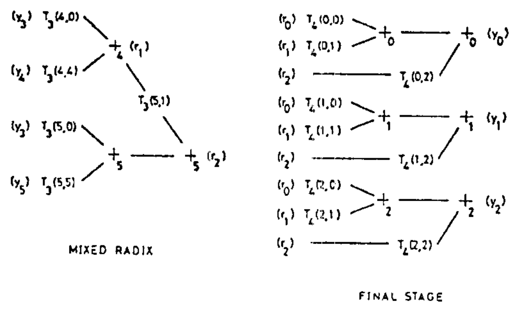

Fig. 5 - Base Extension Array

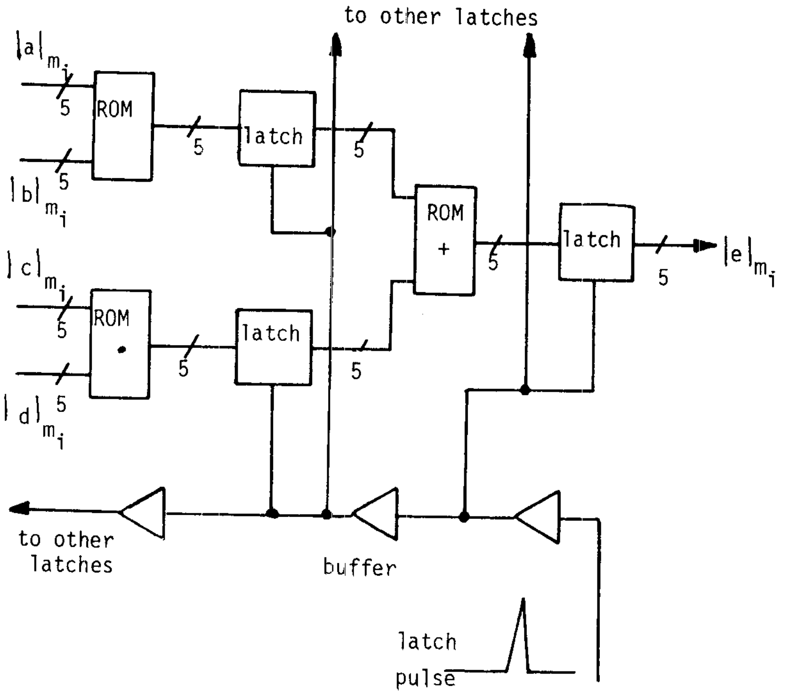

Fig. 6 Pipelining array