A new RNS based DA approach for inner product computation

Vun, Chan Hua; Zhang, Wei; Premkumar, Annamalai Benjamin

2013

Vun, C. H., Premkumar, A. B., & Zhang, W. (2013). A new RNS based DA approach for inner product computation. IEEE transactions on circuits and systems I : Regular papers, 60(8), 2139-2152.

https://hdl.handle.net/10356/98622

https://doi.org/10.1109/TCSI.2013.2239164

A New RNS based DA Approach For Inner Product Computation

C. H. Vun, *Senior Member, IEEE,* A. B. Premkumar, *Senior Member, IEEE,* and W. Zhang, *Member, IEEE*

***Abstract*—This paper presents a novel method to perform inner product computation based on the distributed arithmetic principles. The input data are represented in the residue domain and are encoded using the thermometer code format while the output data are encoded in the one-hot code format. Compared to the conventional distributed arithmetic based system using binary coded format to represent the residues, the proposed system using the thermometer code encoded residues provides a simple means to perform the modular inner products computation due to the absence of the 2^n modulo operation encountered in the conventional binary code encoded system. In addition, the modulo adder used in the proposed system can be implemented using simple shifter based circuit utilizing one-hot code format. As there is no carry propagation involved in the addition using one-hot code, and since the modulo operation can be performed automatically during the addition process, the operating speed of the one-hot code based modulo adder is much superior compared to the conventional binary code based modulo adder. As inner product is used extensively in FIR filter design, SPICE simulation results for an FIR filter implemented using the proposed system is also presented to demonstrate the validity of the proposed scheme.**

***Index Terms*— Distributed arithmetic, inner product, one-hot code, residue number systems, thermometer code.**

I. Introduction

MANY digital signal processing applications make use of functions such as convolution, correlation and filtering, where the inner product computation forms the core of these functions. Distributed Arithmetic (DA) is a well-known technique for calculating inner products without the need for performing multiplication [1]. In addition, compared to the conventional Multiply-Accumulate (MAC) approach, the DA technique allows the computation to be completed in number of execution cycles proportional to the bit-length of the data involved, instead of the number of coefficients in the equation. As such, it provides performance gain when the number of coefficients is more than the bit-length of the data. In DA technique, a look up table (LUT) with bit-serial data addressing is used to perform the product calculation that are then added together to provide the final result.

Residue Number System (RNS) [2] is considered to be suitable for the implementation of high speed digital signal processing due to its parallelism and small data bit-length. RNS allows representation of a big natural number A with a set of smaller natural numbers $\langle a_1,a_2,\ldots,a_M\rangle$ known as the residue digits of the number A. They are derived based on modular arithmetic principles using a selected set of relatively prime numbers $[m_1,m_2,\ldots,m_M]$ called the moduli set. This RNS relationship is expressed as $A \cong \langle a_1,a_2,\ldots,a_M\rangle$.

An important property of RNS is arithmetic operations like addition, subtraction and multiplication of two numbers A and B can be equivalently performed in RNS using their corresponding pair of residue digits a_i and b_i relative to the modulus m_i in an independent and parallel manner, with no carry-propagation occurring between residues of different moduli.

These two properties of RNS, small residue digits size and parallel arithmetic operations are hence ideal for inner product calculation using the DA techniques. The small value residue digits lead to fewer execution cycles due to their short bit-length, while the parallelism enables simultaneous arithmetic operations to be done in multiple independent channels, each reserved for corresponding pair of residue digits belonging to the same modulus.

However, there are a few subtle issues that need to be addressed when implementing the RNS based DA (DA-RNS) technique for inner product calculation in practice. While each datum is now represented in RNS using small residue digits, the residue digits themselves are still encoded in the binary coded (BC) format. As such, there is still overhead due to the localized carry propagation within each channel when performing modular arithmetic operation, albeit to a much smaller degree compared to non-RNS based approach. More importantly, because of the 2^n bit weight associated with the bits in the BC digit, there is a 2^n scaling process during the DA accumulation operation. While this is generally not a problem in non-RNS based system, the 2^n scaling in DA-RNS requires performing a modulo operation with respect to channel modulus and is complicated [3]. Furthermore, the implementation of the modulo adder is also rather convoluted since there is no simple way to perform the modulo operation

Manuscript received June 22, 2012; revised October 30, 2012

C. H. Vun is with the School of Computer Engineering, Nanyang Technological University, Singapore (phone: 65-67904873; fax: 65-67926559; e-mail: aschvun@ntu.edu.sg).

A. B. Premkumar is with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: asannamalai@ntu.edu.sg).

W. Zhang is with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: ZhangWei@ntu.edu.sg).

[2] for BC based residues digits corresponding to generic value of moduli.

As these problems are associated with the BC format used in encoding the residues, we propose to use alternative encoding formats to represent the residues for our DA-RNS implementation. In this work, we propose to use the thermometer code (TC) format to represent the residues and eliminate the complexity involved in performing the 2^n modulo operation. The proposed system also uses modulo adder based on the one-hot code (OHC) format that provides a simple and fast method to perform the modulo-accumulation encountered in DA. This combination provides an elegant solution to practical problems faced in the implementation of conventional BC based DA-RNS systems. As such, it enables benefits in using RNS with the DA to be truly realized in very efficient manner using simple circuit design.

One important practical consideration in using RNS based modular arithmetic is the forward conversion required to first convert a conventional number to its residues. This is likely to be a costly operation and will hinder wide adoption of RNS in real world applications. Our motivation for using the TC encoded residue (TCR) is the availability of a novel RNS based folding ADC [4] where the data generated during the conversion are inherently in the TCR format. As such, there is no extra overhead needed to first convert a datum to its RNS representation. Further arithmetic operations required in signal processing can hence be performed in TCR directly, before it is eventually converted to the conventional binary representation. Fig. 1 illustrates this concept where individual RNS based FIR sub-filter is integrated within each modulus channel of the RNS ADC to perform signal pre-processing before eventually outputting the data in conventional binary number representation. Nevertheless, the proposed scheme can also be used with conventional systems where the binary representation is first converted to the TC based RNS representation during the forward conversion, instead of the usual BC based RNS representation.

The remainder of the paper is organized as follows. Section II provides the background of the DA technique for inner product calculation, and properties of RNS that are advantageous for DA implementation. Section III describes principles used in DA-RNS system based on the conventional BC format, identifying the problem encountered in implementing the concept. Section IV introduces the TC format and describes the TCR based DA-RNS system, showing the simplification achieved by the use of the TCR. Section V discusses modulo adder circuits needed to implement the TCR based DA-RNS, presenting the OHC based modulo adder. Section VI presents the design of an inner product computation unit based on the TCR DA-RNS approach, using the FIR filter application as an example and analyzes its operation. Section VII presents the SPICE based simulation results of the FIR filter implemented using the proposed scheme, demonstrating the validity and practical feasibility of the concept. Section VIII presents a performance evaluation and comparison of the proposed system against the conventional BC based system with Section IX providing the concluding remarks on the new technique proposed in this paper.

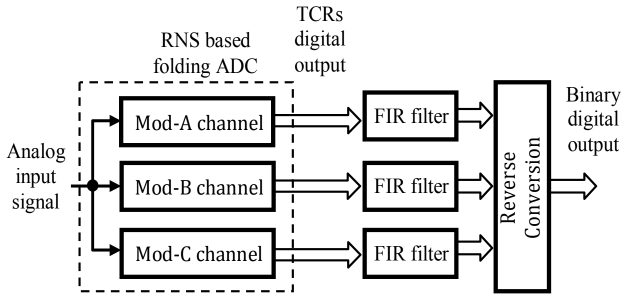

Figure 1. RNS-ADC with pre-processing FIR filter

II. Background

A. Distributed Arithmetic For InnerProduct Computation

Consider the computation of the inner product commonly expressed as follows:

$$y = \sum_{k=0}^{K-1} A_k x_k \tag{1}$$

It is assumed that A_k are fixed values (e.g. filter coefficients of FIR filter) and $x_k = [x_0, x_1, \dots x_{K-1}]$ is the corresponding input vector with K entries, each binary encoded with bit-length of N. Using the normal multiply and accumulate approach, it is obvious that the calculation of this inner product will take K MAC execution cycles, corresponding to the number of coefficients used in (1).

Now consider each entry in x_k to be expressed in terms of its binary encoded bits as follows:

$$x_k = \sum_{n=0}^{N-1} b_{kn} 2^n \qquad b_{kn} \in [0,1] \tag{2}$$

where b_{kn} is the n^{th} bit of x_k and hence has binary value of either 0 or 1, while the 2^n represents the associated weight of the binary bit. The inner product y in (1) can then be written in the form associating it directly with the bit values of the entries in the input vector:

$$y = \sum_{k=0}^{K-1} A_k x_k = \sum_{k=0}^{K-1} A_k \sum_{n=0}^{N-1} b_{kn} 2^n \tag{3}$$

Interchanging the order of the summations and bringing A_k together with the binary bits of x_k:

$$y = \sum_{n=0}^{N-1} \left\{ \sum_{k=0}^{K-1} A_k b_{kn} \right\} 2^n \tag{4}$$

Let $$f(\mathrm{A}_k, b_{kn}) = \sum_{k=0}^{\mathrm{K}-1} \mathrm{A}_k b_{kn} \qquad (5)$$

Hence $$y = \sum_{n=0}^{\mathrm{N}-1} f(\mathrm{A}_k, b_{kn})\ 2^n \qquad (6)$$

The function $f(\mathrm{A}_k, b_{kn})$ contains values representing the sum of products with individual binary bit value b_{kn} of the input vector x_k. Since the b_{kn} bit value can only have value of either 0 or 1, while the value of each A_k is fixed, there are altogether 2^{K} possible combinations values based on (5).

In DA technique, these values are pre-computed and stored in a LUT. The DA's LUT (DALUT) is then successively addressed by using the n^{th} data bit b_{kn} of all x_k entries in parallel, starting with n=0 until n=N-1. The successive outputs are then accumulated as indicated in (6), but with the need to first scaling each output by its individual 2^n factor due to the different weights of the binary bits present in the BC data. Eventually (N-1) accumulated sum corresponds to the value of the inner product y is obtained.

Consider an illustrative example with K=4 inner product computation. Then

$$y = \sum_{k=0}^{3} \mathrm{A}_k x_k$$
$$= \mathrm{A}_0 x_0 + \mathrm{A}_1 x_1 + \mathrm{A}_2 x_2 + \mathrm{A}_3 x_3 \qquad (7)$$

For brevity, further assume that each of the entries in the input vector $x_k = [x_0, x_1, x_2, x_3]$ is of bit-length of N=3 bit as follows:

$$x_0 = \{b_{02}\, b_{01}\, b_{00}\};\ x_1 = \{b_{12}\, b_{11}\, b_{10}\};$$
$$x_2 = \{b_{22}\, b_{21}\, b_{20}\};\ x_3 = \{b_{32}\, b_{31}\, b_{30}\}.$$

The DA based system can then be implemented as shown in Fig. 2, where there will be 2^4=16 entries in the DALUT with values derived based on (6). The relevant LUT entries will then be successively clocked out over three execution cycles using the collective bit pattern of the input vector in a bit-serial manner with combination as follows:

$$t_{cycle} = 0 : b_{k0} = \{b_{00}b_{10}b_{20}b_{30}\}$$
$$t_{cycle} = 1 : b_{k1} = \{b_{01}b_{11}b_{21}b_{31}\}$$
$$t_{cycle} = 2 : b_{k2} = \{b_{02}b_{12}b_{22}b_{32}\}$$

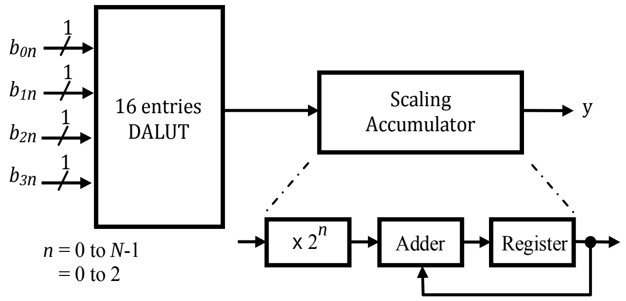

Figure 2: DA system for inner production computation

During each execution cycle, the DALUT output is multiplied by its individual 2^n scaling factor before it is added to the accumulated value obtained in previous execution cycles as indicated in (6). Each execution cycle hence consists of two operations, scaling and accumulation (SAC), which can be performed by the circuit arrangement as illustrated in Fig. 2. In conventional binary number system, the 2^n multiplication is simply done by a logical left shift of the output by the appropriate amount corresponding to its n^{th} value. The adder can be of any binary adder with its output stored in a register for further accumulation with incoming values.

Hence a DA based inner product computation can be completed in N SAC execution cycles, which is equal to 3 in this example. Compared to using the normal MAC approach, where the computation will take K = 4 MAC execution cycles, the DA technique provides a performance gain for inner product computation if N < K, assuming that the execution cycle of one MAC operation consumes same number of clock cycles to one SAC operation. Furthermore, there is no multiplier needed in DA based system to perform the computation which is typically more hardware costly.

B. Residue Number System

RNS is a number system in which an integer A within a legitimate range [0, P) is uniquely defined by a set of positive integers $\langle a_1, a_2, \ldots, a_{\mathrm{M}}\rangle$, which are the remainders (the residues) obtained when A is divided by a set of pair wise relatively prime positive integers $[m_1, m_2, \ldots, m_{\mathrm{M}}]$ (the moduli set), where $\mathrm{P} = \prod_{i=1}^{\mathrm{M}} m_i$. The integer A represented in RNS can then be expressed as $A \cong \langle a_1, a_2, \ldots, a_{\mathrm{M}}\rangle$.

The main feature of the RNS is that an integer within a large dynamic range can be uniquely represented by the set of residues that are of much smaller values, corresponding to the size of the moduli set used in the representation. For instance, using the [7,8,9] moduli set which provides a legitimate range of [0,504), the integer A = 179 can be represented by $\langle 4,3,8\rangle_{7,8,9}$ residue set and the integer B = 254 by $\langle 2,6,2\rangle_{7,8,9}$ residue set.

Furthermore, arithmetic operations between the two integers A and B can also be equivalently performed in RNS based modular arithmetic, using their corresponding residue digits as follows:

$$179\ \Theta\ 254 \ \cong\ \langle 4,3,8\rangle\ \Theta\ \langle 2,6,2\rangle$$
$$= \langle 4\Theta 2,\ 3\Theta 6,\ 8\Theta 2\rangle$$

where $\Theta \epsilon \{+, -, \times\}$. For example:

$$254+179 \ \cong\ \langle 4,3,8\rangle_{7,8,9} + \langle 2,6,2\rangle_{7,8,9}$$
$$= \langle 4+2,\ 3+6,\ 8+2\rangle_{7,8,9}$$
$$= \langle 6,9,10\rangle_{7,8,9}$$
$$= \langle 6,1,1\rangle_{7,8,9}$$

Another advantageous feature of the RNS system is that the operation between each pair of residue digits can be performed in a parallel and in independent manner from the other pairs of residues, as long as the resultant value does not exceed the legitimate range provided by the moduli set used.

Shorter bit-lengths of residues and inherent parallelism suggest that RNS combined with DA would be ideal for inner products computation. The performance gain is dependent on the data bit-length and this is examined next.

III. Binary code based DA-RNS

BC encoded DA-RNS system has been reported in [3], [5] and [6]. However, the number of publications is fewer than what one would normally expect for such seemingly well matched DA and-RNS techniques. The main reason most likely lies in the difficulty of implementing the modulo operation associated with the 2^n scaling factor that originates from the BC encoded residue (BCR). The following is the derivation needed to implement the inner product computation using the BCR based DA-RNS technique which reveals the source of the complication.

Applying RNS principles using $[m_1, m_2, \ldots, m_M]$ moduli for the inner product in (4), it can be rewritten in terms of its residues $y \cong \langle y_1, y_2, .., y_i, .. y_M \rangle$ where

$$y_i = \left| \sum_{n=0}^{N-1} \left\{ \sum_{k=0}^{K-1} A_k\, b_{kn} \right\} 2^n \right|_{m_i}$$

$$= \left| \sum_{n=0}^{N-1} f(A_k, b_{kn})\, 2^n \right|_{m_i}$$

As before, the 2^n factor needs to be decoupled from $f(A_k, b_{kn})$ that is to be stored in the DALUT. This is done by applying the algebra of RNS as follows:

$$y_i = \left| \sum_{n=0}^{N-1} |f(A_k, b_{kn})|_{m_i} \, |2^n|_{m_i} \right|_{m_i} \quad (8)$$

Let $\quad f_{m_i}(A_k, b_{kn}) = |f(A_k, b_{kn})|_{m_i}$

$$= \left| \sum_{k=0}^{K-1} A_k b_{kn} \right|_{m_i} \quad (9)$$

(8) then becomes

$$y_i = \left| \sum_{n=0}^{N-1} f_{m_i}(A_k, b_{kn}) \;\; |2^n|_{m_i} \right|_{m_i} \quad (10)$$

Hence values of $f_{m_i}(A_k, b_{kn})$ can be pre-computed and stored in the DALUT, which can be subsequently clocked out by using the bit-serial stream of the input vector for the accumulation operation as described before. However each of the value needs to be first scaled with the $|2^n|_{m_i}$ factor, which is difficult to be implemented in hardware due to its modulo operation with respect to m_i. An example of a hardware that is needed to perform this modulo accumulation is shown in Fig. 3 [3], illustrating the complications faced in implementation that limits the minimum clock period (i.e. highest clock frequency) achievable in its operation. The difficulty explained in Fig 3 is overcome in TCR based DA-RNS operation, as will be presented next.

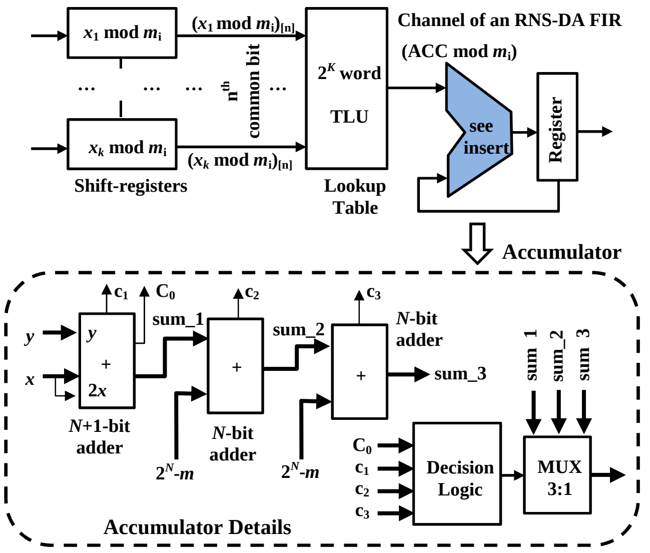

Figure 3: BCR based DA-RNS system for inner product computation (for one of the mod-m_i residue channel) (Adapt from [3] 1999©IEEE)

IV. Thermometer code based DA-RNS

This section first introduces the thermometer code format. An expression is then derived for implementing the DA-RNS based on the TC encoded residue. It is then compared with the implementation presented in section III.

A. *Thermometer Code Format*

Thermometer code refers to the encoding format where the value of a datum is equal to the number of binary '1' bits used to represent the datum. For example, an integer of value of 5 can be encoded using the bit pattern {11111} in TC format. In practice, binary '0's are also added to explicitly indicate the dynamic range (DR) associated with the integer. Hence the value of 5 for an integer with dynamic range of 10 will be encoded as {0000011111}. Evidently, such a format is normally not efficient as the number of bits required to represent typical data (e.g. 8-bit resolution with 0-255 dynamic range) will be excessive.

But using TC format in RNS context is feasible since the DR of each residue is bounded by its modulus. For example, with [7,8,9] moduli set, the bit-length of the corresponding TC encoded residues will be 6, 7 and 8 respectively. These are hence of similar bit-length compared to the binary number that the moduli set can represent (i.e. P = 504). But its advantage lies in a simpler and efficient modulo addition when used in conjunction with its variant, the one-hot code encoded format as shall be demonstrated. More importantly, TCR provides a very simple means to overcome the complication that arises due to the $|2^n|_{m_i}$ factor encountered in DA-RNS based on the BC format. Nevertheless the key to using TC format efficiently in RNS is to minimize the bit-lengths by using a larger number of small moduli rather than few big moduli. For example, it would be more computationally efficient to use a [5,7,8,9] moduli set rather than a [15,16,17] moduli set to cover the range for a 11-bit BC system. As such, the TCR based representation would be particularly suitable for small and medium size dynamic range type of applications.

Mathematically, TC encoded number system is a unary system which is equivalent to a base-1 binary bit system. It is also common to describe it as a no place-value number system, since the positions of its '1' bits in the bit pattern are not important. For our purpose, we will assign a bit weight of 2^0 to all the bits in a TCR, and denote it as an equal place-value number system. The decimal equivalent of the residue represented by TC is hence given by:

$$|x_k|_{m_i} = \sum_{n=1}^{m_i-1} t_{kn}\, 2^0 \qquad t_{kn} \in [0,1] \tag{11}$$

There are some very convenient features associated with TCR based modular arithmetic [7]. Modular addition of two TCRs can be done by concatenating the two TCRs using shifter circuits while the modulo operation can be done by checking a single bit. Additive inverse of a TCR is simply the 1's complement of its value. There is also no ambiguity in taking the additive inverse of value 0, as its 1's complement is equal to its modulus value which reverts to 0 after its modulo operation.

B. *TC based DA-RNS Inner Product*

The derivation of TCR based DA-RNS system for inner product computation follows in the same manner as in the BC based system. The only difference lies in using the TCR's bit expression of (11) in place of the binary code version shown in (2).

$$x_k = \sum_{n=0}^{N-1} b_{kn}\, 2^n \quad \Leftrightarrow \quad |x_k|_{m_i} = \sum_{n=1}^{m_i-1} t_{kn}\, 2^0$$

Hence the residue expression for the TCR based DA-RNS system can be obtained by simply replacing the symbols used in (10) with the TCR equivalents, namely, number of TCR's bit is equal to m_i-1, and all bits are of equal weight, $2^0 = 1$.

$$y_i = \left| \sum_{n=1}^{m_i-1} f_{m_i}(\mathrm{A}_k, t_{kn})\ |2^0|_{m_i} \right|_{m_i}$$

$$= \left| \sum_{n=1}^{m_i-1} f_{m_i}(\mathrm{A}_k, t_{kn}) \right|_{m_i} \tag{12}$$

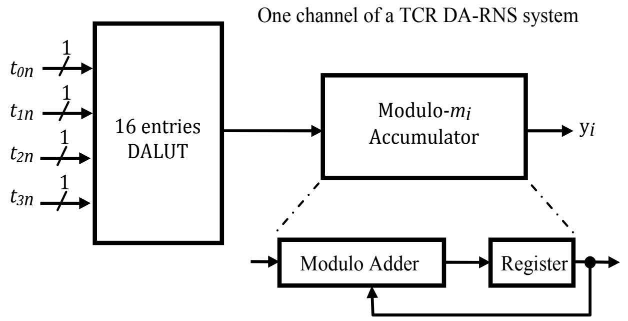

Figure 4: TCR based DA-RNS system (one residue channel)

since $|2^0|_{m_i} = 1$. The expression of $f_{m_i}(\mathrm{A}_k, t_{kn})$ based on (9) is hence:

$$f_{m_i}(\mathrm{A}_k, t_{kn}) = \left| \sum_{k=0}^{\mathrm{K}-1} \mathrm{A}_k t_{kn} \right|_{m_i} \tag{13}$$

(13) describes how entries can be stored in the DALUT. One can also note the absence of the $|2^n|_{m_i}$ scaling which was the cause for complicating earlier BC based implementation.

However, (13) indicates a potential problem in the implementation. As the modulo operation is to be done after the summation, the accumulator is likely to overflow during the accumulation process. This is resolved by expanding (13) using the algebra of residues as shown below where the modulo operation is done after every addition:

$$y_i = \left| \sum_{n=1}^{m_i-1} f_{m_i}(\mathrm{A}_k, t_{kn}) \right|_{m_i}$$

$$= \left| f_{m_i}(\mathrm{A}_k, t_{k0}) + f_{m_i}(\mathrm{A}_k, t_{k1}) + f_{m_i}(\mathrm{A}_k, t_{k2}) + \cdots + f_{m_i}(\mathrm{A}_k, t_{kn}) \right|_{m_i}$$

$$= \left| \left| f_{m_i}(\mathrm{A}_k, t_{k0}) + f_{m_i}(\mathrm{A}_k, t_{k1}) \right|_{m_i} + f_{m_i}(\mathrm{A}_k, t_{k2}) + \cdots + f_{m_i}(\mathrm{A}_k, t_{kn}) \right|_{m_i}$$

Hence a TCR based system can be implemented very simply by using m_{M} channels of modulo adders instead of the equivalent system of Fig. 2 as illustrated in Fig. 4 that shows one channel of the TCR based DA-RNS system.

Without the modulo scaling operation for the TCR based DA-RNS system, each execution cycle only needs to perform the accumulation function, and would require fewer clock cycles compared to the SAC operations. Hence the trade-off for the longer bit-length of TCR is in the simplicity required of the modulo accumulator, the design of which would be much simpler compared to the one shown in Fig. 3. Furthermore, overall performance gain can be obtained if a very efficient implementation of the modulo accumulator can be found such that the system can operate at faster clock rate as well as at higher bit-at-a-time (BAAT) rate [1]. This issue will be discussed in the next section.

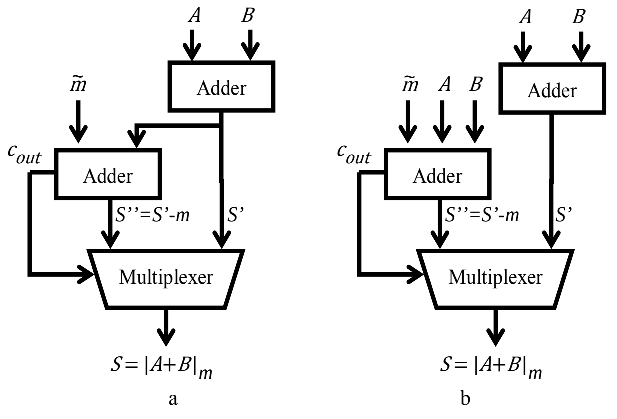

Figure 5: Modulo-m adders for BC residues (BCR)

V. Modular accumulator design

This section describes a few alternatives to implement the modulo adders and then presents a design for the proposed system.

A. BC Based Modulo Adder

For BC based modular arithmetic, the general approach to implement modulo adder for generic modulus is to make use of the architecture as shown in Fig. 5 [2] [8], although there are other possible variations that are based on specific hardware approach [3] and those using multiple adders [9] [10] designs when specific moduli sets are considered. In Fig. 5, two binary adders are used to implement the following modular addition:

$$|A+B|_m = \begin{cases} A+B & if\ A+B<m \\ A+B-m & otherwise \end{cases}$$

Circuit Fig. 5a uses one binary adder to perform the addition of the two operands, A and B to first provide an intermediate sum S'. The modulus m is then subtracted from intermediate sum S', which can be performed through addition with the two's complement of m, $\tilde{m}$ as shown. The carry-out bit from the subtraction is then used to control the output multiplexer, which determines whether the output should be $S' = A+B$ or $S'' = A+B-m$, in effect performing the modulo operation. A variation of the circuit is shown in Fig. 5b where a three operands adder is used to calculate the S'' in parallel, to achieve faster operation at the expense of more complicated binary adder design.

Hence, although there is no carry propagation between the residue channels of different moduli, there is still localized carry propagation occurring within the residue channel since the residues themselves are encoded in BC and hence operate based on the binary adders. Furthermore, the modulo adders of Fig. 5 needs the carry bit of the second binary adder in order to generate the final result. As such, the performance of these modulo adder depends very much on the carry propagation performance of the underlying binary adders used in the implementation. For example, the binary adders of Fig. 5 can be based on the ripple carry full adder which is slow but uses a simple logic structure, or variation of the carry-look ahead full adder which is fast but at much higher logic gates cost.

As the carry bit propagation originates from the BC format used in encoding the residues, we propose an approach similar to earlier TCR to tackle this inefficiency by using an alternative encoding format, the one-hot code, to be used for the modulo adder. The OHC based modulo adder can be implemented using shifter based circuit to perform the addition operation without the carry propagation property. Furthermore, the modulo operation can be performed in a very simple manner compared to the BC version. As such, it is used in conjunction with the proposed TCR base DA-RNS described earlier.

B. One-Hot Code Based Modulo Adder

A one-hot code consists of n bits, but can only have 1 bit asserted at any one time. Hence it is also known as 1-out-of-n encoding scheme. One-hot code is normally used for decoding the address bits for LUT purpose. When it is used to represent the residue in RNS, it is referred to as the one-hot residue (OHR) [11].

In the one-hot code format, the value of the residue corresponds directly to the asserted bit position. Compared to the TCR, the OHR uses one extra bit in order to encode the value corresponding to 0. For example, a residue digit with value of 5 in a modulus-7 system would be encoded as {0100000}, while a value of 0 would be encoded as {0000001}.

While the value of an OHR is intuitively clear from its bit pattern, it lacks formal mathematical properties (e.g. base-1, base-2) and poses difficulty in using it for general mathematical purposes. Nevertheless, its unique usefulness for representing residues lies in the automatic execution of the

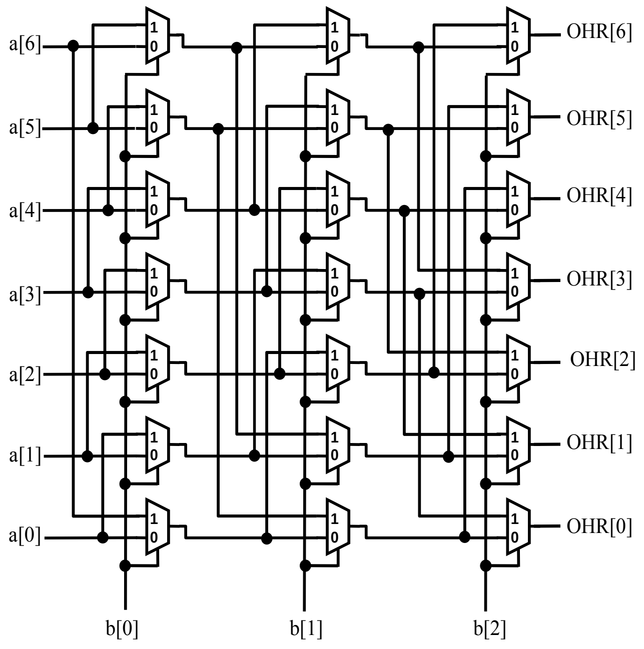

Figure 6: OHR based modulo-7 adder

modulo operation when performing addition and subtraction based on circular shifting technique.

Consider two modulus-7 residues r_1 and r_2 which have numerical value of 4 and 5 respectively expressed in OHR format:

r_1 = 0010000
r_2 = 0100000

The modular sum of these two operands can be found by executing a circular shift operation to one of the operands, based on the value of the other operand. In this example, r_1 will be circular shifted five positions (based on the value of r_2) to its left (assuming its highest value bit position is on the left) which will make its '1' bit wrap around to the lowest value bit position such that r_1 eventually becomes $\{0000100\}$. This implies a numerical value of 2, which is consistent with $|4+5|_7 = 2$. This unique property exhibits a very efficient way to perform automatic modulo operation during the addition.

In our proposed system, a hybrid design is used for the OHR based modulo adder. Fig. 6 shows the circuit schematic of a modulo-7 adder used in the modular accumulator based on this design. The adder consists of multiplexers arranged to form a log based circular shifter. The input a[n] and the output OHR[n] are both OHR encoded while the b[n] input is BC encoded. The BC encoded b[n] controls the amount of circular shift that is applied to a[n]. This effectively executes $|A+B|_7$, with the modulo 7 operation automatically performed as the OHR bits wrap around. The OHC format at the output is also convenient if it is to be used to address a LUT such as a residue-to-binary encoder to present the output in the conventional binary representation.

Compared to the BCR modulo adder, this shifter based OHR modulo adder implementation is much faster as there is no logic gate delay involved in the operation, nor does it have carry propagation issues. The operating speed is solely determined by the delay of the signal passing through the multiplexers. In addition, the number of transistors used to implement the shifter based modulo adder is lower when compared to that in BCR modulo adder based on the ripple carry adders, which is the most area efficient (but slowest) implementation for a binary adder. This will be discussed in detail later in Section VIII.

C. TCR DA-RNS Implementation

Fig. 7 shows how the OHR modulo adder used in the proposed TCR based DA-RNS system. The OHR datum value in the output register, which supplies input A to the OHR modulo adder, is initially set to 0 at the beginning of each accumulation execution cycle. The other input B to the OHR modulo adder is the value output from the DALUT which is encoded in the BC format. Based on the value of the input B, the OHR modulo adder will perform circular shift to the OHR value applied at input A, hence performing the modulo accumulation function as described earlier.

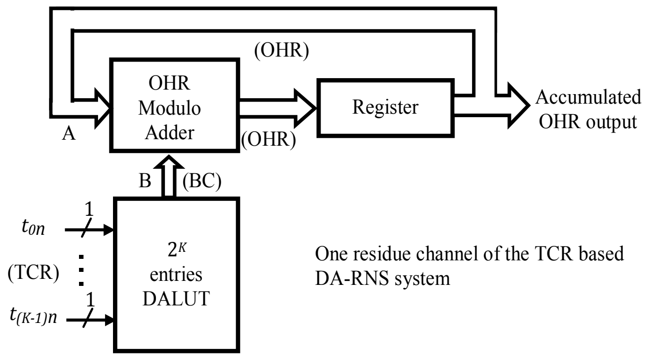

Figure 7: TCR based DA-RNS with OHR modulo accumulator

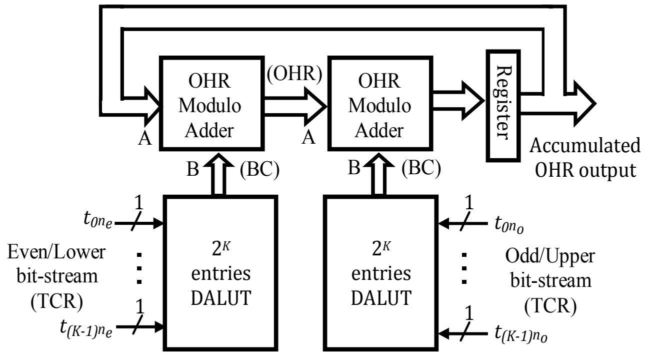

Figure 8: 2BAAT TCR based DA-RNS system

Due to the simplicity of the system (no carry bit propagation or modulo 2^n scaling), it is hence possible to execute the system at higher clock rate compared to the BC DA and the BCR DA-RNS. Because of the same unit bit weight assigned to all bits in the TCR, the design can also be easily extended to operate at 2-bit-at-a-time (2BAAT) [1] or higher to further compensate for the longer bit-length of the TCR. Fig. 8 shows a 2BAAT design, where two OHR modulo adders are simply connected in cascade to sum two DALUT outputs, each driven by one group of bit-serial stream allocated from the TCR input. Since the order of addition is not important, the two groups of bit-serial streams used can be allocated based on the even bits and odd bits of the TCR datum, or the lower half and upper half of the t_n bits of the TCR datum if this is more hardware amenable.

VI. FIR FILTER IMPLEMENATATION

Design and performance analysis of a DA-RNS based FIR digital filter using proposed TCR is presented in this section to demonstrate the practicality of the proposed concept. A FIR low pass filter output $y[n]$ is related to its input signal x[n] through the filter coefficients A_k as follows:

$$y[n] = \sum_{k=0}^{K-1} A_k\, x[n-k]$$

Its operation consists of multiple inner product computations as a series of input data are made available to the filter.

A. FIR Filter Characteristics

A 4th order FIR low pass filter, designed using the Parks-McClellan algorithm with coefficients as shown below is used for this illustrative, but elementary example.

$$y[n] = 3x[n] + 11x[n-1] + 15x[n-2] \\ +11x[n-3] + 3x[n-4] \quad (14)$$

The frequency response of this FIR filter generated using the Scilab simulator is shown in Fig. 9. The corner frequency is chosen to be about 0.1 of the filter operating frequency f_S, with maximum attenuation of -55dB below the pass band occurring at $0.35f_S$.To clearly demonstrate the operating principles of this filter using the proposed method, an input data sequence is generated which contains a signal component with frequency at about $0.06f_S$, i.e. within the pass band of the filter, together with another component having frequency located at about $0.35f_S$. In addition, to simplify the numerical conversion between input data binary number and its RNS representation, the data sequence is assumed to consist of integer values. The signal amplitude is also intentionally kept within bounds such that the resultant output DR can be adequately covered using [5,7,8] moduli set for this numerical example. Based on these approaches, an input data sequence with 51 points is generated with values as follows:

x[n] = { *1, 1, 2, 3, 3, 5, 5, 5, 6, 5, 5, 5, 3, 4, 2, 1,2, 0, 0, 1, 0, 2, 2, 2, 5, 4, 5, 6, 5, 6, 5, 4, 4, 2, 2, 2, 0, 1, 0, 0, 2, 1, 2, 4, 3, 5, 6, 5, 6, 5* } (15)

Using the Scilab simulator, the input data sequence is then applied to the 4th order FIR filter and the resultant output y[n] is obtained as follows:

y[n] = { *3, 14, 32, 57, 86, 118, 154, 187, 212, 226, 230, 226, 212, 190, 165, 133, 100, 71, 47, 28, 17, 21, 39, 61, 89, 125, 162, 194, 215, 230, 237, 230, 212, 183, 147, 114, 86, 61, 39, 21, 17, 28, 47, 71, 100, 133, 168, 201, 227, 237, 233* } (16)

The time domain response of the FIR filter is also generated with these data using the Scilab simulator for visual confirmation of its filtering effect and its operation as intended. The simulated time domain input and output waveforms are shown in Fig. 10.

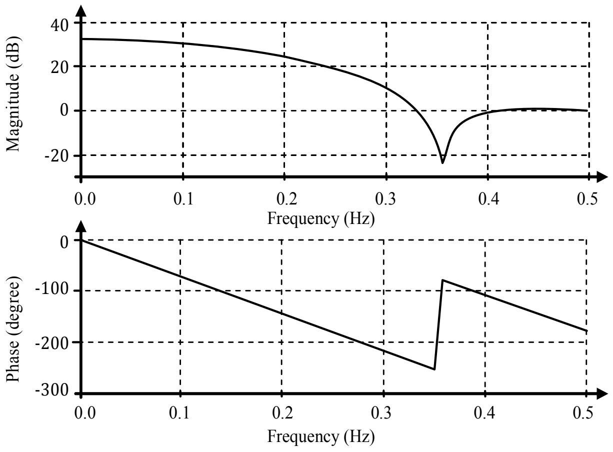

Figure 9: FIR filter frequency response

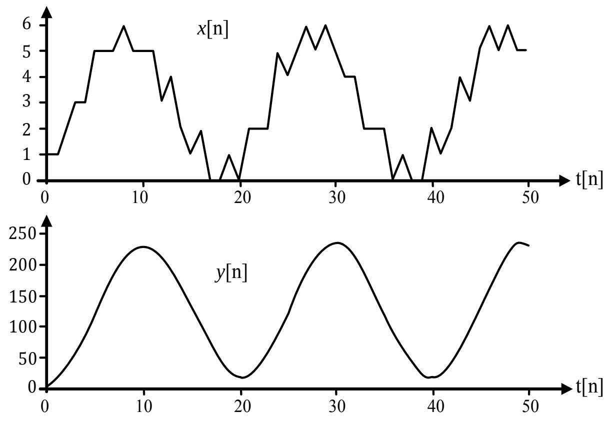

Figure 10: FIR filter input and output waveform

The input x[n] shows significant amount of irregularity due to the high frequency components as well as the quantization effect of rounding the input to integers. The output response shows that the FIR filter is performing an adequate job of filtering the input as intended, sufficient for the illustrative purpose here.

B. TCR Based DA-RNS FIR Filter Design

The FIR filter designed is next translated to the RNS representation. A [5,7,8] moduli set that provides a DR of [0,280) is chosen for the implementation, sufficient to accommodate for the maximum output value observed in (16). Fig. 11 shows the DA-RNS system block diagram based on this design. Three residue channels, one for each modulus with design based on Fig. 7 or Fig. 8, are required to implement these DA-RNS based FIR filters.

The corresponding DALUT' entries for the three different moduli channels are next derived using the expression (13) with the various filter coefficient A_k values shown in (14) as follows:

$$f_{m_i}(A_k, t_{kn}) = \left| \sum_{k=0}^{4} A_k t_{kn} \right|_{m_i} \quad (17)$$

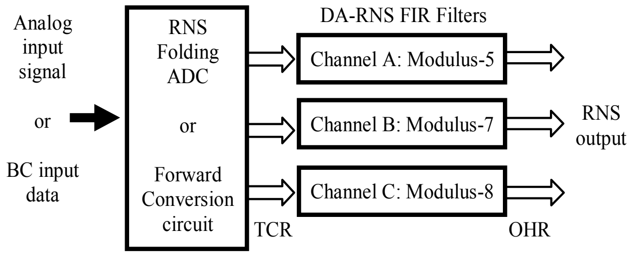

Figure 11: 3-channel DA-RNS FIR filter

Table 1: FIR filter DALUTs contents

Input Code					DALUT entries ($A_0 = A_4 = 3$; $A_1 = A_3 = 11$; $A_2 = 15$)			
t_{1n}	t_{2n}	t_{3n}	t_{4n}	t_{5n}	Entry value	m=5	m=7	m=8
0	0	0	0	0	0	0	0	0
0	0	0	0	1	$A_4 = 3$	$\lvert 3\rvert_5 = 3$	$\lvert 3\rvert_7 = 3$	$\lvert 3\rvert_8 = 3$
0	0	0	1	0	$A_3 = 11$	$\lvert 11\rvert_5 = 1$	$\lvert 11\rvert_7 = 4$	$\lvert 11\rvert_8 = 3$
0	0	0	1	1	$A_3+A_4 = 14$	$\lvert 14\rvert_5 = 4$	$\lvert 14\rvert_7 = 0$	$\lvert 14\rvert_8 = 6$
0	0	1	0	0	$A_2 = 15$	$\lvert 15\rvert_5 = 0$	$\lvert 15\rvert_7 = 1$	$\lvert 15\rvert_8 = 7$
0	0	1	0	1	$A_2+A_4 = 18$	$\lvert 18\rvert_5 = 3$	$\lvert 18\rvert_7 = 4$	$\lvert 18\rvert_8 = 2$
0	0	1	1	0	$A_2+A_3 = 26$	$\lvert 26\rvert_5 = 1$	$\lvert 26\rvert_7 = 5$	$\lvert 26\rvert_8 = 2$
0	0	1	1	1	$A_2+A_3+A_4 = 29$	$\lvert 29\rvert_5 = 4$	$\lvert 29\rvert_7 = 1$	$\lvert 29\rvert_8 = 5$
0	1	0	0	0	$A_1 = 11$	$\lvert 11\rvert_5 = 1$	$\lvert 11\rvert_7 = 4$	$\lvert 11\rvert_8 = 3$
0	1	0	0	1	$A_1+A_4 = 14$	$\lvert 14\rvert_5 = 4$	$\lvert 14\rvert_7 = 0$	$\lvert 14\rvert_8 = 6$
0	1	0	1	0	$A_1+A_3 = 22$	$\lvert 22\rvert_5 = 2$	$\lvert 22\rvert_7 = 1$	$\lvert 22\rvert_8 = 6$
0	1	0	1	1	$A_1+A_3+A_4 = 25$	$\lvert 25\rvert_5 = 0$	$\lvert 25\rvert_7 = 4$	$\lvert 25\rvert_8 = 1$
0	1	1	0	0	$A_1+A_2 = 26$	$\lvert 26\rvert_5 = 1$	$\lvert 26\rvert_7 = 5$	$\lvert 26\rvert_8 = 2$
0	1	1	0	1	$A_1+A_2+A_4 = 29$	$\lvert 29\rvert_5 = 4$	$\lvert 29\rvert_7 = 1$	$\lvert 29\rvert_8 = 5$
0	1	1	1	0	$A_1+A_2+A_3 = 37$	$\lvert 37\rvert_5 = 2$	$\lvert 37\rvert_7 = 2$	$\lvert 37\rvert_8 = 5$
0	1	1	1	1	$A_1+A_2+A_3+A_4 = 40$	$\lvert 40\rvert_5 = 0$	$\lvert 40\rvert_7 = 5$	$\lvert 40\rvert_8 = 0$
1	0	0	0	0	$A_0 = 3$	$\lvert 3\rvert_5 = 3$	$\lvert 3\rvert_7 = 3$	$\lvert 3\rvert_8 = 3$
1	0	0	0	1	$A_0+A_4 = 6$	$\lvert 6\rvert_5 = 1$	$\lvert 6\rvert_7 = 6$	$\lvert 6\rvert_8 = 6$
1	0	0	1	0	$A_0+A_3 = 14$	$\lvert 14\rvert_5 = 4$	$\lvert 14\rvert_7 = 0$	$\lvert 14\rvert_8 = 6$
1	0	0	1	1	$A_0+A_3+A_4 = 17$	$\lvert 17\rvert_5 = 2$	$\lvert 17\rvert_7 = 3$	$\lvert 17\rvert_8 = 1$
1	0	1	0	0	$A_0+A_2 = 18$	$\lvert 18\rvert_5 = 3$	$\lvert 18\rvert_7 = 4$	$\lvert 18\rvert_8 = 3$
1	0	1	0	1	$A_0+A_2+A_4 = 21$	$\lvert 21\rvert_5 = 1$	$\lvert 21\rvert_7 = 0$	$\lvert 21\rvert_8 = 5$
1	0	1	1	0	$A_0+A_2+A_3 = 29$	$\lvert 29\rvert_5 = 4$	$\lvert 29\rvert_7 = 1$	$\lvert 29\rvert_8 = 5$
1	0	1	1	1	$A_0+A_2+A_3+A_4 = 32$	$\lvert 32\rvert_5 = 2$	$\lvert 32\rvert_7 = 4$	$\lvert 32\rvert_8 = 0$
1	1	0	0	0	$A_0+A_1 = 14$	$\lvert 14\rvert_5 = 4$	$\lvert 14\rvert_7 = 0$	$\lvert 14\rvert_8 = 6$
1	1	0	0	1	$A_0+A_1+A_4 = 17$	$\lvert 17\rvert_5 = 2$	$\lvert 17\rvert_7 = 3$	$\lvert 17\rvert_8 = 1$
1	1	0	1	0	$A_0+A_1+A_3 = 25$	$\lvert 25\rvert_5 = 0$	$\lvert 25\rvert_7 = 4$	$\lvert 25\rvert_8 = 1$
1	1	0	1	1	$A_0+A_1+A_3+A_4 = 28$	$\lvert 28\rvert_5 = 3$	$\lvert 28\rvert_7 = 0$	$\lvert 28\rvert_8 = 4$
1	1	1	0	0	$A_0+A_1+A_2 = 29$	$\lvert 29\rvert_5 = 4$	$\lvert 29\rvert_7 = 1$	$\lvert 29\rvert_8 = 5$
1	1	1	0	1	$A_0+A_1+A_2+A_4 = 32$	$\lvert 32\rvert_5 = 2$	$\lvert 32\rvert_7 = 4$	$\lvert 32\rvert_8 = 0$
1	1	1	1	0	$A_0+A_1+A_2+A_3 = 40$	$\lvert 40\rvert_5 = 0$	$\lvert 40\rvert_7 = 5$	$\lvert 40\rvert_8 = 0$
1	1	1	1	1	$A_0+A_1+A_2+A_3+A_4 = 43$	$\lvert 43\rvert_5 = 3$	$\lvert 43\rvert_7 = 1$	$\lvert 43\rvert_8 = 3$

The calculated values for the DALUT table entries are summarized and listed in Table 1. As the FIR filter contains 5 coefficients, there are 32 entries in each of the three DALUTs corresponding to the three moduli [5,7,8].

C. *Numerical Analysis of TCR DA-RNS FIR Filter Response*

Step-by-step calculation of the TCR based DA-RNS filter response is now presented to demonstrate its operation. The input data sequence $x[n]$ of (15) is first translated to the corresponding RNS representations for the [5,7,8] moduli set. For brevity, only the first seven values are shown in Table 2, and are used in the following numerical calculations to validate the response of the filter designed using the TCR based DA-RNS technique. Table 3 shows the corresponding TCR bit patterns of these first seven input data, which are then sent in bit-serial manner to their respective moduli DALUT.

Table 2: RNS representations of x[n]

n (for *x*[*n*])	*Input value*	*Modulo-5*	*Modulo-7*	*Modulo-8*
0	1	1	1	1
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	3	3	3	3
5	5	0	5	5
6	5	0	5	5

Table 3: TCR representations of x[n]

x[*n*]	*Modulo-5*	*Modulo-7*	*Modulo-8*
x[0]	$1 \cong 0001_1$	$1 \cong 000001_1$	$1 \cong 0000001_1$
x[1]	$1 \cong 0001_1$	$1 \cong 000001_1$	$1 \cong 0000001_1$
x[2]	$2 \cong 0011_1$	$2 \cong 000011_1$	$2 \cong 0000011_1$
x[3]	$3 \cong 0111_1$	$3 \cong 000111_1$	$3 \cong 0000111_1$
x[4]	$3 \cong 0111_1$	$3 \cong 000111_1$	$3 \cong 0000111_1$
x[5]	$0 \cong 0000_1$	$5 \cong 011111_1$	$5 \cong 0011111_1$
x[6]	$0 \cong 0000_1$	$5 \cong 011111_1$	$5 \cong 0011111_1$

Starting with time instance n=0, the first group of data are x[0], x[-1], x[-2], x[-3] and x[-4]. At n=1, the second group of data sent are x[1], x[0], x[-1], x[-2], and x[-3]. The data will then progressively advance to subsequent x[n] with its previous four data. In causal system, data prior to x[0] will be considered to be 0. Hence in this case, the response of the FIR filter will reach steady state at time instance n=4. The following shows details of the operation for the three channels, A, B and C corresponding to the three moduli.

i) Channel A for Modulus-5

Table 4 shows the sequence of data involved in the operation of the TCR based DA-RNS system. The x[n] data are sent 1BAAT in bit-serial manner to access the DALUT associated with modulus 5, whose entries are shown in Table 1. The resultant DALUT outputs are indicated under the **DALUT$_{m=5}$** column for each row of bit-serial stream received. For each time instance n, four summation values are provided and are modulo-5 accumulated over four execution cycles as shown under the **Mod-5 Acc** column in Table 4.

The output appears at the 4th execution cycle (i.e. at t_{cycle}=3) of each time instance n and is hence the inner product derived from residues of the input signal entries

Table 4: FIR filter output – Modulus-5 Channel A

n	t_{cycle}	*x*[*n*]	*x*[*n*-1]	*x*[*n*-2]	*x*[*n*-3]	*x*[*n*-4]	DALUT$_{m=5}$	Mod-5 Acc
0	0	1	0	0	0	0	3	3
	1	0	0	0	0	0	0	3
	2	0	0	0	0	0	0	3
	3	0	0	0	0	0	0	3
1	0	1	1	0	0	0	4	4
	1	0	0	0	0	0	0	4
	2	0	0	0	0	0	0	4
	3	0	0	0	0	0	0	4
2	0	1	1	1	0	0	4	4
	1	1	0	0	0	0	3	$\lvert 7\rvert_5 = 2$
	2	0	0	0	0	0	0	2
	3	0	0	0	0	0	0	2
3	0	1	1	1	1	0	0	0
	1	1	1	0	0	0	4	4
	2	1	0	0	0	0	3	$\lvert 7\rvert_5 = 2$
	3	0	0	0	0	0	0	2
4	0	1	1	1	1	1	3	3
	1	1	1	1	0	0	4	$\lvert 7\rvert_5 = 2$
	2	1	1	0	0	0	4	$\lvert 6\rvert_5 = 1$
	3	0	0	0	0	0	0	1
5	0	0	1	1	1	1	0	0
	1	0	1	1	1	0	2	2
	2	0	1	1	0	0	1	3
	3	0	0	0	0	0	0	3
6	0	0	0	1	1	1	4	4
	1	0	0	1	1	1	4	$\lvert 8\rvert_5 = 3$
	2	0	0	1	1	0	1	4
	3	0	0	0	0	0	0	4

and the filter coefficients for time instance *n*. From Table 4, the filter output from modulus-5 channel output for n = 0 to 6 are as follows:

$$y_5[n] = \{3, 4, 2, 2, 1, 3, 4\} \tag{18}$$

ii) Channel B for Modulus-7
Similar steps can be used to derive the modulus-7 channel B response and its output for n = 0 to 6 can be shown to be:

$$y_7[n] = \{3, 0, 4, 1, 2, 6, 0\} \tag{19}$$

iii) Channel C for Modulus 8
In this case the TCR bit-length is 7 bits, the resultant DALUT outputs are obtained at interval of every 7th execution cycle (indicated as t_{cycle}=6) as shown under the **Mod-8 Acc** column in Table 5. The corresponding output for n = 0 to 6 are as follows:

$$y_8[n] = \{3, 6, 0, 1, 6, 6, 2\} \tag{20}$$

Table 5: FIR filter output – Modulus-8 Channel C

n	**t_{cycle}**	***x*[*n*]**	***x*[*n*-1]**	***x*[*n*-2]**	***x*[*n*-3]**	***x*[*n*-4]**	**$\text{DALUT}_{m=8}$**	**Mod-8 Acc**
0	0	1	0	0	0	0	3	3
	1	0	0	0	0	0	0	3
	2	0	0	0	0	0	0	3
	3	0	0	0	0	0	0	3
	4	0	0	0	0	0	0	3
	5	0	0	0	0	0	0	3
	6	0	0	0	0	0	0	3
1	0	1	1	0	0	0	6	6
	1	0	0	0	0	0	0	6
	2	0	0	0	0	0	0	6
	3	0	0	0	0	0	0	6
	4	0	0	0	0	0	0	6
	5	0	0	0	0	0	0	6
	6	0	0	0	0	0	0	6
2	0	1	1	1	0	0	5	5
	1	1	0	0	0	0	3	$\lvert 8\rvert_8 = 0$
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0
3	0	1	1	1	1	0	0	0
	1	1	1	0	0	0	6	6
	2	1	0	0	0	0	3	$\lvert 9\rvert_8 = 1$
	3	0	0	0	0	0	0	1
	4	0	0	0	0	0	0	1
	5	0	0	0	0	0	0	1
	6	0	0	0	0	0	0	1
4	0	1	1	1	1	1	3	3
	1	1	1	1	0	0	5	$\lvert 8\rvert_8 = 0$
	2	1	1	0	0	0	6	6
	3	0	0	0	0	0	0	6
	4	0	0	0	0	0	0	6
	5	0	0	0	0	0	0	6
	6	0	0	0	0	0	0	6
5	0	1	1	1	1	1	3	3
	1	1	1	1	1	0	0	3
	2	1	1	1	0	0	5	$\lvert 8\rvert_8 = 0$
	3	1	0	0	0	0	3	3
	4	1	0	0	0	0	3	6
	5	0	0	0	0	0	0	6
	6	0	0	0	0	0	0	6
6	0	1	1	1	1	1	3	3
	1	1	1	1	1	1	3	6
	2	1	1	1	1	0	0	6
	3	1	1	0	0	0	6	$\lvert 12\rvert_8 = 4$
	4	1	1	0	0	0	6	$\lvert 10\rvert_8 = 2$
	5	0	0	0	0	0	0	2
	6	0	0	0	0	0	0	2

Consolidating the outputs of all three RNS channels' output from (20), (21) and (22) for n=0 to 6:

$$y_5[\text{n}] = \{3, 4, 2, 2, 1, 3, 4\}$$
$$y_7[\text{n}] = \{3, 0, 4, 1, 2, 6, 0\}$$
$$y_8[\text{n}] = \{3, 6, 0, 1, 6, 6, 2\}$$

The output data sequence of the FIR filter, in RNS representation is hence as follows. For n=0 to 6:

$$y[\text{n}] = \{<3,3,3>, <4,0,6>,<2,4,0>,<2,1,1>, <1,2,6>,<3,6,6>,<4,0,2>\}; \tag{21}$$

The correctness of this RNS based output can be confirmed by performing a reverse conversion using the Chinese Remainder Theorem (CRT) [2], which can be shown to produce the following binary representations:

$$y[0] = <3,3,3> \cong 3$$
$$y[1] = <4,0,6> \cong 14$$
$$y[2] = <2,4,0> \cong 32$$
$$y[3] = <2,1,1> \cong 57$$
$$y[4] = <1,2,6> \cong 86$$
$$y[5] = <3,6,6> \cong 118$$
$$y[6] = <4,0,2> \cong 154$$

These calculated values are exactly the same as the first seven values given in (16), hence confirming the correctness of the proposed approach.

VII. Circuit simulations

To further demonstrate the practical feasibility of the proposed approach, circuit level simulations using LTspice SPICE simulator are next performed to ascertain the output of the circuit.

A. 1BAAT Operation For Modulus-5 Channel A

The block diagram for the 1BAAT design is shown in Fig. 7. The DALUTs are constructed using CMOS circuit, and the implementation of modulus-5 channel A's DALUT is shown in Fig. 12 corresponding to 32 entries shown in Table 1.

The 1BAAT operation of the FIR filter (modulus-5 channel A) is then simulated and the timing diagram is shown in Fig. 13. In the simulation, a signal, **Acc_Rst** is used to reset the content of the accumulator output register to 0 for each time

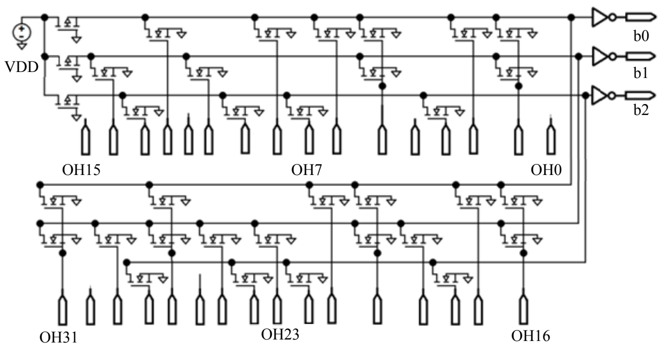

Figure 12: DALUT of modulus-5 Channel A

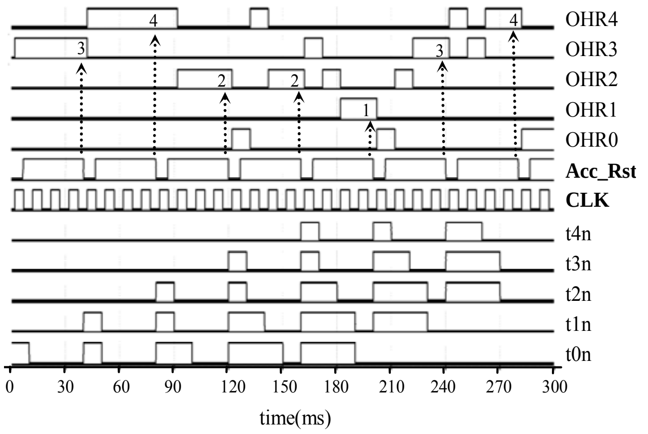

Figure.13: FIR filter 1BAAT timing diagram – Modulus-5 channel A

instance n prior to computing each round of the FIR filter output. Each computation takes 4 execution cycles, which corresponds to 4 clock cycles as indicated by the **CLK** signals in Fig. 13. By using the **Acc_Rst** reset signal as the reference, the output value of the FIR filter computed by the DA-RNS system are captured at its falling edge at various time instances. As shown in Fig. 13, the output in one-hot residue format appears as a sequence {3,4,2,2,1,3,4}, matching exactly what was obtained in (18). Similar 1BAAT design could be used in the modulus-7 Channel B, but would consume 6 execution cycles per computation due to the longer data bit length. An alternative approach is illustrated in the next section.

B. 2BAAT Operation For Modulus-8 Channel C

As the bit-length of the modulus 7 and 8 channels are longer, their 1BAAT operation would take 6 and 7 execution cycles as indicated in Table 5 for modulus-8 channel. To reduce the number of execution cycles due to their longer bit lengths, a 2BAAT design can be used for these two channels using the arrangement as shown previously in Fig. 8. The following presents the design for the modulus-8 channel which can be equally applied to the modulus-7 channel.

Two OHR based modular 8 adders are cascaded as shown in Fig. 14 to enable the 2BAAT operation for modulus 8 channel C. These cascaded adders are then connected in the DA-RNS system as illustrated in Fig. 8.

To further demonstrate the flexibility of the TCR based DA-RNS system, two bit-serial streams are created, one from the input TCR's lower four bits, and the other from the upper three bit padded with one extra '0' bit to balance the two groups. These two groups of data are then sent in parallel to the DA system through two DALUTs. The BC encoded outputs of the two DALUTs are then used to increment the cascaded modulo adders of Fig. 14 that performs the inner product calculation.

Fig. 15 shows the timing diagram captured for the channel C 2BAAT based operation, where each time instance output is completed in four execution (and clock) cycles, the same duration taken by the modulus-5 channel. The output values are captured on the falling edge of the **Acc_Rst** signal. As expected, the output of the system in one-hot residue code format appears as {3,6,0,1,6,6,2} sequence, matching exactly that shown in (20).

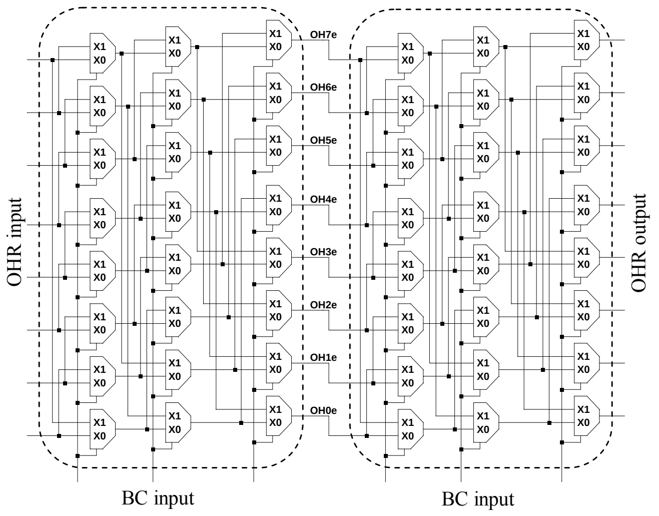

Figure 14: Cascaded modulus-8 adders for 2BAAT operation

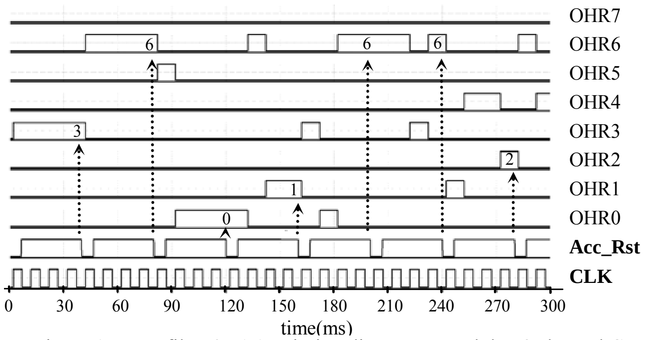

Figure 15: FIR filter 2BAAT timing diagram – Modulus-8 channel C

These simulation results confirm the practical feasibility of the proposed idea. Using a combination of TC, BC and OHC formats for the residues, an efficient means to perform DA-RNS based inner product calculation can be achieved. To compensate for the longer word length of the TCR, higher BAAT rate can be easily used due to the same unit weight of all the TCR's bits.

VIII. Performance Evaluation

It is clear that the key advantage of the TCR based DA operation lies in its simple accumulation operation during the computation of inner products. Compared to the scaling accumulator (Fig. 2) for the binary DA system and the modulo scaling accumulator (Fig. 3) for the BCR DA-RNS system, the TCR based DA-RNS system only needs modulo addition without scaling.

On the other hand, owing to its longer bit lengths, TCR requires more number of execution cycles compared to the BCR DA-RNS system. However, the 2^n and $|2^n|_{m_i}$ factor related to scaling operations needed in binary DA and BCR

DA-RNS would require extra clock cycle(s) for each execution cycle which would then nullify their advantage of shorter bit-lengths. As such, the performance of the TCR-DA system hinges on the efficiency of its modulo adder used in the modulo accumulator. This section compares the performance and complexity of the proposed OHR based modulo adder against the binary adders that are used in the BC based DA-RNS system.

A. Approach

For the modulo adder, we consider popular moduli values between 5 and 15, as these will be more practical in terms of hardware realization and suitable for applications that require small and medium DR (about 2^16 with a combination of moduli within this set). The corresponding BCR based system will then need two binary adders of either 3-bit or 4-bit arranged as shown in Fig. 5a, although it is possible to reduce the latency to one adder using the parallel version of Fig. 5b but at higher hardware cost due to its three input operands.

For the binary adder, two standard representative binary adders are used in the comparison. These are the ripple-carry (RC) adder and the carry-look-ahead (CLA) adder [12], [13]. The RC adder is the most hardware efficient but is the slowest, while the CLA adder is one of the fastest ways to implement binary adder but with tradeoff in hardware circuit complexity. It should be mentioned that there are also other fast binary adders based on parallel prefix operations (e.g. Brent–Kung adder and Kogge–Stone adder), but their speed and logic complexity would be on the same order as the CLA adder for smaller sized moduli used here, and hence are not included. In addition special modulo adders that are optimized for specific classes of moduli value (e.g. 2^n and the likes) are not considered in order to provide a balanced comparison for system based on arbitrary moduli.

Fig. 16(a) shows the logic gate implementation for one bit of the RC full adder [13]. A 3-bit or 4-bit RC adder will use 3 or 4 of this circuit. For a 3-bit CLA adder, the circuit consists of the one shown in Fig. 16(a) for the 1st bit. For the 2nd and the 3rd bits, the portion of the circuit shown within the dotted box in Fig. 16(a) is replaced with those shown in Fig. 16(b) and Fig. 16(c) respectively. For the corresponding OHR modulo adder with modulus value of *m*, the number of multiplexers in the log shifter needed is equal to $m\lfloor \log_2 m \rfloor$, with arrangement as shown in Fig. 6.

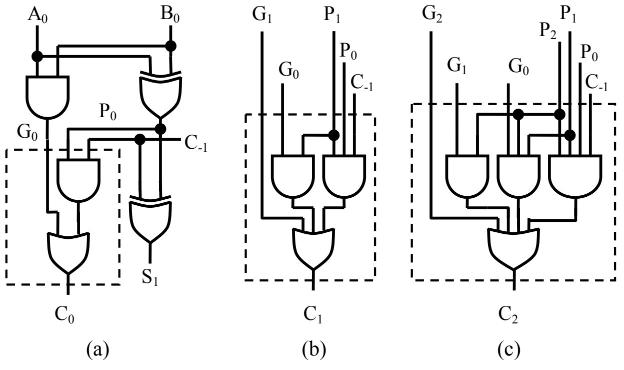

Figure 16. Binary adders (a) Ripple-carry (b) & (c) Carry-look-ahead

Gate count comparison between TC and BC based adders is complicated by the fact that practical circuit realization of the multiplexer is done by using transistor based circuits such as the 4-transistor based CMOS Transmission Gate or the 2-transistor based Pass-Transistor logic. Hence it is more appropriate to compare hardware complexity in terms of transistor count here, although this does not reflect the complexity involved in the wiring of the underlying circuit. In the comparison here, 6 transistors are used for each 2-input XOR logic gate, 4 transistors for all other 2-input logic gates, 2 transistors for each extra input pin and 2 transistors for one NOT gate. For the multiplexer, the 4-transistor based CMOS transmission gate is used as this is a fairly conservative design, with one NOT gate shared among all multiplexers to generate the internal complement shift control signal.

Critical path gate-delay comparison is based on the longest path that a signal propagates through each circuit. For the BC based modulo adder, this is equal to the delay through the two binary adders in order to generate the S'' data for the output multiplexer as shown in Fig. 5a. Each RC adder has a propagation delay equal to (2n+2) due to the carry bit propagation [13]. For the CLA adder, the most optimum implementation will have 6 gate-delays independent of the number of bits [2]. The TCR based modulo adder does not use any combinatorial logic gates in its implementation. Its latency is thus depends solely on the signal propagation delay through the multiplexers.

In order to obtain a more definitive comparison for the signal propagation latency delay, HPSICE simulations based on 45nm technology node are performed for the binary full adders and the log shifter multiplexer based circuit for various moduli considered here. In the simulations, the basic blocks of the OHR modular adder and BCR modular adder, e.g., column of multiplexers and 1-bit adder are first manually created. Based on the layout of their basic blocks, the area and the length of routing wires are estimated, and the RC values of their interconnects are then derived using the PTM model [14]. Transistors used in both designs are also sized (PMOS width = 270nm and NMOS width = 90nm) to give symmetric rise and fall delays as well as sufficient drive. The interconnect parameters together with transistor model cards [14] are then fed to HSPICE for the delay simulation.

B. Results And Discussion

Table 6 summarizes the comparison among the three types of modulo adders in terms of transistor counts (t-cnt) and latency in ps for different moduli. Results in Table 6 show that compared to the BCR RC adder, the OHR based modulo adder has lower order of transistor count for lower moduli, and only slightly higher at the high moduli 13 and 15, but with much superior latency performance throughout. Compared to the BCR CLA adder, the OHR multiplexer based modulo adder is superior in both aspects.

Table 6 . Performance parametric comparison

Mod m	BCR-RC		BCR-CLA		OHR-MUX	
	t-cnt	latency	t-cnt	latency	t-cnt	latency
m=5	158	157 ps	210	115 ps	66	31 ps
m=7	158	157 ps	210	115 ps	90	31 ps
m=8	158	157 ps	210	115 ps	102	31 ps
m=9	210	178 ps	322	115 ps	152	51 ps
m=11	210	178 ps	322	115 ps	184	51ps
m=13	210	178 ps	322	115 ps	216	51 ps
m=15	210	178 ps	322	115 ps	248	51 ps

While these values are broadly indicative of performance, the vast difference in the latency suggests that the TCR-OHR based DA system could potentially operate at much higher clock frequency and hence faster execution cycle, which would largely compensate for its longer bit-length penalty. Taking into consideration the latency due to the $|2^n|_{m_i}$ modulo scaling factor faced in the BCR based system, the proposed system implementation would be of even greater merit, although more detailed analysis of the comparison would need to be performed based on actual implementations. Further, [11] has presented another version of OHR modulo adder based on linear barrel shifter that potentially could further minimize operating power and be implemented with even lower transistor count. However, it would require a bigger size (m^2 versus $m\lfloor\log_2 m\rfloor$) DALUT when used with the TCR based DA system, since its DALUT would need to output the B augend as a full length m-bit one-hot code, compared to the binary code used in our proposed system.

While the proposed OHR-MUX modulo adder is found to be uniformly faster and more area efficient than the high speed BCR-CLA modulo adder for the moduli shown in Table 6, the validity of such an observation toward larger moduli could be further evaluated as follows. First assume that the 20ps propagation delay as observed in Table 6 is incurred for every doubling of the modulus value (i.e. extra column of the multiplexers is added). Direct extrapolation using Table 6 entries would indicate that the propagation delay of the OHR-MUX system will remain superior up to modulus value of $m=2^7=127$, before it reaches the 115ps of a corresponding idealized BCR-CLA based system. Similarly, the transistors count could be shown to be at similar level (about 450) when m=22 for both systems. However, these estimations have omitted the practical implementation complications, in particular the interconnect complexity of the underlying circuitry at large modulus values for both the OHR-MUX and BCR-CLA approaches. As such, concrete answers on their performance can only be obtained by performing detailed circuit layout and measurements for large modulus value**.**

Nevertheless, to exploit the benefits of using RNS principles that reduce the internal circuit complexity of the RNS ADC [4], and to maintain reasonable TCR bit-serial length for the DA operation, we would recommend the use of multiple intermediate size moduli such as [11,13,14,15], instead of a few large moduli to obtain the dynamic range required by the signal processing functions concerned.

IX. Conclusion

This paper presents a novel approach to compute inner products using RNS based DA techniques in which residues are represented by thermometer codes instead of the conventional binary code format. This eliminates the 2^n modulo scaling factor encountered in the binary coded residue version, and results in a very simple accumulator requirement. In addition, use of the one-hot code format for the operation of the modulo adder is proposed for the accumulator. This approach overcomes the carry propagation inefficiencies as well as the complication arising out of repetitive modulo operations that need to be performed in retaining the results of addition in the residue form. As such, it is easier to operate the proposed systems at high clock rates and to scale to parallel bit-serial operations to further enhance the throughput performance.

Design of an DA-RNS based FIR filter is then described and analyzed based on the proposed approach. Its operation is also simulated using SPICE simulator that validates the correctness and confirms the practical feasibility of the proposed scheme. A broad performance comparison against the conventional approach also suggests that there is no penalty incurred in terms of transistor count and latency in the proposed approach.

References

[1] S. A. White, “Applications of distributed arithmetic to digital signal processing: a tutorial review,” *ASSP Magazine, IEEE*, vol. 6, no. 3, pp. 4-19, Jul 1989.

[2] A. Omondi and B. Premkumar, *Residue Number Systems, Theory and Implementation,* Imperial College Press, Singapore, 2007.

[3] A. Garcia, U. Meyer-Base, A. Lloris and F. J. Taylor, “RNS implementation of FIR filters based on distributed arithmetic using field-programmable logic,” in *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, vol. 1, pp. 486-489, Jul 1999.

[4] C. H. Vun and A. B. Premkumar, “RNS encoding based folding ADC,” in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pp. 814-817, 20-23 May 2012.

[5] K. P. Lim and A. B. Premkumar, “A modular approach to the computation of convolution sum using distributed arithmetic principles,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 1, pp. 92-96, Jan 1999.

[6] J. Ramirez, A. Garcia, U. Meyer-Base, F. Taylor, P. G. Fernandez and A. Lloris, “Implementation of RNS-Based Distributed Arithmetic Discrete Wavelet Transform Architectures Using Field-Programmable Logic,” *VLSI Signal Processing*, vol. 33, no. 1-2, pp.171-190, 2003.

[7] C. H. Vun and B. Premkumar, “Thermometer Code Based Modular Arithmetic” in *Engineering and Technology (S-CET), 2012 Spring Congress on*, pp. 534-538, 27-30 May 2012.

[8] S. Pontarelli, G. C. Cardarilli, M. Re and A. Salsano, “Optimized Implementation of RNS FIR Filters Based on FPGAs,” *Journal of Signal Processing Systems,* Spinger, Online First™, 30 Sept 2010.

[9] P. Patronik, K. Berezowski, S. J. Piestrak, J. Biernat and A. Shrivastava, “Fast and energy-efficient constant-coefficient FIR filters using residue number system,” in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pp. 385-390, 1-3 Aug. 2011.

[10] W. Wang, M. N. S. Swamy and M. O. Ahmad, “Novel Design and FPGA Implementation of DA-RNS FIR Filters,” *Journal of Circuits, Systems and Computers*, vol. 13, no. 06, pp. 1233-1249, Dec 2004.

[11] W. A. Chren, “One-hot residue coding for low delay-power product CMOS design,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* , vol. 45, no. 3, pp. 303-313, Mar 1998.

[12] A. Weinberger and J. L. Smith, “A Logic for High-Speed Addition,” *National Bureau Standards*, circular 591, pp. 3-12, 1958.

[13] M.M. Mano and C. R. Kime, *Logic and Computer Design Fundamentals*, Prentice Hall. USA, 1997, pp. 130-137.

[14] Predictive Technology Model [Online]. Available: *http://ptm.asu.edu/*.

Chan Hua Vun (M’85-SM’03) received his Bachelor of Engineering (1st class Honours) and Master of Engineering Science degrees in Electrical & Computer System Engineering from Monash University, Australia, and Ph.D. in Computer Engineering from Nanyang Technological University, Singapore.

He is currently an Associate Professor at the School of Computer Engineering, Nanyang Technological University, Singapore. His current research works focus on embedded signal converter and signal processing based on number theory approach.

Dr. Vun co-founded the IEEE Consumer Electronics Society, Singapore Chapter, and has been involved in numerous program and organizing committees for various IEEE conferences. He is currently also an Associate Editor of the IEEE Consumer Electronics magazine.

Benjamin Premkumar (M’92-SM’99) received his Bachelor of Science and Bachelor of Engineering in Electrical communication in India. He briefly worked in large communication industry in Bangalore (India) in their Research and Development division before proceeding to the US to earn his M.S. and Ph.D. both in Digital Signal processing from the University of Idaho. He was a group leader designing multiprocessor controlled equipment, low speed data circuits and order wire communication systems for the Indian Air Force and Posts and Telegraphs. He worked on speech recognition algorithms during his Master’s program. His Ph.D. thesis was in the area of Synthetic Aperture Radar Signal Processing, a project sponsored by NASA.

He has held various teaching positions since 1991 both in the US and Singapore. He was the recipient of Erskine Fellowship at the University of Canterbury in 2004. Currently he is an Associate Professor in the school of Computer Engineering, Nanyang Technological University, Singapore. His research interests are in the areas of Digital filters and its applications in Wireless Communication, Software Defined Radio, Ultra wide Band Radio and Cognitive Radio. He also works in the area underwater tracking and localization of sources in shallow waters. He has worked on several different number systems for implementing digital filters efficiently in hardware and has successfully applied these systems in communications systems for error detection and correction.

Dr. Premkumar has over 150 publications in international journals and conferences and has authored and co-authored three text books, two on mobile communications and one on number theory.

Wei Zhang (M’xx) received the Bachelor’s and Master’s degrees in electrical engineering from the Harbin Institute of Technology, Harbin, China, in 1999 and 2001, respectively, and the Ph.D. degree in computer engineering from Princeton University, Princeton, NJ, in 2009.

She is currently an Assistant Professor at the School of Computer Engineering, Nanyang Technological University, Singapore. Her research interests include embedded systems, reconfigurable computing, nanotechnology, electronic design automation, and network-on-chip.

Dr Zhang is a member of the Association for Computing Machinery. She received a Princeton Research Prize and a Best Paper Award in 2007 and 2009, respectively.