SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 9, No. 3, October 2012, 325-342

UDK: 621.372.544 **DOI: 10.2298/SJEE1203325S**

Constant-Coefficient FIR Filters Based on Residue Number System Arithmetic

Negovan Stamenković[1], Vladica Stojanović[1]

Abstract: In this paper, the design of a Finite Impulse Response (FIR) filter based on the residue number system (RNS) is presented. We chose to implement it in the (RNS), because the RNS offers high speed and low power dissipation. This architecture is based on the single RNS multiplier-accumulator (MAC) unit. The three moduli set $\{2^{n+1}, 2^n, 2^n - 1\}$, which avoids $2^n + 1$ modulus, is used to design FIR filter. A numerical example illustrates the principles of residue encoding, residue arithmetic, and residue decoding for FIR filters.

Keywords: FIR filters, Residue number system, Reverse converter, Forward converter, Modulo MAC unit.

1 Introduction

The Residue Number System (RNS) has been recognized as one of the efficient alternative number systems which can be used to high-speed hardware implementation of Digital Signal Processing computation algorithms. In RNS, an integer value with large word-length is divided into several relatively small integer copies by a specific moduli set. The addition and multiplication of RNS integers copies are performed in parallel. Each copy is called a channel and the so called RNS channel implements modular arithmetic. In this way, RNS arithmetic does not suffer from inter-channel propagation delay. Performance of the system can be increased by selecting small word-length channels with short internal carry prpagation delay [1, 2]. Due to this feature, many Digital Signal Processing architectures based on RNS have been introduced in the literature [3, 4]. Thus, RNS is an efficient method for the implementation of high-speed Finite Impulse Response (FIR) filters, where dominant operations are addition and multiplication. Implementation issues of RNS based FIR filters show that performance can be considerably increased, in comparison with traditional two's complement binary number system [3 – 5].

[1]Faculty of Science and Mathematics, University of Pristina (at K. Mitrovica), 28220 Kosovska Mitrovica, Lole Ribara 29, Serbia; E-mails: negovanstamenkovic@gmail.com; vlast70@gmail.com

The basic of each RNS is a moduli set with consist of a set of pairwise prime number [6]. Until now, many moduli set have been introduced for RNS [7 – 9]. Among these, the moduli set $\{2^n-1, 2^n, 2^n+1\}$ is most well known. This moduli set can result in simple design of forward and inverse converter, but performance of RNS arithmetic unit is restricted to the time-performance of modulo 2^n+1 channel. The modulo 2^n-1 operations are complex, and are bottleneck for RNS arithmetic unit. Hence, the moduli set $\{2^{n+1}, 2^n, 2^n-1\}$ [9, 10] is used as an alternative for moduli set $\{2^n-1, 2^n, 2^n+1\}$ in this paper. In this moduli set, the moduli $2^{n+1}-1$ is used instead of 2^n+1. The arithmetic unit of RNS system based on moduli set $\{2^{n+1}-1, 2^n, 2^n-1\}$ is faster than those based on moduli set $\{2^n-1, 2^n, 2^n+1\}$.

A technique for implementing a finite impulse response (FIR) digital filter in a residue number system (RNS) is presenter in this paper. From the viewpoint of FIR architecture, single RNS multiplier-accumulator (MAC)-based architecture is applied to each channel FIR filter design.

The paper is organized as follows. After an introduction about RNS and reverse conversion algorithms, the architecture of the constant coefficient FIR filters which have been designed for three moduli sets $\{2^{n+1}-1, 2^n, 2^n-1\}$ will be investigated in the third section. Section 4 discusses a method for binary representation into residues, but the MRC architecture of RNS to binary conversion in Section 5 is discussed. Finally, in Section 6 the implementation of a 27th-order lowpass FIR filter is used to demonstrate design parameters that must be considered in designing an RNS filter.

2 Background

A residue number system (RNS) is defined in terms of k integers that are relatively-prime. That is $m=\{m_1, m_2, \ldots, m_k\}$ where $\gcd(m_i, m_j)=1$ for $i, j=1, \ldots, k$ and $i<j$ and gcd means the greatest common divisor [6, 11, 12]. Any integer X, with $0 \le X < M$, where $M=\prod_{i=1}^{k} m_i$ is presented in the RNS as $X=(x_1, x_2, \ldots, x_k)$, where $x_i=\langle X \rangle_{m_i}$ and $\langle X \rangle_{m_i}$ denotes modulo operation.

Thus RNS is a non-weighted number system which speeds up arithmetic operations by dividing them into smaller operations. Since the arithmetic operation in each moduli channel are independent of the others moduli channel, there is no carry propagation among them and RNS leads to carry-free addition and borrow-free subtraction. Magnitude comparison overflow detection and the sign test are time consuming in RNS.

To perform the residue to binary conversion, that is, to convert the residue number $(x_1, x_2, \ldots, x_k)$ into the integer number X, the Chinese Remainder Theorem (CRT) and mixed-radix conversion (MRC) are generally used.

The CRT is formulated as folows [13 – 15]:

$$X = \left\langle \sum_{i=1}^{k} \langle x_i N_i \rangle_{m_i} M_i \right\rangle_M , \tag{1}$$

where $M_i = M / m_i$ and $N_i = \langle M_i^{-1} \rangle_{m_i}$ is the multiplicative inverse of M_i modulo m_i. The main disadvantage of this approach is that it requires multiplication by the M_i's, which are large numbers, and modulo M operations.

By MRC algorithm the residue numbr $(x_1, x_2, \ldots, x_k)$ can be converted into integer number X as follows:

$$X = a_k \prod_{i=1}^{k-1} M_i + \cdots + a_3 m_1 m_2 + a_2 m_1 + a_1 , \tag{2}$$

where a_i are the mixed-radix digits (MRDs) and they can be obtained from the residues by [11]

$$a_i = \left\langle (\cdots((x_i - a_1)c_{1,i} - a_2)c_{2,i} - \cdots - a_{i-1})c_{i-1,i} \right\rangle_{m_i} \tag{3}$$

and $c_{i,j}$ for $1 \le i \le j < k$ is the multiplicative inverse of m_i modulo m_j, or $\langle c_{ij} \times m_i \rangle_{m_j} = 1$, for $k > 1$ and $a_1 = x_1$.

3 FIR Filter Architecture

Traditional N-tap FIR filter with impulse response coefficients b_k can be described by

$$y(n) = \sum_{k=0}^{N-1} b_k x(n-k) . \tag{4}$$

Possible realization of a FIR filter [16, 17] in transposed form is shown in Fig. 1. The implementation of N trap FIR filter requires the implementation of N multiplications and $N-1$ additions. Multiplication is very costly regarding hardware and computational time because the arithmetic unit performs fixed point computation on numbers represented in 2's complement form. The arithmetic unit consists of a dedicated hardware multiplier and an adder connected to the accumulator so as to be able to efficiently execute the multiply-accumulate operation.

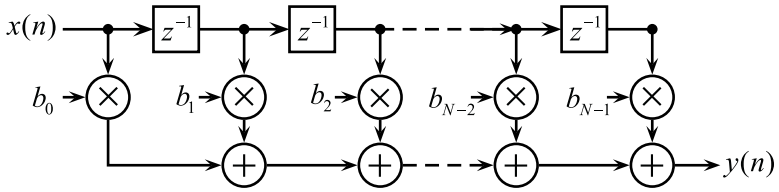

Fig. 1 – *Traditional* FIR *filter architecture*.

Fig. 3 shows the MAC architecture suitable for set of three moduli. The main components of an RNS system are a forward converter, parallel arithmetic channels and a reverse converter. The forward converter encodes a binary number into a residue represented number, with regard to the moduli set. Each arithmetic channel requires modular multiplication and accumulation for each modulo of set. The reverse converter decodes a residue represented number into its equivalent binary number. The arithmetic channels are working in a completely parallel architecture without any dependency, and this results in a considerable speed enhancement.

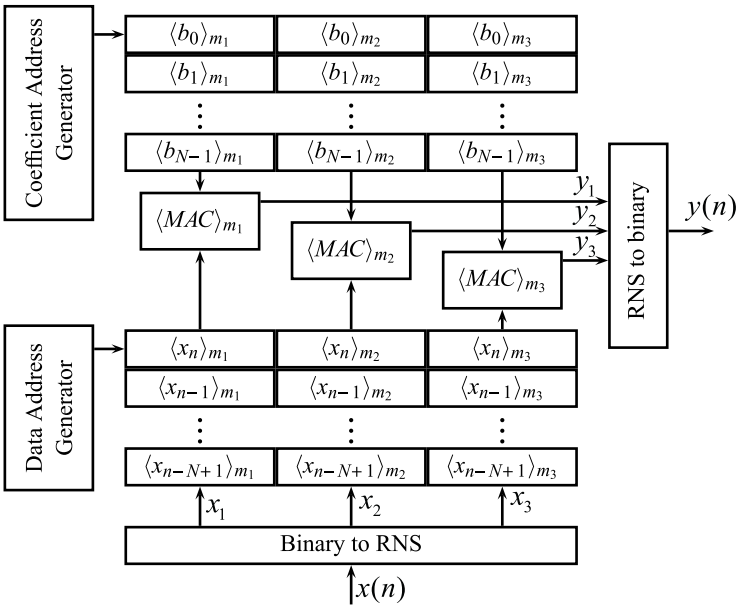

Fig. 2 – RNS *based* FIR *filter architecture for tree modili set*.

The architecture has two separate memory spaces which can be accessed simultaneously. One of the memories can be used to store coefficients and the other to store input data samples both in residue form. FIR filtering is achieved

in the RNS domain by using triple modulo FIR filter blocks. The implementation is generic and assumes 3 moduli (m_1 , m_2 , m_3) selected so as to meet the desired filter precision requirements. The FIR filtering is performed as a series of modulo MAC operations across each moduli m_1 , m_2 and m_3 .

3.1 The Multiply-Accumulate Unit

The associated formula of (4) for RNS FIR filters can be expressed as:

$$\langle y(n)\rangle_{m_i} = \left\langle \sum_{k=0}^{N} \langle b_k \rangle_{m_i} \langle x(n-k)\rangle_{m_i} \right\rangle_{m_i}, \text{ for } i = 1,2,3. \tag{5}$$

It is shown that the design of an FIR filter modulo m_i , (5) is actually a sum of product algorithm, that is, we need one modulo m_i MAC unit.

Fig. 3 shows the modulo m_i MAC unit, which multiplies two numbers, $\langle b_k \rangle_{m_i}$ and $\langle x(n-k)\rangle_{m_i}$, and sums the results. Multiply and accumulate operation can be performed in single cycle by MAC unit. Suppose, $\langle b_k \rangle_{m_i}$ and $\langle x(n-k)\rangle_{m_i}$ are two input sequence. In MAC unit (Fig. 3) the inputs are multiplied and added with zero which is initially stored in the memory (register). The sum is then stored in the memory unit. In the next clock, the next inputs are multiplied and added with previous data stored in the memory. Note, both operations are modular.

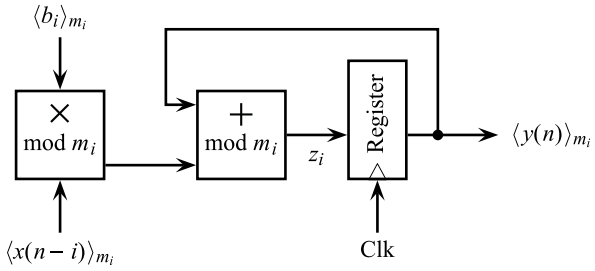

Fig. 3 – *Modulo* m_i MAC *unit*.

For example, initially the multiplication $z_0 = \langle b_0 \rangle_{m_i} \langle x(n)\rangle_{m_i}$ is computed. Then the modulo- m_i result z_0 is added to the residue product $\langle b_1 \rangle_{m_i} \langle x(n-1)\rangle_{m_i}$ to derive the intermediate quantity $z_1 = \langle z_0 + \langle b_1 \rangle_{m_i} \langle x(n-1)\rangle_{m_i}$. The result $\langle y(n)\rangle_{m_i}$ is recursively derived after N additions and multiplications. Hence the final result $y(n)$ is generated by the residue-to-binary conversion of the RNS result $\{\langle y(n)\rangle_{m_1}, \langle y(n)\rangle_{m_2}, \langle y(n)\rangle_{m_3}\}$.

In the following text modular adder and modular multiplier which can be used for MAC unit implementation will be described.

3.1.1 Modulo addition

Modulo (2^n-1) (modulo $2^{n+1}-1$) addition algorithm that avoids double representation of zero is defined by [18]:

$$\langle x_i + y_i \rangle_{2^n-1} = \langle x_i + y_i + 1 - \overline{C}_{out} \rangle_{2^n}, \tag{6}$$

where C_{out} is the carry-out of the addition $x+y$. Fig. 4a depicts the architecture of the corresponding hardware operator which requires carry-propagate adder, a NOR gate and decrementer [19]. Fig. 4b shows the internal logic circuit schematic of a decrementer, based on the conventional n-bit ripple borrow half subtractor. Only n half subtractors are used for constructing the decrement architecture.

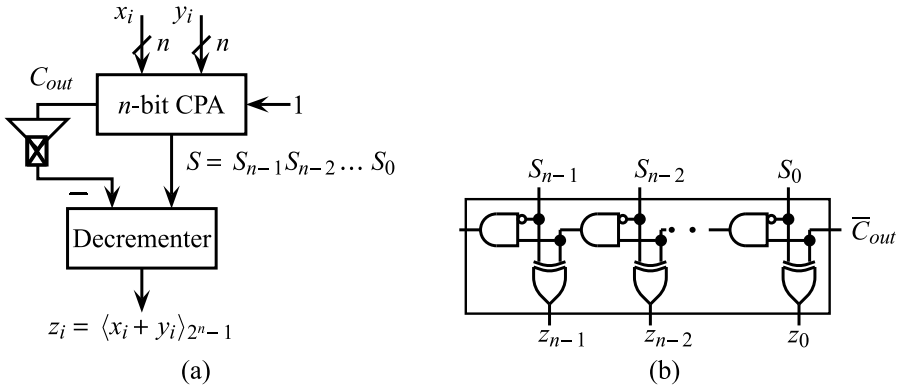

Fig. 4 – (a) *The modular addition with respect to* 2*n*−1 *hardware architecture*; (b) *ripple-borrow decrement stage*.

The modulo 2^n adder can be realized directly with n -bit adder with ignored overflow.

3.1.2 Modulo multiplication

Modulo 2^n-1 (modulo $2^{n+1}-1$) multiplication can be formulated as

$$\langle x_i \times y_i \rangle_{2^n-1} = \left\langle \langle x_i \times y_i \rangle_{2^n} + x_i \times y_i \text{ div } 2^n \right\rangle_{2^n-1}, \tag{7}$$

where $\langle x_i \times y_i \rangle_{2^n}$ corresponds to the low output word and $x_i \times y_i \text{ div } 2^n$ to the high output word of the multiplication $x_i \times y_i$. Therefore, modulo (2^n-1) multiplication can be accomplished by an n bit unsigned multiplication followed by an n bit modulo 2^n-1 addition.

Equation (7) can be rewritten as sum of partial products:

$$\langle x_i \times y_i \rangle_{2^n-1} = \sum_{k=0}^{n-1} \langle pp_{i,k} \rangle_{2^n-1}, \tag{8}$$

where: $x_i = x_{i,n-1}x_{i,n-2},\ldots,x_{i,0}$,

$y_i = y_{i,n-1}y_{i,n-2}\ldots y_{i,0}$, and

$pp_{i,k} = x_k \times (y_{i,n-k-1}\ldots y_{i,0}y_{i,n-1}\cdots y_{i,n-k})$, (implemented using AND gates)

is the k-th partial product modulo (2^n-1).

Note that all n-bit partial products $pp_{i,k}$ have the same magnitude (as opposed to ordinary multiplication, where the partial products have increasing magnitude), i.e., the number of partial product bits to add is the same for all bit positions. The partial product generation for inputs of four bits wide is as shown in **Table 1**.

Table 1

Partial product generation for modulo 2^4-1.

2^6	2^5	2^4	2^3	2^2	2^1	2^0
			$x_{i,0}y_{i,3}$	$x_{i,0}y_{i,2}$	$x_{i,0}y_{i,1}$	$x_{i,0}y_{i,0} = pp_{i,0}$
		$x_{i,1}y_{i,3}$	$x_{i,1}y_{i,2}$	$x_{i,1}y_{i,1}$	$x_{i,1}y_{i,0}$	$x_{i,1}y_{i,3} = pp_{i,1}$
	$x_{i,2}y_{i,3}$	$x_{i,2}y_{i,2}$	$x_{i,2}y_{i,1}$	$x_{i,2}y_{i,0}$	$x_{i,2}y_{i,3}$	$x_{i,2}y_{i,2} = pp_{i,2}$
$x_{i,3}y_{i,3}$	$x_{i,3}y_{i,2}$	$x_{i,3}y_{i,1}$	$x_{i,3}y_{i,0}$	$x_{i,3}y_{i,3}$	$x_{i,3}y_{i,2}$	$x_{i,3}y_{i,1} = pp_{i,3}$

In this multiplication, the bits with weight greater than 2^3, which are to the left of the straight line, are repositioned to the right of the line.

In conventional memory-based technique, the ROM stores the results of the multiplication of all possible values. Here, we extended further to obtain a memoryless-based implementation.

The architecture of proposed implementation of modulo (2^n-1) multiplication is shown in Fig. 5a. Assuming the coefficient word length of 4-bits and input sample word length of 4-bits, in Fig. 5a shows the hierarchical decomposition of a 4×4 Wallace tree logic. For (4×4) bits, four partial products are generated, and are added in parallel. The partial sums are added by using two carry save adders (CSA) and a carry propagate adder with end around carry (CPA with EAC).

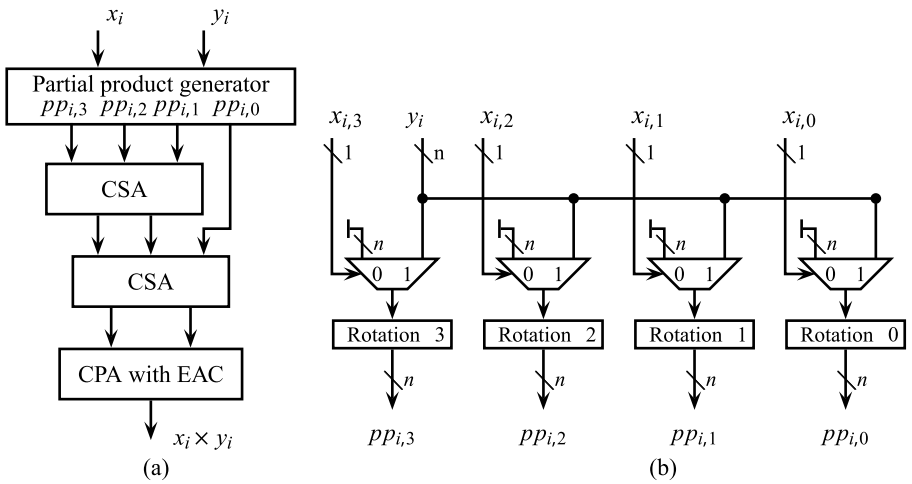

Fig. 5 – (a) *Architecture of modulo* (2^4-1) *multiplier*; (b) *partial product generator*.

The principe of the proposed memoryless-based implementation of partial product generator is shown in Fig. 5b. It consists of n 2-to-1 multiplexers, were n is input sample word length. The partial product is generated by connecting zero and coefficient value to the MUX data inputs, input data bits to the select input, and circular shifting output of the MUX $s-1$ bits left, for $1 \le s \le n$.

Assuming the multiplier Y and multiplicand $x_i = x_{i,3}x_{i,2}x_{i,1}x_{i,0}$ are words length of $n=4$ bits, there will be 4 values for partial product. Note, each of the MUX consist of n number of MUX cells (for 2^n-1 cannel), or $n+1$ number of MUX cells (for $2^{n+1}-1$ cannel) working in parallel.

Modulo 2^n multiplication can be formulated as

$$\langle x_i \times y_i \rangle_{2^n} = \sum_{k=0}^{n-1} pp_{i,k}\,, \tag{9}$$

were $pp_{i,k} = x_i \cdot (y_{i,n-k-1}\, y_{i,n-k-2} \cdots y_0 \overbrace{0\,0 \cdots 0}^{k})$. The partial products are added using carry save adders with the carry out bit discarded.

4 Binary to RNS Conversion

An integer X in the range $[0, M)$, represented in 2^n notation as [20 – 22]:

$$X = \sum_{i=0}^{3n-1} X_i 2^i = N_2 2^{2n} + N_1 2^n + N_0 \tag{10}$$

can be uniquely represented in RNS by the set (x_1, x_2, x_3) for the moduli set $\{m_1, m_2, m_3\}$. Three converters are required in order to obtain the RNS representation of the integer, one for each base element.

The simplest one is the converter for the $m_1 = 2^n$ channel. The value x_1 can be obtained by the remainder of the division of X by 2^n, which can be accomplished by truncating the value X, since:

$$x_1 = \langle X \rangle_{2^n} = X_{n-1} X_{n-2} \cdots X_0 . \tag{11}$$

For the $2^n - 1$ and $2^{n+1} - 1$ channels the calculation of the corresponding residues is more complex, since the final result of the conversion depends on the value of all the X_i bits. Instead of using a division operation to calculate the $2^n - 1$ residue, which is a complex operation and expensive both in terms of area and speed, this calculation can be performed as a sequence of additions, as described below:

$$x_2 = \langle X \rangle_{2^n - 1} = \langle N_2 2^{2n} + N_1 2^n + N_0 \rangle_{2^n - 1} . \tag{12}$$

By taking the equation:

$$\langle 2^n \rangle_{2^n - 1} = 1 , \tag{13}$$

(12) can be rewritten as:

$$x_2 = \langle N_2 + N_1 + N_0 \rangle_{2^n - 1} . \tag{14}$$

Thus the conversion of X to moduli $2^n - 1$ can be performed simply by adding modulo $2^n - 1$ the N_i components of X.

In an identical manner, the $2^{n+1} - 1$ residue can be calculated as:

$$\begin{aligned} x_3 = \langle X \rangle_{2^{n+1} - 1} &= \langle N_2' 2^{2(n+1)} + N_1' 2^{n+1} + N_0' \rangle_{2^{n+1} - 1}, \\ x_3 &= \langle N_2' + N_1' + N_0' \rangle_{2^{n+1} - 1}, \end{aligned} \tag{15}$$

where N_0', N_1' and N_2' ar $n+1$ bit numbers.

5 RNS to Binary Conversion

Given RNS number (y_1, y_2, y_3) with respect to the moduli set $\{2^n, 2^n - 1, 2^{n+1} - 1\})$, the proposed algorithm compute binary equivalent of the RNS number using MRC technique. For proposed moduli set $k = 3$ then the (2) reduces to:

$$y = a_3 2^n (2^n - 1) + a_2 2^n + a_1 = (a_2 + a_3 2^n - a_3) 2^n + a_1,$$
$$y = (a_4 - a_3) 2^n + a_1 = a_5 2^n + a_1. \tag{16}$$

The various multiplicative inverse for proposed moduli set are: $c_{12} = 1$, $c_{13} = 2$ and $c_{23} = -2$ [9]. Mixed-radix digits are computed using (3):

$$a_1 = y_1,$$
$$a_2 = \langle y_2 - y_1 \rangle_{2^n - 1}, \tag{17}$$
$$a_3 = \langle (a_2 - 2(y_3 - y_1))2 \rangle_{2^{n+1} - 1}.$$

Binary number a_4 is only concatenation of MRDs a_2 and a_3, were a_3 is MSB and a_2 is LSB,

$$a_4 = a_2 + a_3 2^n. \tag{18}$$

Finally, a_5 is traditional subtraction of MDRs a_4 and a_3:

$$a_5 = a_4 - a_3. \tag{19}$$

The proposed architecture of RNS-to-binary number conversion is depicted in Fig. 6a. It contains two modulo $2^{n+1} - 1$ subtractors, one modulo $2^n - 1$ and one traditional borrow propagation subtractor (BPS).

The modulo $(2^n - 1)$ subtraction can be expressed as follows:

$$\langle x - y \rangle_{2^n - 1} = \langle x - y - B_{out} \rangle_{2^n}. \tag{20}$$

The borrow out signal (B_{out}), which results from the subtraction of both x and y, can be used in the process of computing modulo $2^n - 1$ subtraction. This is due to the following observations:

$$B_{out} = 1 \quad \text{if} \quad x < y,$$
$$B_{out} = 0 \quad \text{if} \quad x \geq y. \tag{21}$$

Then modulo 2^n subtractor with borrow out feed back into the borrow input (to achieve end-around-borrow), can be used to implement modulo $2^n - 1$ subtractor (20). This type of subtractor is also known as the Borrow-Propagate-Subtractor wit End-Around-Borrow (BPS with EAB). Note, that proposed modulo $(2^n - 1)$ subtraction algorithm, which avoids the double representation of the zero, cover whole dynamic range while modulo subtractor based on the CPA with EAC does not [9]. For example, the the residue-to binary converter, that uses modulo $2^n - 1$ subtractor based on the CPA with end-around carry, generate wrong results for $y_1 = y_2 = y_3$. For $n = 6$ and $y_1 = y_2 = y_3 = 62$

residue-to-binary converter generate $y = 512126$ instead of $y = 62$. It can be concluded, subtractor based on the CPA with end around carry reduce dynamic range to $2^n - 2 < y < (2^n - 1)2^n(2^{n+1} - 1)$.

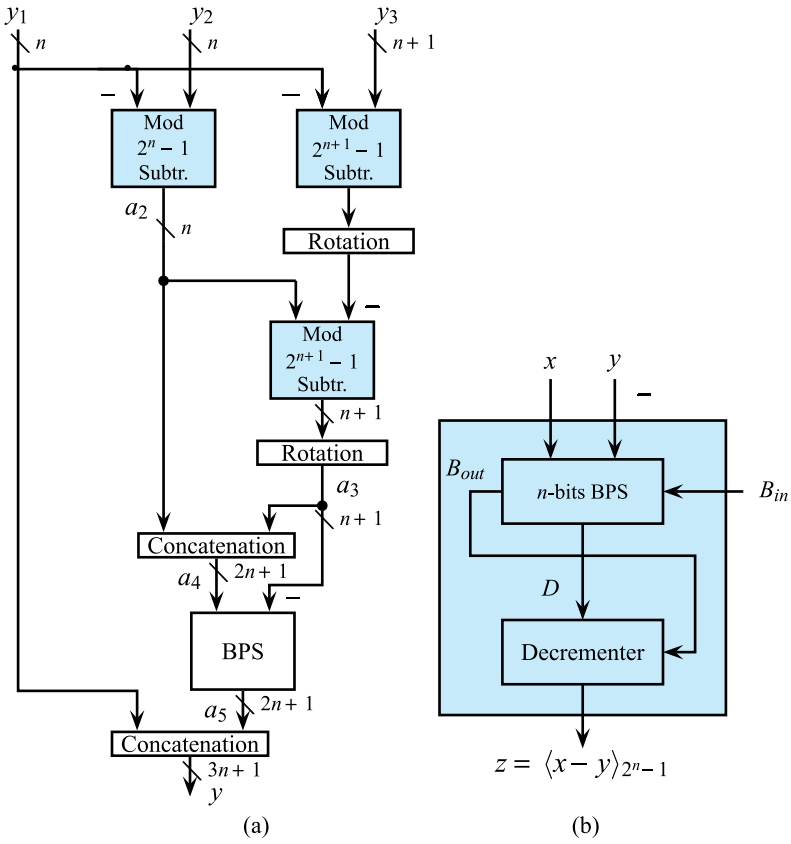

Fig. 6 – (a) *The* MRC *architecture for the moduli set* $\{2^n, 2^{n-1}, 2^{n+1}-1\}$; (b) *modulo* 2^{n-1} *subtractor based on the Borrow Propagate Subtractor with End Around Borrow* (BPS *with* EAB).

If modulo subtractor is performed as an ordinary subtractor with end around borrow (EAB), where the borrow output depends on the borrow input, a combinational logic is created to eliminate an unwanted race condition. Modulo subtractor based on the BPS with EAB is shown on Fig. 6b, but one solution for decrementer is shown in Fig. 4b.

Since $c_{12} = 1$, mixed-radix digit a_2 is available after modulo subtraction $y_2 - y_1$. Next, a_3 is obtained after four operations: the first operation is modulo

subtraction $y_3 - y_1$, a second operation is resulting difference left circular shifting by 1 bit (since $c_{13} = 2$), a third operation is modulo subtraction with a_2 as minuend and results of circular shifting as subtrahend, and a last operation is carried out by left circular shifting by 1 bit (since $c_{23} = -2$) in the result.

Suppose that MRDs a_1, a_2 and a_3 have binary representations as follows

$$\begin{aligned} a_1 &= a_{1,n-1}\, a_{1,n-2} \cdots a_{1,0} \\ a_2 &= a_{2,n-1}\, a_{2,n-2} \cdots a_{2,0} \\ a_3 &= a_{3,n}\, a_{3,n-2} \cdots a_{3,0} \end{aligned} \tag{22}$$

As shown in (18) and (19), the three operand to be added to obtain a_5. These three operands is simplified as two $(2n + 1)$-bit word a_4 and a_5 since a_2 and $a_3 2^n$ together have zeros in all $(2n + 1)$-bit positions:

$$\begin{aligned} a_2 &= \underbrace{0\,0 \cdots 0}_{n+1} \underbrace{a_{2,n-1}\, a_{2,n-2} \cdots a_{2,0}}_{n}, \\ a_3 2^n &= \underbrace{a_{3,n}\, a_{3,n-1} \cdots a_{3,0}}_{n+1} \underbrace{0\,0 \cdots 0}_{n}. \end{aligned} \tag{23}$$

Thus

$$a_4 = a_{3,n}\, a_{3,n-1} \cdots a_{3,0} a_{2,n-1}\, a_{2,n-2} \cdots a_{2,0}\,. \tag{24}$$

The subtraction $a_5 = a_4 - a_3$ can be realized with traditional BPS. Finally, after bits of organization (concatenation), based on the (16) we get the final result as a binary number $y = a_5 2^n + a_1$ after bit organization:

$$y = \underbrace{a_{5,2n}\, a_{5,2n-1} \cdots a_{5,0}}_{2n+1} \underbrace{a_{1,n-1}\, a_{1,n-2} \cdots a_{1,0}}_{n}\,. \tag{25}$$

6 Filter Performance

The design and numerical computation of an FIR filter was done using MATLAB using Parks-McClellan algorithm [23] in a two-step process. The first is to use the `firpmord` command to estimate the order of the optimal Parks- McClellan FIR filter to meet design specifications. The syntax of the command is as follows: `[N,fo,mo,w]=firpmord(f,m,dev)`, where `f=[0.4 0.5]` is the vector of band frequencies, vector `m=[1 0]` contains the desired magnitude response values at the passbands and the stopbands of the filter, and the vector `dev=[0.01 0.1]` has the maximum allowable deviations of the magnitude response of the filter from the desired magnitude

response. The second step is the actual design of the filter, using the `firpm` command `b=firpm(N,fo,mo)` to find the impulse response b_i of the Parks-McClellan FIR filter for our design.

The filter coefficients are shown in **Table 2** for double precision and for 9-bit precision, including the sign bit, in integer notation.

Integer value in the third column in **Table 2** are transformed from floating point value (second column) in two steps. The first step is the conversion of floating point filter coefficients *b* in binary string `b` binary using two MatLAB functions, `Q_1=quantizer('round',Format)` and `b_binary= num2bin(Q_1,b)`. Value `Format` in quantizer MatLAB function creates parameters of binary numbers: `[wordlength,fractionlength]` for signed fixed-point mode. For 9-bit precision format are `wordlength=9` and `fractionlength=8`.

Table 2
The 27th*-order* FIR *lowpass filter coefficients for mouli set* $(2^{n-1}, 2^n, 2^{n+1}-1\}$, *with* $n = 6$.

Coefficients b_i	Double precision	Int.	RNS number		
			$b_{i,1}$	$b_{i,2}$	$b_{i,3}$
$b_0 = b_{27}$	0.004493088227562	1	1	1	1
$b_1 = b_{26}$	−0.023540422135223	−6	58	57	121
$b_2 = b_{25}$	−0.010537273615351	−3	61	60	124
$b_3 = b_{24}$	0.014434554627302	4	4	4	4
$b_4 = b_{23}$	0.017852276297252	5	5	5	5
$b_5 = b_{22}$	−0.014709152081930	−4	60	59	123
$b_6 = b_{21}$	−0.031608802439920	−8	56	55	119
$b_7 = b_{20}$	0.009659673837214	2	2	2	2
$b_8 = b_{19}$	0.051468586283075	13	13	13	13
$b_9 = b_{18}$	0.005218868131868	1	1	1	1
$b_{10} = b_{17}$	−0.084444425211876	−22	42	41	105
$b_{11} = b_{16}$	−0.047670664338718	−12	52	51	115
$b_{12} = b_{15}$	0.179378544446062	46	46	46	46
$b_{13} = b_{14}$	0.413118538653651	106	42	43	106

The second step is the conversion of binary string `b` binary into integer value using two new MATLAB functions: `q_1=quantizer('round', Format)` and `b_int=bin2num(q_1,b_binary)`. In this case value `Format` fractionlength is equal zero, i.e. `Format=[9, 0]`. At last, integer values of filter coefficients are transformed in RNS number. For coefficients forward conversion MATLAB function `mod` can be used.

This paper investigates binary to residue converter for the modulo set $\{64,63,127\}$. In the following example we describe the fixed point-to-residue number system conversion of coefficient b_1. Double precision of filter coefficient b_1 is -0.023540422135223 which is converted to binary number `b_binary=111111010`, than to integer number `b_int=-6`, and at last to RNS number `b_RNS=(58, 57, 121)`.

The simulation, which is done in MATLAB, depicts the effects of this design approach on the filter. Fig. 7 shows a plot of the ideal filter (dotted line) and the actual output. It is shown, the residue number based FIR filter to have a satisfactory attenuation performance.

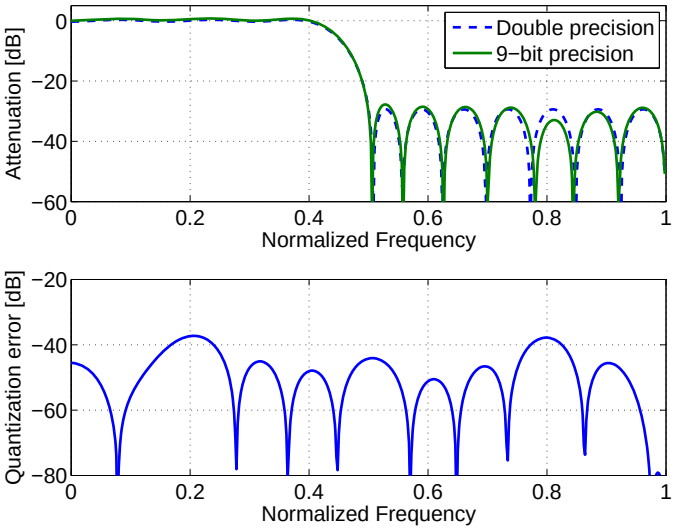

Fig. 7 – *Attenuation response* (*top*) *and quantization error from filter coefficient rounding to* 9*-bits* (*bottom*).

Assume that the data sequence is quantized to 10-bits (including sign) and that filter must be implemented without rounding error. An absolute upper bound on filter response $|y(n)|$ is given by (26):

$$|y(n)| \leq \max\{|x(n)|\} \sum_{k=1}^{30} |b_k| = 476718 \approx 18.86 \text{ bits}. \tag{26}$$

The moduli set $\{63,64,127\}$ provides a dynamic range of 18.96 bits, which is adequate for most practical situations since the bound of 18.86 bits given by (26) is extremely pessimistic.

Fig. 8 shows the impulse response of RNS channels. The coefficient values of the first and the second subfilter must be stored in 6 bit word, but the values

of the third subfilter must be stored in 7 bit word. In $\{64,63,127\}$ residue umber system unit sample is

$$\delta(n)=\begin{cases}(63,15,7) & \text{if} \quad n=0,\\ (0,0,0) & \text{if} \quad n\neq 0.\end{cases} \tag{27}$$

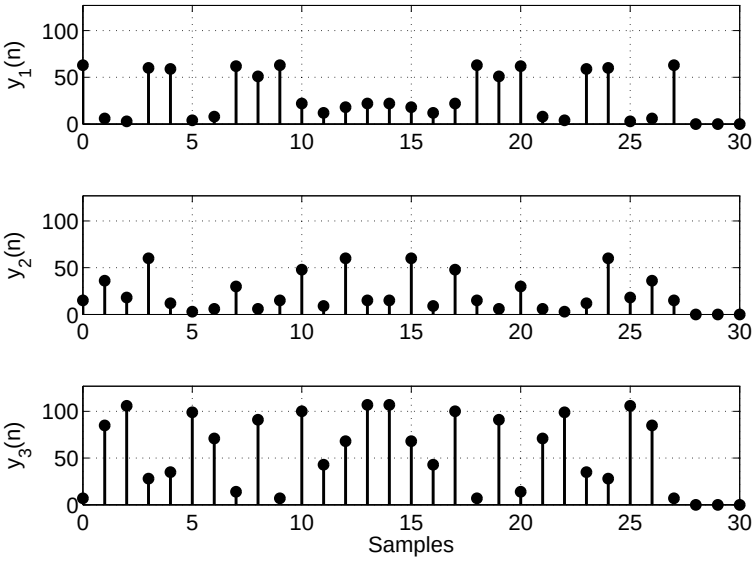

Fig. 8 – *The impulse response of the* RNS *channels*: (2^{n-1}) *above*; 2^n *in themiddle*; $(2^{n+1}-1)$ *down*.

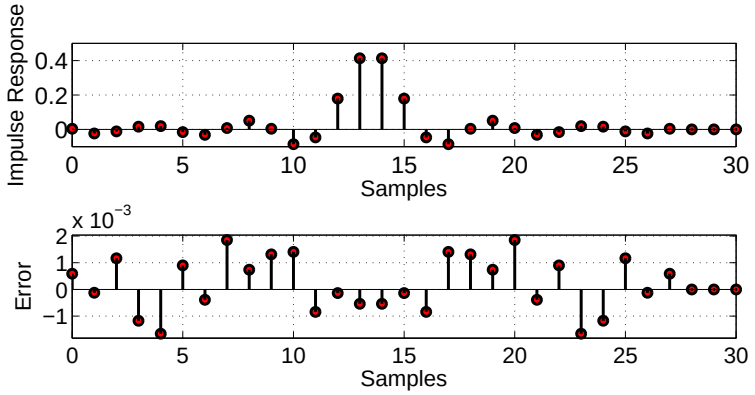

Fig. 9 – *Impulse response of the* RNS *lowpass filter* (*above*); *quantization error for impulse response* (*below*).

For the impulse response of whole filter we can use the mixed-radix conversion technique in order to convert a number, presented in the residue system, in the conventional number system, as shown in Fig. 6. The impulse response for this digital filter is shown in Fig. 9.

The quantization error in the impulse response, resulting from quantizing the coefficients to 9 bits, is shown in Fig. 9 below. It can be seen that 9 bits is sufficient to maintain error which is less than $2 \cdot 10^{-2}$.

7 Conclusion

The Residue Number System has been recognized as one of the efficient alternative number systems which can be used to high-speed hardware implementation of Digital Signal Processing computation algorithms. However, forward and reverse converters are needed to act as interfaces between RNS and the conventional binary digital systems. The overhead of these converters can frustrate the speed efficiency of RNS, and due to this a lot of research has been done to design efficient reverse converters.

This paper presents a study on the state-of-the-art digital signal processing which have been designed for the recently introduced large dynamic range RNS three-moduli sets.

The applications of RNS to constant coefficient FIR filters has not been thoroughly researched yet in the literature, therefore based on our preliminary research, we propose to develop (i) new residue number systems that balance inter-channel slack therefore maximize the use of the clock cycle; (ii) residue channels architectures that exploit slack balancing for low power; and (iii) characterized prototype circuit.

8 Acknowledgement

The authors are grateful to Professor Vidosav Stojanovic for comments and suggestions that improved the presentation in the paper.

This work was supported by the Serbian Ministry of Science and Technological Development, Project No. 32009TR

9 References

[1] P.V.A. Mohan: Residue Number Systems: Algorithms and Architectures, Kluwer Academic Publisher, Dordrecht, Netherlands, 2002.

[2] A. Omodi, B. Prekumar: Residue Number System – Theory and implementation, Imperial College Press, London, UK, 2007.

[3] R. Conway, J. Nelson: Improved RNS FIR Filter Architectures, IEEE Transaction on Circuits and Systems II: Express Briefs, Vol. 51, No. 1, Jan. 2004, pp. 26 – 28.

[4] W.K. Jenkins, B. Leon: The use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters, IEEE Transaction on Circuits and Systems, Vol. 24, No. 4, Apr. 1977, pp. 191 – 201.

[5] G.C. Cardarilli, A. Nannarelli, M. Re: Residue Number System for Low-power DSP Applications, 41st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 04 – 07 Nov. 2007, pp. 1412 – 1416.

[6] H.L. Garner: The Residue Number System, IRE Transaction on Electronic Computer, Vol. EC-8, No. 2, June 1959, pp. 140 – 147.

[7] A. Hiasat, A. Zohdy: Residue-to-binary Arithmetic Converter for the Moduli Set (2^k, 2^k–1, 2^{k-1}–1), IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 45, No. 2, Feb. 1998, pp. 204 – 209.

[8] Y. Wang, X. Song, M. Abdoulhamid, H. Shen: Adder Base Residue to Binary Number Converters for (2^n–1, 2^n, 2^n+1), IEEE Transaction on Signal Processing, Vol. 50, No. 7, July 2002, pp. 1772 – 1779.

[9] P.V.A. Mohan: RNS-to-binary Converter for a New Three-moduli set (2^{n+1}–1, 2^n, 2^n–1), IEEE Transactions on Circuits and Systems II: Express Briefs, Vol. 54, No. 9, Sept. 2007, pp. 775 – 779.

[10] S.H. Lin, M.H. Sheu, C.H. Wang: Efficient VLSI Design of Residue-to-binary Converter for the Moduli Set (2^n, 2^{n+1}–1, 2^n–1), IEICE Transactions on Information and Systems, Vol. E91-D, No. 7, July 2008, pp. 2058 – 2060.

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

[12] F.J. Taylor: Residue Arithmetic: A Tutorial with Examples, Computer, Vol. 17, No. 5, May 1984, pp. 50 – 62.

[13] B. Parhami: Computer Arithmetic: Algorithm and Hardware Designs, Oxford University Press, New York, USA, 2000.

[14] Y. Wang: Residue-to-binary Converters based on New Chinese Remainder Theorems, IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 47, No. 3, March 2000, pp. 197 – 205.

[15] J.W. Chen, R.H. Yao: Efficient CRT-based Residue-to-binary Converter for the Arbitrary Moduli Set, Science China – Information Sciences, Vol. 54, No. 1, Jan. 2011, pp. 70 – 78.

[16] M. Nikolić, M. Lutovac: Sharpening of the Multistage Modified Comb Filters, Serbian Journal of Electrical Engineering, Vol. 8, No. 2, Nov. 2011, pp. 281 – 291.

[17] S. Damjanović, L. Milić: A Family of IIR Two-band Orthonormal QMF Filter Banks, Serbian Journal of Electrical Engineering, Vol. 1, No. 3, Sept. 2004, pp. 45 – 56.

[18] R. Zimmermann: Efficient VLSI Implementation of Modulo (2^n±1) Addition and Multiplication, 14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia, 14 – 16 April 1999, pp. 158 – 167.

[19] J.L. Beuchat: Some Modular Adders and Multipliers for Field Programmable Gate Arrays, 17th International Symposium on Parallel and Distributed Processing, Los Alamitos, CA, USA, 22 – 26 April 2003, pp. 190 – 197.

[20] B. Vinnakota, V.V.B. Rao: Fast Conversion Techniques for Binary-residue Number Systems, IEEE Transaction on Circuits And Systems I: Fundamental Theories And Applications, Vol. 41, No. 12, Dec. 1994, pp. 927 – 929.

[21] S.J. Piestrak: Design of Residue Generators and Multioperand Modular Adders using Carry-save Adders, IEEE Transactions on Computers, Vol. 43, No. 1, Jan. 1994, pp. 68 – 77.

[22] R. Chaves, L. Sousa: (2^n+1, 2^{n+k}, 2^n−1): A New RNS Moduli Set Extension, Euromicro Symposium on Digital System Design, Rennes, France, 31 Aug. – 03 Sept. 2004, pp. 210 – 217.

[23] L.R. Rabiner, J.H. McClellan, T.W. Parks: FIR Digital Filter Design Techniques using Weighted Chebyshev Approximation, Proceedings of the IEEE, Vol. 63, No. 4, April 1975, pp. 595 – 610.