SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 11, No. 3, October 2014, 365-377

UDC: 621.372.54:004.383.3]:512.64 **DOI: 10.2298/SJEE140213026S**

Scaling Function Based on Chinese Remainder Theorem Applied to a Recursive Filter Design

Negovan Stamenković[1], Dragana Živaljević[2], Vidosav Stojanović[2]

Abstract: Implementation of IIR filters in residue number system (RNS) architecture is more complex in comparison to FIR filters, due to introduction of the scaling function. This function performs operation of division by a constant factor, which is usually the power of two, and after that the operation of rounding. In that way dynamic range reduction in digital systems is achieved. There are different methods for scaling operation implementation, already presented in references. In this paper, some RNS dynamic reduction techniques have been analyzed and then application of one selected technique has been presented on example. In all RNS calculations the power of two moduli set $\{2^n-1, 2^n, 2^n+1\}$ has been applied.

Keywords: Residue arithmetic, Digital arithmetic, Scaling, Chinese remainder theorem.

1 Introduction

Residue number system (RNS) is a parallel number representation system. In RNS, an integer with large word-length is divided into several relatively small integers according to a specific moduli set. The additions and multiplications of RNS integers are performed concurrently and independently, and there are no carries among residue channels.

The N th-order recursive digital filter is characterized by the following discrete time-domain relation:

$$y(i) = \sum_{j=0}^{N} b_j x(i-j) - \sum_{j=1}^{N} a_j y(i-j), \tag{1}$$

where b_j is the set of forward coefficients, a_j is the set of reverse coefficients, x_i is the current input, x_{i-j} is the past input and y_{i-j} is the past output.

In order to perform practical implementation of this filter that use RNS arithmetic, the coefficients and input signal have to be coded by converting

[1]University of Priština (at K. Mitrovica), Faculty of Natural Science, 28220 Kosovska Mitrovica, Lole Ribara 29, Serbia; E-mail: negovanstamenkovic@gmail.com

[2]University of Niš, Faculty of Electronic Engineering, A. Medvedeva 14, 18000 Niš, Serbia

variables from the floating point system into integers, by multiplying by an appropriate conversion factor, K , and rounding the result to the nearest integer. Hence:

$$y(i) = \frac{1}{K}\left\{\sum_{j=0}^{N} B_j x(i-j) - \sum_{j=1}^{N} A_j y(i-j)\right\}, \tag{2}$$

where B_j and A_j are scaled up forward and reverse coefficients, respectively. Implementation of FIR filters does not require scaling [1, 2], but for the implementation of IIR filters scaling is necessary.

Scaling up is easy. Computation is simplified by multiplying coefficients with 2^l . Note that 2^l has been also transformed into RNS. Scaling down (in the following text only term 'scaling' will be used) in recursive filters is required in RNS code because stable recursive equations generally have floating point coefficients that cannot be represented in an integer number system. If we choose the proper moduli set, then the scaling factor K can be a product of several moduli or a single modulus, as it will be shown in the following text. This factor will be used to derive a sufficient condition that ensures that the RNS output $y(n)$ does not exceed its dynamic range M . The filter consists of three identical sections which compute

$$y_i(i) = \left\langle \frac{1}{K}\left\{\sum_{j=0}^{N} \langle B_j \rangle_{m_i} \langle x(i-j)\rangle_{m_i} - \sum_{j=1}^{N} \langle A_j \rangle_{m_i} \langle y(i-j)\rangle_{m_i}\right\}\right\rangle_{m_i}, \tag{3}$$

where $\langle x \rangle_{m_i}$ denotes the operation x modulo m_i .

In a typical IIR design using RNS, a system is implemented as a collection of recursive and nonrecursive system, each defined in terms of an FIR structure, as shown in Fig. 1. Each FIR system may be implemented directly from their coefficients [3].

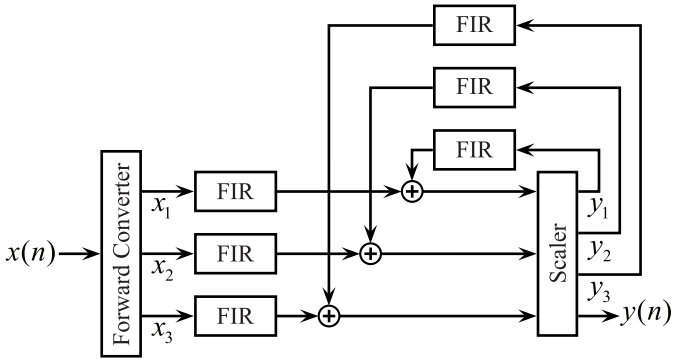

Fig. 1 – *RNS implementation of IIR filter using two FIR sections per channel and scaler.*

Therefore, for stable filter, the recursive part should be scaled to control dynamic range growth. The scaling operation may be implemented with mixed-radix conversion [4], Chinese remainder theorem (CRT) [5, 6], or new CRT-I [7]. For implementation of scaling algorithms there are two approaches available in the literature including LUT-based approaches [8, 9] and adder-based approaches [5, 10].

Hence, high efficient implementation of scaling is one of the critical issues in applications of RNS in recursive filter design. In this paper, a $K = 2^n$ scaling architecture based on the special moduli set, is presented. The scaling approach is CRT based. The scaling in the first and in the third channel is subtractor based while the second channel exploits CRT to generate the scaled residue [11].

This paper is organized as follows: Section 2 provides the detailed derivation of the proposed RNS scaling algorithm. Section 3 describes the implementation architecture of the proposed RNS scaler. The paper is concluded in Section 4.

2 RNS Scaling Technique

Certain scaling down operations have been proposed [5, 8, 12] for scaling an integer in the RNS. Let RNS number $X = (x_0, \ldots, x_k)$ be the input to the scaling process, Y the output, and K the scaling factor. In that case we have

$$Y = \left\lfloor \frac{X}{K} \right\rfloor = \frac{X - \langle X \rangle_K}{K} \tag{4}$$

where $\lfloor x \rfloor$ is the floor function, also called the greatest integer function. Since X is an unsigned integer number, then (4) becomes

$$\begin{aligned} y_i &= \langle Y \rangle_{m_i} \\ &= \langle \langle X - \langle X \rangle_K \rangle_{m_i} \langle K^{-1} \rangle_{m_i} \rangle_{m_i} \\ &= \langle \langle x_i - \langle X \rangle_K \rangle_{m_i} \langle K^{-1} \rangle_{m_i} \rangle_{m_i}. \end{aligned} \tag{5}$$

The scaling is completely defined by the set of residues $(y_0, \ldots, y_k)$.

A sufficient condition for the existence of $\langle K^{-1} \rangle_{m_i}$ in (5) is that Greatest Common Divisors (GCD) have value GCD $(K, m_i) = 1$. The main problem in finding residue Y is to obtain value of $\langle K^{-1} \rangle_{m_i}$. If the scaling factor K is the product of some moduli, that is, $K = \prod_{i=1}^{k} m_i$, then $\langle X \rangle_K$ is easily available. For example, if $K = m_1$, the value of $\langle X \rangle_K$ is directly available from the first residue x_1, as it will be shown in the following text. However, if $K = \prod_{i=1}^{k} m_i$,

since GCD $(K, p_i) \neq 1$ when $1 \le i \le k$, another algorithm has to be used to compute y_i.

The following lemmas are necessary for our RNS computation.

Lemma 1: *Let* $1 \le a \le m-1$ *and GCD* $(a,m)=1$, *then a multiplicative inverse* a^{-1} *which satisfy* $\langle a^{-1} \times a \rangle_m = 1$, *is*

$$a^{-1} = \begin{cases} 1, & \text{for } a = 1, \\ \dfrac{mk+1}{a}, & \text{for } 1 \le k \le a-1. \end{cases}$$

Lemma 2: *The multiplication in modulo* $2^n - 1$ *of a residue number* x_1 *by* 2^k, *where* k *is a natural number, is carried out by* k*-bit circular left shifting. Therefore:*

$$\langle x_1 \times 2^k \rangle_{2^n-1} = x_{1,n-k-1} \cdots x_{1,1} x_{1,0} \underbrace{x_{1,n-1} \cdots x_{1,n-k}}_{k}.$$

The multiplication of a residue number x_1 *by* 2^{n-k} *is carried out by k-bit circular right shifting.*

Lemma 3: *If* $\langle X \rangle_m = a$, $k \mid X$ *and GCD* $(k,m)=1$, *then modular operation after scaling is*

$$\left\langle \frac{X}{k} \right\rangle_m = \langle k^{-1} a \rangle_m,$$

where $k \mid X$ *denotes that the* X *is divisible by* k.

Consider the well-known 3-moduli set $\{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^n + 1\}$ which has a dynamic range approximately equal to 3*n* bits. To reconstruct the binary number from its residues $X = (x_1, x_2, x_3)$, the Chinese reminder theorem (CRT) is generally used [13 – 15] according to

$$X = \left\langle m_2 m_3 \langle M_1^{-1} \rangle_{m_1} x_1 + m_1 m_3 \langle M_2^{-1} \rangle_{m_2} x_2 + m_1 m_2 \langle M_3^{-1} \rangle_{m_3} x_3 \right\rangle_M, \qquad (6)$$

where $M = m_1 m_2 m_3$, $M_i = M / m_i$ and M_i^{-1} is the multiplicative inverse of M_i modulo m_i.

The multiplicative inverse for given moduli set is shown as follows: $\langle M_1^{-1} \rangle_{m_1} = 2^{n-1}$, $\langle M_2^{-1} \rangle_{m_2} = 2^n - 1$ and $\langle M_3^{-1} \rangle_{m_3} = 2^{n-1} + 1$. By replacing these values in (6) we obtain

$$X = \left\langle 2^n (2^n + 1) 2^{n-1} x_1 + (2^{2n} - 1)(2^n - 1) x_2 + 2^n (2^n - 1)(2^{n-1} + 1) x_3 \right\rangle_{2^n (2^{2n} - 1)} \qquad (7)$$

or

$$X = \Big\langle 2^n(2^n+1)2^{n-1}x_1 + 2^n(2^{2n}+2^n-1)x_2 + x_2 + 2^n(2^{2n-1}+2^n-2^{n-1}-1)x_3 \Big\rangle_{2^n(2^{2n}-1)}. \quad (8)$$

Since $\langle 2^n\ X\rangle_{2^n(2^{2n}-1)} = 2^n\langle X\rangle_{2^{2n}-1}$, $2^n - 2^{n-1} = 2^{n-1}$, $\langle 2^{2n}\rangle_{2^{2n}-1} = 1$ and using Lemma 2, (8) can be further simplified

$$X = 2^n\Big\langle (2^{2n-1}+2^{n-1})x_1 + (2^{2n}-1)x_2 - 2^n x_2 + (-2^{2n-1}+2^{n-1})x_3 \Big\rangle_{(2^{2n}-1)} + x_2 . \quad (9)$$

After partial modulo $2^{2n}-1$ operation we obtain

$$X = 2^n\Big\langle (2^{2n-1}+2^{n-1})x_1 - 2^n x_2 + (-2^{2n-1}+2^{n-1})x_3 \Big\rangle_{(2^{2n}-1)} + x_2 . \quad (10)$$

Finally, the calculation of CRT (6) can be written as

$$X = \langle A + B + C\rangle_{2^{2n}-1}\ 2^n + x_2 , \quad (11)$$

where

$$\begin{aligned} A &= \langle (2^{2n-1}+2^{n-1})\ x_1\rangle_{2^{2n}-1}, \\ B &= \langle (-2^n)\ x_2\rangle_{2^{2n}-1}, \\ C &= \langle (-2^{2n-1}+2^{n-1})\ x_3\rangle_{2^{2n}-1}. \end{aligned} \quad (12)$$

Only logic operation can be used to evaluate operands A, B and C.

Since $y_2 = \langle X\rangle_{m_2}$, then by using (12)

$$y_2 = \langle A + B + C\rangle_{2^n} . \quad (13)$$

Thus, n last significant bits of $A+B+C$ is y_2. The resultant residue digits of the scaled output for all modulus channels are identical to the scaled integer output with a scaling error not greater than one.

Assuming that the $2n$ bit expressions of x_1, x_3, x_3 are given by (the MSB's are given first):

$$\begin{aligned} x_1 &= \underbrace{0\,0\ldots0}_{n}\ \underbrace{x_{1,n-1}x_{1,n-2}\ldots x_{1,0}}_{n} \\ x_2 &= \underbrace{0\,0\ldots0}_{n}\ \underbrace{x_{2,n-1}x_{2,n-2}\ldots x_{2,0}}_{n} \\ x_3 &= \underbrace{0\,0\ldots0}_{n-1}\ \underbrace{x_{3,n}x_{3,n-1}\ldots x_{3,0}}_{n+1} \end{aligned} \quad (14)$$

the value A can be rewritten in binary form by substituting binary value of residue x_1 as

$$A = x_{1,0}\ \underbrace{x_{1,n-1}\ldots x_{1,0}}_{n}\ \underbrace{x_{1,n-1}\ldots x_{1,1}}_{n-1}. \tag{15}$$

The value B can be rewritten in binary form by substituting binary value of residue x_2 as

$$\begin{aligned} B &= \underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\ldots\overline{x}_{2,0}}_{n}\underbrace{11\ldots1}_{n} = \\ &= \langle \underbrace{0\,0\ldots0}_{n}\underbrace{11\ldots1}_{n} + \underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\cdots\overline{x}_{2,0}}_{n}\underbrace{0\,0\ldots0}_{n}\rangle_{2^{2n}-1}. \end{aligned} \tag{16}$$

The value C can be rewritten in binary form by substituting binary value of residue x_3 as

$$C = \langle \overline{x}_{3,0}\underbrace{11\ldots1\overline{x}_{3,n}}_{n}\ \underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1} + x_{3,n}\ \underbrace{x_{3,n-1}\ldots x_{3,0}}_{n}\ \underbrace{0\,0\ldots0}_{n}\rangle_{2^{2n}-1}. \tag{17}$$

The addition in (17) can be rewritten as shown in (18)

$$C = \langle \overline{x}_{3,0}\ \underbrace{x_{3,n-1}\ldots x_{3,0}}_{n}\ \underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1} + x_{3,n}\underbrace{11\ldots1\,\overline{x}_{3,n}}_{n}\ \underbrace{0\,0\ldots0}_{n-1}\rangle_{2^{2n}-1} \tag{18}$$

Four binary numbers in the summation of $C+B$ can be reduced into two numbers as follows

$$\begin{aligned} \left\langle C+B\right\rangle_{2^{2n}-1} &= \langle \overline{x}_{3,0}\ \underbrace{x_{3,n-1}\ldots x_{3,0}}_{n}\ \underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1} + x_{3,n}\underbrace{11\ldots1}_{n-1}\ \overline{x}_{1,n}\ \underbrace{0\,0\ldots0}_{n-1} \\ &+ \underbrace{0\,0\ldots0}_{n}\underbrace{11\ldots1}_{n} + \underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\ldots\overline{x}_{2,0}}_{n}\underbrace{0\,0\ldots0}_{n}\rangle_{2^{2n}-1}. \end{aligned} \tag{19}$$

Firstly, summation of the second and the third binary number on the right hand side of equation (19) has to be obtained. If $x_{3,n}=1$, summation of the second and the third binary number gives $\underbrace{11\ldots1}_{n-1}\,\underbrace{11\ldots1}_{n-1}$. On the other hand, if $x_{3,n}=0$ summation of the second and the third binary number gives $\underbrace{10\,0\ldots0}_{n-1}\,\underbrace{0\,11\ldots1}_{n-1}$. Thus, we can conclude

$$x_{3,n}\underbrace{11\ldots1\overline{x}_{3,n}}_{n-1}\ \underbrace{0\,0\ldots0}_{n-1} + \underbrace{0\,0\ldots0}_{n}\underbrace{11\ldots1}_{n} = 1\ \underbrace{x_{3,n}x_{3,n}\ldots x_{3,n}}_{n}\ \underbrace{11\ldots1}_{n-1}. \tag{20}$$

By substituting the (20) into (19) it is obtained that

$$\langle C+B\rangle_{2^{2n}-1}=\langle \overline{x}_{3,0}\underbrace{x_{3,n-1}\ldots x_{3,0}}_{n}\underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1}+1\underbrace{x_{3,n}x_{3,n}\ldots x_{3,n}}_{n}\underbrace{11\ldots1}_{n-1} +\underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\cdots\overline{x}_{2,0}}_{n}\underbrace{0\,0\ldots0}_{n}\rangle_{2^{2n}-1}. \tag{21}$$

Since $x_3\le 2^n$ and $x_{3,n}$ and $x_{3,n-i}$, for $i=1,...,n$, can never be 1 at the same time, (21) can be simplified by combining n bits marked with $x_{3,n}$ in the second number with corresponding n bits $x_{3,n-i}$, $i=1,2,\ldots,n$ in the first number by using logic OR operation. Thus

$$\langle C+B\rangle_{2^{2n}-1}=\langle \overline{x}_{3,0}\underbrace{\breve{x}_{3,n-1}\ldots\breve{x}_{3,0}}_{n}\underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1}+1\underbrace{0\,0\ldots0}_{n}\underbrace{11\ldots1}_{n-1} +\underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\cdots\overline{x}_{2,0}}_{n}\underbrace{0\,0\ldots0}_{n}\rangle_{2^{2n}-1}, \tag{22}$$

where $\breve{x}_{3,i}=x_{3,i}\vee x_{3,n}$, $i=0,1,\ldots,n-1$ and $\vee$ denotes logic OR operation. At last we consider summation of the second and the third number on the right side of (22). If $x_{2,n-1}=1$, end-around-carry summation of the second and the third number of (22) gives $(1\underbrace{\overline{x}_{2,n-2}\ldots x_{2,0}}_{n-1}\,0\underbrace{11\ldots1}_{n-1})$. On the other hand, if $x_{2,n-1}=0$, binary addition with end-around-carry of the second and the third number of (22) gives $(0\underbrace{\overline{x}_{2,n-2}\ldots x_{2,0}}_{n-1}\,1\underbrace{0\,0\ldots0}_{n-1})$. We can conclude

$$\langle 1\underbrace{0\,0\ldots0}_{n}\underbrace{11\ldots1}_{n-1}+\underbrace{\overline{x}_{2,n-1}\overline{x}_{2,n-2}\ldots\overline{x}_{2,0}}_{n}\underbrace{0\,0\ldots0}_{n-1}\rangle_{2^{2n}-1}= =x_{2,n-1}\underbrace{\overline{x}_{2,n-2}\ldots\overline{x}_{2,0}}_{n-1}\overline{x}_{2,n-1}\underbrace{x_{2,n-1}\cdots x_{2,n-1}}_{n-1}. \tag{23}$$

Therefore, the four numbers in the summation of $C+B$ are reduced into following two numbers

$$\langle C+B\rangle_{2^{2n}-1}=\langle \overline{x}_{3,0}\underbrace{\breve{x}_{3,n-1}\ldots\breve{x}_{3,0}}_{n}\underbrace{\overline{x}_{3,n-1}\ldots\overline{x}_{3,1}}_{n-1}+ +x_{2,n-1}\underbrace{\overline{x}_{2,n-2}\cdots\overline{x}_{2,0}}_{n-1}\overline{x}_{2,n-1}\underbrace{x_{2,n-1}\cdots x_{2,n-1}}_{n-1}\rangle_{2^{2n}-1}. \tag{24}$$

Example 1: *Let* $n=4$, $2^{2n}-1=255$, $x_2=10_{10}=1010_2$, $x_3=10_{10}=01010_2$. *Then by using* (12)

$$C+B=\left\langle(-2^4)\times10+(-2^7+2^3)\times10\right\rangle_{255} =\left\langle-160-1200\right\rangle_{255}=170. \tag{25}$$

Using proposed algorithm with $x_{3,4} = 0$ *then*

$$C = \overline{x}_{3,0}\ \overline{x}_{3,3}\ \overline{x}_{3,2}\ \overline{x}_{3,1}\ \overline{x}_{3,0}\overline{x}_{3,3}\overline{x}_{3,2}\overline{x}_{3,1} = 1101\,0010\,,$$

$$B = x_{2,3}\overline{x}_{2,2}\overline{x}_{2,1}\overline{x}_{2,0}\overline{x}_{2,3}x_{2,3}x_{2,3}x_{2,3} = 11010111\,.$$

By modulo adding of these two binary numbers together we obtain the following result:

$C =$	1101 0010	
$B =$	1101 0111	+
Sum vector =	1 1010 1001	
	→ 1	EAC
Correct result: $C + B =$	1010 1010	170

That result is correct.

We will check the values of these numbers *A*, *B* and *C* with the MATLAB® script below.

```
n=6;
m=[2^n-1, 2^n, 2^n+1]; mp=2^(2*n)-1;
X=19191;
x1=mod(X,m(1));
x2=mod(X,m(2));
x3=mod(X,m(3));
% ------------------------------------------------
% The default orientation of the binary output is
% Right-MSB;
% ------------------------------------------------
xb1=de2bi(x1,2*n);
xb2=de2bi(x2,2*n);
xb3=de2bi(x3,2*n);
Ab=[xb1(2:n) xb1(1:n) xb1(1)];
q1(1:n-1)=xb2(n);
B_b=[q1, not(xb2(n)),not(xb2(1:n-1)),xb2(n)];
C_b=[not(xb3(2:n)),xb3(1:n)|xb3(n+1),not(xb3(1))];
Y=mod(bi2de(Ab)+bi2de(B_b)+bi2de(C_b),mp)*2^n+x2;
Check=[X Y]           % X and Y schould be equal
```

To implement the modulo addition of three $2n$-bit numbers (*A*, *B* and *C*) efficiently, we may use $2n$ full adders as carry-save-adders (CSA) to convert the three $2n$-bit numbers into two. The carry-out from the most significant bit (c_{2n}) is fed to the least significant bit position (c_0). Then fast $2n$-bit carry-propagate-adder (CPA) with end-around-carry (EAC), is used to perform the modulo addition of two numbers to obtain the final result. The architecture is shown in Fig. 2.

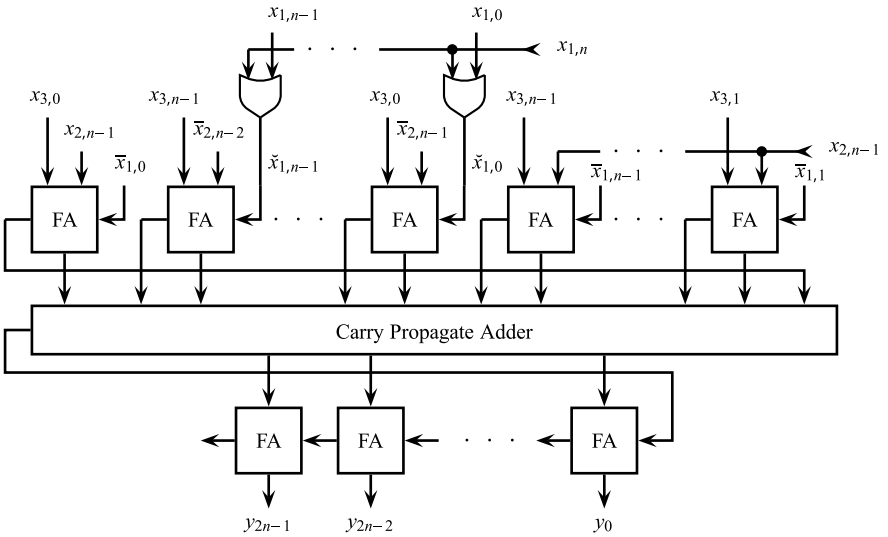

Fig. 2 – *The implementation operand preparation and modulo* $(2^{2n}-1)$ *addition of three* 2*n-bit numbers A, B and C.*

Scaling the integer variable X by constant integer $K = m_2$, can be obtained using (4) and replacing X with (6) [5]

$$Y = \left\lfloor \left\langle m_3 \langle M_1^{-1} \rangle_{m_1} x_1 + \frac{m_1 m_3}{m_2} \langle M_2^{-1} \rangle_{m_2} x_2 + m_1 \langle M_3^{-1} \rangle_{m_3} x_3 \right\rangle_{m_1 m_3} \right\rfloor. \quad (26)$$

Since $y_1 = \langle Y \rangle_{m_1}$ and $\langle \langle X \rangle_{m_1 m_2} \rangle_{m_1} = \langle X \rangle_{m_1}$, then

$$y_1 = \left\langle \left\langle m_3 \langle M_1^{-1} \rangle_{m_1} x_1 + \frac{m_1 m_3}{m_2} \langle M_2^{-1} \rangle_{m_2} x_2 + m_1 \langle M_3^{-1} \rangle_{m_3} x_3 \right\rangle_{m_1 m_3} \right\rangle_{m_1}. \quad (27)$$

By computing each of the residue in (27) independently, it can be reduced to

$$y_1 = \left\langle 2^n x_1 + \left\lfloor 2^{2n} - 2^n - 1 + \frac{1}{2^n} \right\rfloor x_2 = \left\langle x_1 - x_2 \right\rangle_{2^n - 1}. \quad (28)$$

It can be seen that scaling for the channel $2^n - 1$ needs only one modulo subtraction.

Since $y_3 = \langle Y \rangle_{m_3}$ and $\langle \langle X \rangle_{m_1 m_3} \rangle_{m_3} = \langle X \rangle_{m_3}$ then

$$y_3 = \left\langle \left\lfloor 2^{2n} - 2^n - 1 + \frac{1}{2^n} \right\rfloor x_2 + 2^n x_3 \right\rangle_{2^n + 1} = \left\langle x_2 - x_3 \right\rangle_{2^n + 1}, \quad (29)$$

because $\langle 2^n \rangle_{2^n+1} = -1$. Thus, the scaling for the channel $2^n + 1$ also needs only one modulo subtraction.

The second term of (26) is truncated to product approximation shown in (28) and (29). Since $0 \le x_2 < 2^n$, then $0 \le x_2 / 2^n < 1$ and $\lfloor x_2 / 2^n \rfloor = 0$. It can be concluded that the proposed algorithm does not introduce the scaling error.

Example 2: *To demonstrate the validity of the above scaling procedure we provide an example. Consider the integer* $X = 3987$ *with the RNS representation of* $(12,3,9)$ *for the moduli set* $\{15,16,17\}$. *Using* (28) *and* (29)*, the scaled output in RNS representation is:*

$$\begin{aligned} y_1 &= \langle 12 - 3 \rangle_{15} = 9 \\ y_2 &= \langle 249 \rangle_{16} = 9 \\ y_3 &= \langle 3 - 9 \rangle_{17} = 11 \end{aligned} \tag{30}$$

because $A + B + C = 249$. *It can be verified that the RNS number* $(9,9,11)$ *is equivalent to the integer* 249 *computed directly from* $\lfloor 3987 / 16 \rfloor$.

3 Architecture of RNS Scaler

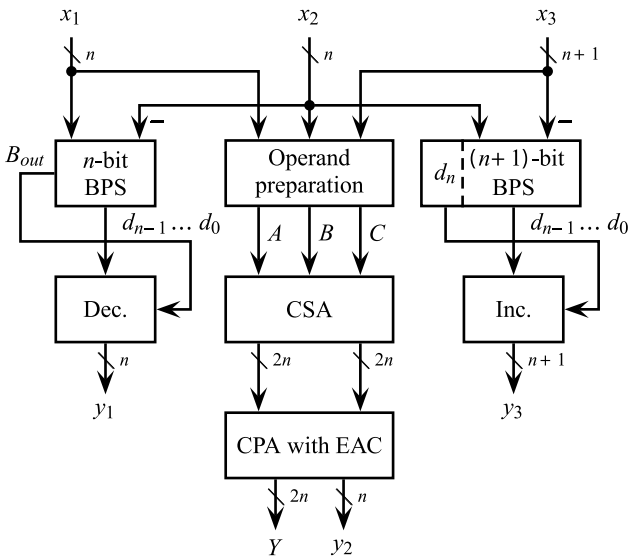

Fig. 3 – *Architecture of the proposed RNS scaler, where* $Y = \langle A + B + C \rangle_{2^{2n}-1}$ *and* y_2 *is n last significant bits.*

The complete RNS scaling architecture [16] is shown in Fig. 3. The 2*n*-bit number $Y = \langle A + B + C \rangle_{2^{2n}-1}$ is scaled integer $X = (x_1, x_2, x_3)$, but scaled residue y_2 is his *n* last significant bits. Thus, scaled output is available in RNS and in weighted binary form. Since, x_2 is an *n*-bit number, the 3*n*-bit integer *X* can be realized with only concatenation of x_2 and *Y*, without the use of hardware (11).

As shown above, scaling in channels $2^n - 1$ and $2^n + 1$ needs only subtractors. In following text we propose both subtractors for generating scaling numbers y_1 and y_3.

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

$$\langle x - y \rangle_{2^n-1} = \begin{cases} \langle x - y \rangle_{2^n}, & \text{if } x \geq y, \\ \langle x - y - 1 \rangle_{2^n}, & \text{if } x < y. \end{cases} \tag{31}$$

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:

$$\begin{aligned} B_{out} &= 1 \text{ if } x < y \\ B_{out} &= 0 \text{ if } x \geq y. \end{aligned} \tag{32}$$

Thus, it is easy to show that it is possible to express $\langle x - y \rangle_{2^n-1}$ as $\langle x - y - B_{out} \rangle_{2^n}$. Then modulo 2^n subtractor, with borrow out feedback signal, can be used to implement modulo $2^n - 1$ subtractor in (31). This type of subtractor is also known as the Borrow-Propagate-Subtractor with End-Around-Borrow (BPS with EAB). Decrementer can be composed of a half-subtractor array or a data-out MUX array and a selection modulo [17]. The proposed modulo $(2^n - 1)$ subtraction algorithm avoids the double representation of zero.

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

$$\langle x - y \rangle_{2^n+1} = \begin{cases} \langle x - y \rangle_{2^{n+1}}, & \text{if } x \geq y \\ \langle x - y + 1 \rangle_{2^{n+1}}, & \text{if } x < y \end{cases} \tag{33}$$

where $x \leq 2^n - 1$ and $y \leq 2^n$. The borrow bit is ignored. The MSB-bit, 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:

$$\begin{aligned} d_n &= 1 \text{ if } x < y, \\ d_n &= 0 \text{ if } x \geq y. \end{aligned} \tag{34}$$

Thus, it is easy to show that it is possible to express $\langle x-y\rangle_{2^n+1}$ as $\langle x-y+d_n\rangle_{2^n}$. Then modulo 2^n subtractor, with borrow-out feedback signal, can be used to implement modulo 2^n+1 subtractor in (33). Proposed modulo (2^n+1) subtraction algorithm avoids the double representation of zero. The modulo 2^n+1 subtraction for operands in modulo 2^n and modulo 2^n+1 representation can be performed by a conventional borrow-propagate-subtractor (BPS). Incrementer is composed of a half-adder array or a data MUX array [18].

Example 3: *Based on* (33), $y_3=\langle x_2-x_3\rangle_{2^n+1}$ *can be computed as*

x_2 =	0 0101	
x_3 =	0 1010	−
Difference vector =	1 1 1011	$B_{out}\ d_n\ d_{n-1}\ldots d_0$
	↳ 1	*End around* d_n
Correct result: y_3 =	0 1100	

Note, y_2 *is expressed as an* $(n+1)$*-bit number. Ignoring the borrow bit* B_{out} *yields to the correct result of* 12.

4 Conclusion

In this paper, architecture of 2^n scaling implementation for the traditional moduli set $\{2^n-1, 2^n, 2^n+1\}$ is proposed. The RNS scaling algorithm is developed based on classical CRT. For realization of scaling architecture, only one CPA with EAC and two subtractors are required. The first subtractor is modulo 2^n-1 and the second subtractor is modulo 2^n+1. This scaler is very suitable for building new types of recursive filter design.

The scaled residues are interfaced directly to the weighted binary system without the additional need for the residue-to-binary conversion.

5 Acknowledgement

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

6 References

[1] D. Zivaljevic, N. Stamenkovic, V. Stojanovic: Digital Filter Implementation based on the RNS with Diminished-1 Encoded Channel, 35th International Conference on Telecommunications and Signal Processing, Prague, Czech Republic, 03 – 04 July 2012, pp. 662 – 666.

[2] N. Stamenkovic, V. Stojanovic: Constant-coefficient FIR Filters based on Residue Number System Arithmetic, Serbian Journal of Electrical Engineering, Vol. 9, No. 3, Oct. 2012, pp. 325 – 342.

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

[5] C.H. Chang, J.Y.S. Low: Simple, Fast and Exact RNS Scaler for the Three-moduli Set {2n−1,2n,2n+1}, IEEE Transaction on Circuits and Systems I: Regular Papers, Vol. 58, No. 11, Nov. 2011, pp. 2686 – 2697.

[6] M. Griffin, F. Taylor, M. Sousa: New Scaling Algorithms for the Chinese Remainder Theorem, 22nd Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 31 Oct. – 02 Nov. 1988, Vol. 1, pp. 375 – 378.

[7] J.Y.S. Low, C.H. Chang: A New RNS Scaler for {2n−1,2n,2n+1}, International Symposium on Circuits and Systems, Rio de Janeiro, Brazil, 15 – 18 May 2011, pp. 1431 – 1434.

[8] A. Garcia, A. Lloris: A Look-up Scheme for Scaling in the RNS, IEEE Transactions on Computers, Vol. 48, No. 7, July 1999, pp. 748 – 751.

[9] Y. Kong, B. Phillips: Fast Scaling in the Residue Number System, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 17, No. 3, March 2009, pp. 443 – 447.

[10] M.A.P. Shenoy, R. Kumaresan: A Fast and Accurate RNS Scaling Technique for High Speed Signal Processing, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 37, No. 6, June 1989, pp. 929 – 937.

[11] N. Stamenkovic, D. Zivaljevic, V. Stojanovic: The Use of Residue Number System in the Design of the Optimal All-pole IIR Digital Filters, 36th International Conference on Telecommunications and Signal Processing, Rome, Italy, 02 – 04 July 2013, pp. 722 – 726.

[12] G.A. Jullien: Residue Number Scaling and other Operations using ROM Arrays, IEEE Transactions on Computers, Vol. C-27, No. 4, April 1978, pp. 325 – 336.

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

[14] S. Andraos, H. Ahmad: A New Efficient Memoryless Residue to Binary Converter, IEEE Transactions on Circuits and Systems, Vol. 35, No. 11, Nov. 1988, pp. 1441 – 1444.

[15] Z. Wang, G. Jullien, W.C. Miller: An Efficient 3-modulus Residue to Binary Converter, IEEE 39th Midwest symposium on Circuits and Systems, Ames, IA, USA, 18 – 21 Aug. 1996, Vol. 3, pp. 1305 – 1308.

[16] N. Stamenkovic: Digital Filter Implementation using RNS-binary Arithmetic, LAP Lambert Academic Publishing, 2014.

[17] S. Bi, W.J. Gross, W. Wang, A. Al-Khalili, M.N.S. Swamy: An Area-reduced Scheme for Modulo 2n-1 Addition/subtraction, 5th International Workshop on System-on-chip for Real-time Applications, Banff, Alberta - Canada, 20 – 24 July 2005, pp. 396 – 399.

[18] S. Veeramachaneni, L. Avinash, K.M. Kirthi, M.B. Srinivas: A Novel High-speed Binary and Gray Incrementer/decrementer for an Address Generation Unit, International Conference on Industrial and Information Systems, Peradeniya, Sri Lanka, 09 – 11 Aug. 2007, pp. 427 – 430.