Scientific Journal of Gdynia Maritime University

No. 124/22, 42–56
ISSN 2657-6988 (online)
ISSN 2657-5841 (printed)
DOI: 10.26408/124.04

Submitted: 02.06.2022
Accepted: 21.07.2022
Published: 31.12.2022

HIGH-SPEED BINARY-TO-RESIDUE CONVERTER DESIGN USING 2-BIT SEGMENTATION OF THE INPUT WORD

Robert Smyk[1]*, Maciej Czyżak[2]

[1] Gdańsk University of Technology, 11/12 Gabriela Narutowicza Street, 80-233 Gdańsk, Poland, e-mail: robert.smyk@pg.edu.pl, ORCID 0000-0001-9365-4633

[2] The University of Applied Sciences in Elbląg, 1 Wojska Polskiego Street, 82-300 Elbląg, Poland, ORCID 0000-0002-2249-1619

*Corresponding author

Abstract: In this paper a new approach to the design of the high-speed binary-to-residue converter is proposed that allows the attaining of high pipelining rates by eliminating memories used in modulo *m* generators. The converter algorithm uses segmentation of the input binary word into 2-bit segments. The use and effects of the input word segmentation for the synthesis of converters for five-bit moduli are presented. For the number represented by each segment, the modulo *m* reduction using a segment modulo *m* generator is performed. The use of 2-bit segments substantially reduces the hardware amount of the layer of input modulo *m* generators. The generated residues are added using the multi-operand modulo *m* adder based on the carry-save adder (*CSA*) tree, reduction of the number represented by the output *CSA* tree vectors to the 2*m* range and fast two-operand modulo *m* additions. Hardware amount and time delay analyses are also included.

Keywords: Binary-to-residue conversion, residue number system, FPGA.

1. INTRODUCTION

The Residue Number System (*RNS*) [Szabo and Tanaka 1967; Cardarilli, Nannarelli and Re 2007; Luan 2014] is a non-weighted number system that allows the fast realization of addition, subtraction and multiplication without carries between the digits of the number. The *RNS* had its beginnings in ancient China, but renewed interest at the end of the 1950s arose when its application for multiplication and fault detection in computers was examined [Szabo and Tanaka1967]. There were also attempts to design *RNS* arithmetic units for general-purpose computers, but difficulties in the realization of operations like division, sign determination, magnitude comparison and conversion to weighted systems limited the use of the *RNS* to the selected areas of cryptography and digital signal processing, where it could be useful for high-speed signal processors. The other applications are in low-power [Cardarilli, Nannarelli and Re 2007] and fault-tolerant arithmetics [Luan 2014]. Usually, the input to residue processors is encoded in the weighted system,

such as the natural binary system or two's complement number system; therefore, as the first step, the conversion to *RNS* has to be performed. Several converters have been presented in the literature [Alia and Martinelli 1990; Premkumar 2002; Piestrak 1994b; Czyżak 2004; Czyżak and Smyk 2008].

In this work we present a new design method for a high-speed converter for five-bit moduli. The aim of the work was to obtain a converter structure that did not limit the attainable pipelining rate. The memories were replaced by fine-gained division of the input word referred to as segmentation. The converter algorithm was based on segmentation of the input binary word into of 2-bit segments. For modulo *m* reduction of the number represented by each segment, small 2-bit modulo *m* generators were used. They performed the computation of the residue represented by the given segment using the idea of the look-up table with the 2-bit segment as the address and the residue as the addressed content. The look-up table was implemented as the block of five logic functions of two variables. The obtained residues for all segments were added using the multi-operand modulo *m* adder based on the carry save adder (CSA) tree with the successive modulo 2*m* reduction of the number represented by the *CSA* tree output vectors and the final two-operand modulo *m* adder.

Section 2 presents the definition of the *RNS*, Section 3 the details related to the binary to *RNS* (*B/RNS*) conversion, multioperand addition and segmentation, while Section 4 details the hardware implementation of the *B/RNS* converter and its properties related to the hardware amount and delay.

2. RESIDUE NUMBER SYSTEM

The Residue Number System is determined by its base $B = \{m_1, m_2, \dots, m_n\}$, where m_i, $i = 1,2,3, \dots, n$ are nonnegative integers termed the moduli. The number range of the system is $M = \prod_{i=1}^{n} m_i$. If the moduli are pairwise relatively prime, *i.e.* if $\gcd(m_j, m_k) = 1$, $j \neq k$, $j,k = 1,2, \dots, n$, then every integer X from $[0, M-1]$, can be represented by the *n*-tuple $(x_1, x_2, \dots, x_n)$, where $x_i = |X|_{m_i}$, in a one-to-one correspondence manner. The residue operations can be defined as

$$(x_1, x_2, \dots, x_n) \otimes (y_1, y_2, \dots, y_n) = (z_1, z_2, \dots, z_n), \tag{1}$$

where $z_i = |x_i \otimes y_i|$, and $\otimes$ may denote addition, subtraction or multiplication. As it can be seen from the above formula the operations are performed in small integer rings $R(m_i)$, $i = 1,2, \dots, n$. The condition of mutual primality assures that the mapping between the ring modulo M and the direct sum of rings, $i = 1,2, \dots, n$ is isomorphic. This mapping can be performed using the Chinese Remainder Theorem [Szabo 1967] or the mixed-radix conversion or the core function [Miller 1986]. The sign of the number is encoded in the following manner. For signed

numbers denoted as X, if M is even, $X = N$ for $N < \frac{M}{2}$, and $X = N - M$. If M is odd, $X = N$ for $N < \frac{(M-1)}{2}$, and $X = N - M$, if $N \geq \frac{(M-1)}{2}$.

3. BINARY-TO-RESIDUE CONVERSION

The binary-to-residue conversion is the process of finding the set of residues, *i.e.* the residue representation $\left(|A|_{m_1}, |A|_{m_2}, \ldots, |A|_{m_n}\right)$, for the number A represented in a certain binary code, *e.g.* the natural binary code. Below we shall consider the principles of conversion algorithms for unsigned numbers. Assume that a non-negative integer A from $\lfloor 0, 2^p - 1 \rfloor$ is represented in the natural binary code $A \leftrightarrow \mathbf{A} = \left(a_{p-1}, a_{p-2}, \ldots, a_0\right)$, where the bits $a_i \in \{0,1\}$. Then the residue $|A|_{m_i}$ can be computed as

$$|A|_{m_i} = \left|\sum_{i=0}^{p-1} a_i \, 2^i\right|_{m_i} = \left|\sum_{i=0}^{p-1} a_i \left|2^i\right|_{m_i}\right|_{m_i}, \; i = 1,2,\ldots,n \tag{2}$$

The interval $\lfloor 0, 2^p - 1 \rfloor$ can be converted to the subrange of the *RNS* number range M. The most straightforward conversion method relies upon the implementation of the direct mapping A→ $|A|_{m_i}$, $i = 1,2,\ldots,n$. This mapping can be implemented by using n look-up tables in parallel. As look-up tables the memories can be used or blocks of logic functions (Fig. 1).

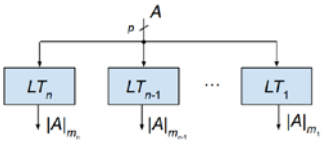

Fig. 1. *B/RNS* converter based on direct mapping

This approach can be used in practice for $p \leq 10$. Generally, it depends upon the time delay $t_{LT}(2^p \times \lceil \log m \rceil)$ of the applied memories with respect to their operation in the pipelined mode. Their delay may considerably reduce the achievable pipelining rate. In order to attain higher pipelining rates, components with a smaller delay have to be used. This can be attained in various ways.

The hardware implementation of *B/RNS* converters by (2) calls for calculation of $\left|2^i\right|_{m_j}$, $i = 1,2,\ldots,p-1$, $j = 1,\ldots,n$ and addition of the residues $\left|2^i\right|_{m_j}$. The conversion process can be divided into two stages, with the first one termed preprocessing and the second one multioperand modulo m_i addition for i-th channel (Fig. 2). The preprocessing encompasses the computation of $\left|2^i\right|_{m_j}$,

$i = p_1, \ldots, p, j = 1, \ldots, n$ for individual i or certain groups of the input word A termed segments. However, some initial j up to p_1 computation of initial $\left|2^i\right|_{m_j}$ is not needed for a certain number p_1 of least significant digits, if their sum does not exceed $m_j - 1$. In Fig. 2 the principle of segmentation process is illustrated. The input word is divided into the first segment with a length q_0 that represents a number smaller than m, certain number of regular q -bit segments and the final segment with a binary length $q_F < q$. For each q -bit segment a separate modulo m_j generator can be used. As stated above, there is no need for the modulo generator for the segment of the least significant bits, when its binary length $q_0 \leq \lceil \log m_j \rceil$.

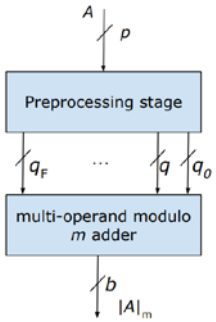

Fig. 2. The general scheme of one channel of the *B/RNS* converter

Source: own study.

Each $\left|2^i\right|_{m_j}$ has its $b = \lceil \log_2 m_j \rceil$-bit binary representation $Y_j^i = \left|2^i\right|_{m_j} \leftrightarrow (y_{b-1}, y_{b-2}, , \ldots, y_0)$. So, if $a_i = 1$ in (2) Y_j^i is the *i*-th addend. This idea is called wire splitting. However, in this case we have $p - 1$ operands to be added with their consecutive addition modulo m_j. However, the introduction of $q > 1$ allows the reduction of the number of addends when computing (2).

The computation of $\left|2^i\right|_{m_j}$ or the residues represented by the individual segments does not have to be used in certain special cases. The first one is when the moduli have the special form $m_k = 2^k - 1$. Avizienis [Avizienis 1964; 1971] proved that for such moduli the binary input word A can be divided into k-bit segments that can be directly added without the reduction modulo $2^k - 1$ for each segment. This is due to the property $\left|2^i\right|_{2^k-1} = \left|2^{j+ik}\right|_{2^k-1}$ and $\left|2^k\right|_{2^k-1} = 1$. This is the s. c. periodicity property. Piestrak [Piestrak 1994a,b] generalized this property to the moduli of other forms by using the periodicity $P(m)$ or half-periodicity $HP(m)$ of the series $\left|2^i\right|_{m_j}$.

Periodicity denotes that $\left|2^i\right|_m$ and $\left|2^{j+lP(m)}\right|_m$, $l = 0,1,2, \ldots$ have the same residue. The periodicity property allows us to divide the input word into segments

of $P(m)$ - length which can be directly added without generation of the residue corresponding to the sum of the given segment. This method is practical when $P(m)$ for the moduli of interest is small. But for 5- or 6-bit modli the dispersion of $P(m)$ values make this approach impractical. Certain moduli have a $P(m)$ close to m [Czyżak 2013]. The above consideration may lead to the conclusion that the segment length q, number of segments n_s and complexity of individual q-bit segment modulo m generators and the resulting number of addends using the multi-operand modulo m adder should be balanced. It seems that for 5-bit moduli, the advantageous segment lengths are $q = 2,3,4,5$. We remark that the first segment with a $\lceil \log m \rceil - 1$ bits length eliminates the need for the modulo m generator for this segment, because the number represented by this segment is the residue modulo m itself.

3.1. Multioperand modulo m addition

The multi-operand modulo m addition can be achieved in various ways. The four basic schemes for hardware n-operand modulo m adders (*MOMA*) are given in Figure 3. The first structure is the tree of $(n-1)$ two-operand modulo m adders (*MA*). The second structure uses the first stage of the multi-operand binary adder (*MOBA*). Such an adder is usually implemented as the n-operand *CSA* tree (Fig. 4) followed by the carry-propagate adder (*CPA*), referred to here as the binary adder (*BA*). The obtained binary sum S of the residues is further reduced by the use of the final modulo m generator. For a smaller S, it may be implemented as one $\lceil \log S \rceil$-bit look-up table. The third structure can be used in the case when $\lceil \log S \rceil > l_{LUT}$, where l_{LUT} is the allowable length of the look-up table address. In this case the division of S into two segments can be applied. In this case one *LUT* is addressed by l_{LUT} most significant bits of S, and the remaining $\lceil \log S \rceil - l_{LUT}$ bits constitute the second operand of the two-operand modulo m adder. The fourth structure makes use of the n-operand *CSA* tree with the reduction of the sum represented by the carry and save vectors to the $2m$ range with the successive two-operand modulo m adder.

Because the fourth approach will be used in the implemented converter, it should be reviewed. Let the sum $C + S$ and let $C \leftrightarrow c = (c_{k_1+1}, c_{k_1}, c_{k_1-1}, \dots, c_1, 0)$ and $S \leftrightarrow s = (s_{k_1}, s_{k_1-1}, \dots, s_1, 0)$. In the approach presented by Piestrak [Piestrak 1994a,b], each of the vectors, c and s, are divided into two segments $c = (c_H, c_L)$, $s = (s_H, s_L)$, so that the value represented jointly by c_L and s_L fulfils the relationship $C_L + S_L < m$. Moreover, $|C_H + S_H|_m$ can be computed by the look-up using the vector $w = (c_H, s_H)$ being the concatenation of c_H and s_H as the address of the look-up table. The length of w depends upon the number of operands of the *CSA* tree and the value of m with respect to $2^{\lceil \log m \rceil}$ [Piestrak 1994a,b]. Since the sum of obtained residues does not exceed $2m$, the final two-operand modulo m adder can be applied.

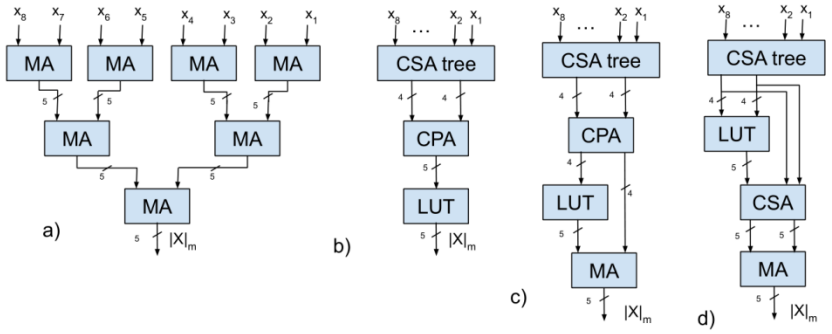

Fig. 3. a) Multi-operand modulo *m* adder based on two-operand modulo adders (*MA*), b) *MOMA* based on *MOBA* and direct modulo *m* reduction using single *LUT*, c) *MOMA* based on *MOBA* and segmentation of the binary sum and modulo *m* reduction, d) *MOMA* based on direct *CSA* tree output segmentation

Source: own study.

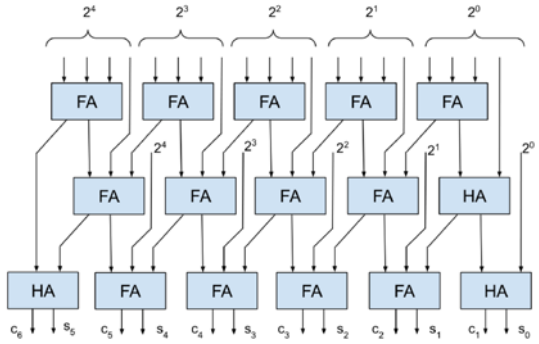

Fig. 4. Five operand 5-bit CSA tree

Source: own study.

3.2. Segmentation

The number of segments can be determined as:

$$n_s = \left\lfloor \frac{p-(\lceil \log_2 m \rceil - 1)}{b} \right\rfloor + \left\lceil \frac{|p-\lceil \log_2 m \rceil + 1|_b}{b} \right\rceil + 1 \tag{3}$$

The set of segments, starting from the least significant position, consists of the first $\lceil \log_2 m \rceil - 1$ bit segment, a group of $\left\lfloor \frac{p-(\lceil \log_2 m \rceil - 1)}{b} \right\rfloor$-bit segments and the additional segment of the length $b_n = |l - \lceil \log_2 m \rceil + 1|_b$. For each segment with the exclusion of the first segment, a small *b*-bit modulo *m* generator is applied (or b_n-bit generator for the most significant segment provided that it exists). The obtained $n_s - 1$ words may constitute input operands for the *MOMA*.

The numbers of segments obtained as the result of the segmentation process for the chosen *l*, *m*, and *b* are given in Table 1.

Table 1. Numbers of segments for 5-bit modulus and chosen *l* and *b*

l	11	12	13	14	15	l6	24	32
***b* = 1**	7	8	9	10	11	12	20	28
***b* = 2**	4	5	5	6	6	7	11	15
***b* = 3**	3	4	4	4	5	5	8	10
***b* = 4**	3	3	3	4	4	4	6	8

Source: own study.

For example, using $b = 4$, $m = 29$ and $p = 15$ in (3) we receive

$$n_s = \left\lfloor \frac{15-(5-1)}{4} \right\rfloor + \left\lceil \frac{|15-5+1|_4}{4} \right\rceil + 1 = 4 \tag{4}$$

We shall present the procedure of calculation of values of segments x_i, x_{i+1} for the 12-bit input operand $A \leftrightarrow \{a_{11}, a_{10}, \ldots, a_0\}$ in the following example.

Example 1. Calculating the values of segments a_i, a_{i+1} for *m*=29

$A_{s_0} = \sum_{i=0}^{3} \left| a_i 2^i \right|_m$, $A_{s_1} = \sum_{i=4}^{5} \left| a_i 2^i \right|_m$, $A_{s_2} = \sum_{i=6}^{7} \left| a_i 2^i \right|_m$, $A_{s_3} = \sum_{i=8}^{9} \left| a_i 2^i \right|_m$ and $XA_{s_4} = \sum_{i=10}^{11} \left| a_i 2^i \right|_m$.

We can divide it as follows: the segment s_0 represents first 4 bits with indexes $s_0 = \{a_3, \ldots, a_0\}$, and the remaining segments $s_1 = \{a_5, a_4\}$, $s_2 = \{a_7, a_6\}$, $s_3 = \{a_9, a_8\}$ and $s_4 = \{a_{11}, a_{10}\}$ organized by two bits. The procedure of the block conversion for the number $A = 3545 = \{1,1,0,1,1,1,0,1,1,0,0,1\}$ is depicted in Figures 5a and 5b.

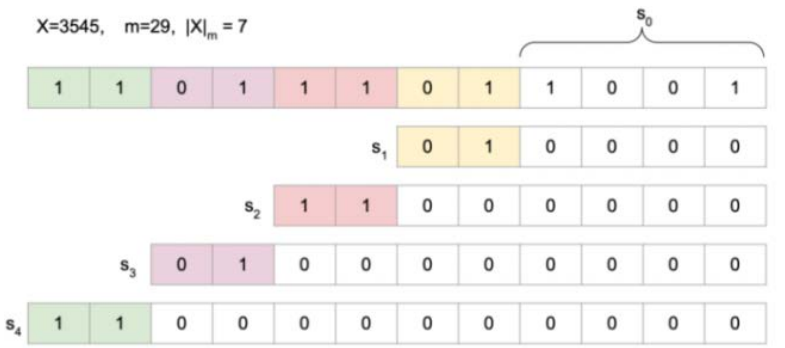

Fig. 5a. The principle of splitting the input operand during the B / RNS conversion

Source: own study.

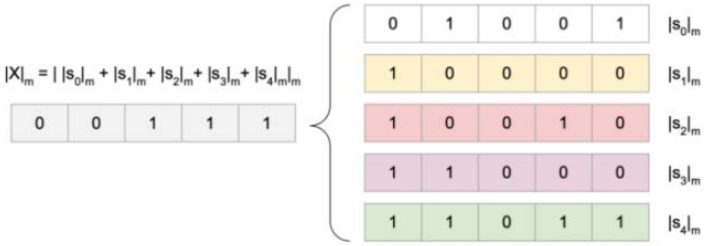

Fig. 5b. The principle of calculating the sum of the segments

Source: own study.

The individual blocks can be treated as vectors of bits shifted relative to each other at first by 4-bits and then by 2-bits. The s_0 block does not require modulo reduction, because it is 4-bit, i.e. with a value lower than the smallest moduli equal to 17. The other blocks, $s_1 - s_4$, are reduced by modulo *m*. In fact this operation can be performed using simple logic functions. A value of particular segments reduced by modulo m should be $A_{s_0} = 9$, $A_{s_1} = 16$, $A_{s_2} = 18$, $A_{s_3} = 24$, $A_{s_4} = 27$. As a result of the calculations, we get $|A|_m = \left|A_{s_0} + A_{s_1} + A_{s_2} + A_{s_3} + A_{s_4}\right|_m = 7$.

4. B/RNS CONVERTER ARCHITECTURE

In Figure 4 the general architecture of a *B/RNS* converter using segmentation is shown. A *B/RNS* converter is designed using segment modulo generators (*SMG*), the *MOMA* based on *MOBA* in the *CSA* tree form and the final modulo generator (*FMG)*. It is assumed that the input mapping is performed with the use of small modulo generators realized with the use of logic functions. At the first stage the processed number is decomposed into small segments with the segment length $b = 2$ and subsequently the residues of the numbers represented by the individual segments are computed.

For the considered designs, the following assumptions have been made:

- all converter stages are realized using simple logic, easy to design with the *ASIC* approach. Latches can be placed between every layer of gates to attain a high pipelining rate. However, high-speed solution may require the realization of *LUT* as the set of logic functions;
- all converter channels have the same binary width, *a*. Every channel may have the same structure for the given length of the input word. Using the realistic assumption, we made use of small moduli, $b = 5$ or $b = 6$ bits, the *RNS* range may reach a significant value of about 2^{92} bits;
- the segment size *b* at the *FMG* input is the same in each channel.

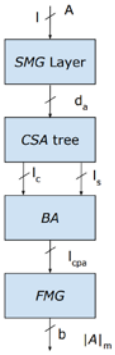

Fig. 4. The general architecture of one converter channel

Source: own study.

4.1. Two bit SMG

For each two-bit segment we can easily determine the logic functions of two variables and compute their hardware amount. We can also calculate the number of inputs to the given weight 2^i to the *MOMA*. Now we shall analyse the hardware amount of the individual generators for 2-bit segments. Each segment generator can be treated as a five-logic function of two variables. First we show the logic function design for individual generators.

In Table 2 the principle of creating logic functions (*LF*) for the 2-bit *SMG* is given.

Table 2. The principle of the realization of 2-bit segment modulo generator

b_{i+1}	b_i	**Residues for individual** b_{i+1}, b_i	y_4	y_4	y_4	y_4	y_4
0	0	$\|\|0 \cdot 2^{i+1}\|_m + \|0 \cdot 2^i\|_m\|_m$	0	0	0	0	0
0	1	$\|\|0 \cdot 2^{i+1}\|_m + \|1 \cdot 2^i\|_m\|_m$	$y_{4,1}$	$y_{3,1}$	$y_{2,1}$	$y_{1,1}$	$y_{0,1}$
1	0	$\|\|1 \cdot 2^{i+1}\|_m + \|0 \cdot 2^i\|_m\|_m$	$y_{4,2}$	$y_{3,2}$	$y_{2,2}$	$y_{1,2}$	$y_{0,2}$
1	1	$\|\|1 \cdot 2^{i+1}\|_m + \|1 \cdot 2^i\|_m\|_m$	$y_{4,3}$	$y_{3,3}$	$y_{2,3}$	$y_{1,3}$	$y_{0,3}$

Source: own study.

In Table 3 an example of the logic functions for $i = 4$, $m = 29$ is shown. We can see that each Y function may have 8 possible logic functions of variables b_{i+1}, b_i in binary form.

Table 3. The values of sums represented by the segment reduced modulo m for $i = 4$, $m = 29$

b_5	b_4	**Residues for individual** b_{i+1}, b_i	Y_{seg}	y_4	y_4	y_4	y_4	y_4
0	0	$\|\|0 \cdot 2^5\|_{29} + \|0 \cdot 2^4\|_{29}\|_{29}$	0	0	0	0	0	0
0	1	$\|\|0 \cdot 2^5\|_{29} + \|1 \cdot 2^4\|_{29}\|_{29}$	16	1	0	0	0	0
1	0	$\|\|1 \cdot 2^5\|_{29} + \|0 \cdot 2^4\|_{29}\|_{29}$	3	0	0	0	1	1
1	1	$\|\|1 \cdot 2^5\|_{29} + \|1 \cdot 2^4\|_{29}\|_{29}$	19	1	0	0	1	1

Source: own study.

Table 4 gives all possible logic functions in the minimized form for m = 29 that can occur within 2-bit *SMG*.

Table 4. Minimized forms of the individual logic functions in 2-bit *SMG*

fl0	fl1	fl2	fl3	fl4	fl5	fl6	fl7
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	$b_1 b_0$	$b_1 \overline{b_0}$	b_1	$\overline{b_1} b_0$	b_0	$\overline{b_1} b_0 + b_1 \overline{b_0}$	$b_1 + b_0$

Source: own study.

In Table 5 shows the number of bits with given weight 2^i obtained for all 2-bit *SMG* generators for individual p = 12,…,32

Table 5. Number of bits with individual weights of 2^i for m = 29

p	2^4	2^3	2^2	2^1	2^0
12	5	4	3	5	4
16	7	5	5	7	6
24	10	10	9	10	10
32	13	13	12	13	13

Source: own study.

Using the data from Table 5, remembering that every triple operand set requires one *CSA* block, the numbers of required *CSA* blocks are given in Table 6.

Table 6. Number of *CSA* tree operands

l	12	14	16	24	32
2^0	4	5	6	10	14
2^1	4	5	6	10	14
2^2	3	3	4	6	8
2^3	2	2	2	2	2
2^4	2	2	2	2	2

Source: own study.

Knowing the number of *CSA* tree operands, we can determine the number of levels in the *CSA* tree (Table 7). The appropriate recursive formula binding the number of levels and operands is given in [Huang 1979].

Table 7. Number of levels in the *CSA* tree

l	12	14	16	24	32
2^0	2	3	3	5	6
2^1	2	3	3	5	6
2^2	1	1	2	3	4
2^3	1	1	1	1	1
2^4	1	1	1	1	1

Source: own study.

Table 8. Example hardware amounts *A* for modulus *m* = 29 and input word lengths p = 12, 14, 16, 24, 32

p	segments	A [GE]
12	4	32.64
14	5	41.62
16	6	52.26
24	10	80.52
32	14	116.48

Source: own study.

5. ARCHITECTURE AND IMPLEMENTATION

To achieve multi-operand modulo *m* addition, the structure from Figure 2d has been used. Figure 4 depicts the high-level architecture of the converter channel while Figure 5 shows a more detailed, low-level architecture of *B/RNS* converter channel.

In the first stage of the converter, the look-up tables *LF*1 - *LF*4 determine the residues for the 2-bit segments of the input word.

Next the residues are summed up in the *CSA* tree. The output is then reduced to the range [0,2*m*-1]. The output carry vector, *c* and sum vector, *s*, are divided into two parts in accordance with the algorithm given in the previous section. The output carry vector *c* is divided into two subvectors, c_H and c_L. Similarly, the *s* vector is divided into two subvectors, s_H and s_L, representing the numbers S_H and S_L. The binary length of vectors c_L and s_L are selected in such a way so that $c_L + s_L < m$, hence the sum $S = |C_H + S_H|_m + C_L + S_L < 2m$ at the *CSA*1 output is smaller than $2m$. The look-up table (*LT*) addressed by vector $w = (c_H, s_H)$ computes the residue represented by this vector and then in *CSA*1 it is added with the numbers represented by c_L and s_L. The sum received at the *CSA*1 output is denoted as X_2. Then *CSA*2 is used to compute $X_1 + X_2 - m$. Finally the $X_1 + X_2$ and *CSA*2 output vectors are added in the two parallel binary ripple-carry adders, *CPA*1 and *CPA*2. If the *CPA*2 output sum is positive, it is selected by the *MUX*1 as the correct result, else the *CPA*1 output is chosen.

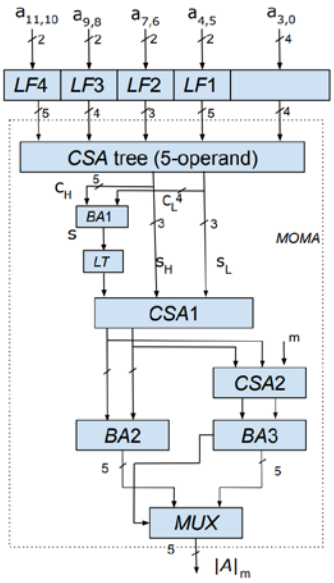

Fig. 5. The architecture of one channel of 12-bit converter for m = 29

Source: own study.

5.1. Analysis of converter hardware amount

In this subsection we shall analyze the converter hardware amount and delay. The converter hardware amount $A_{conv}(p, B)$ can be expressed as

$$A_{conv}(p, B) = \sum_{i=1}^{n} A(p, m_i) \tag{6}$$

where $B = \{m_1, m_2, \dots, m_n\}$. Using $b = \lceil \log m \rceil$ $A(p, m_i)$ can be represented as

$$\begin{aligned} A(p, m_i) = A_{SEG(p,m_i)} + A_{CSAt(m_s,b)} + A_{BA1(b)} + A_{LT(2^b \times b)} \\ + A_{CSA1(3,b)} + A_{CSA2(3,b+1)} + A_{BA2(b)} + A_{BA3(b+1)} + A_{MUX(b)} \end{aligned} \tag{7}$$

where

$$A_{CSA(m,b)} = (m_s + 2)b \cdot A_{FA2d1} \tag{8}$$

$$A_{BA(b)} = (b - 1)A_{FA2d1} + A_{HA2d1} \tag{9}$$

$$A_{MUX(b)} = b \cdot A_{MUX(2-1)} \tag{10}$$

We first analyze the hardware amount of the segment modulo m generators layer. The individual functions have the following hardware amounts:

Table 9. Hardware amounts of individual logic functions

	$fl1$	$fl2$	$fl3$	$fl4$	$fl5$	$fl6$	$fl7$
Logic function	$b_1 b_0$	$b_1 \overline{b_0}$	b_1	$\overline{b_1} b_0$	b_0	$\overline{b_1} b_0 + b_1 \overline{b_0}$	$b_1 + b_0$
Amount [GE]	1	2	0	1	0	2	2

Source: own study.

Table 10. Hardware amounts for individual components of a single channel *B/RNS* converter

$A_{SEG(12,29)}$	$A_{CSAt(5,5)}$	$A_{BA1(3)}$	$A_{LT(2^4\times5)}$	$A_{CSA1(3,5)}$	$A_{CSA2(3,6)}$	$A_{BA2(5)}$	$A_{BA3(6)}$	$A_{MUX(5)}$
4.2*FA*	3.5*FA*	2*FA*+1*HA*	6.42*FA*	1.5*FA*	1.6*FA*	4*FA*+1*HA*	5*FA*+1*HA*	3.5*FA*

Source: own study.

Based on (7) and Table 10, the total hardware complexity for one channel of the *B/RNS* converter is $A(p, m_i) = 46.92FA$.

5.2. Analysis of converter time delay

The time delay of one channel of *B/RNS* converter t_{convD} can be represented as

$$t_{convD} = t_{seg_i} + t_{CSAt(l_s)} + t_{BA1(b)} + t_{LT(2^4\times5)} + \\ + t_{CSA1(1)} + t_{CSA2(1)} + t_{BA3(b+1)} + t_{MUX(2-1)} \tag{11}$$

$t_{CSA1(l_s)} = l_s \cdot t_{FA}$ where l_s is the number of the *CSA* layers. This can be computed using the formula from [Huang 1979].

Table 11. Full adder time delay parameters with fan-out = 1

	t_{AC0}	t_{BC0}	t_{CIC0}	t_{AS}	t_{BS}	t_{CIS}
Delay [ns]	0.23	**0.269**	0.153	0.229	0.265	0.182

Source: own study.

Based on Table 11, we assume $t_{FA} = 0.269ns$. The delay of the *BA* in the ripple-carry form can be determined as

$$t_{BA(b)} = \max(t_{AC0}^{HA}, t_{BC0}^{FA}) + (b-2)\, t_{CIC0}^{FA} + \max(t_{CIC0}^{FA}, t_{AS}^{FA}, t_{BS}^{FA}) \tag{12}$$

After substituting the values from Table 11 we obtain $t_{BA(6)} = 0.825\, ns$ and that gives $t_{BA(5)} = 3.63$. Such a delay allows more than a 1GHz pipelining frequency.

Table 12. Delays of *B/RNS* converter individual components in t_{FA}

t_{seg_i}	$t_{CSAt(5,5)}$	$t_{BA1(b)}$	$t_{LT(2^4\times5)}$	$t_{CSA1(3,5)}$	$t_{CSA2(3,6)}$	$t_{BA3(6)}$	$t_{MUX(5)}$
0.598	3	3.06	1.28	1	1	3.63	0.7

Source: own study.

Total delay of the converter is $t_{convD} = 14.268t_{FA}$, and $t_{convD} = 3.83ns$.

6. CONCLUSIONS

This paper proposes the design of a high-speed *B/RNS* converter. The converter architecture allows, theoretically, the application of high pipelining speeds at the *FA* level, if needed. This is obtained by segmentation of the input word into 2-bit segments. In the presented simplified solution, a 6-bit modulo adder is treated as a hardware unit with respect to the pipelining, that limits the pipelining speed at 1 GHz versus about 4 GHz for *FA* level pipelining. The converter input layer that performs the initial modulo generation for fragments of the input word has been designed using 2-bit segments that considerably simplifies the design of this layer. The whole segmentation procedure has been shown in detail and illustrated with corresponding examples.

REFERENCES

Alia, G., Martinelli, E., 1990, *Short Note: VLSI Binary-Residue Converters for Pipelined Processing*, The Computer Journal, Vol. 33, No. 5, pp. 473–475.

Avižienis, A., 1964, *A Set of Algorithms for a Diagnosable Arithmetic Unit*, Jet Propulsion Laboratory, California Institute of Technology, USA.

Avižienis A., 1971, *Arithmetic Codes: Cost and Effectiveness Studies for Applications in Digital System Design*, IEEE Trans. Comput., Vol. C-20, pp. 1322–1331.

Cardarilli, G.C., Nannarelli, A., Re, M., 2007, *Residue Number System for Low-power DSP Applications*, Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, IEEE, pp. 1412–1416.

Czyżak, M., 2004, *High-Speed Binary-to-Residue Converter with Improved Architecture*, 27th International Conference on Fundamentals of Electrotechnics and Circuit Theory, Gliwice-Niedzica, pp. 431–436.

Czyżak, M., 2013, *Digital Structures for High-Speed Signal Processing*, Politechnika Gdańska, pp. 94–95.

Czyżak, M., Smyk R., 2008, *High-Speed FPGA Pipelined Binary-to-Residue Converter*, Poznan University of Technology, Academic Journals Electrical Engineering, No. 58, pp. 65–72.

Huang, K., 1979, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley & Sons.

Luan, Z., Chen, X., Ge, N., Wang, Z., 2014, *Simplified Fault-Tolerant FIR Filter Architecture Based on Redundant Residue Number System*, Electronics Letters, Vol. 50, No. 23, pp. 1768–1770.

Miller, D.D., Altschul, R.E., King, J.R., Polky, J.N., 1986, *Analysis of the Residue Class Core Function of Akushskii, Burcev, and Pak. In Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, pp. 390–401.

Piestrak, S.J., 1994a, *Design of High-Speed Residue-to-Binary Number System Converter Based on Chinese Remainder Theorem*, Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 508–511.

Piestrak, S.J., 1994b, *Design of Residue Generators and Multioperand Modulo Adders Using Carry-Save Adders*, IEEE Trans. Comp., Vol. 43, pp. 68–77.

Premkumar, A.B., 2002, *A Formal Framework for Conversion from Binary to Residue Numbers*, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 49, No. 2, pp. 135–144.

Szabo, N.S., Tanaka, R.J., 1967, *Residue Arithmetic and its Applications to Computer Technology*, New York, McGraw-Hill.