POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 91 Electrical Engineering 2017

DOI 10.21008/j.1897-0737.2017.91.0014

Robert SMYK*
Maciej CZYŻAK*

HIGH LEVEL SYNTHESIS IN FPGA OF TCS/RNS CONVERTER

The work presents the design process of the TCS/RNS (two's complement–to–residue) converter in Xilinx FPGA with the use of HLS approach. This new approach allows for the design of dedicated FPGA circuits using high level languages such as C++ language. Such approach replaces, to some extent, much more tedious design with VHDL or Verilog and facilitates the design process. The algorithm realized by the given hardware circuit is represented as the program in C++. The performed design experiments had to show whether the obtained structures of TCS/RNS converter are acceptable with respect to speed and hardware complexity. The other aim of the work was to examine whether it is enough to write the program in C++ with the use of basic arithmetic operators or bit–level description is necessary. Finally, we present the discussion of results of the TCS/RNS converter design in Xilinx Vivado HLS environment.

KEYWORDS: high–level synthesis, residue number system, FPGA, C++ language, two's complement–to–residue converter

1. INTRODUCTION

The simple FPGA circuits can be designed using schematic approach but more complex ones require the use of hardware description languages as VHDL or Verilog. The FPGA architecture can be described using the structural or behavioural approach. The former requires the definition of components and appropriate signals that connect blocks of the architecture. However VHDL is a high level language but it is more adapted to the description of hardware and not of algorithms. On the other hand a wide number of well–known and verified algorithms have been implemented in C/C++. It makes that the use of C/C++ is a more natural approach. In last twenty years high level synthesis (HLS) techniques have been extensively studied and a number of HLS tools have been developed [1]. HLS, also known as behavioural synthesis, is the technology which automatically translates behavioural design descriptions in C/C++ into

* Gdansk University of Technology.

register transfer level (RTL). The HLS approach considerably reduces the development time from weeks to days, but the price paid may be the greater hardware complexity of the obtained architecture. So it seems indispensable to verify experimentally how the C++ constructs map onto the FPGA architecture and identify the operations which lead to the substantial increase of hardware complexity. In this work we have considered the use of the HLS technique for the design of two's complement–to–residue converter.

The Residue Number System(RNS) [2–4] is the non–weighted number system that allows for fast realization of addition, subtraction and multiplication without carries between the digits of the number. The RNS had its beginnings in ancient China but the renewed interest arose at the end of 50's of XX century when its application to fault detection in computers was examined [2]. There were also attempts to design RNS arithmetic units for general–purpose computers but difficulties in realization of operations such as division, sign detection, magnitude comparison and conversion to weighted systems have limited the use of the RNS to selected areas of cryptography and digital signal processing where it can be useful for high–speed signal processors. The other applications are in low–power and fault–tolerant arithmetics. Usually the input to residue processors is encoded in a weighted system such as the natural binary system or two's complement, therefore as the first step the conversion to the RNS has to be performed. Several converters were presented in the literature [4–8]. The DSP systems based on residue arithmetic are becoming more complex, therefore the design methods are sought for that would speed up the design and testing process. One approach to attain this goal would be the use of high level FPGA synthesis. We have performed experiments in order to state which instructions and how should be used when describing an architecture to obtain an effective structure with respect to hardware complexity.

In Section 2 the residue number system is reviewed, in Section 3 we analyze the problem of two's complement–to–residue conversion. In Section 4 we give the converter algorithm and in Section 5 we present the results of high level synthesis of two's complement–to–residue converter.

2. THE RESIDUE NUMBER SYSTEM

The residue number system is determined by its base, $B=\{m_1,m_2,...,m_n\}$ where m_i, i = 1,2,3,...,n, are nonnegative integers termed the moduli. The number range M of the system is $M=\prod_{i=1}^{n} m_i$. If the moduli are pairwise relatively prime, *i.e.* if $\gcd(m_j,m_k)=1,\ j\neq k,\ j,k=1,2,..,n$, then every integer X from $[0,M-1]$, is represented by the n–tuple $(x_0,x_1,...,x_{l-1})$, where

$x_i = |X|_{m_i}$, in one–to–one correspondence manner.. The residue operations can be defined as $(x_1, x_2, ..., x_n) \otimes (y_1, y_2, ..., y_n) = (z_1, z_2, ..., z_n)$, where $z_i = |x_i \otimes y_i|_{m_i}$,and $\otimes$ may denote addition, subtraction or multiplication. As seen from the above formula the operations are performed in small integer rings $R(m_i)$, *i*=1,2,..., *n*. The condition of mutual primality assures that the mapping between the ring modulo M and the direct sum of $R(m_i)$, *i*=1,2,...,*n*. is isomorphic. This mapping can be performed using the Chinese Remainder Theorem or the mixed–radix conversion[2, 3].

3. TWO'S COMPLEMENT–TO–RESIDUE CONVERSION

The binary–to–residue conversion is the process of finding the set of residues, *i.e.* the residue representation $(|X|_{m_n}, |X|_{m_{n\ 1}}, ..., |X|_{m_1})$, for the number $X = (x_{l-1}, x_{l-2}, \ldots\ldots, x_0)$ represented in a certain *l*–digit binary code, $x_i \in (0,1)$, *eg.* the natural binary, one's complement or two's complement. Below we shall consider only two's complement representation and we assume that $|-2^l| + |2^l - 1| < M$.

$$|X|_m = \left|\sum_{i=0}^{l-1} x_i 2^i\right|_m = \left|\sum_{i=0}^{l-1} x_i |2^i|_m\right|_m \tag{1}$$

The hardware implementation (Fig. 1) of conversion by (1) requires, in general, the computation of $|2^i|_m$, $i = 0,1,2,...,l-1$, summation of $|2^i|_m$ for these *i*, for which $x_i \neq 0$ and the modulo *m* operation. The most direct approach to compute $|2^i|_m$ is s.c. wire splitting where the x_i wire is splitted, in general, into $\lceil \log m \rceil$ wires with each wire representing the power of 2^i that is present in the binary representation of $|2^i|_m$. This approach seems to be impractical for longer words due to the complex wiring and the large number of addends. Premkumar [6] proposed computation of $|2^i|_m$, instead of storing, but his approach leads to the structures with the large hardware amount.

Piestrak [5] shown that to determine the residue of the number represented by the given segment, the computation of $|2^i|_m$ can be avoided by using the property of periodicity or half–periodicity of the series $|2^i|_m$. Periodicity means that $\left|2^j\right|_m$ and $\left|2^{j+i\cdot P(m)}\right|_m$ have the same residues modulo *m*, where *P*(*m*) is called the period of the modulus.

In this approach the converted word is divided into segments of *P*(*m*)–bit length that can be directly added. This approach can be useful when only one residue is generated and *P*(*m*) is small. If we have a base consisting of 5–6–bit moduli this approach becomes impractical if *P*(m_i) are different for the individual moduli of the RNS base. The makes that the converters for the individual residue channels call for various hardware amounts and may have different delays.

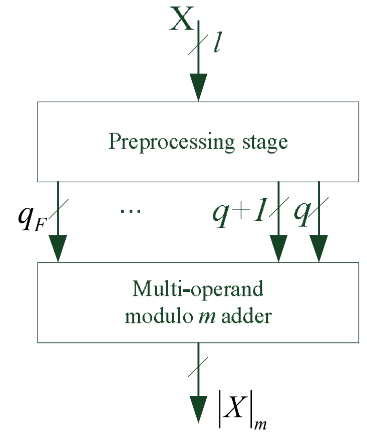


Fig. 1. The general scheme of one channel of the B/RNS converter, where $q = \lfloor \log m \rfloor$ is a length of segment and $q_F = (l - \lfloor (l-q)/n_s \rfloor \cdot n_s - q)$ is a length of the most significant segment and *l* is a length of the binary representation of *X*, n_s – number of segments

In order to diminish the number of addends in (1), we can divide the representation of *X* into segments with the first segment of $q = \lfloor \log_2 m \rfloor$ bits, $n_s - 2$ $\lfloor (l-q)/n_s \rfloor$–bit segments and one final (MSB) $q_F = (l - \lfloor (l-q)/n_s \rfloor \cdot n_s - q)$– bit segment. The first segment has usually $\lceil \log m \rceil - 1$ bitlength, so there is no need to use the modulo generation for this segment because it represents the residue modulo *m* itself. We can generate the residues for the individual segments and then compute their sum as

$$|X|_{m_i} = \left|\sum_{s=0}^{n_s-1} X_s\right|_{m_i} = \left|\sum_{s=0}^{n_s-1} |X_s|_{m_i}\right|_{m_i} = \sum_{s=0}^{n_s-1} \left|\sum_{i=s}^{s+l_s} x_i 2^i\right|_{m_i} \qquad (2)$$

where *s* is the index of segment, n_s – number of segments and l_s – is the segment length.

If l is sufficiently small (l <10÷12) the conversion can be performed, using only one segment, by memory look–up applying, for example, ROM $\left(2^l \times \lceil \log m \rceil\right)$, provided that the memory block will not limit the pipelining frequency. In FPGAs, which are our consideration, ROMs can be used that are placed outside the FPGA matrix or the decomposed memory represented by LUTs with the 4–6 bit address. Their use imposes a form of dividing the input word into smaller segments. However the problem becomes more difficult when we consider conversion of two’s complement numbers.

Assume that a signed integer X is represented in two's complement code using $1+1$–bit representation $X=\left(x_{1+1}, x_{1\text{-}1}, x_{1\text{-}2}, \ldots, x_0\right)$, $x_i \in \{0,1\}$. Signed integers are usually represented in the RNS in such a manner that for M odd, the number range is $[-(M-1)/2,(M-1)/2]$ and for M even, $[-M/2, M/2-1]$. Assume henceforth M even without loss of generality. Then the interval $[0, 2^{l-1}-1]$ will be converted into the part of the interval $[0, M/2-1]$ and the interval $[-2^{l-1}, -1]$ is converted to the subrange of $[-M/2, M-1]$. For $X < 0$ we have to determine $\left|M - X\right|_m$, that can be obtained as $m-\left|X\right|_{m_i}$.

For X<0 we want to represent $-X$ as a sum of negative or zero numbers represented by the consecutive segments. We have

$$-X = -X_{s-1} + (-X_{s-2}) + \ldots + (-X_0) \tag{3}$$

Then

$$\left|M - X\right|_{m_i} = \left|\left|0 - X_{s-1}\right|_{m_i} + \left|0 - X_{s-2}\right|_{m_i} + \ldots + \left|0 - X_0\right|_{m_i}\right|_{m_i}. \tag{4}$$

Moreover we have

$$\left|0 - X_{s-1}\right|_{m_i} = -\left|X_{s-1}\right|_{m_i} = m_i - \left|X_{s-1}\right|_{m_i} \tag{5}$$

In order to use this form for conversion we have first to recover the absolute value $|X|$ from X^* being the 2’s complement representation of $-X$. We have

$$X^* = 2^l - X, \tag{6}$$

hence

$$X = 2^{l+1} - X^*, \tag{7}$$

$$X = (2^{l+1} - 1 - X^*) + 1, \tag{8}$$

where the expression in parenthesis denotes the negation of X. Therefore once X is recovered form X^* we can perform

$$\left|M - X\right|_{m_i} = \left|\left|m_i - X_{s-1}\right|_{m_i} + \left|m_i - X_{s-2}\right|_{m_i} + \ldots + \left|m_i - X_0\right|_{m_i}\right|_{m_i} \tag{9}$$

4. TWO'S COMPLEMENT TO RESIDUE CONVERTER ALGORITHM

The presented converter utilizes the principle of segmentation of the input word. The input word is divided into segments of six–bit length with the possible exception of the first (msb) segment which can be shorter if the length of the input word is not an integer multiple of 6.

The input vector has a following form $(x_1,...,x_0)$. It is divided into segments in such a manner that each segment contains 5 bits of the x input vector and the sign bit s, for example, for 15–bits we have the vector

$$(\mathrm{s}, x_{14}, x_{13}, x_{12}, x_{11}, x_{10}, \mathrm{s}, x_9, x_8, x_7, x_6, x_5, \mathrm{s}, x_4, x_3, x_2, x_1, x_0)$$

The sign bits are used to signal that the given segment represents the negative number. If the msb bit of the input word is equal to 0, so the number is non negative, we use the formula

$$|A|_m = \left| \left|\sum_{i=10}^{15} x_i \cdot 2^i\right|_m + \left|\sum_{i=5}^{9} x_i \cdot 2^i\right|_m + \left|\sum_{i=0}^{4} x_i \cdot 2^i\right|_m \right|_m , \tag{10}$$

and for negative numbers

$$|A|_m = \left| \left|m - \sum_{i=10}^{15} x_i \cdot 2^i\right|_m + \left|m - \sum_{i=5}^{9} x_i \cdot 2^i\right|_m + \left|m - \sum_{i=0}^{4} x_i \cdot 2^i\right|_m \right|_m \tag{11}$$

Once the residues for the segments have been obtained their sum modulo m has to be determined by using a multi–operand modulo adder. Such adder can be realized as the tree of $n/2$ two operand adders or by performing first binary summation and next the modulo m reduction of the sum.

The formulas (10) and (11) can be implemented as the structure given in Fig. 2.

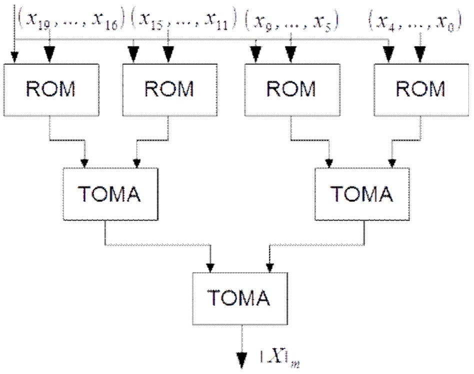
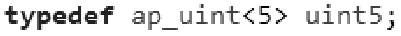


Fig. 2. TCS/RNS converter structure based on ROMs

5. HIGH LEVEL SYNTHESIS OF TCS/RNS CONVERTER

The synthesis of the presented converter has been carried out in the Xilinx HLS integrated environment. This approach radically shortens the project development time in FPGAs. The standard FPGA design process requires the description of circuit operation at RTL niveau with the use of the hardware description in VHDL or Verilog. This description can be automatically translated to the netlist. The focus of the netlist abstraction layer is to define the Boolean functionality of the design with appropriate area, performance and power, what is the final stage of an FPGA implementation. In case of the ASIC design the elaboration of appropriate masks is needed for the fabrication of the VLSI circuit. The essence of the approach related to the use HLS involves applying the high–level programming language such as C/C++ as well for design as for testing.

There is no need to simulate and test the algorithm outside the Xilinx environment. In the classical approach these steps are carried out externally and once the algorithm is deemed correct the design of FPGA implementation may start. Within the Xilinx HLS environment several programming mechanisms have been applied that facilitate high–level synthesis. The high–level synthesis requires an adequate description of the input and the output and internal registers of the system. A need emerged to introduce arithmetic types with the selectable bit length. In order to make it possible new parametrized class types have been introduced such as, for example, `ap_int`, defined in `<ap_int.h>`. The `ap_int` type is used to define input and output signals with the wordlength from 1 do 1024 bits. This class disposes over suitable constructors which are used to create objects representing system input and system output as well as internal signals. The parametrized type can be used directly or we can introduce a new name for the parametrized type as in Fig. 3. For example, we can define 5–bit `unsigned int type .` as `uint5`

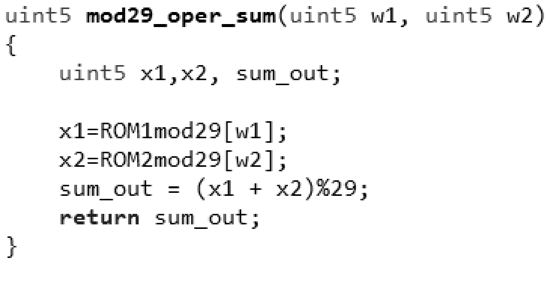


```
typedef ap_uint<5> uint5;
```

Fig. 3. Introduction of the new name for `ap_unt <5>`

The TCS/RNS converters were synthesized in Xilinx Vivado HLS using 6–bit modulo generators implemented with the use of ROMs and adder tree consisting of 5–bit Two–Operand Modular Adders (TOMA). Below we shall show results of experiments which have been performed in order to find such the programmatical description of the fragment of the converter that leads to the optimal converter architecture with respect to minimum hardware complexity. In system architectures that use the RNS, the crucial operation that considerably influences the complexity, is the modulo reduction operation. The first factor that determines its complexity is the binary size of the modulus. It can be a small modulus m_i, being one of the system moduli or the RNS number range M. For DSP systems with the moderate number range, m_i binary size belongs to [3, 12] and M to [20, 50]. The second factor is the binary size of the word to be reduced. The direct approach to perform modulo reduction operation is to carry out integer division and find the remainder. The C++ version used in Vivado HLS allows to use the standard modulo '%' operator. In the architecture being the result of HLS synthesis this operation is implemented with division and remainder determination. This, however, requires the integer divider in the system. The divider is usually iterative and introduces considerable delay. Such modulo reduction operation has a general character and can be performed independently of the relationship between the number to be reduced and the modulus. But in certain cases the modulo reduction can be significantly simplified. If, for example, $X<2m$, we compute $r = X \ mod \ m$ by calculating $d= X - m$, then if $d<0$, then $r = X$ else $r = d$. In the program only if–else instruction is needed. If X exceeds $2m$, nested if instruction can be used.

In Fig. 4 the C++ function is shown that corresponds to the block with two five–bit inputs *w1* and *w2* and five–bit output. The input signals *w1* and *w2* address ROM memories and *w1* and *w2* are treated as representations of the binary numbers. Next summation of the residues *x1* and *x2* is performed and subsequently the reduction modulo 29 using % operator is made. ROM memories are implemented as one–dimensional int tables `const uint5 ROM1mod29[32],ROM2mod29[32]`.



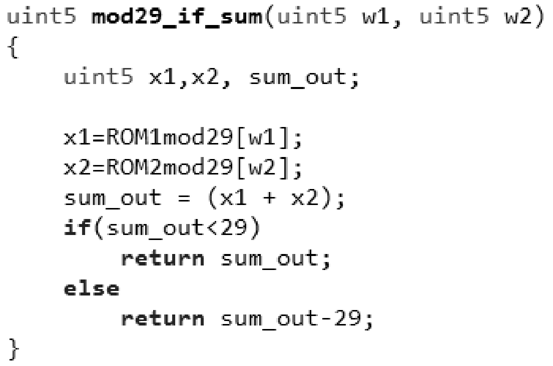
```
uint5 mod29_oper_sum(uint5 w1, uint5 w2)
{
    uint5 x1,x2, sum_out;

    x1=ROM1mod29[w1];
    x2=ROM2mod29[w2];
    sum_out = (x1 + x2)%29;
    return sum_out;
}
```

Fig. 4. B/RNS conversion for 10–bit word using memory look–up and modulo reduction with % operator

In the next experiment mod 29 operator (%) was replaced by if–else instruction (Fig. 5).

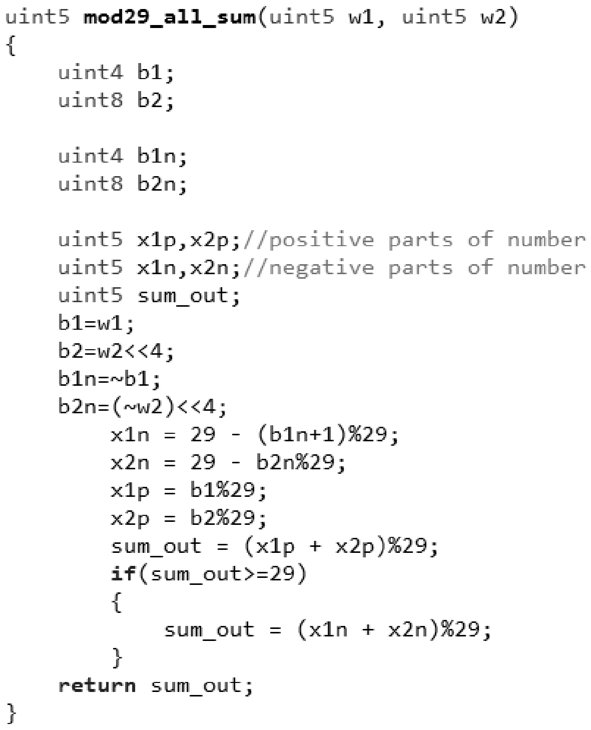
In the last experiment the direct structure based on (10) and (11) was benchmarked (Fig. 6).

The synthesis results of the above models are presented in Fig. 7. It can be observed that the use of C++ in–built operators leads to the more complex architecture (Fig. 7c). The best results have been obtained using ROMs and modulo reduction using if–else and subtraction. We can draw, at least in this case, a conclusion that direct high–level description may accelerate the implementation but the result is less effective with respect to hardware amount .



```
uint5 mod29_if_sum(uint5 w1, uint5 w2)
{
    uint5 x1,x2, sum_out;

    x1=ROM1mod29[w1];
    x2=ROM2mod29[w2];
    sum_out = (x1 + x2);
    if(sum_out<29)
        return sum_out;
    else
        return sum_out-29;
}
```

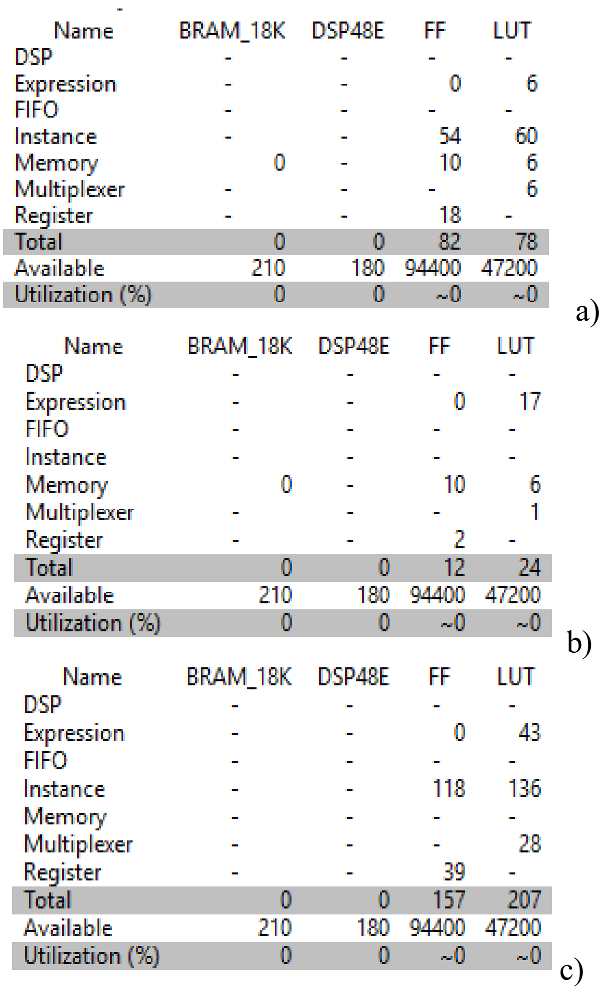
Fig. 5. B/RNS conversion for 10–bit word using memory look–up and modulo reduction using if–else



```
uint5 mod29_all_sum(uint5 w1, uint5 w2)
{
    uint4 b1;
    uint8 b2;

    uint4 b1n;
    uint8 b2n;

    uint5 x1p,x2p;//positive parts of number
    uint5 x1n,x2n;//negative parts of number
    uint5 sum_out;
    b1=w1;
    b2=w2<<4;
    b1n=~b1;
    b2n=(~w2)<<4;
        x1n = 29 - (b1n+1)%29;
        x2n = 29 - b2n%29;
        x1p = b1%29;
        x2p = b2%29;
        sum_out = (x1p + x2p)%29;
        if(sum_out>=29)
        {
            sum_out = (x1n + x2n)%29;
        }
    return sum_out;
}
```

Fig. 6. TCS/RNS memoryless conversion for 10–bit word using modulo reduction with if–else

In order to obtain an effective system structure the high–level description should reflect the properties of the hardware in the given FPGA design environment.



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	6
FIFO	-	-	-	-
Instance	-	-	54	60
Memory	0	-	10	6
Multiplexer	-	-	-	6
Register	-	-	18	-
Total	0	0	82	78
Available	210	180	94400	47200
Utilization (%)	0	0	~0	~0

a)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	17
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	10	6
Multiplexer	-	-	-	1
Register	-	-	2	-
Total	0	0	12	24
Available	210	180	94400	47200
Utilization (%)	0	0	~0	~0

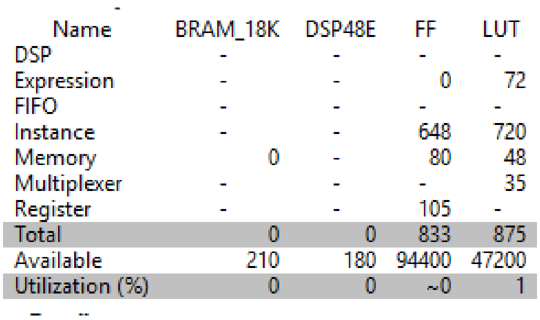
b)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	43
FIFO	-	-	-	-
Instance	-	-	118	136
Memory	-	-	-	-
Multiplexer	-	-	-	28
Register	-	-	39	-
Total	0	0	157	207
Available	210	180	94400	47200
Utilization (%)	0	0	~0	~0

c)

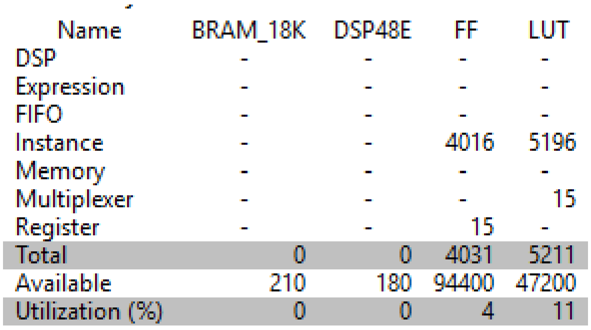
Fig. 7. The results of various methods of modulo reduction a) using ROMs, adders and divider b) ROM and subtraction (if–else) c) using adders and dividers chosen of the synthesis tool

For the description of the TCS/RNS converter for the synthesis in Xilinx Vivado HLS two approaches have been selected. The first, being the most limited, is based on adders and ROMs and most lavish makes use of the high–level description of the algorithm.. The synthesis results for Xilinx Atrix xc7a75tlftg256 are given in Fig 8 and Fig 9. It turns out that the optimized description of the TCS/RNS algorithm requires about 6 times less hardware than in the case of the direct high–level synthesis. The testbench result of the TCS/RNS converter is given in Fig. 10.



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	72
FIFO	-	-	-	-
Instance	-	-	648	720
Memory	0	-	80	48
Multiplexer	-	-	-	35
Register	-	-	105	-
Total	0	0	833	875
Available	210	180	94400	47200
Utilization (%)	0	0	~0	1

Fig. 8. Synthesis results for 16–bit TCS/RNS converter based on ROMs
(target device xc7a75tlftg256–2l)



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	-	4016	5196
Memory	-	-	-	-
Multiplexer	-	-	-	15
Register	-	-	15	-
Total	0	0	4031	5211
Available	210	180	94400	47200
Utilization (%)	0	0	4	11

Fig. 9. Synthesis results for 16–bit TCS/RNS converter based on adders
(target device xc7a75tlftg256–2l)

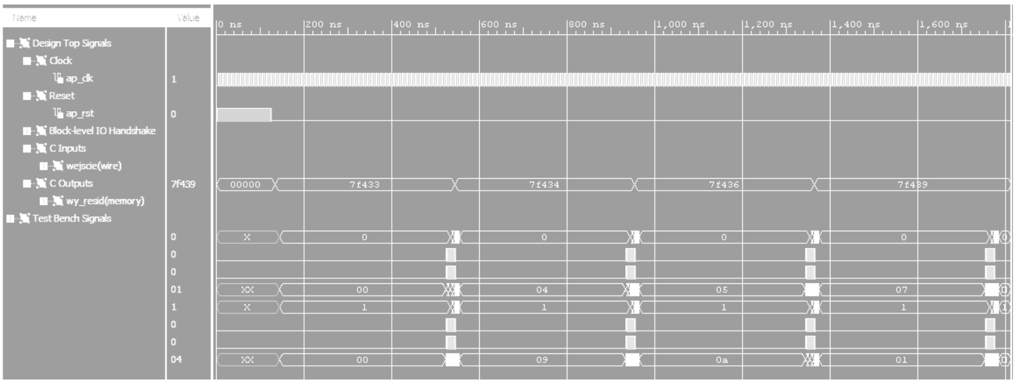


Fig. 10. TCS/RNS converter testbench result

6. CONCLUSIONS

The paper presents the results of design experiments using high level synthesis approach for TCS/RNS converter design. The experiments has been performed using Xilinx Vivado HLS. The aim of the experiments was to judge the influence of the form of the description of the system architecture on the hardware complexity of the TCS/RNS converter. In residue systems modulo reduction is the crucial operation with respect to hardware complexity. It was stated that the direct realization of modulo reduction using standard C++ operator leads to more complex architectures than the use of modulo reduction based on subtractions and comparisons which was in accordance with the expectations. The direct realization of fragment of converter using C++ modulo operator gave in result about six times greater hardware requirement than in the case of reduction based on additive operations and comparison. A conclusion can be drawn that however the HLS approach can considerably shortens the development process but it should be considered in common careful use with the standard operators.

REFERENCES

[1] Meeus W, Van Beeck K., Goedemé T., Meel J., Stroobandt D., An overview of today's high–level synthesis tools, DOI 10.1007/s10617–012–9096–8, Springer, 2012.

[2] Szabo N.S. and Tanaka R.J., Residue Arithmetic and its Applications to Computer Technology, New York, McGraw–Hill, 1967.

[3] Soderstrand M. et al., Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, IEEE Press, NY, 1986.

[4] Alia G., Martinelli E., "VLSI binary–residue converters for pipelined processing," Computer J., vol. 33, no.5, pp. 473–475, 1990.

[5] Piestrak S.J., Design of residue generators and multioperand modulo adders using carry–save adders, IEEE Trans. Comp., Volume 43, Pages 68–77, Jan. 1994.

[6] Premkumar A.B., A formal framework for conversion from binary to residue numbers, IEEE Trans. Circuits and Systems–II, Volume 49, Number 2, Pages 135–144, Feb.2002.

[7] Czyżak M., High–speed binary–to–residue converter with improved architecture, 27th Int. Conf. on Fundamentals of Electrotechnics and Circuit Theory, Gliwice–Niedzica, May 26–29, Pages 431–436, 2004.

[8] Premkumar A.B., Improved memoryless RNS forward converter based on periodicity of residues, IEEE Trans. Circuits and Systems–II, Express Briefs, Volume 53, Number 2, Pages 133–137, Feb. 2006.

(Received: 10. 02. 2017, revised: 28. 02. 2017)