Using both Binary and Residue Representations for Achieving Fast Converters in RNS

Seyed Mahdi Jameii[1] *, Shiva Taghipour[2] and Mohammad Azad[3]
1) Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran
2, 3) Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.
jamei@shahryariau.ac.ir; sh.taghipour@srbiau.ac.ir; azadm@iau-saveh.ac.ir

Received: 2011/02/18; Accepted:2011/04/03 Pages: 91-104

Abstract

In this paper, a new method is introduced for improving the efficiency of the Residue Number System, which uses both binary and residue representations in order to represent a number. A residue number system uses the remainder of the division in several different modules. Conversion of a number to smaller ones and carrying out parallel calculations on these numbers greatly increase the speed of the arithmetic operations. But as a result of using the non-weighted system, the performance of some calculations such as comparison and reverse conversion is extraordinarily difficult as compared with binary representation. We can use the benefits of both representations in the novel system and this scheme has a simple implementation for reverse/forward conversion.

Keywords: *Binary Presentation, Residue Number System, Forward Convertor, Backward Convertor*

1. Introduction

RNS is a collection of positive integers named a module and a set of modules which are called a module set [1][2]. In this system any integer number X can be represented as the vector of its residue module, such as $X = \{x_1, x_2, ..., x_n\}$ while x_i $x_i$denotes operation X modm_i[2] m_i.

The system will have the largest possible dynamic range if all modules are relatively pair wise primes [3]. The dynamic range of a module set is called M and is expressed as Eq. (1).

$$M = \prod_{i=1}^{n} m_i \quad (1)$$

Each binary operation between two integers (X, Y) in range M is defined as Eq. (2).

$$X \bullet Y = \{x_1 \bullet y_1, x_2 \bullet y_2, ..., x_n \bullet y_n\} \quad (2)$$

It is evident that such operations are performed in "residue-parallel" in modulo m_i.

In non-weighted number systems such as RNS, all operations are performed independently on each residue [1]. Thus using RNS is one of the effective ways to achieve parallelism on arithmetic level in VLSI systems [4]. Therefore, the propagation delay between modules is completely deleted. This means that no carry propagation among modules is needed, and the system can be used to reduce the complexity of calculations in many applications. Some applications of RNS are digital communications [5], digital signal processing [6], fault tolerance [7],[8]. Also the performance of RNS depends on the choice of the module set and on the converters which convert the binary representation to residue and vice versa. To improve the system efficiency, in this paper the module set $\{2^n, 2^n-1, 2^n+1\}$ [9] is used which is popular and has simple circuits. Also we use binary representation for the module set $\{2^n\}$ and residue representation for the modules $\{2^n-1, 2^n+1\}$ to enjoy the benefits of these two representation.

The paper is organized as follows:

In section 2 the usual Residue Number System is illustrated. Section 3 presents the proposed Residue Number System. In section 4 some comparisons are done based on the area and delay of forward/backward/addition circuit of traditional RNS and the proposed novel method. Finally section 5 concludes the paper.

2. Usual Residue Number System

In this section, there is an overview on the forward/ backward/ arithmetic operations in usual RNS.

2.1 Arithmetic Operations

In this part, instead of studying each arithmetic operations, we are going to introduce the summation in RNS since, multiplication and subtraction can be done using addition also addition is used by both forward and backward convertors. Therefore as addition is one of the fundamental operations it will be described presently.

In Residue Number System, two operands of the additional operations are defined as $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$ in module set $\{2^n, 2^n-1, 2^n+1\}$, thus the addition of two numbers (X, Y) in range M is defined as Eq. (3). [1]

$$X + Y = \{x_1 + y_1 \bmod m_1, x_2 + y_2 \bmod m_2, ..., x_n + y_n \bmod m_n\} \quad (3)$$

Consider x_i and y_i as the remainders of X and Y to m_i (which m_i is the ith module) , $0 < x_i < m_i, 0 < y_i < m_i$. Therefore With performing addition operation in this module the result lies between 0 and 2m-2. Therefore:

$$if\,(x_i + y_i) < m_i \Rightarrow out = x_i + y_i$$

$$if\,(x_i + y_i) \geq m_i \Rightarrow out = x_i + y_i - m_i$$

Now, we look up the addition operation separately in module set $\{2^n-1, 2^n, 2^n+1\}$ with the reminder set $\{x_1, x_2, x_3\}$.

2.1.1 Addition in module $2^n - 1$

If the result of this module is greater than or equal to $2^n - 1$, it should be added by the complement of this module which is $2^n - (2^n - 1) = 1$. On the other hand, if the result is less than the selected module, no additional calculation is needed. Therefore the result can be defined as follows:

$if\,(x_1 + y_1) < 2^n - 1 \Rightarrow out = x_1 + y_1$

$if\,(x_1 + y_1) \geq 2^n - 1 \Rightarrow out = x_1 + y_1 - 1$

One of the fastest adders in this module is described in Figure 1.

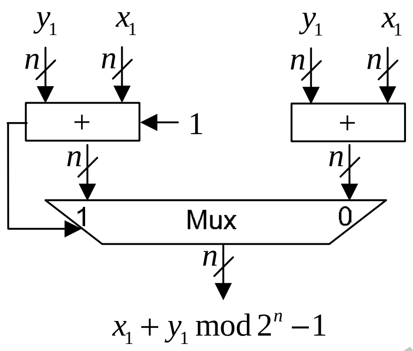

Figure 1. Adder circuit in module $2^n - 1$.

2.1.2 Addition in module 2^n

As mentioned, in all modules if the result of adding two numbers is greater than or equal to module, it should be added by the complement of that module which is here $2^n - (2^n) = 0$. Therefore the result is defined as follows:

$if\,(x_2 + y_2) < 2^n \Rightarrow out = x_2 + y_2$

$if\,(x_2 + y_2) \geq 2^n \Rightarrow out = x_2 + y_2$

Note that to calculate the reminder of any binary number like X, mod 2^n, the result can be obtained by simply considering the least significant bits of X. Therefore in addition operation, we can easily add up two binary numbers, and simply ignore the output carry. The corresponding circuit is shown in Figure 2.

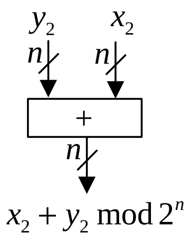

Figure 2. Adder circuit in module 2^n.

2.1.3 Addition in module $2^n + 1$

In this module like the previous module, If the result is greater than or equal to module, it should be added by the complement of module which is $2^n - (2^n + 1) = -1$, as known the second complement of module $2^n + 1$ is 011..1, thus it has a 0 at its highest position and 1 at all the other positions [10], On the other hand, if the result is less than the selected module, no additional calculation is needed. Thus the result can be defined as follows:

$if\,(x_3 + y_3) < 2^n + 1 => out = x_3 + y_3$

$if\,(x_3 + y_3) \geq 2^n + 1 => out = x_3 + y_3 + ((\overline{2^n + 1}) + 1)$

The circuit of adder for this module is shown in Figure 3.

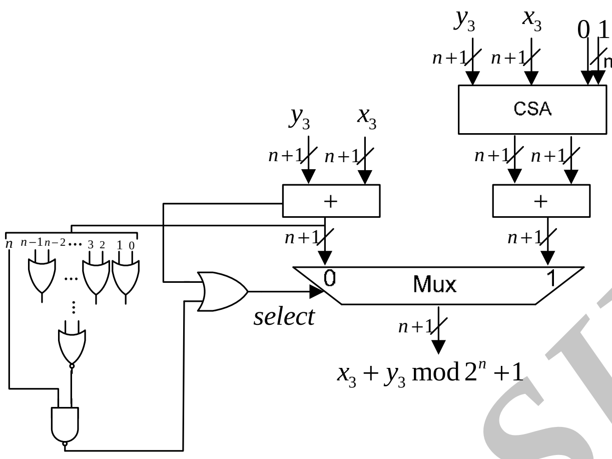

Figure 3. adder circuit in module $2^n + 1$.

2.2 Forward Converter

For using the Residue Number System, in the first step, the numbers should be converted from usual representation such as binary to residue representation which is named as forward converter. Suppose that number A has 3n bits in its binary representation then the residue is computed as the modulus reminder $\{2^n - 1, 2^n, 2^n + 1\}$.

$A = (a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n a_{n-1}...a_0)$

$= (a_{3n-1}a_{3n-2}...a_{2n}) \times 2^{2n} + (a_{2n-1}...a_n) \times 2^n + (a_{n-1}...a_0)$ Now consider:

$k_2 = (a_{3n-1}a_{3n-2}...a_{2n})$

$k_1 = (a_{2n-1}...a_n)$

$k_0 = (a_{n-1}...a_0)$

Therefore:

$A = k_2 \times 2^{2n} + k_1 \times 2^n + k_0$

2.2.1 Remainder calculations of A in module $2^n - 1$

$$\begin{aligned}
&(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n a_{n-1}...a_0) \bmod 2^n - 1 \\
&= ((a_{3n-1}a_{3n-2}...a_{2n}) \times 2^{2n}) \bmod 2^n - 1 + \\
&\quad ((a_{2n-1}...a_n) \times 2^n \bmod 2^n - 1) + \\
&\quad ((a_{n-1}...a_0) \bmod 2^n - 1) \\
&= (a_{3n-1}a_{3n-2}...a_{2n}) \bmod 2^n - 1 + \\
&\quad (a_{2n-1}...a_n) \bmod 2^n - 1 + \\
&\quad (a_{n-1}...a_0) \bmod 2^n - 1
\end{aligned}$$

$$= k_2 \bmod 2^n - 1 + k_1 \bmod 2^n - 1 + k_0 \bmod 2^n - 1$$
$$= (k_2 + k_1 + k_0) \bmod 2^n - 1$$
$$\Rightarrow \quad |A|_{2^n-1} = |\sum_{i=0}^{2} k_i|_{2^n-1}$$

Therefore, the remainder of a number in module $2^n - 1$ can be calculated by adding up each n-bit partition with its succeeding n-bit partition. The corresponding circuit is shown in Figure 4.

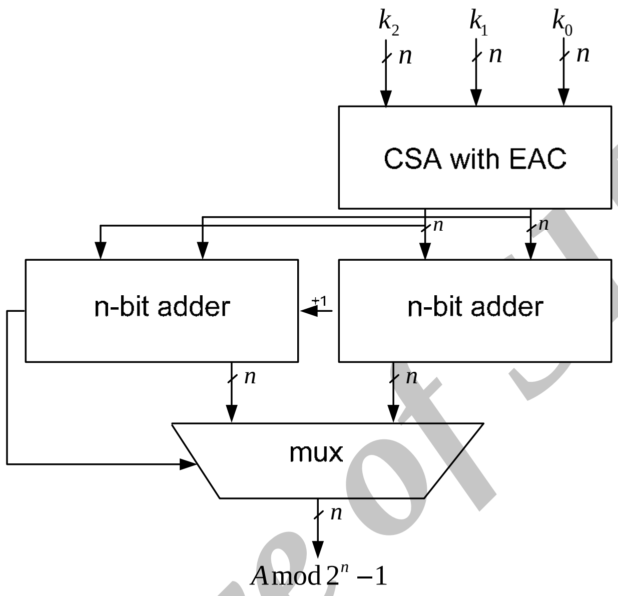

Figure 4. Circuit of binary to residue converter in module $2^n - 1$.

2.2.2 Remainder calculations of A in module 2^n

$$(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n a_{n-1}...a_0) \bmod 2^n$$
$$= ((a_{3n-1}a_{3n-2}...a_{2n}) \times 2^{2n}) \bmod 2^n +$$
$$((a_{2n-1}...a_n) \times 2^n \bmod 2^n) +$$
$$((a_{n-1}...a_0) \bmod 2^n)$$
$$= (a_{n-1}...a_0) \bmod 2^n = k_0 \bmod 2^n = k_0$$
$$\Rightarrow \quad |A|_{2^n} = |k_0|$$

The first and second partitions are multiples of 2^n thus the residue of these bits will be ignored. Therefore it is sufficient to consider only n right low order bits to calculate the reminder of 3n bit A to module 2^n . The corresponding circuit is shown in Figure 5.

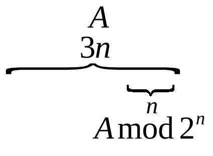

Figure 5. Circuit of binary to residue converter in module 2^n .

2.2.3 Remainder calculations of A in module 2^n+1

$$\begin{aligned}
&(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_na_{n-1}...a_0) \bmod 2^n+1\\
&=((a_{3n-1}a_{3n-2}...a_{2n})\times 2^{2n}) \bmod 2^n+1\\
&\quad+((a_{2n-1}...a_n)\times 2^n \bmod 2^n+1)\\
&\quad+((a_{n-1}...a_0) \bmod 2^n+1)\\
&=(a_{3n-1}a_{3n-2}...a_{2n}) \bmod 2^n+1\\
&\quad-(a_{2n-1}...a_n) \bmod 2^n+1\\
&\quad+(a_{n-1}...a_0) \bmod 2^n+1\\
&=(k_2 \bmod 2^n+1)-(k_1 \bmod 2^n+1)+(k_0 \bmod 2^n-1)=(k_2-k_1+k_0) \bmod 2^n+1
\end{aligned}$$

$$\Rightarrow \quad |A|_{2^n+1}=|\sum_{i=0}^{2}(-1)^i k_i\ |_{2^n+1}$$

Consequently, to calculate the remainder of a number in the module 2^n+1, we should add up n most positive bits with the next n negative bits and so on. The corresponding circuit is shown in Figure 6.

2.3 Backward Converter

One of the usual methods to calculate number X by means of its residue is to use Chinese Reminder Theorem. Consider the reminders as $\{x_1,x_2,x_3\}$ in module set $\{2^n-1,2^n,2^n+1\}$. The following equations are used.

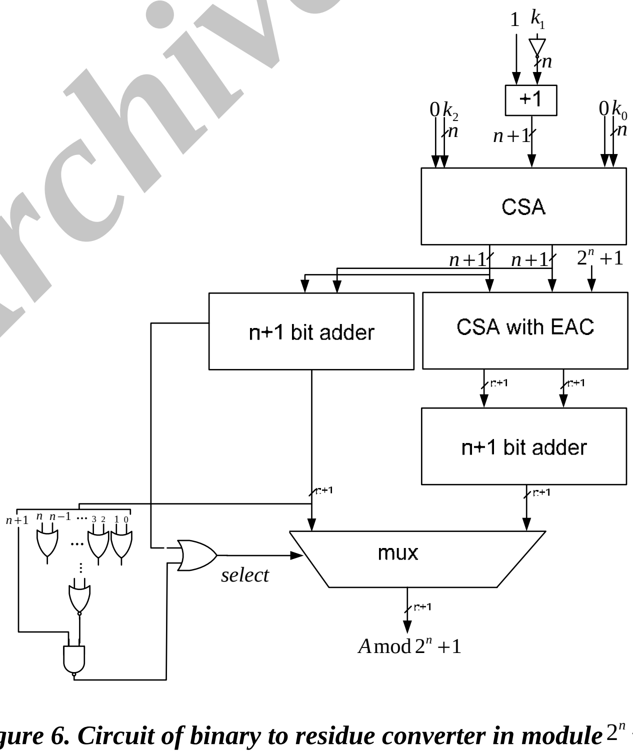

Figure 6. Circuit of binary to residue converter in module 2^n+1.

$$X = \left\langle \sum_{i=1}^{n} (X_i \times N_i)_{m_i} \cdot M_i \right\rangle_M \quad (4)$$

Where:

$$M_i = \frac{M}{m_i} \quad (5)$$

$$N_i = < M_i^{-1} >_{m_i}, i = 1, 2, ..., n \quad (6)$$

The required circuit for conversion of the RNS to binary is shown in Figure 7.

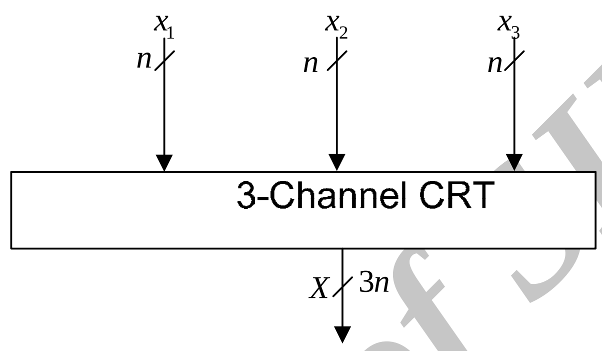

Figure 7. RNS to binary converter via Chinese Remainder Theorem

3. Proposed Residue Number System

In this paper we proposed a novel representation for the Residue Number System. This representation is a combination of residue display with binary presentation. In this method, we assume the n low order bits of original number in binary representation (which is weighted) and the other 2n most significant bits in the residue presentation. The forward and backward converters improve saliently, which is described as follows.

3.1 Arithmetic Operations

3.1.1 Addition in module $2^n - 1$

As mentioned before, this module uses residue representation. But because the first n bits are in binary representation, the output carry affects the result of this module (which will describe later in 3.1.2), also because the carry digit is 0 or 1, the result of circuit should be added with 0, 1 or 2. The proposed circuit is shown in Figure 8 (cin is the output carry of the adder in module 2^n).

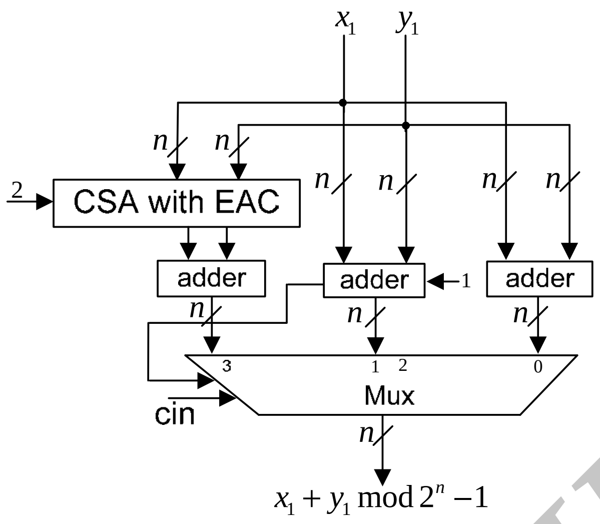

Figure 8. adder circuit in module $2^n - 1$

3.1.2 Addition in module 2^n

There is no difference between binary and module representation of a number in this module set. Therefore the arithmetic operations do not change in this module. However, as we assume this part as a binary representation, the output carry is important and should be propagated to the residue part.

The adder circuit of the proposed method is described in Figure 9.

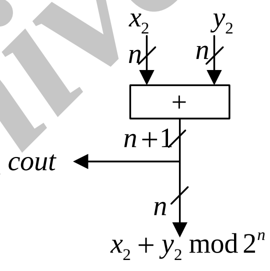

Figure 9. Adder circuit in module 2^n

3.1.3 Addition in module $2^n + 1$

The adder circuit for the module $2^n + 1$, is also similar to the adder in module 2^n-1 and is described in Figure 10.

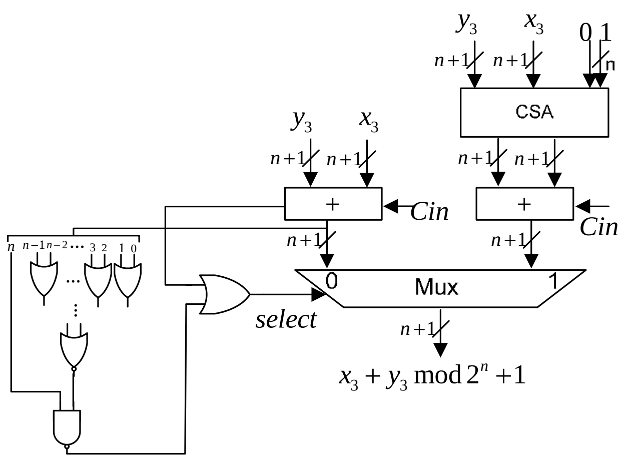

Figure 10. adder circuit in module 2^n+1

3.2 Forward Converter

Suppose that number A has 3n bits in its binary presentation but because it is assumed that the n low order bits are in binary representation, the reminders only computed in the modulus $\{2^n-1, 2^n+1\}$.

Therefore in the first step the n less significant bits are moved, because they should be left in binary representation. Now A has only 2n most significant digits.

$A=(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n)$

3.2.1 remainder calculation of A in module 2^n-1

$$(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n) \bmod 2^n-1$$
$$=((a_{3n-1}a_{3n-2}...a_{2n})\times 2^{2n}) \bmod 2^n-1+$$
$$((a_{2n-1}...a_n)\times 2^n \bmod 2^n-1) = (a_{3n-1}a_{3n-2}...a_{2n}) \bmod 2^n-1+$$
$$(a_{2n-1}...a_n) \bmod 2^n-1$$
$$=k_2 \bmod 2^n-1+k_1 \bmod 2^n-1$$
$$=(k_2+k_1) \bmod 2^n-1$$
$$\Rightarrow \quad |A|_{2^n-1}=|\sum_{i=1}^{2} k_i|_{2^n-1}$$

Therefore, to calculate the remainder of a 2n bit number in the module 2^n-1 in novel method, each n-bit partition is added up with its succeeding n-bit. See Figure 11.

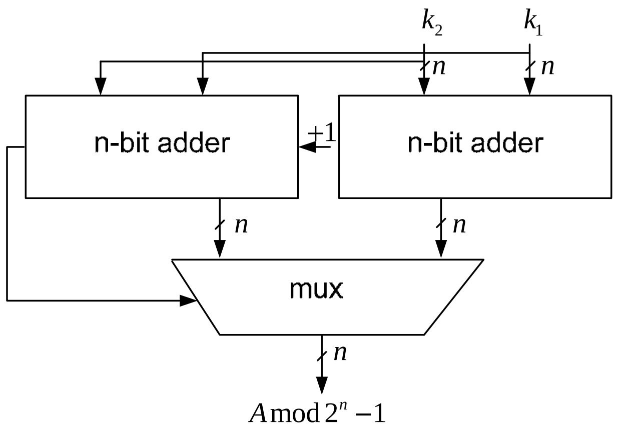

Figure 11. Adder circuit in module 2^n-1

3.2.2 remainder calculation of A in module 2^n+1

$$(a_{3n-1}a_{3n-2}...a_{2n}a_{2n-1}...a_n) \bmod 2^n+1$$
$$=((a_{3n-1}a_{3n-2}...a_{2n})\times 2^{2n}) \bmod 2^n+1+$$
$$((a_{2n-1}...a_n)\times 2^n \bmod 2^n+1) \;\; = (a_{3n-1}a_{3n-2}...a_{2n}) \bmod 2^n+1$$
$$-(a_{2n-1}...a_n) \bmod 2^n+1 \;\; = (k_2 \bmod 2^n+1)-(k_1 \bmod 2^n+1) = (k_2-k_1) \bmod 2^n+1$$

ð $$|A|_{2^n+1}=\left|\sum_{i=1}^{2}(-1)^i k_i\right|_{2^n+1}$$

Consequently, to calculate the remainder of a number in the module 2^n+1, n most positive bits are added up with the next n negative bits. See Figure 12.

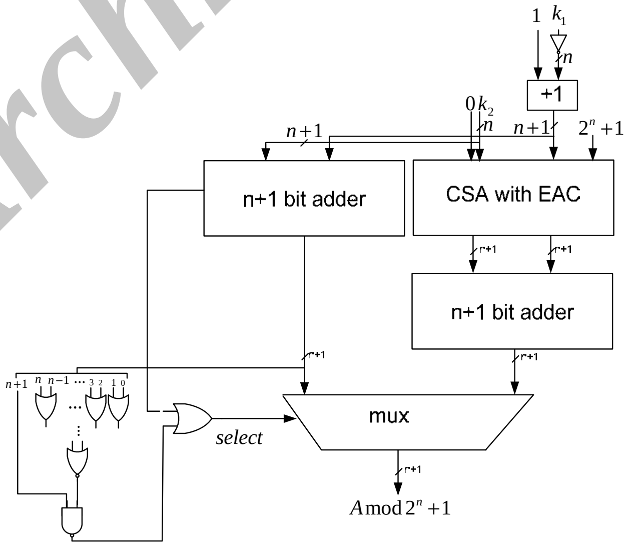

Figure 12. Circuit of binary to residue converter in module 2^n+1.

3.3 Backward Converter

Consider the remainders as $\{r_1, r_2, r_3\}$ in module set $\{2^n - 1, 2^n, 2^n + 1\}$, as discussed before these remainders are both in binary and residue representation systems. This means that while r_1 and r_3 are in residue representation, r_2 is in the binary representation. Also the final result should have 3n bits which the first n low orders bits should be always equal to r_2. Because only the module set $\{2^n - 1, 2^n + 1\}$ are in residue mode, a backward converter is used to convert these module set. It converts a module representation to binary representation and places the result as the 2n bits highest order bits. Also the remainder of module $\{2^n\}$ which is r_2, is placed in n low order.

Therefore, in order to calculate number X from its residue in the novel method with the Chinese Remainder Theorem, it should be used only for the module set $\{2^n - 1, 2^n + 1\}$.

The proposed circuit for converting residue to binary representation is shown in Figure 13.

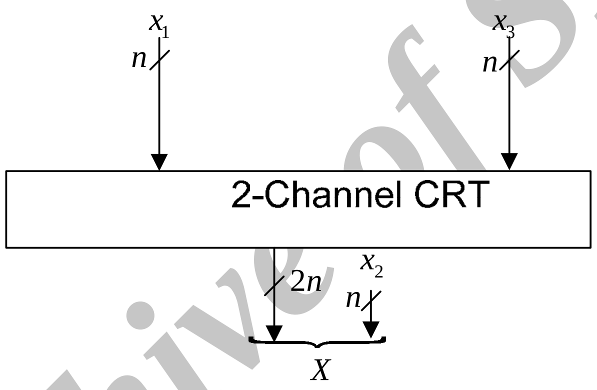

Figure 13. RNS to binary converter in novel RNS

4. Comparison

In this paper a novel representation for the Residue Number System is presented which uses both residue and binary representations. In this case, the system makes use of both binary and residue representation advantages.

A comparison is done for forward/backward and arithmetic operations between the proposed method and previous methods in Table 1. Table 2 presents the total delay and hardware complexity of both presentations. As illustrated in Table 2, the comparing result demonstrates some improvements in terms of area and delay for both converters.

In addition to remembered comparisons, the proposed reverse converter reduces 3-channel CRT to 2-channel CRT. Therefore, it decreases converter hardware delay and cost as compared with usual Residue Number System.

5. Conclusions

In this paper, a novel hybrid method is presented. This method is composed of RNS and binary representation systems and combines them. The comparing result demonstrates that new method improves speed and decreases hardware costs in system converters as compared with usual RNS.

6. References

[1] M. Hosseinzadeh, K. Navi, S. Gorgin, "A New Moduli Set for Residue Number System: {r^{n-2},r^{n-1},r^{n}}", International Conference on Electrical Engineering 11-12 April 2007.

[2] A.Hariri, K. Navi, R. Rastegar, "A new high dynamic range moduli set with efficient reverse converter". Elsevier Journal of Computers and Mathematics with Applications, 2008, 55(4): 660-668.

[3] A. Molahosseini, K. Navi, O. Hshemipour, A. Jalali, "An efficient architecture for designing reverse converters based on a general three moduli set", Elsevier Journal of Systems Architecture, in Press, 2008.

[4] M.A.bayoumi and P. Srinivasan, "Parallel arithmetic: from algebra to arvhitectuer", Proc. IEEE international Symposium on Circuits and Syetems, pp. 2630-2633, 1990.

[5] A. D. Re, A. Nannareli and M. Re, "A Tools for Arithmetic Generation of RTL-Level VHDL Description of RNS FIR Filters," IEEE Proceeding of the Design, Automation and Test in Europe Conference and Exhibition, pp. 686-687, 2004.

[6] S. Yen, S. Kim, S. Lim and S. Moon, "RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis," IEEE Transactions On Computers, Vol. XX, No. Y, pp. 461-472, 2003.

[7] A. Chren, Jr., "One-Hot Residue Coding for Low Delay-Power Product CMOS Design," IEEE Transactions On Circuits And Systems II: Analog And Digital Signal Processing, Vol. 45, No. 3, Mar. 1998.

[8] A. F. Gonzalez, and P. Mazumdar, Redundant Arithmetic, "Algorithms and Implementations," Integration: The VLSI Journal, Vol. 30, No. 1, pp. 13-53, 2000.

[9] Y. Wang, X. Song, M. Aboulhamid, and H. Shen, "Adder based residue to binary numbers converters for (2n-1, 2n, 2n+1)," IEEE Trans, Signal Processing, vol. 50, no.7, pp.1772-1779, 2002.

[10] A.A. Hiasat, "High-Speed and Reduced Area Modular Adder Structures for RNS," IEEE Trans. Computers, pp. 84-89, 2002.

		Module	Area					Delay
			F.A	H.A	NOT	MUX	AND/OR	
Forward Converter	Proposed	2^n	-	-	-	-	-	-
		2^n-1	2n-2	2	-	1	-	$1t_{mux}+1t_{HA}+(n-1)t_{FA}$
		2^n+1	4n	4	n	1	n+1	$1t_{mux}+(n+2)t_{HA}+nt_{FA}+(2+\log n)t_{and}+1t_{not}$
	[1]	2^n	-	-	-	-	-	-
		2^n-1	3n-2	2	-	1	-	$1t_{mux}+1t_{HA}+nt_{FA}$
		2^n+1	4n-1	n+4	n	1	n+2	$1t_{mux}+1t_{not}+(n+3)t_{HA}+nt_{FA}+(2+\log n)t_{and}$
Addition	Proposed	2^n	n-1	1	-	-	-	$1t_{HA}+(n-1)t_{FA}$
		2^n-1	3n-3	n+3	-	2	-	$2t_{mux}+2t_{HA}+(n-1)t_{FA}$
		2^n+1	2n+2	n+1	-	1	n+1	$1t_{mux}+1t_{HA}+(n+1)t_{FA}+(2+\log n)t_{and}$
	[1]	2^n	n-1	1	-	-	-	$1t_{HA}+(n-1)t_{FA}$
		2^n-1	2n-2	2	-	1	-	$1t_{mux}+1t_{HA}+(n-1)t_{FA}$
		2^n+1	2n	n+3	-	1	n+1	$1t_{mux}+2t_{HA}+nt_{FA}+(2+\log n)t_{and}$

		Module	Unit Gate Area	Unit Gate Delay	Time Complexity
Forward Converter	Proposed	2^n	-	-	-
		2^n-1	$14n-4$	$4n-1$	$56n^2-30n+4$
		2^n+1	$30n+19$	$6n+\log n+8$	$180n^2+30n\log n+354n+19\log n+152$
	[1]	2^n	-	-	-
		2^n-1	$21n-4$	$4n+3$	$84n^2+47n-12$
		2^n+1	$34n-3$	$6n+\log n+10$	$204n^2+3n\log n+328n+19\log n-30$
Addition	Proposed	2^n	$7n-3$	$4n-2$	$28n^2-26n+6$
		2^n-1	$25n-5$	$4n+2$	$100n^2+30n-10$
		2^n+1	$19n+20$	$4n+\log n+9$	$76n^2+19n\log n+251n+20\log n+180$
	[1]	2^n	$7n-3$	$4n-2$	$28n^2-26n+6$
		2^n-1	$14n-4$	$4n-1$	$56n^2-30n+4$
		2^n+1	$19n+14$	$4n+\log n+7$	$76n^2+19n\log n+189n+14\log n+98$