The CSI Journal on
Computer Science and Engineering
Vol. 9, No. 2 & 4 (b), 2011
Pages 52-62
Regular Paper

Efficient Modular Binary Signed-Digit Multiplier for the moduli set $\{2^n-1, 2^n, 2^n+1\}$

Maryam Saremi[1] Somayeh Timarchi[1,2]

[1]Department of Electrical and Computer Engineering, Shahid Beheshti University, G. C., Tehran, Iran
[2]Cyberspace Institute, Shahid Beheshti University, G. C., Tehran, Iran

Abstract

Arithmetic operations on Binary Signed-Digit (BSD) number system are performed in a constant time due to the carry free addition capability in redundant number system. Applying BSD number representation to Residue Number System (RNS) leads to a number system called BSD-RNS. There are three BSD-RNS encodings: 2's complement, 1-out-of-3, and posibit-negabit. Up to now, BSD-RNS multipliers with 2's complement and 1-out-of-3 encoding have been reported in the related literatures. In this work, we propose posibit-negabit BSD-RNS multiplier for the moduli set $\{2^n-1, 2^n, 2^n+1\}$. Afterwards, we compare it to the existing BSD-RNS multipliers. Synthesis results show that the proposed architecture is more efficient than the two existing BSD-RNS multipliers from the viewpoint of delay, area, and power consumption.

Keywords: Digital Computer Arithmetic, Residue Number System (RNS), Binary Signed-Digit (BSD) Number Representation, Carry Free Addition, Modular Multiplier.

1. Introduction

Addition and multiplication are basic arithmetic operations and improving their performance lead to high-performance applications such as digital signal processing [1], FIR filtering [2], communication [3], and cryptography [4]. In conventional number system like binary number system, the speed of addition and multiplication operations is limited by the size of input operands. To improve the speed of these operations, non-conventional number systems like Residue Number System (RNS) and redundant number system have been suggested in the literatures. RNS is an integer number system which is defined by a set of primary positive integers as the moduli. Utilizing the moduli set, a large integer number is represented by smaller numbers called residues and arithmetic operations are performed in parallel on the residues instead of large numbers. Consequently, the speed of performing the operations in RNS is more than conventional number system. Generally, the performance of residue arithmetic operations is reliant on the selected moduli set and the representation of residues. The moduli set $\{2^n-1, 2^n, 2^n+1\}$ are greatly utilized in the literatures due to simple structure of modular adders with this moduli set. The modular adders can be constructed of non-modular (modulo 2^n) adder as well as End-Around-Carry (EAC) unit. One of the important characteristic of RNS is limited carry propagation in each modulo [5]. The remainder of carry propagation in each module is removed by utilizing a redundant number system to represent the residues. In redundant number system such as Binary Signed-Digit (BSD) number system [6], addition operation is performed in a constant time without carry propagation, i.e., the carry generated in each position is absorbed by the next position as an input carry. So, the carry propagation chain in BSD number system is removed. By utilizing BSD representation for RNS residues, the achieved number system is called BSD-RNS in the literatures. So far, 2's complement [7], 1-out-of-3 [8], and posibit-negabit (also called Pos-Neg) [9] encodings have been used in RNS. Then 2's complement BSD-RNS [7], 1-out-of-3 BSD-RNS [10], and Pos-Neg

BSD-RNS [11], [12] are achieved. Modular adders based on these encodings are reported in [7], [10], and [12], respectively. Modular multipliers based on 2's complement and 1-out-of-3 BSD encodings have been proposed in [7] and [13], respectively. However, Pos-Neg BSD-RNS multiplier has not been suggested in the open literatures. In this work, we utilize Pos-Neg BSD encoding to implement Pos-Neg BSD-RNSmultiplier and compare it with existing modular multipliers. The comparison results show that the proposed BSD-RNS multiplier is more efficient than the other BSD-RNS multipliers from the viewpoint of delay, area, and power.

The rest of this paper is prepared as follows: Section 2 provides backgrounds on RNS, 2's complement BSD-RNS, 1-out-of-3 BSD-RNS, and Pos-Neg BSD-RNS. The proposed BSD-RNS multiplier is described in section 3. Synthesis results and comparisons are studied and analysed in section 4. Finally, this paper is concluded in the last section.

2. Background and Discussion

2.1. Background on RNS

The RNS is defined in terms of primary positive integers P= $\{p_1, p_2, \ldots, p_r\}$ as the moduli set where GCD $(p_i, p_j)=1$ for $i \neq j$. GCD (x, y) is defined as the Greatest Common Divisor of x and y. In RNS, any unsigned integer A in the range [0, M-1], where M is the dynamic range and is defined as$M = p_1 \times p_2 \times \ldots p_n$is given by a unique r-tuple $(A_1, A_2, \ldots, A_r)$. The residue A_i is obtained as follows [14]:

$$A_i = A \bmod p_i = |A|_{p_i} \tag{1}$$

Where $i \in \{0, 1, \ldots, r\}$ and A_i is in the range $[0, p_i\text{-}1]$. Besides, for signed integer A in the range (-M/2, M/2], its residue is defined as follows [14]:

$$A_i = |A|_{p_i} = \text{sign}(A) \times |\text{abs}(A)|_{p_i} - \text{sign}(A) \times p_i \tag{2}$$

Where, A_i is in the range $(-p_i/2, p_i/2]$, sign (A) and |abs (A)| are defined as follows [14]:

$$\text{sign}(A) = \begin{cases} 1, & A \geq 0 \\ -1, & A \leq 0 \end{cases} \tag{3}$$

$$|\text{abs}(A)| = \begin{cases} A, & A \geq 0 \\ -A, & A \leq 0 \end{cases} \tag{4}$$

Let's assume A and B are two integers represented in RNS by $(A_1, A_2, \ldots, A_r)$ and $(B_1, B_2, \ldots, B_r)$, respectively. As mentioned above, arithmetic operations in RNS such as addition/subtraction and multiplication are performed in parallel as follows [14]:

$$A \circ B = |A_1 \circ B_1|_{p_1}, |A_2 \circ B_2|_{p_2}, \ldots, |A_r \circ B_r|_{p_r} \tag{5}$$

The '$\circ$' symbol indicates an operation such as addition, subtraction, and multiplication.

2.2. Background on BSD-RNS Encodings

An integer number X in BSD number system with n-digit is represented as follows:

$$X = x_0 r^0 + x_1 r^1 + \cdots + x_{n-1} r^{n-1} = \sum_{i=0}^{n-1} x_i r^i \tag{6}$$

Where, x_i is a binary signed-digit, 2^i is the weight of i-th digit x_i. Considering radix-2, x_i is in the range {-1, 0, 1}.

Utilizing BSD number representation in RNS, BSD-RNS is achieved. The residues in this system are represented by BSD number representation.

Let's assume A is a number in BSD-RNS with the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$. Then, A is represented by three residues $\{A_1, A_2, A_3\}$, where A_i ($i \in \{1, 2, 3\}$). According to (6), each residue A_i is represented with n-digit that each digit is encoded with BSD number representation as follows:

$$A_i = a_0 r^0 + a_1 r^1 + \cdots + a_{n-1} r^{n-1} = \sum_{i=0}^{n-1} a_i r^i \tag{7}$$

In the following, we review the three existing BSD-RNS encodings:

2.2.1. 2's Complement BSD-RNS Encoding

2's complement BSD encoding is commonly used to represent residues in RNS. In the number system, the residue A_i in (7) is represented with n digits where each digit (a_i) in the range {-1, 0, 1} is represented with two bits ($a_i^1 a_i^0$) in 2's complement format as follows:

$$A_i = (a_{n-1}^1 a_{n-1}^0 \ldots a_1^1 a_1^0 a_0^1 a_0^0) \tag{8}$$

Two's complement representation of $a_i \in \{-1, 0, 1\}$ is depicted in table 1 (a). Due to the fact, the encoding is called 2's complement BSD-RNS.

2.2.2. 1-Out-of-3 BSD-RNS Encoding

1-out-of-3 BSD encoding is a subset of m-out-of-n encoding as described in [8], [15]. This encoding is a code of length n that has m numbers of '1's in each code word [15]. This encoding is represented in table 1 (b). As shown in this table, {-1, 0, 1} are represented by {100, 010, 001}, respectively. Considering (7), in 1-out-of-3 BSD encoding, A_i is represented with n digits that each digit (a_i) is shown by three bits ($a_i^2 a_i^1 a_i^0$) as follows:

$$A_i = (a_{n-1}^2 a_{n-1}^1 a_{n-1}^0 \ldots a_1^2 a_1^1 a_1^0 \;\; a_0^2 a_0^1 a_0^0) \tag{9}$$

2.2.3. Pos-Neg BSD-RNS Encoding

The posibit-negabit encoding has been proposed in [9]. In this work, this number representation is called Pos-Neg BSD encoding. Each BSD digit a_i with this encoding is represented by two bits namely posibit (a_i^+) and negabit (a_i^-) with equal weight. A negabit is a bit in the range {-1, 0}, where -1 and 0 are encoded by 0 and 1, respectively. A posibit is a normal bit in the range {0, 1}. Pos-Neg BSD encoding is shown in table 1 (c). As shown in this table, -1 and 1 are represented with [00] and [11], respectively. However, there are two representations for zero: [01] and [10] shown in table 1 (c). This encoding has an interesting feature described in the following: If we change the polarities of (a_i^-) and (a_i^+), the values of digit will not change, i.e., $[a_i^+ a_i^-]$ or $[a_i^- a_i^+]$, have equal values. The residue A_i in

Pos-Neg BSD-RNS encoding is represented with n-digit that each digit (a_i) is shown with ($a_i^+ a_i^-$) as follows:

$$A_i = (a_{n-1}^+ a_{n-1}^- \dots a_1^+ a_1^- a_0^+ a_0^-) \quad (10)$$

Table 1. Binary representations of the three existing BSD encodings

BSD Digit	(*a*) 2's complement BSD		(*b*) 1-out-of-3 BSD			(*c*) Pos-Neg BSD	
a_i	a_i^1	a_i^0	a_i^2	a_i^1	a_i^0	a_i^+	a_i^-
-1	1	1	1	0	0	0	0
0	0	0	0	1	0	0	1
						1	0
1	0	1	0	0	1	1	1

2.3. Background on BSD-RNS Adders

2.3.1. 2's Complement BSD-RNS Adder

Generally, the addition of two n-digit BSD input Generally, the addition of two n-digit BSD input operands is performed in a constant time that equal to the addition of two redundant digits. Up to now, a variety of modular adders based on 2's complement BSD representation have been proposed in the literatures and is called 2's complement BSD-RNS adder. In [7], the most efficient architecture of 2's complement BSD-RNS adder has been proposed and its structure is shown in Figure 1(a) and in this work, is briefly called MSDA. As shown in this figure, MSDA is composed of n SDFA units composed of two blocks ADD1* and ADD2*.

The inputs of ADD1* are four redundant digits (x_{i-1}, y_{i-1}, x_i, and y_i) and its outputs are intermediate sum (u_i) and intermediate carry (v_i) represented by one bit in the range {0, 1}. These intermediate signals enter to the ADD2* block as inputs. The output of this block is final sum (s_i) and is obtained by subtracting the intermediate signals, i.e., $s_i=v_{i-1}-u_i$. As shown in this figure, the final carry (v_{n-1}) produced in the most significant position (msp) is re-entered into the least significant position (lsp) based on the type of modulus.

2.3.2. 1-Out-of-3 BSD-RNS Adder

The 1-out-of-3 BSD-RNS adder has been recently proposed in [10] and the block diagram of this adder is shown in Figure 1(b). As mentioned above, the addition of two n-digit BSD input operands is performed in a constant time. Therefore, the performance of 1-out-of-3 BSD-RNS adder is reliant on BSD_FA basic component. This basic component is composed of two blocks namely, Generate and Compute. Intermediate signals i.e. intermediate sum (U_i) and intermediate carry (C_i) are generated in the Generate block and encoded with 1-out-of-3 BSD number representation. The final sum is obtained by this formula ($S_i=U_i+C_{i-1}$) in the Compute block.

For each position like i-th position, there are five inputs (x_i, y_i, x_{i-1}, y_{i-1}, and c_{i-1}) and two outputs (C_i, S_i). For the first position, the inputs and outputs are (x_0, y_0, x_{-1}, y_{-1}, and c_{-1}) and (C_0, S_0), respectively. Three inputs (x_{-1}, y_{-1}, c_{-1}) are obtained by re-entering of (x_{n-1}), (y_{n-1}), and (c_{n-1}) based on the type of modulus. In modulo 2^n-1, these signals are rotated without any change; however, in modulo 2^n+1, they are replaced with opposite sign, i.e., ($x_{-1}=-x_{n-1}$), ($y_{-1}=-y_{n-1}$), and ($c_{-1}=-c_{n-1}$). Also, in modulo 2^n, the re-entered signals are replaced with zero value.

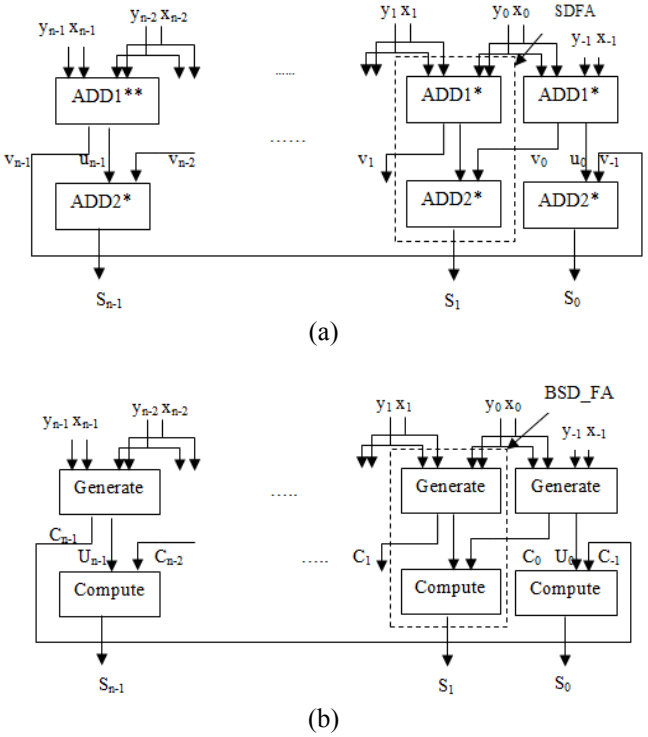
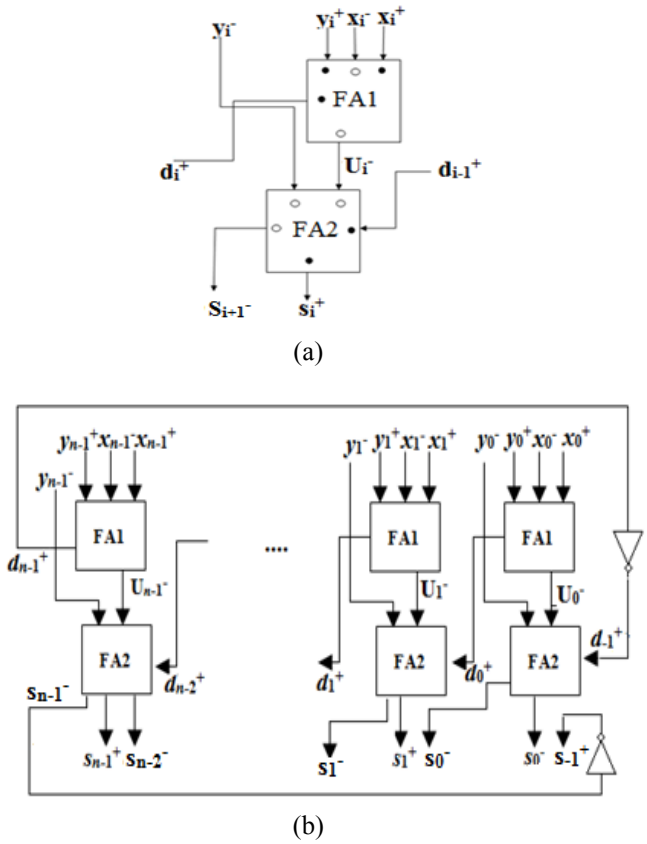


Figure 1. BSD-RNS adders for (a) 2's complement operands [7], and (b) 1-out-of-3 operands [10]

2.3.3. Pos-Neg BSD-RNS Adder

A Pos-Neg BSD-RNS adder is recently proposed in [12]. This modular adder is constructed of n basic components shown in Figure 2(a). The basic component of Pos-Neg BSD-RNS adder structure is a standard 4:2 compressor that is described in [11]. As shown in Figure 2(a), this component is composed of two full adders, namely FA1 and FA2. The inputs of FA1 are (x_i^+, x_i^-, and y_i^+) with the same weight which produce two outputs: U_i^- (intermediate sum) with the same weight and d_i^+ (output carry) with double-weight. The output carry (d_i^+) in position i, where i ϵ {0, 1, …, n-1}, is absorbed by the next position (i+1) as input carry.

The inputs of FA2 are (y_i^-, d_{i-1}^+, and U_i^-) with the same weight and produce two outputs, namely, S_i^+ (Pos-final sum) with the same weight and S_{i+1}^- (Neg-final sum) with double-weight. In Pos-Neg BSD-RNS adder, the final output carry (d_{n-1}^+) and Neg-final sum (S_n^-) in the msp are re-entered into the lsp based on the type of moduli as d_{-1}^+ and S_0^-, respectively. For modulo 2^n-1, these values are re-entered into the lsp without any changes. For modulo 2^n, an input carry in the lsp (d_{-1}^+) is '0' with zero value and Neg-final sum (S_0^-) is '1' with zero value. While, in modulo 2^n+1, d_{n-1}^+ and S_n^- are first inverted and re-entered into the lsp as shown in Figure 2(b). In this module, the final signal in the lsp are represented by [$S_0^- S_{-1}^+$]. As mentioned above, this reversal of posibit and negabit positions are not important and don't have any effect in the final result.



(a)

(b)

Figure 2. The Pos-Neg BSD-RNS adder: (a) The basic component of Pos-Neg BSD-RNS adder in position *i* [11], and (b) the Pos-Neg BSD-RNS adder for modulo 2^n+1 [12]

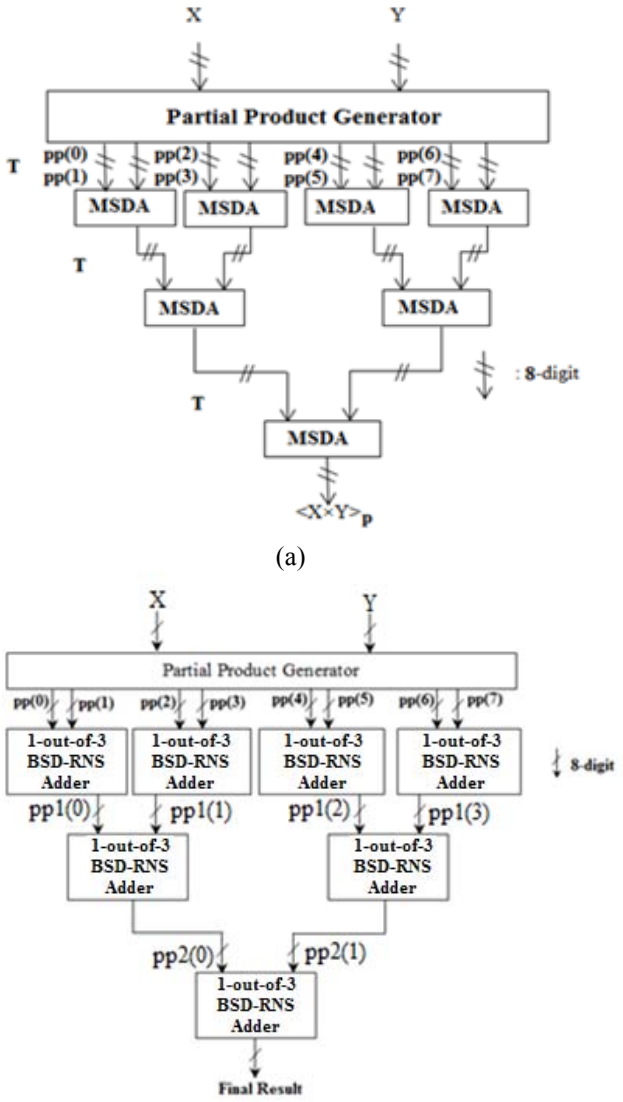
2.4. Background on BSD-RNS Multiplier

2.4.1. 2's Complement BSD-RNS Multiplier

Generally, any modular multiplier is composed of two fundamental parts namely, partial product generation and partial product reduction. Therefore, the efficiency of multiplication is improved by reducing the number of partial products or accelerating their accumulation. Besides, the performance of partial products accumulation depends on the numbering system used to represent the operands. So far, a variety of modular multi pliers tructures based on 2's complement BSD number representation have been proposedin the literatures. In these modular multipliers, generation of partial products has been performed in the same way. The only difference among these structures is the type of 2's complement BSD-RNS adder utilized for accumulation of partial products. In [7], one of the most efficient 2's complement BSD-RNS multiplier has been discussed and its structure is shown in Figure 3(a). As shown in this figure, the accumulations of partial products are performed according to 2's complement BSD-RNS adder that is described in section 2.3.1.

2.4.2. 1-Out-of-3 BSD-RNS Multiplier

The 1-out-of-3 BSD-RNS multiplier has been recently reported in [13]. As shown in Figure 3(b), its structure is the same way with 2's complement BSD-RNS multiplier. This structure is composed of two important parts. The first part: Generation of partial products with 1-out-of-3 BSD encoding and the second part: Reduction of these partial products by 1-out-of-3 BSD-RNS adder. The partial products generation is similar to the process of 2's complement BSD-RNS partial products generation, but they are represented by 1-out-of-3 BSD encoding instead of 2's complement BSD representation. Also, the partial product reduction is performed with 1-out-of-3 BSD-RNS adder that proposed in [10].



(a)

(b)

Figure 3. BSD-RNS multipliers for (a) 2's complement [7] and, (b) 1-out-of-3 [13] operands

2.5. Discussion

In this section, for clear discussion, we present examples for 2's complement BSD-RNS, 1-out-of-3 BSD-RNS, and Pos-Neg BSD-RNS encodings and additions.

Example 1:
Let's assume an integer number A and the moduli set P= $\{p_1, p_2, p_3\} = \{2^h-1, 2^h, 2^h+1\}$ with h=4, where n= $[\log_2{}^{p1}]$, m= $[\log_2{}^{p2}]$, v= $[\log_2{}^{p3}]$. Find the residue A_i with 2's complement BSD-RNS, 1-out-of-3 BSD-RNS, and Pos-Neg BSD-RNS encodings.

Solution 1:
The three residues A_1, A_2, and A_3 are represented in BSD-RNS by $(a_{n-1},\ldots, a_1, a_0)_{BSD\text{-}RNS}$, $(a_{m-1}^{/},\ldots, a_1^{/}, a_0^{/})_{BSD\text{-}RNS}$, and $(a_{v-1}^{//}, \ldots, a_1^{//}, a_0^{//})_{BSD\text{-}RNS}$, respectively, where a_i, $a_j^{\square}$, $a_k^{\square\square} \epsilon \{-1, 0, 1\}$ and $i\epsilon\{0, 1, \ldots, n-1\}$, $j\epsilon\{0, 1, m-1\}$, and $k\epsilon\{0, 1, \ldots, v-1\}$, as show in table 2.

Example 2:
Let's assume X and Y are BSD numbers, $X=(1\ \bar{1}\ 1\ 1)_{BSD}$, $Y=(1\ 1\ \bar{1}\ 0)_{BSD}$. Find $|X+Y|_P$ based on 2's complement BSD-RNS, 1-out-of-3 BSD-RNS, and Pos-Neg BSD-RNS addition for the moduli set $\{2^n-1, 2^n, 2^n+1\}$ where n=4.

Solution 2:
The results are shown in tables 3, 4, and 5.

Table 2. Solution to example 1

	General Representation	**Example**
Integer	A	137
RNS moduli	p_1,p_2, p_3	15,16,17
RNS residues	A_1,A_2,A_3	$A_1=2$, $A_2=9$, $A_3=1$
BSD-RNS	$A_1=(a_{n-1}, \ldots, a_1, a_0)_{BSD\text{-}RNS}$, $A_2=(a_{m-1}^{/}, \ldots, a_1^{/}, a_0^{/})_{BSD\text{-}RNS}$, $A_3=(a_{v-1}^{//}, \ldots, a_1^{//}, a_0^{//})_{BSD\text{-}RNS}$	$A_1=2 = (1\ \bar{1}\ 0)_{BSD\text{-}RNS}$, $A_2=9 = (1\ 0\ 1\ \bar{1})_{BSD\text{-}RNS}$, $A_3=1 = (0\ 0\ 1\ \bar{1})_{BSD\text{-}RNS}$
2's complement BSD-RNS	$(a_t^1a_t^0 \ldots a_1^1a_1^0 \quad a_0^1a_0^0)$ $t\epsilon\{n, m, v\}$	$A_1=2 = (01\ 11\ 00)$, $A_2=9 = (01\ 00\ 01\ 11)$, $A_3=1 = (00\ 00\ 01\ 11)$
1-out-of-3 BSD-RNS	$(a_t^2a_t^1a_t^0 \ldots a_1^2a_1^1a_1^0$ $a_0^2a_0^1a_0^0)$ $t\epsilon\{n, m, v\}$	$A_1=2 = (001\ 100\ 010)$, $A_2=9 = (001\ 010\ 001\ 100)$, $A_3=1 = (010\ 010\ 001\ 100)$
Pos-Neg BSD-RNS	$(a_t^+a_t^- \ldots a_1^+a_1^- \quad a_0^+a_0^-)$ $t\epsilon\{n, m, v\}$	$A_1=2 = (11\ 00\ 01)$, $A_2=9 = (11\ 01\ 11\ 00)$, $A_3=1 = (01\ 10\ 11\ 00)$

Table 3. Two's complement BSD-RNS addition of Example2

	$P=2^4-1$				**$P=2^4$**				**$P=2^4+1$**			
position i	3	2	1	0	3	2	1	0	3	2	1	0
BSD number X	1	-1	1	1	1	-1	1	1	1	-1	1	1
BSD number Y	1	1	-1	0	1	1	-1	0	1	1	-1	0
Intermediate sum u_i	0	0	0	1	0	0	0	1	0	0	0	0
Intermediate carry v_{i-1}	0	0	1	1	0	0	1	0	0	0	0	0
Final sum s_i	0	0	1	0	0	0	1	-1	0	0	0	0

Table 4. 1-out-of-3 BSD-RNS addition of Example 2

	$P=2^4-1$				**$P=2^4$**				**$P=2^4+1$**			
position i	3	2	1	0	3	2	1	0	3	2	1	0
BSD number X	1	-1	1	1	1	-1	1	1	1	-1	1	1
BSD number Y	1	1	-1	0	1	1	-1	0	1	1	-1	0
Intermediate sum U_i	0	0	0	-1	0	0	0	-1	0	0	0	1
Intermediate carry C_{i-1}	0	0	1	1	0	0	1	0	0	0	0	-1
Final sum s_i	0	0	1	0	0	0	1	-1	0	0	0	0

Table 5. Pos-Neg BSD-RNS addition of example 2

	$P=2^4-1$				$P=2^4$				$P=2^4+1$			
position i	3	2	1	0	3	2	1	0	3	2	1	0
BSD number X	1	-1	1	1	1	-1	1	1	1	-1	1	1
BSD number Y	1	1	-1	0	1	1	-1	0	1	1	-1	0
FA1 input X_i^+	1	0	1	1	1	0	1	1	1	0	1	1
FA1 input X_i^-	1	0	1	1	1	0	1	1	1	0	1	1
FA1 input Y_i^+	1	1	0	0	1	1	0	0	1	1	0	0
FA2 input Y_{i}^-	1	1	0	1	1	1	0	1	1	1	0	1
FA2 input d_{i-1}^+	0	1	1	1	0	1	1	0	0	1	1	0
FA2 input U_i^-	1	1	0	0	1	1	0	0	1	1	0	0
Final sum S_{i-1}^-	1	0	1	1	1	0	0	1	1	0	0	0
Final sum S_i^+	0	1	1	0	0	1	1	1	0	1	1	1
Final sum s_i	0	0	1	0	0	0	0	1	0	0	0	0

3. Proposed Pos-Neg Bsd-Rns Multiplier

Let's assume the BSD-RNS with the moduli set $\{2^n-1, 2^n, 2^n+1\}$. The multiplication operations in the three moduli are performed in parallel. Assuming Pos-Neg BSD-RNS, in this section, we generally discuss on modulo m multiplication of two Pos-Neg BSD residues where $m \in \{2^n-1, 2^n, 2^n+1\}$. Any modular multiplier is composed of two important steps namely: a) modular partial product generation and b) modular reduction of the partial products. In the following, we first discuss about modular partial product generation step and then, we focus on the reduction structure.

3.1. Modular Partial Product Generation and Digit Production

The first step of multiplication is consisting of the two following levels: 1) Pos-Neg BSD digit production and generation of normal BSD partial products, 2) rotating BSD partial products according to modulo m rotation rules. Let's assume X and Y are two Pos-Neg BSD residues in modulo p $\in \{2^n-1, 2^n, 2^n+1\}$. The multiplication X×Y is expanded as follows:

$$X \times Y = (x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0) \times (y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + \cdots + y_0) = \sum_{i=0}^{n-1} y_i \times (x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0) \quad (11)$$

Since X and Y are n-digit numbers, there are n partial products represented by pp (0), pp (1), …, and pp (n-1) where obtained as follows:

$$X \times Y = \sum_{i=0}^{n-1} pp_i$$

$$pp_i = y_i \times (x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0) = (z_{n-1}, z_{n-2}, \dots, z_0) \quad (12)$$

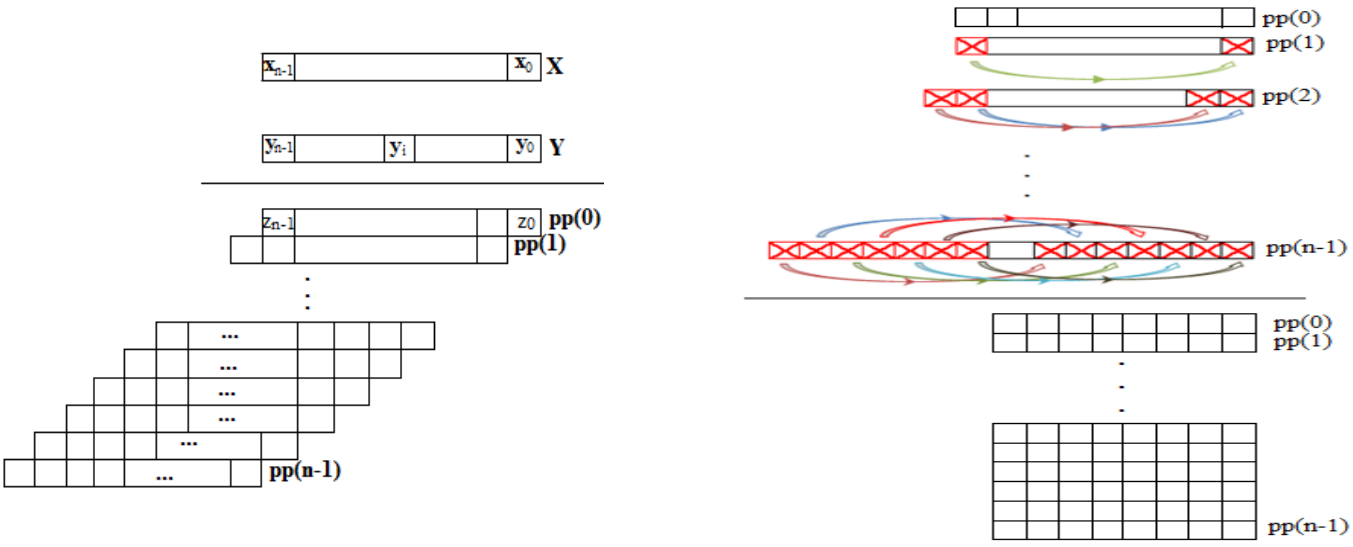
Where, z_k ($k \in \{0, 1, \dots, n-1\}$) is achieved by multiplying the i-th digit of Y (y_i) by the k-th digit of X (x_k). Each digit (z_k) of any partial product is represented with Pos-Neg BSD encoding ($z_k^+ z_k^-$) as shown in table 6. According to the table, there are eight different combinations for ($x_k \times y_i$). If one of the two digits (x_k or y_i) is zero i.e. "01" or "10", then the result will be '0'. If both digits have same sign, then digit production will be '1' with "11" representation. Finally, if two digits have the opposite sign, i.e., (00, 11) or (11, 00), then z_k will be '-1' with "00" representation.

After simplifying table 6, (z_k^+), (z_k^-) are calculated by (13) and (14), respectively.

$$z_k^+ = \overline{(\overline{x_k}.y_i^+) + (x_k^+.\overline{y_i^-})} \quad (13) \qquad z_k^- = (x_k^+.y_i^+) + (\overline{x_k}.\overline{y_i^-}) \quad (14)$$

Table 6. Digit production of two digits with posibit-negabit BSD encoding

x_k	y_i	**Digit Production (z_k)**
00	00	11
11	11	11
01	Don't care	01
Don't care	01	01
10	Don't care	10
Don't care	10	10
00	11	00
11	00	00



(a) (b)

Figure 4. Modular partial products generation in: (a) BSD, and (b) BSD-RNS

Table 7. Rules of partial products rotation for modulo: 2^n-1 and 2^n+1

2^n-1	2^n+1
$pp(0)=\left\|a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ \ldots+a_0\right\|_{2^n-1}$	$pp(0)=\left\|a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ \ldots+a_0\right\|_{2^n+1}$
$pp(1)=\left\|a_{n-1}2^{n}+a_{n-2}2^{n-1}+\ \ldots+a_0 2+0\right\|_{2^n-1}$ $=\left\| a_{n-2}2^{n-1}+\ldots+a_0 2+a_{n-1}\right\|_{2^n}$	$pp(1)=\left\|a_{n-1}2^{n}+a_{n-2}2^{n-1}+\ \ldots+a_0 2+0\right\|_{2^n+1}$ $=\left\| a_{n-2}2^{n-1}+\ldots+a_0 2-a_{n-1}\right\|_{2^n}$
$pp(2)=\left\|a_{n-1}2^{n+1}+a_{n-2}2^{n}+\ldots+a_0 2^2+0+0\right\|_{2^n-1}$ $=\left\|a_{n-3}2^{n-1}+\ldots+a_0 2^2+a_{n-1}2+a_{n-2}\right\|_{2^n}$	$pp(2)=\left\|a_{n-1}2^{n+1}+a_{n-2}2^{n}+\ldots+a_0 2^2+0+0\right\|_{2^n+1}$ $=\left\|a_{n-3}2^{n-1}+\ldots+a_0 2^2-a_{n-1}2-a_{n-2}\right\|_{2^n}$
$pp(n-1)=\left\|a_{n-1}2^{2n-2}+\ldots a_0 2^{n-1}+\underbrace{0+\ldots+0}_{n-1}\right\|_{2^n-1}$ $=\left\|a_0 2^{n-1}+a_{n-1}2^{n-2}+\ldots+a_2 2+a_1\right\|_{2^n}$	$pp(n-1)=\left\|a_{n-1}2^{2n-2}+\ldots a_0 2^{n-1}+\underbrace{0+\ldots+0}_{n-1}\right\|_{2^n+1}$ $=\left\|a_0 2^{n-1}+a_{n-1}2^{n-2}+\ldots-a_2 2-a_1\right\|_{2^n}$

The final partial products of two input operands are shown in Figure 4(a). According to this figure, the generated partial products are not aligned and have to be rotated considering modulo m rotation rules. Utilizing RNS rotation rules for modulo p ϵ {2^n-1, 2^n, 2^n+1}, the generated partial products are aligned.

Table 7 indicates the rotation rules of partial products for the moduli set {2^n-1, 2^n, 2^n+1}. As shown in this table, the i most significant digits of (i+1)-th partial product (i.e. pp'(i)) are rotated into the least significant positions based on the modulus. In modulo 2^n-1, the i rotated digits are located in the least significant i digits without any changes. However, in modulo 2^n+1, the i lest significant digits are replaced by the i most significant digits with opposite sign, and in modulo 2^n, they are replaced with zero. The rotated partial products achieved according to the rules are shown in Figure 4(b).

3.2. Reduction of Partial Products

The second step utilizes the Pos-Neg BSD-RNS adder proposed in [12] to reduce the partial products and obtain the final result as shown in Figure 5. In this structure, each partial product is shown in a row with a rectangular shape that is divided into n squares. Each square in a partial product is explanatory of a digit with Pos-Neg BSD encoding. The reduction of partial products in this structure is performed in $\log_2^n$ steps (n is the length of input operands). In the first step, n/2 Pos-Neg BSD-RNS addition operations are performed in parallel for n/2 groups of partial products i.e., {pp (0), pp (1)}, {pp (2), pp (3)}, …, and {pp (n-2), pp (n-1)}. In this step, the partial products are reduced to n/2 and denoted as pp1 (0), pp1 (1), …, and pp1(n/2-1). In the second step, the same process is performed for the results of the first step and this process will continue until the numbers of partial products reduces to two. In the last step, two remainder partial products are accumulated together with Pos-Neg BSD-RNS adder and the final result is achieved. The delay of a reduction step of partial products is depicted with T_{adder} that is equal by delay of performing a Pos-Neg BSD-RNS adder proposed in [12].

Pos-Neg BSD-RNS multiplication algorithm, $|X\times Y|_p$, where X and Y are two 8-digit Pos-Neg BSD numbers is shown in Fig 6, symbolically. The Pos-Neg Modular Partial

Product Generator, as the first step, generates 8 partial products. The reduction of partial products is done in the second step. As shown in the figure, the reduction is performed in 3 levels that equal to $\log_2^8$. In the first level, four Pos-Neg BSD-RNS addition operations are performed for four groups of partial products as shown in the figure. The results of the additions are denoted as pp1 (0), pp1(1), pp1(2), and pp1(3). Then, the same addition process is performed for the results of first level. We consider two addition operations for the two pairs {pp1(0), pp1(1)} and {pp1(2), pp1(3)}. The results are symbolized as pp2(0) and pp2(1), respectively. The two partial products pp2 (0) and pp2 (1) are accumulated together and the final result is obtained.

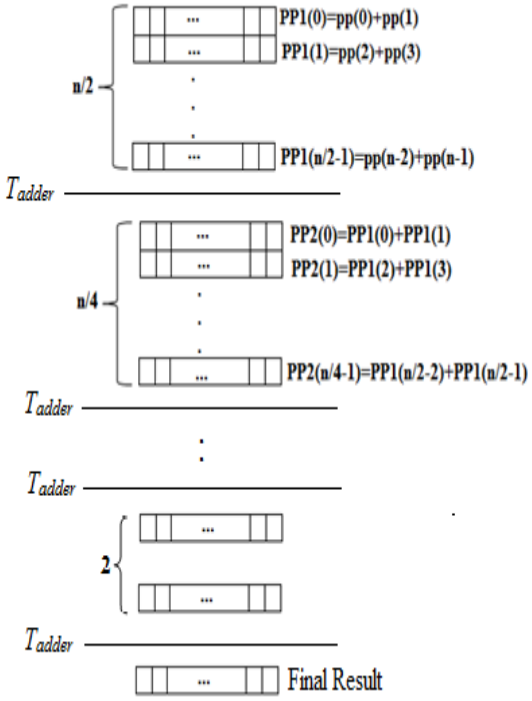


Figure 5. The tree structure for accumulation of partial products proposed in [12]

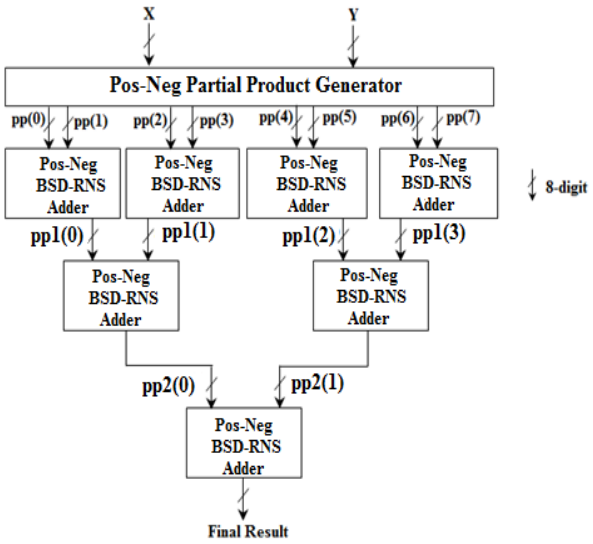


Figure 6. Proposed BSD-RNS multiplier with Pos-Neg BSD encoding

4. Synthesis Results and Comparisons

In this paper, we utilized posibit-negabit encoding in RNS to implement BSD-RNS multiplier. To evaluate speed, area, and power dissipation of the proposed BSD-RNS multiplier, firstly, we generated the structural VHDL descriptions of them for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ where (n=8 and n=16). Secondly, we synthesized the structures using Synopsys Design Vision tool and the TSMC 180 nm CMOS standard cell library with typical conditions (1.8 V, 25□c). Then, we compared the proposed structure with existing 2's complement BSD-RNS and 1-out-of-3 BSD-RNS multipliers proposed in [7] and [13], respectively.

The synthesis results of area, delay, and power of 2's complement BSD-RNS, 1- out-of-3 BSD-RNS, and Pos-Neg BSD-RNS multipliers are reported in tables 8, 9, and 10, respectively. Delay, area, and power results are given in ns, μm^2, and mw, respectively.

Table 8. Area of BSD-RNS multipliers for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ (n=8 and 16) in μm^2

n	2's complement BSD-RNS			1-out-of-3 BSD-RNS			Pos-Neg BSD-RNS		
	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1
8	36690.2	22712.65	38170.44	34810.77	21954.24	34854.01	30193.73	21531.48	31073.56
16	135614.1	90770.8	142303.3	129303.8	87727.14	129410.2	96036.49	68826.54	120106.3

Table 9. Delay of BSD-RNS multipliers for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ (n=8 and16) in ns

N	2's complement BSD-RNS			1-out-of-3 BSD-RNS			Pos-Neg BSD-RNS		
	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1
8	1.73	1.7	1.74	1.59	1.58	1.6	1.4	1.36	1.41
16	2.69	2.48	2.54	2.27	2.14	2.28	1.86	1.89	1.89

Table 10. Power consumption of BSD-RNS multipliers for the moduli set{2^n-1, 2^n, 2^n+1}(n=8 and16) in mw

n	2's complement BSD-RNS			1-out-of-3 BSD-RNS			Pos-Neg BSD-RNS		
	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1	2^n-1	2^n	2^n+1
8	20.2079	8.9727	20.9256	22.1438	9.6099	21.7315	18.7015	7.7814	19.5201
16	78.9516	37.8025	87.9746	90.5531	47.456	94.0302	75.4042	33.366	82.5366

Table 11. Performance improvement percentage of the proposed Pos-Neg BSD-RNS multiplier in comparison to conventional BSD-RNS multipliers for the moduli set {2^8-1, 2^8, 2^8+1}

BSD-RNS multiplier	Area Improvement			Delay Improvement			Power Improvement		
	2^8-1	2^8	2^8+1	2^8-1	2^8	2^8+1	2^8-1	2^8	2^8+1
2's Complement	17%	5%	19%	19%	20%	18.9%	7%	13%	7%
1-out-of-3	13%	1.92%	11%	12%	14%	12%	16%	19%	10%

Table 12. Performance improvement percentage of the proposed Pos-Neg BSD-RNS multiplier in comparison to conventional BSD-RNS multipliers for the moduli set {2^{16}-1, 2^{16}, 2^{16}+1}

BSD-RNSmultiplier	Area Improvement			Delay Improvement			Power Improvement		
	2^{16}-1	2^{16}	2^{16}+1	2^{16}-1	2^{16}	2^{16}+1	2^{16}-1	2^{16}	2^{16}+1
2's Complement	29%	24%	16%	31%	24%	26%	5%	12%	6%
1-out-of-3	26%	22%	7%	18%	12%	17%	17%	30%	12%

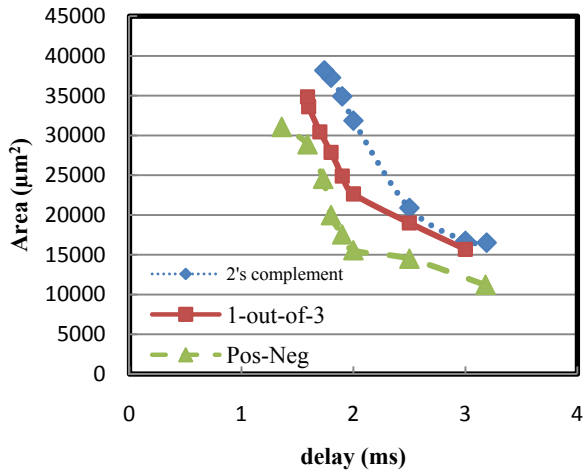


Figure 7. Area of the BSD-RNS multipliers versus different delays for 2^8+1

The synthesis results indicate that the proposed modular multiplier has the least delay, area, and power consumption among the BSD-RNS multipliers. Utilizing standard compressor 4:2 in the Pos-Neg BSD-RNS adder for accumulation of partial products leads to have more efficient BSD-RNS multiplier than two others BSD-RNS multipliers. Also, the improvement percentages of proposed modular multiplier in comparison to 2's complement BSD-RNS and 1-out-of-3 BSD-RNS multipliers are illustrated in tables 11 and 12.

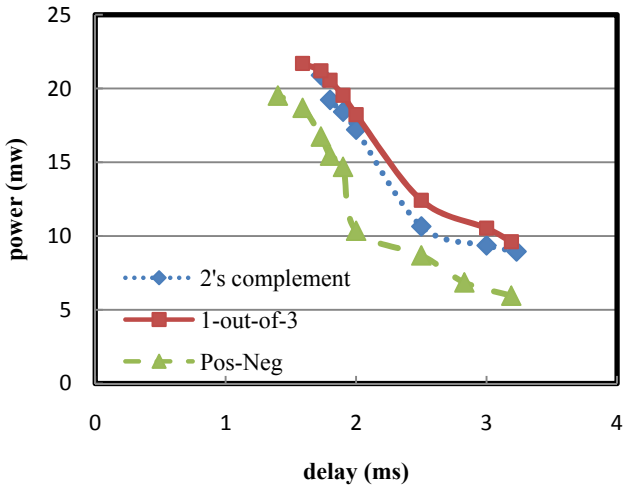


Figure 8. Power consumption of the BSD-RNS multipliers versus different delays for 2^8+1

Consequently, it can be concluded that the proposed Pos-Neg BSD-RNS multiplier is the best modular multiplier

among the BSD-RNS multipliers from the viewpoint of delay, area, and power.

For better comparison, the syntheses were done for different delays starting from the least delay of each multiplier. Area and power versus delay are shown in Figure 7, and Figure 8 for n = 8. These figures show that the area and power of three BSD-RNS multipliers are increased with decreasing their delays. However, for different delays, the proposed multiplier outperform area, and power of the two others multipliers.

5. Conclusion

Binary Signed-Digit (BSD) has been utilized in RNS for implementation of high-speed residue arithmetic that is called BSD-RNS. In this paper, we focus on designing Pos-Neg BSD-RNS multiplier. It is depicted that the proposed design consumes less delay, area, and power than existing modular multipliers with 2's complement and 1-out-of-3 BSD number representations. The improvement percentage of delay, area, and power of the proposed BSD-RNS multiplier for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ with two different length (n=8 and n=16) were presented. For example, for modulo 2^8-1, the proposed structure consumes 19% less delay, 17% less area, and 7% less power, compared to 2's complement BSD-RNS multiplier.

Besides, for the modulo 2^8-1 the proposed architecture consumes 12% less delay, 13% less area, and 16% less power than 1-out-of-3 BSD-RNS multiplier reported in [13]. So it can be concluded that the proposed Pos-Neg BSD-RNS multiplier in terms of delay, area, and power consumption is appropriate than 1-out-of-3 BSD-RNS multiplier reported in [13]. So it can be concluded that the proposed Pos-Neg BSD-RNS multiplier in terms of delay, area, and powerconsumption is appropriate than the other two modular multipliers and can be used in the applications that require frequent multiplications such as digital signal processing [1], digital filtering [2], and communication [3].

References

[1] I. Kouretas, and V. Paliouras, "High-radix Residue Arithmetic Bases for Low-power DSP Systems," *IEEE Intl Conf. Digital Signal Processing*, pp. 1- 6, 2009.

[2] R. Conway, and J. Nelson, "Improved RNS FIR Filter Architectures," *IEEE Trans. Circuits and Systems-II*, vol. 51, no. 1, pp. 26-28, 2004.

[3] A. S. Madhukumar, and F. Chin, "Enhanced Architecture for Residue Number System-based CDMA for High-rate Data Transmission," *IEEE Trans. Wireless*, vol. 3, no. 5, pp. 1363-1368, 2004.

[4] L. Zhining, and B. J. Phillips, "An RNS-enhanced Microprocessor Implementation of Public Key Cryptography," *IEEE Intl Conf. Signals, Systems and Computers*, pp.1430-1434, 2007.

[5] S. Timarchi, and M. Fazlali, "Generalized Fault-tolerant Stored-Unibit-Transfer RNS Multiplier for Moduli Set {2n-1, 2n, 2n+1}," *IET Journal on Computers and Digital Techniques*, vol. 6, no. 3, pp. 269-276, 2012.

[6] A. Avizienis, "Signed-digit Number Representations for Fast Parallel Arithmetic," *IRE Trans. EIectronic Computers*, vol. 5, no. 2, pp. 389-400, 1961.

[7] S. Wei, "Modular Multipliers Using a Modified Residue Addition Algorithm with Signed-Digit Number Representation," *IEEE Intl Conf. Engineers and Computer Scientists*, pp. 62-70, 2009.

[8] F. Kharbash, and G. M.Chaudhry, "Binary Signed Digit Number Adder Design with Error Detection Capability," *IEEE Intl Symp. Signal Processing and Its Applications*, pp. 1-4, 2007.

[9] G. Jaberipur, B. Parhami, and M. Ghodsi, "Weighted Two-Valued Digit-Set Encodings: Unifying Efficient Hardware Representation Schemes for Redundant Number Systems," *IEEE Trans. Circuits and Systems I*, vol. 52, no. 7, pp. 1348-1357, 2005.

[10] M. Saremi, and S. Timarchi, "1-out-of-3 Binary Signed-Digit Moular Adder," *IEEE Intl Conf. Information and Knowledge Technology*, pp. 43-51, 2013.

[11] S. Timarchi, P. Ghayour, and A. Shahbahrami, "A Novel High-Speed Low-Power Binary Signed-Digit Adder," *IEEE Intl Symp. Computer Architecture and Digital Systems*, pp. 357-363, 2012.

[12] S. Timarchi, M. Saremi, M. Fazlali, and G. Gaydadjiev, "High-speed Binary Signed-digit RNS Adder with Posibit and Negabit Encoding," *IEEE Intl Conf. Very Large Scale Integration*, pp. 63-71, 2013.

[13] M. Saremi, and S. Timarchi, "Efficient 1-out-of-3 Binary Signed-Digit Multiplier for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$," *IEEE Intl Symp. Computer Architecture and Digital Systems*, pp. 26-32, 2013.

[14] Sh. Wei, and K.. Shimizu, "Residue Arithmetic Circuits Based on Signed-Digit Number Representation and the VHDL Implementation," *IEEE Intl Symp. Very Large Scale Integration Systems*, pp. 218-221, 1999.

[15] P. K. Lala, *Self Checking and Fault-tolerance Digital Design*, San Francisco: Morgan Kaufmann Publishers, 2001.

Maryam Saremi Maryam Saremi received the B.Sc. degree in electronic engineering from Shahid Chamran University in 2011. She received M.Sc. degree in Shahid Beheshti University of Tehtan, Iran in electrical and electronic engineering in 2013. In 2014, she joined the department of Electrical Engineering of the Khatam Alanbia of behbahan University as an educator. She has researched about arithmetic computer, Residue Number System, Redundant Number System and reported four papers up to now.
E-mail: m.saremi@mail.sbu.ac.ir

Somayeh Timarchi received her B.Sc. degree in Computer Engineering (Hardware) from ShahidBeheshti University in 2002 .She received M.Sc. and Ph.D. degrees in Computer System Architecture from Sharif University of Technology and ShahidBeheshti University, in 2004 and 2010, respectively. In 2010, she joined the Department of Electrical Engineering of the ShahidBeheshti University as an Assistant Professor.

She has authored or co-authored more than 20 publications on journals and conference proceedings. One of them is among the 10 highly cited papers during 2010-2013 in Integration, the VLSI Journal. Her research interests include Computer Arithmetic, Residue and Redundant Number System, VLSI Design, Computer Architecture, low-power and ultra-low-power digital circuits.

E-mail: s_timarchi@sbu.ac.ir

Paper Handling Data:

Submitted: 18.11.2013
Received in revised form: 09.03.2014
Accepted: 06.04.2014
Corresponding author: Dr. Somayeh Timarchi,
Department of Electrical and Computer Engineering,
Shahid Beheshti University, G. C., Tehran, Iran.