Efficient MRC-Based Residue to Binary Converters for the New Moduli Sets \( \{2^n, 2^n - 1, 2^{n+1} - 1\} \) and \( \{2^{2n}, 2^n - 1, 2^{n-1} - 1\} \)

Amir Sabbagh MOLAHOSSEINI\(^{(a)}\), Chitra DADKHAH\(^{(b)}\), Keivan NAVI\(^{(c)}\), Nonmembers, and Mohammad ESHGHI\(^{(d)}\), Member

SUMMARY In this paper, the new residue number system (RNS) moduli sets \( \{2^{2n}, 2^n - 1, 2^{n+1} - 1\} \) and \( \{2^{2n}, 2^n - 1, 2^{n-1} - 1\} \) are introduced. These moduli sets have 4n-bit dynamic range and well-formed moduli which can result in high-performance residue to binary converters as well as efficient RNS arithmetic unit. Next, efficient residue to binary converters for the proposed moduli sets based on mixed-radix conversion (MRC) algorithm are presented. The converters are ROM-free and they are realized using carry-save adders and modulo adders. Comparison with the other residue to binary converters for 4n-bit dynamic range moduli sets shown that the presented designs based on new moduli sets \( \{2^{2n}, 2^n - 1, 2^{n+1} - 1\} \) and \( \{2^{2n}, 2^n - 1, 2^{n-1} - 1\} \) are improved the conversion delay and result in hardware savings. Also, the proposed moduli sets can lead to efficient binary to residue converters, and they can speed-up internal RNS arithmetic processing, compared with the other 4n-bit dynamic range moduli sets.

key words: residue to binary converter, mixed-radix conversion (MRC), residue number system (RNS), computer arithmetic

1. Introduction

The residue number system (RNS) is a carry-free number system, which has been used as an alternative to the weighted number system for achieving high-performance computing systems [1]. In RNS, the weighted operands are converted into residue representations. Then, arithmetic operations such as addition, subtraction and multiplication are performed on RNS numbers in parallel without carry propagation between residue digits. Hence, RNS results in parallel and high-speed addition, subtraction and multiplication. [2], [3]. The RNS has many applications in digital signal processing (DSP) [4], [5]. In particular, RNS is an interesting and useful tool for the implementation of high-speed FIR filters [6], [7]. Also, implementation issues of some RNS-based DSP processors are reported in [8] and [9]. RNS has applications in image processing systems, especially RNS image coding can offers high-speed VLSI implementation of secure image processing algorithms [10]. In addition to these, redundant RNS offers new approaches to the design of the error detection and correction codes [11], [12].

The dynamic range (DR) of an RNS system is defined in terms of the product of the moduli of the moduli set, and it denotes the interval of integers can be uniquely represented in RNS. The proper selection of a moduli set has an important role in the design of the RNS systems, because the speed of internal RNS arithmetic circuits as well as the complexity of residue to binary converter depend on the form and the number of the moduli [2]. The powers of two related numbers such as \( 2^k \) and \( 2^k \pm 1 \), have been extensively used in RNS moduli sets. Because, RNS systems based on these moduli can be efficiently implemented by using usual binary hardwares. Specially, the modulo adders and multipliers for the moduli of the form \( 2^k \) are just the binary adders and multipliers with truncation [13]. Also, operations in modulo \( 2^k - 1 \) are regular and can be easily performed by end-around carry scheme. But modulo \( 2^k + 1 \) operations are much more complex and are usually the bottleneck for RNS arithmetic units [14].

The moduli sets which have been suggested for RNS, can be classified based on their DR. The 3n-bit DR moduli sets are \( \{2^n - 1, 2^n, 2^n + 1\} \) [15], \( \{2^{n-1} - 1, 2^n - 1, 2^n\} \) [16], and \( \{2^n - 1, 2^n, 2^{n+1} - 1\} \) [17], [18]. The DR provided by these moduli sets is not sufficient for applications which require larger dynamic range. Hence, 4n-bit DR four-moduli sets such as \( \{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\} \) [19], [20], \( \{2^n - 1, 2^n, 2^n + 1, 2^n - 1\} \) [20]–[22] and \( \{2^n - 3, 2^n + 1, 2^n - 1, 2^n\} \) [23] have been introduced. These four-moduli sets include balanced moduli, but some of their multiplicative inverses have inelegant forms which resulted in increasing the cost and the delay of the residue to binary converter. Furthermore, with increasing the number of moduli in the moduli set, the complexity of RNS is also increase. So, the RNS systems based on four-moduli sets are more complex than those based on three-moduli sets. Hence, three-moduli sets with supporting larger DR than the traditional three-moduli sets such as \( \{2^n - 1, 2^n, 2^{n+1} + 1\} \) [24], \( \{2^n - 1, 2^n + 1, 2^{n+1} + 1\} \) [25] and \( \{2^n, 2^{n+1} - 1, 2^{n+1}\} \) [26] have been proposed. The first and second moduli sets have 4n-bit DR, while the third has 5n-bit DR. Due to the mathematical properties of the moduli sets \( \{2^n - 1, 2^n + 1, 2^{n+1}\} \) and \( \{2^n, 2^{2n} - 1, 2^{2n} + 1\} \), they have simple multiplicative inverses for residue to binary converter design, but the modulo \( 2^{2n} + 1 \) leads to increasing the execution latency of the RNS arithmetic units. Recently, in [27] the general three-moduli set \( \{2^n, 2^n - 1, 2^n\} \) where \( \alpha < \beta \) introduced for offering large dynamic range with low complexity, and an adder-based residue to binary converter for
this moduli set is presented in [27]. The residue to binary converter architecture of [27] is very simple and efficient but the time performance of arithmetic unit of RNS systems based on its moduli set is restricted to the modulo $2^n + 1$. Also, with the general three-moduli set $\{2^n, 2^n - 1, 2^n+1 \}$ where $\alpha < \beta$, we cannot derive a 4n-bit DR moduli set.

In this paper, after describing necessary background in Sect. 2, we concentrate on 4n-bit DR moduli sets, and introduce two new 4n-bit DR RNS moduli sets $\{2^{2n}, 2^n - 1, 2^{n+1} - 1 \}$ and $\{2^{2n}, 2^n - 1, 2^{n-1} - 1 \}$ where $n \geq 4$. All of the moduli of the proposed moduli sets are in the forms $2^k$ or $2^k - 1$, and this leads to simple and efficient implementation of the binary to residue and residue to binary converters as well as internal RNS arithmetic circuits. Next, efficient designs of residue to binary converter for the moduli sets $\{2^{2n}, 2^n - 1, 2^{n+1} - 1 \}$ and $\{2^{2n}, 2^n - 1, 2^{n-1} - 1 \}$ based on mixed-radix conversion (MRC) algorithm are presented in Sect. 3. The converter architectures are adder-based and pipelinable, resulting in high-performance hardware designs. The proposed residue to binary converters have lower hardware costs and better conversion delays than the other residue to binary converters for 4n-bit DR moduli sets, and also the new introduced moduli sets are result in efficient binary to residue converters and they can speed-up internal RNS arithmetic processing than the other 4n-bit DR moduli sets as shown in Sect. 4.

2. Background

A residue number system is defined in terms of relatively-prime moduli set $\{P_1, P_2, \ldots, P_n\}$ that is gcd($P_i, P_j$) = 1 for $i \neq j$. A weighted number $X$ can be represented as $X = (x_1, x_2, \ldots, x_n)$, where

\[ x_i = X \mod P_i = [X]_{P_i}, \quad 0 \leq x_i < P_i \]

Such a representation is unique for any integer $X$ in the range $[0, M - 1]$, where $M = P_1P_2\cdots P_n$ is the dynamic range of the moduli set $\{P_1, P_2, \ldots, P_n\}$ [28]. Addition, subtraction and multiplication on residues can be performed in parallel, because there is no carry propagation between residue digits in RNS.

Each RNS-based processing system consists of a binary to residue converter, an arithmetic unit, and a residue to binary converter. The binary to residue converter encode a weighted binary number into a residue represented number, with regard to the moduli set, and can be implemented by using multi-operand modular adders. The RNS arithmetic unit includes modular adder, subtractor and multiplier for each modulo channel. The residue to binary converter decode a residue represented number into its equivalent weighted binary number. The residue to binary converter is very important part of the RNS systems. Because, conversion delay should not counteract the speed gain of RNS arithmetic unit. Besides, the residue to binary conversion can be used for performing difficult RNS operations such as division, sign detection and magnitude comparison [2]. The algorithms of residue to binary conversion are principally based on Chinese remainder theorem (CRT) and mixed-radix conversion. Now, consider the RNS number $(x_1, x_2, \ldots, x_n)$ with moduli set $(P_1, P_2, \ldots, P_n)$. In the following, the residue to binary conversion algorithms are described.

By CRT [2], the number $X$ can be calculated from its residue representation by

\[ X = \sum_{i=1}^{m} [x_iP_i]_{P_i} \]

where

\[ M = P_1P_2\cdots P_n \]
\[ M_i = M/P_i \]
\[ N_i = [M_i^{-1}]_{P_i} \]

The $N_i$ is the multiplicative inverse of $M_i$ modulo $P_i$. The CRT can be implemented in parallel channels followed by a modulo $M$ adder. The problem is that, this modulo adder is very big, and it can results in inefficient hardware implementation for residue to binary converter.

By MRC [2], [29], an RNS number can be converted into its equivalent weighted binary number $X$ as follows

\[ X = \prod_{i=1}^{n-1} P_i + \cdots + v_3P_2P_1 + v_2P_1 + v_1 \]

The coefficients $v_i$ can be obtained from residues by

\[ v_1 = x_1 \]
\[ v_2 = \left( (x_2 - v_1)P_1^{-1} \right)_{P_2} \]
\[ v_3 = \left( (x_3 - v_1)P_1^{-1}P_2^{-1} - v_2 \right)_{P_3} \]

In the general case, we have

\[ v_n = \left( (\ldots ((x_n - v_1)P_1^{-1}P_2^{-1}\ldots - v_{n-1})P_{n-1}^{-1} - v_{n-2}) \ldots - v_2 \right)_{P_n} \]

where $[P_i^{-1}]_{P_i}$ denotes the multiplicative inverse of $P_i$ modulo $P_j$. MRC is a sequential algorithm which is not suitable for moduli sets with more than four moduli. But, for two and three-moduli sets with well-formed moduli, MRC can lead to simple residue to binary conversion equations which can be efficiently realized in hardware.

3. Residue to Binary Converters

In this section, MRC is used to derive efficient residue to binary conversion algorithms for the moduli sets $\{2^{2n}, 2^n - 1, 2^{n+1} - 1 \}$ and $\{2^{2n}, 2^n - 1, 2^{n-1} - 1 \}$. Also, adder-based hardware implementations of the conversion algorithms are presented.
3.1 Converter for the Moduli Set \{2^{2n}, 2^n - 1, 2^{n+1} - 1\}

For design of residue to binary converter, firstly the values of moduli of the moduli set must substitute in conversion algorithm formulas. Then, the resulted equations should be simplified by using arithmetic properties. Finally, simplified equations would realize using hardware components such as full adders, half adders and logic gates. The following Theorems, Lemmas and Properties are needed for derivation of residue to binary conversion algorithm.

**Theorem 1:** The moduli set \{2^{2n}, 2^n - 1, 2^{n+1} - 1\} includes pairwise relatively prime moduli.

**Proof:** Since \(2^n - 1\) and \(2^{n+1} - 1\) are two odd numbers, and \(2^{2n}\) is an even number, it is clear that \(2^{2n}\) is relatively prime to the numbers \(2^n - 1\) and \(2^{n+1} - 1\). Also, the moduli \(2^n - 1\) and \(2^{n+1} - 1\) are relatively prime, because they are previously used in the RNS moduli set \(\{2^n - 1, 2^n, 2^{n+1} - 1\}\) [17].

**Lemma 1:** The multiplicative inverse of \(2^{2n}\) modulo \((2^n - 1)\) is \(k_1 = 1\).

**Proof:** It is easy to find that,
\[
\left[2^{2n}\right]_{2^n-1} = 1
\]
(11)
Therefore,
\[
\left[k_1 \times 2^{2n}\right]_{2^n-1} = \left[1 \times 2^{2n} \times 2^{2n}\right]_{2^n-1} = \left[1 \times 1 \times 2^{2n}\right]_{2^n-1} = 1
\]
(12)

**Lemma 2:** The multiplicative inverse of \(2^n\) modulo \((2^{n+1} - 1)\) is \(k_2 = 2^2\).

**Proof:** By substituting the value of \(k_2\), we have
\[
\left[k_2 \times 2^n\right]_{2^{n+1}-1} = \left[2^2 \times 2^n\right]_{2^{n+1}-1} = \left[2^{2n+2}\right]_{2^{n+1}-1} = \left[2^{n+1} \times 2^n\right]_{2^{n+1}-1} = \left[1 \times 1 \times 2^{n+1}\right]_{2^{n+1}-1} = 1
\]
(13)

**Lemma 3:** The multiplicative inverse of \((2^n - 1)\) modulo \((2^{n+1} - 1)\) is \(k_3 = -2^2\).

**Proof:** It is clear that,
\[
\left[k_3 \times (2^n - 1)\right]_{2^{n+1}-1} = \left[-2 \times (2^n - 1)\right]_{2^{n+1}-1} = \left[-2^n + 2n\right]_{2^{n+1}-1} = \left[-1 + 2n\right]_{2^{n+1}-1} = 1
\]
(14)

**Theorem 2:** In the RNS based on the moduli set \(\{P_1, P_2, P_3\} = \{2^{2n}, 2^n - 1, 2^{n+1} - 1\}\), the number \(X\) can be calculated from its corresponding residues \((x_1, x_2, x_3)\) by
\[
X = x_1 + 2^n Z + 2^{2n}(2^n - 1)T
\]
(15)
where
\[
Z = \left|x_2 - x_1\right|_{2^n-1}
\]
(16)
\[
T = \left|((x_3 - x_1)2^n - Z(-2))\right|_{2^{n+1}-1}
\]
(17)

**Proof:** By letting \(P_1 = 2^{2n}\), \(P_2 = 2^n - 1\), \(P_3 = 2^{n+1} - 1\) and the values of multiplicative inverses from Lemmas 1, 2 and 3 into (6)–(9), Eqs. (15)–(17) are obtained.

Theorem 2 is used for designing an efficient residue to binary converter. But before implementing it, according to the following properties, (15)–(17) can be simplified for deriving an efficient hardware design.

**Property 1:** The residue of a negative residue number \((-v)\) in modulo \((2^n - 1)\) is the one’s complement of \(v\), where \(0 \leq v < 2^n - 1\) [26].

**Property 2:** The multiplication of a residue number \(v\) by \(2^p\) in modulo \((2^n - 1)\) is carried out by \(P\) bits circular left shift, where \(P\) is a natural number [26].

Consider the three-moduli set \(\{2^{2n}, 2^n - 1, 2^{n+1} - 1\}\) and let the corresponding residues of the integer number \(X\) be \((x_1, x_2, x_3)\). The residues have bit-level representation as
\[
x_1 = (x_{1,2n-1} x_{1,2n-2} \cdots x_{1,1} x_{1,0})_{2}
\]
(18)
\[
x_2 = (x_{2,n-1} x_{2,n-2} \cdots x_{2,1} x_{2,0})_{2}
\]
(19)
\[
x_3 = (x_{3,n} x_{3,n-1} \cdots x_{3,1} x_{3,0})_{2}
\]
(20)
\[
\text{Equation (15) can rewrite as}
\]
\[
X = x_1 + 2^n H
\]
(21)
where
\[
H = Z + (2^n - 1)T
\]
(22)
\[
Z = \left|x_2 - x_1\right|_{2^n-1}
\]
(23)
\[
T = \left|((x_3 - x_1)2^n - Z(-2))\right|_{2^{n+1}-1}
\]
(24)

Firstly, (23) is simplified as follow
\[
Z = \left|w_1 + w_2\right|_{2^n-1}
\]
(25)

where
\[
w_1 = \left|x_2\right|_{2^n-1} = \left|x_{2,n-1} \cdots x_{2,1} x_{2,0}\right|_{2^n-1} = x_{2,n-1} \cdots x_{2,1} x_{2,0}
\]
(26)
\[
w_2 = \left|-x_1\right|_{2^n-1} = \left|-(x_{1,2n-1} \cdots x_{1,1} x_{1,0})\right|_{2^n-1} = -(x_{1,2n-1} \cdots x_{1,1} x_{1,0})_{2^n-1}
\]
(27)

Now, (27) can be divided in two parts as below
\[
w_{21} = \left|-x_{1,n-1} \cdots x_{1,1} x_{1,0}\right|_{2^n-1} = x_{1,n-1} \cdots x_{1,1} x_{1,0}
\]
(28)
\[
w_{22} = \left|-2^n(x_{1,n-1} \cdots x_{1,1} x_{1,0})\right|_{2^n-1} = x_{1,n-1} \cdots x_{1,1} x_{1,0}
\]
(29)

Therefore, (25) become as follows
\[
Z = \left|w_1 + w_{21} + w_{22}\right|_{2^n-1}
\]
(30)

Next, we simplify (24) by rewriting it as
\[ T = \left| 2Z - 2^3(x_3 - x_1) \right|_{2^{n+1}-1} \]  
\[ (31) \]

Equation (31) can be separated into four parts

\[ w_3 = \left| 2Z \right|_{2^{n+1}-1} = \left| 2(0Z_{n-1} \cdots Z_1Z_0) \right|_{2^{n+1}-1} \]
\[ = Z_{n-1} \cdots Z_1Z_00 \]  
\[ w_4 = \left| -2^3x_3 \right|_{2^{n+1}-1} = \left| -2^3(x_3n \cdots x_31x_30) \right|_{2^{n+1}-1} \]
\[ = \bar{x}_{3,n-3} \cdots \bar{x}_{3,1}\bar{x}_{3,0}\bar{x}_{3,n-1}\bar{x}_{3,n-2} \]  
\[ w_5 = \left| 2^3x_1 \right|_{2^{n+1}-1} = \left| 2^3(x_{1,2n-1} \cdots x_{1,1}x_{1,0}) \right|_{2^{n+1}-1} \]
\[ = \left| 2^3(x_{1,2n-1} \cdots x_{1,n+2}x_{1,n+1}) \right|_{2^{n+1}-1} \]
\[ = \left| 2^3 \times 2^{n+1}(00x_{1,2n-1} \cdots x_{1,n+2}x_{1,n+1}) \right|_{2^{n+1}-1} \]
\[ = x_{1,2n-2} \cdots x_{1,n+2}x_{1,n+1}00x_{1,2n-1} \]
\[ w_{51} = \left| 2^3(x_{1,n} \cdots x_{1,1}x_{1,0}) \right|_{2^{n+1}-1} \]
\[ = x_{1,n-3} \cdots x_{1,1}x_{1,0}x_{1,n}x_{1,n-1}x_{1,n-2} \]
\[ w_{52} = \left| 2^3 \times 2^{n+1}(00x_{1,2n-1} \cdots x_{1,n+2}x_{1,n+1}) \right|_{2^{n+1}-1} \]
\[ = x_{1,2n-2} \cdots x_{1,n+2}x_{1,n+1}00x_{1,2n-1} \]
\[ (34) \]

The above equation can be parsed as follow

\[ w_{51} = \left| 2^3(x_{1,n} \cdots x_{1,1}x_{1,0}) \right|_{2^{n+1}-1} \]
\[ = x_{1,n-3} \cdots x_{1,1}x_{1,0}x_{1,n}x_{1,n-1}x_{1,n-2} \]
\[ w_{52} = \left| 2^3 \times 2^{n+1}(00x_{1,2n-1} \cdots x_{1,n+2}x_{1,n+1}) \right|_{2^{n+1}-1} \]
\[ = x_{1,2n-2} \cdots x_{1,n+2}x_{1,n+1}00x_{1,2n-1} \]
\[ (35) \]
\[ (36) \]

Therefore, (31) can be calculated by

\[ T = \left| w_3 + w_4 + w_{51} + w_{52} \right|_{2^{n+1}-1} \]
\[ (37) \]

Finally, \( H \) in (22) is obtained as

\[ H = P - Q \]
\[ (38) \]

where

\[ P = Z + 2^nT = T_n \cdots T_1T_0Z_{n-1} \cdots Z_0 \]
\[ (39) \]
\[ Q = T = 0 \cdots 00T_n \cdots T_1T_0 \]
\[ (40) \]

\[ \text{Example 1:} \] Consider the moduli set \( \{2^{2n}, 2^n - 1, 2^{n+1} - 1\} \) where \( n = 4 \). The weighted number \( X \) can be calculated from its RNS representation \((97, 12, 29)\) as follow

For \( n = 4 \) the moduli set is \( \{256, 15, 31\} \) and also residues have binary representation as

\[ x_1 = 97 = (01100001)_2 \]
\[ x_2 = 10 = (1000)_2 \]
\[ x_3 = 29 = (11101)_2 \]

By letting values of residues and \( n = 4 \) in (26), (28), (29) and (30), we have

\[ w_1 = (11000)_2 = 12 \]
\[ w_2 = (1001)_2 = 9 \]
\[ w_3 = (11100)_2 = 14 \]
\[ Z = 12 + 9 + 14 \]
\[ = 5 = 0(0101)_2 \]

Next, From (32), (33), (35), (36) and (37), we have

\[ w_3 = (01010)_2 = 10 \]
\[ w_4 = (10000)_2 = 16 \]
\[ w_5 = (11000)_2 = 24 \]
\[ w_{52} = (01000)_2 = 8 \]
\[ T = 10 + 16 + 24 + 8 \]
\[ = 27 = (11011)_2 \]

Finally, by substituting the required values in (39), (40), (38) and (21), we have

\[ P = (1101110101)_2 = 437 \]
\[ Q = (0000111011)_2 = 27 \]
\[ H = 437 - 27 = 410 = (110011010)_2 \]
\[ X = 97 + 2^8 \times 410 = (11001101001100001)_2 = 105057 \]

To verify the result, we calculate the residues as follow

\[ x_1 = [105057]_{256} = 97 \]
\[ x_2 = [105057]_{15} = 12 \]
\[ x_3 = [105057]_{31} = 29 \]

Therefore, the weighted integer number \( 105057 \) in the RNS based on the moduli set \( \{256, 15, 31\} \) has representation as \((97, 12, 29)\).

Hardware architecture of the residue to binary converter for the moduli set \( \{2^{2n}, 2^n - 1, 2^{n+1} - 1\} \) is shown in Fig. 1. Implementation is based on Eqs. (30), (37), (38) and (21). Firstly the operand preparation unit (OPU) prepares the needed operands (26), (28) and (29) by using simply manipulating the routing of the bits of the residues, and some NOT gates. \( 2n \) NOT gates are used for performing the inversions of (28) and (29). Implementation of (30) requires a 3-operand modulo \( (2^n - 1) \) adder \[30\] which consists of an \( n \)-bit carry-save adder (CSA) with end-around carry (EAC) followed by a modulo \( (2^n - 1) \) adder. The modulo \( (2^n - 1) \) adder can be implemented with different methods \[30\]–\[33\]. In this paper, we considered the carry propagate adder (CPA) with EAC for the implementation of the modulo adders of the residue to binary converters. Therefore, modulo \( (2^n - 1) \) adder relies on an \( n \)-bit CPA with EAC \[30\]. The delay of an CSA with EAC is the same as that of a full adder (FA), and the delay of an CPA with EAC is twice of the delay of a regular CPA, while has the same hardware complexity.

Realization of (37) relies on a 4-operand modulo
Table 1 Characterization of each part of the proposed residue to binary converter for moduli set \([2^{2n}, 2^n - 1, 2^{n-1} - 1]\).

<table>
<thead>
<tr>
<th>Parts</th>
<th>FA</th>
<th>NOT /AND pairs</th>
<th>XOR /OR pairs</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU1</td>
<td>n</td>
<td>2n</td>
<td>n</td>
<td>t_{NOT}</td>
</tr>
<tr>
<td>CSA1</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>t_{EA}</td>
</tr>
<tr>
<td>CPA1</td>
<td>n</td>
<td>n</td>
<td>2n</td>
<td>t_{EA}</td>
</tr>
<tr>
<td>OPU2</td>
<td>n-1</td>
<td>n</td>
<td>n</td>
<td>t_{NOT}</td>
</tr>
<tr>
<td>CSA2</td>
<td>n</td>
<td>1</td>
<td>n</td>
<td>t_{EA}</td>
</tr>
<tr>
<td>CPA2</td>
<td>n+1</td>
<td>n</td>
<td>2n+2</td>
<td>t_{EA}</td>
</tr>
<tr>
<td>OPU3</td>
<td>n-1</td>
<td>n</td>
<td>n</td>
<td>t_{NOT}</td>
</tr>
<tr>
<td>CPA3</td>
<td>n+1</td>
<td>n</td>
<td>n</td>
<td>t_{EA}</td>
</tr>
<tr>
<td>Total</td>
<td>6n+1</td>
<td>4n+2</td>
<td>3</td>
<td>n \times (6n+6) \times t_{EA} + 3t_{NOT}</td>
</tr>
</tbody>
</table>

Subtractor can be implemented by a \((2n+1)\)-bit regular CPA with ‘1’ carry-in, and \((n+1)\) NOT gates. Also, since the inversion of \(Q(40)\) has \(n\) bits of 1’s,\(n\) FA’s of the CPA3 are reduced to \(n\) pairs of XOR/OR gates. It should be noted that the carry-out of CPA3 will be ignored. Also, since \(x_1\) is a \(2n\)-bit number, no computational hardware is needed to compute \(x_1 + 2^{n}H\) in (21). The desired result is obtained by concatenating \(x_1\) with \(H\). Details of area and delay, of each part of the converter are shown in Table 1.

3.2 Converter for the Moduli Set \([2^{2n}, 2^n - 1, 2^{n-1} - 1]\)

By using a similar derivation like before, the residue to binary converter for the moduli set \([2^{2n}, 2^n - 1, 2^{n-1} - 1]\) is obtained as follow.

**Theorem 3:** The moduli set \([2^{2n}, 2^n - 1, 2^{n-1} - 1]\) consists of pairwise relatively prime moduli.

**Proof:** Similar to the proof of theorem 1, it can be seen that \(2^{2n}\) is relatively prime to the numbers \(2^n - 1\) and \(2^{n-1} - 1\). Also, since the moduli \(2^n - 1\) and \(2^{n-1} - 1\) previously used together in the RNS moduli set \([2^{n-1} - 1, 2^n - 1, 2^n]\) [16], they are relatively prime.

**Lemma 4:** The multiplicative inverse of \(2^{2n}\) modulo \(2^{n-1} - 1\) is \(k_4 = 2^{n-3}\).

**Proof:** It is obvious that,

\[
\left|k_4 \times 2^{2n}\right|_{2^{n-1}-1} = \left|2^{n-3} \times 2^{2n}\right|_{2^{n-1}-1} = \left|2^{3n-3}\right|_{2^{n-1}-1} = \left|2^{n-1} \times 2^{n-1} \times 2^{n-1}\right|_{2^{n-1}-1} = 1 \times 1 \times 1 = 1
\]

**Lemma 5:** The multiplicative inverse of \((2^n - 1)\) modulo \((2^{n-1} - 1)\) is \(k_5 = 1\).

**Proof:** First, it can be seen that,

\[
\left|2^n\right|_{2^{n-1}-1} = \left|2 \times 2^{n-1}\right|_{2^{n-1}-1} = 2
\]

Now, we have

\[
\left|k_5 \times (2^n - 1)\right|_{2^{n-1}-1} = \left|1 \times (2^n - 1)\right|_{2^{n-1}-1} = \left|1 \times (2 - 1)\right|_{2^{n-1}-1} = 1
\]

**Theorem 4:** In the RNS based on the moduli set \(\{P_1, P_2, P_3\} = \{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\), the number \(X\) can be

![Image](image-url)
calculated from its corresponding residues \((x_1, x_2, x_3)\) by

\[
X = x_1 + 2^{2n}Z + 2^{2n}(2^n - 1)T
\]  
(44)

where

\[
Z = \left| x_2 - x_1 \right|_{2^{n-1}}
\]  
(45)

\[
T = \left| (x_3 - x_1)2^{n-3} - Z \right|_{2^{n-1}}
\]  
(46)

**Proof:** By substituting \(P_1 = 2^{2n}, P_2 = 2^{n} - 1, P_3 = 2^{n-1} - 1\) and the values of multiplicative inverses from Lemmas 1, 4 and 5 into (6)–(9), Eqs. (44)–(46) are obtained.

Theorem 4 can be simplified by employing Properties 1 and 2, for reducing hardware complexity. The steps for simplifying (45) are same with those of Theorem 2, and hence we only note mathematical equations without any description.

\[
X = x_1 + 2^{2n}H
\]  
(47)

\[
H = Z + (2^n - 1)T
\]  
(48)

\[
Z = \left| w_1 + w_{21} + w_{22} \right|_{2^{n-1}}
\]  
(49)

where

\[
w_1 = \left| x_2 \right|_{2^{n-1}} = \left| x_{2,n-1} \cdots x_{2,1}x_{2,0} \right|_{2^{n-1}} = x_{2,n-1} \cdots x_{2,1}x_{2,0}
\]  
(50)

\[
w_{21} = \left| -x_{1,n-1} \cdots x_{1,1}x_{1,0} \right|_{2^{n-1}} = \bar{x}_{1,n-1} \cdots \bar{x}_{1,1}\bar{x}_{1,0}
\]  
(51)

\[
w_{22} = 2^n \left| x_{1,2n-1} \cdots x_{1,1}x_{1,0} \right|_{2^{n-1}}
\]  
(52)

Now, (46) can be simplified as follows

\[
T = \left| 2^{n-3}x_3 - 2^{n-3}x_1 - Z \right|_{2^{n-1}}
\]  
(53)

By dividing (53) in separate parts, we have

\[
w_3 = \left| 2^{n-3}x_3 \right|_{2^{n-1}} = \left| 2^{n-3}x_{3,n-2} \cdots x_{3,1}x_{3,0} \right|_{2^{n-1}} = x_{3,n-2} \cdots x_{3,1}x_{3,0}
\]  
(54)

\[
w_4 = \left| -2^{n-3}x_1 \right|_{2^{n-1}} = \left| -2^{n-3}x_{1,2n-1} \cdots x_{1,1}x_{1,0} \right|_{2^{n-1}} = -2^{n-3}x_{1,2n-1} - x_{1,1}x_{1,0}
\]  
(55)

The above equation can be parsing as follow

\[
w_{41} = \left| -2^{n-3}2^{2n-2}(0 \cdots 0 x_{1,2n-1}x_{1,2n-2}) \right|_{2^{n-1}}
\]  
(56)

\[
w_{42} = \left| -2^{n-3} \times 2^n(x_{1,2n-3} \cdots x_{1,1}x_{1,0}) \right|_{2^{n-1}} = -2^{n-3} \times 2^n(x_{1,2n-3} \cdots x_{1,1}x_{1,0})
\]  
(57)

\[
w_{43} = \left| -2^{n-3} \times 2^n(x_{1,n-2}x_{1,1}x_{1,0}) \right|_{2^{n-1}} = -2^{n-3}x_{1,n-2}x_{1,1}x_{1,0}
\]  
(58)

For the last term of (53), we have

\[
w_5 = \left| Z \right|_{2^{n-1}} = \left| (Z_{n-1} \cdots Z_1Z_0) \right|_{2^{n-1}} = (Z_{n-1} \cdots Z_1Z_0)
\]  
(59)

By dividing (59), \(w_{51}\) and \(w_{52}\) are obtained as follow

\[
w_{51} = \left| -2^{n-1}(0 \cdots 0 0 Z_{n-1}) \right|_{2^{n-1}} = 1 \cdots 1 \overline{Z}_{n-1}
\]  
(60)

\[
w_{52} = \left| (Z_{n-2} \cdots Z_1Z_0) \right|_{2^{n-1}} = \overline{Z}_{n-2} \cdots \overline{Z}_1Z_0\n\]  
(61)

Since both least significant \((n - 3)\) bits of \(w_{41}\) in (56) and most significant \((n - 2)\) bits of \(w_{51}\) in (60) are 1’s, we can use the following vectors instead of \(w_{41}\) and \(w_{51}\).

\[
w_{41} = \bar{x}_{1,2n-1}\bar{x}_{1,2n-2} \cdots \bar{1} \cdots \bar{1} \overline{Z}_{n-1}
\]  
(62)

\[
w_{51} = \left| 1 \cdots 1 \overline{1} \right|_{2^{n-1}} = 2^{n-1} - 1 = 0
\]  
(63)

We know that,

\[
w_{51} = \left| Z_{n-1} \right|_{2^{n-1}} = 2^{n-1} - 1 = 0
\]  
(64)

Therefore, (53) can be calculated by

\[
T = \left| w_3 + w_{42} + w_{43} + w_{52} + w_{41} \right|_{2^{n-1}}
\]  
(65)

Finally, (48) can be computed by

\[
H = P - Q
\]  
(66)

where

\[
P = Z + 2^n\overline{T} = T_{n-2} \cdots T_1T_0(\overline{Z}_{n-1} \cdots \overline{Z}_1Z_0)
\]  
(67)
$$Q = T = \frac{0 \cdots 00}{T_{n-2} \cdots T_1 T_0}_n$$  \hspace{1cm} (68)

**Example 2:** Given the moduli set \(\{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\) where \(n = 4\). The RNS number \((73, 14, 5)\) can be converted into its equivalent weighted number \(X\) as follows:

For \(n = 4\) the moduli set is \(\{256, 15, 7\}\) and residues have binary representation as:

- \(x_1 = 73 = (01001001)_2\)
- \(x_2 = 14 = (1110)_2\)
- \(x_3 = 5 = (101)_2\)

Substituting values in (49)–(52), we have:

- \(w_1 = (1110)_2 = 14\)
- \(w_{21} = (1011)_2 = 11\)
- \(w_{22} = (0110)_2 = 6\)
- \(Z = [14 + 11 + 6]_{15} = 1 = (0001)_2\)

Then, From (54), (57), (58), (61), (62) and (65), we have:

- \(w_3 = (011)_2 = 3\)
- \(w_{42} = (101)_2 = 5\)
- \(w_{43} = (101)_2 = 5\)
- \(w_{52} = (110)_2 = 6\)
- \(w'_{41} = (101)_2 = 5\)
- \(T = [3 + 5 + 5 + 6 + 5]_7 = 3 = (011)_2\)

Finally, letting values in (66)–(68) and (47),

- \(P = (01100001)_2 = 49\)
- \(Q = (00001011)_2 = 3\)
- \(H = 49 - 3 = 46 = (01011110)_2\)
- \(X = 73 + 2^8 \times 46 = (010111001001001)_2 = 11849\)

To verify the result, we have:

- \(x_1 = [11849]_{256} = 73\)
- \(x_2 = [11849]_{15} = 14\)
- \(x_3 = [11849]_7 = 5\)

So, the weighted number 11849 in the RNS based on the moduli set \(\{256, 15, 7\}\) has representation as \((73, 14, 5)\).

Also, hardware realization of the residue to binary converter for the moduli set \(\{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\) is based on (49), (65), (66) and (47), and it is depicted in Fig. 2. It can be seen that, the residue to binary converter composed of a 3-operand modulo \((2^n - 1)\) adder, a 5-operand modulo \((2^{n-1} - 1)\) adder and a \((2n-1)\)-bit CPA. Since the proposed residue to binary converter for the moduli set \(\{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\) has a similar architecture as the residue to binary converter for the moduli set \(\{2^{2n}, 2^n - 1, 2^{n+1} - 1\}\), details about hardware implementation have been omitted. The characterization of different parts of the residue to binary converter for the moduli set \(\{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\) is presented in Table 2.

### 4. Performance Evaluation

The residue to binary converters which presented in Sect. 3 are designed for the new \(4n\)-bit DR moduli sets \(\{2^{2n}, 2^n - 1, 2^{n-1} - 1\}\).
Table 2  Characterization of each part of the proposed residue to binary converter for moduli set \( (2^n, 2^n - 1, 2^{n-1} - 1) \).

<table>
<thead>
<tr>
<th>Parts</th>
<th>FA</th>
<th>NOT</th>
<th>XOR AND pairs</th>
<th>XOR OR pairs</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU4</td>
<td>–</td>
<td>2n</td>
<td>–</td>
<td>–</td>
<td>( t_{\text{NOT}} )</td>
</tr>
<tr>
<td>CSA4</td>
<td>( n )</td>
<td>–</td>
<td>–</td>
<td>( t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CPA4</td>
<td>( n )</td>
<td>–</td>
<td>–</td>
<td>( t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>OPU5</td>
<td>–</td>
<td>3n</td>
<td>–</td>
<td>( t_{\text{NOT}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CSA5</td>
<td>( n-1 )</td>
<td>–</td>
<td>–</td>
<td>( t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CSA6</td>
<td>( n-1 )</td>
<td>–</td>
<td>–</td>
<td>( t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CSA7</td>
<td>3</td>
<td>–</td>
<td>( n-4 )</td>
<td>( t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CPA5</td>
<td>( n-1 )</td>
<td>–</td>
<td>–</td>
<td>( (2n-2)t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>OPU6</td>
<td>( n-1 )</td>
<td>–</td>
<td>( n )</td>
<td>( t_{\text{NOT}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>CPA6</td>
<td>( n-1 )</td>
<td>–</td>
<td>( n )</td>
<td>( (2n-1)t_{\text{EAC}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
<tr>
<td>Total</td>
<td>( 6n-1 )</td>
<td>( 6n-1 )</td>
<td>( 2n-4 )</td>
<td>( (6n+1)t_{\text{EAC}} + 3t_{\text{NOT}} )</td>
<td>( 2t_{\text{EAC}} )</td>
</tr>
</tbody>
</table>

\( 2^{n+1} - 1 \) and \( 2^{2n}, 2^n - 1, 2^n - 1 - 1 \). Hence, to verify the performance of these residue to binary converters, they have to be compared with the other residue to binary converters for 4n-bit DR moduli sets. Such moduli sets are \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1) \), \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1) \), \( (2^n - 3, 2^n + 1, 2^n - 1, 2^n + 3) \), \( (2^n - 1, 2^n, 2^{2n+1} - 1) \) and \( (2^n - 1, 2^n, 2^{2n} + 1, 2^{3n} + 1) \). These moduli sets belong to the category of 4n-bit DR moduli sets, and each one has its own properties. The moduli sets \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1) \) and \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1) \) were introduced in [19] and [21], respectively. Recently, new designs of the residue to binary converters for these moduli sets based on MRC have been proposed by Mohan et al. [20]. As shown in [20], the residue to binary converters of [20] have better performance than those of [19] and [21]. Therefore, we only compare our residue to binary converters with the residue to binary converters of [20]. Also, a high-speed converter for the moduli set \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1) \) is presented in [22]. Sheu et al. [23] proposed the balanced moduli set \( (2^n - 3, 2^n + 1, 2^n - 1, 2^n + 3) \), but because of using moduli of the forms \( 2^k \pm 3 \), their residue to binary converter needs three ROMs, and this resulted in increasing the cost and the delay of the converter. Recently, Molahosseini et al. [24] proposed the modulus \((2^k + 1)-1\)-free moduli set \( (2^n - 1, 2^n, 2^{2n+1} - 1) \) for having fast residue arithmetic unit. The converter presented in [24] derived by using a two-level implementation of MRC algorithm, and this leads to increasing the latency.

Wang et al. [25] designed a residue to binary converter architecture for the moduli set \( (2^n - 1, 2^n + 1, 2^{2n} + 1) \). They used a modified CRT formulation to derive an adder-based residue to binary converter for this moduli set. The general structure of the residue to binary converter of [25] consists of two 2n-bit CSAs with EAC, a 2n-bit modulo \((2^{2n} - 1)\) adder and a 4n-bit binary adder. They used the method of [32] for performing the modulo \((2^{2n} - 1)\) addition, and by using different methods for implementing the 4n-bit binary adder, two versions of the residue to binary converter are presented in [25]. The residue to binary converters of [20]–[24] as well as the proposed designs, are using CPAs with EAC and regular CPAs for implementing modulo of the forms \((2^k - 1)\) adders and vector binary adders, respectively. Therefore, for a fair comparison, we consider a 2n-bit CPA with EAC, and a 4n-bit regular CPA for implementation of the modulo \((2^{2n} - 1)\) adder and 4n-bit binary adder of [25], respectively. In this case, the delay of the residue to binary converter of [25] is \((1 + 1 + 4n + 4n)t_{FA} = (8n + 2)t_{FA} \), where \( t_{FA} \) denotes the delay of an FA. Furthermore, the proposed residue to binary converters can be compared with the converter of the 5n-bit DR moduli set \( (2^n, 2^{2n} - 1, 2^{2n} + 1) \) [26], because similar to the proposed moduli sets, the moduli set \( (2^n, 2^{2n} - 1, 2^{2n} + 1) \) is an unconventional three-moduli set which offers larger DR than 3n-bit. Similar to the other converters, we consider a 4n-bit CPA with EAC for the implementation of the modulo \((2^{2n} - 1)\) adder of the residue to binary converter of [26]. Table 3 compares the hardware requirements and conversion delays of the residue to binary converters. It can be seen that, the proposed residue to binary converters have less conversion delay than all other converters. Furthermore, the hardware requirements of the proposed converters are less than other converters except the residue to binary converter of the moduli set \( (2^n, 2^{2n} - 1, 2^{2n} + 1) \). The residue to binary converter of the moduli set \( (2^n, 2^{2n} - 1, 2^{2n} + 1) \) [26] has the lowest hardware cost. Also, comparison of the proposed residue to binary converters with together showed that, the converter for the moduli set \( (2^n, 2^n - 1, 2^{n+1} - 1) \) is faster, while the residue to binary converter for the moduli set \( (2^{2n}, 2^n - 1, 2^{n+1} - 1) \) has lower hardware cost.

Until now, it is shown that the proposed residue to binary converters have better performance than the other residue to binary converters for 4n-bit DR moduli sets. But other important issues in the design of RNS systems are the speed of the internal RNS arithmetic processing and the binary to residue conversion. Here, we used the method of [26] for comparing the time-performance of the different RNS moduli sets. It is the magnitude of the largest moduli that dictates the speed of arithmetic operations; however, speed and cost also depend on the moduli chosen [2]. Thus, the modulo \( 2^n \) determines the overall speed of the RNS systems based on the proposed moduli sets \( (2^n, 2^n - 1, 2^{n+1} - 1) \) and \( (2^n, 2^{n} - 1, 2^{n-1} - 1) \). Also, the overall speed of arithmetic unit of RNS systems based on the moduli sets \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1) \), \( (2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1) \), \( (2^n - 3, 2^n - 1, 2^n + 1, 2^n + 3) \) and \( (2^n - 1, 2^n, 2^{2n+1} - 1) \) are restricted to the time performance of the moduli \( 2^{n+1} + 1, 2^n + 1, 2^n + 3 \) and \( 2^{2n+1} + 1 \), respectively. In addition to these, in the moduli sets \( (2^n - 1, 2^n + 1, 2^{2n} + 1) \) and \( (2^n, 2^n - 1, 2^{2n} + 1) \), the critical modulus is \( 2^{2n} + 1 \). Next, we consider the unit gate delays of parallel-prefix modular adders of [33],[34] and [35] for estimating the delay of addition in moduli of the forms \( 2^k + 1, 2^n + 1 \) and \( 2^n + 3 \), respectively. It should be noted that, there is no modular adder which specially designated for modulus of the forms \( 2^k + 3 \). Hence, we must use the generic modular adders for performing modulo \( 2^n + 3 \) addition. Also, modulo \( 2^k \) adder can perform addition with approximately the same speed as the modulo \( 2^k - 1 \) adder.
be simply implemented. The modulo 2

They are faster than the four-moduli sets. It can be seen that, while the proposed moduli sets have three

ger binary number into its equivalent RNS representation. Also, reduction circuits for

mented with multi-operand modular adders (MOMAs) [31].

Table 4 Comparison the speed of the different moduli sets.

<table>
<thead>
<tr>
<th>Moduli set</th>
<th>Critical modulo</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>({2^n, 2^n, 2^n+1, 2^n+1})</td>
<td>(2^n+1)</td>
<td>(2\log(n)\cdot 6)</td>
</tr>
<tr>
<td>({2^n, 2^n+1, 2^n+1})</td>
<td>(2^n+1)</td>
<td>(2\log(n+1)\cdot 6)</td>
</tr>
<tr>
<td>({2^n-3, 2^n-1, 2^n-1, 2^n+3})</td>
<td>(2^n+3)</td>
<td>(2\log(n+1)\cdot 7)</td>
</tr>
<tr>
<td>({2^n-1, 2^n, 2^n+1})</td>
<td>(2^n-1)</td>
<td>(2\log(n+1)\cdot 5)</td>
</tr>
<tr>
<td>({2^n, 2^n+1, 2^n+1})</td>
<td>(2^n+1)</td>
<td>(2\log(n)\cdot 8)</td>
</tr>
<tr>
<td>({2^n, 2^n, 2^n-1, 2^n+1})</td>
<td>(2^n)</td>
<td>(2\log(n)\cdot 5)</td>
</tr>
<tr>
<td>({2^n, 2^n, 2^n-1})</td>
<td>(2^n)</td>
<td>(2\log(n)\cdot 6)</td>
</tr>
</tbody>
</table>

Hence, each of the proposed moduli sets as well as the moduli sets \(\{2^n-1, 2^n, 2^n+1\}\) and \(\{2^n, 2^n-1, 2^n+1\}\), requires two MOMAs. But, the MOMAs for the moduli \(2^n-1\) and \(2^n+1\) have longer delay and higher cost than the MOMAs for the moduli \(2^n-1\) and \(2^n+1\) which are needed for our moduli sets. Furthermore, for the moduli sets \(\{2^n-1, 2^n, 2^n+1\}\) and \(\{2^n-1, 2^n, 2^n+1, 2^n+1\}\), we need three MOMAs. Particularly, binary to residue converter for the moduli set \(\{2^n-3, 2^n-1, 2^n+3\}\) requires four MOMAs. Therefore, it is clear that the proposed moduli sets can result in more efficient binary to residue converters than the other moduli sets.

5. Conclusion

In this paper, efficient residue to binary converters for the new RNS moduli sets \(\{2^n, 2^n-1, 2^n+1\}\) and \(\{2^n, 2^n-1, 2^n+1\}\) are presented. The converters for these moduli sets are achieved by an adder-based implementation of the MRC algorithm, with better performance compared to the other residue to binary converters for 4n-bit DR moduli sets. Also, it is shown that, with the new proposed moduli sets, the binary to residue converter as well as the internal RNS arithmetic circuits can be implemented efficiently, resulting in high-performance RNS systems.

References


[20] P.V.A. Mohan and A.B. Premkumar, “RNS-to-binary converters for two four-moduli set \( \{2^n-1, 2^n, 2^n+1, 2^n+1-1\} \) and \( \{2^n-1, 2^n, 2^n+1, 2^n+1+1\} \),” IEEE Trans. Circuits Syst. I, vol.54, no.6, pp.1245–1254, 2007.


Keivan Navi received his B.Sc and M.Sc in Computer Hardware engineering from Shahid Beheshti University in 1987 and Sharif University of Technology in 1990, respectively. He also received his Ph.D. of Computer Architecture from Paris XI University in 1995. He is currently associate professor in faculty of electrical and computer engineering of Shahid Beheshti University. His research interests include VLSI design, Single Electron Transistors (SET), Carbon Nano Tube, Computer Arithmetic, Interconnection Network and Quantum Computing. He has published over 30 ISI and research journal papers and over 70 IEEE, international and national conference paper.

Mohammad Eshghi (BS’78, Ms’89, Phd’94) is born in Shahroud, Iran in 1954 and got his BS in electrical engineering from Sharif University in 1978. He got his MS degree in EE from Ohio University, Athens, Ohio, and his Ph.D in EE from Ohio State University, Columbus, Ohio, USA, in 1989 and 1994, respectively. He is now with the Electrical and Computer Engineering Faculty at Shahid Beheshti University, Tehran Iran. His research interests including Digital Signal processing, Digital circuit design, implementation and test on Field Programmable Gate Array (FPGA), and quantum computing.