Efficient RNS Scalers for the Extended Three-Moduli Set $(2^n - 1, 2^{n+p}, 2^n + 1)$

Ahmad Hiasat

Abstract—The scaling problem in Residue Number System was addressed in many publications. A significant focus was given to scale the three moduli set $(2^n - 1, 2^n, 2^n + 1)$ by 2^n, where n is a positive integer. This paper presents a scaling structure design for the moduli sets $(2^n - 1, 2^{n+p}, 2^n + 1)$ scaled by 2^n, where $0 \leq p \leq n$. The new design has the delay of a full-adder and a modular adder. The proposed structure is smaller, faster and more power-efficient than the most recent published work for the same moduli set and the same scaling factor. VLSI synthesis results showed an average area reduction of $(9.8\text{-}27.9)$ percent, an average time reduction of $(13.9\text{-}20.8)$ percent and an average power reduction of $(15.9\text{-}22.3)$ percent. The paper also presents a time-efficient scaling structure for the extended three moduli set $(2^n - 1, 2^{n+p}, 2^n + 1)$ scaled by 2^{n+p}, where $1 \leq p \leq n$. The delay of this structure is one half-adder more than the delay of the aforementioned structure.

Index Terms—Residue number system, chinese remainder theorem, scalers, converters, computer arithmetic

1 Introduction

In all computationally-demanding applications that depend on Addition, Subtraction and Multiplication (ASAM), the efficiency of such applications decreases with increasing the word-length [1]. Dealing with sub-word lengths enhances the parallelism of computations and increases considerably the performance of these applications [1], [2].

Residue Number System (RNS) is a non-weighted representation that expresses huge word-length binary-numbers into smaller word-length digits [2], [3], [4]. Unlike weighted number systems (e.g., binary and decimal systems), the non-weighted representation of RNS allows parallel processing on all residue digits with no carry propagation from one residue digit to another. This differentiating feature enables RNS to be used in computationally-demanding ASAM-based applications [2], [3], [4]. The positive implications of using RNS were clearly demonstrated in different applications including signal processing, cryptography, communications systems and other applications [3], [4], [5], [6], [7], [8], [9], [10], [11], where [7], [8], [9] are recent applications of the moduli set considered in this paper.

Dealing with RNS processors requires designing forward decoders (binary to residue converters) and reverse decoders (residue to binary converters) [3], [4]. Many researchers presented efficient decoding structures. These decoders are essential components in interfacing a RNS processor to the binary world [3], [4]. Similarly, designing efficient ASAM-components for RNS processors was addressed and significant results were reported [3], [4]. When ASAM-computations exceed an allowable range within any processor, an overflow takes place which leads to erroneous results [1], [2]. Like binary-based processors, RNS-based processors need to deal efficiently with computational overflows [2], [3], [4]. These overflow occurrences are treated using scalers. In RNS, scaling implies dividing a residue representation by a constant. While forward and reverse decoders are needed once per each set of data upon entering or exiting a RNS-based processor, scaling is needed frequently during the course of computations to which each data set is subject to. Therefore, designing an efficient scaler is an essential part of designing a RNS-based processor.

Many papers dealt with RNS scaling problem. In this section, it is not meant to list all work related to RNS scaling, but to identify a few of these. The work in [12] proposed two scaling algorithms, where each has its own error margin. Another magnitude scaling technique was presented in [13]. The technique was based on the use of a redundant channel. The approach in [14] introduced a precise scaling for RNS using look-up tables. Another technique was presented in [15], where the design utilized smaller ROMs and multi-operand adders and resulted in a reduced scaling error. The two look-up-cycle scaling scheme proposed in [16] allowed a reduction in ROM sizes. A fully adder-based approach for scaling residue numbers was presented in [17]. An area-efficient scaler for the moduli set $(2^n - 1, 2^n, 2^n + 1)$ was presented in [18]. A new approach for a precise ROM-based scaling was introduced in [19], where this design compares favorably with other ROM-based designs. Another efficient scaling scheme for signed integers was presented in [20]. The relatively recent memoryless scaling algorithm presented in [21] for the three-moduli set $(2^n - 1, 2^n, 2^n + 1)$ has no scaling error and has a better area and time performance than many previous ones. However, the work of [22] presented an algorithm for scaling the three-moduli set $(2^n - 1, 2^n, 2^n + 1)$ by a programmable power-of-two factor. A signed integer RNS scaler for the three-moduli set $(2^n - 1, 2^n, 2^n + 1)$ was presented in [23]. The most recent work in [24] presented a scaler design for the extended three moduli set $(2^n - 1, 2^{n+p}, 2^n + 1)$, where this set was originally proposed in [25]. The three-moduli set scaler was used as a building block to propose a four-moduli set scaler [24].

This paper adopts the following notations:

- The extended three moduli set of consideration is $(m_1, m_2, m_3) = (2^n - 1, 2^{n+p}, 2^n + 1)$, where n and p are positive integers such that $0 \leq p \leq n$.
- The dynamic range is defined by $M = \prod_{i=1}^{3} m_i$.
- The residue representation of an integer X, where $X \in [0, M)$, is (R_1, R_2, R_3).
- $R_i = \langle X \rangle_{m_i}$, the minimum non-negative remainder obtained when X is divided by the modulus m_i.
- The binary representations of the residue digits (R_1, R_2, R_3) are given by:
  - $R_1 = \sum_{i=0}^{n-1} r_{1i} 2^i = r_{1(n-1)} \cdots\cdots r_{11} r_{10}$
  - $R_2 = \sum_{i=0}^{n+p-1} r_{2i} 2^i = r_{2(n+p-1)} \cdots\cdots r_{21} r_{20}$
  - $R_3 = \sum_{i=0}^{n} r_{3i} 2^i = r_{3n} \cdots\cdots r_{31} r_{30}$
- $\hat{m}_i = \frac{M}{m_i}$, whereas, $\langle \hat{m}_i^{-1} \rangle_{m_i}$ is the multiplicative inverse of $\hat{m}_i$ (i.e., $\left\langle \hat{m}_i \langle \hat{m}_i^{-1} \rangle_{m_i} \right\rangle_{m_i} = 1$)
- $\lceil . \rceil$ is the ceiling value of $(.)$ (i.e., the smallest integer equal to or greater than $(.)$. However, $\lfloor . \rfloor$ is the floor value of $(.)$ (i.e., the largest integer equal to or less than $(.)$.
- Logical operators: AND, OR, NOT, XOR and XNOR are defined by the notations, $\wedge$, $\vee$, $\overline{(.)}$, $\oplus$ and $\odot$, respectively.
- The most frequent basic arithmetic units used, are: Full-Adder (FA), Half-Adder (HA) and Half-Adder-Like (HAL) circuit. A HAL produces the sum and carry of adding two bits while assuming the third bit to be 1. A HAL consists of an OR and an XNOR and has the same complexity of a HA.
- For positive integers a and b such that, $a < 2^n - 1$ and $b < n$, modulo $(2^n - 1)$ properties imply that multiplying a by 2^b modulo $(2^n - 1)$ is equivalent to rotating the binary value of a to the left b bits. A modulo $(2^n - 1)$ of a negative number $-a$ is the one's complement of a.

• *The author is with the Computer Engineering Department, Princess Sumaya University for Technology, Amman 11941 Jordan. E-mail: a.hiasat@psut.edu.jo.*

Manuscript received 7 Aug. 2016; revised 19 Dec. 2016; accepted 9 Jan. 2017. Date of publication 15 Jan. 2017; date of current version 15 June 2017.
Recommended for acceptance by W. Liu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2652474

TABLE 1
Truth Table for ASFA1 Circuit Adding r_{21}, r_{3n} and $\overline{r}_{31}$

r_{21}	r_{3n}	$\overline{r}_{31}$	c_2	s_1
0	0	0	0	0
0	0	1	0	1
0	1	0	X	X
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	X	X
1	1	1	1	1

TABLE 2
Truth Table for ASFA2 Circuit Adding r_{20}, $\overline{r}_{3n}$ and $\overline{r}_{30}$

r_{20}	$\overline{r}_{3n}$	$\overline{r}_{30}$	c_1	s_0
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	X	X
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- A modulo (2^n+1) addition property: adding two non-negative integers A and B is expressed as [26]: $\langle A+B+1\rangle_{(2^n+1)} = \langle\langle A+B\rangle_{2^n} + \overline{c}_{out}\rangle_{(2^n+1)}$, where $\overline{c}_{out}$ is the complement of the output carry (i.e., Complemented End-Around Carry (CEAC)).

The remaining of the paper is organized as follows: Section 2 proposes a scaling structure for the three-moduli set $(2^n-1, 2^n, 2^n+1)$ scaled by 2^n. Section 3 presents hardware structures for scaling the extended set $(2^n-1, 2^{n+p}, 2^n+1)$ by 2^{n+p} and by 2^n. Section 4 compares the performance of the proposed structures with other functionally-similar published ones.

2 Scaling the Three Moduli Set $(2^n-1, 2^n, 2^n+1)$, the Scaling Factor $= 2^n$

For a three moduli set, the Chinese Remainder Theorem (CRT) has the form [1],

$$X = \hat{m}_1\langle\hat{m}_1^{-1}\rangle_{m_1}R_1 + \hat{m}_2\langle\hat{m}_2^{-1}\rangle_{m_2}R_2 + \hat{m}_3\langle\hat{m}_3^{-1}\rangle_{m_3}R_3 - MJ, \quad (1)$$

where J is an integer variable.

The multiplicative inverses of the moduli set $(2^n-1, 2^n, 2^n+1)$, are [27], $\langle\hat{m}_1^{-1}\rangle_{m_1} = 2^{n-1}$, $\langle\hat{m}_2^{-1}\rangle_{m_2} = \langle -1\rangle_{m_2}$, and $\langle\hat{m}_3^{-1}\rangle_{m_3} = \langle -2^{n-1}\rangle_{m_3}$, whereas $\hat{m}_1 = 2^n(2^n+1)$, $\hat{m}_2 = (2^{2n}-1)$ and $\hat{m}_3 = 2^n(2^n-1)$. Upon substituting these values in (1), dividing by 2^n, considering the integer value and applying modulus $(2^{2n}-1)$:

$$\left\lfloor\frac{X}{2^n}\right\rfloor = \Big\langle 2^{n-1}\Big(\langle(2^n+1)R_1\rangle_{(2^{2n}-1)} + \langle -2R_2\rangle_{(2^{2n}-1)} \langle -(2^n-1)R_3\rangle_{(2^{2n}-1)}\Big)\Big\rangle_{(2^{2n}-1)}. \quad (2)$$

For this moduli set, the residue digits of the scaled value $\lfloor\frac{X}{2^n}\rfloor$ are: $S_1 = \langle\lfloor\frac{X}{2^n}\rfloor\rangle_{(2^n-1)}$, $S_2 = \langle\lfloor\frac{X}{2^n}\rfloor\rangle_{2^n}$ and $S_3 = \langle\lfloor\frac{X}{2^n}\rfloor\rangle_{(2^n+1)}$. Using the same approach of [21] and [24] leads to:

$$S_1 = \langle R_1 - R_2\rangle_{(2^n-1)} \quad (3)$$

$$S_3 = \langle R_2 - R_3\rangle_{(2^n+1)}. \quad (4)$$

The value of S_1 in (3) is realized by a modulo (2^n-1) subtractor [21], [24]. However, to realize (4), R_3 needs further simplification. As it consists of $(n+1)$ bits, R_3 can be written as: $R_3 = 2^n r_{3n} + \hat{R}_3$, where the binary representation of $\hat{R}_3$ is given by $\hat{R}_3 = \overbrace{r_{3(n-1)}\cdots r_{31}r_{30}}^{n}$. Applying modulus (2^n+1) to the term $-R_3$ in (4) leads to:

$$\begin{aligned}\langle -R_3\rangle_{(2^2+1)} &= \langle -2^n r_{3n} - \hat{R}_3\rangle_{(2^2+1)}\\ &= \langle r_{3n} + (2^n+1) - \hat{R}_3\rangle_{(2^n+1)}\\ &= \langle r_{3n} + (2^n - 1 - \hat{R}_3) + 2\rangle_{(2^n+1)}\\ &= \left\langle \overline{\hat{R}}_3 + 2 + r_{3n}\right\rangle_{(2^n+1)}.\end{aligned} \quad (5)$$

Substituting (5) into (4):

$$S_3 = \left\langle R_2 + \overline{\hat{R}}_3 + (1 + r_{3n}) + 1\right\rangle_{(2^2+1)}. \quad (6)$$

Using modulo (2^n+1) addition property and observing that the term $(r_{3n}+1) = (2r_{3n} + \overline{r}_{3n})$, then (6) is written as:

$$S_3 = \left\langle \left\langle R_2 + \overline{\hat{R}}_3 + (2r_{3n} + \overline{r}_{3n})\right\rangle_{2^n} + \overline{c}_{out}\right\rangle_{(2^n+1)}. \quad (7)$$

Rewriting (7) using binary notation,

$$S_3 = \left\langle \begin{array}{l} \langle\ r_{2(n-1)}\cdots\cdots\ r_{22}\ r_{21}\ r_{20} \\ +\ \overline{r}_{3(n-1)}\cdots\cdots\ \overline{r}_{32}\ \overline{r}_{31}\ \overline{r}_{30} \\ +\ 0\cdots\cdots\cdots\cdots\ 0\quad r_{3n}\ \overline{r}_{3n}\rangle_{2^n} \\ +\qquad\qquad\qquad\qquad\qquad\quad \overline{c}_{out} \end{array}\right\rangle_{(2^n+1)}. \quad (8)$$

Computing S_3 given in (8) requires a CSA followed by a modulo (2^n+1) adder. The CSA adds the three vectors of (8). The output of the CSA consists of a sum vector: $S = s_{n-1}\cdots s_1 s_0$ and a carry vector: $C = c_{n-1}\cdots c_1\overline{c}_n$, where $\overline{c}_n$ is the CEAC. The Most Significant (MS) $(n-2)$ bits of (8) are computed using $(n-2)$ HAs. Adding r_{21}, $\overline{r}_{31}$ and r_{3n} of (8) to produce s_1 and c_2 needs an Application-Specific Full-Adder-1 (ASFA1) circuit. The truth table of the ASFA1 is shown in Table 1, where 'X' refers to a don't care condition (i.e., $r_{3n} = r_{31} = 1$, or alternatively, $r_{3n} = 1$ and $\overline{r}_{31} = 0$). Minimizing the outputs of the truth table given in Table 1 produces: $c_2 = r_{3n} \vee (r_{21} \wedge \overline{r}_{31})$ and $s_1 = (r_{21} \wedge r_{31}) \vee (r_{21} \wedge r_{3n}) \vee (\overline{r}_{21} \wedge \overline{r}_{31} \wedge \overline{r}_{3n})$. Similarly, adding r_{20}, $\overline{r}_{30}$ and $\overline{r}_{3n}$ to produce s_0 and c_1 requires an Application-Specific Full-Adder-2 (ASFA2) circuit. The truth table of the ASFA2 is shown in Table 2 leads to: $c_1 = r_{20} \vee (\overline{r}_{3n} \wedge \overline{r}_{30})$ and $s_0 = (\overline{r}_{20} \wedge r_{3n}) \vee (\overline{r}_{20} \wedge r_{30}) \vee (r_{20} \wedge \overline{r}_{3n} \wedge \overline{r}_{30})$. The ASFA1 and ASFA2 circuits are shown in Fig. 1, where each has an area of 6 gates and has a delay of 2 gates. The two circuits constitute the least two significant bits of the CSA in the S_3 channel.

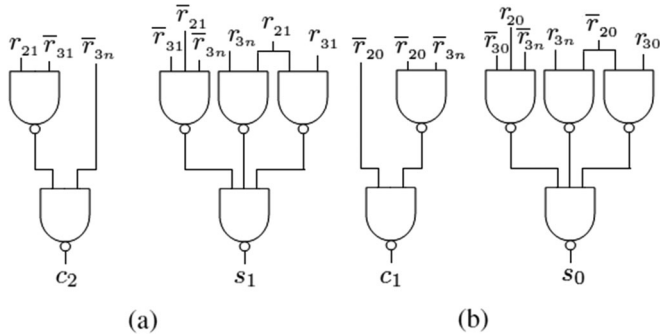

Fig. 1. The application-specific falf-adder (ASFA) circuits, (a) ASFA1 generating c_2 and s_1, (b) ASHF2 generating c_1 and s_0.

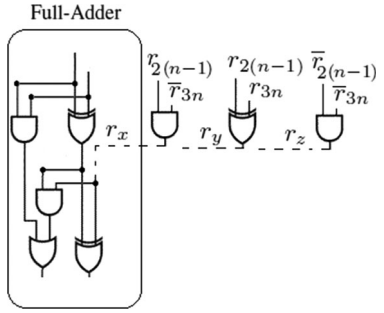

Fig. 2. Generation of the bits r_x, r_y and r_z of (19) does not add any delay.

S_2 is obtained by applying modulo 2^n to (2):

$$S_2 = \Big\langle \Big\langle 2^{n-1}\Big(\langle (2^n+1)R_1 \rangle_{(2^{2n}-1)} + \langle -2R_2 \rangle_{(2^{2n}-1)} \\ \langle -(2^n-1)R_3 \rangle_{(2^{2n}-1)} \Big) \Big\rangle_{(2^{2n}-1)} \Big\rangle_{2^n}. \tag{9}$$

The 2^{n-1} multiplier in (9) is implemented by $(n-1)$-bit left rotation. Using the modulo (2^n-1) properties, the terms on the RHS of (9) are simplified as follows, where $\star$ denotes a concatenation operator:

$$\begin{aligned}(2^n+1)R_1 &= R_1 \star R_1 \\ &= \overbrace{r_{1(n-1)} \cdots\cdots r_{10}}^{n} \overbrace{r_{1(n-1)} \cdots\cdots r_{10}}^{n}\end{aligned} \tag{10}$$

$$\begin{aligned}\langle -2R_2 \rangle_{(2^{2n}-1)} &= \overbrace{1 \cdots 1\, \overline{r}_{2(n-1)}}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}1}^{n} \\ &= \overbrace{0 \cdots 0\, \overline{r}_{2(n-1)}}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n} \\ &\quad + (2^{2n} - 2^{n+1} + 1) \\ &= \underline{R}_2 + (2^{2n} - 2^{n+1} + 1),\end{aligned} \tag{11}$$

where:

$$\underline{R}_2 = \overbrace{0 \cdots 0\, \overline{r}_{2(n-1)}}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n}. \tag{12}$$

However,

$$\begin{aligned}\langle (1-2^n)R_3 \rangle_{(2^{2n}-1)} &= (1-2^n)(2^n r_{3n} + \hat{R}_3) \\ &= (1-2^n)\hat{R}_3 + (1-2^n)2^n r_{3n} \\ &= \overline{\hat{R}}_3 \star \hat{R}_3 + (2^n-1) + (2^n-1)r_{3n} \\ &= \overline{\hat{R}}_3 \star \hat{R}_3 + (2^n-1)(1+r_{3n}).\end{aligned} \tag{13}$$

Substituting (10), (11) and (13) into (2) leads to:

$$\lfloor \tfrac{X}{2^n} \rfloor = \Big\langle 2^{n-1}\Big(\widetilde{R}_1 + \underline{R}_2 + \widetilde{R}_3 + K \Big) \Big\rangle_{(2^{2n}-1)}, \tag{14}$$

where, $\widetilde{R}_1 = R_1 \star R_1$, $\widetilde{R}_3 = \overline{\hat{R}}_3 \star \hat{R}_3$ and $K = \langle (2^{2n} - 2^{n+1} + 1) + (2^n - 1)(1 + r_{3n}) \rangle_{(2^{2n}-1)}$.

Simplifying K is performed as follows:

$$K = \begin{cases} 2^{2n} - 2^n, & r_{3n} = 0 \\ 0, & r_{3n} = 1 \end{cases}. \tag{15}$$

The binary form of (15) is:

$$K = \overbrace{\overline{r}_{3n} \cdots\cdots \overline{r}_{3n}}^{n} \overbrace{0 \cdots\cdots\cdots 0}^{n}. \tag{16}$$

The two terms $\underline{R}_2$ and K given in (12) and (16), respectively, can be added in the following manner,

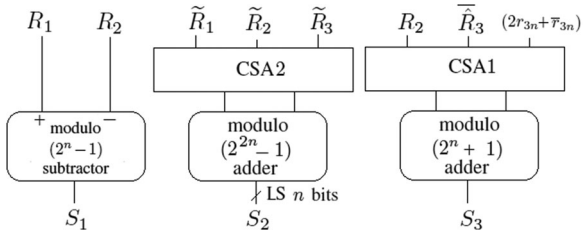

Fig. 3. Block diagram of the proposed scaler for the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by $2^n, 0 \le p \le n$.

$$\begin{aligned}\underline{R}_2 + K = &\overbrace{0 \cdots 0\, \overline{r}_{2(n-1)}}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n} \\ &+ \overbrace{\overline{r}_{3n} \cdots\cdots \overline{r}_{3n}}^{n} \overbrace{0 \cdots\cdots\cdots\cdots 0}^{n}.\end{aligned} \tag{17}$$

Adding the two binary forms on the RHS of (17) depends on the two bits $\overline{r}_{2(n-1)}$ and $\overline{r}_{3n}$. Applying modulo $(2^{2n}-1)$, the results of addition in (17) are given by:

$$\begin{cases} \overbrace{0 \cdots\cdots 0\, 0}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n}, & \text{if } \overline{r}_{3n}\overline{r}_{2(n-1)} = 00 \\ \overbrace{0 \cdots\cdots 0\, 1}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n}, & \text{if } \overline{r}_{3n}\overline{r}_{2(n-1)} = 01 \\ \overbrace{1 \cdots\cdots 1\, 1}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}0}^{n}, & \text{if } \overline{r}_{3n}\overline{r}_{2(n-1)} = 10 \\ \overbrace{0 \cdots\cdots 0\, 0}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20}1}^{n}, & \text{if } \overline{r}_{3n}\overline{r}_{2(n-1)} = 11 \\ \uparrow \quad \cdots \quad \uparrow\ \uparrow \qquad\qquad \uparrow & \\ \underbrace{r_x \cdots\cdots r_x}_{n} \underbrace{r_y \overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20} r_z}_{n} & = \underline{R}_2 + K \end{cases} \tag{18}$$

where $r_x = \overline{r}_{3n} \wedge r_{2(n-1)}$, $r_y = r_{3n} \oplus r_{2(n-1)}$ and $r_z = \overline{r}_{3n} \wedge \overline{r}_{2(n-1)}$. As shown in Fig. 2, the generation of these three bits does not add any delay to the FA to which they are connected to. Defining $\widetilde{R}_2 = \underline{R}_2 + K$, then the binary representation of $\widetilde{R}_2$ is given by:

$$\widetilde{R}_2 = \overbrace{r_x \cdots\cdots r_x r_y}^{n} \overbrace{\overline{r}_{2(n-2)} \cdots\cdots \overline{r}_{20} r_z}^{n}. \tag{19}$$

Substituting $\widetilde{R}_2 = \underline{R}_2 + K$ into (14) leads to,

$$\lfloor \tfrac{X}{2^n} \rfloor = \Big\langle 2^{n-1}\Big(\widetilde{R}_1 + \widetilde{R}_2 + \widetilde{R}_3 \Big) \Big\rangle_{(2^{2n}-1)}. \tag{20}$$

The implementation of (20) requires a $2n$-bit CSA followed by a modulo $(2^{2n}-1)$ adder. If the value of R_2 is concatenated to the output of this modulo adder, where R_2 forms the Least Significant (LS) part, the result is X (i.e., a residue to binary converter). The LS n bits of the modulo $(2^{2n}-1)$ adder constitute S_2. Hence, this modulo $(2^{2n}-1)$ adder can be stripped down keeping the components necessary to compute just the LS n bits [24].

The proposed hardware structure for the scaler under consideration is shown in Fig. 3. It consists of two CSAs (i.e., CSA1 and CSA2) and three modular adders. CSA1 consists of $(n-2)$ HAs and 12 gates while CSA2 consists of n FAs. The longest path delay is the delay of the S_2 channel which is delay of 1 FA and a modulo $(2^{2n}-1)$ adder.

Example 1. For the moduli set $(31, 32, 33)$ where $n = 5$, it is intended to scale the residue digits $(14, 20, 27)$ by 32.

In the computations next, the subscript 2 refers to a binary value. $R_1 = (01110)_2$, $R_1 = (10100)_2$ and $R_1 = (11011)_2$. Using (3), $S_1 = \langle 14 - 20 \rangle_{31} = 25$. Using (4), $S_3 = \langle 20 - 27 \rangle_{33} = 26$. Using (18), $r_x = \overline{r}_{3n} \wedge r_{2(n-1)} = \overline{0} \wedge 1 = 1$, $r_y = r_{3n} \oplus r_{2(n-1)} = 0 \oplus 1 = 1$, $r_z = \overline{r}_{3n} \wedge$

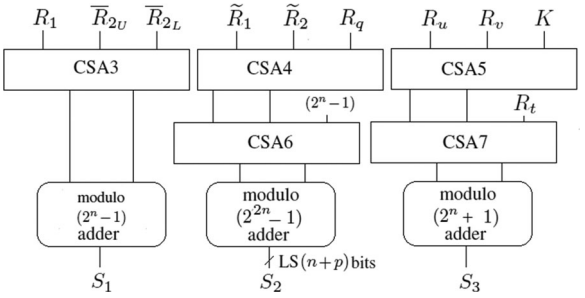

Fig. 4. Block diagram of the proposed scaler for the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^{n+p}, $0 < p \leq n$.

$\overline{r}_{2(n-1)} = \overline{0} \wedge \overline{1} = 0$. Thus, $\widetilde{R}_1 = (01110\ 01110)_2 = 462$ and $\widetilde{R}_2 = (11111\ 10110)_2 = 1014$. Using (14), $\widetilde{R}_3 = (00100\ 11011)_2 = 155$. Using (20), $\lfloor \frac{X}{2^5} \rfloor = \langle 2^4(462+1014+155) \rangle_{(2^{2n}-1)} = 521$. $S_2 = \langle 521 \rangle_{32} = 9$. In fact, $X = 16{,}692$, where, $\lfloor \frac{16{,}692}{32} \rfloor = 521$.

Repeating the same example but with changing only R_3 to be 32. This causes $r_{3n} = 1$, $r_x = r_y = r_z = 0$. $S_1 = \langle 14 - 20 \rangle_{31} = 25$ (no change). $S_3 = \langle 20 - 32 \rangle_{33} = 21$. Similarly, $\widetilde{R}_1 = 462$ (no change) and $\widetilde{R}_2 = (00000\ 10110)_2 = 22$. Using (14), $\widetilde{R}_3 = (11111\ 00000)_2 = 992$. Using (20), $\lfloor \frac{X}{2^5} \rfloor = \langle 2^4(22+462+992) \rangle_{(2^{2n}-1)} = 87$. $S_2 = \langle 87 \rangle_{32} = 23$. In fact, $X = 2{,}804$, where, $\lfloor \frac{2{,}804}{32} \rfloor = 87$.

3 Scaling the Three Moduli Set $(2^n-1, 2^{n+p}, 2^n+1)$

For the extended moduli set $(2^n-1, 2^{n+p}, 2^n+1)$, the multiplicative inverses are [28]: $\langle \hat{m}_1^{-1} \rangle_{m_1} = \langle 2^{n-p-1} \rangle_{m_1}$, $\langle \hat{m}_2^{-1} \rangle_{m_2} = \langle -1 \rangle_{m_2}$ and $\langle \hat{m}_3^{-1} \rangle_{m_3} = \langle -2^{n-p-1} \rangle_{m_3}$, where $1 \leq p < n$. However, $\hat{m}_1 = 2^{n+p}(2^n+1)$, $\hat{m}_2 = (2^{2n}-1)$ and $\hat{m}_3 = 2^{n+p}(2^n-1)$. Substituting these values into (1) and simplifying the resulting expression lead to:

$$\begin{aligned} X = {} & 2^{2n-1}(2^n+1)R_1 - (2^{2n}-1)R_2 \\ & - 2^{2n-1}(2^n-1)R_3 - MJ. \end{aligned} \tag{21}$$

3.1 The Scaling Factor $= 2^{n+p}$

Upon dividing (21) by 2^{n+p}, considering the integer value and applying modulus $(2^{2n}-1)$:

$$\begin{aligned} \left\lfloor \frac{X}{2^{n+p}} \right\rfloor = \Big\langle 2^{n-p-1} \Big(& \langle (2^n+1)R_1 \rangle_{(2^{2n}-1)} + \langle -2R_2 \rangle_{(2^{2n}-1)} \\ & + \langle -(2^n-1)R_3 \rangle_{(2^{2n}-1)} \Big) \Big\rangle_{(2^{2n}-1)}. \end{aligned} \tag{22}$$

Applying modulo (2^n-1) to (22) to obtain S_1:

$$S_1 = \Big\langle 2^{n-p} \langle R_1 - R_2 \rangle_{(2^n-1)} \Big\rangle_{(2^n-1)}. \tag{23}$$

R_2 in (23) has $(n+p)$ bits. These additional p bits form a new partial variable that needs to be taken into consideration. Furthermore, the value of $(R_1 - R_2)$ in (23) needs to be rotated to the left by $(n-p)$ bits.

Splitting the binary value of R_2 into upper part $R_{2_U} = \overbrace{0\ \cdots\cdots\ 0}^{n-p}\overbrace{r_{2(n+p-1)}\ \cdots\cdots\ r_{2(n+1)}\ r_{2n}}^{p}$ and lower part $R_{2_L} = \overbrace{r_{2(n-1)}\ \cdots\cdots\cdots\ r_{22}\ r_{21}\ r_{20}}^{n}$ then:

$$R_2 = R_{2_U} 2^n + R_{2_L}. \tag{24}$$

Substituting (24) into (23):

$$S_1 = \big\langle 2^{n-p}(R_1 - R_{2_U} - R_{2_L}) \big\rangle_{(2^n-1)}. \tag{25}$$

Using the (2^n-1) properties, (25) is written as:

$$S_1 = \Big\langle 2^{n-p} \big\langle R_1 + \overline{R}_{2_U} + \overline{R}_{2_L} \big\rangle_{(2^n-1)} \Big\rangle_{(2^n-1)}. \tag{26}$$

The value of S_1 given in (26) is computed using a CSA (i.e., CSA3 in the S_1 channel of Fig. 4) followed by a modulo (2^n-1) adder.

Similarly, applying modulo (2^n+1) to (22) produces S_3:

$$S_3 = \langle 2^{n-p} R_3 - 2^{n-p} R_2 \rangle_{(2^n+1)}. \tag{27}$$

In (27), the term $-2^{n-p}R_2$ has the binary notation:

$$\begin{aligned} & \langle -2^{n-p} R_2 \rangle_{(2^n+1)} \\ & = \left\langle - \overbrace{r_{2(n+p-1)} \cdots\cdots r_{20}}^{n+p} \star \overbrace{0 \cdots\cdots 0}^{n-p} \right\rangle_{(2^n+1)} \\ & = \left\langle - \overbrace{r_{2(n+p-1)} \cdots r_{2p}}^{n} \star \overbrace{r_{2(p-1)} \cdots r_{20} 0 \cdots 0}^{n} \right\rangle_{(2^n+1)} \\ & = \left\langle - \overbrace{r_{2(n+p-1)} \cdots r_{2p}}^{n} 2^n - \overbrace{r_{2(p-1)} \cdots r_{20} 0 \cdots 0}^{n} \right\rangle_{(2^n+1)} \\ & = \left\langle \overbrace{\overline{r}_{2(n+p-1)} \cdots \overline{r}_{2p}}^{n} + \overbrace{\overline{r}_{2(p-1)} \cdots \overline{r}_{20} 1 \cdots 1}^{n} + 2 \right\rangle_{(2^n+1)}. \end{aligned}$$

The last expression can be written as,

$$\langle -2^{n-p} R_2 \rangle_{(2^n+1)} = \langle R_t + R_u + 2^{n-p} + 1 \rangle_{(2^n+1)}, \tag{28}$$

where: $R_t = \overbrace{r_{2(n+p-1)} \cdots r_{2p}}^{n}$, and $R_u = \overbrace{\overline{r}_{2(p-1)} \cdots \overline{r}_{20} 0 \cdots 0}^{n}$.

Applying modulus (2^n+1) to $2^{n-p}R_3$ in (27) leads to:

$$\langle 2^{n-p} R_3 \rangle_{(2^n+1)} = \big\langle 2^{n-p} 2^n r_{3n} + 2^{n-p} \hat{R}_3 \big\rangle_{(2^n+1)}. \tag{29}$$

The term, $\langle 2^{n-p} \hat{R}_3 \rangle_{(2^n+1)}$ of (29) is simplified as follows:

$$\begin{aligned} & \big\langle 2^{n-p} \hat{R}_3 \big\rangle_{(2^n+1)} \\ & = \left\langle \overbrace{0 \cdots 0\, r_{3(n-1)} \cdot\cdot\cdot r_{3p}}^{n} \star \overbrace{r_{3(p-1)} \cdot\cdot\cdot r_{30}\ 0 \cdots 0}^{n} \right\rangle_{(2^n+1)} \\ & = \left\langle \overbrace{0 \cdots 0\, r_{3(n-1)} \cdot\cdot\cdot r_{3p}}^{n} 2^n + \overbrace{r_{3(p-1)} \cdot\cdot\cdot r_{30}\ 0 \cdots 0}^{n} \right\rangle_{(2^n+1)} \\ & = \left\langle - \overbrace{0 \cdots 0\, r_{3(n-1)} \cdot\cdot\cdot r_{3p}}^{n} + \overbrace{r_{3(p-1)} \cdot\cdot\cdot r_{30}\ 0 \cdots 0}^{n} \right\rangle_{(2^n+1)} \\ & = \left\langle \overbrace{1 \cdots 1\, \overline{r}_{3(n-1)} \cdot\cdot\cdot \overline{r}_{3p}}^{n} + \overbrace{r_{3(p-1)} \cdot\cdot\cdot r_{30}\ 0 \cdots 0}^{n} + 2 \right\rangle_{(2^n+1)} \\ & = \left\langle \overbrace{r_{3(p-1)} \cdot\cdot\cdot r_{30}\, \overline{r}_{3(n-1)} \cdot\cdot\cdot \overline{r}_{3p}}^{n} + (2^n - 2^{n-p} + 2) \right. \end{aligned}$$

Defining $R_v = \overbrace{r_{3(p-1)} \cdots r_{30}\, \overline{r}_{3(n-1)} \cdots \overline{r}_{3p}}^{n}$, then the last equation can be written as:

$$\big\langle 2^{n-p} \hat{R}_3 \big\rangle_{(2^n+1)} = \langle R_v + (2^n - 2^{n-p} + 2) \rangle_{(2^n+1)}. \tag{30}$$

Substituting (30) into (29) and observing that $2^{n-p}2^n r_{3n} = \langle -2^{n-p} r_{3n} \rangle_{(2^n+1)}$:

$$\langle 2^{n-p} R_3 \rangle_{(2^n+1)} = \langle R_v - 2^{n-p} - 2^{n-p} r_{3n} + 1 \rangle_{(2^n+1)}. \tag{31}$$

TABLE 3
Truth Table for Adding the Two Bits $(\overline{r}_{3p} + r_{3n})$ of (34)

r_{3n}	$\overline{r}_{3p}$	carry	sum
0	0	0	0
0	1	0	1
1	0	X	X
1	1	1	0

Substituting (31) and (28) into (27) produces:

$$S_3 = \langle R_t + R_u + R_v + 2 + K \rangle_{(2^n+1)}, \tag{32}$$

where $K = \langle -2^{n-p} r_{3n} \rangle_{(2^n+1)} = 2^n - 2^{n-p} r_{3n} + 1$.

The value of K is written as:

$$\begin{aligned} K &= \begin{cases} 0, & r_{3n} = 0 \\ 2^n - 2^{n-p} + 1, & r_{3n} = 1 \end{cases} \\ &= \overbrace{r_{3n} \cdots\cdots r_{3n}}^{p} \overbrace{0 \cdots\cdots 0\, r_{3n}}^{n-p}. \end{aligned} \tag{33}$$

Realizing (32) is shown in the S_3 channel of Fig. 4, where modulo $(2^n + 1)$ addition property is used. CSA5 adds $R_u + R_v + K$ while CSA7 adds the result of CSA5 with CEAC to R_t. The results of CSA7 with CEAC are added using a modulo $(2^n + 1)$ adder. The operands applied to CSA5 are processed as follows:

$$\left\langle \begin{array}{l} \langle \overbrace{\overline{r}_{2(p-1)} \cdots\cdots \overline{r}_{21} \overline{r}_{20}}^{p} \overbrace{0 \cdots\cdots\cdots 0\ 0}^{n-p} \\ + r_{3(p-1)} \cdots\cdots r_{31} r_{30}\ \overline{r}_{3(n+p-1)} \cdots\cdots\ \overline{r}_{3p} \\ + r_{3n} \cdots\cdots r_{3n} r_{3n}\ 0 \cdots\cdots\cdots\ 0\ r_{3n} \rangle_{2^n} \\ + \qquad\qquad\qquad\qquad\qquad\qquad \overline{c}_{out} \end{array} \right\rangle_{2^n+1}. \tag{34}$$

According to (34), evaluating $\langle R_u + R_v + K \rangle_{2^n}$ as a sum vector and a carry vector is performed as follows:

- The truth table for adding $\overline{r}_{3p}$ to r_{3n} is given in Table 3, where the sum = $\overline{r}_{3p} \wedge \overline{r}_{3n}$ and the carry = r_{3n}.
- Adding the MS p bits of (34) requires p ASFA circuits (referred to as ASFA3). These p ASFA3 circuits constitute the MS p bits of CSA5 of Fig. 4. The truth table for ASFA3 is given in Table 4. The corresponding logic equations are given by: carry $= \overline{r}_{2i} \wedge (r_{3n} \vee r_{3i})$ and sum $= (r_{2i} \wedge r_{3i}) \vee (r_{2i} \wedge r_{3n}) \vee (\overline{r}_{2i} \wedge \overline{r}_{3i} \wedge \overline{r}_{3n})$.

Applying modulo 2^{n+p} to (22) to compute S_2 leads to:

$$S_2 = \left\langle \left\langle 2^{n-p-1}((2^n+1)R_1 - 2R_2 + (1-2^n)R_3) \right\rangle_{(2^{2n}-1)} \right\rangle_{2^{n+p}}. \tag{35}$$

Using the modulo $(2^n - 1)$ properties, the three terms on the RHS of (35) can be expressed as:

$$\begin{aligned} \widetilde{R}_1 &= (2^n+1)R_1 \\ &= \overbrace{\overline{r}_{1(n-1)} \cdots\cdots r_{10}}^{n} \overbrace{\overline{r}_{1(n-1)} \cdots\cdots r_{10}}^{n} \end{aligned} \tag{36}$$

$$\begin{aligned} \widetilde{R}_2 &= \langle -2R_2 \rangle_{(2^{2n}-1)} \\ &= \overbrace{1 \cdots 1}^{n-p-1} \overbrace{\overline{r}_{2(n+p-1)} \cdots\cdots \overline{r}_{20}}^{n+p} \overbrace{1}^{1} \end{aligned} \tag{37}$$

$$\begin{aligned} \widetilde{R}_3 &= \langle (1-2^n)R_3 \rangle_{(2^{2n}-1)} \\ &= \overline{\hat{R}}_3 \star \hat{R}_3 + (2^n - 1)(1 + r_{3n}), \end{aligned} \tag{38}$$

TABLE 4
Truth Table for ASFA3 Circuit in (34) for Adding $\overline{r}_{2i}$, r_{3n} and r_{3i}, Where $0 \le i \le (p-1)$

$\overline{r}_{2i}$	r_{3n}	r_{3i}	carry	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	X	X

where (38) is identical in its derivation to (13).

In binary notation, (38) is written as:

$$\begin{aligned} \widetilde{R}_3 = &\overbrace{\overline{r}_{3(n-1)} \cdots\cdots \overline{r}_{30}}^{n} \overbrace{\overline{r}_{3(n-1)} \cdots\cdots r_{30}}^{n} \\ &+ \overbrace{0 \cdots\cdots\cdots 0\ r_{3n}}^{n} \overbrace{1 \cdots\cdots\cdots 1\ \overline{r}_{3n}}^{n} \end{aligned}. \tag{39}$$

If $r_{3n} = 0$, (39) can be rewritten in the form:

$$\begin{aligned} \widetilde{R}_3 = &\overbrace{\overline{r}_{3(n-1)} \cdots\cdots \overline{r}_{30}}^{n} \overbrace{\overline{r}_{3(n-1)} \cdots\cdots r_{30}}^{n} \\ &+ \overbrace{0 \cdots\cdots\cdots\cdots 0\ 0}^{n} \overbrace{1 \cdots\cdots\cdots 1\ 1}^{n} \end{aligned}. \tag{40}$$

In (39), the case $r_{3n} = 1$ implies that $\overbrace{\overline{r}_{3(n-1)} \cdots\cdots \overline{r}_{30}}^{n} = \overbrace{1 \cdots\cdots 1}^{n}$ and implies that $\overbrace{\overline{r}_{3(n-1)} \cdots\cdots r_{30}}^{n} = \overbrace{0 \cdots\cdots 0}^{n}$. Therefore, if $r_{3n} = 1$, (39) is rewritten as:

$$\begin{aligned} \widetilde{R}_3 = &\overbrace{1 \cdots\cdots\cdots 1}^{n} \overbrace{0 \cdots\cdots\cdots 0}^{n} \\ &+ \overbrace{0 \cdots\cdots 0\ 1}^{n} \overbrace{1 \cdots\cdots\cdots 1\ 0}^{n} \end{aligned}, \tag{41}$$

when adding the two RHS terms of (41) and applying modulus $(2^{2n} - 1)$, there will be an output carry that can be re-inserted as the LS bit in the second term, hence:

$$\begin{aligned} \widetilde{R}_3 = &\overbrace{0 \cdots\cdots\cdots 0}^{n} \overbrace{0 \cdots\cdots\cdots\cdots 0}^{n} \\ &+ \overbrace{0 \cdots\cdots 0\ 0}^{n} \overbrace{1 \cdots\cdots\cdots 1\ 1}^{n}. \end{aligned} \tag{42}$$

Unifying (40) and (42) for both cases of r_{3n} produces:

$$\begin{aligned} \widetilde{R}_3 = &\overbrace{\overline{r}_{q(n-1)} \cdots\cdots r_{q0}}^{n} \overbrace{\overline{r}_{3(n-1)} \cdots\cdots r_{30}}^{n} \\ &+ \overbrace{0 \cdots\cdots\cdots\cdots 0\ 0}^{n} \overbrace{1 \cdots\cdots\cdots\cdots 1}^{n}, \end{aligned} \tag{43}$$

where $r_{qi} = \overline{r_{3i} \vee r_{3n}}$. Alternatively, (43) is written as:

$$\widetilde{R}_3 = R_q + (2^n - 1), \tag{44}$$

where, $R_q = \overbrace{\overline{r}_{q(n-1)} \cdots\cdots r_{q0}}^{n} \overbrace{\overline{r}_{3(n-1)} \cdots\cdots r_{30}}^{n}$.

This leads to rewriting (35) in the form,

$$S_2 = \left\langle \langle 2^{n-p-1}(\widetilde{R}_1 + \widetilde{R}_2 + R_q + (2^n - 1)) \rangle_{(2^{2n}-1)} \right\rangle_{2^{n+p}}. \tag{45}$$

The channel producing S_2 in Fig. 4 shows the block diagram of implementing (45). CSA4 adds $\widetilde{R}_1$, $\widetilde{R}_2$ and R_q, where each consists

of $2n$ bits. Formulating R_q requires n NOR gates. This formulation adds no delay (i.e., similar to r_x, r_x and r_z in Fig. 2. CSA6 in Fig. 4 adds the sum and carry vectors from CSA4 to (2^n-1). The result of CSA6 is applied to a modulo $(2^{2n}-1)$ adder. The LS $(n+p)$ bits of the modular adder constitute S_2.

If the residue digit R_2 is concatenated to the output of the modulo adder, where R_2 forms the LS part, the result is X (i.e., a residue to binary converter for the extended moduli set).

Example 2. Given the moduli set $(31, 128, 33)$, the intention is to scale the residue digits $(14, 80, 27)$ by $2^7 = 128$, where $n = 5$, $p = 2$ and $r_{3n} = 0$.

$R_1 = (01110)_2$, $R_2 = (1010000)_2$ and $R_3 = (11011)_2$. $R_{2_U} = (00010)_2$, $\overline{R}_{2_U} = (11101)_2$, while, $R_{2_L} = (10000)_2$, $\overline{R}_{2_L} = (01111)_2$. Using (26), $S_1 = \left\langle 2^{n-p}(R_1 + \overline{R}_{2_U} + \overline{R}_{2_L})\right\rangle_{(2^n-1)} = \langle 2^3(01110 + 11101 + 01111)_2\rangle_{31} = 30$. Using (32), $S_3 = \langle R_t + R_u + R_v + \ 2 + K\rangle_{(2^n+1)}$, $R_t = (10100)_2$, $R_u = (11000)_2$, $R_v = (11001)_2$ and $K = (00000)_2$, therefore, $S_3 = \left\langle (10100)_2 + (11000)_2 + (11001)_2 + 2 + (00000)_2\right\rangle_{(2^n+1)} = 5$. Using (45), $S_2 = \langle\langle 2^{n-p-1}(\widetilde{R}_1 + \widetilde{R}_2 + R_q + (2^n-1))\rangle_{(2^{2n}-1)}\rangle_{2^{n+p}}$, Using (36), $\widetilde{R}_1 = 462$. Using (37), $\widetilde{R}_2 = \langle -2R_2\rangle_{(2^{2n}-1)} = \langle -160\rangle_{1023} = 863$. Using (44), $\widetilde{R}_3 = R_q + (2^n-1)$, $R_q = \overbrace{00100}^{5}\overbrace{11011}^{5} = 155$, $\widetilde{R}_3 = 155 + 31 = 186$. Therefore, $S_2 = \langle\langle 2^2(462 + 863 + 155 + 31)\rangle_{1023}\rangle_{128} = 33$. Actually, $X = 118{,}992$, while $\lfloor\frac{X}{2^7}\rfloor = 929$.

Repeating for residue digits $(14, 80, 32)$, where $R_3 = (100000)_2$ and $r_{3n} = 1$. $S_1 = 30$ (stays unchanged). When computing S_3, R_v and K change to: $R_v = (00111)_2 = 7$ and $K = (11001)_2 = 25$. Hence, $S_3 = \left\langle (10100)_2 + (11000)_2 + (00111)_2 + 2 + (11001)_2\right\rangle_{(2^n+1)} = 12$.

When computing S_2, $\widetilde{R}_3$ changes as follows: $\widetilde{R}_3 = R_q + (2^n-1)$, $R_q = (\overbrace{00000}^{5}\overbrace{00000}^{5})_2 = 0$, $\widetilde{R}_3 = 0 + 31 = 31$. Hence, $S_2 = \langle\langle 2^2(462 + 863 + 0 + 31)\rangle_{1023}\rangle_{128} = 53$. In fact, $X = 39{,}632$, while $\lfloor\frac{X}{2^7}\rfloor = 309$.

3.2 The Scaling Factor $= 2^n$

Reconsidering (21) and dividing both sides by 2^n leads to:

$$\left\lfloor\frac{X}{2^n}\right\rfloor = \left\langle (2^n+1)2^{n-1}R_1 - 2^nR_2 + \left\lfloor\frac{R_2}{2^n}\right\rfloor - (2^n-1)2^{n-1}R_3\right\rangle_{(2^{2n}-1)}. \tag{46}$$

The term -2^nR_2 in (46) can be simplified as follows:

$$\begin{aligned}\langle -2^nR_2\rangle_{(2^{2n}-1)} &= \left\langle -2^n(R_{2_U}2^n + R_{2_L})\right\rangle_{(2^{2n}-1)}\\ &= \left\langle -2^{2n}R_{2_U} - 2^nR_{2_L}\right\rangle_{(2^{2n}-1)}\\ &= \left\langle -R_{2_U} - 2^nR_{2_L}\right\rangle_{(2^{2n}-1)}.\end{aligned} \tag{47}$$

The term $\left\lfloor\frac{R_2}{2^n}\right\rfloor$ in (46) is also simplified as follows:

$$\begin{aligned}\left\lfloor\frac{R_2}{2^n}\right\rfloor &= \left\lfloor\frac{R_{2_U}2^n + R_{2_L}}{2^n}\right\rfloor\\ &= R_{2_U} + \left\lfloor\frac{R_{2_L}}{2^n}\right\rfloor\\ &= R_{2_U}\end{aligned}. \tag{48}$$

Substituting (47) and (48) into (46) produces:

$$\left\lfloor\frac{X}{2^n}\right\rfloor = \left\langle (2^n+1)2^{n-1}R_1 - 2^nR_{2_L} - (2^n-1)2^{n-1}R_3\right\rangle_{(2^{2n}-1)}. \tag{49}$$

Taking 2^{n-1} as a common factor and applying modulo $(2^{2n}-1)$ to (49) to all the terms lead to:

$$\begin{aligned}\left\langle\left\lfloor\frac{X}{2^n}\right\rfloor\right\rangle_{(2^{2n}-1)} = \Big\langle 2^{n-1}\Big(&\langle(2^n+1)R_1\rangle_{(2^{2n}-1)}\\ &+ \left\langle -2R_{2_L}\right\rangle_{(2^{2n}-1)}\\ &- \langle(2^n-1)R_3\rangle_{(2^{2n}-1)}\Big)\Big\rangle_{(2^{2n}-1)}.\end{aligned} \tag{50}$$

Equation (50) has an identical form to (2), where R_2 has been replaced with R_{2_L}. Therefore, Fig. 3 realizes (50) and provides a scaler for the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by the factor 2^n, for $1 \le p < n$, where just the LS n bits of R_2 are used.

Example 3. Considering the moduli set $(31, 128, 33)$ and $X =$, with residue value of $X = (14, 84, 27)$. This implies that $R_{2_L} = (10100)_2 = 20$. Therefore, the analysis in the first part of Example 1 holds for this case.

Similarly, when repeating for residue digits, $X = (14, 84, 32)$, where $R_3 = (100000)_2$ and $r_{3n} = 1$. This also implies that $R_{2_L} = (10100)_2 = 20$. Therefore, the analysis in the second part of Example 1 holds for this case too.

3.3 Scaling the Three Moduli Set $(2^n-1, 2^{2n}, 2^n+1)$

The multiplicative inverses of the moduli set $(2^n-1, 2^{2n}, 2^n+1)$ are [29]: $\left\langle\hat{m}_1^{-1}\right\rangle_{m_1} = 2^{n-1}$, $\left\langle\hat{m}_2^{-1}\right\rangle_{m_2} = \langle -1\rangle_{m_2}$, and $\left\langle\hat{m}_3^{-1}\right\rangle_{m_3} = 2^{n-1}$. Following the same approach of the previous sections, it can be concluded that scaling X by 2^{2n} leads to S_1 and S_3 as given in (23) and (27), respectively, where $n = p$. However, $S_2 = \langle 2^{n-1}(2^n+1)R_1 - R_2 + 2^{n-1}(2^n-1)R_3\rangle_{(2^{2n}-1)}$, which can be simplified using the same simplification approach of (35). Similarly, scaling X by 2^n leads to (50).

4 Performance, Comparison and VLSI Synthesis

The work in [21] presented a scaler for the three moduli set $(2^n-1, 2^n, 2^n+1)$ scaled by 2^n. The structure of the scaler in [21] was compared with [13], [14], [16], [17] and [18] and shown to be considerably more efficient in terms of area and time. The work in [24] presented a scaler for the three moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^n for $0 \le p \le n$. The scaler of [24] was compared with the one in [21]. The two scalers of [21] and [24] are the most relevant, recent and efficient scalers found in the literature. Hence, the proposed scaler for the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^n is compared with the one in [21] (for the case $p = 0$) and with that in [24] (for the cases: $0 \le p \le n$). The comparison held in this section is based on VLSI synthesis tools.

The other structure proposed in this paper for the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^{n+p} for $1 \le p \le n$ will also be evaluated using VLSI tools.

The scalers proposed in this paper along with the most recent ones in [21] and [24] were all modeled using Verilog HDL, synthesized using Synopsys Design Compiler (G-2012.06) and mapped to 65 nm Synopsys DesignWare Digital Logic Libraries. All modeled and synthesized designs were optimized, checked and verified to be correctly operational. The power consumption was estimated using Synopsys Power Compiler.

In all synthesized structures, modulo (2^n-1) adder of [30] and modulo (2^n+1) adder of [31] were used. The modulo (2^n-1) proposed in [30] is composed of a preprocessing unit followed by a parallel-prefix unit. While the parallel-prefix unit is a conventional structure, the preprocessing unit uses Ling's definition for computing the carry [32]. This definition of carry eliminates one level of computational logic, thus, providing a very high speed modulo adder. The modulo (2^n+1) introduced in [31] proposes a parallel-

TABLE 5
Synthesis Results of the Scaler of the Moduli Set $(2^n-1, 2^{n+p}, 2^n+1)$ Scaled by 2^n, $0 \le p \le n$

Scaler	n	p	Area, μm^2	Delay, ps	Power, μW
Proposed	7	0	2,051	446	68.1
		2	2,072	452	68.9
		5	2,082	453	70.1
		7	2,085	455	70.5
	11	0	3,729	524	95.9
		4	3,745	530	97.8
		8	3,770	539	101.2
		11	3,763	541	102.0
	15	0	5,129	553	117.5
		5	5,163	581	120.2
		10	5,181	585	125.7
		15	5,173	584	125.2
[21]	7	0	2,882	541	85.8
	11	0	5,163	584	130.3
	15	0	7,043	641	146.9
[24]	7	0	2,209	581	77.2
		2	2,271	588	79.1
		5	2,322	596	84.4
		7	2,375	591	84.7
	11	0	4,025	651	112.1
		4	4,118	673	115.5
		8	4,203	682	117.3
		11	4,266	685	122.0
	15	0	5,612	695	141.2
		5	5,701	699	147.4
		10	5,763	707	153.6
		15	5,889	716	154.0

prefix structure that recirculates the carries in every level of the parallel-prefix structure. The speed of this adder approaches that of integer and modulo (2^n-1) adders.

The following synthesis results were obtained,

- The scaler of the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^n.

  The design proposed in this paper (i.e., Fig. 3), the design presented in (Fig. 5 of [21]) and the design presented in (Figs. 3 and 4 of [24]) were all synthesized for values of $n = 7, 11, 15$ and different values of p. The scaler in [21] was synthesized for the case $p = 0$. Results were reported in Table 5 after place & route phase. Reductions in area and time achieved by the proposed scaler are shown in Table 6.

TABLE 6
Area, Time-Delay and Power Reductions of the Proposed Scaler for the Moduli Set $(2^n-1, 2^{n+p}, 2^n+1)$ Scaled by 2^n, $0 \le p \le n$

Proposed scaler compared with	n	p	% Area Reduction	% Delay Reduction	% Power Reduction
[21]	7	0	28.8	17.6	20.6
	11	0	27.8	10.3	26.4
	15	0	27.2	13.7	20.0
[24]	7	0	7.1	23.2	11.8
		2	8.8	23.1	12.9
		5	10.3	24.0	16.9
		7	12.2	23.0	16.8
	11	0	7.3	19.5	14.5
		4	9.1	21.2	15.3
		8	10.3	21.0	13.7
		11	11.8	21.0	16.4
	15	0	8.6	20.4	16.8
		5	9.4	16.9	18.5
		10	10.1	17.3	18.2
		15	12.2	18.4	18.7

TABLE 7
Synthesis Results of the Proposed Scaler for the Moduli Set $(2^n-1, 2^{n+p}, 2^n+1)$ Scaled by 2^{n+p}, $1 \le p \le n$

n	p	Area, μm^2	Delay, ps	Power, μW
7	2	2,325	527	78.2
	4	2,683	538	83.3
	6	3,026	540	87.5
	7	3,303	546	93.3
11	2	4,104	601	122.5
	5	4,709	612	129.2
	8	5,249	615	138.0
	11	5,797	626	141.6
15	2	5,635	631	140.3
	5	6,294	639	149.6
	10	7,133	650	158.7
	15	8,108	664	166.1

The percentage of reduction was computed based on the formula: $\frac{\text{Competitive} - \text{Proposed}}{\text{Competitive}} \times 100$ percent.

Examining Table 6 reveals that compared with the scaler in [21], the scaler proposed in Fig. 3 achieved an average area reduction of 27.9 percent, an average time reduction of 13.9. percent and an average power reduction of 22.3 percent. When compared with the recent scaler presented in [24], the proposed scaler in Fig. 3 achieved considerable reductions. Table 6 indicates that the average reductions in area, time and power are 9.8, 20.8 and 15.9 percent, respectively.

- The scaler of the moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ scaled by 2^{n+p}.

  The proposed scaler in this paper (Fig. 4, $1 \le p \le n$), for values of $n = 7, 11, 15$ and different values of p, was also synthesized. Results after place & route layout are listed in Table 7. To the best of the authors' knowledge, there is no scaler for the same extended moduli set scaled by 2^{n+p}. Hence, there is no comparison with a functionally-similar competitive work. Although different in their scaling factors, the new proposed scaler in Fig. 4 is still more time-efficient compared with the time of the one in [24]. The average time of the proposed 2^{n+p} scaler is 8.6 percent less than the one in [24]. Expectedly, the area and power of the 2^{n+p} scaler are larger due to the difference in the scaling factor which imposes additional partial variables.

5 Conclusion

This work presented two scaling structures. The first structure scales the extended three moduli set $(2^n-1, 2^{n+p}, 2^n+1)$ by 2^n, where $0 \le p \le n$. Compared with the most recent similar work, this structure showed considerable reductions in area, time and power. The second structure scales the same extended set by 2^{n+p}, where $1 \le p \le n$. Although this scaler is an area-demanding structure due to the 2^{n+p} scaling factor which generates additional partial variables, experimental analysis showed that the structure is very time-efficient.

References

[1] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd edition. New York, NY, USA: Oxford Univ. Press, 2010.

[2] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*. New York, NY, USA: McGraw Hill, 1967.

[3] M. A. Soderstrand, W. Jenkins, G. Jullien, and F. Taylor, Eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York, NY, USA: IEEE Press, 1986.

[4] P. V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures*. Berlin, Germany: Springer, 2002.

[5] T. Keller, T. H. Liew, and L. Hanzo, "Adaptive redundant residue number system coded multi-carrier modulation," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 11, pp 2292–2301, Nov. 2000.

[6] R. Chaves and L. Sousa, "RDSP: A RISC DSP based on residue number system," in *Proc. Symp. Digit. Syst. Des. Archit. Methods Tools*, Sep. 2003, pp. 128–135.

[7] J. Ambrose, H. Pettenghi, D. Jayasinghe, and L. Sousa, "A randomised multi-modulo residue number system architecture for double-and-add to prevent power analysis side channel attacks," *IET Circuit Devices Syst.*, vol. 7, no. 5, pp. 283–293, Sep. 2013.

[8] C. Vun, A. Premkumar, and W. Zhang, "A new RNS based DA approach for inner product computation," *IEEE Trans. Circuit Syst. CAS-I*, vol. 60, no. 9, pp. 2139–2152, Aug. 2013.

[9] X. Zheng, B. Wang, C. Zhou, X. Wei, and Q. Zhang, "Parallel DNA arithmetic operation with one error detection based on 3-moduli set," *IEEE Trans. Nanoscience*, vol. 15, no. 5, pp. 499–507, Jul. 2016.

[10] C. B. Dutta, P. Garai, and A. Sinha, "Design of a reconfigurable DSP processor with bit efficient residue number system," *Int. J. VLSI Des. Commun. Syst.*, vol. 3, no. 5, pp. 175–189, Oct. 2012.

[11] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi, "Efficient RNS implementation of elliptic curve point multiplication over GF(p)," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 8, pp 1545–1549, Aug. 2013.

[12] M. Griffin, F. Taylor, and M. Sousa, "New scaling algorithms for the chinese remainder theorem," in *Proc. 22nd Asilomar Conf. Signals Syst. Comput.*, 1988, pp. 375–378.

[13] M. Shenoy and R. Kumaresan, "A fast and accurate RNS scaling technique for high speed signal processing," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 37, no. 6, pp. 929–937, Mar. 1988.

[14] F. Barsi and M. Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations," *IEEE Trans. Signal Process.*, vol. 43, no. 10, pp. 2427–2430, Oct. 1995.

[15] Z. Ulman and M. Czyzak, "Highly parallel, fast scaling of numbers in nonredundant residue arithmetic," *IEEE Trans. Signal Process.*, vol. 46, no. 2, pp. 487–496, Feb. 1998.

[16] A. Garcia and A. Lloris, "A look-up scheme for scaling in the RNS," *IEEE Trans. Comput.*, vol. 48, no. 7, pp. 748–751, Jul. 1999.

[17] M. Dasygenis, K. Mitroglou, D. Soudris, and A. Thanailakis, "A full-adder-based methodology for the design of scaling operation in residue number system," *IEEE Trans. Circuits Syst.*, vol. 55, no. 2, pp. 546–558, Mar. 2008.

[18] J. Ye, S. Ma, and J. Hu, "An efficient 2^n RNS scaler for moduli set $(2^n-1, 2^n, 2^n+1)$," in *Proc. IEEE Symp. Inf. Sci. Eng.*, Dec. 20–22, 2008, pp. 511–515.

[19] K. Yinan and B. Phillips, "Fast scaling in the residue number system," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 3, pp. 443–447, Mar. 2009.

[20] S. Ma, J. Hu, Y. Ye, L. Zhang, and X. Ling, "A 2^n scaling scheme for signed RNS integers and its VLSI implementation," *Sci. China Series F: Inf. Sci.*, vol. 53, no. 1, pp. 203–212, 2010.

[21] C. H. Chang, J. Low, and S. Yung, "Simple, fast, and exact RNS scaler for the three-moduli set $(2^n-1, 2^n, 2^n+1)$," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 58, no. 11, pp. 2686–2697, Nov. 2011.

[22] J. Y. S. Low and C. H. Chang, "A VLSI efficient programmable power-of-two scaler for $(2^n-1, 2^n, 2^n+1)$," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 59, no. 12, pp. 2911–2919, Dec. 2012.

[23] T. Tay, C. H. Chang, and J. Low, "Efficient VLSI implementation of 2^n scaling of signed integer in RNS $(2^n-1, 2^n, 2^n+1)$," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 10, pp. 1936–1940, Oct. 2013.

[24] L. Sousa, "2^n RNS scalers for extended 4-moduli sets," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3322–3334, Dec. 2015.

[25] R. Chaves and L. Sousa, "$\{2^n+1, 2^{n+k}, 2^n1\}$: A new RNS moduli set extension," in *Proc. Euromicro Symp. Digit. Syst. Des.*, 2004, pp. 210–217.

[26] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Proc. 14th IEEE Symp. Comput. Arithmetic*, Apr. 14–16, 1999, pp. 158–167.

[27] A. Sweidan and A. Hiasat, "New efficient memoryless, residue to binary converter," *IEEE Trans. Circuits Syst.*, vol. 35, no. 11, pp. 1441–1444, Nov. 1988.

[28] A. Hiasat, "A sign detector for a group of three-moduli sets," *IEEE Trans. Comput.*, vol. 65, no. 12, pp 3580–3590, Dec. 2016.

[29] A. Hiasat and A. Sweidan, "Residue-to-binary decoder for an enhanced moduli set," *IEEE Proc.-Comput. Digit. Techn.*, vol. 151, no. 2, pp. 127–130, Mar. 2004.

[30] G. Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos, and C. Efstathiou, "New architectures for modulo 2^n-1 adders," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, Dec. 2005, pp. 1–4.

[31] H. T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-one modulo 2^n+1 adder design," *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1389–1399, Dec. 2002.

[32] H. Ling, "High-speed binary adder," *IBM J. R&D*, vol. 25, pp. 156–166, May 1981.