Received: 28 July 2018 | Revised: 11 November 2019 | Accepted: 3 December 2019

DOI: 10.4218/etrij.2018-0408

ORIGINAL ARTICLE

ETRI Journal WILEY

Efficient programmable power-of-two scaler for the three-moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$

MohammadReza Taheri[1] | **Keivan Navi**[1] | **Amir Sabbagh Molahosseini**[2]

[1]Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran

[2]Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran

Correspondence
Keivan Navi, Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.
Email: navi@sbu.ac.ir

Scaling is an important operation because of the iterative nature of arithmetic processes in digital signal processors (DSPs). In residue number system (RNS)–based DSPs, scaling represents a performance bottleneck based on the complexity of inter-modulo operations. To design an efficient RNS scaler for special moduli sets, a body of literature has been dedicated to the study of the well-known moduli sets $\{2^n - 1, 2^n, 2^n + 1\}$ and $\{2^n, 2^n - 1, 2^{n+1} - 1\}$, and their extension in vertical or horizontal forms. In this study, we propose an efficient programmable RNS scaler for the arithmetic-friendly moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$. The proposed algorithm yields high speed and energy-efficient realization of an RNS programmable scaler based on the effective exploitation of the mixed-radix representation, parallelism, and a hardware sharing technique. Experimental results obtained for a 130 *nm* CMOS ASIC technology demonstrate the superiority of the proposed programmable scaler compared to the only available and highly effective hybrid programmable scaler for an identical moduli set. The proposed scaler provides 43.28% less power consumption, 33.27% faster execution, and 28.55% more area saving on average compared to the hybrid programmable scaler.

KEYWORDS

ASIC, computer arithmetic, digital signal processing, programmable scaler, residue number system (RNS) scaler, VLSI Design

1 | INTRODUCTION

Digital signal processors (DSPs) lie at the heart of many digital products in numerous areas, including multimedia processing, telecommunications, and cryptography. Power consumption and performance are key factors that should be considered in the design of such products. Arithmetic parallelism can be exploited to achieve high-performance and power-efficient implementations of DSPs because many DSP algorithms, such as the finite impulse response, infinite impulse response, fast Fourier transform, and discrete wavelet transform, rely on sum-of-products computations. The parallelism of such arithmetic operations can be realized by using the residue number system (RNS) as an alternative number representation instead of the weighted binary number system [1]. The parallel and modular arithmetic nature of the RNS enhances the performance of the RNS data path and reduces power dissipation without degrading latency [2]. However, the iterative nature of DSP algorithms necessitates the prevention of overflow to avoid generating erroneous results. At first glance, selecting a moduli set that guarantees a sufficiently large dynamic range may appear to be the most appropriate method for solving the overflow issue. This method can be applied by

increasing the number of moduli or increasing the size of one or more moduli. However, the downsides of adopting such an approach lead to overall system performance degradation, increased circuit timing uncertainly, and higher delay variation in the system data path. Therefore, scaling operations (ie, division by an integer) are employed to reduce the word length of intermediate results. Scaling in the RNS is known as a complex operation due to the use of inter-modulo computations. The long delay and significant hardware complexity of scaling in DSP applications has turned efficient scaler design into a major issue when designing RNS-based DSPs.

Several contributions to RNS scaling problems have been reported in the literature. Previous RNS scaler schemes can be categorized based on different criteria [3–5]. The accuracy of an RNS scaler is the first criterion. According to this criterion, an RNS scaler can be categorized into two classes. In the first class, the Chinese remainder theorem approach can be utilized to derive inexact scaled results with some fractional errors [6–10]. Scalers belonging to this class may not be appropriate for certain critical applications. The second class focuses on the exact RNS scaling problem. In such techniques, in the first step, a ceiling function is applied to the division of a weighted number X by a scaling factor. In the second step, modulo reduction is applied to the result of the first step. In the latter step, an expected result is generated via base extension or other iterative algorithms [11–13]. Another criterion is the scaling factor of an RNS scaler, which can be categorized into three main classes: scaling using one modulus or the product of several moduli, scaling using a co-prime factor with the moduli of an RNS, and scaling using a power-of-two factor. Most recent contributions to the RNS scaling problem are based on the third class because implementations using the first and second classes result in poor performance and considerable hardware costs [14–17]. The hardware implementation technique is another criterion and can be classified into two groups. The hardware realization of the first group requires lookup tables [6–12,15,16], while the second group, which is known as adder-based implementation, uses combinational logic [3–5,13,14,17–20]. The implementation of the former approach suffers from two main problems. First, there is a dramatic increase in hardware costs when increasing the dynamic range. Second, processing throughput is restricted and degraded by poor pipelining efficiency.

In recent years, the design of exact, memoryless, power-of-two scalers has attracted the attention of many researchers owing to the excellent efficiency of such systems [5]. In [4], a high-performance, low-hardware-cost exact scaler was proposed for the three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. A 2^n signed integer scaler, which is an extension of the method in [4], was proposed in [18] based on a low-complexity and high-performance hardware implementation. In [19], an efficient programmable power-of-two scaler was presented for the three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ to reduce the risk of over-scaling problems (ie, scaling more than necessary, which leads to unnecessary accuracy reduction). In this method, the scaling factor can be identified at run time. Recent works in [5] and [20] have presented power-of-two scalers for the extended moduli set $\{2^n - 1, 2^{n+p}, 2^n + 1\}$. The former presented an accurate and efficient 2^n scaler for the augmented three-moduli set, as well as its extension set an with an additional co-prime moduli. The latter presented 2^n and 2^{n+p} scalers for the augmented moduli set $\{2^n - 1, 2^{n+p}, 2^n + 1\}$ with efficient hardware structures. Another class of moduli sets in the form of $\{2^a, 2^b - 1, 2^c - 1\}$ with $c = b - 1$ or $c = b + 1$ has also attracted the attention of researchers due to its efficient and modulo-$(2^n + 1)$-free internal arithmetic operations [21–28]. Based on the impressive theoretical properties of this type of moduli set, different algorithms and hardware implementations for a variety of RNS operations have been proposed in the literature. In the search for an efficient hardware implementation of the reverse converter, several contributions have been proposed for the moduli sets $\{2^n, 2^n - 1, 2^{n\pm1} - 1\}$ [21–23], $\{2^{n+p}, 2^n - 1, 2^{n-1} - 1\}$ [24], and $\{2^{2n}, 2^n - 1, 2^{n\pm1} - 1\}$ [25]. For $\{2^n, 2^n - 1, 2^{n+1} - 1\}$, efficient sign detectors have also been proposed in [26] and [27]. Recently, a remarkable effort resulted in the efficient hardware implementation of a sign-magnitude comparator for the moduli set $\{2^n, 2^n - 1, 2^{n+1} - 1\}$ [28]. Additionally, an elegant design approach for the enhanced arithmetic unit of an RNS was proposed in [29] based on a shared hardware/software method. The chosen moduli set consists of an even modulus 2^n with remaining moduli of type $2^m - 1$. Although a few contributions have been made regarding the design of RNS scalers for the aforementioned three-moduli sets [30,31], scaling problems with programmability features for this class of moduli sets have not yet been considered.

This paper proposes a novel exact scaler for the efficient moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ $(p \in [0, n])$. The proposed scaler has the ability to determine the scaling factor at run time. To achieve this goal, for the first time, a programmable power-of-two scaler is presented for this augmented three-moduli set. By using a variable scaling factor 2^λ, up to half of the dynamic range of a given RNS can be scaled during runtime for the particular case of $p = n$. The design approach in [29] can be considered as a potential application of the proposed programmable scaler to ensure its reliability. The efficiency of the proposed RNS scaler has a direct impact on the efficiency of various applications, such as that presented in [29]. The remainder of this paper is organized as follows. Section 2 introduces notational conventions that are used throughout this paper as well as the essential theoretical background of the RNS. The algorithm and architecture for the proposed programmable power-of-two RNS scaler are presented in Section 3. The very-large-scale integration (VLSI) architecture of the proposed scaler based on the derived formulations is presented in Section 4. Performance evaluations based on gate-level and

experimental results are presented in Section 5. Section 6 contains our concluding remarks.

2 | BACKGROUND

2.1 | Notations

The following notations have been adopted in this paper:

- $\langle\alpha\rangle_\beta$ denotes the residue of α modulo β.
- $\lfloor\partial\rfloor$ is the greatest integer that is less than or equal to ∂.
- $\left\langle m_i^{-1}\right\rangle_{m_j}$ is the multiplicative inverse of $\langle m_i\rangle_{m_j}$.
- $A_{\alpha:\beta}$ refers to an $(\alpha-\beta+1)$-bit slice of the A vector from position β to position α (ie, $A_{\alpha:\beta}=A_\alpha A_{\alpha-1}\cdots A_{\beta+1}A_\beta$)
- $CLS\ (\alpha,\ \beta)$ indicates shifting α circularly to the left by β bits.
- $\overline{\omega}$ denotes the one's complement of ω vector.
- $\|$ is an operator that concatenates two integer numbers in a binary representation.
- $||_\alpha^\beta$ refers to a vector that is formed by β concatenated α strings (i.e., $||_\alpha^\beta=\underbrace{\alpha||\alpha||\cdots||\alpha}_{\beta}$).
- $\wedge$, $\vee$, $\oplus$, and $\odot$ are bitwise AND, OR, XOR, and XNOR operators, respectively.

2.2 | RNS preliminaries

An RNS is characterized by a finite set of the L co-prime numbers $\{m_1, m_2, \ldots, m_L\}$ with $L\geq 2$, where m_i are moduli. The dynamic range M of an RNS is defined by the product of all the moduli. Any integer $X\in[0,M)$ has a unique representation with an L-tuple $(x_1, x_2, \ldots, x_L)$, where x_i is the minimum non-negative remainder of the division of X by m_i. Based on the isomorphic relationship between $\langle X\circ Y\rangle_M$ and $\left(\langle x_1\circ y_1\rangle_{m_1},\ldots,\langle x_L\circ y_L\rangle_{m_L}\right)$, where $\circ\in\{+,-,\times\}$ and $X,Y\in[0,M)$, the computations of residues are performed in parallel in independent arithmetic channels [2]. The scaled result of a given weighted number X with a scaling factor of K can be expressed as follows [7]:

$$S=\left\lfloor\frac{X}{K}\right\rfloor=\frac{X-\langle X\rangle_K}{K}. \tag{1}$$

In the RNS $\{m_1, m_2, \ldots, m_L\}$ with the mixed-radix representation $[v_L, v_{L-1}, \ldots, v_1]$, the weighted number X can be derived as follows [2]:

$$X=v_L\prod_{i=1}^{L-1}m_i+\cdots+v_3\,m_2\,m_1+v_2\,m_1+v_1, \tag{2}$$

where the mixed-radix coefficients v_i can be formulated as the following expressions:

$$\begin{aligned}v_1&=x_1,\\ v_2&=\left\langle\left(x_2-x_1\right)\left\langle m_1^{-1}\right\rangle_{m_2}\right\rangle_{m_2},\\ v_3&=\left\langle\left(\left(x_3-x_1\right)\left\langle m_1^{-1}\right\rangle_{m_3}-v_2\right)\left\langle m_2^{-1}\right\rangle_{m_3}\right\rangle_{m_3},\end{aligned} \tag{3}$$

which can be generalized as follows:

$$\begin{aligned}v_i=\Big\langle\Big(\Big(\Big(\Big(\left(x_i-x_1\right)\left\langle m_1^{-1}\right\rangle_{m_i}-v_2\Big)\left\langle m_2^{-1}\right\rangle_{m_i}-v_3\Big)\\ \left\langle m_3^{-1}\right\rangle_{m_i}-\cdots-v_{i-1}\Big)\left\langle m_{i-1}^{-1}\right\rangle_{m_i}\Big\rangle_{m_i}.\end{aligned} \tag{4}$$

The following modular arithmetic properties were exploited in this study to simplify the hardware realization of the proposed scaling algorithm.

Property 1 [4]:

$$\langle 2^\rho\times z\rangle_{2^n-1}=\left\langle CLS\left(z,\langle\rho\rangle_n\right)\right\rangle_{2^n-1},\quad 0\leq z\leq 2^n-1. \tag{5}$$

Property 2 [4]:

$$\langle -z\rangle_{2^n-1}=\left\langle\bar{z}\right\rangle_{2^n-1},0\leq z\leq 2^n-1. \tag{6}$$

Property 3: k-bit binary number R modulo 2^n-1, where $n<k\leq 2n$ is equal to $\langle R_1+R_2\rangle_{2^n-1}$, $R_1=R_{n-1:0}$, and $R_2=\left(||_0^{2n-k}\right)||R_{k-1:n}$.

Proof: Because $n<k\leq 2n$, the following statement holds:

$$\begin{aligned}\langle R\rangle_{2^n-1}=\left\langle\left(\|_0^{2n-k}\right)\|R_{k-1:n}+2^n\times R_{n-1:0}\right\rangle_{2^n-1}=\\ \left\langle\left(\|_0^{2n-k}\right)\|R_{k-1:n}+R_{n-1:0}\right\rangle_{2^n-1}\cdots\end{aligned} \tag{7}$$

Property 4: For any integer ρ,$\langle 2^\rho\rangle_{2^n-1}=\left\langle 2^{\langle\rho\rangle_n}\right\rangle_{2^n-1}$.

Proof: Consider $\rho=i\times n+r$ with $r=\langle\rho\rangle_n$. Therefore, the following statement holds:

$$\begin{aligned}\langle 2^\rho\rangle_{2^n-1}=\left\langle 2^{i\times n+r}\right\rangle_{2^n-1}=\left\langle\underbrace{2^n\times\cdots\times 2^n}_{i}\times 2^r\right\rangle_{2^n-1}\\ =\langle 2^r\rangle_{2^n-1}=\left\langle 2^{\langle\rho\rangle_n}\right\rangle_{2^n-1}\cdots\end{aligned} \tag{8}$$

For the three-moduli set $\{2^{n+p},\ 2^n-1,\ 2^{n+1}-1\}$, the weighted number X can be obtained using mixed-radix conversion (MRC) as follows:

$$X=x_1+2^{n+p}\left(v_2+(2^n-1)\,v_3\right)=x_1+2^{n+p}Y, \tag{9}$$

where

$$Y=v_2+(2^n-1)\,v_3. \tag{10}$$

3 | PROPOSED PROGRAMMABLE POWER-OF-TWO SCALER ALGORITHM

By using MRC, the scaled integer S with a programmable scaling factor 2^{λ} for the moduli set $\{2^{n+p}, 2^n-1, 2^{n+1}-1\}$ can be obtained from the following equation:

$$S = \left\lfloor \frac{X}{2^{\lambda}} \right\rfloor = \left\lfloor \frac{x_1}{2^{\lambda}} \right\rfloor + 2^{n+p-\lambda} Y = x_{1,n+p-1:\lambda} + 2^{n+p-\lambda} Y. \tag{11}$$

Because the results of scaling should be directly processed by the arithmetic unit of a given RNS in each moduli channel, it is necessary to generate a scaled number S in residue form. The scaled residue corresponding to the modulo 2^{n+p} channel can be obtained by applying the modulo 2^{n+p} to (11) as follows:

$$s_1 = \langle S \rangle_{2^{n+p}} = \begin{cases} \left\langle Y \,||\, x_{1,n+p-1:\lambda} \right\rangle_{2^{n+p}} & \lambda < n+p, \\ \langle Y \rangle_{2^{n+p}} & \lambda = n+p. \end{cases} \tag{12}$$

From (12), it is clear that the calculation of Y lies at the heart of realizing s_1. The multiplicative inverses required for calculating Y are $\langle m_1^{-1} \rangle_{m_2} = \langle 2^{n-p} \rangle_{m_2}$, $\langle m_1^{-1} \rangle_{m_3} = \langle 2^{n-p+2} \rangle_{m_3}$, and $\langle m_2^{-1} \rangle_{m_3} = -2$, where $m_1 = 2^{n+p}$, $m_2 = 2^n - 1$, and $m_3 = 2^{n+1} - 1$.

Proof: According to the definition of the multiplicative inverse, it is easy to derive the following results:

- $\left\langle \langle 2^{n-p} \rangle_{m_2} \times 2^{n+p} \right\rangle_{m_2} = \left\langle 2^{2n} \right\rangle_{2^n-1} = 1.$
- $\left\langle \left\langle 2^{n-p+2} \right\rangle_{m_3} \times 2^{n+p} \right\rangle_{m_3} = \left\langle 2^{2n+2} \right\rangle_{2^{n+1}-1} = 1.$
- $\langle -2 \times (2^n - 1) \rangle_{m_3} = \left\langle -2^{n+1} + 2 \right\rangle_{2^{n+1}-1}$
  $= \left\langle -2^{n+1} + 2 + 2^{n+1} - 1 \right\rangle_{2^{n+1}-1} = 1.$

By substituting the values of $\langle m_1^{-1} \rangle_{m_2}$, $\langle m_1^{-1} \rangle_{m_3}$ and $\langle m_2^{-1} \rangle_{m_3}$ into (3), v_2 and v_3, can be simplified using the following expressions:

$$\begin{aligned} v_2 &= \left\langle 2^{n-p} \left(x_2 - x_1 \right) \right\rangle_{2^n-1} = \\ &\left\langle CLS\left(x_2, n-p \right) - x_1 \,||\, \left(||_0^{n-p} \right) \right\rangle_{2^n-1}, \end{aligned} \tag{13}$$

$$v_3 = \left\langle 2^{n-p+3} x_1 - 2^{n-p+3} x_3 + 2 v_2 \right\rangle_{2^{n+1}-1}. \tag{14}$$

The derivation of Y considering the operands v_2 and v_3 can be defined as follows:

$$\begin{aligned} Y &= \langle v_2 + (2^n - 1) v_3 \rangle_{2^{2n+1}} = \langle v_3 \,||\, v_2 - v_3 \rangle_{2^{2n+1}} \\ &= \left\langle Y_x + Y_y + 1 \right\rangle_{2^{2n+1}}, \end{aligned} \tag{15}$$

where $Y_x = v_3 \,||\, v_2$ and $Y_y = \left(||_1^n \right) \,||\, \overline{v}_3$.

For the modulo $2^n - 1$ and $2^{n+1} - 1$ channels, s_i is given by

$$s_i = \left\langle \left(x_i - \tilde{x}_1 \right) b_i \right\rangle_{m_i}, \tag{16}$$

where $\tilde{x}_1 = \langle x_1 \rangle_{2^{\lambda}}$ and $b_i = \left\langle 2^{-\lambda} \right\rangle_{m_i}$. The scaled residues of the modulo m_2 and m_3 channels can be simplified according to (17), (18), (19), and (20).

For the modulo m_2 channel, Properties 1 to 4 and the equality $\left\langle \left(||_1^n \right) \right\rangle_{2^n-1} = 0$ are considered in the derivation of the scaled residue s_2 according to the value of λ as follows:

$$s_2 = \begin{cases} \left\langle \begin{array}{l} CLS\left(x_2, \langle 2n - \lambda \rangle_n \right) \\ + \overline{x}_{1,\lambda-1:0} \,||\, \left(||_1^{n-\lambda} \right) + \left(||_1^n \right) \end{array} \right\rangle_{2^n-1} & \lambda \le n, \\ \left\langle \begin{array}{l} CLS\left(x_2, \langle 2n - \lambda \rangle_n \right) \\ + \overline{x}_{1,\lambda-1:\lambda-n} + \overline{x}_{1,\lambda-n-1:0} \,||\, \left(||_1^{2n-\lambda} \right) \end{array} \right\rangle_{2^n-1} & \lambda > n. \end{cases} \tag{17}$$

It should be noted that for $p = 0$, the second condition does not appear as $\lambda \le n$. Therefore, the formulation of s_2 for $p = 0$ can be simplified as

$$s_2 = \left\langle CLS\left(x_2, n - \lambda \right) + \overline{x}_{1,\lambda-1:0} \,||\, \left(||_1^{n-\lambda} \right) \right\rangle_{2^n-1}. \tag{18}$$

For the modulo m_3 channel, the scaled residue s_3 considering $\left\langle \left(||_1^{n+1} \right) \right\rangle_{2^{n+1}-1} = 0$ and Properties 1 to 4 can be derived using the following detailed formulations according to the value of λ:

$$s_3 = \begin{cases} \left\langle \begin{array}{l} CLS\left(x_3, \langle 2n - \lambda + 2 \rangle_{n+1} \right) \\ + \overline{x}_{1,\lambda-1:0} \,||\, \left(||_1^{n-\lambda+1} \right) + \left(||_1^{n+1} \right) \end{array} \right\rangle_{2^{n+1}-1} & \lambda \le n+1, \\ \left\langle \begin{array}{l} CLS\left(x_3, \langle 2n - \lambda + 2 \rangle_{n+1} \right) \\ + \overline{x}_{1,\lambda-1:\lambda-n-1} \\ + \overline{x}_{1,\lambda-n-2:0} \,||\, \left(||_1^{2n-\lambda+2} \right) \end{array} \right\rangle_{2^{n+1}-1} & \lambda > n+1. \end{cases} \tag{19}$$

It is notable that for $p \in \{0, 1\}$, the second condition is not satisfied as $\lambda \le n + 1$. Therefore, the formulation of s_3 for $p \in \{0, 1\}$ can be shortened to

$$s_3 = \left\langle CLS\left(x_3, n - \lambda + 1 \right) + \overline{x}_{1,\lambda-1:0} \,||\, \left(||_1^{n-\lambda+1} \right) \right\rangle_{2^{n+1}-1}. \tag{20}$$

The concept behind the consideration of the dummy terms $\left\langle \left(||_1^n \right) \right\rangle_{2^n-1}$ and $\left\langle \left(||_1^{n+1} \right) \right\rangle_{2^{n+1}-1}$ in the first conditions of (17) and (19) will be discussed in the next section.

Numerical Example: Consider the moduli set $\{64, 15, 31\}$ with $n = 4$ and $p = 2$. The goal is to achieve power-of-two

scaling of the weighted number $X = (23456)_{10}$ with a residue representation (32, 11, 20) using different scaling factors of 2^1, 2^2, 2^4, and 2^6. Based on these scaling factors, the scaled values of X are equal to $S = 11728$, 5864, 1466, and 366, respectively. The numerical computation of s_1, s_2, and s_3 for these scaling factors is outlined in Table 1.

4 | VLSI ARCHITECTURE OF THE PROPOSED SCALER

The proposed programmable RNS scaler for the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ consists of three independent blocks that perform modular scaling on each channel in parallel. The detailed implementation of each block will be discussed in the following sections. Three types of shifters are utilized for the realization of these blocks. The type-I shifter performs logical right shifting, where the empty bit positions in its most significant output bits are filled with zeroes. The type-II shifter performs circular left shifting and the type-III shifter performs a logical left shift on the one's complement of its input data. The empty least significant bits in the output of the type-III shifter are padded with ones.

TABLE 1 Computation of scaled residues for different scaling factors of 2^1, 2^2, 2^4, and 2^6 based on numerical examples

λ	i	s_i
1	1	$\langle Y \,\|\|\, x_{1,5:1}\rangle_{2^6} = \langle 101101110 \,\|\|\, 10000\rangle_{2^6}$ $= (10000)_2 = (16)_{10}$
	2	$\langle CLS(x_2, 3) + \bar{x}_{1,0} \,\|\|\, (\|\|_1^3) + (\|\|_1^4)\rangle_{2^4-1}$ $= \langle 1101 + 1111 + 1111\rangle_{2^4-1} = (13)_{10}$
	3	$\langle CLS(x_3, 4) + \bar{x}_{1,0} \,\|\|\, (\|\|_1^4) + (\|\|_1^5)\rangle_{2^5-1}$ $= \langle 01010 + 11111 + 11111\rangle_{2^5-1} = (10)_{10}$
2	1	$\langle Y \,\|\|\, x_{1,5:2}\rangle_{2^6} = \langle 101101110 \,\|\|\, 1000\rangle_{2^6}$ $= (101000)_2 = (40)_{10}$
	2	$\langle CLS(x_2, 2) + \bar{x}_{1,1:0} \,\|\|\, (\|\|_1^2) + (\|\|_1^4)\rangle_{2^4-1} =$ $\langle 1110 + 1111 + 1111\rangle_{2^4-1} = (14)_{10}$
	3	$\langle CLS(x_3, 3) + \bar{x}_{1,1:0} \,\|\|\, (\|\|_1^3) + (\|\|_1^5)\rangle_{2^5-1}$ $= \langle 00101 + 11111 + 11111\rangle_{2^5-1} = (5)_{10}$
4	1	$\langle Y \,\|\|\, x_{1,5:4}\rangle_{2^6} = \langle 10110111010\rangle_{2^6}$ $= (111010)_2 = (58)_{10}$
	2	$\langle x_2 + \bar{x}_{1,3:0} + (\|\|_1^4)\rangle_{2^4-1}$ $= \langle 1011 + 1111 + 1111\rangle_{2^4-1} = (11)_{10}$
	3	$\langle CLS(x_3, 1) + \bar{x}_{1,1:0} \,\|\|\, (\|\|_1^3) + (\|\|_1^5)\rangle_{2^5-1}$ $= \langle 01001 + 11111 + 11111\rangle_{2^5-1} = (9)_{10}$
6	1	$\langle Y_{8:1}\rangle_{2^6} = \langle 10110111\rangle_{2^6}$ $= (110111)_2 = (47)_{10}$
	2	$\langle CLS(x_2, 2) + \bar{x}_{1,5:2} + \bar{x}_{1,1:0} \,\|\|\, (\|\|_1^2)\rangle_{2^4-1}$ $= \langle 1110 + 0111 + 1111\rangle_{2^4-1} = (6)_{10}$
	3	$\langle CLS(x_3, 4) + \bar{x}_{1,5:1} + \bar{x}_{1,0} (\|\|_1^4)\rangle_{2^5-1}$ $= \langle 01010 + 01111 + 11111\rangle_{2^5-1} = (25)_{10}$

4.1 | s_1 generator

From (12), the generation of s_1 can be obtained from an $(n + p)$-bit slice from position λ to position $n + p + \lambda - 1$ in the vector that is generated by the concatenation of Y and x_1. The implementation of s_1 for the proposed moduli set is realized using a $(3n + p + 1)$-bit type-I shifter (Shifter 1) with λ as the shift value and the concatenation of Y and x_1 as the data input. The $(n + p)$-least significant bits of the Shifter 1 output yield the expected value. As stated previously, the well-organized realization of Y plays a crucial role in determining how well the scaler block operates in an RNS. For calculating the Y signal, the VLSI architecture is defined as follows. To implement the v_2 architecture, two intermediate signals S and C can be considered. From (13), for $p = 0$, S and C are equal to x_2 and $\bar{x}_1$, respectively. For $p > 0$, S and C can be generated from an n-bit carry save adder (CSA) with end-around carry (EAC) [32]. The inputs of the CSA are $v_{2,a} = CLS(x_2, n - p)$, $v_{2,b} = \bar{x}_{1,n+p-1:p}$, and $v_{2,c} = \bar{x}_{1,p-1:0} \,||\, (||_1^{n-p})$. The addition of S and C in the modulo $2^n - 1$ should be performed to realize the value of v_2. Figures 1A and 1B illustrate the hardware architectures of v_2 for $p = 0$ and $p > 0$, respectively.

The substantial computational delay of v_3 in its serial form with v_2 can be mitigated using a multiplexer-based approach, which parallelizes v_2 and v_3 computations. To this end, we removed the imposed delay of v_2 from the v_3 computational critical path as follows. Considering (14), $\langle 2v_2\rangle_{2^{n+1}-1}$ can be described for two different cases. For $S + C \neq 2^n - 1$, $\langle 2v_2\rangle_{2^{n+1}-1}$ is equal to $\langle S \,||\, 0 + C \,||\, c_o\rangle_{2^{n+1}-1}$ ($c_o = 1$ if $S + C > 2^n$-1). Otherwise, the value of $\langle 2v_2\rangle_{2^{n+1}-1}$ is equal to zero and can be detailed as $\langle S \,||\, 0 + C \,||\, \xi\rangle_{2^{n+1}-1}$. The correction bit ξ is equal to one when S and C are bit-wise complements ($S + C = 2^n - 1$). ξ is generated using

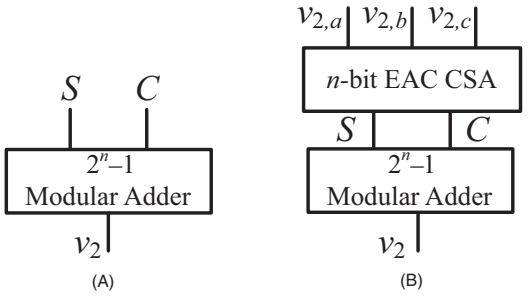

FIGURE 1 (A) Schematic of v_2 for $p = 0$. (B) Schematic of v_2 for $p > 0$

the available $P_{n-1:0}$ and $G_{n-1:0}$ in a parallel prefix structure as $\xi = P_{n-1:0}\overline{G}_{n-1:0}$ [33]. When considering these cases, because c_o and ξ cannot both receive a value of one simultaneously, $\langle 2v_2\rangle_{2^{n+1}-1}$ can be expressed as $\langle S \,||\, 0 + C \,||\, \tau\rangle_{2^{n+1}-1}$, where $\tau = c_o \vee \xi$. As a result, v_2, $v_{3,\tau=0}$ and $v_{3,\tau=1}$ are computed concurrently and the expected result of v_3 based on τ is selected from $v_{3,\tau=0}$ and $v_{3,\tau=1}$. For $\tau = 0$ and $\tau = 1$, v_3 can be expressed as follows:

$$v_{3,\tau=0} = \begin{cases} \langle I_0 + x'_3 + J_0 + J_1\rangle_{2^{n+1}-1} & p \le 2, \\ \langle I_0 + I_1 + x'_3 + J_0 + J_1\rangle_{2^{n+1}-1} & p > 2, \end{cases} \quad (21)$$

$$v_{3,\tau=1} = \begin{cases} \langle I_0 + x'_3 + J_0 + J_2\rangle_{2^{n+1}-1} & p \le 2, \\ \langle I_0 + I_1 + x'_3 + J_0 + J_2\rangle_{2^{n+1}-1} & p > 2, \end{cases} \quad (22)$$

where I_0 is equal to $x_{1,n-2:0} \,||\, 0 \,||\, x_{1,n-1}$ and $x_{1,n+p-2:p-1} \,||\, x_{1,n+p-1}$ for $p = 0$ and $p > 0$, respectively. I_1 can be represented as $x_{1,p-3:0} \,||\, \left(||_0^{n-p+3}\right)$. x'_3 is also obtained by inverting each bit of x_3, followed by a $\langle n - p + 3\rangle_{n+1}$-bit circular left shift. For $p < 2$ and $p \ge 2$, J_0 can be represented as $S \,||\, 0$ and $S \,||\, x_{1,p-2}$, respectively. J_1 and J_2 are equal to $C \,||\, 0$ and $C \,||\, 1$, respectively. Because the three (four) first terms of v_3 when $p \le 2$ ($p > 2$) are similar for both values of τ and the difference in the last term is only in one bit, relative to the achieved speedup, the hardware redundancy of this parallelism is insignificant. The n most significant bits of J_1 and J_2 are absolutely identical terms and the difference is only in the least significant bit. Therefore, one n-bit EAC CSA with C, $z_{n:1}$ and $w_{n:1}$ as input operands can be utilized to save additional hardware elements instead of using J_1 and J_2 in the reduction process. The remaining terms f_0, f_1, f_2, and f_3 are also yielded by AND, OR, XOR, and XNOR operators because the least significant bits of z, and w should be incremented by constant values of zero and one, respectively. The hardware architecture of v_3 is depicted in Figures 2A and 2B for $p \le 2$ and $p > 2$, respectively. In Figure 2, z and z' are the carry vectors, and w and w' are the summation outputs of the EAC CSA. Furthermore, the expressions of Q_0, Q_1, Q_2, and Q_3 are defined as follows:

$$\begin{aligned} Q_0 &= w'_{n-1:0} \,||\, f_0, \\ Q_1 &= z'_{n-1:1} \,||\, f_1 \,||\, z'_0, \\ Q_2 &= w'_{n-1:0} \,||\, f_2, \\ Q_3 &= z'_{n-1:1} \,||\, f_3 \,||\, z'_0, \end{aligned} \quad (23)$$

where $f_0 = z_0 \oplus w_0$, $f_1 = z_0 \wedge w_0$, $f_2 = z_0 \odot w_0$, and $f_3 = z_0 \vee w_0$.

According to (15), Y can be obtained from two different adders. The first is an adder with $\overline{v}_3$ and $v_{3,0} \,||\, v_2$ as input signals and one as a carry input. The second is an adder with $\left(||_1^n\right)$ and $v_{3,n:1}$ as input vectors and a carry input driven by the first adder's carry output. Based on its constant inputs, the structure of the second adder can be modified to achieve enhanced performance. This approach is appropriate for a parallel prefix adder structure, where the carry input of a binary adder can be applied in the final stage of the adder with insignificant latency. For another type of realization using ripple carry adders, a multiplexing approach is more appropriate. In this paper, we consider the parallel-prefix-adder-based realization. An example of the realization of Y using a parallel prefix adder for $n = 4$ is presented in Figure 3. It is clear that this approach imposes less latency on the adder architecture compared to the previous method presented in [34]. The overall architecture of the Y generator is also presented in Figure 4.

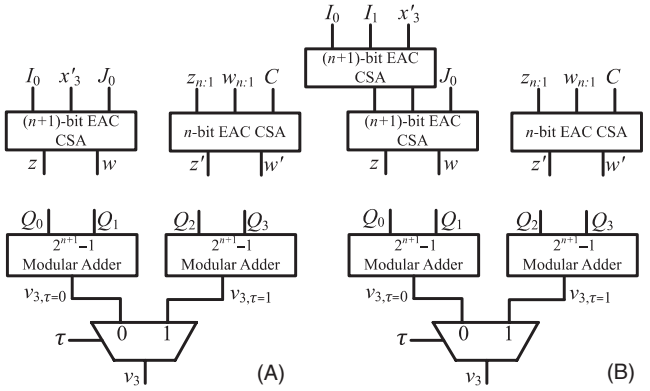

FIGURE 2 (A) Schematic of v_3 for $p \le 2$. (B) Schematic of v_3 for $p > 2$

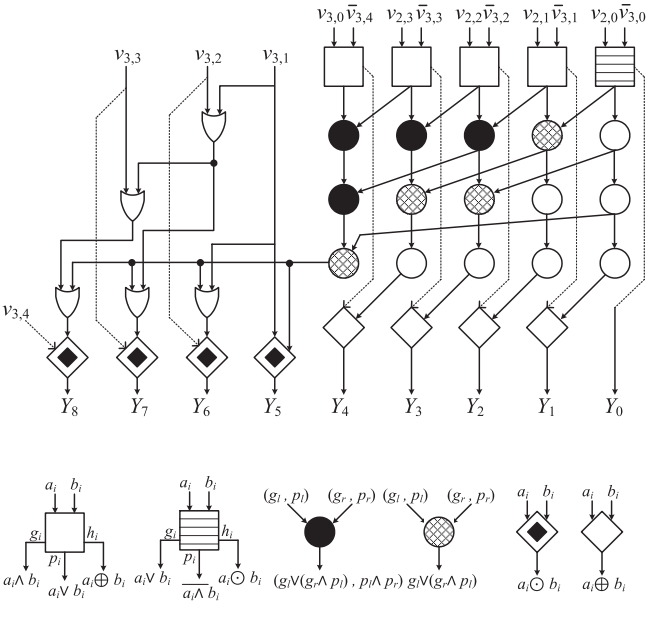

FIGURE 3 Realization of Y with a parallel prefix adder for $n = 4$

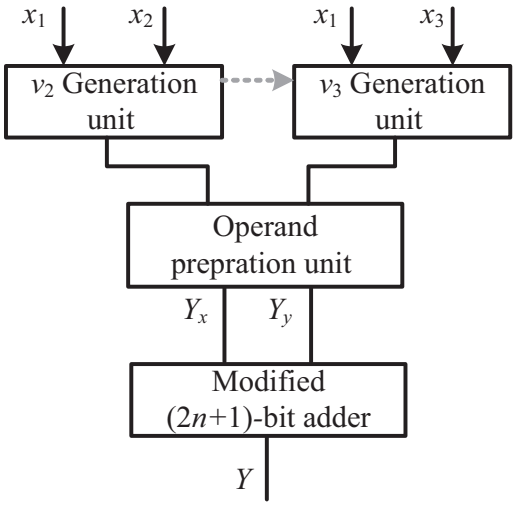

FIGURE 4 Overall architecture of the Y generator

4.2 | s_2 generator

According to (18), for the special case of $p = 0$, the first and second terms are yielded by an n-bit type-II shifter (Shifter 2) and type-III shifter (Shifter 3), respectively, followed by a modulo $2^n - 1$ adder to produce s_2. The shift value for both Shifter 2 and Shifter 3 is $n - \lambda$. From (17), for $p > 0$, the first term in s_2 is equal for both conditions and the differences lie in the second and third terms. The first term is realized through a $\langle 2n - \lambda \rangle_n$-bit circular left shift of x_2 by an n-bit type-II shifter (Shifter 4). For the second and third terms, it is worth noting that based on λ, which is determined at run time, two distinct hardware realizations increase the hardware cost of s_2. We will demonstrate that the hardware implementation of the second and third terms can be realized independently from λ by using $\left\langle \left(||_1^n \right) \right\rangle_{2^n-1}$ as a dummy term in the first condition of (17). By adopting this approach, the second and third terms of s_2 can be produced simultaneously through a $2n - \lambda$-bit left shift of the one's complement of x_1 and padding with ones by using a $2n$-bit type-III shifter (Shifter 5). The n most significant bits and n least significant bits of the output of Shifter 5 yield the second and third terms, respectively. Finally, s_2 is derived by an EAC CSA followed by a modulo $2^n - 1$ adder.

4.3 | s_3 generator

For $p \in \{0, 1\}$, s_3 consists of only two terms (see (20)). The first is produced by an $(n + 1)$-bit type-II shifter (Shifter 6) and the second is produced by an $(n + 1)$-bit type-III shifter (Shifter 7). Next, the outputs of Shifters 6 and 7 are fed into a modulo $2^{n+1} - 1$ adder to generate the expected value s_3. For Shifters 6 and 7, the shift value is $n - \lambda + 1$. According to (19), for $p > 1$, similar to s_2, the first term in s_3 has the same value for both conditions of the scaling factor. The first term in s_3 is a circular left shift of x_3 with a value of $\langle 2n - \lambda + 2 \rangle_{n+1}$ produced by an $(n + 1)$-bit type-II shifter (Shifter 8). The mechanism for generating the second and third terms of s_3 is analogous to that applied to the second and third terms of s_2. Therefore, the realization of the second and third terms of s_3 can be accomplished independently of λ to avoid hardware redundancy. To this end, a $(2n + 2)$-bit type-III shifter (Shifter 9) is utilized to produce the second and third terms of s_3 concurrently. This shifter generates a $(2n + 2)$-bit output with $2n - \lambda + 2$ as the shift value. The second and third terms of s_3 are yielded by the $n + 1$ most significant bits and $n + 1$ least significant bits of the output of Shifter 9, respectively. These three terms produce s_3 with an EAC CSA, which is followed by a modulo $2^{n+1} - 1$ adder.

The overall architecture of the proposed programmable scaler for the VLSI implementation of the proposed algorithm with different p-values is presented in Figure 5.

5 | PERFORMANCE EVALUATIONS AND COMPARISONS

5.1 | Theoretical evaluation

In this study, the unit-gate (U-G) model [35] was adopted to conduct a technology-independent assessment of the area and time requirements of the evaluated designs. According to the U-G model, two-input monotonic gates, such as AND and OR gates, have one unit of area and delay. In this model, both the area and delay of two-input XOR, XNOR, and 2:1 multiplexer gates are considered as two units. This model also considers an area of three units and delay of two units for half adders, and an area of seven units and delay of four units for full adders. The area and delay of inverters are not considered in the U-G model. Furthermore, it is assumed that multi-input logic gates are constructed from two-input basic logic gates in an efficient manner. The binary adder, modulo $2^n - 1$ adder, modulo $2^n + 1$ adder, and different types of CSAs and shifters are the main requirements for the architectures of the reviewed designs. In this theoretical evaluation, the adders adopted in [36,37] were considered. Binary adders are considered as $1.5n\lceil \log_2 n \rceil + 5n$ and $2\lceil \log_2 n \rceil + 3$ units of area and delay, respectively [32]. The modulo $2^n - 1$ adders are considered to have an area of $3n\lceil \log_2 n \rceil + 5n$ units and imposes a delay of $2\lceil \log_2 n \rceil + 3$ units. The area complexity and delay of modulo $2^n + 1$ diminished-one adders are $4.5n\lceil \log_2 n \rceil + 0.5n + 6$ units and $2\lceil \log_2 n \rceil + 3$ units, respectively [36]. For shifters and CSAs, the area and delay differ based on the structures of their components.

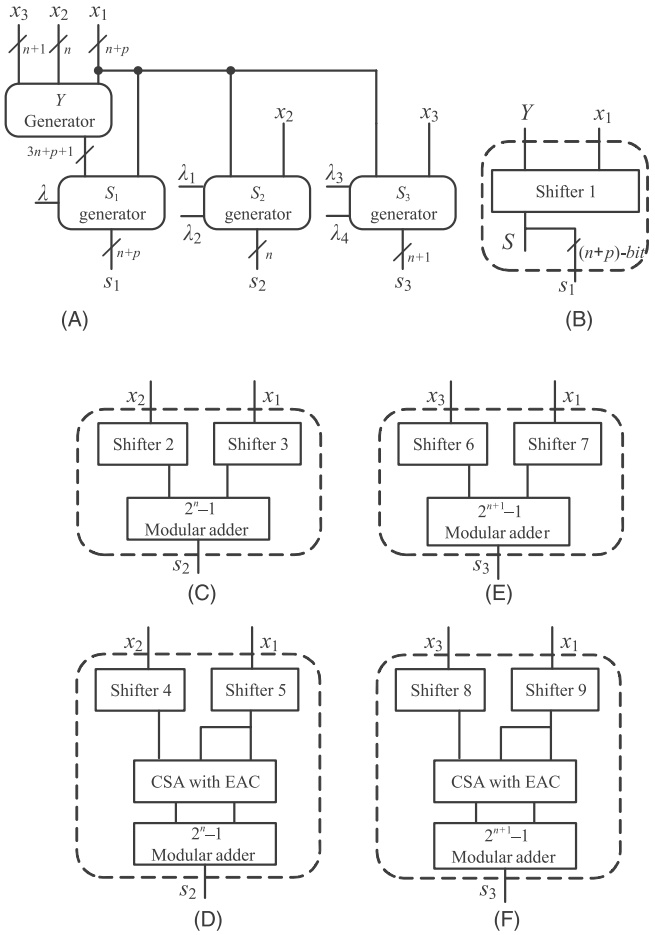

FIGURE 5 Block diagram of the proposed programmable scaler. (A) Overall architecture, (B) s_1 generator, (C) s_2 generator for $p = 0$, (D) s_2 generator for $p > 0$, (E) s_3 generator for $p \in \{0, 1\}$, and (F) s_3 generator for $p > 1$. The λ_i values are determined according to the s_2 and s_3 generator subsections

In additional to the proposed programmable scaler, our comparison pool included the following works.

1. Hybrid programmable power-of-two scaler: To the best of our knowledge, no specific programmable scaler for the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ has been developed. An appropriate method for evaluating the proposed programmable scaler is to consider a hybrid approach as a primary reference. This type of evaluation is a conventional method that has been adopted in numerous studies, such as [5] and [19]. In general, the hybrid RNS scaler performs the RNS scaling algorithm in three steps. In the first step, by using a reverse converter (RC), residues are decoded into a weighted value. Next, the weighted value is scaled by a binary scaler (BS). Finally, a forward converter (FC) unit produces the expected scaled residues [5]. For the sake of fair comparison, the calculation of weighted number X from its residues (as first step of programmable hybrid scaler for the moduli set $\{2^{n+p}, 2^n-1, 2^{n+1} - 1\}$) in the hybrid model was performed by concatenating Y and x_1 based on (9).
2. [19]: Among state-of-the-art RNS scalers, a design that offers programmability can be found in [19]. This scaler provides a fully combinational architecture for an arbitrary moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. The evaluation of [19] in this paper is considered for $\left\{2^{n+\lfloor p/3 \rfloor} - 1, 2^{n+\lfloor p/3 \rfloor}, 2^{n+\lfloor p/3 \rfloor} + 1\right\}$. It is worth noting that the proposed moduli set and the balanced three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ are not in the same class. One major advantage of the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ is that it has more efficient modular arithmetic operations, especially multiplication, when compared to the balanced three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. The advantage of the arithmetic unit of the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ over the balanced three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ stems from the existence of the modulus $2^n + 1$ in the latter moduli set. A recent study [38] demonstrated the superiority of the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ in terms of its arithmetic unit. However, based on the conjugation of $2^n - 1$ and $2^n + 1$ in $\{2^n - 1, 2^n, 2^n + 1\}$, this moduli set has much simpler inter-modulo operations, such as RC, compared to the class of the proposed moduli set. In applications for which the arithmetic unit has a significant impact on system performance, employing the $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$ RNS is more affordable.

Because the circuit area and delay of the proposed programmable scaler are strongly dependent on the p-value, various theoretical complexity descriptions can be derived based on the U-G model. To avoid ambiguity, the maximum p-value ('n') was considered as the worst-case scenario for area and delay complexity assessment. According to the U-G model, the area and delay of the considered designs can be estimated as follows.

Based on the assumptions above, the areas and delays of the RC, BS, FC blocks of the hybrid programmable power-of-two scaler are $(3.5n-1)\lceil \log_2 n \rceil + (7.5n+7.5)\lceil \log_2 (n+1) \rceil + 55n + 29$ and $4\lceil \log_2 (n+1) \rceil + 22$, $(8n+2)\lceil \log_2 (4n+1) \rceil - 2^{\lceil \log_2 (4n+1) \rceil} + 1$ and $2\lceil \log_2 (4n+1) \rceil$, and $(3n)\lceil \log_2 n \rceil + (3n+3)\lceil \log_2 (n+1) \rceil + 41n + 11$ and $2\lceil \log_2 n \rceil + 15$ units, respectively.

According to the scaling algorithm in [19], the scaler architecture contains three parallel channels. By considering $\left\{2^{n+\lfloor p/3 \rfloor} - 1, 2^{n+\lfloor p/3 \rfloor}, 2^{n+\lfloor p/3 \rfloor} + 1\right\}$ as the moduli set for the maximum p-value ('n'), the U-G areas of the first, second, and third channels when considering Figure 1 in [19] can be estimated as follows. In the first channel of the scaler architecture, the area of Shifters 1 and 2, and the modulo $2^{n+\lfloor n/3 \rfloor} - 1$ adder are $2(n + \lfloor n/3 \rfloor)\lceil \log_2 (n + \lfloor n/3 \rfloor) \rceil$, $2(n + \lfloor n/3 \rfloor)\lceil \log_2 (n + \lfloor n/3 \rfloor) \rceil - 2^{\lceil \log_2 (n + \lfloor n/3 \rfloor) \rceil} + n + \lfloor n/3 \rfloor + 2$, and $3(n + \lfloor n/3 \rfloor)\lceil \log_2 (n + \lfloor n/3 \rfloor) \rceil + 5n + 5\lfloor n/3 \rfloor$ units, respectively. In the second channel, the areas of the CSA,

modulo $2^{2(n+\lfloor n/3 \rfloor)}-1$ adder, and Shifter 3 are $15n+15\lfloor n/3 \rfloor$, $6(n+\lfloor n/3 \rfloor)\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 18n + 18\lfloor n/3 \rfloor + 1$, and $6(n+\lfloor n/3 \rfloor)\lceil \log_2(3n+3\lfloor n/3 \rfloor)\rceil - 2^{\lceil \log_2(3n+3\lfloor n/3 \rfloor)\rceil} + 1$ units, respectively. In the third channel, the areas of Shifter 4, the CSA, and modulo $2^{n+\lfloor n/3 \rfloor}+1$ diminished-one adder are $2(n+\lfloor n/3 \rfloor)\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 3n + 3\lfloor n/3 \rfloor$, $3n+3\lfloor n/3 \rfloor + 7$, and $4.5(n+\lfloor n/3 \rfloor)\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 0.5n + 0.5\lfloor n/3 \rfloor + 6$ units, respectively. The imposed latencies of the first, second, and third channels are $4\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 4$, $2\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 2\lceil \log_2 3(n+\lfloor n/3 \rfloor)\rceil + 10$, and $4\lceil \log_2(n+\lfloor n/3 \rfloor)\rceil + 13$ units of delay, respectively.

The proposed RNS scaler consists of three parallel channels. For the maximum p-value ('n'), performance estimation was conducted as follows. According to Figure 5, the area and delay imposed on the first channel by generating Y are $(3.5n-1)\lceil \log_2 n \rceil + (7.5n+7.5)\lceil \log_2(n+1)\rceil + 55n + 29$ and $4\lceil \log_2(n+1)\rceil + 22$ units, respectively. The area and delay of Shifter 1 are $(8n+2)\lceil \log_2(4n+1)\rceil - 2^{\lceil \log_2(4n+1)\rceil} + 1$ and $2\lceil \log_2(4n+1)\rceil$ units, respectively. In the second channel, Shifter 4 has $2n\lceil \log_2 n \rceil$ units of area and $2\lceil \log_2 n \rceil$ units of delay. Shifter 5 requires $4n\lceil \log_2 2n \rceil + 2n - 2^{\lceil \log_2 2n \rceil} + 1$ units of area and $2\lceil \log_2 2n \rceil + 1$ units of delay to compute the second and third terms of s_3. The EAC CSA requires $7n$ units of area and 4 units of delay. The area and delay of the modulo 2^n-1 adder are considered to be $3n\lceil \log_2 n \rceil + 5n$ and $2\lceil \log_2 n \rceil + 3$ units, respectively. The structure of the third channel is similar to that of the second channel. In the third channel, Shifter 8 has $2(n+1)\lceil \log_2(n+1)\rceil$ units of area and $2\lceil \log_2(n+1)\rceil$ units of delay. Shifter 9 requires $(4n+4)\lceil \log_2(2n+2)\rceil + 2n - 2^{\lceil \log_2(2n+2)\rceil} + 3$ units of area and $2\lceil \log_2(2n+2)\rceil + 1$ units of delay to compute the second and third terms of s_3. The EAC CSA requires $7n+7$ units of area and 4 units of delay. The area and delay of the modulo $2^{n+1}-1$ adder are considered to be $(3n+3)\lceil \log_2(n+1)\rceil + 5n + 5$ and $2\lceil \log_2(n+1)\rceil + 3$ units, respectively. In summary, the delay of the proposed scaler is equal to the delay of the first channel and the area of the scalar is equal to the sum of the areas of the channels. The total areas and imposed delays of the proposed scaler, hybrid scaler, and the design in [19] for $p = n$ are listed in Table 2.

TABLE 2 Area and delay comparisons between programmable scalers for $p = n$ based on the U-G model

Design	Area
Proposed	$(8.5n-1)\lceil \log_2 n \rceil + (12.5n+12.5)\lceil \log_2(n+1)\rceil + (8n+2)\lceil \log_2(4n+1)\rceil - 2^{\lceil \log_2(4n+1)\rceil} + (4n)\lceil \log_2 2n \rceil - 2^{\lceil \log_2 2n \rceil} + (4n+4)\lceil \log_2(2n+2)\rceil - 2^{\lceil \log_2(2n+2)\rceil} + 83n + 46$
Hybrid	$(6.5n-1)\lceil \log_2 n \rceil + (10.5n+10.5)\lceil \log_2 n+1\rceil + (8n+2)\lceil \log_2(4n+1)\rceil - 2^{\lceil \log_2(4n+1)\rceil} + 96n + 41$
[19]	$19.5(n+\lfloor n/3 \rfloor)\lceil \log_2 n+\lfloor n/3 \rfloor\rceil + 6(n+\lfloor n/3 \rfloor)\lceil \log_2 3n+3\lfloor n/3 \rfloor\rceil - 2^{\lceil \log_2 n+\lfloor n/3 \rfloor\rceil} + 45.5n - 2^{\lceil \log_2 3n+3\lfloor n/3 \rfloor\rceil} + 45.5\lfloor n/3 \rfloor + 17$
	Delay
Proposed	$4\lceil \log_2(n+1)\rceil + 2\lceil \log_2(4n+1)\rceil + 22$
Hybrid	$4\lceil \log_2(n+1)\rceil + 2\lceil \log_2 n \rceil + 2\lceil \log_2(4n+1)\rceil + 37$
[19]	$4\lceil \log_2 n+\lfloor n/3 \rfloor\rceil + 13\,(n<5)$ $2\lceil \log_2 n+\lfloor n/3 \rfloor\rceil + 2\lceil \log_2 3(n+\lfloor n/3 \rfloor)\rceil + 10\,(n\geq 5)$

5.2 | Experimental evaluation

For a more realistic evaluation of the moduli set $\{2^{n+p}, 2^n-1, 2^{n+1}-1\}$ with upper and lower bounds for the p-value, hardware descriptions of the proposed RNS scaler, hybrid RNS scaler, and scaler from [19] were generated using Verilog HDL and a well-known arithmetic unit that was presented in [39]. Functional verification was conducted using ModelSim for $n = 4$ ($p = 0$, 4), 8 ($p = 0$, 8), and 16 ($p = 0$, 16). Verilog codes were synthesized and mapped to the TSMC 130 *nm* standard cell library using the Synopsys Design Compiler with 0.01 *ns* time intervals and delay constraints in the range of 0.8-8 *ns*. The experimental results for the ASIC implementations of the scalers are listed in Tables 3 and 4. It should be noted that the dynamic ranges of the proposed moduli set $\{2^{n+p}, 2^n-1, 2^{n+1}-1\}$ and the moduli set $\{2^{n+\lfloor p/3 \rfloor}-1, 2^{n+\lfloor p/3 \rfloor}, 2^{n+\lfloor p/3 \rfloor}+1\}$ are not equal based on the different types of moduli. Because the dynamic range of a moduli set has a significant impact on its efficiency, (24) was utilized to normalize the performance metrics for any given moduli set in comparison to the proposed moduli set.

$$\Delta_{\mathrm{F,\,Normalized}} = \Delta_{\mathrm{F}} \times \frac{\mathrm{DR_{Proposed}}}{\mathrm{DR_F}} \tag{24}$$

In (24), Δ, F, and DR refer to the performance metric, specific moduli set, and dynamic range, respectively. The performance metric for the design from [19] is presented in a normalized form according to (24) in Tables 3 and 4.

As shown in Table 3, the proposed scaler outperforms its hybrid counterpart in terms of area, delay, A×D, and A×D^2 for $n = 4$, 8, and 16 when considering the upper and lower bounds of the p-value. These improvements can be easily explained because the critical path and hardware overhead of the proposed scaler are considerably smaller than those of the hybrid structure. Furthermore, the proposed scaler

TABLE 3 Performance metrics for synthesized programmable scalers

		Proposed		Hybrid		[19]*	
n	*p*	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)
4	0	5875.38	1.43	10 817.60	2.21	9447.30	1.71
	4	10 264.84	1.71	13 352.36	2.61	30 935.38	3.78
8	0	15 624.27	1.71	21 380	2.63	27 172.13	2.28
	8	20 262.09	2.15	31 970.40	3.1	155 527	10.18
16	0	37 091.53	2.06	50881.13	3.12	66 426.25	2.7
	16	52 793.2	2.43	70 233.29	3.59	198 688	6.38
		A×D	$A\times D^2$	A×D	$A\times D^2$	A×D	$A\times D^2$
4	0	8401.80	12 014.58	23 906.9	52 834.25	17 708.83	16 646.29
	4	17 552.87	30 015.42	34 849.66	90 957.61	32 169.7	33 453.27
8	0	26 717.51	45 686.94	56 229.4	147 883.3	31 247.94	35 935.15
	8	43 563.5	93 661.51	99 108.24	307 235.54	199 059	254 775.7
16	0	76 408.55	157 401.6	15 8749.13	495 297.3	89 675.44	121 061.86
	16	128 287.48	311 738.56	252 137.51	905 173.66	318 690.2	509 904.32

*Normalized values with respect to (24).

TABLE 4 Comparison of synthesized programmable scalers

n	*p*	Dynamic power (mW)	Leakage power (pW)	Energy (pJ)
Proposed				
4	0	2.29	185.62	4.13
	4	4.81	299.74	8.22
8	0	6.26	484.98	10.71
	8	9.97	625.15	21.43
16	0	15.5	978.79	31.93
	16	26.59	1408.9	64.61
Hybrid				
4	0	4.56	267.76	10.08
	4	7.82	321.8	20.41
8	0	12.07	551.8759	31.75
	8	19.98	830.56	61.93
16	0	28.11	1221.5	87.70
	16	42.83	1796.2	153.76
[19]*				
4	0	2.78	322.35	2.62
	4	10.06	877.7	10.48
8	0	8.16	698.367	9.39
	8	51.76	4335.37	66.25
16	0	20.80	1801.21	28.08
	16	67.2	6513.53	107.51

*Normalized values with respect to (24).

outperforms the scaler presented in [19] in terms of area, delay, A×D, and $A\times D^2$ when considering normalization.

Table 4 lists the total dynamic power dissipations, leakage powers, and energy consumptions of the proposed and hybrid scalers, as well as the design from [19], for $n = 4$, 8, and 16 with respect to the upper and lower bounds of the p-value. The main reason for the power and energy dissipation improvements of the proposed scaler compared to the hybrid scaler is the smaller number of logic gates in the proposed scaler. The proposed scaler also outperforms the scaler presented in [19] in terms of power and energy dissipation when considering normalization.

6 | CONCLUSIONS

In this paper, we presented a fully combinational programmable RNS scaler for the special moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$. The proposed programmable RNS scaler exploits a parallel structure combined with a hardware sharing technique to achieve high speed and cost efficiency, as well as memoryless VLSI realization. Up to half of the dynamic range for the maximum p-value can be scaled by utilizing our programmable power-of-two scaler. Experimental evaluations based on TSMC 130 *nm* ASIC technology demonstrated that for the moduli set $\{2^{n+p}, 2^n - 1, 2^{n+1} - 1\}$, the proposed programmable scaler performs scaling operations approximately 33.27% faster with an area savings of 28.55% and 43.28% greater power efficiency compared to a hybrid design. Furthermore, experimental evaluations demonstrated the superiority of the proposed programmable scaler compared to the programmable scaler presented in [19] with average improvements of 70.93% in terms of area, 57.91% in terms of delay, and 59.3% in terms of power when considering normalization.

ACKNOWLEDGMENTS

The authors would like to thank Dr. B. Yoberd for his literature contributions.

CONFLICT OF INTEREST

The authors declare no potential conflicts of interest.

ORCID

Keivan Navi https://orcid.org/0000-0002-5586-7766

REFERENCES

1. C. -H. Chang et al., *Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications*, IEEE Circ. Syst. Mag. **15** (2015), no. 4, 26–44.
2. G. L. Bernocchi et al., *Low-power adaptive filter based on RNS components*, in Proc. IEEE Int. Symp. Circuits Syst. (New Orleans, LA, USA), May (2007), pp. 3211–3214.
3. J. Y. S. Low, T. F. Tay, and C.-H. Chang, *A unified {2^n–1, 2^n, 2^n+1} RNS scaler with dual scaling constants*, in Proc. IEEE Asia Pacific Conf. Circuits Syst. (Kaohsiung, Taiwan), Dec. 2012, pp. 296–299.
4. C. -H. Chang and J. Y. S. Low, *Simple, fast, and exact RNS scaler for the three-moduli set {2^n – 1, 2^n, 2^n+1}*, IEEE Trans. Circuits Syst. I **58** (2011), no. 11, 2686–2697.
5. L. Sousa, *2^n RNS scalers for extended 4-moduli sets*, IEEE Trans. Comput. **64** (2015), no. 12, 3322–3334.
6. Z. D. Ulman and M. Czyzak, *Highly parallel, fast scaling of numbers in nonredundant residue arithmetic*, IEEE Trans. Signal Process. **46** (1998), no. 2, 487–496.
7. Z. Ulman, M. Czyzak, and J. Zurada, *Effective RNS scaling algorithm with the Chinese remainder theorem decomposition*, in Proc. IEEE Pacific Rim Conf. Commun. Comput. Signal Process. (Victoria, Canada), May 1993, pp. 528–531.
8. M. Griffin, F. Taylor, and M. Sousa, *New scaling algorithms for the Chinese remainder theorem*, in Proc. Twenty-Second Asilomar Conf. Signals, Syst. Comput. (Pacific Grove, CA, USA), 1988, pp. 375–378.
9. M. Griffin, M. Sousa, and F. Taylor, *Efficient scaling in the residue number system*, in Proc. Int. Conf. Acoustics, Speech, Signal Process. (Glasgow, UK), May 1989, pp. 1075–1078.
10. M. A. P. Shenoy and R. Kumaresan, *A fast and accurate RNS scaling technique for high speed signal processing*, IEEE Trans. Signal Process. **37** (1989), no. 6, 929–937.
11. Y. Kong and B. Phillips, *Fast scaling in the residue number system*, IEEE Trans. VLSI Syst. **17** (2009), no. 3, 443–447.
12. F. Barsi and M. C. Pinotti, *Fast base extension and precise scaling in RNS for look-up table implementations*, IEEE Trans. Signal process. **43** (1995), no. 10, 2427–2430.
13. M. Dasygenis et al,, *A full-adder-based methodology for the design of scaling operation in residue number system*, IEEE Trans. Circuits Syst. I **55** (2008), no. 2, 546–558.
14. S. Ma et al., *A 2^n scaling scheme for signed RNS integers and its VLSI implementation*, Sci. China Inf. Sci. **53** (2010), no. 1, 203–212.
15. A. García and A. Lloris, *A look-up scheme for scaling in the RNS*, IEEE Trans. Comput. **48** (1999), no. 7, 748–751.
16. A. Garcia and A. Lloris, *RNS scaling based on pipelined multipliers for prime moduli*, in Proc. IEEE Workshop Signal Process. Syst. SIPS Design Implementation (Cambridge, MA, USA), Oct. 1998, pp. 459–468.
17. N. Burgess, *Scaling an RNS number using the core function*, in Proc. IEEE Symp. Comput. Arithmetic (Santiago de Compostela, Spain), June 2003, pp. 262–269.
18. T. F. Tay, C.-H. Chang, and J. Y. S. Low, *Efficient VLSI implementation of 2^n scaling of signed integer in RNS {2^n – 1, 2^n, 2^n + 1}*, IEEE Trans. VLSI Syst. **21** (2012), no. 10, 1936–1940.
19. 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} RNS*, IEEE Trans. Circuits Syst. I **59** (2012), no. 12, 2911–2919.
20. A. Hiasat, *Efficient RNS scalers for the extended three-moduli set {2^n–1, 2^{n+p}, 2^n+1}*, IEEE Trans. Comput. **66** (2017), no. 7, 1253–1260.
21. W. Wang et al., *A high-speed residue-to-binary converter for three-moduli (2^k, 2^{k-1}, 2^{k-1} – 1) RNS and a scheme for its VLSI implementation*, IEEE Trans. Circuits Syst. II **47** (2000), no. 12, 1576–1581.
22. P. V. A. Mohan, *RNS-to-binary converter for a new three-moduli set {2^{n+1} – 1, 2^n, 2^n–1}*, IEEE Trans. Circuits Syst. II **54** (2007), no. 9, 775–779.
23. A. Hiasat, *An efficient reverse converter for the three-moduli set (2^{n+1} – 1, 2^n, 2^n –1)*, IEEE Trans. Circuits Syst. II **64** (2016), no. 8, 962–966.
24. M. M. Latha, R. R. Rachh, and P. V. A. Mohan, *RNS-to-binary converters for a three-moduli set {2^{n-1} – 1, 2^n–1, 2^{n+k}}*, IETE J. Edu. **58** (2017), no. 1, 20–28.
25. A. S. Molahosseini et al., *Efficient MRC-based residue to binary converters for the new moduli sets {2^{2n}, 2^n–1, 2^{n+1} – 1} and {2^{2n}, 2^n–1, 2^{n-1} – 1}*, IEICE Trans. Inf. Syst. **92** (2009), no. 9, 1628–1638.
26. M. Xu, Z. Bian, and R. Yao, *Fast sign detection algorithm for the RNS moduli set {2^{n+1} – 1, 2^n–1, 2^n}*, IEEE Trans. VLSI Syst. **23** (2014), no. 2, 379–383.
27. V. Niras and Y. Kong, *Fast sign-detection algorithm for residue number system moduli set {2^n – 1, 2^n, 2^{n+1} – 1}*, IET Comput. Digital Tec. **10** (2016), no. 2, 54–58.
28. S. Kumar and C.-H. Chang, *A VLSI-efficient signed magnitude comparator for {2^n–1, 2^n, 2^{n+1} – 1} RNS*, EE Int. Symp. Circuits Syst. (Montreal, Canada), May 2016, 1966–1969.
29. P. Patronik and S. J. Piestrak, *Hardware/software approach to designing low-power RNS-enhanced arithmetic units*, IEEE Trans. Circuits Syst. I **64** (2017), no. 5, 1031–1039.
30. A. Hiasat, *New residue number system scaler for the three-moduli set {2^{n+1} – 1, 2^n, 2^n–1}*, Computers **7** (2018), no. 3, 46.
31. A. Hiasat and L. Sousa, *On the design of RNS inter-modulo processing units for the arithmetic-friendly moduli sets {2^{n+k}, 2^n–1, 2^{n+1} – 1}*, Comput. J. **62** (2018), no. 2, 292–300.
32. S. J. Piestrak, *Design of residue generators and multioperand modular adders using carry-save adders*, IEEE Trans. Comput. **43** (1994), no. 1, 68–77.
33. H. T. Vergos and G. Dimitrakopoulos, *On modulo 2^n + 1 adder design*, IEEE Trans. Comput. **61** (2010), no. 2, 173–186.
34. A. E. Zarandi et al., *Reverse converter design via parallel-prefix adders: Novel components, methodology, and implementations*, IEEE Trans. VLSI Syst. **23** (2014), no. 2, 374–378.
35. A. Tyagi, *A reduced-area scheme for carry-select adders*, IEEE Trans. Comput. **42** (1993), no. 10, 1163–1170.
36. L. Kalampoukas et al., *High-speed parallel-prefix module 2^n–1 adders*, IEEE Trans. Comput. **49** (2000), no. 7, 673–680.

37. H. T. Vergos, C. Efstathiou, and D. Nikolos, *Diminished-one modulo 2^n+1 adder design*, IEEE Trans. Comput. **51** (2002), no. 12, 1389–1399.
38. R. Muralidharan and C.-H. Chang, *Area-power efficient modulo 2^n-1 and modulo 2^n+1 multipliers for $\{2^n-1, 2^n, 2^n+1\}$ based RNS*, IEEE Trans. Circuits Syst. I **59** (2012), no. 10, 2263–2274.
39. R. Zimmermann, Binary adder architectures for cell-based VLSI and their synthesis, PhD Thesis, Swiss Federal Institute of Technology, Zurich, 1998.

AUTHOR BIOGRAPHIES

MohammadReza Taheri received his BS degree in Computer Hardware Engineering from Isfahan University, Isfahan, Iran. He obtained his MS degree in Computer System Architecture from the Science and Research Branch of IAU, Tehran, Iran. He is currently pursuing his PhD in Computer Architecture at Shahid Beheshti University, Tehran, Iran. He has also been a member of the Nanotechnology and Quantum Computing Laboratory of Shahid Beheshti University since 2009. His current research interests include computer arithmetic and VLSI design.

Keivan Navi received his BS and MS degrees in Computer Hardware Engineering from Beheshti University, Tehran, Iran in 1987 and the Sharif University of Technology, Tehran, Iran in 1990, respectively. He also received a PhD in Computer Architecture from Paris XI University, Paris, France, in 1995. He is currently a Professor with the faculty of Computer Science and Engineering at Beheshti University. His research interests include VLSI design, computer arithmetic, circuit techniques for emerging technologies, quantum computing, and interconnection networks.

Amir Sabbagh Molahosseini received his BS degree from the Shahid Bahonar University of Kerman, Kerman, Iran in 2005, and MS degree and PhD in Computer Engineering from the Science and Research Branch of the Islamic Azad University, Tehran, Iran in 2007 and 2010, respectively. He is currently an Assistant Professor in the Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran. He has also been a visiting researcher at the Signal Processing Systems Group, INESC-ID, IST, University of Lisbon, Lisbon, Portugal. His current research interests are computer arithmetic and alternative computing systems. He is currently an Associate Editor for the Journal of Electrical and Computer Engineering. He is also a Senior Member of IEEE.