A Generic Modulo-$(2^n \pm \delta)$ Addition Algorithm via Two-Valued Digit Encoding

Saeid Gorgin[1], Amirhossein Sadr[2,3], Dara Rahmati[3], Jungrae Kim[1]
[1] Department of Electrical and Computer Engineering, Sungkyunkwan University, Republic of Korea
[2] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[3] Department of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran
me@sgorgin.com, a.sadr@ipm.ir, d_rahmati@sbu.ac.ir, dale40@skku.edu

***Abstract*—Modular adders are essential arithmetic components in Residue Number System (RNS)-based applications, including digital signal processing, cryptography, and machine learning. These applications consistently push the boundaries of dynamic range (DR) and operating frequency, making the design of efficient generic modular adders a critical and evolving challenge. This paper presents a novel algorithm for modulo-$(2^n \pm \delta)$ addition, where δ is an integer within the range $0 \leq \delta \leq 2^{n-1} - 1$. The proposed approach leverages a two-valued digit (twit) for encoding the value of $\pm\delta$ and uses a faithful representation of operands. In this representation, each operand is encoded as an n-bit unsigned number augmented by a twit value $\{0, \pm\delta\}$. The algorithm efficiently performs modular addition by speculating and adjusting the twit value in the addition result. When the result exceeds the modulus, it subtracts $2^n \pm \delta$ by ignoring the carry-out and adjusting the speculated twit value. This adjustment is achieved through an XOR operation between the carry-out and the speculated twit value, simplifying the modular reduction process. The proposed design has been synthesized for practical n ($4 \leq n \leq 16$) using a FreePDK 45 nm process. The results demonstrate superior performance across key metrics such as delay, area, and power consumption compared to previous designs, highlighting the efficacy and scalability of the approach.**

Keywords—Residue Number System (RNS), Modular addition, Generic adder, Two-Valued Digit (Twit).

I. INTRODUCTION

The RNS is one of the most well-known unconventional number representations, attracting significant interest in fields such as communications [26], cryptography [1], neural networks [27], and digital signal/image processing [21]. These applications demand high-performance, low-power arithmetic circuits. RNS achieves its advantages through a high level of parallelism and reduced carry propagation, making it particularly attractive for addition- and multiplication-intensive workloads (e.g., convolution operations [29]). Thus, the efficiency of modular addition is critical for applications relying on RNS, as modular multipliers essentially perform successive modular additions [20], [11], [25].

Extensive research has focused on optimizing modular addition for specific moduli, such as $2^n \pm 1$, to balance delay within the classical moduli set $\tau = \{2^n, 2^n \pm 1\}$ [35], [30], [32], [34], [3], [5], [33], [14], [15], [29]. More recently, innovative designs have been proposed for moduli, such as $(2^n \pm 3)$ [6], [16] and $(2^n - 2^k - 1)$ [18]. These designs aim to increase the number of co-prime moduli for greater dynamic range or to enable higher operating frequencies by reducing channel width. However, these approaches are not sufficient to meet the growing demand for higher performance and greater dynamic range.

A promising response to this challenge is the development of efficient generic modulo adders that can handle any required co-prime modulus. While previous works have proposed generic designs, such as [2] and [12], these efforts are relatively modest compared to the extensive optimizations designed for $2^n \pm 1$ adders. These designs typically rely on computing the sum of two operands and applying a reduction step when the sum exceeds the modulus, requiring at least two Carry-Propagate Adders (CPAs) and a multiplexer to select the correct result.

In this paper, we present a new efficient generic modulo-$(2^n \pm \delta)$ addition algorithm, where δ is an integer in the range $0 \leq \delta \leq 2^{n-1} - 1$. The proposed design introduces a novel representation where the $\pm\delta$ bias is encoded as a twit value $\{0, \pm\delta\}$. In this encoding, all codewords are valid, allowing redundant representations. The algorithm avoids needing two CPAs and a multiplexer by speculating the twit value with a simple 4-input combinational logic block. This block takes, as inputs, the most significant bit and twit values of each operand. Additionally, it generates a compensating value to ensure the sum of the speculated twit and compensating value aligns with the weighted sum of the block inputs. The final result is computed by summing the compensating value with the remaining operand bits. This summation can occasionally produce a carry-out when the speculated twit value equals one. In such cases, the twit value is corrected to zero (i.e., reducing $\mp\delta$) by XORing the carry-out with the speculated twit value, ensuring the final result's correctness while maintaining the design's efficiency.

We evaluate the proposed design using both analytical and experimental methods. Analytical evaluation highlights the algorithm's superiority over prior designs. Synthesis results, performed using Synopsys Design Compiler with the FreePDK 45nm technology library, confirm the analytical findings. The results demonstrate improvements in delay, area, and power consumption. This paper makes the following contributions:

- Introduces an efficient use of the twit representation for residue encoding in modulo-$(2^n \pm \delta)$.
- Proposes a generic modulo- $(2^n \pm \delta)$ addition algorithm.
- Achieves low-cost subtraction equivalent to conventional binary representations.
- Provides comprehensive analysis and comparison of the proposed design against related works.
- Demonstrates fault tolerance using Reconfigurable Modular Adders (RMAs) for fault-tolerant RNS processors.

The remainder of this paper is structured as follows. Section II provides an overview of RNS, generic modulo-$(2^q \pm \delta)$ addition, and weighted bit set encoding with twit representation. Section III reviews related works. Section IV details the proposed generic modulo addition algorithm. Section V presents evaluations and comparisons with prior designs. Finally, Section VI concludes the paper.

2576-2265/25/$31.00 ©2025 IEEE
DOI 10.1109/ARITH64983.2025.00023

***Availability*:** To promote reproducibility, transparency, and further advancements in the field, the HDL codes developed during this study have been made publicly available at https://github.com/GorginSaeid/Generic-Modulo-Adder.

II. Background

A. Residue Number System (RNS)

The RNS encoding is an alternative integer representation in which a number $X \in [0, M)$ is represented as $X_{RNS} = (x_1, \dots, x_k)$, where for $1 \le i \le k$, $x_i = |X|_{m_i} \in [0, m_i)$ (i.e., the remainder of integer division X/m_i, read as X modulo m_i). The dynamic range (DR) is given by $M = m_1 \times m_2 \times \dots m_k$, assuming that the moduli $\{m_1, \dots, m_k\}$ are mutually prime, which is commonly the case [20].

RNS representation enables parallel computation across different residues, while the long carry chains in doing arithmetic on conventional binary numbers $X, Y \in [0, M)$, are broken into small bit-widths of the k parallel residue channels (one per each modulo m_i). The arithmetic operation (typically addition or multiplication) is performed on the corresponding residues x_i and y_i which are generated via the forward binary-to-modulo-m_i conversion for $1 \le i \le k$. The results of a series of RNS operations (e.g., in convolution [7]) are then subject to reverse conversion back to binary via the Chinese Remainder Theorem (CRT).

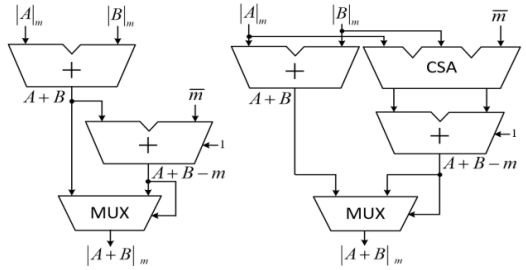
B. Generic Modulo Addition Algorithm

A generic modulo addition algorithm computes the result of $|A + B|_m = (A + B)\ mod\ m$, where m is of the form $2^n \pm \delta$, and A and B are remainders from division by m (i.e., $A, B < m$). Unlike specific modulo operations that exploit predefined structures of the modulus, such as $2^n \pm 1$ or $2^n \pm 3$, generic algorithms must accommodate a wider range of modulus values. This flexibility introduces additional complexity, particularly in managing carries and performing reduction steps, which are inherently more challenging in generic designs. The general modulo addition operation is mathematically expressed in Equation (1), adhering to the conventional definition:

$$|A+B|_m = \begin{cases} A+B, & \text{if } A+B < m \\ A+B-m, & \text{if } A+B \ge m \end{cases} \quad (1)$$

The direct implementation of Equation (1) is illustrated in Fig. 1(a) [2]. To avoid the computationally expensive propagation of two n-bit CPA on critical path delay, both $A + B$ and $A + B - m$ can be computed simultaneously in two parallel paths. In the path handling $A + B - m$, a Carry-Save Adder (CSA) is utilized to sum A, B, and two's complement of m, before the final n-bit carry propagation [31]. The correct result is then selected between the two parallel computations based on the carry-out of the speculated sum, as demonstrated in Fig. 1(b). This approach reduces latency and improves the overall efficiency of the generic modulo adder.

Generic modulo adders play a critical role in increasing the dynamic range and reducing the bit-width of data channels, enabling higher operating frequencies in digital systems. However, designing such components is inherently complex due to the need to support a wide range of modulus values and the associated challenges of end-around carry handling. The most significant advancements in this area will be reviewed in Section III-B.



(a) Sequential implementation [2] (b) Parallel implementation [31]

Fig. 1. Abstract view of generic Modulo-m addition

C. Two-Valued Digit

Similar to a binary digit, or bit, which can take on two values, 0 or 1, a two-valued digit represents two distinct values, denoted as α and β [23]. In this work, the two-valued digit is specifically employed to encode the values $\pm\delta$ as $\{0, \pm\delta\}$. This encoding reduces storage requirements and simplifies the arithmetic operations required in the proposed modular reduction algorithm.

The foundation of this approach lies in the Weighted Bit-Set (WBS) encoding framework, which generalizes conventional binary representation schemes [13]. WBS encoding represents digits as weighted bit-sets, enabling efficient arithmetic operations such as addition, subtraction, and carry handling. The two-valued digit, referred to as a "twit", extends this concept by introducing a gap-based representation that preserves redundancy benefits while minimizing hardware overhead.

A twit is formally defined as a binary variable capable of representing two integer values, determined by a lower value L and a gap G. Formally, a twit represents the set $\{L, L + G\}$. In this work, we define $L = 0$ and $G = \pm\delta$, leading to the digit set $\{0, \pm\delta\}$. This representation is compact and efficient, eliminating the need for additional bits while maintaining compatibility with VLSI implementations [13].

III. Prior Works

A. Specific Modulo Adders

As previously mentioned, extensive research has focused on the development of modulo-$(2^n \pm 1)$ adders. Alongside these designs, several proposals address specific moduli, such as $(2^n \pm 3)$, employing innovative techniques that leverage predictable carry patterns. These designs often achieve optimized latency and area efficiency through specialized hardware architectures, including parallel-prefix and carry-select adder mechanisms.

For example, efficient VLSI implementations for modulo-$(2^n - 1)$ adders have been proposed, optimizing both area and delay [35]. Parallel-prefix adders that eliminate the end-around carry for modulo-$(2^n - 1)$ addition have also been introduced, significantly minimizing delay [17]. Enhanced diminished-one adders for modulo-$(2^n + 1)$ arithmetic have been developed, focusing on simplified carry recirculation [30], while fast modulo-$(2^n + 1)$ adders based on parallel-prefix architectures have been proposed [32]. Ling-carry and sparse-prefix architectures have been utilized to improve area and delay trade-offs in large modulo-$(2^n -$

1) designs [3]. Extensions to these methods introduced parallel-prefix architectures for modulo-$(2^n + 1)$ arithmetic [5]. Other contributions include sparse carry computation methods for modulo- $(2^n \pm 1)$ designs [34], efficient adder/subtractor designs [33], and signed-LSB residue approaches for modular arithmetic [14].

A double $\{0, 1, 2\}$ representation for modulo- $(2^n - 3)$ adders have been proposed to reduce carry propagation delay for small moduli [6]. Further advancements include the Diminished-3 representation for modulo- $(2^n + 3)$, which optimizes performance using balanced parallel-prefix techniques [16]. Additionally, flexible modular adders for $2^n - 2^k - 1$ have been developed using carry-correction techniques to improve speed and efficiency [18].

B. Generic Modulo Adders

One of the earliest generic modular adder architectures was proposed in [2], utilizing simple binary adders and multiplexers. Later, a generic residue adder using one binary adder and a feedback register was introduced in [4]. In [10], hybrid adders for $2^n \pm \delta$ moduli were proposed, merging carry-propagate and carry-save structures. This approach was further refined in [31] to improve delay characteristics.

A power-efficient CPA-based design combining Kogge-Stone and Ladner-Fischer adders to reduce overhead and achieve near-optimal delay was presented in [24]. Parallel-prefix architectures suitable for residue number systems were investigated in [19], achieving a balance between hardware complexity and performance. A unified modular adder/subtractor with improved speed, area, and power metrics was developed in [28]. In [12], general $2^n \pm \delta$ architectures leveraging tristate-based multiplexers were introduced, achieving marginal improvements. Finally, a flagged-prefix addition method focusing on reduced power and area for arbitrary moduli was proposed in [8]. Furthermore, some studies, such as [9], have explored generic adders implemented on FPGA platforms. However, these designs heavily rely on FPGA-specific architectural features and are not discussed in this paper.

IV. Generic Modulo-$(2^n \pm \boldsymbol{\delta})$ Operations

This section provides a detailed explanation of the proposed algorithm. We begin by introducing the operand encoding method, focusing on the use of twit encoding in order to represent operands in the modulo $(2^n \pm \delta)$. Next, the addition algorithm is described in detail, accompanied by numerical examples to illustrate the process. Following this, we discuss the negation process within the proposed format and extend the discussion to the subtraction operation. Finally, we outline our approach for designing reconfigurable adders that support various moduli based on modulo-$(2^n \pm \delta)$ adders. This design lays the groundwork for a fault-tolerant RNS processor, enabling more robust and flexible computation.

A. Operand Representation

The proposed representation encodes each operand as an n-bit unsigned integer, augmented by a twit value $\{0, \pm\delta\}$, and stores it in memory as an $(n + 1)$-bit number, represented as $a_{n-1}a_{n-2} \dots a_2a_1(a_0\delta_a)$. This approach is conceptually similar to the double least significant bit (LSB) representation [22] in terms of memory usage, but instead of using an extra bit in the LSB position, it employs a twit. It simplifies hardware implementation while maintaining compatibility across different values of $\pm\delta$. All codewords are valid within this representation. For negative values of δ $(-\delta)$, every representable value in modulo-$(2^n - \delta)$ has more than one equivalent form, where representations with a negative value or greater than the modulus are also considered after a modulo operation. However, for positive values of δ $(+\delta)$ (i.e., modulo-$(2^n + \delta)$), only certain values possess dual representations.

The mathematical properties of two-valued digit make this representation particularly beneficial for modular arithmetic. Using this format, modular operations are streamlined by efficiently handling the end-around carry of $\pm\delta$. This simplifies the result adjustment, allowing the desired final value to be computed with lower computational overhead.

Although a remainder from a modulus $2^n - \delta$ can be represented using only n-bit, the proposed $(n + 1)$-bit representation (including the twit) offers significant advantages. The extra bit enables a unified representation for both $2^n - \delta$ and $2^n + \delta$, facilitating a generic modulo-$(2^n \pm \delta)$ system. This unified framework enhances versatility and provides computational fault tolerance, as will be further detailed in Section IV-D.

Example 1: Consider $n = 5$ and $\delta = 7$.

- For modulo-$(2^5 - 7)$, the value 15 can be represented as $0111\ (1\mathbf{0})$ (using $\delta_a = \mathbf{0}$) or $1011\ (0\mathbf{1})$ (using $\delta_a = \mathbf{1}$). In the first representation, where $\delta_a = 0$, $0111\ (1\mathbf{0})$ corresponds directly to 15, as $(01111)_2 = 15$. In the second representation, where $\delta_a = 1$, $1011\ (0\mathbf{1})$ corresponds to 15 because $(10110)_2 = 22$, and $22 - 7 = 15$.
- For modulo-$(2^5 + 7)$, the value 15 can be represented as to $0111\ (1\mathbf{0})$ or $0100\ (0\mathbf{1})$. Similarly, in the first representation ($\delta_a = \mathbf{0}$), $0111\ (1\mathbf{0})$ corresponds directly to 15. In the second representation, where $\delta_a = \mathbf{1}$, $0100\ (0\mathbf{1})$ corresponds to 15 because $(01000)_2 = 8$, and $8 + 7 = 15$.

B. Addition Algorithm

Besides utilizing parallel paths to enhance the performance of the addition algorithm, Equation (1) can be reformulated as Equation (2) to simplify the comparison between $A + B$ and m. Given $m = 2^n \pm \delta$, the comparison reduces to checking the carry-out of $A + B + \theta$, where $\theta = \delta$ and $\hat{n} = n$ for modulo-$(2^n - \delta)$. While for modulo-$(2^n + \delta)$, θ is the two's complement of δ and $\hat{n} = n + 1$.

$$|A + B|_{2^n \pm \delta} = \begin{cases} |A + B|_{2^{\hat{n}}}, & \text{if } A + B + \theta < 2^{\hat{n}} \\ |A + B + \theta|_{2^{\hat{n}}}, & \text{if } A + B + \theta \geq 2^{\hat{n}} \end{cases} \quad (2)$$

Unlike this conventional approach, the proposed addition algorithm takes a different route, relying on speculating the twit value in the result. Algorithm 1 describes the process in detail. By examining the most significant bits (i.e., a_{n-1} and b_{n-1}) and the twit of two operands (i.e., δ_a and δ_b), the algorithm speculates a twit value (i.e., δ_ε) and computes a compensating value (i.e., Z). After summing the lower bits of the two operands along with Z, the result includes an n-bit sum (i.e., $s_{n-1}s_{n-2} \dots s_1s_0$) and a carry-out value (i.e., C_{out}). The carry-out is then used to apply an end-around carry with a value of $-\delta$ for modulo-$(2^n + \delta)$ or $+\delta$ for modulo-$(2^n -$

δ). Since the speculated twit value δ_ε already incorporates the adjustment for $\pm\delta$, the end-around carry is performed by zeroing the final twit (δ_s) using an XOR operation with δ_ε. Notably, when the twit value is zero, the carry-out will also be zero.

Algorithm 1: Modulo-$(2^n \pm \delta)$ Addition ($\delta < 2^{n-1}$)

Inputs: $A = a_{n-1}a_{n-2} \dots a_2a_1(a_0\delta_a)$,
$B = b_{n-1}b_{n-2} \dots b_2b_1(b_0\delta_b)$.
Output: $S = s_{n-1}s_{n-2} \dots s_2s_1(s_0\delta_s)$.
Intermediate variable: $Z = z_{n-1}z_{n-2} \dots z_1z_0, \delta_\varepsilon$

❶Compute intermediate Z, and speculated twit δ_ε:

$$Z = \underbrace{|(a_{n-1} + b_{n-1})2^{n-1} + \delta_a + \delta_b|_{2^n \pm \delta}}_{V} - \delta_\varepsilon,$$

$$\text{where } \delta_\varepsilon = \begin{cases} 0, & V = 0 \\ 1, & V > 0 \end{cases}$$

❷Compute sum and carry-out:

$$C_{out}s_{n-1}s_{n-2} \dots s_1s_0 = a_{n-2} \dots a_1a_0 + b_{n-2} \dots b_1b_0 + z_{n-1}z_{n-2} \dots z_1z_0$$

❸Final twit adjustment:

$$\delta_s = C_{out} \oplus \delta_\varepsilon.$$

The hardware implementation of this algorithm is illustrated symbolically in Fig. 2. The process begins with the computation of Z (step ❶) and continues with a CSA in step ❷. Notably, the computation of $a_i \oplus b_i$ can be performed simultaneously with the corresponding logic to compute Z. The second part of step ❷ uses a CPA to sum the intermediate results, $\Sigma = \sigma_{n-1}\,\sigma_{n-2} \dots \sigma_2\sigma_1\sigma_0$ (where $\sigma_{n-1} = z_{n-1}$) and $\Phi = \varphi_{n-1} \dots \varphi_2\varphi_1 0$. The carry out from the CPA is then used in step ❸ to finalize the twit value δ_s. The CPA can employ various carry acceleration techniques, such as the Kogge-Stone parallel-prefix tree, to achieve a trade-off between performance and hardware cost.

Theorem 1: Behavior of Carry-Out When $\delta_\varepsilon = 0$
Statement: In Algorithm 1, if $\delta_\varepsilon = 0$, then $C_{out} = 0$.
Proof: From Algorithm 1, when $\delta_\varepsilon = 0$, the value of $Z = 0$. Since the lower $n-1$ bits of A and B (i.e., $a_{n-2} \dots a_1a_0$ and $b_{n-2} \dots b_1b_0$) are both less than 2^{n-1}, their sum will also be less than 2^n. Therefore, the carry-out (C_{out}) cannot be 1.■

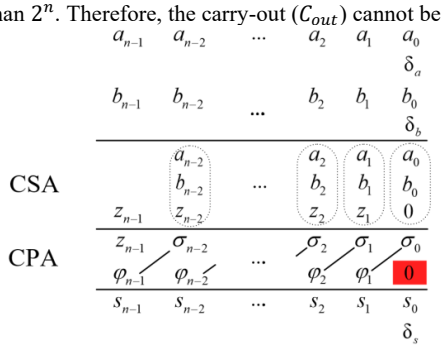


Fig. 2. Hardware realization of Algorithm 1 in symbolic representation ($\sigma_i = a_i \oplus b_i \oplus z_i$ and $\varphi_{i+1} = a_ib_i \vee a_iz_i \vee b_iz_i$)

The final two parts of Algorithm 1 (❷ and ❸) are straightforward, as they involve computing the sum and adjusting the twit value. So, we provide further clarification for step ❶, where the intermediate variable V and the compensating value Z are calculated. Since the behavior of $V = |(a_{n-1} + b_{n-1})2^{n-1} + \delta_a + \delta_b|_{2^n \pm \delta}$ depends on whether the modulo operation is $(2^n + \delta)$ or $(2^n - \delta)$, we explain each case separately.

- For modulo-$(2^n + \delta)$:
  Here, V can take the values:
  $$\{0, \delta, 2\delta, 2^{n-1}, 2^{n-1} + \delta, 2^{n-1} + 2\delta, 2^n\}.$$
  - When $V = 0$ the speculated twit value δ_ε is zero, and $Z = 0$.
  - When $V \neq 0$, $\delta_\varepsilon = 1$, and Z takes the values:
    $$\{0, \delta, 2^{n-1} - \delta,\ 2^{n-1}, 2^{n-1} + \delta,\ 2^n - \delta\}.$$
    In its n-bit form, Z can be expressed as:
    $$\{00 \dots 00, 0\delta_{n-2} \dots \delta_1\delta_0,\quad 0\overline{\delta_{n-2}} \dots \overline{\delta_1}\,\overline{\delta_0} + 1,$$
    $$10 \dots 00, 1\delta_{n-2} \dots \delta_1\delta_0, 1\overline{\delta_{n-2}} \dots \overline{\delta_1}\,\overline{\delta_0} + 1\}.$$
- For modulo-$(2^n - \delta)$:
  Here, V can take the values:
  $$\{0, \delta, 2^{n-1} - 2\delta, 2^{n-1} - \delta, 2^n - 3\delta,$$
  $$2^n - 2\delta, 2^{n-1}\}.$$
  - When $V = 0$, $\delta_\varepsilon = 0$, and $Z = 0$.
  - When $V \neq 0$, $\delta_\varepsilon = 1$, and Z takes the values:
    $$\{0, 2\delta, 2^{n-1} - \delta, 2^{n-1}, 2^{n-1} + \delta, 2^n - 2\delta\}.$$
    In its n-bit form, Z can be expressed as:
    $$\{00 \dots 00,\ \delta_{n-2} \dots \delta_1\delta_0 0, 0\overline{\delta_{n-2}} \dots \overline{\delta_1}\,\overline{\delta_0} + 1,$$
    $$10 \dots 00, 1\delta_{n-2} \dots \delta_1\delta_0,\ \overline{\delta_{n-2}} \dots \overline{\delta_1}\,\overline{\delta_0} + 2\}.$$

Example 2: Consider $n = 5$ and $\delta = 7$.

Modulo-$(2^5 + 7)$
$A = 14$ and $B = 19$

$A = 14$	0	1	1	1	0	0
$B = 19$	1	0	0	1	1	0
$A[n-2,0]$		1	1	1	0	
$B[n-2,0]$		0	0	1	1	
$Z = 2^{n-1} - \delta$	0	1	0	0	1	
Σ	0	0	1	0	0	
Φ	1	0	1	1	0	
$S = 33$	1	1	0	1	0	1

$\delta_\varepsilon = 1$, and $C_{out} = 0$

$S = |14 + 19|_{39} = 33$

Modulo-$(2^5 - 7)$
$A = 23$ and $B = 10$

$A = 23$	1	1	1	1	0	1
$B = 10$	0	1	0	1	0	0
$A[n-2,0]$		1	1	1	0	
$B[n-2,0]$		1	0	1	0	
$Z = 2^{n-1}$	1	0	0	0	0	
Σ	1	0	1	0	0	
Φ	1	0	1	0	0	
$S = 8$	0	1	0	0	0	0

$\delta_\varepsilon = 1$, and $C_{out} = 1$

$S = |23 + 10|_{25} = 8$

C. *Subtraction Algorithm*

The subtraction algorithm in the proposed number representation follows a similar procedure to conventional binary arithmetic, where subtraction is performed by adding the two's complement of the subtrahend. In this representation, the complement of the subtrahend is obtained the same as the two's complement procedure, by inverting all its bits and adding 1 to the least significant bit position.

Thus, to compute $|A - B|_{2^n \pm \delta}$, we can equivalently compute $|A + \bar{B} + 1|_{2^n \pm \delta}$. Here, 2^n can be written as an n-bit number with all bits set to 1, plus 1 (i.e., $1 \dots 11 + 1$). Also, the subtrahend B consists of the n-bit value $b_{n-1}b_{n-2} \dots b_2b_1b_0$ along with a twit δ_b. The twit δ_b evaluates to 0 when not set and $\pm\delta$ when $\delta_b = 1$. So, for $2^n \pm \delta - B$ (equivalent to $-B$), the operation involves:

$$\bar{B} + 1 = \overline{b_{n-1}}\,\overline{b_{n-2}} \dots \overline{b_2}\,\overline{b_1}\,\overline{b_0}\,\overline{\delta_b} + 1.$$

While for adding the $+1$, the position φ_0 (shaded in red in Fig. 2) must be updated to reflect the subtraction signal (*Sub.*), where φ_0 is 0 for addition and set to "*Sub.*" for subtraction. Consequently, the subtraction cost in this representation matches that of conventional binary subtraction, requiring only one XOR gate compared to addition. The corresponding circuit for combined addition and subtraction, which accounts for this XOR gate overhead, is illustrated in Fig. 3.

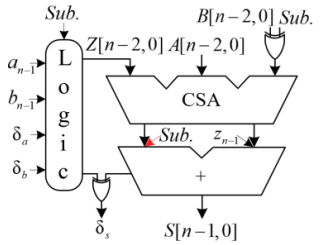


Fig. 3. Combined Addition/Subtraction circuit.

To clarify the functionality of the subtraction algorithm, we provide examples for $n = 5$, $\delta = +7$, where both $A - B$ and $B - A$ are computed. These examples demonstrate that the subtraction algorithm maintains consistency and efficiency, with the additional cost of a single XOR gate for computing the complement of the subtrahend.

Example 3: $A = 24$ and $B = 19$, for Modulo-$2^5 + 7$:

$A = 24$	1	0	0	0	1
					1
$B = 19$	0	1	1	0	0
					1
$A[n-2,0]$		0	0	0	1
$\overline{B[n-2,0]}$		0	0	1	1
$Z = 0$	0	0	0	0	0
Σ	0	0	0	1	0
Φ	0	0	0	1	1
$S = 5$	0	0	1	0	1
					0

$S = |24 - 19|_{39} = 5$

$B = 19$	0	1	1	0	0
					1
$A = 24$	1	0	0	0	1
					1
$B[n-2,0]$		1	1	0	0
$\overline{A[n-2,0]}$		1	1	1	0
$Z = 0$	0	0	0	0	0
Σ	0	0	0	1	0
Φ	1	1	0	0	1
$S = 34$	1	1	0	1	1
					1

$S = |19 - 24|_{39} = 34$

D. Toward a Fault-Tolerant RNS Processor

The unified design strategy for modulo-$(2^n \pm \delta)$ adders enables the synthesis of RMAs that can process inputs for multiple moduli based on configuration signals provided by control circuitry (i.e., δ) [14]. This reconfigurability facilitates fault-tolerant designs with significantly reduced hardware redundancy compared to conventional full-replication approaches.

As shown in Fig. 4, for a system with K moduli, instead of implementing K dedicated adders, the design requires $K + \Omega$ RMAs, where Ω represents the number of spare units (Gray shaded in Fig. 4). These spare adders remain idle during normal operation and are activated when a fault is detected in one of the operational adders. The shift-switch logic seamlessly replaces the faulty adder with one of the Ω spare RMAs with the corresponding configuration, ensuring continuous operation with minimal disruption.

While this fault-tolerant strategy introduces a latency penalty during reconfiguration, the benefits of reduced hardware overhead and increased fault resilience make it a highly cost-effective solution. This approach is particularly advantageous for reliable RNS processors in applications where fault tolerance and efficiency are critical, such as safety-critical systems.

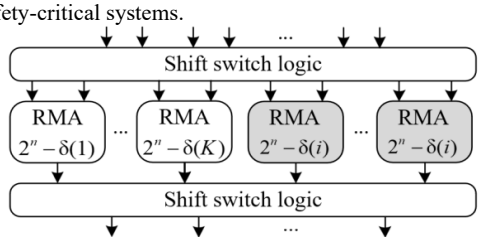


Fig. 4. Abstract view of a Fault-tolerant K-moduli RNS processor via reconfigurable modulo-$(2^n \pm \delta(i))$ adder (adopted form [14])

V. Performance Evaluation and Comparisons

A. Methodology

We implemented our proposed design along with three of the most prominent prior works for comparison. These include the first generic modulo adder design by [2], the main improvement achieved through parallelization by [31], and one of the best-optimized designs presented in [12]. Additionally, we included a simple binary adder as a baseline for comparison. All designs are implemented using HDL and verified with extensive random test vectors, as well as manually generated vectors for corner cases.

For evaluation, we utilized both analytical estimations to obtain a rough understanding of each design's performance and synthesis results for a more detailed examination of delay, area, and power consumption. The HDL implementations are synthesized using Synopsys Design Compiler with the FreePDK 45 nm technology under identical conditions to ensure a fair comparison. Following synthesis, we perform comprehensive post-synthesis verification to validate the correctness of the synthesis process.

B. Analytical Evaluation

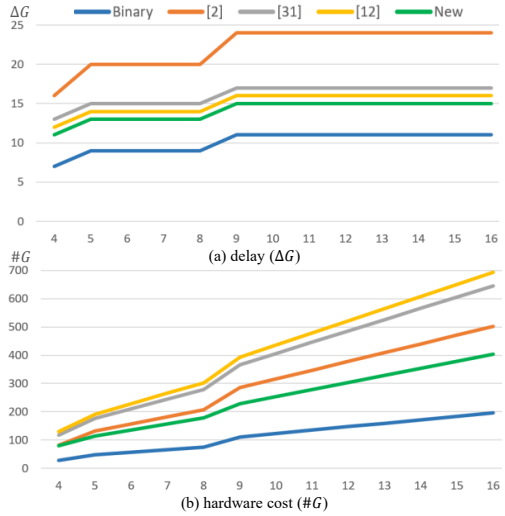
Table I presents various designs' delays and hardware costs, analyzed based on fundamental components such as the CSA, carry generation tree, XOR gate, multiplexer, and other combinational logic. The delay and area metrics are expressed in terms of ΔG (i.e., the delay of a simple 2-input gate) and $\#G$ (i.e., the number of 2-input gates), respectively. For delay analysis, XOR gates and multiplexers incur a delay of 2 ΔG, CSA have a delay of 4 ΔG, Carry generation trees (CGTs) are modeled with a delay of $(1 + 2\lceil \log_2 n \rceil)\Delta G$, where n is the bit-width. While for area Analysis, XOR gates and multiplexers require $3n$ $\#G$, CSA have a hardware cost of $9n$ $\#G$, and CGTs have a hardware cost of $(3 + 3n\lceil \log_2 n \rceil - 3n)$ $\#G$.

Critical Path Delay (CPD) components are highlighted in bold in Table I. For example, in the design by [31], two adders (each implemented as a carry generation tree and an XOR gate) are used, but only one adder contributes to the CPD. In [12], multiplexers are implemented with three-state buffers. As an exception, their delay is considered $1\Delta G$, denoted as $\mathbf{1}^*$ in the table. For n-input combinational logic (CL), the delay is typically $\lceil \log_2 n \rceil \Delta G$ with a hardware cost of n $\#G$. In our proposed design, we utilize four 4-input combinational logic blocks, resulting in a combinational delay of $2\Delta G$ and a hardware cost of $4\#G$. However, this combinational logic does not affect the CPD of the design, as it operates in parallel with the first XOR gate of the CSA.

TABLE I. Analytical Evaluation Based on Basic Components in Terms of Delay (ΔG) and Hardware Cost ($\#G$)

Designs	*n*-bit components				CL	Delay ΔG	Cost $\#G$
	CSA	CGT	XOR	MUX			
Binary	0	**1**	**1**	0	0	$3 + 2\lceil \log_2 n \rceil$	$3 + 3n\lceil \log_2 n \rceil$
[2]	0	**1+1**	**1+1**	**1**	0	$8 + 4\lceil \log_2 n \rceil$	$6 + 7n + 6n\lceil \log_2 n \rceil$
[31]	**1**	**1**+1	**1**+1	**1**	0	$9 + 2\lceil \log_2 n \rceil$	$6 + 16n + 6n\lceil \log_2 n \rceil$
[12]	**1**	**1**+1	**1**	**1***+1*	0	$8 + 2\lceil \log_2 n \rceil$	$6 + 19n + 6n\lceil \log_2 n \rceil$
New	**1**	**1**	**1**	0	1	$7 + 2\lceil \log_2 n \rceil$	$3 + 13n + 3n\lceil \log_2 n \rceil$

Based on the two rightmost columns of Table I, we present Fig. 5, which illustrates the delays and hardware costs of the aforementioned designs in terms of ΔG and $\#G$, while varying n from 4-bit to 16-bit. As expected, the simple binary design outperforms all others in both parameters. However, the proposed design demonstrates lower delay and hardware cost compared to the best prior works, [12] and [2], respectively. It is important to emphasize that this analytical estimation provides only a rough approximation of each design's performance. The actual results may differ when considering critical factors in real-world implementations or more detailed simulations, such as fan-in, fan-out, the number of metal tracks, and other physical design constraints.



(a) delay (ΔG)

(b) hardware cost ($\#G$)

Fig. 5. Analytical evaluation of different designs for n ($4 \le n \le 16$)

C. Performance in other RNS operations

In modular arithmetic, as in conventional binary arithmetic, addition and subtraction serve as the foundational operations for nearly all computations. These operations are critical for enabling higher-level functions such as modular multiplication, multiply-and-accumulate (MAC) operations, and forward and reverse conversions. Their significance is especially pronounced in scenarios involving sequential implementations, where efficient execution directly influences overall performance [23].

For example, in an n-bit multiplication, $n-1$ addition operations are required, with MAC operations introducing one additional accumulation step. Consequently, the proposed design is expected to achieve an improvement of approximately $(n-1)\Delta G$ for multiplication and $n\,\Delta G$ for MAC operations.

Forward conversion operations, which manage $m \times n$-bit numbers, highlight the importance of efficient modular addition and subtraction even further. These conversions typically involve $m-1$ modular multiplications with constant coefficients of δ and $m-1$ addition/subtraction operations. So, any improvement in modular addition or subtraction has a direct and significant impact on the efficiency of these forward conversions. In reverse converters, while modular addition plays a role in computations, its relative importance is diminished compared to other dominating operations.

Although, these estimations are based on the analytical improvements for suggested designs which are supported by synthesis outcomes (See Section V-D), however, for a comprehensive evaluation it is essential to implement and synthesize corresponding circuits.

D. Synthesis Results

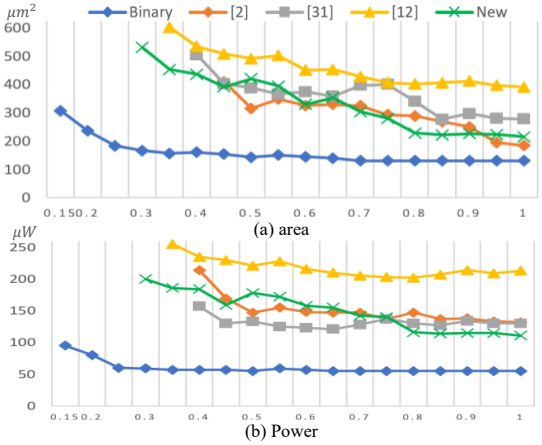
The synthesis results of selected modulo-$(2^n+\delta)$ adders for two different channel widths (i.e., $n, \delta = 8, 11$ and $16, 63$) are summarized in Table II. For comparison, we include the synthesis results of a simple binary adder as a baseline. According to Table II, the proposed design demonstrates varying degrees of improvement depending on the channel width. For $n = 8$, the proposed design achieves a 12% speedup and a 18% area reduction, while for $n = 16$, the improvements are 5% in speed and 17% in area compared to the best existing designs.

In terms of power consumption, the proposed design consumes more power than [31] for $n = 8$, whereas for $n = 16$, the power consumption is comparable. To ensure a fair comparison, considering the differing delays of these designs, we evaluate the Power Delay Product (PDP). The proposed design outperforms [31] in PDP by 3% for $n = 8$ and 8% for $n = 16$, making it a more power-efficient option when delay is factored in.

TABLE II.
SYNTHESIS REPORT OF DIFFERENT DESIGNS FOR $n, \delta = 8, 11$ AND $16, 63$.

(n, δ)	Designs	Delay ps	Area μm^2	Power μW	PDP $ps \times \mu W$
$(8, 11)$	Binary	205	297	96	19680
	[2]	415.9	607	230	95657
	[31]	350.7	734	**192**	67334
	[12]	365.5	615	252	92106
	New	**307.6**	**498**	212	**65211**
$(16, 63)$	Binary	316	522	182	57512
	[2]	501.1	1061	406	203447
	[31]	442.2	1254	393	173785
	[12]	427.3	1209	497	212368
	New	**406.8**	**876**	**392**	**159466**

To validate the design further, we investigated area and power across various time constraints. By sweeping time constraints from 1ns to the minimum achievable delay in 50ps decrements steps, we generated a comprehensive view of each design's characteristics, as shown in Fig. 6. While some outliers exist due to synthesizer behavior, the overall trends confirm the efficiency of the proposed design.



(a) area

(b) Power

Fig. 6. Synthesis report with different time constraints for $n, \delta = 8, 11$

Additionally, in Table III, we attempted to reproduce the synthesis results reported in [12]. However, due to limitations such as the unavailability of source code and differences in synthesis environments, we were unable to fully replicate their results. To provide a meaningful comparison, we included a simple binary adder as a reference point. Our analysis indicates that the proposed design consistently achieves better delay and area metrics compared to prior works. While the proposed design does not exhibit the lowest power consumption in 2 out of 8 cases, its superior PDP and delay demonstrate that it outperforms other designs in overall efficiency.

TABLE III. SYNTHESIS RESULTS OF DELAY, AREA, AND POWER OF PROPOSED DESIGN AND THE PRIOR WORKS

Modulo	Designs	Delay ps	Area μm^2	Power μW	PDP $ps \times \mu W$
167	Binary	184	236	76	13984
	[12]	333	572	236	78588
	[31]	335	449	135	45225
	[2]	410	417	175	71750
	New	**231**	**250**	**105**	**24255**
757	Binary	217	275	95	20615
	[12]	434	651	296	128464
	[31]	412	617	**189**	77868
	[2]	416	527	225	93600
	New	**325**	**447**	197	**64025**
1777	Binary	201	339	118	23718
	[12]	400	778	345	138000
	[31]	405	679	218	88290
	[2]	518	575	244	126392
	New	**336**	**552**	**217**	**72912**
2579	Binary	250	453	146	36500
	[12]	459	857	359	164781
	[31]	376	989	287	107912
	[2]	467	839	**337**	157379
	New	**341**	**539**	238	**81158**
6173	Binary	217	430	146	31682
	[12]	525	982	416	218400
	[31]	436	958	300	130800
	[2]	516	682	283	146028
	New	**370**	**680**	**274**	**101380**
11353	Binary	265	598	200	53000
	[12]	427	1031	451	192577
	[31]	428	973	298	127544
	[2]	525	785	342	179550
	New	**310**	**612**	**273**	**84630**
27073	Binary	235	668	212	49820
	[12]	434	1095	493	213962
	[31]	503	888	307	154421
	[2]	545	815	342	186390
	New	**316**	**677**	**275**	**86900**

To evaluate the scalability of the proposed design, we synthesized adders with channel widths ranging from $n = 4$ to $n = 16$. The results, illustrated in Fig. 7, show that the proposed design outperforms prior works in terms of delay, area, and power across all tested widths. These results underscore the design's adaptability and overall advantage in critical parameters for modular arithmetic operations. By comparing with the best-performing component from each of the referenced works, the proposed design achieves average improvements of 21% in delay, 20% in area, 14% in power consumption, and 31% in PDP. These investigations highlight the versatility and efficiency of the proposed design, establishing it as a robust choice for reconfigurable modular arithmetic with minimal trade-offs.

Fig. 7. Synthesis report with different channel bit widths (n) and moduli (m)

As part of our final investigation, we set $\delta = 1$ to compare our proposed design with the diminished one modulo-$(2^n + 1)$ adder introduced in [35], widely regarded as one of the most efficient designs in the field. The code for this adder is publicly available on https://iis-people.ee.ethz.ch/~zimmi/. While the results indicate that our proposed design does not surpass the performance of the diminished one modulo-$(2^n + 1)$ adder, the differences are minimal. This demonstrates that our design achieves competitive performance. It is worth noting that our proposed design is highly versatile and can operate without restrictions on δ, a significant advantage over certain generic adders that fail to support $\delta = 1$ [12].

VI. CONCLUSOIN

This paper presents a novel algorithm for efficient modular addition in RNS, targeting moduli of the form $(2^n + \delta)$, where δ is an integer within $0 \leq \delta \leq 2^{n-1} - 1$. By leveraging a two-valued digit (twit) encoding for $\pm\delta$, the proposed approach achieves significant improvements in delay, area, and power consumption compared to prior designs. The introduction of twit-based residue encoding not only simplifies modular reduction but also eliminates the need for multiple CPAs and multiplexers typically required in generic modular adders. The analytical and experimental results demonstrate the efficacy of the proposed design across practical n ($4 \leq n \leq 16$). Synthesis using the FreePDK 45nm process demonstrates that, compared with the best-performing components from referenced works, the proposed design achieves average improvements of 21% in delay, 20% in area, 14% in power consumption, and 31% in PDP.

Additionally, the fault-tolerant capabilities enabled by RMAs enhance the reliability of RNS processors while reducing hardware redundancy. By addressing the challenges of dynamic range and operating frequency, this work contributes a robust and efficient solution for modular arithmetic in modern computing systems.

ACKNOWLEDGMENT

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2023-00228970, Development of Flexible SW·HW Conjunctive Solution for On-edge Self-supervised Learning, 30%), IITP grant funded by the Korea government (MSIT) (RS-2019-II190421, Artificial Intelligence Graduate School Program (Sungkyunkwan University), 30%), and the Technology Innovation Program (or Industrial Strategic Technology Development Program-Public-Private Joint Investment Advanced Semiconductor Talent Development Project) (RS-2023-00237136, Development of CXL/DDR5-based memory subsystem for AI accelerators, 40%).

REFERENCES

[1] Z. Ahmadpour and G. Jaberipur, "Up to 8k-bit Modular Montgomery Multiplication in Residue Number Systems with Fast 16-bit Residue Channels," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1399-1410, Jun. 2021.

[2] M. A. Bayoumi, G. A. Jullien, and W. C. Miller, "A VLSI Implementation of Residue Adders," *IEEE Transactions on Circuits and Systems*, vol. 34, no. 3, pp. 284–8, Mar. 1987.

[3] G. Dimitrakopoulos, D.G. Nikolos, D. Nikolos, H.T. Vergos, and C. Efstathiou, "New Architectures for Modulo $2^n - 1$ Adders," in *Proceedings of IEEE International Conference on Electronics, Circuits, and Systems*, Sep. 2008.

[4] M. Dugdale, "VLSI Implementation of Residue Adders Based on Binary Adders," *IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, pp. 325–329, Mar. 1992.

[5] C. Efstathiou, H.T. Vergos, and D. Nikolos, "Fast Parallel-Prefix Modulo-$(2^n + 1)$ Adders," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1211-1216, Sep. 2004.

[6] H. Fatemi, and G. Jaberipur, "Double {0,1,2} Representation Modulo-$(2^n - 3)$ Adders," in *Proceedings of 21st International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 119-122, May 2014.

[7] J. Garland and D. Gregg, "Low Complexity Multiply-Accumulate Units for Convolutional Neural Networks with Weight-Sharing," *ACM Transactions on Architecture and Code Optimization.*, vol. 15, no. 3, p. 31:1-31:24, Sep. 2018.

[8] T. Gupta and S. Akhter, "Design and Implementation of Area-Power Efficient Generic Modular Adder using Flagged Prefix Addition Approach," in *7th International Conference on Signal Processing and Communication (ICSC)*, pp. 302–307, Nov. 2021.

[9] T. Gupta, G. Verma, and S. Akhter, "FPGA Implementation and Performance Analysis of Parallel Prefix Structures for Modular Adders Design," *Circuits, Systems, and Signal Processing*, Sep. 2024.

[10] A. Hiasat, "High-Speed and Reduced-Area Modular Adder Structures for RNS," *IEEE Transactions on Computers*, vol. 51, no. 1, pp. 84-89, Jan. 2002.

[11] A. Hiasat, "A Suggestion for a Fast Residue Multiplier for a Family of Moduli of the form $(2^n - (2^p \pm 1))$," *The Computer Journal*, vol. 47, no. 1, pp. 93–102, Jan. 2004.

[12] A. Hiasat, "General Modular Adder Designs for Residue Number System Applications," *IET Circuits, Devices & Systems*, vol. 12, pp. 424-431, Mar. 2018.

[13] G. Jaberipur, B. Parhami, and M. Ghodsi, "Weighted two-valued digit-set encodings: unifying efficient hardware representation schemes for redundant number systems," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 7, pp. 1348-1357, Jul. 2005.

[14] G. Jaberipur and B. Parhami, "Unified Approach to the Design of Modulo-$(2^n \pm 1)$ Adders Based on Signed-LSB Representation of Residues," in *Proceedings of 19th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 57-64, 2009.

[15] G. Jaberipur and S. Nejati, "Balanced Minimal Latency RNS Addition for Moduli Set $\{2^{n-1}, 2^n, 2^{n+1}\}$," in *Proceedings of 18th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 159-165, Jun. 2011.

[16] G. Jaberipur and SM. Cherati, "Modulo-$(2^n + 3)$ Parallel Prefix Addition via Diminished-3 Representation of Residues," in *Proceedings of IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp. 135-142, Jun. 2019.

[17] L. Kalamboukas, D. Nikolos, C. Efstathiou, H.T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers,* Vol. 49, No. 7, special issue on computer arithmetic, pp. 673-680, Jul. 2000.

[18] S. Ma, J. H. Hu, and C. H. Wang, "A Novel Modulo $2^n - 2^k - 1$ Adder for Residue Number System," *IEEE Transactions Circuits and Systems I: Regular Papers*, vol. 60, pp. 2962– 2972, May 2013.

[19] P. M. Matutino, H. Pettenghi, and R. Chaves, "RNS Arithmetic Units for Modulo $(2^n \pm k)$," in *Proceedings of Euromicro Conference on Digital System Design*, pp. 795–802, Sep. 2012.

[20] P. V. A. Mohan, *Residue Number Systems*. Springer International Publishing, Oct. 2016.

[21] N. N. Nagornov, P. A. Lyakhov, M. V. Valueva and M. V. Bergerman, "RNS-Based FPGA Accelerators for High-Quality 3D Medical Image Wavelet Processing Using Scaled Filter Coefficients," *IEEE Access*, vol. 10, pp. 19215-19231, Feb. 2022.

[22] B. Parhami, "Double-least-significant-bits 2's-complement number representation scheme with bitwise complementation and symmetric range," *IET circuits, devices & systems*, vol. 2, pp. 179-186, 2008.

[23] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs,* 2nd edition, Oxford University Press, New York, 2010.

[24] R. A. Patel, M. Benaissa, N. Powell, and S. Boussakta, "Novel Power-Delay-Area-Efficient Approach to Generic Modular Addition," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 6, pp. 1279–1292, Jun. 2007.

[25] H. Pettenghi, S. Cotofana, and L. Sousa, "Efficient Method for Designing Modulo $(2^n \pm k)$ Multipliers," *Journal of Circuits, Systems and Computers*, vol. 23, no. 1, pp. 1–20, 2014.

[26] J. Ramírez, A. García, U. Meyer-Baese, and A. Lloris, "Fast RNS FPL-based Communications Receiver Design and Implementation," *in Proceedings of Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, vol. 2438, 2002.

[27] N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Res-DNN: A Residue Number System-Based DNN Accelerator Unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 658-671, Feb. 2020.

[28] T. F. Tay and C.-H. Chang, "A New Unified Modular Adder/Subtractor for Arbitrary Moduli," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 53–56, May. 2015.

[29] M. V. Valueva, N. N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov, "Application of the Residue Number System to Reduce Hardware Costs of the Convolutional Neural Network Implementation," *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, Nov. 2020.

[30] H.T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-One Modulo-$(2^n + 1)$ Adder Design," *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1389-1399, Dec. 2002.

[31] H. T. Vergos, "On the Design of Efficient Modular Adders," *Journal of Circuits, Systems and Computers*, vol. 14, no. 5, pp. 965–972, Oct. 2005.

[32] H. T. Vergos, "Fast Modulo $2^n + 1$ Adder Architectures," *in Proceedings of 22nd Conference on Design on Circuits and Integrated Systems*, pp. 476–481, 2007.

[33] E. Vassalos, D. Bakalis, and H. T. Vergos, "On the Design of Modulo $2^n \pm 1$ Subtractors and Adders/Subtractors," *Circuits, Systems and Signal Processing*, vol. 30, pp. 1445–1461, Dec. 2011.

[34] H. T. Vergos and G. Dimitrakopoulos, "On Modulo $2^n + 1$ Adder Design," *IEEE Transactions on Computers*, vol. 61, no. 2, pp. 173-186, Feb. 2012.

[35] R. Zimmermann, "Efficient VLSI Implementation of Modulo $2^n + 1$ Addition and Multiplication," in *Proceedings of 14th IEEE Symposium on Computer Arithmetic (ARITH),* pp. 158-167, Apr. 1999.