A Generalized Residue Number System Design Approach for Ultralow-Power Arithmetic Circuits Based on Deterministic Bit-Streams

Kamyar Givaki, Ahmad Khonsari, M. H. Gholamrezaei, Saeid Gorgin, *Senior Member, IEEE*, and M. Hassan Najafi, *Member, IEEE*

***Abstract*—The peak power consumption has become an important concern in the hardware design process of some of today's applications, such as energy harvesting (EH) and bio-implantable (BI) electronic devices. The limited peak harvested power in EH devices and heating concerns in BI devices are the main reasons for power control's importance in these devices. This article proposes a generalized design approach for ultralow-power arithmetic circuits. The proposed circuits are based on residue number system (RNS) combined with deterministic bit-streams. The resulting circuits can be used in systems with a restricted power budget. We suggest several approaches to design generic hardware-efficient adders, multipliers, multiply–accumulate (MAC) unit, forward converters (FCs), and reverse converters (RCs). Using the proposed approach, designing these components for any moduli of the RNS can be performed through simple bit-width adjustments in the circuits. The synthesis results show that the proposed adder achieves, on average, 69% and 2% lower area compared to the bit-serial and a state-of-the-art RNS adder, respectively. Furthermore, the proposed multiplier outperforms the bit-serial, interleaved, and a state-of-the-art design for multiplying RNS numbers by, on average, 57%, 60%, and 77% in terms of power consumption, respectively. The efficiency of our approach is shown via two essential applications, digital signal processing, and machine learning. We implement an FFT engine using the proposed method. Compared to prior RNS implementations, our design achieves 47% lower power consumption. We also implement a CNN accelerator's processing element (PE) with the proposed computation elements. Our design provides considerable speedup and lower power consumption compared to a state-of-the-art ultralower-power design.**

***Index Terms*—Addition, forward conversion, low power, multiplication, residue number system (RNS), reverse conversion.**

Manuscript received 12 September 2022; revised 17 December 2022; accepted 10 February 2023. Date of publication 1 March 2023; date of current version 20 October 2023. This work was supported in part by the National Science Foundation (NSF) under Grant 2019511; in part by the Louisiana Board of Regents Support Fund under Grant LEQSF(2020-23)-RD-A-26; in part by the Generous Gifts from Cisco; in part by Xilinx; and in part by Nvidia. This article was recommended by Associate Editor I. O'Connor. *(Corresponding author: Ahmad Khonsari.)*

Kamyar Givaki is with the School of Electrical and Computer Engineering, University of Tehran, Tehran 1417935840, Iran (e-mail: givakik@ut.ac.ir).

Ahmad Khonsari is with the School of Electrical and Computer Engineering, University of Tehran, Tehran 1417935840, Iran, and also with the School of Computer Sciences, Institute for Research in Fundamental Sciences, Tehran 1953833511, Iran (e-mail: a_khonsari@ut.ac.ir).

M. H. Gholamrezaei and Saeid Gorgin are with the Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea (e-mail: gholamrezaei@chosun.kr; gorgin@chosun.ac.kr).

M. Hassan Najafi is with the School of Computing and Informatics, University of Louisiana, Lafayette, LA 70503 USA (e-mail: najafi@louisiana.edu).

Digital Object Identifier 10.1109/TCAD.2023.3250603

I. Introduction

In ENERGY harvesting (EH) devices, the required energy is obtained from the environment by inertial kinetic, electromagnetic, and thermoelectric technologies. The amount of the harvested power (energy in the time unit) in these devices (regardless of the employed technology) is critically limited, which constraints the application's peak power consumption. Consequently, there is a tradeoff between the functionality of the circuit and the size of the harvesting part of the EH devices [1], [2], [3]. Designing ultralow-power hardware circuits is critical to addressing the current challenges of using EH devices. In addition to the peak power problem, the generated heat is another important concern in many bio-implantable (BI) devices which arise the need for ultralow-power computing schemes [4]. Residue number system (RNS) is a number system that offers low-power and low-area addition, subtraction, and multiplication compared to conventional binary arithmetic [5], [6]. These make RNS an attractive solution for BI devices. RNS has been used in various real-world applications, from digital signal processing to machine learning, since the 1960s [7], [8], [9], [10], [11]. An RNS system is represented by its *moduli set*, which is defined as a set of L pairwise relatively prime positive integers. In this representation, a number X is represented by L smaller numbers with fewer bit-width that are the remainders of X when divided by the members of the moduli set [5], [12]. The lower bit-width results in lower processing time since the critical path of the arithmetic circuits is directly affected by the computations' bit-width. Converting a number represented in the conventional binary system to its RNS equivalent is called *forward conversion*. A number in the RNS format is not easily interpretable. A *reverse conversion* technique is used to convert the RNS representation back to its binary equivalent. Choosing an optimum moduli set is an important challenge when employing RNS in a system. The selected moduli set significantly impacts the representation efficiency and the complexity of the computation and conversion circuits [5]. A more complex system often consumes more power consumption. The effects of each stage on the complexity of an RNS system are described as follows.

1) *Forward Conversion:* Computing the remainder of a number when divided by other numbers (modulus) can be challenging and requires a significant effort to implement in hardware. Although several generalized forward conversion methods are proposed in the literature, the existing methods are power- and area-hungry for most

moduli. Therefore, designing hardware-efficient generalized forward converters (FCs) is an ongoing research.

2) *Modular Arithmetic:* Computational elements are the most sensitive parts of an RNS system to the selected moduli set. Since RNS computations (i.e., additions and multiplications) are performed in a modular manner, designing adders(subtractor) and multipliers for some moduli requires massive design efforts, and the outcome may not be power- or area-efficient. Furthermore, a moduli set contains several members; the modular computations for each member are performed with different critical path delays. Due to the dissimilarity in the critical path delay of different moduli, the modulus with the longest critical path delay (the slowest) controls the system's execution time. Therefore, selecting an efficient moduli set is critical for any RNS-based computational system. Developing a simple design strategy that allows circuit design of modular operations regardless of the selected modulus and guarantees the same critical path for modulus with the same bit-width can help wide use of RNS in cost-efficient design of complex systems. For example, $(2^n-1,\ 2^n,\ 2^n+1)$ is a moduli set frequently used in literature due to its simplicity in designing the computation and conversion circuits.

3) *Reverse Conversion:* Conventionally, chinese remainder theorem (CRT), and mixed-radix conversion are used to convert a number from RNS back to the conventional binary format [5]. Many hardware methods based on these schemes have been proposed [13], [14], [15]. Most of these techniques are not area- and power-efficient for all different moduli sets, limiting the use of RNS. Therefore, designing generalized reverse converters (RCs) with low power demand can also play a critical role in growing the use of RNS in today's commercial products.

This work considers these three stages and proposes a novel methodology that offers a simple design flow for power-efficient forward conversion, computation, and reverse conversion circuits for *any* modulus in *any* RNS system. We alter the conventional binary representation of each remainder to *a uniform bit-stream representation*, which makes the proposed design hardware efficient.

Our proposed circuit for a multiplication operation receives its residues inputs in binary and converts them into bit-stream format using a modified version of the approaches proposed in [16] and [17]. In this approach, the two binary operands of the multiplication operation are first encoded to two bit-streams. A clock-division technique [18] is then applied to the two generated bit-streams to make them uncorrelated. Next, the multiplication operation is performed by bitwise ANDing the two bit-streams. A counter at the last stage is responsible for converting back the output bit-stream to binary format. The method offers a significant reduction in power consumption and area occupation, though processing data in redundant bit-stream format often results in higher processing latency and energy consumption compared to pure binary approaches.

In summary, the main contributions of this work are as follows.

1) We propose a new bit-stream generator that converts binary numbers (residues) into their bit-stream equivalents. Using this new bit-stream generator with our previously proposed RC and the modular multiplier, the length of the required bit-streams reduces significantly, resulting in a considerable reduction in latency and energy consumption. Moreover, the proposed bit-stream generator has a lower hardware footprint than its predecessors.
2) To the best of our knowledge, we propose the first generalized and efficient RNS adder (subtractor), FC, and some fused operation units by exploiting bit-stream processing. Our designs can be used for any modulus with only some minor adjustments. These adjustments are applied to support moduli with different bit-widths. The proposed adder (subtractor) is more hardware efficient than the current RNS adders/subtractors. Moreover, by adding a few logic gates, the proposed modular adder (subtractor) acts as a simultaneous modular adder and modular subtractor. Furthermore, the proposed multiplier is modified slightly to compute some fused operations like $(A\times B+C)$ and $(A\times B+C\times D)$ with no additional hardware cost. Similar to the other proposed hardware component, the proposed RC and the modular multiplier can be easily adjusted to support any modulus with any bit-width.
3) Our experiments on several binary bit-widths with different moduli sets show that higher cardinality of the moduli set can make the proposed architecture more capable of being used in real-world applications. The higher cardinality of the moduli set can be directly translated to lower execution time and energy consumption. Furthermore, our experiments show that the proposed approach offers an ultralow-power computational scheme by combining RNS and deterministic bit-stream computing [19], which suits applications with ultralow-power demands, such as EH and BI devices. Furthermore, we implemented two real-world applications (a neural network processing element (PE) and a fast Fourier transform (FFT) engine) using the proposed hardware components. Our evaluations show the proposed components' effectiveness in implementing real-world applications.

The remainder of this article is organized as follows. Section II discusses the necessary background on RNS and deterministic bit-stream computing, and describes the previous related works. Section III investigates the proposed methodology and provides a detailed insight into the micro-architecture of the proposed computation approach. Section IV provides the results and compares the proposed method with prior works. Finally, Section V concludes this article.

II. Preliminaries and Related Contributions

A. RNS in Brief

RNS is a computational system that represents numbers by their residues when divided by the members of a set of integer numbers (called *Moduli set*). The set consists of several relatively prime integers, each called a *modulus*. The cardinality of the set equals L, $(m_1, m_2, \ldots, m_L)$. Using this system a number X is represented by a set of L other numbers, $(x_1, x_2, \ldots, x_L)$ where $x_i = X \bmod m_i$. Generally, each $x_i, i = 1\cdots L$, is significantly smaller than the original number, X. Consequently, residues need a lower bit-width to be represented compared

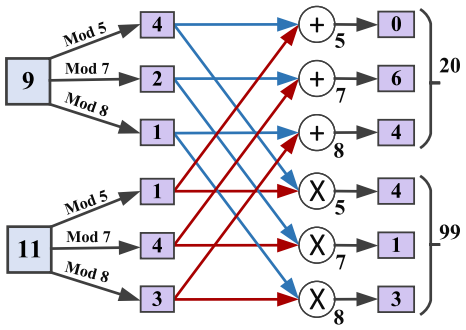


Fig. 1. Two-input addition and a two-input multiplication using RNS.

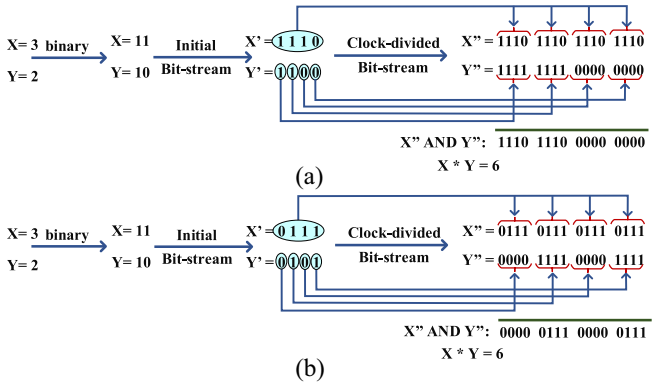


Fig. 2. Example of bit-stream generation and multiplication using the (a) method in [16] and (b) one possible way of the method in [17].

to the original value, X. Therefore, when using RNS, complex operations, such as high-bit-width multiplication, can be split into simpler operations with lower bit-width running in parallel [5], [12]. The number of unique values that can be represented using a specific RNS is called the *Dynamic Range* of that system and is computed by multiplying all the members of the moduli set, i.e., $D = m_1 \times m_2 \times \cdots \times m_L$.

Fig. 1 exemplifies the process of addition and multiplication in RNS. In this example, the main operation is split into three parallel suboperations. Each suboperation needs a 3-bit adder or multiplier, while the original multiplication requires 7-bit, and the original addition requires 5-bit computation circuit. The results of the subcomputations can exceed the highest possible value for the modulus. For example, $2 \times 4 = 8$ is greater than the highest possible value for $m_2 = 7$. In this case, the result should be changed to 1. This shows that the operations, here addition and multiplication, in an RNS system should be modular. This is one of the main challenges in designing efficient circuits for RNS systems. This work presents a novel approach for designing modular multipliers and adders for RNS-based systems.

B. Forward Conversion

Several techniques have been proposed to convert a binary number to its RNS equivalent. Many of these methods need read-only memories (ROMs) to store the residues of power 2 s when divided by a modulus [20]. This increases the required area and power cost to perform the conversion. Many optimizations have been proposed to mitigate the hardware costs of these techniques. But these still cannot satisfy the area and power budget of ultralow-power systems. Some other methods try to generate the residues using arithmetic functions instead of precalculated values stored in ROMs. Designing these functions for some moduli involves a complex process, and the result is not hardware efficient [21], [22]. As several channels exist in an RNS system, some techniques are proposed to share some of the hardware between different channels to save area. But these methods are only available for a few moduli such as $(2^n \pm 1)$ [23], [24].

C. Addition and Multiplication

Modular adders are investigated in prior works. Many arithmetic techniques are employed to design efficient modular adders [25], [26], [27], [28]. One-hot coding [29] and thermometer coding [30] are employed in several works. General modular adders, however, are not as hardware-efficient as customized modular adders. The lack of such techniques raises the need to design hardware-efficient modular adders.

Many methods have also been proposed to perform modular multiplication, one of the main concerns when dealing with RNS. We can categorize the previously proposed modular multipliers into two categories: 1) general and 2) customized. Customized modulo multipliers target a specific modulus, for example $(2^n - 1, 2^n, 2^n + 1)$, and try to optimize the multiplier's hardware or timing properties only for that moduli set [31], [32], [33], [34]. The general modulo multipliers can be broken into two subcategories: 1) look-up table (LUT)-based and 2) arithmetic-based. The LUT-based methods are suitable when the moduli are small. When the binary bit-width of the operands is large, the LUT size increases in quadratic trend, making their implementation impractical [35], [36], [37]. When the moduli grow, it is necessary to use arithmetic-based methods.

In arithmetic-based methods, several combinational circuits, such as multipliers, adders, multiplexers, and in some cases, a few memory cells are responsible for computing the modular multiplication [38], [39], [40]. Bajard et al. [38] proposed a modular multiplication method based on the Montgomery modulo-reduction algorithm that uses an auxiliary base to handle multiplication in relatively large modules. Kornerup [41] proposed another method based on the Montgomery reduction algorithm. In the Montgomery reduction algorithm, the least significant digits of the partial products are used to determine the quotient digits, and shift-and-add is used as its reduction step [42]. Blakely [40] proposed a method that computes modular multiplication using a bit-serial method. Some other methods employ unconventional number codings to present more efficient RNS multiplications. For example, Cardarilli et al. [43] and Vun et al. [29] used one-hot coding in which a stream of several 0s and one 1 represents the value of the operand. Moreover, a multivoltage system is proposed in [44] that decreases the power consumption of an RNS system. This method computes the slowest RNS channel with a higher voltage.

We proposed a low-power bit-stream-based modular multiplier suited for RNS computations in [16]. For the multiplication of two n-bit residues, the design uses two n-bit binary counters and two comparators to generate bit-streams based on the clock division method of [18] and generates bit-streams as depicted in Fig. 2(a). The initial bit-stream in the

figure is obtained by comparing the input operands with the output of an n-bit up-counter in n cycles. If the counter's output is less than the input operand, a 1 bit is added to the bit-stream; otherwise, a 0 bit is added. The generated bit-streams are ANDed using an AND gate. The number of 1s in the output bit-stream is counted using a modular counter which resets when it reaches a specific state. The multiplication result is the counter's value after 2^{2n} cycles. We replaced the counters and comparators of the bit-stream generator with two identical finite state machines (FSMs) (with 2^n states) and two n-to-1 multiplexers (MUXs) in [17]. These modifications reduce power consumption and area occupation compared to the design of [16]; however, the number of processing cycles for both methods are similar. An example of a two-input multiplication using the method of [17] is illustrated in Fig. 2(b). The FSM used in [17] is based on the low-discrepancy (LD) sequences proposed in [45] and [46] to convert the residues to bit-stream representation.

D. Reverse Conversion

There is a large body of work in the literature to convert the numbers back from RNS to standard weighted binary representation. Many of these methods are developed based on CRT-I, CRT-II, CRT-III, and Mix-Radix conversion theorems [5]. Almost all these methods need multiplicative inverse that must be computed online or stored in LUT, making them inefficient for large moduli [47], [48]. Although several memoryless methods have been proposed, these methods are proposed only for specific moduli sets and cannot be generalized to all moduli sets [49]. Other methods, such as conversion using functions, need many logical elements (e.g., full adders) and LUTs, making them hardware inefficient [50], [51]. In general, the previously proposed RCs are not power- and area-efficient enough to be implemented in edge devices with a limited area and power budget.

We proposed a generalized RC that uses an LUT with 2^L rows (L is the number of moduli in the moduli set) in [17]. In that design, L n-bit residues are converted to L parallel 2^n-bit bit-streams. In each clock cycle, L parallel bits are used to select the corresponding row of the LUT. The outputs of the LUT in each clock cycle are modularly accumulated to calculate the binary equivalent of the input residues. Here, we also used the FSM proposed in [45] and [46] to convert the residues to bit-streams and the output of this method is prepared after 2^n clock cycles.

E. Bit-Stream Computing in Brief

Stochastic computing (SC) [19], [52], [53], [54] is an unconventional computing paradigm in which computations are performed in a probabilistic sense on uniform bit-streams. The probability of seeing a 1 in a stochastic bit-stream determines the value of the bit-stream. A unipolar number X in the [0, 1] interval is represented by a bit-stream in which the probability of seeing a 1 is X. For example, any bit-stream with 50% of 1 s is a representation for $X = 0.5$ [19], [55].

In SC, arithmetic operations, such as addition, multiplication, or more complex operations, such as exponentiation and hyperbolic tangent, can be performed by extremely simple logic gates [53], [54]. For example, multiplication as a power-hungry arithmetic operation in traditional binary computing is performed by using a single AND gate, scaled addition is conducted using an MUX unit.

A stream of 2^n bits can precisely represent an n-bit binary number. A binary to stochastic bit-stream converter consists of a register to hold the target number, a comparator, and a number generator, such as a counter or a linear feedback shift register (LFSR). The target number is loaded into the register, and the number generator generates a new number in each cycle. If the generated number is less than the target number, the comparator produces a 1; otherwise, a 0 is produced. The number generator can be a random number generator, such as an LFSR to generate a random bit-stream (e.g., 101011) or it can be an up- or down-counter to generate a unary bit-stream (e.g., 111100). Both random and unary bit-streams can be converted back to binary representation by using a counter to count the number of 1s in the bit-stream [55].

The inputs of the SC multiplier (AND gate) must be *uncorrelated*, which means zero average correlation between the two input bit-streams of the multiplier, to guarantee the correctness of the output [19], [54]. The same property is needed in scaled addition between the primary inputs and the select input of the MUX. The traditional SC [52] suffered from random fluctuations in the bit-streams and was unable to guarantee this zero correlation requirement. Recently, some deterministic methods of SC were proposed [18], [19], [56]. By restructuring bit-streams, these methods guarantee the needed zero correlation or independence and can provide high accuracy computations using SC logic. In traditional SC, the independence requirement between operand bit-streams was provided by using different random number sources when generating bit-streams. Instead, with the deterministic SC, this requirement is guaranteed by clock dividing bit-streams [18], rotation of bit-streams [18], or using bit-streams with relatively prime lengths [56]. With these methods, each bit of the first operand (bit-stream) sees each bit of the second operand (bit-stream) exactly once. The computation results are completely accurate, free of random fluctuations or correlation errors. More recently, a fast converging deterministic method of processing bit-streams was also proposed in [57] by generating LD bit-streams. All these deterministic methods guarantee completely accurate results if generating and processing bit-streams for *the product of the length of input bit-streams*. Therefore, accurate processing of two n-bit precision binary numbers requires generating and processing two 2^{2n}-bit bit-streams.

III. Proposed Method

This section proposes a novel approach for designing ultralow-power modular adders/subtractors and multipliers. The proposed architecture can be easily adjusted to perform modular operations for any arbitrary moduli. Our method offers a critical path delay (processing time) proportional to the bit-width of operands. Therefore, it is a solution to the unbalanced critical path delay of different moduli in conventional RNS systems. Unbalanced critical path delay in RNS channels can cause underutilization of some hardware components. In conventional RNS, hardware circuits for different moduli may have different critical path delays for each computing lane. For example, the critical path delay for a conventional modular multiplier for modulus 9 is lower than that for modulus 13; consequently, in an RNS system containing these two moduli,

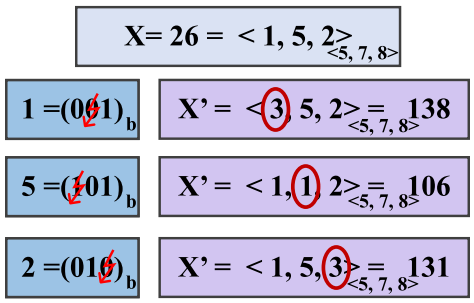


Fig. 3. Illustration of the RNS sensitivity of high to faults.

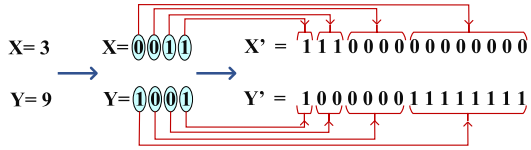


Fig. 4. Examples of the proposed bit-stream generation.

when the module 9 multiplier finishes its computation it stops until the other multiplier also finishes its computation. A similar circumstance may happen in the combination of RNS and SC. The proposed design guarantees to avoid this unbalanced situation if all the moduli in a moduli set have the same binary bit-width. We also propose a low-cost method to design the forward and RCs for use with the proposed computation elements. In the proposed architecture, we use the bit-stream logic to design an architecture capable of changing the modules efficiently. We first describe the process of converting inputs to their equivalent bit-streams. After describing the proposed FC, we explain our architecture for a modular adder(subtractor) and a modular multiplier. The proposed multiplier can also compute some fused operations. In the end, we discuss the RC. Compared to other SC-based designs, our proposed design produces an exact (completely accurate) output in lower clock cycles and with lower hardware cost at the cost of increasing the vulnerability to fault compared to conventional bit-stream-based designs because RNS computation is highly sensitive to faults. A single bit-flip or any inaccuracy in computations may result in high output error. Increasing or decreasing even one of the residues by one unit can cause a significant error in the RNS-based computation. Therefore, it is important to ensure all required cycles are passed before using the results. Fig. 3 illustrates this shortcoming via an example.

A. Bit-Stream Generation

Instead of representing residues in binary format, our method process residues in bit-stream format. The bit-stream generator iterates each bit of a binary number equal to its positional weight. Therefore, an n-bit residue is represented with a $(2^n - 1)$-bit bit-stream. Fig. 4 shows two example bit-streams generated with this approach. We implement the bit-stream generator using an FSM and an n-to-1 MUX. The FSM controls the MUX to pass the proper bit in each clock cycle.

B. Forward Conversion

Any RNS system requires a forward conversion unit to convert numbers from binary to RNS equivalent. We propose a low-cost generic converter that can be easily adjusted to perform the conversion of any binary number to residue format. The architecture consists of a bit-stream generator and a modular counter. The modular counter counts the number of 1s in its input until it reaches m_i and then resets. Fig. 5(a) shows the proposed architecture for the FC where A_i is the ith bit of the binary input A. Any RNS system has several channels. Assuming that the employed RNS system has L channels, L residues must be calculated to represent a binary number accurately. As Fig. 5(b) shows the bit-stream generator can be shared in calculating all L residues. In this scenario, only one bit-stream generator sends its output to L distinct modular counters. Each counter resets after seeing the maximum value of the module m_i, $i = 1, .., L$. Converting an n-bit binary number to its corresponding residues using the proposed circuit takes $(2^n - 1)$ clock cycles. To design a generic modular counter, the reset signal of a simple counter is connected to a comparator that compares the counter's latest state with the modulus register that stores the modulus value.

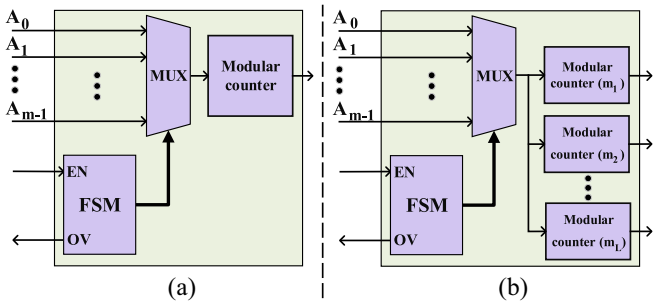


Fig. 5. Architecture of the proposed FC. (a) FC for a single modulus. (b) FC for a moduli set with L moduli.

C. Addition (Subtraction)

Fig. 6(a) shows the architecture of the proposed adder (subtractor) for one modulus. For an n-bit addition (subtraction), the proposed circuit consists of a bit-stream generator (an FSM and an n-to-1 MUX) and a modular up(down) counter. The architecture of the adder is similar to the FC except that the initial value of the adder's modular counter can be loaded in parallel at the beginning of the computations. Using this architecture the adder(subtractor) completes its calculations in $(2^n - 1)$ clock cycles where the input operands are n-bit binary numbers. For a moduli set with L moduli, L copies of the circuit of Fig. 6(a) are connected together. An RNS system always has more than one computing channel. Since the bit-stream generator consumes a high amount of power and occupies a vast area, we share it between all the RNS system channels as presented in Fig. 6(b). In this approach, the bit-width of the shared elements must be equal to the system's largest bit-width.

In what follows, we describe the calculation process of one channel of the proposed adder(subtractor) to calculate $A + (-)B$.

1) The first operand (A) is loaded into the modular counter, and the second operand (B) is sent to the MUX input of the bit-stream generator. Also, the FSM resets.
2) The bit-stream generator generates a bit-stream corresponding to the second operand (B).

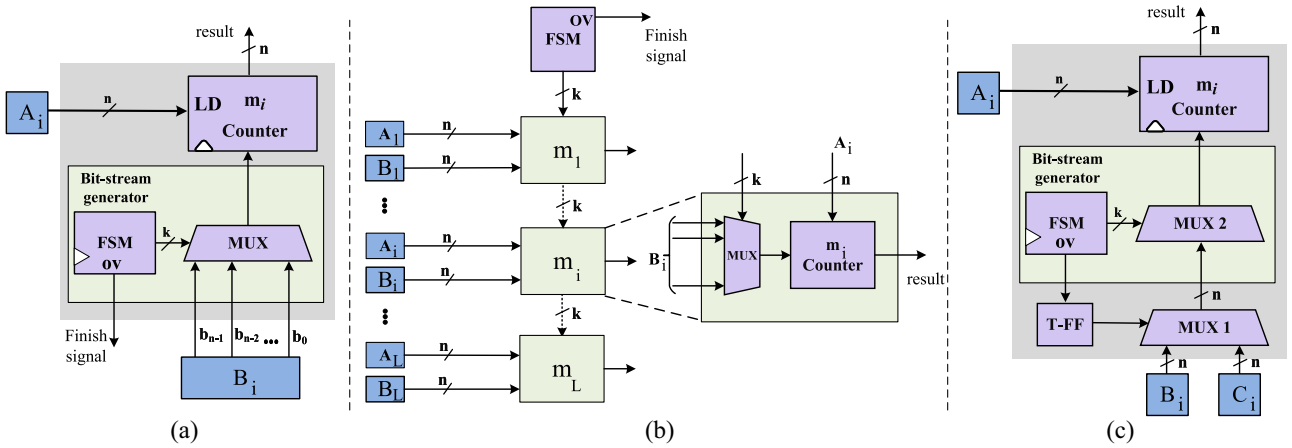


Fig. 6. Block diagram of the microarchitecture of (a) proposed adder for one modulus; (b) proposed adder for a moduli set with a shared counter; and (c) proposed multioperand adder for one modulus.

3) In each clock cycle, the modular counter counts up for addition or counts down for subtraction if a 1 is generated in the bit-stream.
4) For addition operation, the modular counter resets to zero when its value exceeds the greatest possible value in its corresponding modulus.
5) For subtraction operation, when the counter's value reaches zero, the next 1 in the MUX's output sets the value of the modular counter to the greatest possible value in its corresponding modulus.
6) The calculation completes when the overflow signal of the bit-stream generator's FSM turns 1.

The proposed architecture can be modified to perform multioperand addition. Fig. 6(c) shows the microarchitecture of a three-input adder for $(A + B + C)$. This circuit includes an n-bit 2-to-1 MUX and a T-flip-flop in addition to the proposed two-input adder. The T-flip-flop is shared between all channels. Similar to the two-input adder, the first operand (A) is loaded to the modular counter at the beginning of the computations. Then, input B is fed into the bit-stream generator using MUX 1 in Fig. 6(c).

In the end, the T-flip flop changes the select input of the MUX in the first layer and feeds C to the bit-stream generator. This architecture can be extended to add more than three inputs by changing the MUXs in the first layer and the logic for selecting its inputs. In a generalized view, the system completes an S-input addition in $(S-1) \times (2^n - 1)$ clock cycles. We note that for these addition and subtraction operations, the moduli set should be selected such that the values remain in the dynamic range of the moduli set.

It is worth mentioning that the proposed adder (subtractor) is kept simple to be easily modified to support different modular operations. It can be used individually in pure RNS systems where designing modular adders (subtractors) needs massive effort. Moreover, the proposed design can perform simultaneous addition and subtraction using an up/down counter with a negligible hardware overhead. Consequently, only one hardware component is required for modular addition and modular subtraction, which decreases the hardware cost compared to the case that modular adder and modular subtractor are separated components. The proposed adder (subtractor) is a proof of concept that shows how we can modify the multiplier proposed in Section III-D to perform other arithmetic operations efficiently. We use this concept to design the proposed multiply–accumulate (MAC) and fused operation units.

D. Multiplication

Fig. 7(a) shows the microarchitecture of the proposed multiplier. Each multiplier has two bit-stream generators, an AND gate, several registers, a comparator, and a counter. The inputs are two residue numbers in binary format. The two bit-stream generators are connected in a cascaded fashion based on the clock division method of [18] and convert the inputs into two deterministic bit-streams. Note that our method is not limited to the clock division method and can be used with other deterministic methods of SC [19]. We choose the clock division approach here because sequence generation using this technique is simple and low cost.

The generated bit-streams are bit-wise ANDed using the AND gate. The output of the AND operation is connected to the modular counter in the last layer. This counter holds the binary RNS result, and its value is ready and credible (contains accurate result) in the last cycle of the computations. Using this architecture, it is sufficient to adjust the bit-width of the registers, comparator, MUXs, bit-stream generators, and the modular counter at the last layer to support any other modules.

For an accurate multiplication, each bit of the bit-stream corresponding to the first input must see all bits of bit-stream corresponding to the second operand exactly once. The clock division method [18] guarantees this interaction by repeating each bit of the second bit-stream for the length of the first bit-stream. We cascade the bit-stream generators to generate such bit-streams. In what follows, we elaborate on this process.

1) First, both bit-stream generators are reset.
2) The first bit-stream generator is always enabled and generates the bits of the bit-stream corresponding to the value of the first operand. When one round of bit-stream generation is complete, the overflow (OV) signal is enabled. The bit-stream generator then resets and regenerates the same bit-stream. Throughout each round of generating the first bit-stream, the second bit-stream generator stays in the same state, and its output bit does not change. In each clock cycle, the outputs of the

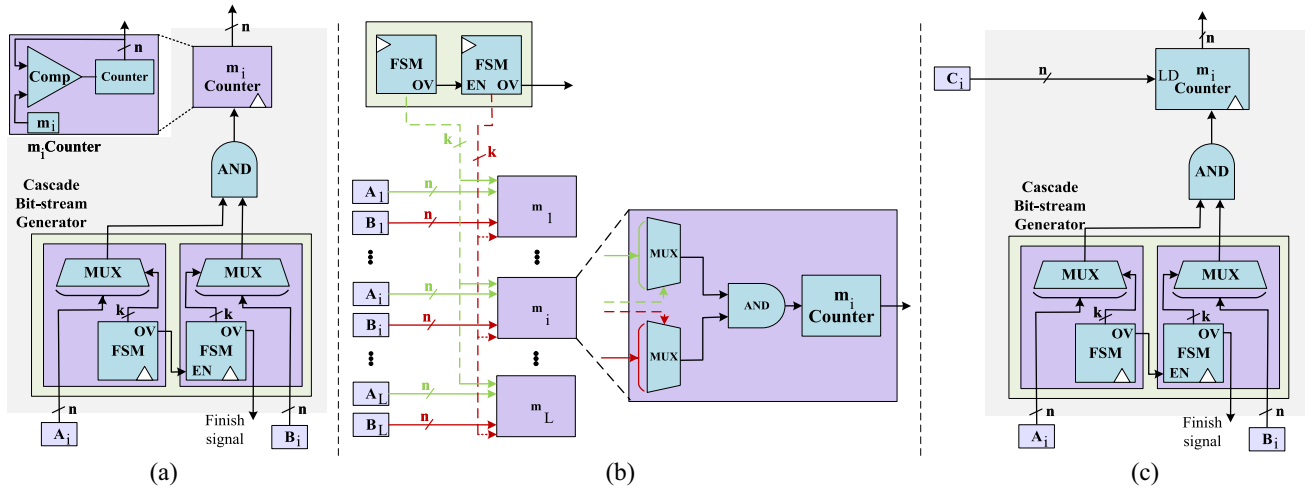


Fig. 7. Block diagram of the microarchitecture of (a) proposed multiplier; (b) proposed multiplier for a moduli set with shared counters; and (c) processing parts of the proposed MAC unit.

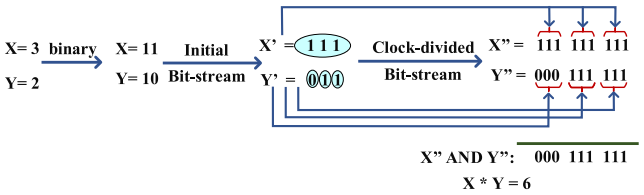


Fig. 8. Example of multiplication using the proposed multiplier for one channel (the effect of the modular counter is eliminated for the sake of simplicity).

two generators are ANDed together. The modular counter counts up if the output of the AND gate is 1.

3) When the first generator completes one round, its overflow (OV) signal enables the second generator's enable (En) pin. Consequently, the second generator goes forward and generates its next bit.
4) This cycle continues until the overflow pin of the second generator turns 1.

Fig. 8 shows an example of multiplying two 2-bit numbers by converting them to bit-stream representation and extending the bit-streams using the clock division method.

Similar to addition, there is more than one computing channel in the multiplier. The two bit-stream generators contribute a significant portion of the proposed circuits' hardware cost. Consequently, we share the FSMs between all of the bit-stream generators in all RNS channels. Fig. 7(b) illustrates the architecture with shared components.

The input operands of the RNS multiplier are n-bit binary numbers and the bit-stream corresponding to each input operand is $(2^n - 1)$ bits. Each bit from the first operand bit-stream must see all bits of the second bit-stream exactly once. Since each bit of input bit-streams is processed at one clock cycle and we use the clock division method [18], a precise multiplication takes $(2^{2n} - 2^{(n+1)} + 1)$ clock cycles which is less than 2^{2n}, the required number of clock cycles with the design of [16] and [17]. Generally, residues used in RNS-based calculation have low bit-width; therefore, the proposed design mitigates the processing time compared to non-RNS standard bit-stream-based multiplication. We note that using moduli set with higher cardinality usually decreases the residues' bit-width, and our proposed architecture supports computations using any modulus.

The architecture is the same for any modulus with the same bit-width. Therefore, by choosing moduli with the same bit-width, we can avoid asymmetry (unbalanced execution time) in RNS channels, an important concern in designing RNS systems. The unbalanced execution time has a more severe impact when combining RNS and bit-stream processing than conventional RNS. For example, if one channel needs 3 bits (requires 49 cycles to complete calculations) and the other channel needs 5 bits (requires 961 cycles to complete computations), the hardware components for 3-bit computations are not used for the remaining cycles of 912 cycles, which can cause a massive underutilization. In other words, asymmetry does not affect the final result but dramatically decreases the proposed design's hardware efficiency.

E. Fused Operations

MAC operation $(A \times B + C)$ is widely used in today's real-world applications, such as FFT and neural network computations. Conventional binary-based MAC design is complex and consumes massive power and area. Our proposed multiplier is capable of executing the operation without any additional hardware in the datapath (with a minor change in the control unit). If the residues corresponding to input C (i.e., C_is) are loaded to the last layer modular counters of their corresponding channels, $(A \times B + C)$ can be calculated with no additional hardware cost compared to the proposed multiplier. The proposed hardware for MAC operation is shown in Fig. 7(c).

Moreover, the proposed multiplier is able to calculate $(A \times B + C \times D)$. In the proposed multiplier in Section III-D, the last layer modular counter is initiated to zero before starting the multiplication process. If the output modular counter is not reset before the multiplication, the result of the ongoing multiplication is added to the previous result stored in the output register. Therefore, by a simple modification to the circuit's control unit, we achieve a low-cost design for $(A \times B + C \times D)$.

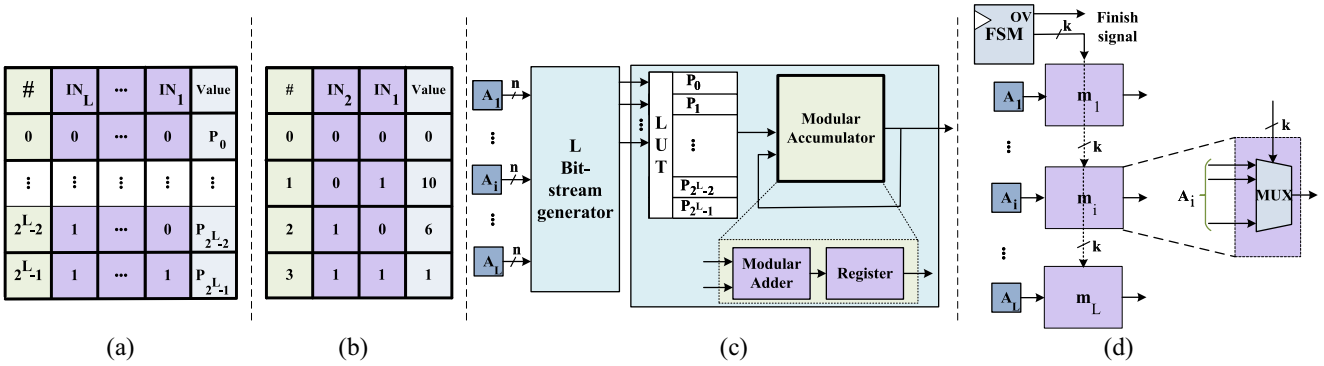


Fig. 9. Proposed RC, (a) LUT contents for the proposed RC; (b) example of LUT contents for $L = 2$; (c) microarchitecture of the proposed RC; and (d) structure of the bit-stream generator.

F. Reverse Conversion

An RC is needed to convert the final results back from RNS to standard binary format. The hardware overhead of such a converter is relatively high. However, in the cases of having several consecutive operations, it is possible to postpone the conversion to the last layer of the operations. As discussed in Section II, there are several methods to perform reverse conversion; however, none of them is suitable for the proposed adder and multiplier as our designs first need a conversion from bit-stream-based RNS to binary-based RNS.

We recently proposed a reverse conversion method to directly convert bit-stream-based RNS to its equivalent binary representation [17]. The RC needs a small LUT. If the cardinality of the selected moduli set is L, the proposed method needs a LUT with 2^L entries. The LUT contains P_is described by

$$P_i = \mathcal{P}(m_1, \ldots, m_L). \tag{1}$$

The inputs of the $\mathcal{P}$ function are bits of bit-streams resulting from converting input residues to bit-stream representation. In any clock cycle, each input can only get two different values. Therefore, 2^L different values need to be stored in the LUT. P_i shows the reversed value of the vector $\mathbf{V}$, where the members of the vector are the bits in the binary representation of i. Since the number of P_is is relatively small, it is possible to calculate P_is offline. For example, if the moduli set is $<8, 7, 5>$, P_5 shows the binary value of an RNS number that its representation is $<1, 0, 1>_{<8,7,5>}$. As illustrated in Fig. 9(a), each entry of the LUT contains the equivalent binary value of the corresponding combination of its inputs. Fig. 9(b) shows an example when the moduli set has two members (e.g., 5 and 3). The first entry shows the equivalent binary value for the RNS number $<0, 0>_{<5,3>}$. The second entry shows the equivalent binary value for the RNS number $<0, 1>_{<5,3>}$, and so on.

Fig. 9(c) shows the microarchitecture of the proposed RC. The design contains a bit-stream generator, an LUT, and a modulo-D accumulator, where D is the dynamic range of the moduli set. The modulo-D accumulator has a modulo-D adder and a register. The structure of the bit-stream generator is illustrated in Fig. 9(d). At the beginning of the process, the counter in the bit-stream generator and the accumulator value is reset. In each clock cycle, based on the value of the input, a bit-vector V is generated. The value corresponding to vector V is read from the LUT and is sent to the modulo accumulator. The circuit iterates for 2^m cycles, where m is the bit-width of the residues. The accumulated value after these cycles is the final output. To prevent the result from exceeding the dynamic range, the accumulation must be done in the modulo of the moduli set's dynamic range.

The proposed RC can be directly connected to the output bit-streams from the proposed multiplier and adder. In this setup, the output counter of the multiplier and adder are eliminated from their designs. Also, the bit-stream generator in Fig. 9(c) is eliminated, and the converter is directly connected to the bit-stream output of the multiplier/adder. This saves a significant area and power consumption by merging the multiplier/adder with the RC.

IV. Experimental Results

We evaluate the efficiency of the proposed designs by comparing them with prior implementations. The proposed RNS adder is evaluated by comparing it with three prior RNS addition methods for 8- to 64-bit binary input operands. The first method is based on the thermometer coding [30]. The second method is a bit-serial implementation of the modular adder. In this design, we replaced the parallel adder in a generic modular adder [50] with a bit-serial counterpart. Since modular adders for a customized moduli set are more power- and area-efficient than the generalized ones, we also compare our proposed adder with a customized adder to show the effectiveness of the proposed method in terms of area occupation and power consumption. Therefore, the third method is a high-performance customized adder for moduli set $(2^n - 1, 2^n, 2^n + 1)$ [31].

The proposed RNS multiplier is also compared with five prior designs for 8- to 64-bit binary input operands. These methods are the method of [31], the interleaved method of [40], a modified fully bit-serial version of the interleaved method [40], and our previous bit-stream-based methods of [16] and [17]. The moduli set $(2^n - 3, 2^n - 1, 2^n)$ is used to evaluate the proposed and all prior methods except the method of [31]. Zimmermann [31] proposed a customized design for the moduli set $(2^n - 1, 2^n, 2^n + 1)$. The moduli set selected in [31] is the most popular moduli set in literature, and their proposed method is more efficient than the methods with generic moduli sets. Therefore, if our design outperforms their method, we can expect to outperform prior methods for other moduli sets.

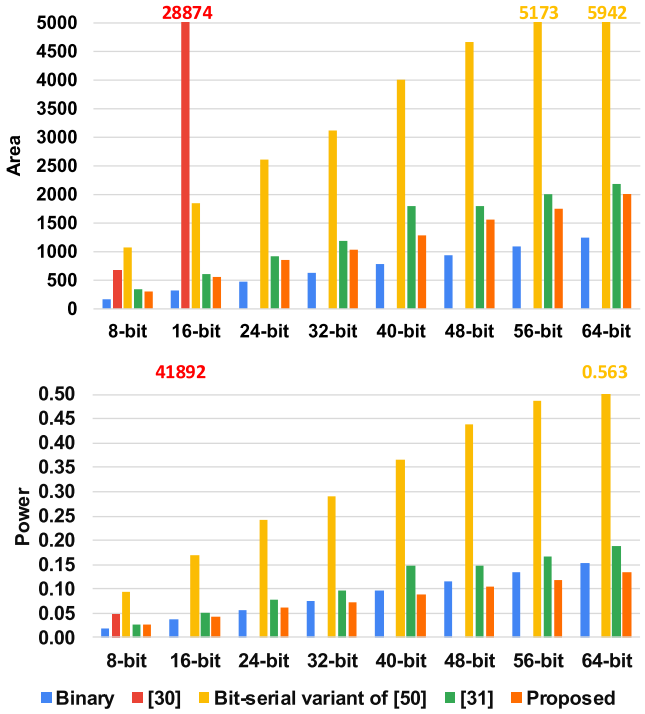


Fig. 10. Synthesis results for two input **addition** in 45-nm technology. The results are shown for 8- to 64-bit binary operands. (a) Area occupation (μm^2) and (b) power consumption (mW).

Next, we compare the proposed circuits for MAC operation with prior designs to show the effectiveness of the proposed circuits. The proposed FC is compared with two other FCs of [58] and [20]. The proposed RC is compared with another efficient RC [48]. Since the proposed RC is fully compatible with the proposed adder and multiplier, we also evaluate the adder and multiplier followed by the proposed RC. Furthermore, to investigate the impact of the moduli set's cardinality on the efficiency of the proposed methods, we implement adders and multipliers for 16- to 64-bit data-width with different moduli sets. Finally, we implement FFT engines and convolutional neural networks (CNNs) PEs to evaluate the proposed method in real-world applications. We implement these applications with the proposed logic with several configurations and compare them with binary and RNS implementations.

For hardware cost comparison, we described all these methods using VHDL hardware description language. We used the Synopsys Design Compiler to synthesize the designs using the FreePDK 45-nm library [59]. We compare different designs of addition and multiplication in terms of area and power.

A. Adder

Fig. 10 shows the synthesis results for the proposed adder compared to prior methods. As can be seen, the proposed adder offers 2% and 69% lower area compared to the customized adder for moduli set ($2^n - 1$, 2^n, $2^n + 1$) [31] and a bit-serial variant of [50], respectively, for 8- to 64-bit additions. The proposed method uses RNS to perform computations. Since there are three moduli in the moduli set, the bit-width of the RNS operation n for each binary precision m can be computed by

$$\left\lceil n = \frac{m}{3} \right\rceil. \tag{2}$$

TABLE I
RNS BIT-WIDTHS AND THE REQUIRED CLOCK CYCLES FOR ADDITION AND MULTIPLICATION (SOME EXAMPLES)

Binary bit-width (m)	8	16	24	32
RNS bit-width (n)	3	6	8	11
# of cycles for addition	$2^3 - 1$	$2^6 - 1$	$2^8 - 1$	$2^{11} - 1$
# of cycles for multiplication	$2^{3\times2} - 2^4 + 1$	$2^{6\times2} - 2^7 + 1$	$2^{8\times2} - 2^9 + 1$	$2^{11\times2} - 2^{12} + 1$

Table I shows some examples of the required bit-width and number of clock cycles for RNS computations with the proposed adder and multipliers when the cardinality of the moduli set is three.

We have also compared the proposed adder with the adder based on the thermometer coding proposed in [30]. Due to the massive hardware costs of the thermometer coding-based adder, we decided to only put synthesis results for only 8- and 16-bit additions in Fig. 10. The thermometer coding-based adder finishes the computations in one cycle at the cost of several orders of magnitude more hardware footprint. The hardware complexity of the thermometer-based adder is $\mathcal{O}(2^n)$, where n is the bit-width of the inputs of the modular adder. Three different modular adders are needed for the selected moduli set. Hence, the hardware cost of the thermometer-based method increases rapidly when n increases. As a result, adding two 16-bit binary input operands occupies a massive area and consumes very high power. Our proposed method outperforms the thermometer-based method by 98% and 93% in terms of area and power consumption, respectively. As seen in Fig. 10, the power consumption of the implemented method follows the same trend as its area occupation. The simple binary adder outperforms others in both area and power because the other techniques use modular operations, which need more hardware resources. However, in many situations, individual adders are not typically used; therefore, adders compatible with the other components are a requirement yet to gain the advantages of the proposed approach.

B. Multiplier

Fig. 11 compares the hardware area and power consumption of the implemented multipliers. As can be seen, the proposed multiplier outperforms the other designs in area occupation and power consumption for 8- to 64-bit multiplications. For instance, for 16-bit binary multiplication, the proposed method saves area by 59%, 57%, and 67% compared to the customized, interleaved, and fully bit-serial implementations, respectively. The proposed multiplier also offers, on average, 57%, 60%, and 77% lower power consumption compared to the customized, interleaved, and fully bit-serial implementations, respectively. The high hardware cost of the interleaved and the bit-serial method is due to using several registers in their designs. Our proposed method needs ($2^{2n} - 2^{n+1} + 1$) clock cycles to complete the computations; however, the other

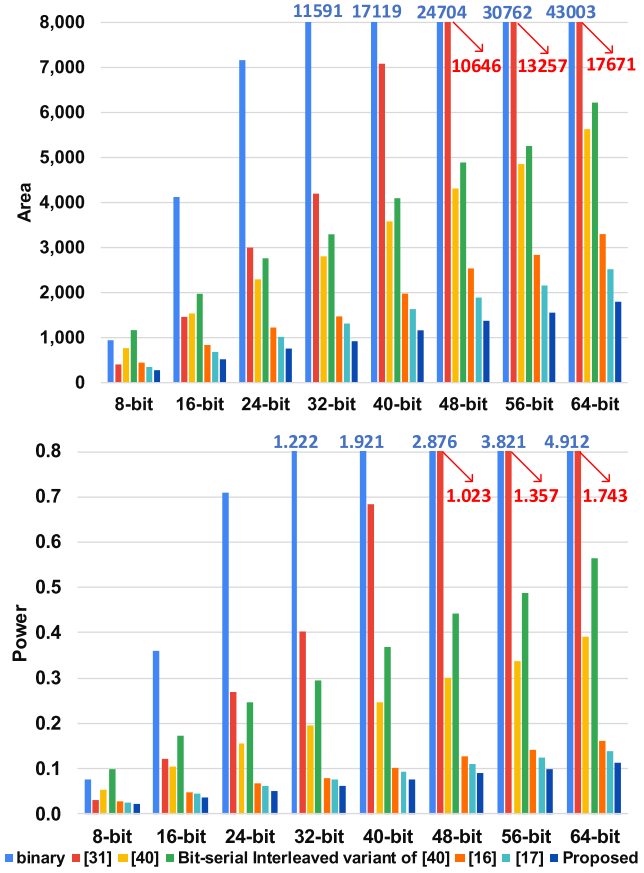


Fig. 11. Synthesis results for two input **multiplication** in 45-nm technology. The results are shown for 8- to 64-bit operands. (a) Area occupation (μm^2) and (b) power consumption (mW).

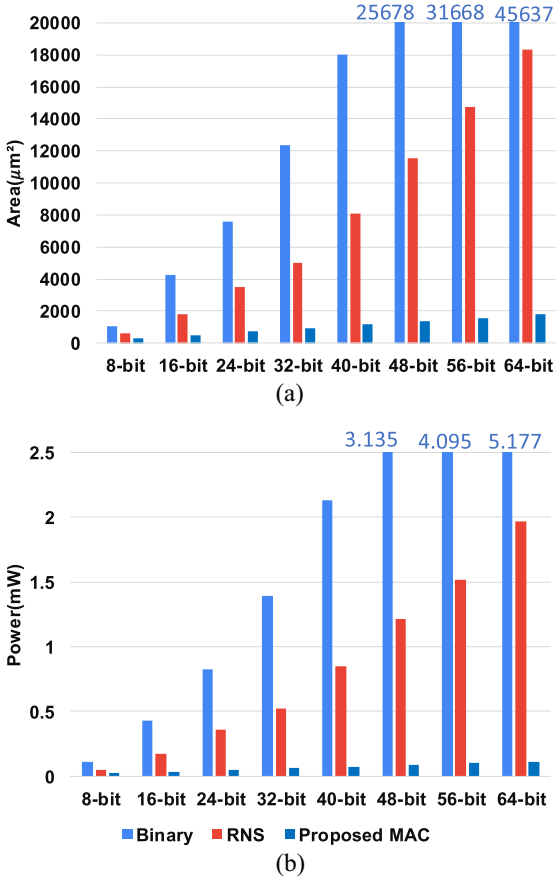


Fig. 12. Synthesis results for the **MAC** design. (a) Area occupation (μm^2) and (b) power consumption (mW).

TABLE II
SYNTHESIS RESULTS FOR FCS

Measure	Bit-width	FC in [20]	FC in [58]	**Proposed FC**
Area (μm^2)	8-bit	442.1	304.6	**245.0**
	16-bit	1054.5	583.4	**445.8**
	24-bit	1937.7	932.5	**660.3**
	32-bit	2812.9	1671.2	**824.6**
Power (mW)	8-bit	1.86E-02	1.80E-02	**9.93E-03**
	16-bit	3.51E-02	3.76E-02	**1.87E-02**
	24-bit	5.33E-02	6.05E-02	**2.76E-02**
	32-bit	6.83E-02	9.65E-02	**3.44E-02**

methods need one, n, and n^2 clock cycles, respectively. This will increase the energy consumption of the proposed method compared to others. Finally, the proposed method outperforms the binary implementation in both area and power.

Also, compared to the previously proposed combination of RNS and bit-stream processing [16], [17], the proposed multiplier of this work requires lower hardware resourses because the employed bit-stream generator has a lower hardware cost. In addition, the proposed multiplier performs a single multiplication faster than the multipliers in [16] and [17] because they complete their calculations in 2^{2n} clock cycles.

1) MAC Operation: The area and power overhead of the proposed MAC design compared to the proposed multiplier is negligible. The slight hardware cost overhead is due to the additional input registers for the third input of the operation.

TABLE III
SYNTHESIS RESULTS FOR RCS AND COMBINATION OF ADDER/MULTIPLIER AND THE PROPOSED RC

Binary Bit-Width	Method	Area (μm^2)	Power (mW)
8 ($m = 3$)	RC in [48]	391.9	3.38E-2
	RC in [17]	254.8	1.81E-2
	Proposed RC	**211.6**	**1.52E-2**
	Multiplier [31] + RC [48]	669.7	5.67E-02
	Multiplier + RC [17]	309.3	1.88E-02
	Proposed (multiplier + RC)	**217.7**	**1.62E-02**
16 ($m = 6$)	RC in [48]	803.4	7.01E-02
	RC in [17]	592.7	3.41E-02
	Proposed RC	**480.1**	**2.88E-02**
	Multiplier [31] + RC [48]	1922.2	2.05E-01
	Multiplier + RC [17]	674.3	3.64E-02
	Proposed (multiplier + RC)	**487.1**	**2.96E-02**
32 ($m = 11$)	RC in [48]	1533.6	1.32E-01
	RC in [17]	1187.3	6.43E-02
	Proposed RC	**1007.1**	**5.60E-02**
	Multiplier [31] + RC [48]	5327.9	6.18E-01
	Multiplier + RC [17]	1406.9	7.15E-02
	Proposed (multiplier + RC)	**1022.6**	**5.68E-02**

Fig. 12 compares the hardware cost of the proposed MAC unit with the binary and RNS counterparts. As can be seen in Fig. 12, the proposed method offers, on average, 92% and

TABLE IV
SYNTHESIS RESULTS FOR **ADDITION AND MULTIPLICATION** FOR DIFFERENT MODULI SET

			Addition				Multiplication			
Binary precision (m)	Moduli set	RNS bit-width (n)	Power (mW)	Area (μm^2)	# of clock cycles	Energy (pJ)	Power (mW)	Area (μm^2)	# of clock cycles	Energy (pJ)
$m = 64$	4194304, 4194303, 4194301	22	2.01E+03	1.35E-01	4.19E+06	7.64E+05	1.80E+03	1.14E-01	1.76E+13	2.81E+12
	131072, 131071, 131069, 131067	17	2.08E+03	1.43E-01	1.31E+05	1.97E+04	1.84E+03	1.19E-01	1.72E+10	2.49E+09
	2048, 2047, 2045, 2043, 2041, 2039	11	2.09E+03	1.45E-01	2.05E+03	2.40E+02	1.86E+03	1.21E-01	4.19E+06	4.11E+05
	512, 511, 509, 507, 505, 503, 499, 493	9	2.28E+03	1.65E-01	5.11E+02	6.66E+01	2.02E+03	1.35E-01	2.61E+05	2.43E+04
	128, 127, 125, 123, 121, 119, 113, 109, 107, 103	7	2.26E+03	1.65E-01	1.27E+02	1.45E+01	1.98E+03	1.37E-01	1.61E+04	1.46E+03
	64, 63, 61, 59, 55, 53, 47, 43, 41, 37, 31, 29	6	2.26E+03	1.76E-01	6.30E+01	8.76E+00	2.04E+03	1.45E-01	3.97E+03	4.14E+02
$m = 48$	131072, 131071, 131069	17	1.56E+03	1.06E-01	1.31E+05	1.46E+04	1.37E+03	8.97E-02	1.72E+10	1.88E+09
	8192, 8191, 8189, 8187	13	1.61E+03	1.12E-01	8.19E+03	7.89E+02	1.42E+03	9.26E-02	6.71E+07	5.53E+06
	1024, 1023, 1021, 1019, 1015	10	1.57E+03	1.10E-01	1.02E+03	8.33E+01	1.39E+03	9.25E-02	1.05E+06	7.26E+04
	128, 127, 125, 123, 121, 119, 113	7	1.58E+03	1.15E-01	1.27E+02	9.35E+00	1.40E+03	9.63E-02	1.61E+04	9.63E+02
	64, 63, 61, 59, 55, 53, 47, 43, 41	6	1.68E+03	1.29E-01	6.30E+01	5.36E+00	1.54E+03	1.09E-01	3.97E+03	2.90E+02
$m = 32$	2048, 2047, 2045	11	1.04E+03	7.22E-02	2.05E+03	1.20E+02	9.20E+02	6.06E-02	4.19E+06	2.06E+05
	512, 511, 509, 507	9	1.14E+03	8.18E-02	5.11E+02	3.05E+01	1.01E+03	6.75E-02	2.61E+05	1.22E+04
	128, 127, 125, 123, 121	7	1.13E+03	8.28E-02	1.27E+02	6.73E+00	9.97E+02	6.82E-02	1.61E+04	6.38E+02
	64, 63, 61, 59, 55, 53	6	1.12E+03	8.75E-02	6.30E+01	3.64E+00	1.03E+03	7.27E-02	3.97E+03	1.76E+02
	32, 31, 29, 27, 25, 23, 19	5	1.14E+03	8.87E-02	3.10E+01	1.54E+00	9.90E+02	7.16E-02	9.59E+02	3.71E+01
$m = 16$	64, 63, 61	6	5.60E+02	4.30E-02	6.30E+01	1.71E+00	5.16E+02	3.64E-02	3.97E+03	7.80E+01
	32, 31, 29, 27	5	6.53E+02	5.03E-02	3.10E+01	8.73E-01	5.65E+02	4.12E-02	9.59E+02	1.86E+01
	16, 15, 13, 11, 7	4	6.49E+02	5.32E-02	1.50E+01	4.87E-01	5.91E+02	4.41E-02	2.23E+02	5.31E+00

73% lower power consumption compared to the implemented binary and RNS designs, respectively.

C. Forward Converter

To evaluate the hardware efficiency of the proposed FC, we compare our FC with the conversion methods in [58] and [20]. In contrast with many prior works that implement specific modules, [58] and [20] propose efficient generic FCs. We implemented these three methods with different bit-widths from 8 to 32 bits. Table II compares the hardware costs of the implemented FCs for four different bit-widths. As can be seen, the proposed FC outperforms the FC in [20] in terms of area and power, on average, by 75% and 52%, respectively. Moreover, our FC offers 36% and 56% less area and power compared to the FC in [58]. These results confirm the efficiency of the proposed FC for devices with a restricted power budget.

D. Reverse Converter

To show the hardware efficiency of the proposed RC, we compare it with two efficient RC presented in [17] and [48]. The RC in [48] is designed only for the specific moduli set $(2^m-1, 2^m, 2^m+1)$. But similar to our RC, the RC in [17] can be used with any moduli set. Table III compares the hardware cost of the proposed RC and the two other RCs when $m = 3$ ($n = 8$), $m = 6$ ($n = 16$), and $m = 11$ ($n = 32$). The proposed RC outperforms the RC of [17] by, on average, 14% saving in power consumption and 9% lower area. Furthermore, the proposed RC offers, on average, 45% and 70% lower area and power compared to the RC in [48]. The main reason for the lower area of the proposed RC compared to the RC in [17] is the difference in the FSMs used to generate bit-streams. The design of [17] uses the FSM-based bit-stream generator proposed in [45] to convert the residues to bit-stream representation. But the FSM used in this work iterates each bit of a binary number equal to its positional weight. It should be noted that the cost of input registers are considered in the synthesis results reported in [17]; however, in the results reported in Table III, we do not consider the cost of these registers.

TABLE V
REAL-WORLD APPLICATIONS WIDELY USED IN EMBEDDED SYSTEMS

Application	Formula
FFT	$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$
DCT	$X_k = \sum_{i=0}^{N-1} x_i cos[\frac{\pi}{N}(i+\frac{1}{2})k]$
FIR filters	$\sum_{i=0}^{N} b_i.x[n-i]$
ConvNet	$F(\sum_i w_i x_i + b)$

The proposed RC can be used after the proposed adder and multiplier to directly convert their output bit-stream to binary format (the output counters are omitted). In this case, the area and power consumption of the combined circuit is less than the sum of the area and power of the adder/multiplier and the RC. Table III shows that the combined circuit of multiplier and RC can save both area and power by at least 40% compared to the case of implementing separate designs for the multiplier and RC. Moreover, the comparisons show that the proposed design decreases the required power by near an order of magnitude and area by at least a factor of three compared to a combination of the multiplier of [31] and the RC of [48].

E. Different Moduli Sets

The required number of clock cycles to complete computations using the proposed approach depend on the cardinality of the selected moduli set. In the above sections, the moduli set had three members; hence computations of operations with long bit-widths need many clock cycles to complete computations. This results in a very long processing time and very high energy consumption, which is unacceptable in many applications. For example, $(2^{22*2}-2^{23}+1)$ clock cycles are required for a 64-bit multiplication in an RNS system with three moduli. Increasing the cardinality of the selected moduli set is a solution to decrease the number of processing cycles. We

TABLE VI
SYNTHESIS RESULTS FOR **FFT ENGINES** WITH DIFFERENT CONFIGURATIONS IN 45-NM TECHNOLOGY

Bit-width	8-bit				16-bit				32-bit			
Method	Binary	RNS	[17]	**Proposed**	Binary	RNS	[17]	**Proposed**	Binary	RNS	[17]	**Proposed**
Area (μm^2)	4241.5	2538.4	1099.1	**1127.2**	14742.1	8013.2	2088.3	**2057.8**	45074.3	21729.5	3824.3	**3478.9**
Power (mW)	0.395	0.208	0.113	**0.105**	1.512	0.795	0.197	**0.179**	5.517	2.357	0.339	**0.292**

investigate the impact of the number of modules in the moduli set on power, energy, processing time, and area of the circuits. Table IV shows that the moduli sets with more members negligibly increase the power and area but exponentially decrease the number of clock cycles. This significantly reduces the energy consumption and processing time. Consequently, there is a tradeoff between area/power and energy/processing time. Since the proposed approach offers a low-effort design process and the moduli's area and power overhead are negligible, we suggest using the moduli set with the largest cardinality.

F. Proposed Method in Real-World Applications

Table V shows the formulas for computing FFT, finite impulse response (FIR) filter, discrete Cosine transform (DCT), and CNNs, which are widely used in many real-world applications [61], [62]. Addition, multiplication, and fused operations are the main building blocks of these functions. In contrast to high-performance computing platforms like CPUs [63], [64] and GPUs [65], [66] that are optimized for computational performance, EH and BI devices must satisfy power constraints. Therefore, using the proposed method can lead to low-power and low-area implementation of these functions.

We first investigate the efficiency of the proposed designs in implementing several FFT engines. FFT is an efficient algorithm to compute the discrete Fourier transform (DFT). We implemented the FFT algorithm with different number of points (64, 128, 256, and 512 points) and different data precisions (8-, 16-, and 32-bit) in binary, RNS [17], and our proposed method. These FFT processors have similar architectures except their PEs. The building blocks (butterfly elements) of each FFT engine with the same bit-width are similar. Therefore, we only compare the hardware cost of these elements, and use the complete implementations to investigate the conversion overheads. Next, we implement the PE of the CNN accelerator proposed in [60] with: 1) binary; 2) bit-stream-based RNS method of [17]; and 3) proposed computational elements. Our implementation consists of the computational and combinational parts of the PE in [60].

1) FFT Design: Table VI compares the hardware area cost of a butterfly element for FFT calculation with several computing schemes with different bit-widths. The proposed method utilizes, on average, 78% less area than the binary-based design. Also, it occupies 62% and 6% less area compared to the conventional RNS-based design and the design of [17], respectively. There is no need to convert from/to RNS iteratively in FFT processors, and a single binary-to-RNS conversion on the primary inputs is sufficient. The computations in the intermediate layers are all performed in the RNS domain. Finally, an RNS-to-binary conversion will convert the final results back to their corresponding binary representation. Therefore, RCs are placed at the last layer of computations in the RNS-based design, the design of [17] and the implementation based on the proposed method. Our synthesis results show that the overhead of these conversions for FFT calculation with more than 64 points is less than 10%.

TABLE VII
SYNTHESIS RESULTS OF PE OF CNN ACCELERATOR

Method	Area (μm^2)	Power (mW)	CP Delay (ns)
[60]	8145.64	0.477	5.43
[17]	1340.32	0.152	0.73
Proposed	**1309.54**	**0.149**	**0.71**

Table VI also compares the power consumption of the implemented FFT butterfly elements. For 8-bit data precision, the proposed design outperforms binary-based and RNS-based implementations with 72% and 47% lower power consumption, respectively. Table VI also shows that the power efficiency of the proposed method increases by increasing data-width. For example, for 16-bit data, the proposed method offers 74% and 86% less power consumption than the RNS-based and binary-based implementations, respectively. The bit-stream-based design of [17] consumes, on average, 12% more power compared to the proposed design. Since the number of clock cycles to complete the computations with the two bit-stream-based methods, i.e., [17] and our method, are exponentially greater than the binary and RNS baseline methods, these two methods need more processing time and energy consumption.

2) CNN Design: To further evaluate the proposed method, we implemented the PE of [60] with binary, bit-stream-based method of [17], and the proposed components. In our evaluation, we eliminate memories and buffers since they are implemented with similar logic in all three designs. All designs are implemented with 16-bit binary precision. Table VII shows the synthesis results. As it can be seen, the proposed design provides lower area and power usage. Although the proposed design outperforms the binary implementation by 6.22× and 3.13× more efficient area occupation and power consumption, it only offers near 1% better power and area compared to [17]. Therefore, reducing the number of clock cycles and as a direct result, reducing processing time and energy consumption is the most important advantage of the proposed design compared to [17].

V. CONCLUSION

In this work, we proposed a novel approach for ultralow-power design of modular adders and multipliers. The proposed method guarantees the same critical path latency for modular adders or multipliers with the same data bit-width. The proposed approach offers a cost-efficient design in terms of area occupation and power consumption. The ultralow-power and low-area property of the proposed design makes it an

excellent choice for edge devices, especially the EH and BI devices. We also proposed an efficient bit-stream-based RNS-to-binary convertor that outperforms the state-of-the-art.

REFERENCES

[1] S. Priya and D. J. Inman, *Energy Harvesting Technologies*, vol. 21. New York, NY, USA: Springer, 2009.

[2] J. Olivo, S. Carrara, and G. De Micheli, "Energy harvesting and remote powering for implantable biosensors," *IEEE Sensors J.*, vol. 11, no. 7, pp. 1573–1586, Jul. 2011.

[3] P. D. Mitcheson, "Energy harvesting for human wearable and implantable bio-sensors," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol.*, 2010, pp. 3432–3436.

[4] G. Lazzi, "Thermal effects of bioimplants," *IEEE Eng. Med. Biol. Mag.*, vol. 24, no. 5, pp. 75–81, Sep./Oct. 2005.

[5] B. Parhami, *Computer Arithmetic*. New York, NY, USA: Oxford Univ. Press, 2010.

[6] M. Hosseinzadeh, K. Navi, and S. Gorgin, "A new moduli set for residue number system:(r^n-2, r^n-1, r^n)," in *Proc. Int. Conf. Electr. Eng.*, 2007, pp. 1–6.

[7] G. C. Cardarilli, A. Nannarelli, and M. Re, "Residue number system for low-power DSP applications," in *Proc. Conf. Rec. 41st Asilomar Conf. Signals Syst. Comput.*, 2007, pp. 1412–1416.

[8] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *Proc. 25th Int. Conf. Field Program. Logic Appl. (FPL)*, 2015, pp. 1–6.

[9] A. V. Ananthalakshmi and P. Rajagopalan, "VLSI implementation of residue number system based efficient digital signal processor architecture for wireless sensor nodes," *Int. J. Inf. Technol.*, vol. 11, no. 4, pp. 829–840, 2019.

[10] A. Roohi, M. Taheri, S. Angizi, and D. Fan, "RNSiM: Efficient deep neural network accelerator using residue number systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2021, pp. 1–9.

[11] S. Angizi, A. Roohi, M. Taheri, and D. Fan, "Processing-in-memory acceleration of MAC-based applications using residue number system: A comparative study," in *Proc. Great Lakes Symp. VLSI*, 2021, pp. 265–270. [Online]. Available: https://doi.org/10.1145/3453688.3461529

[12] H. L. Garner, "The residue number system," presented at the Western Joint Comput. Conf., 1959, pp. 146–153.

[13] A. A. E. Zarandi, A. S. Molahosseini, M. Hosseinzadeh, S. Sorouri, S. Antão, and L. Sousa, "Reverse converter design via parallel-prefix adders: Novel components, methodology, and implementations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 2, pp. 374–378, Feb. 2015.

[14] 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, Exp. Briefs*, vol. 54, no. 9, pp. 775–779, Sep. 2007.

[15] H. Pettenghi, R. Chaves, and L. Sousa, "Method to design general RNS reverse converters for extended moduli sets," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 12, pp. 877–881, Dec. 2013.

[16] K. Givaki et al., "Using residue number systems to accelerate deterministic bit-stream multiplication," in *Proc. IEEE 30th Int. Conf. Appl. Spec. Syst. Archit. Process. (ASAP)*, 2019, p. 40.

[17] K. Givaki et al., "High-performance deterministic stochastic computing using residue number system," *IEEE Design Test*, vol. 38, no. 6, pp. 60–68, Dec. 2021.

[18] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2016, pp. 1–8.

[19] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.

[20] W. Jenkins and B. Leon, "The use of residue number systems in the design of finite impulse response digital filters," *IEEE Trans. Circuits Syst.*, vol. CS-24, no. 4, pp. 191–201, Apr. 1977.

[21] A. B. Premkumar, E. L. Ang, and E. M.-K. Lai, "Improved memoryless RNS forward converter based on the periodicity of residues," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 2, pp. 133–137, Feb. 2006.

[22] A. B. Premkumar, "A formal framework for conversion from binary to residue numbers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 49, no. 2, pp. 135–144, Feb. 2002.

[23] G. Bi and E. Jones, "Fast conversion between binary and residue numbers," *Electron. Lett.*, vol. 24, no. 19, pp. 1195–1197, 1988.

[24] F. Pourbigharaz and H. M. Yassine, "Simple binary to residue transformation with respect to 2^m+1 moduli," *IEE Proc. Circuits Devices Syst.*, vol. 141, no. 6, pp. 522–526, 1994.

[25] A. A. Hiasat, "High-speed and reduced-area modular adder structures for RNS," *IEEE Trans. Comput.*, vol. 51, no. 1, pp. 84–89, Jan. 2002.

[26] A. Hiasat, "General modular adder designs for residue number system applications," *IET Circuits Devices Syst.*, vol. 12, no. 4, pp. 424–431, 2018.

[27] H. T. Vergos and C. Efstathiou, "On the design of efficient modular adders," *J. Circuits Syst. Comput.*, vol. 14, no. 5, pp. 965–972, 2005.

[28] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Trans. Comput.*, vol. 43, no. 1, pp. 68–77, Jan. 1994.

[29] C. H. Vun, A. B. Premkumar, and W. Zhang, "A new RNS based DA approach for inner product computation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 8, pp. 2139–2152, Aug. 2013.

[30] F. Jafarzadehpour, A. S. Molahosseini, A. A. E. Zarandi, and L. Sousa, "Efficient modular adder designs based on thermometer and one-hot coding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2142–2155, Sep. 2019.

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

[32] R. Muralidharan and C.-H. Chang, "Radix-4 and radix-8 booth encoded multi-modulus multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 11, pp. 2940–2952, Nov. 2013.

[33] D. Bakalis, H. T. Vergos, and A. Spyrou, "Efficient modulo $2^n \pm 1$ squarers," *Integration*, vol. 44, no. 3, pp. 163–174, 2011.

[34] Z. Wang, G. A. Jullien, and W. C. Miller, "An algorithm for multiplication modulo ($2^n - 1$)," in *Proc. 39th Midwest Symp. Circuits Syst.*, vol. 3, 1996, pp. 1301–1304.

[35] M. A. Soderstrand and C. Vernia, "A high-speed low-cost modulo pimultiplier with RNS arithmetic applications," *Proc. IEEE*, vol. 68, no. 4, pp. 529–532, Apr. 1980.

[36] G. A. Jullien, "Implementation of multiplication, modulo a prime number, with applications to number theoretic transforms," *IEEE Trans. Comput.*, vol. C-29, no. 10, pp. 899–905, Oct. 1980.

[37] M. Dugdale, "Residue multipliers using factored decomposition," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 9, pp. 623–627, Sep. 1994.

[38] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS montgomery modular multiplication algorithm," *IEEE Trans. Comput.*, vol. 47, no. 7, pp. 766–776, Jul. 1998.

[39] E. D. Di Claudio, F. Piazza, and G. Orlandi, "Fast combinatorial RNS processors for DSP applications," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 624–633, May 1995.

[40] G. R. Blakely, "A computer algorithm for calculating the product AB modulo M," *IEEE Trans. Comput.*, vol. C-100, no. 5, pp. 497–500, May 1983.

[41] P. Kornerup, "A systolic, linear-array multiplier for a class of right-shift algorithms," *IEEE Trans. Comput.*, vol. 43, no. 8, pp. 892–898, Aug. 1994.

[42] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.

[43] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Impact of RNS coding overhead on FIR filters performance," in *Proc. Conf. Rec. 41st Asilomar Conf. Signals Syst. Comput.*, 2007, pp. 1426–1429.

[44] I. Kouretas and V. Paliouras, "RNS multi-voltage low-power multiply-add unit," in *Proc. 17th IEEE Int. Conf. Electron. Circuits Syst.*, 2010, pp. 9–12.

[45] S. Asadi and M. H. Najafi, "Late breaking results: LDFSM: A low-cost bit-stream generator for low-discrepancy stochastic computing," in *Proc. 57th ACM/IEEE Des. Autom. Conf. (DAC)*, 2020, pp. 1–2.

[46] S. Asadi, M. H. Najafi, and M. Imani, "A low-cost FSM-based bit-stream generator for low-discrepancy stochastic computing," in *Proc. Des. Autom. Test Eur. Conf. Exhibit. (DATE)*, 2021, pp. 908–913.

[47] Y. Wang, X. Song, M. Aboulhamid, and H. Shen, "Adder based residue to binary number converters for $[2^n-1, 2^n, 2^n+1]$," *IEEE Trans. Signal Process.*, vol. 50, no. 7, pp. 1772–1779, Jul. 2002.

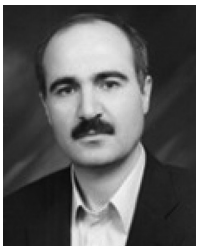
[48] M. Bhardwaj, A. Premkumar, and T. Srikanthan, "Breaking the 2*n*-bit carry propagation barrier in residue to binary conversion for the $[2^n-1, 2^n, 2^n+1]$ moduli set," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 45, no. 9, pp. 998–1002, Sep. 1998.

[49] S. Andraos and H. Ahmad, "A new efficient memoryless residue to binary converter," *IEEE Trans. Circuits Syst.*, vol. 35, no. 11, pp. 1441–1444, Nov. 1988.

[50] P. V. A. Mohan, "RNS to binary conversion using diagonal function and Pirlo and Impedovo monotonic function," *Circuits Syst. Signal Process.*, vol. 35, no. 3, pp. 1063–1076, 2016.
[51] N. Burgess, "Scaled and unscaled residue number system to binary conversion techniques using the core function," in *Proc. 13th IEEE Symp. Comput. Arithmetic*, 1997, pp. 250–257.
[52] B. R. Gaines, "Stochastic computing," in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 149–156.
[53] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.
[54] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.
[55] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, 2013.
[56] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1644–1657, May 2017.
[57] M. H. Najafi, D. J. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2018, pp. 1–8.
[58] R. M. Capocelli and R. Giancarlo, "Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa," *IEEE Trans. Circuits Syst.*, vol. 35, no. 11, pp. 1425–1430, Nov. 1988.
[59] "NCSU FreePDK 45nm library." [Online]. Available: https://research.ece.ncsu.edu/eda/freepdk/freepdk45/
[60] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[61] Y. Lu, T. J. Kazmierski, and L. Liu, "A bit-serial variable-accuracy FFT processor for energy-harvesting systems," in *Proc. IEEE Asia–Pacific Conf. Circuits Syst. (APCCAS)*, 2018, pp. 299–304.
[62] K. Qiu et al., "ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 315–327.
[63] R. Bera et al., "Hermes: Accelerating long-latency load requests via perceptron-based off-chip load prediction," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 1–18.
[64] T. A. Khan et al., "Whisper: Profile-guided branch misprediction elimination for data center applications," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 19–34.
[65] S. Darabi et al., "Morpheus: Extending the last level cache capacity in GPU systems using idle GPU core resources," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 228–244.
[66] S. Darabi, N. Mahani, H. Baxishi, E. Yousefzadeh-Asl-Miandoab, M. Sadrosadati, and H. Sarbazi-Azad, "NURA: A framework for supporting non-uniform resource accesses in GPUs," *Proc. ACM Measur. Anal. Comput. Syst.*, vol. 6, no. 1, pp. 1–27, 2022.

Kamyar Givaki received the B.Sc. degree in computer engineering from the K.N. Toosi University of Technology, Tehran, Iran, in 2012, and the M.Sc. degree in computer engineering from the Isfahan University of Technology, Isfahan, Iran, in 2016. He is currently pursuing the Ph.D. degree in computer engineering with the University of Tehran, Tehran.

His research is focused on application-specific accelerators, computer arithmetic, and approximate computing.



Ahmad Khonsari received the B.Sc. degree in electrical and computer engineering from Shahid Beheshti University, Tehran, Iran, in 1991, the M.Sc. degree in computer engineering from the Iran University of Science and Technology, Tehran, in 1996, and the Ph.D. degree in computer science from the University of Glasgow, Glasgow, U.K., in 2003.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Tehran, Tehran, and a Researcher with the School of Computer Science, Institute for Research in Fundamental Sciences, Tehran. His research interests include simulation and data analysis, performance modeling/evaluation, wired/wireless networks, cloud and distributed systems, quantum information processing, and high-performance computer architectures.

M. H. Gholamrezaei received the B.S. degree in computer architecture engineering from Shahid Beheshti University, Tehran, Iran, in 2020.

He is currently a Graduate Student with the Department of Computer Engineering, Chosun University, Gwangju, South Korea. His research interests include computer arithmetic, embedded systems, and design automation.

Saeid Gorgin (Senior Member, IEEE) received the B.S. degree in computer engineering from Islamic Azad University South Tehran Branch, Tehran, Iran, in 2001, the M.S. degree in computer engineering from Islamic Azad University Tehran Science and Research Branch, Tehran, in 2004, and the Ph.D. degree in computer system architecture from Shahid Beheshti University, Tehran, in 2010.

He is currently an Associate Professor of Computer Engineering with the Department of Electrical Engineering and Information Technology, Iranian Research Organization for Science and Technology, Tehran. He is also a Visiting Scientist with the Computer Systems Laboratory, Department of Computer Engineering, Chosun University, Gwangju, South Korea. His current research interests include computing systems, computer arithmetic, and VLSI design.

M. Hassan Najafi (Member, IEEE) received the B.Sc. degree in computer engineering from the University of Isfahan, Isfahan, Iran, in 2011, the M.Sc. degree in computer architecture from the University of Tehran, Tehran, Iran, in 2014, and the Ph.D. degree in electrical engineering from the University of Minnesota, Twin Cities, MN, USA, in 2018.

He is currently an Assistant Professor with the School of Computing and Informatics, University of Louisiana, Lafayette, LA, USA. He has authored/coauthored more than 60 peer-reviewed papers and has been granted five U.S. patents with more pending. His research interests include stochastic and approximate computing, unary processing, in-memory computing, and hyperdimensional computing.

Dr. Najafi received the 2018 EDAA Outstanding Dissertation Award, the Doctoral Dissertation Fellowship from the University of Minnesota, and the Best Paper Award at the ICCD'17, in recognition of his research. He has been an Editor of the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS.