Hardware/Software Approach to Designing Low-Power RNS-Enhanced Arithmetic Units

Piotr Patronik and Stanisław J. Piestrak, *Member, IEEE*

***Abstract*—In this paper, we propose a new approach to use a residue number system (RNS) to design an arithmetic unit to parallelize execution of addition and multiplication. The chosen RNS is defined by a moduli set composed of one larger even modulus 2^k and all remaining moduli of the type $2^n - 1$, selected to fit into the word size of a typical general-purpose processor, e.g., 32 or 64 b. The RNS operations are implemented in hardware, except for the reverse conversion, which is implemented in software, allowing not only to save hardware area but also offering the ease of run-time changing of the dynamic range, which in turn can result in reducing both power consumption and execution time. Simulation experiments were performed on synthesized seven-operation arithmetic units with varying dynamic range for three applications: constant-coefficient filtering, matrix multiplication, and large Montgomery multiplication. The results show that thanks to smaller modular multipliers, RNS arithmetic units have smaller both area and delay, and, consequently, they allow to achieve up to over 20% energy saving for a constant-coefficient filter application, up to over 28% for the matrix multiplication, and up to 27% for Montgomery multiplication, compared with executions using a positional arithmetic unit.**

***Index Terms*—Application-specific integrated circuit (ASIC), arithmetic unit, digital signal processing (DSP), instruction set extensions, parallel architectures, residue number system (RNS).**

I. Introduction

THE Residue Number System (RNS) is a non-positional number system, in which integers are represented by the sets of independent digits called *residues*. An inherent parallelism of the RNS representation has incited designers to consider it for hardware implementations of datapath designs, especially in digital signal processing (DSP) applications [1], [2]. However, hardware datapaths, despite they can be optimized in many sophisticated ways, cannot be modified due to their fixed nature. Thus, the processor designs employing novel, parallel, arithmetic architectures seem to be a promising way of research, especially in the light of diminishing possibilities resulting from simple scaling of the VLSI circuits.

To date, a number of various specialized processors provided with RNS-enhanced computation capabilities have been proposed, e.g. in [3]–[8]. Here, we concentrate on the most recent of them from [7] and [8]. In [7], it is proposed an approach ranging from low-level architecture using the 3-moduli RNS $\{2^n - 1, 2^n + 1, 2^k\}$ with flexible $k > n$ to compilation techniques applied on the level of basic blocks. The five-instruction set to manipulate modular data was suggested: {RADD, RSUB, RMUL, FC, RC}—respectively: residue addition, subtraction, and multiplication, as well as RNS forward and reverse conversion, assuming that the reverse conversion (the operation executed to bring the number back to the positional representation) can be scheduled among other instructions. For the latter design, we have identified the following limitations, which we would like to resolve by proposing a new architecture:

(i) It uses only three moduli, hence to achieve a given dynamic range, wider channels must be used, which are more complex and slower than for a carefully selected set with $r \geq 4$ moduli.

(ii) Its moduli set contains one modulus of the type $2^n + 1$, whose representation is rather inefficient, because only $2^n + 1$ out of 2^{n+1} combinations are actually used (i.e. slightly more than 50%).

(iii) The reverse converter is an extra hardware block introducing its area and static power consumption.

(iv) The assumed parallel instruction scheduling requires a special processor architecture, either VLIW or multiple issue, which also consumes excessive power and may easily off-set any savings resulting from alternative RNS data representation.

While the idea of using an RNS-based co-processor is not new, in [8] and here, we propose a new approach to the design of an RNS-based modular datapath, which allows to replace one wide arithmetic channel with a larger number of narrow channels and attempts to avoid any unnecessary overhead. To ensure maximally large dynamic range for a given processor word size, we have limited the selection of odd moduli only to those of the type $2^n - 1$, for which the most efficient arithmetic circuitry can be implemented in hardware. Numerous hardware implementations of the basic modular arithmetic circuits for these low-cost moduli can be found: residue generators (used to build forward converters) and multi-operand modular adders (MOMAs) [9], [10], 2-operand adders [11]–[13], multipliers [14]–[16], and residue multiplier-accumulate units (MACs) and complete datapaths [17]–[19]. We propose to use an r-moduli set ($r \geq 3$) composed of one even modulus 2^k and $r - 1$ pairwise relatively prime moduli of the type $2^n - 1$, which all fit into a processor's word (e.g. 32- or 64-bit). This newly proposed drop-in architecture using energy-efficient arithmetic components is activated by

Manuscript received June 28, 2016; revised December 5, 2016; accepted February 3, 2017. Date of publication March 14, 2017; date of current version April 20, 2017. This paper was recommended by Associate Editor P. Julian.

P. Patronik is with the Department of Computer Engineering, Faculty of Electronics, Wrocław University of Technology, 50-370 Wrocław, Poland; part of his research was done when he was on leave from the Institut Jean Lamour (UMR 7198 CNRS), Université de Lorraine, F-54506 Vandoeuvre-Lès-Nancy, France (e-mail: piotr.patronik@pwr.wroc.pl).

S. J. Piestrak is with the Institut Jean Lamour (UMR 7198 CNRS), Université de Lorraine, F-54506 Vandoeuvre-Lès-Nancy, France (e-mail: stanislaw.piestrak@univ-lorraine.fr).

Digital Object Identifier 10.1109/TCSI.2017.2669108

three special RNS arithmetic instructions. Forward conversion, which is relatively cheap in terms of area and energy consumption, is also implemented in hardware activated by a special instruction for residue generation. Reverse conversion which incurs significantly more complex hardware, unlike in any other known work, here will be implemented in software, hence without incurring any hardware overhead and offering the possibility of dynamic changing of the computation precision by dropping out the results of one or more residue arithmetic channels. In [8], we have presented the first version of this new architecture limited to one 5-moduli RNS with the dynamic range of 32 bits. Here, the design ideas of [8] will be presented in a more general context, including selection criteria of RNS moduli sets providing a given dynamic range (32 or 64 bits), all with suitable performance estimations and analysis.

Numerous applications requiring computations with varying precision have been reported in the literature, including those requiring: improved accuracy at the expense of performance and energy efficiency with run-time precision selection [20], adaptability of software defined radio to different communication standards [21], [22], high-speed and energy-efficient computations in the financial and insurance industry [23], [24], mixed precision methodologies applicable to Monte Carlo (MC) simulations [25], and low-power consumption in various DSP applications [26]–[34]. Two most commonly used solutions to resolve the problem of dynamically varying precision have been reported: the clock gating in ASIC circuits [35] and implementation of reconfigurable circuits using FPGA devices [36]. However, the former requires dividing arithmetic datapaths on subcircuits capable of handling smaller operands, which can be also configured (enabled or disabled) to make part of circuits handling larger operands, depending on the current precision needed. As a result, the circuitry handling the largest operands cannot attain the best possible characteristics. The major limitations of the FPGA-based solutions are: the necessity to store alternative configurations in a highly reliable "golden" memory and the delay introduced by the time needed to reprogram the device. By proposing here the hardware/software-based solution relying on using RNS number representation, we intended to avoid drawbacks of both of the above mentioned design styles.

This paper is organized as follows. In Section II, the theoretical background on RNS and selection criteria of RNS moduli sets are presented. In Section III, the architecture of a modular arithmetic unit is detailed and the case study of the RNS arithmetic unit used to support the 32-bit processor is presented. In Section IV, the rough complexity evaluation of the RNS arithmetic unit and the experimental evaluation of power consumption for sample applications are presented. Section V presents some conclusions and suggestions for future research.

II. Background on RNS

A. Basic Properties of RNS

An RNS is defined by a set of r pairwise prime positive integers called *moduli* $\{m_1, \ldots, m_r\}$. Its dynamic range (DR) is equal to $M = \prod_{i=1}^{r} m_i$. Let X be a non-negative integer such that $0 \leq X < M$. The remainder of the integer division of X by the modulus m_i, denoted $x_i = |X|_{m_i}$ and such that $0 \leq |X|_{m_i} \leq m_i - 1$ (i.e. of size $a_i = \lceil \log_2 \rceil$ bits), is called a *residue mod* m_i. It can be obtained using a *residue generator* mod m_i (see [9]), which is the basic building block of the binary-to-residue (input) converter composed of r such blocks m_i, producing the RNS representation of $X \xrightarrow{RNS} \{x_1, \ldots, x_r\}$. The Chinese Remainder Theorem (CRT) [1] stipulates that non-negative integers X may be written in RNS as a unique set of residues $\{x_1, \ldots, x_r\}$ provided that $0 \leq X < M$.

Consider three non-negative integers X, Y, Z such that $0 \leq X, Y, Z < M$ and whose residue representations are $\{x_1, \ldots, x_r\}$, $\{y_1, \ldots, y_r\}$, and $\{z_1, \ldots, z_r\}$. In RNS, any of three arithmetic operations $\circ \in \{+, -, \times\}$ performed on numbers represented by a set of residues, $z_i = |x_i \circ y_i|_{m_i}$, $i \in \{1, \ldots, r\}$, is equivalent to the same operation performed on their positional counterparts, i.e. $Z = |X \circ Y|_M$. Hence, using RNS may be viewed as a method for paralellization of arithmetic operations at a very low level, because the basic arithmetic operations can be performed in independent channels with no cross-channel interactions. The major savings due to RNS (in power, area, and delay) are in multiplication, because of the significant reduction of partial product matrices of $O(n^2)$ complexity. This advantage, however, comes with some limitations and at some cost. First, the complexity of modulo calculations must be as low as possible, which is easily achieved by using the moduli of the form $2^n - 1$ (besides only one even modulus 2^k). Second, the reverse conversion could be very complex, if implemented in hardware. Third, due to non-positional representation of numbers, every number comparison and sign detection of a number may require reverse conversion to be performed or including specifically designed circuitry.

B. Selection Criteria of RNS Moduli Sets

To maximize performance of the residue datapaths, it is desirable that the moduli selected to form an RNS are *low-cost*, i.e., they have particularly efficient hardware implementations. The modulus of the type 2^k (k is positive integer) is the best choice but only one even modulus can be used; therefore it is desirable to choose one with k as large as possible. The second and third best choices are odd moduli which are Mersenne and Fermat numbers, i.e., those of the form $2^n - 1$ and $2^n + 1$, respectively.

Here, we are interested in moduli sets providing the dynamic range as large as possible for a given total size of residue operands $a_{all} = \sum_{i=1}^{r} a_i$ and having the most efficient hardware implementations of the basic arithmetic operations. Hence, the most obvious candidates are: one even modulus 2^k and the moduli of the type $2^n - 1$ which, as mentioned earlier, have among known odd moduli the most efficient hardware implementations of all basic arithmetic circuits. The set of pairwise relatively prime moduli of the type $2^n - 1$ can be easily determined as follows. First, we can take advantage of the following property: $2^n - 1$ and $2^m - 1$ are relatively prime if and only if n and m are relatively prime [37]. The latter property implies that only one modulus of the form $2^n - 1$ where

TABLE I
SAMPLE r-MODULI SETS ($r \geq 3$) PROVIDING THE 32-BIT DYNAMIC RANGE WITH COMPLEXITY ESTIMATIONS OF MACs

r	Moduli set $n_1, \{n_2, \ldots, n_r\}$	Total number of FAs
3	$12,\ \{11, 9\}$	280
	$13,\ \{10, 9\}$	272
4	$12,\ \{9, 7, 4\}$	224
	$11,\ \{9, 7, 5\}$	221
	$10,\ \{9, 8, 5\}$	210
5	$13,\ \{7, 5, 4, 3\}$	190
	$9,\ \{8, 7, 5, 3\}$	192
6	$4,\ \{11, 7, 5, 3, 2\}$	218

n is even can be chosen. However, the most general selection guidelines can be found in the comprehensive theoretical study presented in [38]. Henceforth, we will assume that the moduli appear in the following order: $\{2^{n_1}, 2^{n_2}-1, \ldots, 2^{n_r}-1\}$, where $n_2 > n_3 > \cdots n_r$.

In general, for a given dynamic range, several sets of moduli can be found, which differ in the number of moduli r and then, for a given r, in the size of the even modulus 2^{n_1} which then directly influences the composition of the odd moduli set. Therefore, to convey a reader with the possible choices and the optimization problems encountered, we have considered the following example. Because residue MAC units mod $2^n - 1$ and mod 2^n are the main arithmetic circuits which must be added, we will provide preliminary rough estimations of their complexity, measured in terms of the number of full-adders (FAs). Their conceptually simplest implementations, which can be obtained using carry-save adders according to [17], consists of n^2 FAs—for MACs mod $2^n - 1$, and $[k(k+)]/2$ FAs—for MACs mod 2^k. The latter suggests that to take full advantage of the particularly efficient implementation of MACs mod 2^k, k should be larger than n.

Example 1: Consider the overall hardware complexity of MACs needed to implement an RNS arithmetic unit for the 32-bit dynamic range. Table I details several r-moduli sets ($r \geq 3$) providing the 32-bit dynamic range as well as the complexity estimations of MACs (expressed in terms of the number of FAs) which have been considered. All moduli sets $\{2^{n_1}, 2^{n_2}-1, \ldots, 2^{n_r}-1\}$ are sufficiently characterized by their exponents n_i, which thus provide direct information about the bit sizes of all moduli used. It seems that such a description is also more convenient, because the inspection of the exponents of the odd moduli $2^{n_i} - 1$, $2 \leq i \leq r$, which all must be pairwise relatively prime, can quickly reveal that no other alternative exists for a given number of moduli r.

It is seen that two 5-moduli sets enjoy the lowest similar complexity, although amongst them, the moduli set with larger even modulus (i.e. with $n_i = \{13, \quad 7, 5, 4, 3\}$) could provide a slight advantage of lower average power consumption, because of higher probability of having larger number of operands with leading zeros in the MAC unit mod 2^{13} than its counterpart mod 2^9.

The data given in Table I can lead to the following observations:

- *Increasing the number of moduli but only up to $r = 5$ leads to steady decreasing of the number of FAs.*
- *For $r = 5$ no other alternative exists, so the difference of the size of the even modulus for two moduli sets equals to 4.*
- *The best selection of the moduli for $r = 6$ (enclosed) does not allow to take the full advantage of the hardware efficiency of the even modulus: its size is determined by the smallest sizes of five odd moduli (which must be pairwise relatively prime).*

III. PROPOSED RNS PROCESSOR ARCHITECTURE

In this section, based on [8], we will first present the instruction set introduced to support RNS-based arithmetic operations and then we will detail the architecture proposed by presenting the case study of an RNS arithmetic unit for the 5-moduli set. Special care will be taken to detail the implementation of the reverse conversion.

A. Instruction Set

For simplicity, we focus only on unsigned numbers. Should signed numbers be needed, relatively simple correction is needed in forward and reverse conversions [1]. Assuming that multiplication is the most costly operation implemented in hardware of the general-purpose processor, we suggest to remove it and replace with modular multiplication. Thus, hardware implementation of our RNS arithmetic unit can execute the following modular instructions mod m_i, $1 \leq i \leq r$:

- `residue`—calculation of a set of r residues mod m_i (i.e. the input conversion);
- `add_m`—modulo addition; and
- `mul_m`—modulo multiplication.

To enable software implementation of the reverse conversion as well remaining RNS operations involving all residues (comparison, sign detection, etc.), the following positional instructions are also needed:

- `add` – addition;
- `sub` – subtraction;
- `and` – bit-wise AND; and
- `shr` – logical shift right.

B. Case Study

To illustrate the major design issues of the architecture proposed, here we will consider design and implementation aspects of the RNS arithmetic unit which can be used to support the 32-bit processor. The complexity estimations of MACs for several moduli sets which could fit into the register size of 32 bits, presented in Table I of Example 1, have revealed that the 5-moduli set $\{m_1, m_2, m_3, m_4, m_5\} = \{8192, 127, 31, 15, 7\}$ involves the lowest complexity (which will be confirmed by the synthesis results, presented in Section IV). Its dynamic range is $M = 3386449920$, i.e. ≈ 31.65 bits.

Fig. 1 shows the register layout of the residues x_i and the block diagram of the arithmetic unit proposed. From the point of view of the basic processor, it is the same component as a standard positional arithmetic (-logic) unit, although with different implementation of a few operations:

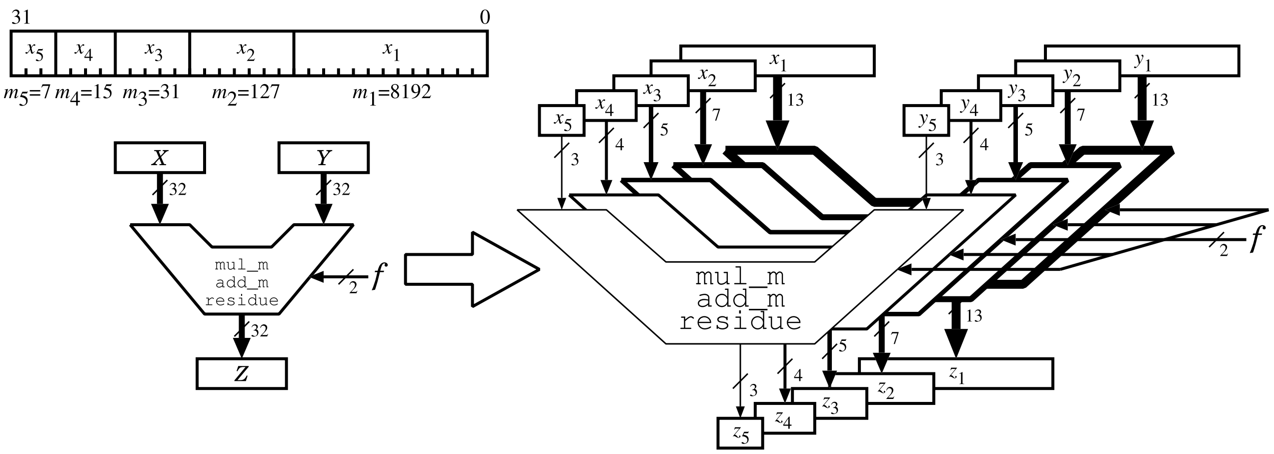

Fig. 1. Sample 32-bit RNS arithmetic unit: a) data and register layout; b) block diagram.

TABLE II
EXAMPLE OF SUCCESSIVE (MODULAR) APPROXIMATIONS OF THE FINAL RESULT X_5 FOR THE 32-BIT RNS

Moduli subset	Number of bits
$\{8192\}$	13.00
$\{8192, 127\}$	19.98
$\{8192, 127, 31\}$	24.94
$\{8192, 127, 31, 15\}$	28.84
$\{8192, 127, 31, 15, 7\}$	31.65

- residue calculation: $x_i = |X|_{m_i}$;
- addition: $z_i = |x_i + y_i|_{m_i}$; and
- multiplication: $z_i = |x_i \cdot y_i|_{m_i}$.

Let for $i = 1 \dots 5$, $M_i = \prod_{j=1}^{i} m_i$ and X_i respectively denote successive approximations of the dynamic range and of the result, where $X_1 = x_1$ and X_5 is the final value of the reverse conversion. The reverse conversion is implemented using the following recursive Mixed-Radix Conversion (MRC) formula [1]:

$$\begin{aligned} X_i &= d_1 + m_1 d_2 + \cdots + M_{i-1} d_i, \\ d_1 &= x_1, \\ &\dots, \\ d_i &= \left| \frac{x_i - X_{i-1}}{M_{i-1}} \right|_{m_i}. \end{aligned} \tag{1}$$

Its software implementation follows in an assembly pseudocode of Fig. 4, given in the Appendix.

It can be seen that the application of Eqn (1), implemented as consecutive steps in Fig. 4 of calculating digits followed by calculation of the positional number may be used to provide successive (modular) approximations of the final result X_5, which are listed in Table II. We thus have the dynamic range of 13 bits after the first conversion step (which is nothing else but a simple truncation) and then, respectively 19.98, 24.94, 28.84, and 31.65 bits—thus, depending on the desired dynamic range, a few digits can be skipped, still achieving the exact positional result. To achieve higher efficiency, the conversion may be also optimized independently for every fixed number of steps, so that some alternative reverse conversion assembly code versions allowing for variable computation precision could be made available for the user.

The above observations suggest one important class of applications of the RNS-based design methodology presented here— those involving applications requiring dynamic changing of the precision of computations. This could require minor modifications of the reverse conversion or providing multiple versions for different precision scenarios. The latter could rely e.g. on changing of the order of moduli in the reverse conversion: e.g. $n_1 = 2^{13}$ and only $n_2 = 127$ for DR=20; $n_1 = 2^{13}$ and only $n_5 = 7$ for DR=16. Any change of precision is very easy to provide, because it involves only changes in software without any hardware modifications.

IV. EXPERIMENTAL EVALUATION

In this section, we will present synthesis results providing area and delay estimations, evaluation of power and energy consumption obtained for three sample programs, and the analysis of the impact of varying precision on the number of executed instructions.

A. Synthesis Results

The proposed arithmetic unit was described in parametric structural Verilog with RTL constructs for positional operations (in channel mod 2^{n_1}). We have used the residue generators of [9], the adders of [11], and the multipliers of [15], all mod $2^{n_i} - 1$ for $3 \le n_i \le 13$ and $2 \le i \le r$. Although the synthesized arithmetic units are basically combinational circuits, to ensure realistic synthesis results, we have supplemented the designs with input and output registers. A traditional positional arithmetic unit using RTL operators of addition and multiplication was used as a reference counterpart for comparison. Despite it is customary to pipeline high complexity arithmetic circuits, notably multipliers, in our examples we have skipped this technique, because it consumes additional power for pipeline registers and adds another degree of freedom to the design.

The circuits were synthesized using Cadence RTL Compiler over ST 65nm low power library, providing synthesized netlists. The smallest delay at which the synthesizer still

TABLE III
TIME AND AREA CHARACTERISTICS OF TWO'S COMPLEMENT AND THE MOST EFFICIENT RNS ARITHMETIC UNITS

DR [bits]	Number system: $n_1, \{n_2, \ldots, n_r\}$	Delay [ns]	Area [μm^2]
32	RNS1 5M: 13, $\{7, 5, 4, 3\}$	1752	18432
	RNS2 5M: 9, $\{8, 7, 5, 3\}$	1751	18662
	Two's complement	2134	25932
64	RNS3 3M: 33, $\{16, 15\}$	2237	52092
	RNS4 4M: 34, $\{11, 10, 9\}$	2184	50051
	RNS5 5M: 27, $\{11, 10, 9, 7\}$	2087	48490
	RNS6 6M: 16, $\{13, 12, 11, 7, 5\}$	2186	46394
	RNS7 7M: 17, $\{13, 11, 8, 7, 5, 3\}$	2200	44423
	Two's complement	2552	77950

reported a non-negative delay slack was assumed the minimal delay of the designs. Table III shows the resulting delay and cell area of the synthesized circuits. The positional two's complement arithmetic circuitry realizing addition/subtraction, multiplication and shifting needs respectively for DR with 32 and 64 bits, over 40% and 75% of area and over 21% and 16% more delay than its RNS counterpart (e.g. the 32-bit multiplier itself occupies about 12μm^2 of the total 15μm^2 of the positional two's complement arithmetic unit area. The RNS-based unit consists of residue adders and multipliers (including the 13-bit NB multiplier for the even channel), as well as the two's complement adder/subtractor and shifter (needed to execute reverse conversion in software). Clearly, a large partial product matrix followed by the carry-save adder tree puts the two's complement multiplier unit at a disadvantage.

The observations made in Example 1 have led us to use an exhaustive search of all moduli sets providing dynamic ranges considered (since the number of moduli sets suitable for dynamic ranges of 32 and 64 bits is fairly limited). The design space required considering several moduli sets, each composed of one even modulus 2^{n_1} and $r-1$ odd moduli only of the type $2^{n_i}-1$, additionally with varying number of moduli r. In particular, for the dynamic range of 64 bits, the design space consists of several moduli sets with different number of moduli and such that in all cases $n_1 > n_i$, $2 \le i \le r$. As a result, Table III shows the r-moduli sets with 32- and 64-bit dynamic range, which were selected from several alternatives for each r, to optimize energy consumption. For the 64-bit dynamic range, it is clearly seen that the larger is r, the smaller is the total area and delay but also (implicitly) the slower is the reverse RNS conversion.

B. Evaluation of Power and Energy Consumption

Because the advantages of the RNS unit could be off-set by reverse conversion, we have tried to quantify it as well. For the synthesized arithmetic units, the execution of sample programs was simulated (being a data stream from the point of view of the arithmetic unit). One was a filter application with a linearly growing number of multiply-add operations while two other were matrix- and Montgomery multiplications. It seems that their expressions are general enough to reveal any differences between the power consumption of the RNS and two's complement arithmetic units, because none of them favors either representation. Note also that we consider the most simplistic implementation of the loop to make comparison as fair as possible, and that we neglect here a minor overhead resulting from an increased complexity of the instruction decoder, resulting from a few additional instructions. The power consumption was observed during execution via analysis of the VCD dumps by the PrimeTime power simulator. While the calculated power had two components—combinational and sequential (from the input and output registers), we have reported only the combinational part, which consists of both static (leakage) and dynamic power.

TABLE IV
PERFORMANCE CHARACTERISTICS FOR FILTER APPLICATION WITH 32- AND 64 BIT DYNAMIC RANGES

N	Number system	Power [mW]	Time [μs]	Energy [nJ]	Energy red. [%]
8	Two's complement/32	4.221	0.07	0.306	–
	RNS1+FC+RC	2.841	0.18	0.508	−65.77
	RNS2+FC+RC	3.168	0.18	0.565	−84.64
	Two's complement/64	11.600	0.09	1.011	–
	RNS 3M+FC+RC	7.170	0.17	1.186	−17.32
	RNS 4M+FC+RC	6.420	0.19	1.234	−22.02
	RNS 5M+FC+RC	6.614	0.21	1.407	−39.16
	RNS 6M+FC+RC	6.030	0.25	1.529	−51.21
	RNS 7M+FC+RC	5.272	0.29	1.508	−49.10
64	Two's complement/32	4.532	0.55	2.495	–
	RNS1+FC+RC	3.398	0.77	2.608	−4.50
	RNS2+FC+RC	3.757	0.77	2.880	−15.41
	Twos' complement/64	12.300	0.66	8.137	–
	RNS 3M+FC+RC	8.582	0.92	7.868	3.31
	RNS 4M+FC+RC	7.821	0.93	7.242	10.99
	RNS 5M+FC+RC	7.930	0.91	7.245	10.95
	RNS 6M+FC+RC	7.690	0.99	7.598	6.62
	RNS 7M+FC+RC	7.014	1.03	7.191	11.62
512	Two's complement/32	4.592	4.37	20.089	–
	RNS1+FC+RC	3.532	5.48	19.344	3.71
	RNS2+FC+RC	3.890	5.47	21.280	−5.93
	Two's complement/64	12.400	5.26	65.177	–
	RNS 3M+FC+RC	8.869	6.93	61.437	5.74
	RNS 4M+FC+RC	8.072	6.80	54.862	15.83
	RNS 5M+FC+RC	8.319	6.52	54.247	16.77
	RNS 6M+FC+RC	8.099	6.86	55.592	14.71
	RNS 7M+FC+RC	7.459	6.94	51.757	20.59

The first sample program calculated a sum-of-products $S = \sum_{i=0}^{N-1} x_i c_i$. For the RNS arithmetic unit, we have simulated complete filtering application with one forward conversion (FC), multiplication, and addition per loop iteration (assuming that coefficients are stored in residue form) and with one reverse conversion (RC) after the loop. All the values of samples and the coefficients at the input were random. Table IV shows obtained performance characteristics: the average power, the time of the execution of the complete program, and the total energy consumed. The last column shows the reduction percentage of energy consumption achieved over respective (32 or 64 bit) two's complement arithmetic units.

Despite the overhead of the forward- and reverse conversions in filter application putting the RNS-based implementation in a disadvantage, our design still proved superior to the positional two's complement arithmetic unit. The energy savings ranged from 3.71% in the case of minimal reduction for the filter with 32-bit dynamic range and up to 20.59% for the filter with 64-bit dynamic range. Obviously, the energy savings for a pure RNS system without taking into account conversions would be significantly larger (around 30%, depending on the number of moduli) and in this case RNS would be always more beneficial, regardless of the number of iterations.

TABLE V

PERFORMANCE CHARACTERISTICS FOR MATRIX MULTIPLICATION WITH 32- AND 64-BIT DYNAMIC RANGES

Matrix size	Number system	Power [mW]	Time [μs]	Energy [nJ]	Energy red. [%]
8×8	Two's complement/32	4.533	4.37	19.8	–
	RNS1 5M+2FC+RC	2.872	9.87	28.3	−43.04
	RNS2 5M+2FC+RC	3.169	9.86	31.2	−57.66
	Two's complement/64	12.300	5.25	64.6	–
	RNS 3M+2FC+RC	7.420	8.59	63.7	1.36
	RNS 4M+2FC+RC	6.583	10.34	68.1	−5.42
	RNS 5M+2FC+RC	6.693	11.75	78.6	−21.74
	RNS 6M+2FC+RC	6.307	14.27	90.0	−39.35
	RNS 7M+2FC+RC	5.664	16.33	92.6	−43.23
16×16	Two's complement/32	4.544	34.96	158.9	–
	RNS1 5M+2FC+RC	3.078	53.82	165.7	−4.27
	RNS2 5M+2FC+RC	3.383	53.76	181.9	−14.47
	Two's complement/64	12.300	42.01	516.7	–
	RNS 3M+2FC+RC	8.164	52.66	429.9	16.79
	RNS 4M+2FC+RC	7.192	59.27	426.2	17.51
	RNS 5M+2FC+RC	7.281	64.08	466.6	9.70
	RNS 6M+2FC+RC	6.854	74.99	514.0	0.53
	RNS 7M+2FC+RC	6.175	83.35	514.7	0.39
32×32	Two's complement/32	4.549	279.71	1272.4	–
	RNS1 5M+2FC+RC	3.340	330.10	1102.6	13.35
	RNS2 5M+2FC+RC	3.646	329.73	1202.2	5.52
	Two's complement/64	12.300	336.07	4133.6	–
	RNS 3M+2FC+RC	8.754	357.19	3126.8	24.36
	RNS 4M+2FC+RC	7.775	380.19	2956.0	28.49
	RNS 5M+2FC+RC	7.881	393.04	3097.5	25.07
	RNS 6M+2FC+RC	7.464	443.22	3308.2	19.97
	RNS 7M+2FC+RC	6.740	477.59	3219.0	22.13

TABLE VI

PERFORMANCE CHARACTERISTICS FOR MATRIX MULTIPLICATION WITH 32- AND 64-BIT DYNAMIC RANGES (WITHOUT CONVERSIONS)

Matrix size	RNS rM	Power [mW]	Time [μs]	Energy [nJ]	Energy red. [%]
8×8	RNS1	3.805	3.59	13.7	31.09
	RNS2	4.047	3.58	14.5	26.79
	3M	9.527	4.58	43.6	32.45
	4M	8.666	4.47	38.8	39.99
	5M	8.937	4.27	38.2	40.89
	6M	8.654	4.48	38.7	40.01
	7M	7.929	4.51	35.7	44.69
16×16	RNS1	3.769	28.70	108.2	31.90
	RNS2	4.054	28.67	116.2	26.84
	3M	9.582	36.63	351.0	32.06
	4M	8.705	35.78	311.5	39.72
	5M	8.983	34.18	307.0	40.58
	6M	8.696	35.82	311.5	39.72
	7M	7.994	36.04	288.1	44.23
32×32	RNS1	3.797	229.64	871.9	31.47
	RNS2	4.088	229.38	937.7	26.30
	3M	9.590	293.08	2810.6	32.01
	4M	8.714	286.26	2494.5	39.65
	5M	8.989	273.42	2457.7	40.54
	6M	8.704	286.52	2493.9	39.67
	7M	8.016	288.36	2311.5	44.08

When conversions are taken into account, there is always some break-even point due to constant overheads from the conversions: in our case, this point is in the case of hundreds of taps for filters with 32-bit dynamic ranges, and of tens of taps for the filters with 64-bit dynamic ranges.

Table V shows performance characteristics for multiplication of matrices filled with random values, while taking into account forward conversions of both input matrices (2FC) and reverse conversion (RC) of the final result. As a result, the conversion overheads of both time and energy are significantly larger due to $O(n^2)$ complexity of the conversions (the number of inputs and outputs). Nevertheless, the gain is still obtained due to the $O(n^3)$ complexity of the matrix multiplication. As for the energy consumption, in the worst case of the 8×8 matrix, there is a loss of 57.66% for the 5-moduli RNS with the 32-bit dynamic range, whereas in the best case of the 32×32 matrix and the 5-moduli RNS for the 64-bit dynamic range, the observed gain is of 28.49%. The break-even point lies somewhere around the 10×10 matrix. Without taking into account of conversions, in all cases the gain is about 35%, which seems to be the asymptotic value for the version with the conversions, because the $O(n^3)$ complexity of the matrix multiplication dominates over the $O(n^2)$ complexity of the conversions. To give a more complete picture of energy consumption, we have performed power simulations of matrix multiplication on RNS units without any conversions, and their results are shown in Table VI. It may be easily seen that there are significant savings of about 26–32% for the 32-bit dynamic range and about 32–44% for the 64-bit dynamic range. In general, the savings grow with the number of moduli that is what could be expected owing to the sizes of the multiplication units in the residue channels.

Finally, to check performance of the newly proposed architecture for computations involving very large operands such as those used in cryptography, we have performed simulation experiments for Montgomery multiplication. Amongst several methods to perform Montgomery multiplication, we have selected one given in [39], adopted to our RNS representation. This modular multiplication can be expressed by the following equation

$$\left|\frac{xy}{2^R}\right|_N = \left\lfloor \frac{xy + N\left|-\frac{xy}{N}\right|_{2^R}}{2^R} \right\rfloor,$$

where $x, y < N$, N odd, and $2^R > N$. We have used the basic version of the implementation with sequential calculation of the direct product xy followed by the modular multiplication to obtain the factor $\left|-\frac{xy}{N}\right|_{2^R}$, the calculation of the product $N\left|-\frac{xy}{N}\right|_{2^R}$, and finally the addition of two obtained terms. We have used radix-p arithmetic, where p was selected so that intermediate values can be represented using actually used RNS moduli set. The overall complexity of the operation is $O(\alpha^2)$, where α is the number of radix-p digits of the operands x and y. Conversely, the number of reverse conversions is $O(\alpha)$ and it was expected that the energy overhead from reverse conversion will be offset by the gains from multiplications. Table VII presents performance characteristics obtained using the RNS-based arithmetic units and their positional 32- and 64-bit two's complement counterparts. The column "Operand size" specifies the bit size of numbers multiplied, with the second factor as a radix, which is respectively 12- and 28-bit for 32- and 64-bit implementations. Energy savings are observed for all operand sizes of 512, 1024, and 2048 bits, with essential energy savings of over 16% and 27% obtained for 1024- and 2048-bit operands, respectively. Thus, for the basic implementation of Montgomery multiplication presented here, the results of simulation experiments indicate potential advantages of the RNS implementation over its two's complement counterpart. To note, however, that the related work in the existing literature explored several other aspects of the RNS Montgomery multiplication, so that further

TABLE VII
PERFORMANCE CHARACTERISTICS FOR MONTGOMERY MULTIPLICATION WITH 32- AND 64-BIT DYNAMIC RANGES

Operand size	Number system	Power [mW]	Time [μs]	Energy [nJ]	Energy red. [%]
512/12	Two's compl./32	4.085	40.01	163.434	–
	RNS1	3.257	49.12	159.981	2.11
	RNS3	2.728	56.46	154.035	5.75
512/28	Two's compl./64	11.300	9.55	107.896	–
	RNS 5M	6.940	16.33	113.325	−5.03
1024/12	Two's compl./32	4.148	158.93	659.249	–
	RNS1	3.473	163.03	566.193	14.12
	RNS3	2.925	187.41	548.165	16.85
1024/28	Two's compl./64	11.600	35.67	413.776	–
	RNS 5M	7.588	45.69	346.709	16.21
2048/12	Two's compl./32	4.193	626.19	2625.625	–
	RNS1	3.637	578.81	2105.138	19.82
	RNS3	3.072	665.37	2044.014	22.15
2048/28	Two's compl./64	11.700	141.54	1656.054	–
	RNS 5M	8.088	148.50	1201.053	27.47

optimization using our proposed arithmetic units is a research subject on its own and hence deserves further investigations.

C. Analysis of the Impact of Varying Precision on the Number of Executed Instructions

To convey some ideas about the impact of varying precision on the number of executed instructions required by reverse conversion, we have analyzed the RNS arithmetic processing unit with DR=64, using the optimal 7-moduli set of Table III. Simple taking into account successive residues (in the decreasing order of moduli), allows to obtain the final result on 17, 30, 41, 49, 56, 61, and 64 bits. Fig. 2 shows the number of instructions executed by the 7-moduli RNS arithmetic processing unit with DR=64 bits and varying precision. First, compared to the initial version of this paper [8], we have reduced the number of executed instructions up to about 25% for the whole DR=64 bits. Second, it is seen that the number of executed instructions is less than a half of those executed for the full dynamic range, when the required precision involves only 4 out of 7 moduli. The number of executed instructions depends on the number and selection of moduli taken into account, e.g. the number of instructions may be different if the subset $\{m_1, m_2, m_4\}$ is used for reverse conversion than for the subset $\{m_1, m_2, m_3\}$. Therefore, the varying precision required by an application can be adapted freely by suitably changing the subset of moduli for which the reverse conversion is actually executed, without affecting overall performance.

Fig. 3 details the results of the estimations of power and energy consumption obtained for various arithmetic processing units with varying precision up to DR=32: using the positional two's complement number representation and two RNS-based versions. The dashed power curves are bound to the left axis whereas their energy counterparts marked by continuous curves are bound to the right axis. The calculations used 16x16 matrix multiplications on random input values. The RNS implementations used complete forward and reverse conversions as described above. We have adjusted the precision of the calculations in the RNS arithmetic units by zeroing selected channels and implementing reverse conversion code to ignore zeroed channels, while in positional two's complement

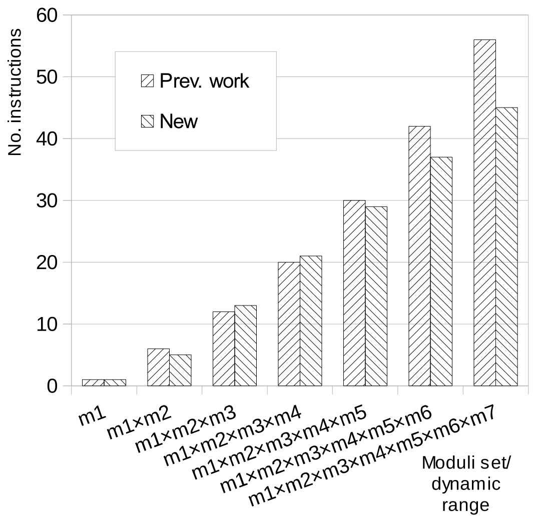

Fig. 2. Number of instructions of the reverse conversion code executed by the 7-moduli RNS arithmetic processing unit with DR=64 and varying precision.

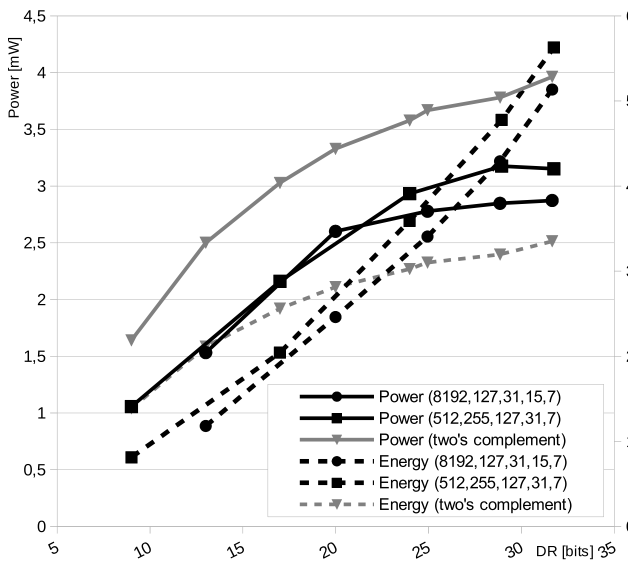

Fig. 3. Power and energy consumption of 16x16 matrix multiplication with varying precision in 32-bit arithmetic units.

units we have supplied input values such that the result never exceeded given dynamic range. The power for both positional two's complement and RNS arithmetic units clearly grows along with dynamic range and this growth is somewhat slowed nearing to maximal dynamic range, as the width of the channels is getting smaller. The energy per calculation in positional two's complement unit is directly proportional to the power (as it is the product of power and time), while in RNS arithmetic units the energy component carries yet reverse conversion cost. Accordingly, the energy consumed by RNS arithmetic units is smaller for small dynamic ranges, because the power of RNS units is small and there is simple reverse conversion which is getting larger towards higher dynamic ranges. The break-even point occurs for the dynamic range of around 20 bits. Hence, RNS arithmetic unit presents energy advantage for calculations requiring smaller dynamic

```
f_mul_m digits, (0,0,0,0,1), acc
f_residue, acc, 0, acc
f_mul_m acc, (-1, -1, -1, -1, 0), acc
f_add_m acc, digits, acc
f_mul_m acc, ( 1/m1%m5, 1/m1%m4, 1/m1%m3, 1/m1%m1, 1), digits

f_mul_m digit, (0,0,0,1,0), acc
f_residue, acc, 0, acc
f_mul_m acc, (-1/(1<<n1)%m5, -1/(1<<n1)%m4, -1/(1<<n1)%m3, 0, 0), acc
f_add_m acc, digits, acc
f_mul_m acc, ( 1/m1%m5, 1/m1%m4, 1/m1%m3, 1, 1), digits

f_mul_m digit, (0,0,1,0,0), acc
f_residue, acc, 0, acc
f_mul_m acc, (-1/(1<<(n1+n2))%m5, -1/(1<<(n1+n2))%m4, 0, 0, 0), acc
f_add_m acc, digits, acc
f_mul_m acc, ( 1/m3%m5, 1/m3%m4, 1, 1, 1), digits

f_mul_m digits, (0,1,0,0,0), acc
f_residue, acc, 0, acc
f_mul_m acc, (-1/(1<<(n1+n2+n3))%m5, 0, 0, 0, 0), acc
f_add_m acc, digits, acc
f_mul_m acc, ( 1/m4%m5, 1, 1, 1, 1), digits

f_and digits, (m5,0,0,0,0), acc
f_shr acc, n4, acc
f_sub digits, acc, digits

f_and digits, (m5,m4,0,0,0), acc
f_shr acc, n3, acc
f_sub digits, acc, digits

f_and digits, (m5,m4,m3,0,0), acc
f_shr acc, n2, acc
f_sub digits, acc, digits
```

Fig. 4. Reverse conversion assembly code for the 5-moduli set.

range while retaining the possibility of calculations using any dynamic range of up to above 31 bits, the dynamic range comparable with capability of the positional two's complement arithmetic unit.

V. Conclusion

In this paper, we have proposed a new design approach of an enhanced arithmetic unit of the processor, implemented using residue number system (RNS). Unlike no earlier design, we have proposed a shared hardware/software approach, in which a few simple modular operations are implemented in hardware, whereas reverse conversion is implemented in software, thus allowing to save static power consumption due to the latter circuit. The datapath partitioning inherent to RNS accompanied by software implementation of reverse conversion allows to fully disable some of calculation channels. The latter not only reduces both power consumption and execution time in traditional RNS applications, but also makes the architecture proposed here of interest for applications requiring dynamic changing of the computation precision. While we suggest removal of a full k-bit multiplier (k is the size of the general-purpose processor, e.g. $k = 32$ or 64 bits), we retain the possibility of having a significantly smaller positional multiplier, to implement the even residue channel mod 2^p, $p < k$. The power simulation results show that thanks to smaller modular multipliers RNS arithmetic units have smaller both area and delay, and consequently they allow to achieve up to over 20% energy saving for all three tested applications: constant-coefficient filtering, matrix multiplication, and large Montgomery multiplication, compared to executions using a positional arithmetic unit. The proposed technique may further be efficiently implemented into software compilers, making RNS implementation transparent to a programmer (at least partially). Further research will also include studying various aspects of dynamic changing precision of computations of the approach proposed.

Appendix

The reverse conversion code of Fig. 4 was written aiming at minimizing the number of instructions (it is an improved version of the one presented in [8]). While here we present the maximally shortest code we were able to obtain, we believe that further optimization is still possible. It consists of two parts. First, we calculate five MRC digits d_1 to d_5 according to Eqn (1), in four 5-instruction groups, each storing the results in a variable named *digits*. We use the multiplication by 1 in one channel with 0's in the other channels to extract the previous MRC digit, which is then used to calculate residues in the channels for more significant digits. Once all digits have been calculated, the positional value of the resulting number is obtained using a sequence of shifts and subtractions. This results in three instructions per each digit, yielding the final result in the same variable.

References

[1] A. Omondi and B. Premkumar, *Residue Number Systems: Theory and Implementation*. London, U.K.: Imperial College Press, 2007.

[2] P. Albicocco, G. C. Cardarilli, A. Nannarelli, and M. Re, "Twenty years of research on RNS for DSP: Lessons learned and future perspectives," in *Proc. 14th Int. Symp. Integr. Circuits (ISIC)*, Dec. 2014, pp. 436–439.

[3] M. F. Griffin and F. J. Taylor, "A residue number system reduced instruction set computer (RISC) concept," in *Proc. Int. Conf. Acoust. Speech, Signal Process. (ICASSP)*, vol. 4. May 1989, pp. 2581–2584.

[4] H. T. Vergos, "A 200-MHz RNS core," in *Proc. Eur. Conf. Circuits Theory Des. (ECCTD)*, vol. 2. Espoo, Finland, Aug. 2001, pp. II-249–II-252.

[5] J. Ramirez, A. Garcia, S. López-Buedo, and A. Lloris, "RNS-enabled digital signal processor design," *Electron. Lett.*, vol. 38, no. 6, pp. 266–268, Mar. 2002.

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

[7] R. Chokshi, K. S. Berezowski, A. Shrivastava, and S. J. Piestrak, "Exploiting residue number system for power-efficient digital signal processing in embedded processors," in *Proc. Int. Conf. Compilers, Archit., Synth. Embedded Syst. (CASES)*, Grenoble, France, Oct. 2009, pp. 19–27.

[8] P. Patronik and S. J. Piestrak, "Design of a low-power RNS-enhanced arithmetic unit," in *Proc. IEEE 7th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Florianopolis, Brazil, Mar. 2016, pp. 151–154.

[9] 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.

[10] S. J. Piestrak, "Design of multi-residue generators using shared logic," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Rio de Janeiro, Brazil, May 2011, pp. 1435–1438.

[11] R. A. Patel and S. Boussakta, "Fast parallel-prefix architectures for modulo $2^n - 1$ addition with a single representation of zero," *IEEE Trans. Comput.*, vol. 56, no. 11, pp. 1484–1492, Nov. 2007.

[12] J. Chen and J. Stine, "Parallel prefix Ling structures for modulo $2^n - 1$ addition," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Arch., Process.*, Boston, MA, USA, Jul. 2009, pp. 16–23.

[13] T.-B. Juang, C.-C. Chiu, and M.-Y. Tsai, "Improved area-efficient weighted modulo $2^n + 1$ adder design with simple correction schemes," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 3, pp. 198–202, Mar. 2010.

[14] R. Zimmerman, "Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication," in *Proc. IEEE Symp. Comput. Arithmetic*, Adelaide, Australia, Apr. 1999, pp. 158–167.

[15] C. Efstathiou, H. Vergos, and D. Nikolos, "Modified Booth modulo $2^n - 1$ multipliers," *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 370–374, Mar. 2004.

[16] R. Muralidharan and C. H. Chang, "Area-power efficient modulo $2^n - 1$ and modulo $2^n + 1$ multipliers for $2^n - 1$, 2^n, $2^n + 1$ based RNS," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 10, pp. 2263–2274, Oct. 2012.

[17] R. Conway and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 51, no. 1, pp. 26–28, Jan. 2004.

[18] S. J. Piestrak and K. S. Berezowski, "Architecture of efficient RNS-based digital signal processor with very low-level pipelining," in *Proc. IET Irish Signals Syst. Conf.*, Galway, Ireland, Jun. 2008, pp. 127–132.

[19] S. J. Piestrak and K. S. Berezowski, "Design of residue multipliers-accumulators using periodicity," in *Proc. IET Irish Signals Syst. Conf.*, Galway, Ireland, Jun. 2008, pp. 380–385.

[20] M. J. Schulte and E. E. Swartzlander, "A family of variable-precision interval arithmetic processors," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 387–397, May 2000.

[21] A. P. Vinod and E. M. K. Lai, "Low power and high-speed implementation of FIR filters for software defined radio receivers," *IEEE Trans. Wireless Commun.*, vol. 5, no. 7, pp. 1669–1675, Jul. 2006.

[22] K. G. Smitha and A. P. Vinod, "A reconfigurable channel filter for software defined radio using RNS," *J. Signals Process. Syst.*, vol. 67, no. 3, pp. 229–237, Jun. 2012.

[23] C. Brugger *et al.*, "Mixed precision multilevel Monte Carlo on hybrid computing systems," in *Proc. IEEE Conf. Comput. Intell. Financial Eng. Econ. (CIFEr)*, Mar. 2014, pp. 215–222.

[24] C. D. Schryver, Ed., *FPGA Based Accelerators for Financial Applications*. Cham, Switzerland: Springer, 2015.

[25] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. W. Leong, and D. B. Thomas, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, Monterey, CA, USA, Feb. 2012, pp. 57–66.

[26] J. Oliver, V. Akella, and F. Chong, "Efficient orchestration of sub-word parallelism in media processors," in *Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit. (SPAA)*. New York, NY, USA, 2004, pp. 225–234.

[27] J. Park, J. Choi, and K. Roy, "Dynamic bit-width adaptation in DCT: An approach to trade off image quality and computation energy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 787–793, May 2010.

[28] A. Madanayake *et al.*, "Low-power VLSI architectures for DCT/DWT: Precision vs approximation for HD video, biomedical, and smart antenna applications," *IEEE Circuits Syst. Mag.*, vol. 15, no. 1, pp. 25–47, 1st Quert., 2015.

[29] J. Hormigo, J. Villalba, and E. L. Zapata, "CORDIC processor for variable-precision interval arithmetic," *J. VLSI Signal Process.*, vol. 37, no. 1, pp. 21–39, May 2004.

[30] S. Yoshizawa and Y. Miyanaga, "Use of a variable wordlength technique in an OFDM receiver to reduce energy dissipation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 9, pp. 2848–2859, Oct. 2008.

[31] N. Nakasato *et al.*, "GRAPE-MPs: Implementation of an SIMD for quadruple/hexuple/octuple-precision arithmetic operation on a structured ASIC and an FPGA," in *Proc. IEEE 6th Int. Symp. Embedded Multicore SoCs (MCSoC)*, Sep. 2012, pp. 75–83.

[32] Y. Emre and C. Chakrabarti, "Energy and quality-aware multimedia signal processing," *IEEE Trans. Multimedia*, vol. 15, no. 7, pp. 1579–1593, Nov. 2013.

[33] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 5, pp. 497–510, May 2004.

[34] Y.-H. Huang, "High-efficiency soft-error-tolerant digital signal processing using fine-grain subword-detection processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 291–304, Feb. 2010.

[35] H. Kaul *et al.*, "A 1.45 GHz 52-to-162 GFLOPS/W variable-precision floating-point fused multiply-add unit with certainty tracking in 32 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Techn. Papers*, Feb. 2012, pp. 182–184.

[36] U. Meyer-Baese, *Digital Signal Processing With Field Programmable Gate Arrays*, 4th ed. Berlin, Germany: Springer, 2014.

[37] K. H. Rosen, *Elementary Number Theory and Its Application*, 6th ed. Boston, MA, USA: Addison-Wesley, 2011.

[38] M. Abdallah and A. Skavantzos, "A systematic approach for selecting practical moduli sets for residue number systems," in *Proc. Syst. Theory, 27th Southeastern Symp.*, Mar. 1995, pp. 445–449.

[39] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, Jun. 1996.

Piotr Patronik received the M.Sc. and Ph.D. degrees in computer engineering from the Wrocław University of Technology in 2001 and 2006, respectively. He is currently an Assistant Professor with the Faculty of Electronics, Wrocław University of Technology, Wrocław, Poland. During academic year 2012–2013, he was on leave with the Institut Jean Lamour, Université de Lorraine, Nancy, France, where he was involved in the ARDyT Project funded by the ANR. His research interests include design of VLSI digital circuits, computer arithmetic, fault-tolerant computing, and parallel computing.

Stanisław J. Piestrak received the Ph.D. degree in computer science from the Wrocław University of Technology in 1982 and the Habilitation degree in computer science from the Gdansk University of Technology, Poland, in 1996. He is currently a Professor with the Institut Jean Lamour/Université de Lorraine, Nancy, France. His research interests include design of VLSI digital circuits, fault-tolerant computing (self-checking circuits design, coding theory, and reconfigurable systems), and computer arithmetic (design of hardware for applications using residue number system).