Fast and Energy-Efficient Constant-Coefficient FIR Filters Using Residue Number System

Piotr Patronik[1], Krzysztof Berezowski[1], Stanisław J. Piestrak[2], Janusz Biernat[1], Aviral Shrivastava[3]
[1]Institute of Computer Engineering, Control, and Robotics Wrocław University of Technology, Wrocław, Poland
{*piotr.patronik,krzysztof.berezowski,janusz.biernat*}*@pwr.wroc.pl*
[2]IRISA, 6 rue de Kerapont, Lannion, France, *piestrak@univ-metz.fr*
[3]Compiler Microarchitecture Lab, Arizona State University, Tempe AZ, 85287, USA, *aviral.shrivastava@asu.edu*

***Abstract*—In this paper, we present constant-coefficient finite impulse response (FIR) filters design using residue number system (RNS) arithmetic. The novelty of our approach rests in an attempt to maximize the accumulated benefit of the application of RNS to the design of constant coefficient filters. To achieve this, we consider the impact of RNS on many layers: from coefficient representation and techniques of sharing of subexpressions in the multiplier block (MB), to its optimized usage in the MB and accumulation pipeline hardware design. As a result, we propose a common subexpression elimination (CSE) based synthesis technique for RNS-based MBs, along with a high-performance RNS-based FIR filter architecture that employs RNS arithmetic principles but implements them mainly using more efficient 2's complement hardware. Several filters with numbers of taps ranging from 25 to 326 and dynamic ranges from 24 to 50 bits have been synthesized using TSMC 90 nm LP kit and Cadence RTL Compiler. Comparison of power, delay, and area of the new filters implemented using the 4- and 5-moduli RNSs against various equivalent 2's complement counterparts show uniform improvement in performance and power efficiency, often accompanied by significant reduction in area/power consumption as compared to 2's complement implementations. We observed up to 22% improvement in peformance (19% reduction in area) within bounded power envelope, or up to 14% reduction in power consumption (12% reduction in area) at same frequency.**

I. Introduction

Finite impulse response (FIR) filters are typically performance critical components of digital signal processing (DSP) applications. In many cases their bandwidth and power consumption requirements can only be met by custom application-specific implementations that trade off flexibility for high performance and power efficiency by hardwiring filter's coefficients directly into the filter hardware.

The problem of designing FIR filters directly from their coefficients has been thoroughly researched over the years [1], [5], [6], [18], [20]. As a result, it is well known that in the transposed form of the FIR filter (Fig. 1a) all multiplications can be integrated into a single *multiplier block* (MB) with its design abstracted as a *multiple constant multiplication (MCM)* problem: given a set of n coefficients $\{c_1, \ldots, c_n\}$ and an input sample x_k, compute simultaneously all products $\{c_1 \cdot x_k, \ldots, c_n \cdot x_k\}$. The resultant structure can be optimized in at least three ways. First, the multiplications can be calculated as sums of shifted operands, with shifts hardwired at no hardware cost. Second, the number of partial products can be reduced by expressing the coefficients in the canonical signed digit (CSD) representation, which guarantees that at least half of their bits are zero [10]. Finally, partial sums of different products can be aggregated for sharing in order to reduce the total number of additions necessary to compute all products. Over the years, that last problem has drawn a significant amount of attention and a variety of sharing methods have been proposed [1].

Consequently, many methods of subexpression sharing in MBs are available — many of them recently surveyed in [1]. They fall into two major categories: graph dependence (GD) methods, e.g. [20], and common subexpression elimination (CSE) methods, e.g. [5]. Either approach attempts to exploit the observation that the more partial sums are shared among products, the more adders are eliminated from the design. In GD methods, partial sums are represented as nodes of a graph, what typically results in greater reduction in the number of adders at the expense of typically longer critical path. In CSE, partial sums are expressed in terms of sums of shifted operands, what usually yields more additions required to compute all products, but allows for explicit control over addition depth [5], thus for trading off performance for complexity.

For the purpose of high-performance yet power-efficient FIR filter design, the Residue Number System (RNS) has long been of interest [14]. Capitalizing on the premises of the Chinese Remainder Theorem (CRT), RNS decomposes the filter structure into a number of parallel and independent *modulo channels*, each being a transposed form filter itself (Fig. 1b). Such a decomposition results in reduced magnitude of per channel operands, therefore enjoys reduced carry propagation and small partial product matrices in the distributed per channel multiplications and additions. RNS however does come with some disadvantages: most prominently, the binary-to-residue (forward) and residue-to-binary (reverse) conversions form an overhead necessary to interact with 2's complement system (TCS) datapath. However, voluminous evidence demonstrates that in the multiply-intensive applications like FIR filters, the advantages of RNS outweigh its overheads and the application of RNS results in performance or/and power-efficiency improvements [8], [12], [13], [16], [18], [19].

Most of the existing literature on RNS-based hardware considers the design of programmable filters. Only few works consider constant-coefficient filters and multipliers by constant using RNS and related hardware [6], [7], [13], [18], [19], [21]. In [19], very high order RNS-based FIR filters with multipli-

978-1-61284-660-6/11/$26.00 © 2011 IEEE

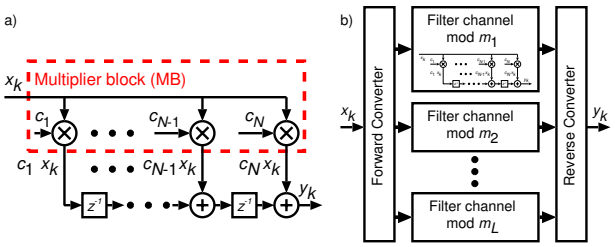

Fig. 1. FIR filter structures: a) Transposed form; b) RNS-based

ers by constant coefficients implemented using custom logic were proposed. The general theory of computing constant-coefficient inner products in RNS taking advantage of the periodicity properties of the series of 2^k taken modulo odd m was presented in [21]. Recently, [13] and [18] discuss a complete filter synthesis of TCS and RNS implementations of both programmable and constant-coefficient FIR filters. However, in either work, the multiplication by constant was implemented with a lookup table and neither exploits other properties of the RNS but the simple coefficient decomposition inherent to RNS representations. A discussion of parallel FIR filters employing number theoretic transforms (NTTs) is given in [7]. However, to the best of our knowledge the implications of using RNS on performance of MCM and constant-coefficient filters in general have not been sufficiently studied yet, though such possibility was mentioned in [6], [18].

II. The Contribution of this Paper

In this paper, we comprehensively revisit all problems related to the RNS-based design of the constant-coefficient filters. We study the impact of RNS on all aspects of such design: the MCM problem, the MB architecture and implementation, and the transposed form accumulation pipeline implementation. As a result, we propose a holistic design framework that rests on our three novel contributions: (i) augmented CSD representations of the constant coefficients that exploits the periodicity properties of the modulo integer constants; (ii) the modifications to the level-constrained CSE algorithm [5] that utilizes these properties for more efficient design of RNS-based MBs; and finally, (iii) the novel MB and accumulation pipeline architectures that, as compared to many alternatives that we have evaluated, minimize the hidden costs of RNS arithmetic hardware. The net result of this effort is the filter synthesis framework that provides uniformly higher performance as compared to TCS implementations, typically with extra improvement in area and/or power consumption.

III. Preliminaries

A. Finite Impulse Response (FIR) Filter

The response of an FIR filter of an order N is given by

$$y(n) = \sum_{k=0}^{N-1} c_k x_{n-k}. \tag{1}$$

The above equation is typically implemented in hardware in the transposed form (Fig. 1a) as it enjoys higher performance and modularity. The RNS realization follows essentially the same hardware structure in each of L channels computing the filter function modulo on the reduced magnitude residues. Additionally, the forward and the reverse converters are necessary to interact with TCS components (Fig. 1b).

B. Residue Number System (RNS)

An RNS is defined by a set of L positive integer *moduli* $\{m_1, m_2, \ldots, m_L\}$, which are pairwise relatively prime. The *dynamic range* M of such RNS equals to $M = \prod_{i=1}^{L} m_i$. The operation of RNS is the consequence of the Chinese Remainder Theorem (CRT) which states that an L-tuple $\{X_1, X_2, \ldots, X_L\}$, where: $X_i = X \bmod m_i = |X|_{m_i}$, $i = 1, 2, \ldots, L$, uniquely represents an integer $|X|_M$. More importantly, when two integers $|X|_M$ and $|Y|_M$ are respectively represented by the sets of residues X_i, Y_i, $i = 1, 2, \ldots, L$, then a set of residues Z_i, $i = 1, 2, \ldots, L$ computed as $Z_i = |X_i \circ Y_i|_{m_i}$, where $\circ \in \{+, -, \times\}$, uniquely represents $Z = |X \circ Y|_M$. Consequently, an RNS datapath can compute interleaved series of additions and multiplications of integers in L parallel and independent channels as the interleaved series of modulo additions and multiplications of their residues. Also, we call two integers X and Y *congruent* mod m_i, or $X \equiv Y \pmod{m_i}$, if $|X|_{m_i} = |Y|_{m_i}$.

The complexity, the delay, and the power consumption of the modular arithmetic hardware relate strongly to the periodicity of the series $|2^k|_{m_i}$. While it is periodic for any odd modulus m_i, odd moduli with small *period* $P(m_i)$ and *half-period* $HP(m_i)$ parameters typically enjoy low implementation costs [17]. Particularly, for every non-negative integer $k, j : k < n$, the low-cost moduli $2^n - 1$ and $2^n + 1$ enjoy

$$2^{k+nj} \equiv |2^k|_{2^n-1}, \quad n \geq 2 \tag{2}$$

$$2^{k+nj} \equiv |(-1)^j 2^k|_{2^n+1}, \quad n \geq 1. \tag{3}$$

what yields very efficient implementations of many arithmetic circuits [2], [4], [6], [11], [12], [16], [17], [21].

Let $X = (x_{n-1} \ldots x_0)$ be an integer mod $2^n - 1$ (or $X = (x_n x_{n-1} \ldots x_0)$ — an integer mod $2^n + 1$), k — a positive integer, and $\bar{X} = (\bar{x}_{n-1} \ldots \bar{x}_0)$ — a bitwise negation of X.

$$|2^k X|_{2^n-1} \equiv (x_{n-k-1} \ldots x_0 x_{n-1} \ldots x_{n-k}) \tag{4}$$

$$|-X|_{2^n-1} \equiv (\bar{x}_{n-1}, \ldots, \bar{x}_0) \tag{5}$$

$$|2^k X|_{2^n+1} \equiv |(x_{n-k-1} \ldots x_0 \bar{x}_{n-1} \ldots \bar{x}_{n-k}) + 2^k - 1|_{2^n+1} \tag{6}$$

$$|-X|_{2^n+1} \equiv |(\bar{x}_n \ldots \bar{x}_0) + 2|_{2^n+1} \tag{7}$$

Finally, it is obvious that

$$|2^k X|_{2^n} = (x_{n-k-1} \ldots x_0 \overbrace{0 \ldots 0}^{k\ 0's}). \tag{8}$$

C. Canonical Signed-Digit Representation

In the multiplication by constant, the number of non-zero bits of the constant determines the number of partial products being added. This number can be reduced by converting filter coefficients to a redundant signed-digit (SD) representation [10] which allows for both positive and negative digits at each bit position (1 and $\bar{1}$). Specifically, its canonical form

(CSD) guarantees the minimum number of non-zero bits in the representation. The conversion of an n-bit number into at most $(n+1)$ bits of the CSD representation is performed by scanning the bits from the least significant bit (LSB) up to the most significant bit (MSB) and replacing the continuous non-zero bit sequences by pairs of non-zero positive and negative bits: $11\ldots1 \rightarrow 100\ldots\bar{1}$. Then the remaining pairs of adjacent bits $\bar{1}1$ are replaced by a pair $0\bar{1}$. The resultant CSD form behaves much like a regular binary representation and can be used in modular calculations as well. In particular, Equations (4), (6), and (8) also hold for the (C)SD representation [16].

IV. Constant-Coefficient FIR Filter Design

A. CSD Representation of Modulo Integers

Although coefficient residues used in modulo channels are simply integers and can be converted to the CSD representation using the standard procedure described above, the fact that they actually are integers mod $2^n \pm 1$ allows us to take advantage of Equations (2) and (3). As a consequence, the MSB of the coefficient residue encoded into a standard CSD form can be considered as a signed digit of the weight 2^0 and added to the remainder of the CSD representation. The resultant SD form can again be reduced to the canonical form which represents a value congruent to the original residue mod $2^n \pm 1$ and may have less non-zero bits that the standard form.

Example 1: Consider a CSD representation of $|27|_{31}$. The binary representation of 27 is $(011011)_b$. By reducing adjacent groups of 1s we obtain $(10\bar{1}10\bar{1})_{SD} = (100\bar{1}0\bar{1})_{CSD}$ with three non-zero bits. However, knowing that 27 is an integer mod 31 we can use the fact that $32 \equiv 1 \bmod 31$, i.e., the MSB and LSB may cancel themselves out. The resultant value $(000\bar{1}00)_{CSD} = -4$ is congruent to 27 mod 31 and has only one non-zero digit. Similarly, a standard CSD form of $|29|_{33}$ is $(100\bar{1}01)_{CSD}$. Again, if 29 is considered to be an integer mod 33, $32 \equiv -1 \bmod 33$, and the resulting congruent value with best CSD representation is $(000\bar{1}00)_{CSD} = |-4|_{33}$.

B. RNS LCCSE algorithm

In the next step, we construct a modular MCM, i.e., derive an MB structure for each residue channel. In fact, any existing MCM technique could be used to compute such an MCM structure. However, similarly to computing the CSD form, we can again exploit the fact that coefficients are integers modulo $2^n \pm 1$, although it requires changes in existing algorithms. For our purposes, we augment the existing level-constrained CSE (LCCSE) algorithm of [5] in order to compute modular MCMs. It is worth noticing that although some algorithms for constructing MBs for RNS FIR filters exist [6], [18], none of them directly addresses the problem of subexpression sharing. Therefore, to the best of our knowledge, our modification of [5] is the first CSE algorithm for modular MCMs.

The comprehensive coverage of the LCCSE can be found in [5]. For our purposes let us briefly say that this algorithm iteratively decomposes the values from the set of values waiting to be decomposed $\{UDS\}$ and puts them into the set of decomposed values $\{DS\}$. Initially, $\{UDS\}$ contains all filter coefficients. In each step, the algorithm picks a value c_k from $\{UDS\}$ and attempts to decompose it into shifts n_1 and n_2 of two values d_1 and d_2 from $\{UDS\} \cup \{DS\} \cup \{1\}$, such that $c_k = \pm d_1 \cdot 2^{n_1} \pm d_2 \cdot 2^{n_2}$.

If the step succeeds, and the decomposition meets the level constraint (the maximum number of additions on the critical path of the block), the result is put into the set $\{DS\}$. If after this step $\{UDS\}$ is empty, the number of adders is minimal and the procedure ends. Otherwise, it is repeated using terms from the set $\{APCS\}$, which is the set of all candidates which can be used to decompose values in $\{UDS\}$. The $\{APCS\}$ is initialized by extracting all the possible combinations of nonzero terms of CSD numbers in $\{UDS\}$.

Our modifications to the LCCSE flow essentially affect the way the values from $\{UDS\}$ are decomposed (lines 5 and 11 in Fig. 2). The impact of these modifications is twofold. First, instead of decomposing filter coefficients directly, we consider their *bases*, i.e., the values that for some integer k minimize the value of an expression

$$b_i = \begin{cases} |\pm 2^k c_i|_{2^n-1} \text{ (channel modulo } 2^n-1) \\ |\pm 2^k c_i|_{2^n+1} \text{ (channel modulo } 2^n+1). \end{cases}$$

Two coefficients c_a, c_b either share the same base or have two distinct bases. Given a set of unique bases of all coefficients, any coefficient may be computed solely through bit rotation and inversion, which we assume to have negligible hardware cost. Therefore, in our modification of the LCCSE algorithm, the set $\{UDS\}$ is initialized with a set of bases b_i rather than the set of coefficients c_k. Then, in each decomposition step, the bases from $\{UDS\}$ are decomposed in a way that accounts for their modular interpretation

$$c_k = \begin{cases} |\pm d_1 \cdot 2^{n_1} \pm d_2 \cdot 2^{n_2}|_{2^n-1} \pmod{2^n-1} \\ |\pm d_1 \cdot 2^{n_1} \pm d_2 \cdot 2^{n_2}|_{2^n+1} \pmod{2^n+1}. \end{cases}$$

Example 2: Consider the modulus 63 and the 2-element set of coefficients $5 = (000101)_b$ and $29 = (011101)_b$. Inversion

Require: Coefficients $c_1 \ldots c_N$, Levels $L_1 \ldots L_k$
1: $\{DS\} = \phi$
2: $\{UDS\}$ = coefficient set
3: $\{APCS\}$ formed from $\{UDS\}$
4: **repeat**
5: Decompose $\{UDS\}$ numbers with $\{UDS\} \cup \{DS\} \cup \{1\}$
6: Perform level constraint check
7: Move decomposed numbers from $\{UDS\}$ to $\{DS\}$
8: **until** No decomposition possible
9: **if** $\{UDS\} \neq \phi$ **then**
10: **repeat**
11: Find all possible decompositions of $\{UDS\}$ numbers with $\{UDS\} \cup \{DS\} \cup \{1\} \cup \{APCS\}$
12: Perform level constraint check
13: Find the smallest subset $\{S\}$ of $\{APCS\}$ to cover all decomposable $\{UDS\}$ numbers
14: Move decomposed numbers from $\{UDS\}$ to $\{DS\}$
15: Move elements of $\{S\}$ from $\{APCS\}$ to $\{UDS\}$
16: **until** No further decomposition possible
17: **end if**{ $\{UDS\} = \phi$ }
Ensure: $\{DS\}$ decompositions

Fig. 2. The modified LCCSE algorithm

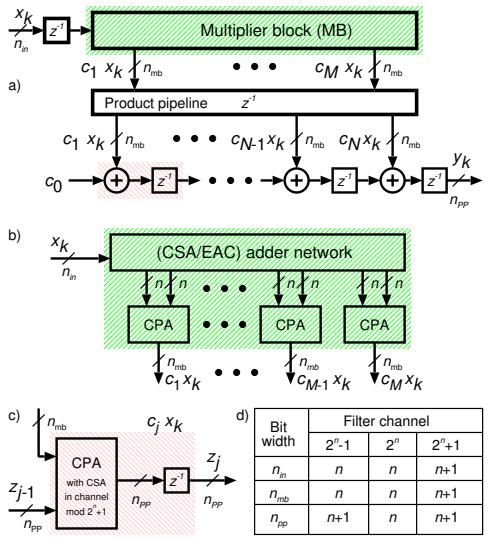

Fig. 3. Implementation of single filter channel: a) general structure; b) multiplier block; c) transpose stage; d) bit-widths of signals in channels

of bits of 29 yields $(100010)_b = 34 = |-29|_{63}$. Rotating this vector by one position left yields $(000101)_b$, i.e. 5. Thus the base of 29 is 5. Therefore, 5 becomes the base for the set.

This results in potentially smaller initial $\{UDS\}$ as well as potentially larger choice of decompositions. Moreover, remember that the coefficients in each channel are already low magnitude residues of the original filter coefficients. Together with their potentially more compact CSD forms, the probability of sharing a partial sum constructed using the modified LCCSE is higher than for the unmodified method. Moreover, due to reduced magnitude of the coefficient residues, it is more likely that the residues of different integer coefficients are identical, what may lead to fewer unique MB outputs.

C. Multiplier Block Architecture

The addition graph computed using the modified LCCSE has to be implemented as an MB. Similarly to [3], we use carry-save adders (CSA) combining carry-save (CS) representation with carry-propagate merging adders (CPA) to achieve high power efficiency. We also take advantage of the results of [9] that some CS pairs may be constructed using one or just no CSA at all. In the MB, an output of any adder is the result of the multiplication by some constant. These intermediate results are called *fundamentals* [9]. Depending on their values, we consider three types of fundamentals. Let f written as $\{f_c, f_s\}$ in the CS form be a fundamental, x_k — an input value, and $0 \leq a, b, 2^i, 2^j < 2^n \pm 1$ — integers corresponding to the decomposed coefficients.

1) Fundamentals of the form $(\pm 2^i \pm 2^j)x_k$ are created by simply assigning $\{f_c, f_s\} = \{\pm 2^i x_k, \pm 2^j x_k)$.
2) Fundamentals of the form $\pm 2^i x_k \pm a x_k$, where the vector ax_k is a fundamental in the CS form, and $\pm 2^i x_k$ is in the binary form are created by adding the three vectors in one CSA layer, i.e. $\{f_c, f_s\} = \pm 2^i x \pm a x_k$.
3) Fundamentals of the form $ax_k + bx_k$, where both vectors ax_k and bx_k are fundamentals in the CS form, are created by adding the four vectors in two CSA layers, i.e. $\{f_c, f_s\} = ax_k + bx_k$.

Since in $2^n \pm 1$ channel all additions have to be performed using end-around-carry (EAC) adders, it is typically of preference to use cheaper EAC-CSA structures where possible, and costly EAC-CPAs only when necessary [17]. Equations (4) through (7) state that the multiplications by signed bits of coefficients are relatively simple (especially in the case of $2^n - 1$ modulus). In channel modulo $2^n + 1$, computations are slightly more complicated as shifts and adds in the manner of $2^n - 1$ channel can only be performed on lower n bits of x_k, while the special case of $x_k = 2^n$ needs to be dealt with separately. Therefore, in the $2^n + 1$ channel the fundamentals 1 and 2 have to be calculated as follows:

$$1.\quad \{f_c, f_s\} = \begin{cases} \{\pm 2^i x_k, \pm 2^j x_k\} & \text{when } x_k < 2^n \\ \{\mp 2^i, \mp 2^j\} & \text{when } x_k = 2^n \end{cases}$$

$$2.\quad \{f_c, f_s\} = \begin{cases} \pm 2^i x_k + a x_k & \text{when } x_k < 2^n \\ \mp 2^i - a & \text{when } x_k = 2^n \end{cases}$$

The third case, where fundamentals are already in CS form, requires no special treatment, while cases 1 and 2 require multiplexers to switch outputs when $x_k = 2^n$. Finally, the constants generated by inversions in channel modulo 2^n+1 are accumulated and added at the beginning of the filter pipeline as c_0 in Fig. 3a. To obtain the multiplication result, the carry-save form is merged into a single result using CPAs. In the channel mod $2^n - 1$, the adder mod $2^n - 1$ of [15] is used and produces an n-bit wide result. In the channel mod $2^n + 1$, the residue is $(n+1)$-bit wide, and since addition mod $2^n + 1$ is less efficient than mod $2^n - 1$ [11], we employ a regular binary adder instead of a mod $2^n + 1$ adder, thus producing an $(n + 1)$-bit value congruent to the actual residue.

D. Filter Pipeline Architecture

The outputs from the MB can be either stored in the product pipeline registers for performance (Fig. 3a) or fed directly to the accumulation stages of the filter pipeline. Each stage adds two values (Fig. 3c): one from the product pipeline and one from the previous pipeline stage. The following notation is used: z_j—accumulated filter value in stage j (z_j is $n + 1$-bit value), $z_{j,i}$—the i-th bit of z_j, x_k—the input value to the filter, c_j—the j-th filter coefficient (in modulo channel), u_j—the output of multiplier block (in general $u_j = x_k c_j$). In both channels modulo $2^n \pm 1$ we have

$$z_j \equiv z_{j-1} + u_j \pmod{2^n \pm 1} \tag{9}$$

where

$$u_j = \begin{cases} (u_{j,n-1} \dots u_{j,0}) & (\text{mod } 2^n - 1) \\ (u_{j,n} \dots u_{j,0}) & (\text{mod } 2^n + 1). \end{cases}$$

Thanks to periodicity, the terms z_j, u_j are written as

$$z_j \equiv \begin{cases} (z_{j,n-1} \dots z_{j,0}) + z_{j,n} & (\text{mod } 2^n - 1) \\ (z_{j,n-1} \dots z_{j,0}) + \bar{z}_{j,n} - 1 & (\text{mod } 2^n + 1) \end{cases} \tag{10}$$

TABLE I
BENCHMARK FILTERS AND MODULI SELECTION

Filter name	w_{pass}/w_{stop} (method)	N	input	DR	n 4M	n 5M
f4	0.15/0.25 (PM)	41	12	24.7	6	-
f5	0.4/0.6 (PM)	26	12	24.7	6	-
f6	0.27/0.29 (LS)	327	16	41.3	-	9
f7	0.27/0.2875 (PM)	72	24	49.2	-	10

PM – Parks-McClellan, LS – least squares

USED RNSES	
Name	Moduli set
4M	$2^n-1, 2^n, 2^n+1, 2^{n+1}-1$ (n even)
5M	$2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1$ (n even)
	$2^n-1, 2^n, 2^n+1, 2^{n+1}+1, 2^{n-1}+1$ (n odd)

$$u_j \equiv (u_{j,n-1} \dots u_{j,0}) + \bar{u}_{j,n} - 1 \pmod{2^n+1}. \quad (11)$$

Substituting, for the mod 2^n-1 channel, we obtain

$$z_j \equiv \begin{pmatrix} (z_{j,n-1} \dots z_{j,0}) + & \\ (u_{j,n-1} \dots u_{j,0}) + & z_{j,n} \end{pmatrix} \pmod{2^n-1}. \quad (12)$$

Substituting, for the mod 2^n+1 channel, we obtain

$$z_j \equiv \begin{pmatrix} (z_{j,n-1} \dots z_{j,0}) + & \\ (u_{j,n-1} \dots u_{j,0}) + & \\ \bar{z}_{j,n} + \bar{u}_{j,n} - & 2 \end{pmatrix} \pmod{2^n+1}. \quad (13)$$

Our final implementation employs an n-bit binary adder with carry-in for the mod 2^n-1 addition in (12), producing an $(n+1)$-bit value. In mod 2^n+1 channel, we add two n-bit vectors and the two bits of weight 2^0 from (13) using one CSA layer ($(n-1)$-HAs and one FA) with inverted EAC followed by the n-bit binary adder with carry-in. The -2 constant gets accumulated with all other constants and added to c_0. Such an architectural choice balances the high speed and high power consumption of the CSA- vs. lower speed and lower power consumption of the CPA-based design.

V. SYNTHESIS RESULTS

We implemented our proposed design methodology in a synthesis tool and synthesized a selection of benchmarks from [1]. The design parameters of the evaluated filters and the RNS selection are summarized in Table I. For comparison, we generated a number of TCS-based implementations from both academic and industrial tools: the Spiral project [20] (Spiral), Matlab FDATool HDL Coder (Matlab), and the implementation that Cadence RTL Compiler derived automatically from a high-level Verilog HDL code (RTL). The complete benchmark set was synthesized using Cadence RTL Compiler v. 8.10 over the TSMC 90nm low power library for both minimum delay, as well as the selected set of arbitrary timing constraints.

It is important to explain why we chose to compare our implementation solely to TCS implementations. We do so for two reasons. First, most other works on RNS-based constant-coefficient FIR filter design deal with abstracted subproblems only (like MCM design), typically without proposing the entire filter architecture and/or design methodology, whereas we found evaluating these choices crucial to the success of any approach. Second, among the two works that deal with complete filter synthesis [13], [18], neither gives enough details to fully reproduce the experimental setup. Additionally, the analysis of the results we obtained for different TCS implementations has made us conclude that most of the performance advantages reported by [18] over Spiral is a sheer result of Spiral's poor performance. And [13] does not even clearly state what kind of TCS implementation was used for the comparison.

TABLE II
FILTERS WITH MINIMAL DELAY

Filter		Delay (D) [ns]	Area [mm^2]	Power (P) [mW]	P×D [ns×mW]
f4	RTL	1.190	0.114	26.27	31
	Spiral	1.625	**0.099**	**23.02**	37
	Matlab	1.230	0.116	26.27	32
	RNS	**1.065**	0.110	25.56	**27**
f5	RTL	1.180	**0.072**	**16.21**	**19**
	Spiral	1.597	0.077	18.49	30
	Matlab	1.187	0.078	17.88	21
	RNS	**1.063**	0.078	18.66	20
f6	RTL	1.589	1.810	441.54	702
	Spiral	4.744	**1.166**	**252.14**	1196
	Matlab	1.966	1.642	384.66	756
	RNS	**1.505**	1.507	373.25	**562**
f7	RTL	1.650	0.529	132.78	219
	Spiral	2.812	0.491	**106.57**	300
	Matlab	1.887	0.522	123.50	233
	RNS	**1.548**	**0.463**	130.45	**202**

All benchmarks circuits we implemented using the architecture from Fig. 3a. Notice that for the TCS filters each tap dynamic range is adjusted to the actual tap requirements, whereas for the RNS filters the output dynamic range has to be maintained along the pipeline. Consequently, the average tap width is narrower in TCS filters than in the RNS filters, what significantly impacts the power consumption.

Our results detailed in Table II show that our design is uniformly faster than any of the TCS implementations. While this may seem to be an obvious result of datapath decomposition, both our explorations, as well as the results of other authors [18] show that such advantage often comes with major area/power consumption penalty. Therefore, the more important conclusion from Table II is that RNS-based implementation yields uniformly higher performance at no or moderate area/power penalty, often along with a significant improvement in power efficiency measured as Power-Delay product. In fact, if Spiral results were removed from the set (because of the lower performance), RNS would become competitive to the evaluated TCS implementations also in terms of power and area. A little more insight into the results are given in Fig. 4, where power-frequency and area-frequency characteristics of different implementations are depicted. The plots show clearly although the quality of the RTL compiler optimizations are very high across the whole frequency range, in the high performance corner RNS-based filters manifest their structural advantages clearly in all metrics. For lower speeds, the RNS implementations cannot really compete because its structural code does not scale as well as the RTL code of the competitors.

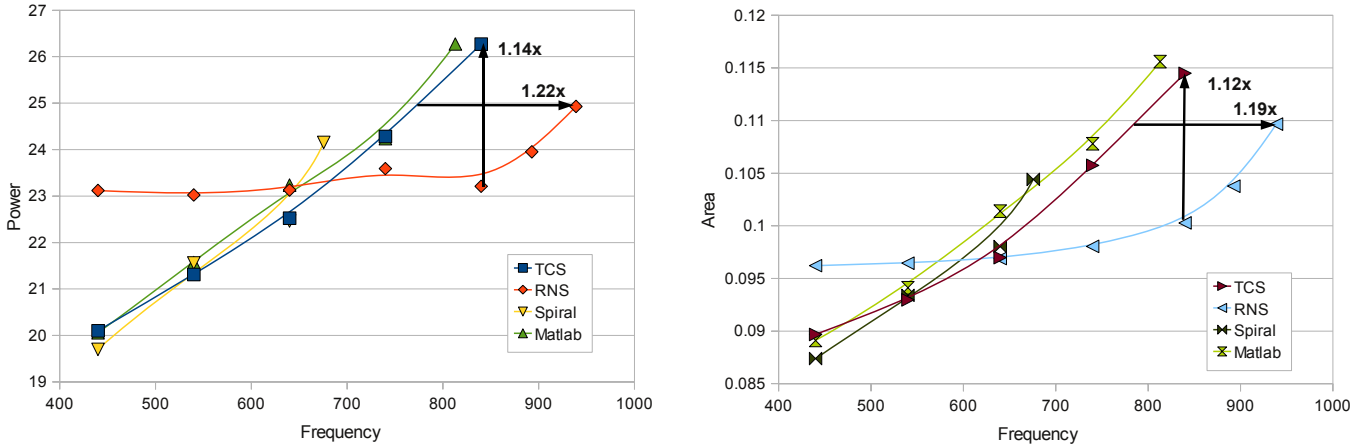

Fig. 4. Power-frequency and area-frequency characteristics of the filter *f4*

VI. Conclusions

We have presented a synthesis framework for high performance, energy efficient, RNS-based, contant-coefficient FIR filters. It comprises a novel representation of modulo constants, an augmented level-constrained subexpression elimination algorithm, and a set of architectural and implementation choices.

In the course of our effort, we found the advantages of RNS far more challenging to exploit for the design of constant-coefficient filters than in the case of the programmable filters, where the benefits come naturally from the decomposition of the integrated multiply-and-accumulate units. The challenge comes from two facts: (i) in the CSE-optimized MB the amount of arithmetic hardware is reduced therefore its efficiency has lesser impact on the overall efficiency and (ii) the TCS implementations of MB that utilize CSE already yield reduced complexity of the TCS implementations of MBs.

Nevertheless, we have demonstrated that the careful application of RNS yields delay and energy efficiency improvements that significantly supersede TCS implementations. This is of ultimate importance because the applications of constant-coefficient filters are primarily in high performance domain, often with a tight power envelope. Our results show this superiority clearly, over a handful of different implementations and filter structures. Unlike other researchers [18], we achieved the improvements without area and/or power dissipation penalty.

References

[1] L. Aksoy, E. Da Costa, P. Flores, and J. Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Trans. CAD*, 27(6):1013–1026, June 2008.

[2] P. Ananda Mohan and A. Premkumar. RNS-to-binary converters for two four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$. *IEEE Trans. Circuits Syst. I*, 54(6):1245–1254, June 2007.

[3] V. Bartlett and A. Dempster. Using carry-save adders in low-power multiplier blocks. In *Proc. ISCAS*, volume 4, pages 222–225, 2001.

[4] B. Cao, C.-H. Chang, and T. Srikanthan. A residue-to-binary converter for a new five-moduli set. *IEEE Trans. Circuits Syst. I*, 54(5):1041–1049, May 2007.

[5] J. H. Choi, N. Banerjee, and K. Roy. Variation-aware low-power synthesis methodology for fixed-point FIR filters. *IEEE Trans. CAD*, 28(1):87–97, Jan. 2009.

[6] R. Conway. Reducing complexity of fixed-coefficient FIR filters. *Electron. Lett.*, 42(20):1185–1186, 2006.

[7] R. Conway. Efficient residue arithmetic based parallel fixed coefficient FIR filters. In *Proc. ISCAS*, pages 1484–1487, 2008.

[8] A. Del Re, A. Nannarelli, and M. Re. Implementation of digital filters in carry-save residue number system. In *Proc. Asilomar Conf. Sig. Syst. Comp.*, volume 2, pages 1309–1313, 2001.

[9] O. Gustafsson, A. Dempster, and L. Wanhammar. Multiplier blocks using carry-save adders. In *Proc. ISCAS*, volume 2, pages II473–II476, 2004.

[10] K. Hwang. *Computer Arithmetic: Principles, Architecture and Design*. John Wiley and Sons, New York, NY, USA, 1979.

[11] 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*, 57(3):198–202, Mar. 2010.

[12] A. Lindahl and L. Bengtsson. A low-power FIR filter using combined residue and radix-2 signed-digit representation. In *Proc. 8th Euromicro Conf. on Digital System Design (DSD'2005)*, pages 42–47, 2005.

[13] A. Nannarelli, M. Re, and G. Cardarilli. Tradeoffs between residue number system and traditional FIR filters. In *Proc. ISCAS*, volume 2, pages II305–II308, Sydney, NSW, Australia, 6–9 May 2001.

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

[15] R. Patel, M. Benaissa, N. Powell, and S. Boussakta. Novel power-delay-area-efficient approach to generic modular addition. *IEEE Trans. Circuits Syst. I*, 54(6):1279–1292, June 2007.

[16] A. Persson and L. Bengtsson. Forward and reverse converters and moduli set selection in signed-digit residue number systems. *J. Signal Process. Syst.*, 56(1):1–15, 2009.

[17] S. J. Piestrak. Design of residue generators and multioperand modular adders using carry-save adders. *IEEE Trans. Comput.*, 43(1):68–77, Jan. 1994.

[18] I. Shuli, M. Petricca, G. Cardarilli, A. Nannarelli, and M. Re. Multiple constant multiplication through residue number system. In *Proc. Asilomar Conf. Sig. Syst. Comp.*, pages 736–739, 2009.

[19] M. Soderstrand and K. Al-Marayati. VLSI implementation of very-high-order FIR filters. In *Proc. ISCAS*, pages 1436–1439, vol. 2, 28 April–3 May 1995.

[20] Y. Voronenko and M. B. Püschel. Multiplierless multiple constant multiplication. *ACM Trans. Alg.*, 3(2), May 2007, Article 11.

[21] A. Wrzyszcz, D. Milford, and E. Dagless. A new approach to fixed-coefficient inner product computation over finite rings. *IEEE Trans. Comp.*, 45(12):1345–1355, Dec. 1996.