Power Efficient Design of Parallel/Serial FIR Filters in RNS

Massimo Petricca, Pietro Albicocco, Gian Carlo Cardarilli, Alberto Nannarelli* and Marco Re
Department of Electrical Engineering, University of Rome "Tor Vergata", Rome, Italy
* Dept. of Informatics & Mathematical Modeling, Technical University, Denmark

***Abstract*—It is well known that the Residue Number System (RNS) provides an efficient implementation of parallel FIR filters especially when the filter order and the dynamic range are high. The two main drawbacks of RNS, need of converters and coding overhead, make a serialized implementation of the FIR filter potentially disadvantageous with respect to filters implemented in the conventional number systems. In this work, we show a number of solutions which demonstrate that the power efficiency of RNS FIR filters implemented serially is maintained in ASIC technology, while in modern FPGA technology RNS implementations are less efficient.**

I. Introduction

In [1], we presented a comparison in terms of delay, area and power dissipation of parallel Finite Impulse Response (FIR) filters implemented in the Residue Number System (RNS) and in the traditional Two's Complement System (TCS). The results of [1] show that for programmable FIR filters of high order the RNS implementation is more power efficient than the TCS one.

One drawback of RNS is the need for input/output converters necessary to interface the filter to the rest of the system (A/D converter, DSPs, etc.) normally in TCS.

Another drawback, is the coding overhead introduced by the representation of the RNS base [2], which affects the parts of the data-paths (e.g. registers and multiplexers) not performing arithmetic operations (e.g. addition and multiplication).

For these reasons, the advantages of RNS in the implementation of serial, or partly parallel, FIR filters are not evident, and the objective of this work is to investigate the impact of serialization on delay, area and power dissipation of programmable FIR filters.

As reference filter we consider a 128-tap FIR filter with high dynamic range (24-36 bit) and sampling frequency of 20 MHz. These types of filters are normally used to extract narrow band signals from a broader band signal. For example, some applications are:

- Reflected radar signal when signals of different amplitude are overlapped.
- Low power signals originating from sensors in a noisy (electromagnetic) environment.
- Receivers of signals from deep-space probes when the incoming signal (to Earth) is weak.

The results of the implementation of this reference filter in a partly serial architecture show that in ASIC (standard cells) technology the RNS implementation is more power efficient than the TCS one, especially if clock gating is used. On the other hand, for FPGA implementations, the TCS filters are more power efficient than RNS filters.

II. Background on RNS

A Residue Number System (RNS) is defined by a set of P relatively prime integers $\{m_1, m_2, \ldots, m_P\}$ which identify the RNS base. Its dynamic range is given by the product $M = m_1 \cdot m_2 \cdot \ldots \cdot m_P$.

Any integer $X \in \{0, 1, 2, \ldots M-1\}$ has a unique RNS representation given by:

$$X \overset{RNS}{\rightarrow} (\ \langle X \rangle_{m_1}, \langle X \rangle_{m_2}, \ldots, \langle X \rangle_{m_P}\ )$$

where $\langle X \rangle_{m_i}$ denotes the operation $X \ mod \ m_i$ [3]. Operations on different m_i (moduli) are done in parallel

$$Z = X \ op \ Y \quad \overset{RNS}{\rightarrow} \quad \begin{cases} Z_{m_1} = \langle X_{m_1} \ op \ Y_{m_1} \rangle_{m_1} \\ Z_{m_2} = \langle X_{m_2} \ op \ Y_{m_2} \rangle_{m_2} \\ \ldots \quad \ldots \quad \ldots \\ Z_{m_P} = \langle X_{m_P} \ op \ Y_{m_P} \rangle_{m_P} \end{cases} \tag{1}$$

As a consequence, operations on large wordlengths can be split into several modular operations executed in parallel and with reduced wordlength [3].

The main advantage in using several moduli of small dynamic range is that the carry-chains in addition are very short (faster hardware) and that table based methods can be used to simplify some operations, as explained for modular multiplication in Sec. II-A.

A. Modular Multiplication

One of the advantages of the use of moduli of small dynamic range which are prime numbers, is that we can transform modular multiplication into a modular addition by the isomorphism technique [4].

The isomorphic transformation is based on the concept of indices that are similar to logarithms, and primitive roots r which are similar to logarithm bases. If m is a prime number, there exists some primitive root for which the following property holds:

Every element in the field $F(m) = \{0, 1, \ldots, m-1\}$, excluding the zero, can be generated by using the equation

$$F(m) = \left\langle r^k \right\rangle_m \tag{2}$$

where k is an integer.

$F(11)$	k			
$\langle r^k \rangle_{11}$	$r=2$	$r=6$	$r=7$	$r=8$
1	0	0	0	0
2	1	9	3	7
3	8	2	4	6
4	2	8	6	4
5	4	6	2	8
6	9	1	7	3
7	7	3	1	9
8	3	7	9	1
9	6	4	8	2
10	5	5	5	5

TABLE I
THE ISOMORPHISM TABLE FOR $m = 11$.

As an example, the generation of the elements for $F(11)$ is shown in Table I. In this case, the four primitive roots are: $\{2,\ 6,\ 7,\ 8\}$.
By property (2), the product of two elements a and b belonging to the field is implemented by

1) Forward transformation of a and b in the corresponding indices.
2) Addition modulo $m-1$ of the two indices.
3) Reverse conversion of the result of the addition to obtain the final result of the modular product.

If the modulus is small (its binary representation is six or seven bits), the forward and reverse transformations can be implemented by look-up tables of reasonable size.
Therefore, the product of a and b modulo m is obtained as:

$$\langle a \cdot b \rangle_m = \langle r^k \rangle_m$$

where

$$k = \langle k_a + k_b \rangle_{m-1} \qquad \text{with } a = \langle r^{k_a} \rangle_m \text{ and } b = \langle r^{k_b} \rangle_m$$

Summarizing, to implement the modular multiplication the following operations are performed:

1) Two Direct Isomorphic Transformations (DIT) to obtain k_a and k_b;
2) One modulo $m-1$ addition $\langle k_a + k_b \rangle_{m-1}$;
3) One Inverse Isomorphic Transformation (IIT) to obtain the product.

The architecture of the isomorphic multiplier is shown in Fig. 1. The scheme is completed by a few gates to detect when one of the two operands is zero (no corresponding index in the isomorphism). On zero detected, the product is set to zero.
An alternative for isomorphic multiplication was proposed in [5]. Because isomorphic multipliers use modular adders in combination with tables, the modular adder in Fig. 1 is replaced by a binary adder and the modulus operation is incorporated in the IIT table[1] Moreover, addition of the constant $-m_i$ is incorporated in one of DIT tables (called DIT*), as well. The resulting scheme, shown in Fig. 2, is faster and consumes less power, as detailed in [5].

[1]Entries in the table are doubled (IIT + IIT*), but the delay in the adder is reduced.

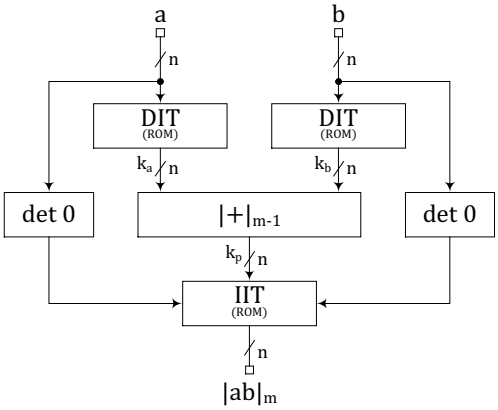


Fig. 1. Structure of basic isomorphic multiplier.

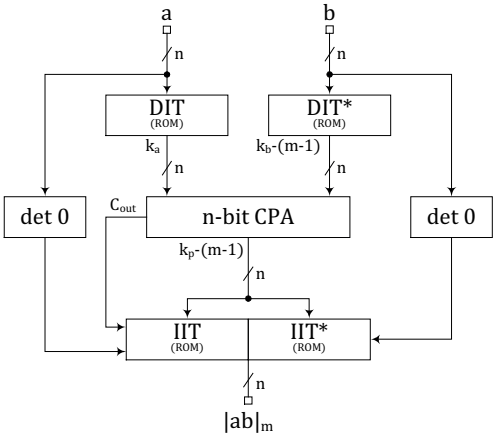


Fig. 2. Structure of modified isomorphic multiplier [5].

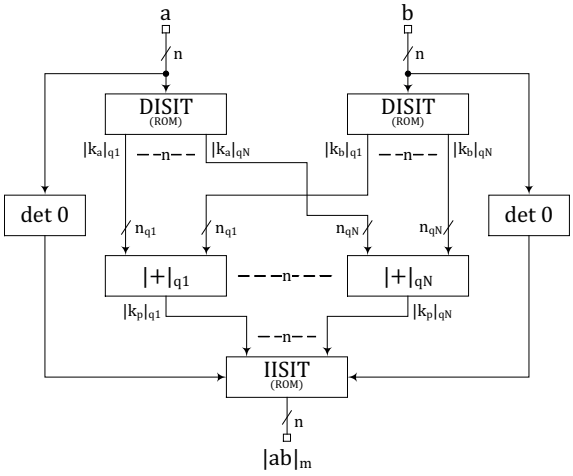


Fig. 3. Structure of isomorphic multiplier from [6].

A different alternative was proposed in [6]. Its architecture is sketched in Fig. 3. In this case, the isomorphic transformation is achieved by a RNS decomposition of the mod $(m-1)$ index computation ($m-1$ is not a prime number anymore for $m > 3$) in a subset of moduli q_i to obtain reduced delay.
For example, for $m = 11 \overset{ISO}{\rightarrow} m-1 = 10 = 2 \times 5$ the indices k_a and k_b can be decomposed as

$k_a \overset{RNS}{\rightarrow} \{\langle k_a \rangle_2, \langle k_a \rangle_5\}$ and $k_b \overset{RNS}{\rightarrow} \{\langle k_b \rangle_2, \langle k_b \rangle_5\}$.
The steps for modular multiplication based on the residues of the isomorphic indices are:

1) n Direct Isomorphic Submodular Indexes Transformations (DISIT) to obtain the two n-tuples $\{\langle k \rangle_{q_1}, \ldots, \langle k \rangle_{q_N}\}$ from the indexes k_a and k_b;
2) n modular additions of the n-tuples element by element;
3) One Inverse Isomorphic Submodular Indexes Transformation (IISIT) to obtain the modular product.

B. Coding Overhead

In [2], we introduced the *"coding overhead"* (OH) defined as the amount of extra bits required in the RNS representation of an integer compared to its TCS representation.
If the dynamic range of the TCS is $D = 2^d$, for the RNS representation the base must be chosen such that

$$\prod_{i=1}^{P} m_i = M \geq D = 2^d \ .$$

Each modulus m_i is encoded in binary for a total number of bits

$$b = \sum_{i=1}^{P} \lceil \log_2 m_i \rceil .$$

The coding overhead is defined as

$$OH = b - d \ .$$

The coding overhead has a significant impact on the parts of the datapath which do not perform arithmetic computations, such as registers and multiplexers.

C. RNS: Summary

Summarizing, the RNS is attractive because:

- It allows the parallelization of addition and multiplication as shown by (1).
- The shortened carry-chain length, due to the decomposition of (1), results in faster addition.
- If the moduli are prime numbers, the multiplication is simplified to an addition plus accesses to tables.

The main drawbacks of RNS are:

- The overhead of input/output conversions.
- Operations such as truncation, division and sign detection are hard to implement.
- The coding overhead can heavily limit the benefits of the RNS in some cases. One of those is described in Sec. IV-A.

III. Filter Architecture

The starting point of our design is a programmable 128-tap FIR filter working at a frequency of 20 MHz

$$y(n) = \sum_{k=0}^{127} a_k x(n-k) \qquad (3)$$

with a 36 bit dynamic range, 18-bit input samples x and 18-bit coefficients a_k.

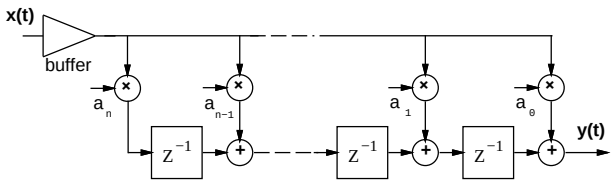


Fig. 4. Parallel implementation of (3) in transposed form.

We consider the following architectures:

1) A parallel implementation of (3) in transposed form, shown in Fig. 4. It requires 128 18x18 multipliers.
2) A serial/parallel implementation in which (3) is decomposed in

$$y(n) = \sum_{i=0}^{15} \left(\sum_{j=0}^{7} a_{8i+j} x(n - 8i + j) \right) \qquad (4)$$

and executed serially on a 16-tap filter (direct form) as depicted in Fig. 5. In this case, 16 18x18 multipliers are required.
Each tap (Fig. 5 bottom) implements the inner convolution of (4) by 8 multiply-add operations executed serially in one multiply-accumulate (MACC) unit.
Consequently, the serial/parallel filter of Fig. 5 is clocked at a frequency $f = 8 \times 20$ MHz $= 160$ MHz.

The two architectures are implemented in the conventional two's complement representation (TCS) and in the Residue Number System (RNS).
Because of (1), in RNS, a FIR filter (3) is decomposed into P filters working in parallel [7]. In this specific case, the 36-bit dynamic range can be parallelized in RNS by utilizing the RNS base of $P = 10$ co-prime moduli:

$$\text{RNS base } = \{3, 5, 7, 11, 13, 17, 19, 23, 31, 32\}$$

The RNS coding overhead is

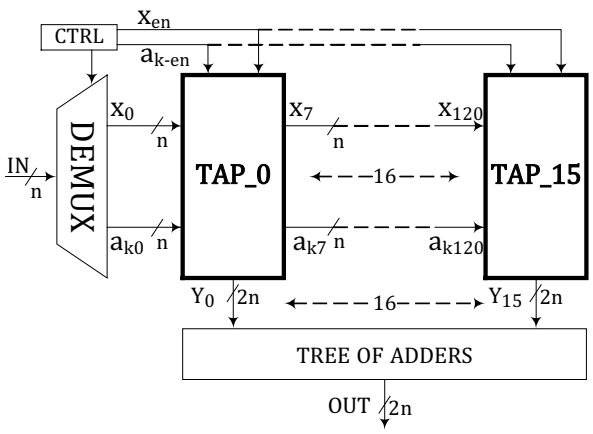
$$\text{OH} = b - d = \left(\sum_{i=1}^{P} \lceil \log_2 m_i \rceil \right) - d = 41 - 36 = 5$$

The coding overhead has a large impact on the RNS filter's area, as explained in Sec. IV-A.
The multiplication in RNS is implemented by the isomorphic transformation by selecting the suitable architecture, among the ones presented in Sec. II-A, depending on the specific modulus and the design constraints.

IV. ASIC (Standard Cells) Implementation

The two architectures (parallel and serial/parallel) are implemented in both TCS and RNS in a 90 nm library of standard cells. The units are synthesized by Synopsys Design Compiler. Table II reports the results for the four implementations. In the table, we introduce E_{pc} (average) energy-per-cycle defined as:

$$E_{pc} = \frac{P_{AVE}}{f_C} = P_{AVE} \cdot T_C$$



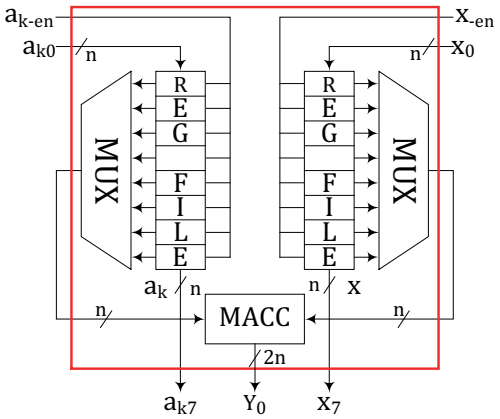


Fig. 5. Serial implementation of 128-tap filter by one 16-tap parallel filter. Top-level (top), serial implementation of the inner convolution of (4) in one TAP_i (bottom).

ASIC IMPLEMENTATION 128-TAP PARALLEL FILTER $f_C = 20$ MHz				
	t_{MAX} $[ns]$	Area $[mm^2]$	$P_{AVE}@f_C$ $[mW]$	E_{pc} $[pJ]$
TCS	6.63	1.15	23.9	1196
RNS	4.05	0.71	16.9	844
ratio	1.63	1.61	1.42	1.42

ASIC IMPLEMENTATION SERIAL/PARALLEL FILTER $f_C = 160$ MHz				
	t_{MAX} $[ns]$	Area $[mm^2]$	$P_{AVE}@f_C$ $[mW]$	E_{pc} $[pJ]$
TCS	5.90	0.28	73.8	461
RNS	3.95	0.40	70.6	441
ratio	1.50	0.69	1.04	1.04

TABLE II
IMPLEMENTATION OF FILTER ARCHITECTURES IN TCS AND RNS.

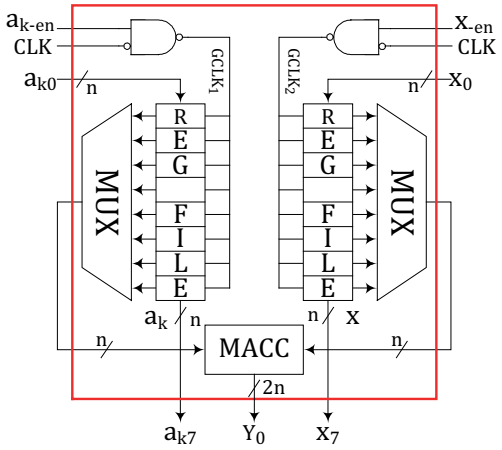


Fig. 6. Implementation of TAP_i with clock-gating.

to have a common metric for units clocked at different frequencies. The term t_{MAX} indicates the delay of the critical path in the filter.

Table II confirms that for high-order parallel filters the RNS implementation is advantageous in all metrics (t_{MAX}, area, and power dissipation) and that the overhead of conversions (input/output) does not offset the benefits obtained by RNS implementation of the operations (multiplication and addition). On the other hand, for the serial/parallel architecture, the TCS implementation is smaller (area) than the RNS one, and it consumes about the same power.

A. Clock Gating and Voltage Scaling

In TCS, the dynamic range of 36 bits at the output of the multipliers is obtained as the product of two 18-bit operands. On the other hand, in RNS, all the parts of the modular datapaths require a constant number of bits that adds up to 41 bits. The RNS coding overhead (OH) in the different parts of the datapath is:

- OH input samples: $\text{OH}_x = 41 - 18 = 23$
- OH coefficients : $\text{OH}_{a_k} = 41 - 18 = 23$
- OH output : $\text{OH}_y = 41 - 36 = 5$

Consequently, the register files of Fig. 5 (bottom) holding the filter coefficients and input samples have sizes:

TCS: $2 \times (8 \times 18 \text{ bit}) = 288$ bits
RNS: $2 \times (8 \times 41 \text{ bit}) = 656$ bits.

These extra flip-flops offset the power savings obtained in the RNS multiply-add units.

However, because the coefficients register file is only loaded when the filter mask is changed (initialization), and the samples $x(n)$ in the other register file are loaded every 8 cycles, by implementing clock-gating the power can be reduced. This simple modification, applied to both TCS and RNS serial/parallel filters, can be implemented as in Fig. 6.

The results of the implementation of the serial/parallel filter by clock-gating the register files of the serial filter are reported in Table III.

From the table, we see that by clock gating the serial filter, we obtain a power dissipation reduction of about 40% in the RNS filter with respect to the TCS filter (the reduction is about 30% for the parallel architecture).

Moreover, as the RNS serial/parallel filter has $t_{MAX} \ll T_C$, we could reduce the supply voltage V_{DD} until $t_{MAX} = T_C$. In our library, a delay increase of $\frac{6.25}{3.89} = 1.6$ can be sustained

ASIC IMPLEMENTATION SERIAL/PARALLEL FILTER $f_C = 160$ MHz (clock-gated)				
	t_{MAX} $[ns]$	Area $[mm^2]$	$P_{AVE}@f_C$ $[mW]$	E_{pc} $[pJ]$
TCS	5.90	0.27	58.3	364
RNS	3.89	0.36	36.0	225
ratio	1.51	0.72	1.61	1.61

TABLE III
IMPLEMENTATION OF CLOCK-GATED SERIAL/PARALLEL TCS AND RNS FILTERS.

FPGA Implementation SERIAL/PARALLEL FILTER $f_C = 160$ MHz				
	AREA			
	slices (%)	LUTs (%)	FFs (%)	DSPs
TCS	20	9	14	16
RNS	59	38	29	0
	POWER at $f_C = 160$ MHz			
TCS	2010 mW			
RNS	3144 mW (+36%)			

TABLE IV
RESULTS OF FPGA IMPLEMENTATION OF SERIAL/PARALLEL TCS AND RNS FILTERS.

if the supply voltage is reduced from $V_{DD} = 1.0\ V$ to $V_{SV} = 0.8\ V$.
The reduction in power dissipation for this Scaled Voltage (SV) implementation can be estimated[2] by

$$P_{AVE}(V_{SV}) = P_{AVE}(V_{DD}) \cdot \frac{V_{SV}^2}{V_{DD}^2} \simeq 23\ mW$$

corresponding to less than half the power dissipated by the TCS filter of Table III.

V. FPGA IMPLEMENTATION

We implemented the serial/parallel filter on a FPGA (Xilinx Virtex-5) equipped with 48 DSP blocks[3] to see if the results obtained for ASIC can be extended to FPGAs as well. The results of the evaluation are reported in Table IV.
The FPGA board is equipped with a power monitor and the power readings include all static, dynamic and FPGA configuration contributions.
As somewhat expected, the full-custom multiplier blocks used in the TCS implementation are more power efficient than the distributed logic used in the RNS filter.
Moreover, clock gating cannot be implemented in the family of FPGAs we used.
This result contrasts with previous experiments [8] done for parallel FIR filters, when FPGA chips did not have full-custom multiplication cores. Added functionalities on modern FPGAs make the RNS implementation of FIR filters in these platforms less attractive.

VI. CONCLUSIONS

In this paper we have explored the effect of different number systems (TCS and RNS) on the power dissipation of high order and large dynamic range FIR filters implemented on resource constrained (area) platforms for which a partially serial implementation of the filter is required.
The results show that for an ASIC (standard cells) platform, serial/parallel FIR filters implemented in RNS consumes significantly less power if clock gating is applied, although their area is larger than TCS filters. Moreover, as the slack $T_C - t_{MAX}$ is larger in RNS, if voltage scaling can be applied, the RNS filter can further reduce the power dissipation.
On the other hand, for modern FPGA platforms, filters implemented in TCS take a big advantage by the available full-custom multipliers that makes the RNS implementation less efficient.

[2]We assume the switching activity is constant.
[3]DSP blocks include a 25×18 multiplier

REFERENCES

[1] A. Nannarelli, M. Re, and G. C. Cardarilli, "Tradeoffs between Residue Number System and Traditional FIR Filters," *Proc. of IEEE International Symposium on Circuits and Systems*, vol. II, pp. 305–308, May 2001.
[2] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Impact of RNS Coding Overhead on FIR Filters Performance," *Proc. of 41st Asilomar Conference on Signals, Systems, and Computers*, pp. 1426–1429, Nov. 2007.
[3] N. Szabo and R. Tanaka, *Residue Arithmetic and its Applications in Computer Technology*. New York: McGraw-Hill, 1967.
[4] I. Vinogradov, *An Introduction to the Theory of Numbers*. New York: Pergamon Press, 1955.
[5] A. Nannarelli, G. C. Cardarilli, and M. Re, "Power-delay tradeoffs in Residue Number System," *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. V, pp. 413–416, May 2003.
[6] D. Radhakrishnan and Y. Yuan, "Novel Approaches to the Design of VLSI RNS Multipliers," *IEEE Transaction on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 39, no. 1, pp. 52–57, Jan. 1992.
[7] M. Sodestrand, W. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.
[8] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Power Characterization of Digital Filters Implemented on FPGA," *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. V, pp. 801–804, May 2002.