ρ-Direct Form Transposed and Residue Number Systems for Filter implementations

(Invited Paper)

Jean-Claude Bajard, Laurent-Stéphane Didier and Thibault Hilaire
LIP6, University Pierre et Marie Curie (UPMC), Paris, France

***Abstract*— This paper deals with a new approach for Infinite Impulse Response (IIR) Filter based on specific structure and arithmetic.**
The ρ-Direct Form II transposed, introduced by G. Li [1], is an numerically efficient structure for FIR or IIR filters. Compared to classical direct forms, it uses more computations but less bits are necessary for the same precision. These properties made it well conditioned for fixed point arithmetic and its implementations are economical and numerically efficient comparing to other forms. In other hand, Residue Number Systems (RNS) offer an interesting parallelism where operations are made on small values. RNS are well known for improving the performances of DSP filters. We compare our RNS approach to fixed-point implementations of DFI and ρ-DFIIt.

I. Introduction

The use of Residue Number Systems (RNS) [2] in digital signal processing has been widely studied [3]. The carry free and parallelism properties of this number system make it adapted to digital signal processing applications, where most of computations are additions and multiplications. Thus, many Finite Impulse Response filter designs (FIR) using RNS have been proposed [4], [5].

However, Infinite Impulse Response (IIR) filters are more representative of classical signal processing and are widely used in control applications. They are generally smaller but require to scale intermediate values during the processing. Equivalent forms of IIR filter having different precision requirements can be designed [1], [6].

In this paper a comparison of two forms of recursive filter and their implementation using fixed-point arithmetic and residue number system is presented. We describe in section II the two forms chosen for this comparison. The residue numbers system and details on the scaling operation are developed in section III. The implementation of an example is detailed in section IV.

II. ρ Direct Form II transposed

An IIR filter is defined by its transfer function

$$h : z \mapsto \frac{b_0 + b_1 z^{-1} + \ldots + b_{n-1} z^{-n+1} + b_n z^{-n}}{1 + a_1 z^{-1} + \ldots + a_{n-1} z^{-n+1} + a_n z^{-n}} \quad (1)$$

that describes its signal processing characteristics (specially the argument and the magnitude of $h\left(e^{j\omega}\right)$ with $\omega \in [0, 2\pi]$) and the input-output relationship, since $\mathcal{Y}(z) = h(z)\mathcal{U}(z)$, where $\mathcal{Y}$ and $\mathcal{U}$ are the $\mathcal{Z}$-transform of the output $y(k)$ and input $u(k)$, respectively. The z^{-1} operator describes the time-shift operator (delay).

Transposed in time-domain, the output of such a filter is computed linearly from the input by the following equation (called Direct Form I (DFI)) :

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i) \quad (2)$$

But some other algorithms (or realizations) can also be used. State-space filters and their cascade and parallel decompositions are often used.

It is very interesting to notice that the Finite Word-Length (FWL) effects are strongly dependant on the realization chosen. Some criteria have been developed to evaluate the round-off noise power [7] and how much the transfer function and the poles are modified by the quantization of the coefficients [8], [6]. The *optimal realization problem* consists then in finding the realization that minimizes these FWL effects.

Li and Hao [9], [10], [1] have presented a new sparse structure called rho-Direct Form II transposed (ρDFIIt). This is a generalization of the transposed direct-form II structure where the conventional time-shift operation ($x(k) \to x(k+1)$) is changed in new operators ρ. It is a sparse realization (with $3n + 1$ parameters where n is the order of the controller), leading so to an economic (few computations) implementation that could be very numerically efficient. As we will see later, this realization has n extra degrees of freedom that can be used to find an *optimal* realization.

Let us define

$$\rho_i : z \mapsto \frac{z - \gamma_i}{\Delta_i}, \text{ and } \varrho_i : z \mapsto \prod_{j=1}^{i} \rho_j(z), \quad 1 \leq i \leq n \quad (3)$$

where $(\gamma_i)_{1\leq i\leq n}$ and $(\Delta_i > 0)_{1\leq i\leq n}$ are two sets of constants.

The idea behind the ρDFIIt is to **reparametrized** the transfer function with $(\alpha_i)_{1\leq i\leq n}$ and $(\beta_i)_{0\leq i\leq n}$ as follows:

$$h(z) = \frac{\beta_0 + \beta_1 \varrho_1^{-1}(z) + \ldots + \beta_{n-1}\varrho_{n-1}^{-1}(z) + \beta_n \varrho_n^{-1}(z)}{1 + \alpha_1 \varrho_1^{-1}(z) + \ldots + \alpha_{n-1}\varrho_{n-1}^{-1}(z) + \alpha_n \varrho_n^{-1}(z)} \quad (4)$$

and to use a classical transposed Direct Form II (see Figure 1), where each operator ρ_i^{-1} is implemented as shown in Figure 2 (the $\left(\varrho_i^{-1}\right)$ are obtained by cascading the $\left(\rho_i^{-1}\right)$). The (α_i) and (β_i) can be directly computed from the (a_i), (b_i), (γ_i) and (Δ_i) (see [9] for details).

The $(\gamma_i)_{1\leq i\leq n}$ and $(\Delta_i)_{1\leq i\leq n}$ are parameters that can be freely chosen. They are used to minimize the L_2 transfer

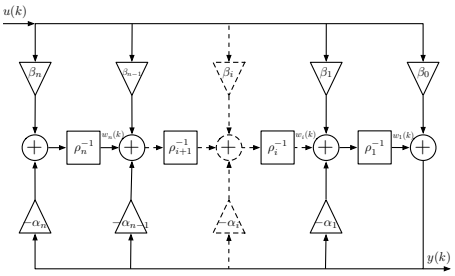

Fig. 1. Generalized ρ Direct Form II

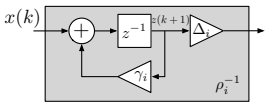

Fig. 2. Realization of operator ρ_i^{-1}

function sensitivity and the roundoff noise gain [9], [6] and greatly improve the overall numerical robustness.

III. Residue Number System Implementation

A. Introduction to RNS

A Residue Number System is defined by a base of co-prime numbers $\{m_0, ..., m_n\}$, and allows to represent integers lower than $M = \prod_{i=0}^{n} m_i$ with their residues modulo m_i. Thus, $X < M$ is represented by $(x_0, ..., x_n)$ with, $x_i = |X|_{m_i} = X \bmod m_i$.

In these systems, operations like additions or multiplications are modular operations independently performed on each modulus (also called channel), and the result is given modulo M. Thus, calculus are distributed on small values and computed in parallel. Summation of products, as in DSP filters, are welcome for this numeration system.

However, the magnitude and the parity of a RNS number can not be obtained easily. As a consequence, general division and scaling remain costly operations compared to additions and multiplications: a conversion, or a partial conversion into a radix system is generally necessary. The conversion from RNS to binary uses the Chinese Remainder Theorem reconstruction (close to Lagrange interpolation), or the Mixed Radix System (MRS) which looks like Newton interpolation. We depict here the MRS version used in this paper. In this algorithm, the digits ζ_i of a mixed radix number X_{MRS} are computed from the residues x_i of a RNS number X and where the modular inverse of m_i modulo m_j is noted $m_i^{-1} \bmod m_j$.

$$\begin{cases} \zeta_0 &= x_0 \\ \zeta_1 &= (x_1 - \zeta_0)\ m_0^{-1} \bmod m_1 \\ \zeta_2 &= \left((x_2 - \zeta_0)\ m_0^{-1} - \zeta_1\right)\ m_1^{-1} \bmod m_2 \\ &\vdots \\ \zeta_n &= \left(\cdots\left((x_n - \zeta_0)\ m_0^{-1} - \zeta_1\right)\ m_1^{-1} - \cdots - \zeta_{n-1}\right)\ m_{n-1}^{-1} \bmod m_n. \end{cases} \tag{5}$$

We just have to evaluate X in the wished radix system with the Horner's scheme :

$$X = \left(\cdots\left(\left(\zeta_n\ m_{n-1} + \zeta_{n-1}\right)\ m_{n-2} + \cdots + \zeta_2\right)\ m_1 + \zeta_1\right)\ m_0 + \zeta_0. \tag{6}$$

B. Scaling and fixed-point representation in RNS

In this paper, we deal with DSP filter based on fixed-point notation (Algorithms 2 and 1), using RNS arithmetic. The computations are performed using integer with scaling function for maintaining the fixed position of the dot. Additions and multiplications are directly performed in RNS. The impact of the scaling in RNS can be soften by choosing an RNS base simplifying this operation. Thus, we consider bases $\mathcal{B} = \{m_0, m_1, \ldots, m_n\}$ where elements are co-prime and of the form $m_0 = 2^k$ and $m_i = 2^k \pm c_i,\ i = 1..n$ [11]. The c_i are selected such that the mixed radix conversion is done only with few shift-and-add operations. We assume that the precision required for the accumulation register is lower than $M = \prod_{i=0}^{n} m_i$.

Scaling: Let consider the RNS number $X = (x_0, \cdots, x_n)$ and the integer scaling factor $T = 2^t$. We want to compute $Y = \left\lfloor \frac{X}{T} \right\rfloor$, which is equivalent to $Y = \frac{X - |X|_T}{T}$ As the exact division by an integer T is performed in RNS by multiplying by its modular inverse $|T|_M^{-1}$, we compute:

$$|Y|_M = \left|(X - |X|_T) \times |T|_M^{-1}\right|_M \tag{7}$$

The value of $|X|_T$ is obtained, for each channel of the residue system, directly from the residue modulo m_0 for $t < k$. Hence, if $T < m_i$ and $t < k$, we have:

$$|X|_T = \left||X|_{m_0}\right|_T = \left||X|_T\right|_{m_i} \tag{8}$$

However, as computing equation (7) is not possible for channel m_0, the modular inverse of 2^t modulo 2^k does not exist. Thus, the construction of $|Y|_{m_0}$requires a partial conversion using the MRS scheme.

Finally, the scaling operation $X \gg t$ for $t \leq k$, is performed as follow :

1) $r \leftarrow x_0 \bmod 2^t$
2) for $i = 1$ to n do $a_i \leftarrow (a_i - r) \times (2^{-t} \bmod m_i)$
3) MR conversion $(\zeta_0, \zeta_1, ..., \zeta_n) \leftarrow (x_0, x_1, ..., x_n)$
4) $x_0 \leftarrow \left(\left(\zeta_n m_{n-1} + ... + \zeta_2\right) m_1 + \zeta_1\right) m_0 + \zeta_0 \bmod 2^k$

IV. Example

We are considering a 6$^{\text{th}}$-order Butterworth filter, which transfer function is given by the matlab command `butter(6,0.125)` (see eq. 9).

$$h(z) = \frac{10^{-5}(2.883z^6 + 17.30z^5 + 43.24z^4 + 57.65z^3 + 43.24z^2 + 17.30z + 2.883)}{z^6 - 4.485z^5 + 8.529z^4 - 8.779z^3 + 5.148z^2 - 1.628z + 0.2166} \tag{9}$$

We also consider two different realizations, the Direct Form I (see eq. (2)), that directly uses the transfer function coefficients and the ρDFIIt, presented in section II, which parameters (γ_i) and (Δ_i) have been optimized to minimize the Finite Wordlength effects [6].

In order to be implemented, the coefficients should be replaced by their fixed-point approximations. The figure 3 exhibits the difference between the transfer function h and its fixed-point version $h^\dagger$ in function of the coefficients' wordlength. It is interesting to notice that the ρDFIIt form requires at least 5 bit-coefficients to approximate the ideal

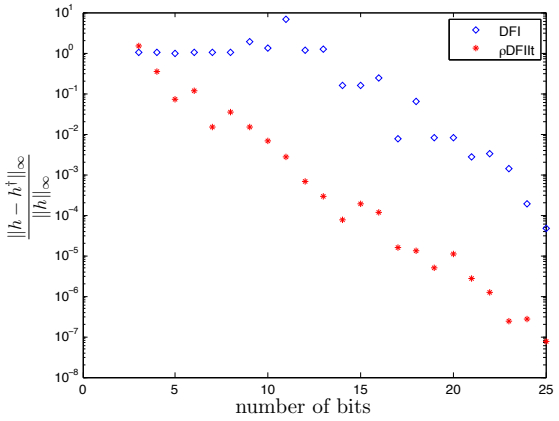

Fig. 3. Relative difference between the ideal transfer function and the fixed-point implemented transfer function

transfer function, whereas 15 bit-coefficients are required in Direct Form I with the same precision.

A. Implementation

In order to simplify both modular operations on each channel and the conversion, the residue base $\{m_0 = 2^k, m_1 = 2^k - 1, m_2 = 2^k + 1\}$ is used here. The following algorithm is used

1) $r \leftarrow x_0 \bmod m_0$
2) $y'_0 = (x_0 - r) \gg t$
   $y_1 = (x_1 - r)(2^t)^{-1} \bmod m_1$
   $y_2 = (x_2 - r)(2^t)^{-1} \bmod m_2$
3) $\zeta_1 = (y_1 - y'_0) \ll t \bmod m_1$
   $\zeta_2 = \big(((y'_0 - y_2) \ll t) - \zeta_1\big) \ll (k-1) \bmod m_2$
4) $y_0 = y'_0 + \big(\zeta_1 \ll (k-t)\big) - \big(\zeta_2 \ll (k-t)\big) \bmod m_0$

The two structures have both been implemented on FPGA, using fixed-point arithmetic and RNS. The designs have been described in VHDL, targeting the Xilinx Virtex4 FPGA familly. The designs have been synthetized, placed and routed using the Xilinx ISE design suite. For the RNS implementations, we used the modular adders described in [12].

Direct Form I

This realization is given by the fixed-point algorithm 2, with 18-bit coefficients, 18-bit variables and 36-bit additions. The constant multiplications are performed by shifts and additions [13], [14]. The same multiplication scheme is used in fixed-point and RNS implementations. RNS version uses the base $\{4096, 4095, 4097\}$, with $k = 12$.

ρDirect Form II transposed

This realization for fixed-point requires only 5-bit coefficients, 10-bit variables and 15-bit additions. The fixed-point algorithm is given by algorithm 1. The RNS version uses the base $\{32, 31, 33\}$, with $k = 5$. Since the size of the moduli is small, the multiplications are implemented with lookup-tables.

B. Results

The delay and area of the implemented designs are summarized in table I. Firstly, it appears that despite more additions and multiplications, both the RNS and fixed-point ρDFIIt outperform the DFI realization with the same arithmetic. This gain is due to the small size of the operands in the ρDFIIt algorithm. This small size allows to choose a small RNS base and thus to tabulate the constant multipliers, reducing the size of the RNS implementation by 13%.

Secondly, the RNS designs appear to be larger and slower than fixed-point designs. We observe that the FPGA technology gives little benefit to RNS operators. Indeed, in this technology, the fastest and smallest adders are ripple carry adders. More precisely, Xilinx architecture provides a fast carry propagation mechanism which makes this kind of adder efficient.

As a consequence, splitting adders into three smaller adders working in parallel gives no reduction on the overall size, making RNS adders larger than binary adders. Furthermore, the carry propagation logic is so fast, that the size of the input size has a reduced impact on the delay. For instance, the delay of a 5-bit adder is 6.6ns while it is 7.1ns for a 15-bit adder. Thus, the gain expected on the delay by reducing the operand size with RNS arithemtic is quite null.

Finally, the scaling operation in RNS has a cost similar to a RNS addition. However, it has an important impact on the size and the delay of the RNS designs.

		delay (ns)	area (slices)
DFI	fixed-point	20.61	1071
	RNS	54.76	3405
ρDFIIt	fixed-point	16.37	206
	RNS tables	17.03	1167

TABLE I
DELAY AND SIZE

C. Remark

It is also possible to consider the following four elements base [15] $\mathcal{B} := \{2^k - 1, 2^k + 1, 2^k - 2^{k-r} - 1, 2^k - 2^{k-r} + 1\}$ and use and extra redondant moduli $m_0 = 2^k$.

For example, when $k = 5$ and $r = 2$, we obtain $\mathcal{B} := (31, 33, 23, 25)$, with $M = 588225 > 2^{19}$ and $m_0 = 2^5$. The inverses of the moduli relatively to the others and the inverses of some power of two allow multiplications reduced to at most two additions (Table II). Note that $1..1 = 10..0(-1)$. This RNS base seems to be suitable for the Algorithm 2.

Then, for $k = 3$, we can consider the RNS base $\mathcal{B} = (5, 9, 7, 11)$ with $M = 3465 > 2^{11}$ and $m_0 = 2^3$. Here clearly look-up tables are welcome. With this kind of implementation we can expect an interesting gain, both in terms of time and area. This approach will be developed in detail in further works.

	m_1	m_2	m_3	m_4
$(m_1)^{-1}$		10000	11	10101
$(m_2)^{-1}$	10000		111	10110
$(m_3)^{-1}$	11011	10111		1100
$(m_4)^{-1}$	11011	100	1100	
$(2^1)^{-1}$	10000	10001	1100	1101
$(2^2)^{-1}$	1000	11001	110	10011
$(2^3)^{-1}$	100	11101	11	10110
$(2^4)^{-1}$	10	11111	1101	1011
$(2^5)^{-1}$	1	100000	10010	10010

TABLE II

BINARY REPRESENTATION OF THE INVERSES

V. DISCUSSION

The ρDirect Form II transposed which gives smaller and faster filter designs in fixed-point arithmetic, is also very interesting in RNS. Our first experimentations with non optimized operators, give some hopeful results concerning the delay (Tab. I). About the area some improvements are expected, such as an implementation with four moduli that should be more appropriate.

If we consider the RNS implementation of FIR [5], we note that the use of very small moduli seems to be a good strategy. We observe that FIR does not need scaling operations as for IIR. In RNS the impact of this operation is very important. It may be interesting to find some scaling methods for residue bases having more moduli, so that some part of the computations may be tabulated.

Actually, the FPGA technology seems not really appropriate for RNS implementations of IIR, compared to binary designs. We are confident that CMOS technology may give more interesting results.

REFERENCES

[1] G. Li, "A polynomial-operator-based dfiit structure for iir filters," *IEEE Trans. on Circuits and Systems-II*, vol. 51, no. 3, pp. 147–151, March 2004.

[2] N. S. Szabó and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*, ser. McGraw-Hill series in information processing and computers, 1967.

[3] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, Eds., *Residue number system arithmetic: modern applications in digital signal processing*. Piscataway, NJ, USA: IEEE Press, 1986.

[4] A. Del Re, A. Nannarelli, and M. Re, "Implementation of digital filters in carry-save residue number system," in *35th Asilomar Conference on Signals, Systems and Computers*. IEEE, 2001, pp. 1309–1313.

[5] G. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Impact of rns coding overhead on fir filters performance," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, Nov. 2007, pp. 1426 –1429.

[6] T. Hilaire and P. Chevrel, "Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context," *EURASIP Journal on Advances in Signal Processing*, vol. special issue on Quantization of VLSI Digital Signal Processing Systems, no. Article ID 893760, doi:10.1155/2011/893760 2011.

[7] C. Mullis and R. Roberts, "Synthesis of minimum roundoff noise fixed point digital filters," in *IEEE Transactions on Circuits and Systems*, vol. CAS-23, no. 9, September 1976.

[8] M. Gevers and G. Li, *Parametrizations in Control, Estimation and Filtering Probems*. Springer-Verlag, 1993.

[9] G. Li and Z. Zhao, "On the generalized DFIIt structure and its state-space realization in digital filter implementation," *IEEE Trans. on Circuits and Systems*, vol. 51, no. 4, pp. 769–778, April 2004.

[10] J. Hao and G. Li, "An efficient structure for finite precision implementation of digital systems," in *Information, Communications and Signal Processing, 2005 Fifth International Conference on*, 2005, pp. 564–568.

[11] J. Bajard, N. Meloni, and T. Plantard, "Efficient rns bases for cryptography," in *IMACS'05 : World Congress: Scientific Computation, Applied Mathematics and Simulation ,Paris (France) July 11-15*, 2005.

[12] J.-L. Beuchat, "Some modular adders and multipliers for field programmable gate arrays," in *Proceedings of the 17th International Parallel & Distributed Processing Symposium*. IEEE, 2003.

[13] V. Lefèvre, "Multiplication by an integer constant: Lower bounds on the code length," in *Proceedings of the 5th Conference on Real Numbers and Computers*, 2003.

[14] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.

[15] L.-S. Didier and P.-Y. Rivaille, "A generalization of a fast rns conversion for a new 4-modulus base," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 1, pp. 46 –50, Jan. 2009.

Input: $u(k)$: 10-bit integer
Output: $y(k)$: 10-bit integer
Data: $x(k)$: array [1..6] of 10-bit integers
Data: Acc: 15-bit integer
begin

```
// Intermediate variables
// States
Acc ← (x1(k) * −15) >> 3;
Acc ← Acc + x2(k) << 2;
Acc ← Acc + x1(k) * 14;
Acc ← Acc + ((u(k) * 10) >> 12);
x1(k + 1) ← Acc >> 4;
Acc ← (x1(k) * −10);
Acc ← Acc + (x3(k) << 3);
Acc ← Acc + (x2(k) * 13);
Acc ← Acc + ((u(k) * 11) >> 8);
x2(k + 1) ← Acc >> 4;
Acc ← (x1(k) * −15) >> 4;
Acc ← Acc + (x4(k) << 3);
Acc ← Acc + (x3(k) * 12);
Acc ← Acc + ((u(k) * 12) >> 6);
x3(k + 1) ← Acc >> 4;
Acc ← (x1(k) * −12) >> 3;
Acc ← Acc + (x5(k) << 2);
Acc ← Acc + (x4(k) * 12);
Acc ← Acc + (u(k) >> 1);
x4(k + 1) ← Acc >> 4;
Acc ← (x1(k) * −9) >> 4;
Acc ← Acc + (x6(k) << 1);
Acc ← Acc + (x5(k) * 11);
Acc ← Acc + ((u(k) * 11) >> 3);
x5(k + 1) ← Acc >> 4;
Acc ← (x1(k) * −15) >> 5;
Acc ← Acc + (x6(k) * 12);
Acc ← Acc + ((u(k) * 13) >> 2);
x6(k + 1) ← Acc >> 4;
// Outputs
y(k) ← x1(k);
```

end

Algorithm 1: ρDFIIt: Fixed-point algorithm

Input: $u(k)$: 18-bit integer input
Output: $y(k)$: 18-bit integer output
Data: $x(k)$: array [1..12] of 18-bit integers
Data: T: array [1..2] of 18-bit integers
Data: Acc: 36 bits integer
begin

```
// Intermediate variables
Acc ← (x7(k) * 123806) >> 5;
Acc ← Acc + ((x8(k) * 92855) >> 2);
Acc ← Acc + ((x9(k) * 116068) >> 1);
Acc ← Acc + (x10(k) * 77379);
Acc ← Acc + ((x11(k) * 116068) >> 1);
Acc ← Acc + ((x12(k) * 92855) >> 2);
Acc ← Acc + ((u(k) * 123806) >> 5);
T1 ← Acc >> 18;
Acc ← (x1(k) * −113552) >> 6;
Acc ← Acc + ((x2(k) * 106674) >> 3);
Acc ← Acc + ((x3(k) * −84339) >> 1);
Acc ← Acc + (x4(k) * 71918);
Acc ← Acc + (x5(k) * −69870);
Acc ← Acc + ((x6(k) * 73475) >> 1);
T2 ← Acc >> 13;
// States
x1(k + 1) ← x2(k);    x2(k + 1) ← x3(k);
x3(k + 1) ← x4(k);    x4(k + 1) ← x5(k);
x5(k + 1) ← x6(k);
Acc ← T1;
Acc ← Acc + (T2 << 10);
x6(k + 1) ← Acc >> 10;
x7(k + 1) ← x8(k);    x8(k + 1) ← x9(k);
x9(k + 1) ← x10(k);   x10(k + 1) ← x11(k);
x11(k + 1) ← x12(k);  x12(k + 1) ← u(k);
// Outputs
y(k) ← x6(k + 1);
```

end

Algorithm 2: Direct Form I: Fixed-point algorithm