\(\rho\)-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 \(\rho\)-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 DFII and \(\rho\)-DFII.

I. INTRODUCTION

The use of residue number systems (RNS) [4] in digital signal processing has widely been studied [2]. 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 [5], [3].

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], [8].

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. \(\rho\) DIRECT FORM II TRANSPOSED

An IIR filter is defined by its transfer function

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

that describes its signal processing characteristics (specially the argument and the magnitude of \(|h(e^{j\omega})|\) with \(\omega \in [0, 2\pi]\) and the input-output relationship, since \(Y(z) = h(z)U(z)\), where \(Y\) and \(U\) are the \(z\)-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 from the input by the following equation (called Direct Form I (DFI)):

\[
y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)
\] (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 dependent on the realization chosen. Some criteria have been developed to evaluate the round-off noise power [6] and how much the transfer function and the pole are modified by the quantization of the coefficients [7], [8]. The optimal realization problem consists then in finding the realization that minimize these FWL effects.

Li and Hao [9], [10], [1] have presented a new sparse structure called rho-Direct Form II transposed (\(\rho\)-DFII). This is a generalization of the transposed direct-form II structure where the conventional time-shift operation \((x(k) \rightarrow x(k+1))\) is changed in new operators \(\rho\). It is a sparse realization (with \(3n + 1\) parameters when \(n\) 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

\[
\rho_i : z \mapsto z - \frac{\gamma_i}{\Delta_i}, \quad \text{and} \quad \varrho_i : z \mapsto \prod_{j=1}^{i} \rho_j(z), \quad 1 \leq i \leq n
\] (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 \(\rho\)-DFII is to reparametrized the transfer function with \((\alpha_i)_{1 \leq i \leq n}\) and \((\beta_i)_{0 \leq i \leq n}\) as follows:

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

and to use a classical transposed Direct Form II (see Figure 1), where each operator \(\varrho_i^{-1}\) is implemented as shown in Figure 2 (the \(\varrho_i^{-1}\) are obtained by cascading the \(\rho_i^{-1}\)). The \((\alpha_i)\) and \((\beta_i)\) can be directly computed from the \((a_i), (b_i), (\gamma_i)\) and \((\Delta_i)\) (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
function sensitivity and the roundoff noise gain [9], [8] 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 \( \{m_1, ..., m_n\} \), and allows to represent integers lower than \( M = \prod_{i=1}^{n} m_i \) with their residues modulo \( m_i \). Thus, \( A < M \) is represented by \( (a_1, ..., a_n) \) with \( a_i = |A|_{m_i} = A \mod m_i \).

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, calculs 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\( \frac{\cdot}{\cdot} \) the MRS scheme.

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 soft by choosing an RNS base simplifying this operation. Thus, we consider bases (where elements are co-prime) of the form: \( B := (2^k + c_i, i = 1..n) \) [11] and an auxiliary element \( m_0 = 2^k \). We assume that the precision required for the accumulation register is lower than \( M = \prod_{i=1}^{n} m_i \). The \( c_i \) are selected such that the mixed radix conversion is done only with few shift and add operations.

**Scaling:** Let consider the RNS number \( X = (x_1, \ldots, x_n) \) and the integer scaling factor \( T = 2^t \). We want to compute \( Y = \lfloor \frac{X}{T} \rfloor \), which is equivalent to \( Y = X \mod |T|_M \)

As the exact division by an integer \( T \) is performed in RNS by multiplying by its modular inverse \( |T|^{-1}_M \), we compute:

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

The value of \( |X|_T \) is obtained, for each channel of the residue system, directly from the auxiliary 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 \( m_0 \), the modular inverse of \( 2^t \mod 2^k \) does not exist. Thus, the construction of \( |Y|_{m_i} \) requires a partial conversion using the MRS scheme.

Finally, the scaling operation \( A >> t \) for \( t \leq k \), is performed as follow:

1) \( r \leftarrow a_0 \mod 2^t \)
2) for \( i = 1 \) to \( n \) do \( a_i \leftarrow (a_i - r) \times (2^{-t} \mod m_i) \)
3) MR conversion \((\zeta_1, \zeta_2, \ldots, \zeta_n) \leftarrow (a_1, a_2, \ldots, a_n)\)
4) \( a_0 \leftarrow ((\zeta_n m_{n-1} + \ldots + \zeta_3) m_2 + \zeta_2) m_1 + \zeta_1 \mod 2^k \)

IV. EXAMPLE

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

\[
h(z) = \left( 10^{-5} \frac{2.8833}{z^6 + 17.305^2 + 43.24z^4 + 57.65z^2 + 43.24z^2 + 17.30z + 2.883} \right)
\]

\[
= \frac{10^{-5} \cdot 2.8833}{z^6 - 4.4885z^5 + 8.629z^4 - 8.779z^3 + 5.148z^2 - 1.628z + 0.2166} \tag{9}
\]

We also consider two different realizations, the Direct Form I, that directly uses the transfer function coefficients and the \( \rho \)DFII, presented in section II, which parameters \((\gamma_i) \) and \((\Delta_j)\) have been optimized to minimize the Finite Wordlength effects [8].

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 \( h^f \) in function of the coefficients’ wordlength. It is interesting to notice that the \( \rho \)DFII form requires at least 5 bit-coefficients to approximate the ideal
transfer function, whereas 15 bit-coefficients are required in Direct Form I (Fig. ??).

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. Thus for the reconstruction after the scaling using the MRS scheme is:

\[
x_0 = \left|x_2 + (x_1 - x_2) \times 2^{k-1} \times (2^k + 1)\right|_{2^k}
\]

(10)

Let note \( X_2 = (x_1 - x_2) \), and split the binary representation of this value in to sets \( X_{2,0} \) and \( X_{2,1} \) so that:

\[
X_2 = 2 \times X_{2,1} + X_{2,0}
\]

Equation (10) can be rewritten as follow:

\[
x_0 = \left|x_2 + X_{2,0} \times 2^{k-1} + X_{2,1}\right|_{2^k}
\]

(11)

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 family. The designs have been synthesized, 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 \( k = 12 \).

\( \rho \text{DFII} \) 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 \( k = 5 \). Because the size of the moduli is small, we implemented the multiplications 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 \( \rho \text{DFII} \) outperform the DFI realization with the same arithmetic. This gain is due to the small size of the operands in the \( \rho \text{DFII} \) 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 arithmetic 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.

<table>
<thead>
<tr>
<th></th>
<th>DFI</th>
<th>( \rho \text{DFII} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay (ns)</td>
<td>20.61</td>
<td>16.37</td>
</tr>
<tr>
<td>area (slices)</td>
<td>1071</td>
<td>206</td>
</tr>
</tbody>
</table>

TABLE I
DELAy AND SIZE

C. Remark

It is also possible to consider the following four elements base [15]:

\[
B := (2^k - 1, 2^k + 1, 2^k - 2^k - r - 1, 2^k - 2^k - r + 1) \text{ and } m_0 = 2^k
\]

(12)

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

Then, for \( k = 3 \), we can consider the RNS base \( B = (5, 9, 7, 11) \) 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.
TABLE II
BINARY REPRESENTATION OF THE INVERSES

<table>
<thead>
<tr>
<th>(m₁)⁻¹</th>
<th>m₁</th>
<th>m₂</th>
<th>m₃</th>
<th>m₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m₂⁻¹)⁻¹</td>
<td>10000</td>
<td>10001</td>
<td>10110</td>
<td></td>
</tr>
<tr>
<td>(m₃⁻¹)⁻¹</td>
<td>1101</td>
<td>1111</td>
<td>1111</td>
<td></td>
</tr>
<tr>
<td>(m₄⁻¹)⁻¹</td>
<td>1101</td>
<td>100</td>
<td>1100</td>
<td></td>
</tr>
</tbody>
</table>

V. DISCUSSION

The ρ-Direct Form II transposed which gives smaller and faster filter designs in fixed point arithmetic, is also very interesting in RNS. Our first experiments 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 [3], 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