RELIABLE IMPLEMENTATION OF LINEAR FILTERS WITH FIXED-POINT ARITHMETIC

Thibault Hilaire, Benoit Lopez

LIP6, Pierre and Marie Curie University (UPMC Univ Paris 06), Paris, France.

ABSTRACT

This article deals with the implementation of linear filters or controllers with fixed-point arithmetic. The finite precision of the computations and the roundoff errors induced may have an important impact on the numerical behavior of the implemented system. Moreover, the fixed-point transformation is a time consuming and error-prone task, specially with the objective of minimizing the quantization impact.
Based on a formalism able to describe every structure of linear filters/controllers, this paper proposes an automatic method to generate fixed-point version of the inputs-to-outputs algorithm and an analysis of the global error added on the output. An example illustrates the approach.

Index Terms— Fixed-point arithmetic, error analysis, filter implementation, code generation.

1. INTRODUCTION

The great majority of embedded signal processing or control algorithms is implemented using digital devices such as general purpose micro-controllers, DSP, ASIC or FPGA. For cost, size and power consumption reasons, the computation on these devices is mainly based on integer arithmetic (rather than floating-point arithmetic). The fixed-point arithmetic is used as an approximation of real numbers based on integers (mantissa) and implicit scaling (exponent). Unfortunately, the quantization of the embedded coefficients and the roundoff errors occurring in the finite precision computations lead to degradation in performance of the algorithm when implemented [1, 2, 3].

Unfortunately, there is no general tool to automatize and analyze the transformations required to obtain the final code from the filter or controller (as mathematical object). This final code may be executed on a device (software implementation) or describing hardware operations to be performed (hardware implementation). This problem is a difficult one, that can be decomposed in several steps [4] as illustrated in Figure 1. First, one should consider the various equivalent realizations to implement the filter. In addition to the classical direct forms or the state-space, some other interesting realizations can be considered, like the ones with the δ-operator (defined by $\delta \triangleq \frac{q-1}{\Delta}$, where q is the classical shift operator, and Δ a strictly positive constant [5]), the ρ-Direct Form II transposed (ρDFIIt) [6], the wave lattice filter, warped filter, and a lot of other specific realizations (LGC, LCW-structures [7], etc.).
They are all equivalent in infinite precision, but they are not anymore equivalent in finite precision, and their *robustness* to the finite precision implementation differs from one realization to another. Moreover, they do not imply the same number of coefficients. Criteria based on transfer function sensitivity, or pole sensitivity [1, 8] can be used as indicators of the Finite Word-Length (FWL) effects, in order to compare them and find the most suitable one. Then, for a given realization, the fixed-point conversion and the accurate error analysis must be done, according to the target. This fixed-point algorithm could then be processed to generate C code for software implementation, or used to generate dedicated hardware operators for FPGA or ASIC targets.

This work has been sponsored by french ANR agency under grant No ANR-11-INSE-008.

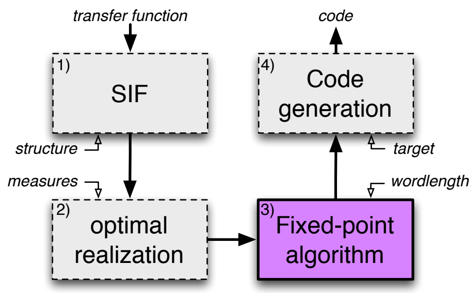

Fig. 1. From transfer function to code

This paper focuses on step 3 that transforms a realization into a fixed-point algorithm, and performs an error analysis (*i.e.* bounds the difference between the real output and the implemented one). A complete formalization of the sum-of-products in fixed-point arithmetic is proposed here.
For that purpose, the framework used to describe the realization (SIF) is reminded in section 2. Section 3 formalizes the fixed-point arithmetic and its propagation through an Abstract Syntax Tree with the use of interval arithmetic. Then the error analysis is applied to the SIF in section 4. Finally, an illustrative example is given in section 5, before the conclusion in section 6.

Notations: Throughout this paper, real numbers are in lowercase, column vectors in lowercase boldface and matrices in uppercase boldface. $[\underline{x}; \overline{x}]$ with square brackets corresponds to an inf-sup representation of an interval with $\underline{x}$ as the infimum and $\overline{x}$ as the supremum. $\langle x_m, x_r \rangle$ with angular brackets correspond to a mid-rad representation of an interval, with x_m as the middle point and x_r as the radius (obviously $x_m = \frac{\underline{x}+\overline{x}}{2}$ and $x_r = \frac{\overline{x}-\underline{x}}{2}$).

2. SPECIALIZED IMPLICIT FRAMEWORK

2.1. Formalism

In order to encompass all these possibilities, the Specialized Implicit Framework (SIF) has been proposed in [9]. It can be used as a unifying framework to describe all the filter realizations and allows the study and comparison of the FWL effects. It is an extension of the state-space framework, modified in order to allow a detailed (but still macroscopic) description of the inline computations to be performed. All the linear realizations, with delays, multiplications by

constants and additions can be represented. Multiple Input Multiple Output filters or controllers are also considered.
So various classical realizations, like q (shift) or δ-state-spaces, classical Direct Forms I and II, cascade or parallel decompositions, mixed structures, etc.. may be then described in a single unifying form (see [9] for details and examples).

Definition 1 (Specialized Implicit Framework) *Equation* (1) *recalls the specialized implicit form that explicitly expresses the parametrization and the intermediate variables used:*

$$\begin{pmatrix} \boldsymbol{J} & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{K} & \boldsymbol{I}_n & \mathbf{0} \\ -\boldsymbol{L} & \mathbf{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \boldsymbol{M} & \boldsymbol{N} \\ \mathbf{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \mathbf{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \qquad (1)$$

where $\boldsymbol{u}(k)$ represents the m inputs, $\boldsymbol{y}(k)$ the p outputs, $\boldsymbol{x}(k)$ is the n stored states (one state per delay) and $\boldsymbol{t}(k+1)$ is the l intermediate variables in the calculations of step k (the column of $\mathbf{0}$ in the second matrix shows that $\boldsymbol{t}(k)$ is not used for the calculation at step k: that characterizes the concept of intermediate variables).

It is implicitly considered throughout the paper that the computations associated to (1) are done according to the Algorithm 1.

```
begin
    // Intermediate variables
    $\boldsymbol{J}.\boldsymbol{t}(k+1) \leftarrow \boldsymbol{M}.\boldsymbol{x}(k) + \boldsymbol{N}.\boldsymbol{u}(k)$
    // state-vector update
    $\boldsymbol{x}(k+1) \leftarrow \boldsymbol{K}.\boldsymbol{t}(k+1) + \boldsymbol{P}.\boldsymbol{x}(k) + \boldsymbol{Q}.\boldsymbol{u}(k)$
    // outputs computation
    $\boldsymbol{y}(k) \leftarrow \boldsymbol{L}.\boldsymbol{t}(k+1) + \boldsymbol{R}.\boldsymbol{x}(k) + \boldsymbol{S}.\boldsymbol{u}(k)$
end
```

Algorithm 1: General algorithm associated to the SIF

Note that $\boldsymbol{J}$ is lower triangular with 1's on its diagonal, so the first value of $\boldsymbol{t}(k+1)$ is first calculated, and then its second value is computed using its first and so on (the computation of $\boldsymbol{J}^{-1}$ is then not necessary).

The coefficients of the realization can be written in a compact form

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix}. \qquad (2)$$

The equivalent transfer function is given by

$$\boldsymbol{H} : z \mapsto \boldsymbol{C_Z}(z\boldsymbol{I}_n - \boldsymbol{A_Z})^{-1}\boldsymbol{B_Z} + \boldsymbol{D_Z}, \quad \forall z \in \mathbb{C} \qquad (3)$$

with

$$\begin{array}{rclrcl} \boldsymbol{A_Z} & = & \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{P}, & \boldsymbol{B_Z} & = & \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{N} + \boldsymbol{Q}, \\ \boldsymbol{C_Z} & = & \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{R}, & \boldsymbol{D_Z} & = & \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{N} + \boldsymbol{S}. \end{array} \qquad (4)$$

2.2. Example

Li and Hao [6, 10] have presented a new sparse structure called ρDFIIt. This is a generalization of the transposed direct-form II structure with the conventional shift operator.It is a sparse realization (with $3n+1$ parameters when n is the order of the filter), leading so to an economic (few computations) implementation that is provcould be very numerically efficient with sensitive filters. Let us define

$$\rho_i : z \mapsto \frac{z - \boldsymbol{\gamma}_i}{\boldsymbol{\Delta}_i}, \text{ and } \varrho_i : z \mapsto \prod_{j=1}^{i} \rho_j(z), \quad 1 \leqslant i \leqslant n. \qquad (5)$$

It can be shown that any transfer function $H(z)$ can be **reparametrized** with $\{\boldsymbol{\alpha}_i\}$ and $\{\boldsymbol{\beta}_i\}$ as follows:

$$H(z) = \frac{\boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \varrho_1^{-1}(z) + \ldots + \boldsymbol{\beta}_n \varrho_n^{-1}(z)}{1 + \boldsymbol{\alpha}_1 \varrho_1^{-1}(z) + \ldots + \boldsymbol{\alpha}_n \varrho_n^{-1}(z)}, \quad \forall z \in \mathbb{C}. \qquad (6)$$

Equation (6) can be, for example, implemented with a transposed direct form II (see Figure 2) with ρ_i^{-1} operators, that is denoted the ρ-Direct Form II transposed (ρDFIIt).

Within the SIF Framework, the ρDFIIt form is described by [8]:

$$\boldsymbol{Z} = \left(\begin{array}{cccc:cccc:c} -1 & & & & \Delta_1 & & & & \beta_0 \\ & \ddots & & & & \Delta_2 & & & 0 \\ & & \ddots & & & & \ddots & & \vdots \\ & & & -1 & & & & \Delta_n & 0 \\ \hdashline -\alpha_1 & 1 & & & \gamma_1 & & & & \beta_1 \\ -\alpha_2 & 0 & \ddots & & & \gamma_2 & & & \beta_2 \\ \vdots & & \ddots & 1 & & & \ddots & & \vdots \\ -\alpha_n & & & 0 & & & & \gamma_n & \beta_n \\ \hdashline 1 & 0 & \ldots & 0 & 0 & \ldots & \ldots & 0 & 0 \end{array}\right) \qquad (7)$$

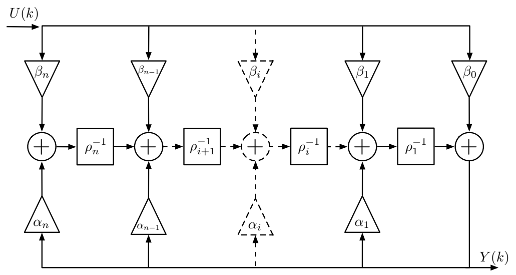

Fig. 2. ρ Direct Form II transposed

3. FIXED-POINT ARITHMETIC

3.1. Formalism and notations

In this article, the signed fixed-point arithmetic is used, with 2's complement representation. Let x be such a fixed-point number with w bits. It can be written as

$$x = -2^m x_m + \sum_{i=\ell}^{m-1} 2^i x_i \qquad (8)$$

where ℓ and m are the position of the *least* significant and *most* significant bits, respectively, and $x_i \in \mathbb{B} = \{0,1\}$ is the i^{th} bit of x as shown in Figure 3. They satisfy $m > \ell$ and $w = m - \ell + 1$ where w is the word-length.
Usually $m > 0$ and $\ell < 0$, but this is not mandatory. 2^ℓ is the quantization step of the representation.
x is represented in machine by the integer X, such that $X = x.2^{-\ell}$. X belongs to $[-2^{w-1}; 2^{w-1} - 1] \cap \mathbb{Z}$.

In the article, the notation (m, ℓ) is used to denote the Fixed-Point Format (FPF) of such a fixed-point number.

Remark 1 It is also common to decompose a fixed-point representation in its integer part (bits x_m to x_0) and fractional part (bits x_{-1}

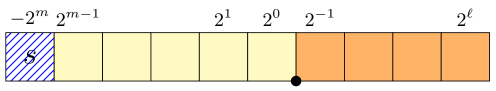

Fig. 3. Fixed-point representation.

to x_ℓ). If these number of bits are denoted i and f respectively, then the following holds: $f = -\ell$ and $i = m + 1$. Although the two notations i, f and (m, ℓ) are equivalent, the latter is more convenient, specially when dealing with negative "*numbers*" of bits for the integer or fractional parts.

3.2. Constant in fixed-point arithmetic

Let c be a real constant. It cannot necessary be exactly represented with finite word-length w, and will be approximated by the closest fixed-point number $\tilde{c}$. The following proposition gives the conversion algorithm.

Proposition 1 (Fixed-point conversion) *Let $c \in \mathbb{R}^*$ a number to be represented in signed fixed-point arithmetic with w bits as word-length. The following steps are used for the conversion and the determination of the FPF (m, ℓ):*

Step 1: Compute $m = \begin{cases} \lfloor \log_2 |c| \rfloor + 1 & \text{if } c > 0 \\ \lceil \log_2 |c| \rceil & \text{if } c < 0 \end{cases}$

Step 2: Compute the integer $C = \lfloor c.2^{w-m-1} \rceil$

Step 3: Check for particular cases:

```
if $C == 2^{w-1}$ then
    $m = m + 1$
    $C = \lfloor c.2^{w-m-1} \rceil$
end
if $C == -2^{w-2}$ then
    $m = m - 1$
    $C = \lfloor c.2^{w-m-1} \rceil$
end
```

Step 4: Deduce $\ell = m + 1 - w$ *and* $\tilde{c} = C2^\ell$.

The operators $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ *and* $\lfloor \cdot \rceil$ *are the* round towards minus infinity, round towards plus infinity *and* round to the nearest integer *operators, respectively.*

Proof: Step 1 finds the appropriate most significant bit m so as to represent c without overflow. m is such that $c \in [-2^m; 2^m - 1]$ and $c \notin [-2^{m-1}; 2^{m-1} - 1]$. Equation on step 1 is based on the dissymmetry of the 2's complement representation around 0.

Step 2 computes the associated integer, rounded to the nearest. And finally, step 3 considers some very special cases, where the rounded-to-the-nearest make C goes out of the range. This is the case, for example, for $c = 127.9$ and $w = 8$: in step 1, m is found to be equal to 7, and c is quantized to $\tilde{c} = 128$ ($C = 128 \cdot 2^0$), that is out of the range $[-128, 127] \cap \mathbb{Z}$ allowed for the FPF $(7, 0)$. The same problem occurs with $c = -128.1$ that is rounded to -128 but is lower than -128 (m is first set to 8, and c is approximated by $\tilde{c} = -128$ and $C = -64 \cdot 2^1$, whereas the FPF $(7, 0)$ is sufficient to represent -128). ■

Proposition 2 *If a variable x is known to be in the interval $[\underline{x}; \overline{x}]$, then the appropriate FPF (m, ℓ) to represent it with w bits is given by:*

$$m = \max(\underline{m}, \overline{m}), \quad \ell = m + 1 - w \tag{9}$$

where $\underline{m}$ and $\overline{m}$ are the most significant bits of $\underline{x}$ and $\overline{x}$, respectively, obtained from proposition 1.

Proof: m is chosen to represent numbers in $[\underline{x}; \overline{x}]$, so m should be high enough to represent $\underline{x}$ **and** $\overline{x}$. ■

3.3. Sum-of-Products and fixed-point operations

As seen in section 2, Algorithm 1 gives the expression to compute the ouput(s) $\boldsymbol{y}(k)$ at each step k from the input(s) $\boldsymbol{u}(k)$, the intermediate variables $\boldsymbol{t}(k)$ and the states $\boldsymbol{x}(k)$. These matrix operations can be decomposed in several Sum-of-Products (SoP), with all the null terms removed.

Let us consider the following Sum-of-Products to be evaluated in fixed-point arithmetic:

$$s = \sum_{k=1}^{n} \boldsymbol{c}_i \cdot \boldsymbol{x}_i \tag{10}$$

where $\{\boldsymbol{c}_i\}$ are some given non-null constants and $\{\boldsymbol{x}_i\}$ some variables, only known to be in the intervals $[\underline{\boldsymbol{x}}_i; \overline{\boldsymbol{x}}_i]$.

To transform this expression in a fixed-point version, an Abstract Syntax Tree (AST) is used to describe the operations to be done and their order. The tree consists of internal nodes that are the operations to perform (only additions and multiplications here) and the leaves are some constants or variables. See Figure 4 for an AST representing $12.2x + (y + 3.11z)$.

In order to provide a fixed-point version of this AST (with same operations performing on integers, and some shifts used to align the binary-point position), it is first required to determine the FPF of each node in the AST. For that purpose, some propagation rules are recursively applied in a bottom-to-up way. Unlike what was proposed in [11], where the FPF of a result of an operation was derived from the FPF of the two operands (so as to avoid overflow), a more precise method is proposed here, based on interval analysis arithmetic so as to avoid the possible over-estimation of the FPF.

An interval is associated to each variable and indicates in which range the variable belongs to. Then, by applying interval arithmetic rules [12], the intervals are propagated (from bottom-to-top) through the AST, in order to determine the interval of each result and their FPF (deduced from proposition 1).

The addition and multiplication operations are now considered: $z \leftarrow x \diamond y$, with $\diamond \in \{+, \times\}$, x with format (m_x, ℓ_x) and values in $[\underline{x}; \overline{x}]$ and $y \in [\underline{y}; \overline{y}]$ with $(m_y; \ell_y)$ as FPF.

Multiplcation $\boldsymbol{z} \leftarrow \boldsymbol{x} \times \boldsymbol{y}$: z will be in the interval $[\underline{z}; \overline{z}]$ with

$$\underline{z} = \min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), \text{ and } \overline{z} = \max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}). \tag{11}$$

Its FPF (m_z, ℓ_z) is given by $m_z = m_x + m_y + 1$ and $\ell_z = \ell_x + \ell_y$. Note that if the multiplier hardware operator is a $w_\circledast$-bit multiplier ($w_\circledast$-bit operands and $2w_\circledast$ bit result) with $w_x > w_\circledast$ (or $w_y > w_\circledast$), then x (or y) should be first right shifted of $w_x - w_\circledast$ bits ($w_y - w_\circledast$ bits respectively) before performing the operation.

Addition $\boldsymbol{z} \leftarrow \boldsymbol{x} + \boldsymbol{y}$: the addition is performed with a $w_\oplus$-bit adder (*i.e.* the operands are $w_\oplus$ bits and the result is on $w_\oplus$ bits plus a carry bit that is not used here since there will be no overflow *by construction*), as follows:

1) the interval of the result is given by $\underline{z} = \underline{x} + \underline{y}$, $\overline{z} = \overline{x} + \overline{y}$
2) its FPF (m_z, ℓ_z) with $w_\oplus$ bits is deduced from proposition 2 (so the addition is performed on format (m_z, ℓ_z)) ;
3) before the addition, x must be right shifted of $\ell_z - \ell_x$ bits (or left shifted of $\ell_x - \ell_z$ if $\ell_z < \ell_x$) and y of $\ell_z - \ell_y$ bits.

Figure 5 summarizes the fixed-point addition.

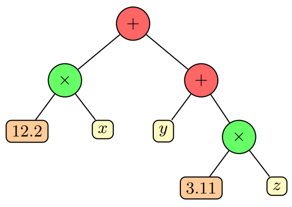

Fig. 4. An AST representing $12.2x + (y + 3.11z)$

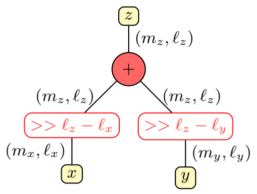

Fig. 5. Fixed-point adder

Remark 2 Moreover, if the FPF of the final result s is known by an other mean (like the one presented in Proposition 4), then the most significant bit of the partial result can be limited to the most significant bit of the final result. Even if an intermediate overflow occurs, it will be compensated in a next addition, thanks to Jackson's Rule [13].

These two rules may be propagated from bottom-to-up in the AST, in order to obtain a fixed-point version of the initial AST.

3.4. Error analysis

In fixed-point arithmetic, any right shift produces an error, called *roundoff* error.

Proposition 3 (Error model) *Let $x(k)$ be a signal with (m, ℓ) as FPF. Right shifting $x(k)$ of d bits is equivalent to add an interval error $[e] = [\underline{e}; \overline{e}]$ with:*

	Truncation	*Round to the nearest*
$[\underline{e}, \overline{e}]$	$[-2^{\ell+d} + 2^{\ell}; 0]$	$[-2^{\ell+d-1} + 2^{\ell}; 2^{\ell+d-1}]$

(12)

Proof: The interval error contains all the possible values formed with bits $x_{\ell+d-1}$ to x_ℓ. ■

So, when FPF have been propagated with preceding rules through the whole SoP, all shift values are known, so as the interval errors added for each right shift. Indeed, all these interval errors can be regrouped in one unique error, with interval addition rule [12]. The error is propagated from bottom-to-top through the SoP, as well as the FPF are.

Finally, a Sum-of-Products implemented in fixed-point arithmetic is equivalent to the original infinite precision sum-of-products, corrupted by the addition of a roundoff error, that is the sum of all the roundoff errors occurring during the fixed-point evaluation.

Remark 3 The fixed-point format and roundoff error depends on the order the operations (additions here) are performed. There are $\prod_{i=1}^{n-1}(2i-1)$ possible ordering for sum of n terms. In [11], an heuristic has been built so as to deal with all these possible ordered SoP. In the following, the considered SoP are the *optimal* ordered SoP, according to the roundoff error and degree of parallelism.

4. APPLICATION TO THE SIF

This work on Sum-of-Products is now applied to linear filters/controllers with input-output algorithms expressed by the SIF.
In order to determine the FPF of the input(s), output(s), states and intermediate variables, the following definition and lemma are required:

Definition 2 (DC-gain and ℓ^∞-norm) *Let H be a Simple-Input-Simple-Output Bounded-Input-Bounded-Output filter, characterized by its transfer function $H(z)$. Denote $h(k)$ its impulse response, and $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$ a state-space realization of H.*
Its DC-gain, denoted $|H|_{DC}$, is the low-frequency gain of the system, i.e. $\lim_{\omega\to 0} |H(e^{j\omega})|$. *It can be computed by*

$$|H|_{DC} = \boldsymbol{c}(\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d \quad or \quad |H|_{DC} = h(0) \tag{13}$$

Its worst-case peak gain, denoted $\|H\|_{\ell^\infty}$ is the largest possible peak value of the output $y(k)$ over all possible inputs $u(k)$:

$$\|H\|_{\ell^\infty} \triangleq \frac{\sup_{k\geqslant 0} |y(k)|}{\sup_{k\geqslant 0} |u(k)|}. \tag{14}$$

It can be computed with the ℓ_1-norm of its impulse response: $\|H\|_{\ell^\infty} = \sum_{k\geqslant 0} |h(k)|$. It can also be computed from its state-space realization: $\|H\|_{\ell^\infty} = \sum_{k\geqslant 0} \left|\boldsymbol{c}\boldsymbol{A}^k\boldsymbol{b}\right| + |d|$.
This worst-case peak gain can be achieved with $u(k) = U.sign(h(k))$, where U is a constant and sign the sign function returning ± 1 or 0.

Remark 4 The infinite sum required to compute the ℓ^∞-norm can be bounded by two convergent finite sums, so that pratically $\|\cdot\|_{\ell^\infty}$ can be evaluated at any required precision [14].

Lemma 1 (Interval through a linear filter) *Let $e(k)$ be a scalar input going through a filter H, and denote $f(k)$ the resulting output.*
If $\forall k, e(k) \in \langle e_m, e_r\rangle$, then $\forall k, f(k) \in \langle f_m, f_r\rangle$ with:

$$f_m = |H|_{DC}\, e_m \quad and \quad f_r = \|H\|_{\ell^\infty} e_r. \tag{15}$$

Proof: Since H is linear, then the output $f(k)$ is the sum of the constant term e_m through the filter and a variable term in $[-e_r; e_r]$ through the filter. The constant term is scaled by the DC-gain of the filter whereas the bound on the second term is scaled by the worst-case gain using (14). ■

By extension, if $\boldsymbol{H}$ is a Multiple Input Multiple Output filter (p inputs, q outputs, $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ as state-space representation), then $\langle\!\langle \boldsymbol{H} \rangle\!\rangle$ denotes the worst-case peak gain matrix, *i.e.* each component of these matrix is the worst-case peak gain of the sub-system associated. $\langle\!\langle \boldsymbol{H} \rangle\!\rangle \in \mathbb{R}^{p\times q}$ such as $\langle\!\langle \boldsymbol{H} \rangle\!\rangle_{i,j} \triangleq \|\boldsymbol{H}_{i,j}\|_{\ell^\infty}$. So $\langle\!\langle \boldsymbol{H} \rangle\!\rangle = \sum_{k\geqslant 0} \left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right| + |\boldsymbol{D}|$ with component-wise absolute value (*i.e.* $|\boldsymbol{A}|_{i,j} \triangleq |\boldsymbol{A}_{i,j}|$).
Then, if the input vector $\boldsymbol{e}(k)$ is component-wisely in $\langle \boldsymbol{e}_m, \boldsymbol{e}_r\rangle$, then $\boldsymbol{f}(k)$ is component-wisely in $\langle \boldsymbol{f}_m, \boldsymbol{f}_r\rangle$ with

$$\boldsymbol{f}_m = |\boldsymbol{H}|_{DC}\, \boldsymbol{e}_m \quad \text{and} \quad \boldsymbol{f}_r = \langle\!\langle \boldsymbol{H} \rangle\!\rangle\, \boldsymbol{e}_r. \tag{16}$$

Proposition 4 *Let the input $\boldsymbol{u}(k)$ of the system be (component-wisely) be in $\langle \boldsymbol{u}_m, \boldsymbol{u}_r\rangle$, then the output $\boldsymbol{y}(k)$, states $\boldsymbol{x}(k)$ and the intermediate variables $\boldsymbol{t}(k)$ are (component-wisely) in $\langle \boldsymbol{y}_m, \boldsymbol{y}_r\rangle$, $\langle \boldsymbol{x}_m, \boldsymbol{x}_r\rangle$ and $\langle \boldsymbol{t}_m, \boldsymbol{t}_r\rangle$ respectively, with*

$$\begin{pmatrix} \boldsymbol{t}_m \\ \boldsymbol{x}_m \\ \boldsymbol{y}_m \end{pmatrix} = |\boldsymbol{H}_u|_{DC}\, \boldsymbol{u}_m, \quad \begin{pmatrix} \boldsymbol{t}_r \\ \boldsymbol{x}_r \\ \boldsymbol{y}_r \end{pmatrix} = \langle\!\langle \boldsymbol{H}_u \rangle\!\rangle\, \boldsymbol{u}_r \tag{17}$$

and

$$\boldsymbol{H}_u : z \to \boldsymbol{N}_1 \left(z\boldsymbol{I}_n - \boldsymbol{A_Z}\right)^{-1} \boldsymbol{B_Z} + \boldsymbol{N_2}, \tag{18}$$

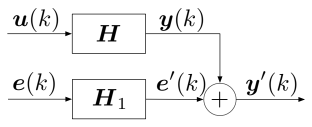

Fig. 6. Equivalent system, with errors separated

$$N_1 \triangleq \begin{pmatrix} J^{-1}M \\ A_Z \\ C_Z \end{pmatrix}, N_2 \triangleq \begin{pmatrix} J^{-1}N \\ B_Z \\ D_Z \end{pmatrix}. \tag{19}$$

Then, proposition 2 can be applied to deduce the FPF of each output, state and intermediate variable.

Proof: Eq (1) can be written as following:

$$\begin{pmatrix} t(k+1) \\ x(k+1) \\ y(k) \end{pmatrix} = \begin{pmatrix} J^{-1}M & J^{-1}N \\ A_Z & B_Z \\ C_Z & D_Z \end{pmatrix} \begin{pmatrix} x(k) \\ u(k) \end{pmatrix} \tag{20}$$

so H_u defined in (18) is the transfer function from $u(k)$ to $\begin{pmatrix} t(k+1) \\ x(k+1) \\ y(k) \end{pmatrix}$. Then lemma 1 is applied to get (17). ■

As seen in section 3.3, the evaluation for each SoP in fixed-point arithmetic may provide an additional error. When implemented, the algorithm 1 is changed in

$$\begin{aligned} J.t(k+1) &\leftarrow M.x(k) + N.u(k) & +e_t(k) \\ x(k+1) &\leftarrow K.t(k+1) + P.x(k) + Q.u(k) & +e_x(k) \\ y(k) &\leftarrow L.t(k+1) + R.x(k) + S.u(k) & +e_y(k) \end{aligned} \tag{21}$$

Denote $e(k)$ the vector with all the added errors $e(k) \triangleq \begin{pmatrix} e_t(k) \\ e_x(k) \\ e_y(k) \end{pmatrix}$.

Proposition 5 *It is then possible to express the implemented system as the initial system with an error $e'(k)$ added on the output(s), as shown in Figure 6. $e'(k)$ is the result of the error $e(k)$ through the transfer function H_e defined by:*

$$H_e : z \to C_Z (zI_n - A_Z)^{-1} M_1 + M_2 \tag{22}$$

with

$$M_1 \triangleq \begin{pmatrix} KJ^{-1} & I_n & 0 \end{pmatrix}, \quad M_2 \triangleq \begin{pmatrix} LJ^{-1} & 0 & I_p \end{pmatrix}. \tag{23}$$

The error added on the output(s) is finally known to be (component-wisely) in the intervals $\langle e'_m, e'_r \rangle$, with

$$e'_m = |H_e|_{DC}\, e_m, \quad e'_r = \langle\!\langle H_e \rangle\!\rangle\, e_r. \tag{24}$$

Proof: H_e comes from a reformulation of equation (21), with $e(k)$ as one input of the system. Then separability principle is applied, since the system H is linear, to get system decomposition shown in Figure 6. ■

5. EXAMPLE

As an illustrative example, the following transfer function to implement has been randomly chosen (with the matlab `drss` command[1]):

$$H(z) = \frac{0.4679z^4 - 1.535z^3 + 0.9085z^2 + 0.6147z - 0.6206}{z^4 - 0.3843z^3 - 0.7342z^2 + 0.1943z + 0.0589} \tag{25}$$

[1]All the numerical values are rounded to four significant digits when displayed but the computations are done in double precision.

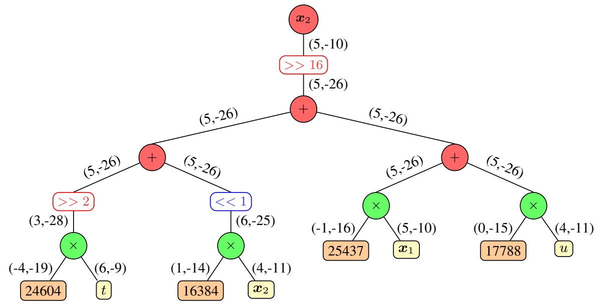

Fig. 7. Fixed-point AST for the computation of x_2

The ρDFIIt structure presented section 2.2 has been chosen. The coefficients $\{\gamma_i\}$ have been chosen so as to minimize the final output error (this optimization process is not detailed here, due to lack of place. With few coefficients, exhaustive search can also be done [6]):

$$\gamma = \begin{pmatrix} -0.1224 & 0.3881 & -0.7620 & 0.8808 \end{pmatrix}^\top. \tag{26}$$

Algorithm 2 exhibits the computations to perform.

```
Input: $u$: the input at step $k$
Output: $y$: the output at step $k$
begin
    // Intermediate variables
    $t \leftarrow x_0 + 0.467892u$ ;
    // Output
    $y \leftarrow t$ ;
    // states
    $x_0 \leftarrow -0.122366x_0 - 1.35548u - 0.00029146t + x_1$ ;
    $x_1 \leftarrow 0.388137x_1 + 0.542843u + 0.046928t + x_2$;
    $x_2 \leftarrow -0.762002x_2 - 0.254215u - 0.00485693t + x_3$;
    $x_3 \leftarrow 0.880823x_3 - 0.141993u + 0.000271706t$;
end
```

Algorithm 2: Example with ρDFIIt

The input u is supposed to be bounded in $[-10; 10]$ and represented with 16 bits, so Proposition 4 proposes to determine the magnitude of each involved variable in the system, and thus their FPF, assuming 16-bit words. Here, the worst-case matrix of H_u is given by

$$\|H_u\|_{\ell^\infty} = \begin{pmatrix} 3.775 & 3.307 & 1.780 & 0.991 & 1.191 & 3.775 \end{pmatrix}^\top$$

so the input u, the output y and the intermediate variable t have $(4, -11)$, $(6, -9)$ and $(6, -9)$ as FPF respectively. The four states have the following FPF: $(6, -9)$, $(5, -10)$, $(4, -11)$ and $(4, -11)$.

The FPF propagation and error analysis shown in section 3.3 have been applied to convert the four SoP in fixed-point arithmetic, assuming 16×16 bits multiplications and 32 bits additions. Figure 7 exhibits the fixed-point AST for the computation of x_2, and Algorithm 3 is the fixed-point version of Algorithm 2 (truncation has been chosen as rounding mode).

From the five SoPs, the following error can be deduced:

$$\underline{e} = \begin{pmatrix} -0.0019531 \\ -0.0019531 \\ -0.0009765 \\ -0.0004882 \\ -0.0004882 \end{pmatrix}, \quad \overline{e} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{27}$$

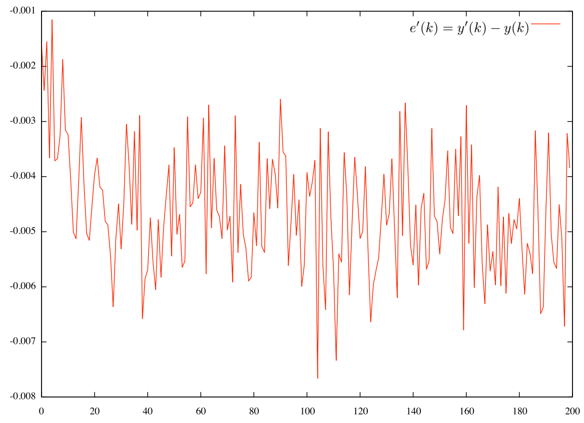

Fig. 8. Evolution of the output error $e'(k) = y'(k) - y(k)$

$|\boldsymbol{H}_e|_{DC}$ and $\|\boldsymbol{H}_e\|_{\ell^\infty}$ are given by

$$|\boldsymbol{H}_e|_{DC} = \begin{pmatrix}1.0706 & 0.9539 & 1.5590 & 0.8848 & 7.4241 & 1\end{pmatrix}$$
$$\|\boldsymbol{H}_e\|_{\ell^\infty} = \begin{pmatrix}1.0756 & 1.2072 & 1.5595 & 3.6539 & 7.4222 & 1\end{pmatrix}$$

so, with Proposition 5, the output error e' can be known to be in $[-0.0105, 0.000928]$.
On Figure 8 is plotted the error $e'(k) = y'(k) - y(k)$ for a white noise input $u(k)$ in $[-10, 10]$. The error between the double floating result and the fixed-point one is relatively small and in the bound predicted by the theory. The mean of the error is around 0.05 that is around the middle of error interval. Of course, the ℓ^∞-norm is a conservative norm (but still achievable), but it gives the worst case error performed by the computations. More specifically, it can give us how many bits of the results are reliable (with faithful rounding).

```
Input: $U$: 16-bit input $(4, -11)$
Output: $Y$: 16-bit output $(6, -9)$
Data: $Rx$: 32-bit registers
Data: $X0$ to $X3$: 16-bit states
// $T$
$R0 \leftarrow 16384 * X0$;
$R1 \leftarrow 30664 * U$;
$R2 \leftarrow (R0 \ll 2) + (R1 \gg 2)$;
$T \leftarrow R2 \gg 16$ ;
// $X0$
$R0 \leftarrow -22208 * U$;
$R1 \leftarrow 16384 * X1$;
$R2 \leftarrow R0 + (R1 \ll 1)$;
$R0 \leftarrow -32078 * X0$;
$R1 \leftarrow -19560 * T$;
$R3 \leftarrow R0 + (R1 \gg 8)$;
$R0 \leftarrow R2 + (R3 \gg 2)$;
$X0 \leftarrow R0 \gg 16$ ;
// $X1$
$R0 \leftarrow 24604 * T$;
$R1 \leftarrow 16384 * X2$;
$R2 \leftarrow (R0 \gg 2) + (R1 \ll 1)$;
$R0 \leftarrow 25437 * X1$;
$R1 \leftarrow 17788 * U$;
$R3 \leftarrow R0 + R1$;
$R0 \leftarrow R2 + R3$;
$X1 \leftarrow R0 \gg 16$ ;
// $X2$
$R0 \leftarrow -20371 * T0$;
$R1 \leftarrow 16384 * X3$;
$R2 \leftarrow (R0 \gg 4) + (R1 \ll 2)$;
$R0 \leftarrow -24969 * X2$;
$R1 \leftarrow -16660 * U$;
$R3 \leftarrow (R0 \ll 1) + R1$;
$R0 \leftarrow R2 + R3$;
$X2 \leftarrow R0 \gg 16$;
// $X3$
$R0 \leftarrow -18611 * U$;
$R1 \leftarrow 18234 * T0$;
$R2 \leftarrow R0 + (R1 \gg 7)$;
$R0 \leftarrow 28863 * X3$;
$R1 \leftarrow (R2 \gg 1) + (R0 \ll 1)$;
$X3 \leftarrow R1 \gg 16$;
// $Y$
$Y \leftarrow T0$;
```

Algorithm 3: Fixed-point algorithm

All these results have been obtained with our tool named *FiPoGen* (as *Fixed-Point Generator*) written in Python. Its code will be soon released under open-source licence.

6. CONCLUSION

Throughout this paper, a more precise method than the one previously introduced in [11] has been proposed to describe automatic fixed-point conversion of linear signal processing or control algorithms. Based on interval arithmetic and linear filter properties, the error analysis shown for Sum-of-Products have been extended to every structure of linear filters/controllers.
Additional work includes code generation for software implementation (algorithm-to-architecture mapping) and hardware implementation (word-length optimization so as to minimize implementation cost such as area under constraints on the output error).

7. REFERENCES

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

[2] B. Widrow and I. Kollár, *Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications*, Cambridge University Press, 2008.

[3] H. Hanselmann, "Implementation of digital controllers - a survey," *Automatica*, vol. 23, no. 1, pp. 7–32, January 1987.

[4] T. Hilaire, "Towards tools and methodology for the fixed-point implementation of linear filters," in *Digital Signal Proc. Workshop (DSP/SPE)*, Jan. 2011, pp. 488–493.

[5] R. Middleton and G. Goodwin, *Digital Control and Estimation, a unified approach*, Prentice-Hall Intern. Editions, 1990.

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

[7] G. Li, J. Chu, and J. Wu, "A matrix factorization-based structure for digital filters," *Signal Processing, IEEE Transactions on*, vol. 55, no. 10, pp. 5108–5112, October 2007.

[8] 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*, January 2011.

[9] T. Hilaire, P. Chevrel, and J.F. Whidborne, "A unifying framework for finite wordlength realizations," *IEEE Trans. on Circuits and Systems*, vol. 8, no. 54, pp. 1765–1774, August 2007.

[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] B. Lopez, T. Hilaire, and L-S. Didier, "Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation," in *Conf. on Design and Architectures for Signal and Image Proc. (DASIP)*, Oct. 2012.

[12] R.E. Moore, *Interval analysis*, Prentice-Hall series in automatic computation. Prentice-Hall, 1966.

[13] L. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," *Audio and Electroacoustics, IEEE Trans. on*, vol. 18, no. 2, pp. 107–122, June 1970.

[14] V. Balakrishnan and S. Boyd, "On computing the worst-case peak gain of linear systems," *Systems & Control Letters*, vol. 19, pp. 265–269, 1992.