TOWARD TOOLS AND METHODOLOGY FOR THE FIXED-POINT IMPLEMENTATION OF LINEAR FILTERS

Thibault Hilaire, IEEE Member
University Pierre et Marie Curie, Paris

ABSTRACT

For embedded systems, Finite Word Length (FWL) effect is still a critical issue in digital filter implementation with fixed-point arithmetic. Few partial solutions exist, but none tackles it in its global nature and consider the complete tradeoff between performance, precision, complexity and hardware cost. This paper proposes to formalize this complex problem and exhibits a unifying approach for the optimal implementation problem of linear filters/controllers. The proposed methodology is based on previously proposed formal description and measures of the FWL effects. A two steps optimization is proposed and a small example emphasizes the process.

Index Terms— Implementation, linear filters, fixed-point arithmetic, optimal realization, methodology.

1. INTRODUCTION

The great majority of embedded signal processing and control algorithms is implemented in digital devices such as DSP, microcontrollers or FPGA. These devices are quite cheap and widely used, but the counterpart is that they have limited resources, in computations or in energy. The fixed-point arithmetic is then often used, but the numerical implementation of the algorithms can suffer from a deterioration in performances and characteristics, due to the finite-precision of the computations. These degradations has two separate origins, corresponding to the quantization of the embedded coefficients and the roundoff errors occurring during the computation.

The constraints on embedded systems are strong (performances/precision, resources, execution time, power consumption, area of the chipset, etc.) and the search for an *optimal implementation* of a given algorithm is a very important topic. For the linear algorithms we are considering here, this problem is studied on three different angles, by people from signal processing, control and computer science. Our purpose here is to aggregate ideas, tools and methods from these three communities and to propose a global solution for the *optimal implementation problem* of linear filters/controllers in fixed-point arithmetic, from their synthesis to their hardware/software implementation and code generation.

The paper is organized as follows. Some classical approaches for the implementation are summarized in section 2, and the optimal realization problem is formalized in section 3. Section 4 exhibits the unifying approach proposed and some *a priori/a posteriori* measures based on formal descriptions of the algorithm and its implementation. Finally, a methodology is proposed in section 5 before being illustrated on an example with the *Finite Wordlength Realization Toolbox* in section 6.

This work has been funded by the engineering school Polytech'Paris-UPMC (University Pierre et Marie Curie, Paris, France) and by the CNRS project *PEPS "ReSyst"*.

2. CLASSICAL TOOLS AND APPROACHES FOR THE IMPLEMENTATION

2.1. State-space realizations

State-space systems and their implementation are widely studied, specially by control community [1, 2]. Let $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$ be a stable, controllable and observable linear discrete time Single Input Single Output (SISO) state-space system, *i.e.*

$$\left\{ \begin{array}{rcl} \boldsymbol{x}(k+1) & = & \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) & = & \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{array} \right. \tag{1}$$

where $u(k)$ is the scalar input, $y(k)$ the scalar output and $\boldsymbol{x}(k)$ is the state vector.
Its input-output relationship is given by the scalar transfer function $h : \mathbb{C} \to \mathbb{C}$ defined by $h : z \mapsto \boldsymbol{c}(z\boldsymbol{I}_n - \boldsymbol{A})^{-1}\boldsymbol{b} + d$.

Since the state-space equations corresponding to a transfer function are not unique, it is possible to select one state-space structure that minimizes the impact of the implementation. Applying a coordinate transformation, defined by $\bar{\boldsymbol{x}}(k) \triangleq \boldsymbol{\mathcal{U}}^{-1}\boldsymbol{x}(k)$ to the state-space system $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$, leads to a new equivalent realization $(\boldsymbol{\mathcal{U}}^{-1}\boldsymbol{A}\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{U}}^{-1}\boldsymbol{b}, \boldsymbol{c}\boldsymbol{\mathcal{U}}, d)$. These two realizations are equivalent in infinite precision but are no more equivalent in finite precision (fixed point arithmetic, floating-point arithmetic, etc.), so it is of interested to evaluate how much the system is sensitive to the coefficient quantization and find one realization among the equivalent ones that minimizes that measure.
It is common to consider the sensitivity of the transfer function with respect to the coefficients. Gevers and Li [1] have proposed the L_2-sensitivity measure to *evaluate* the coefficient roundoff errors. This tractable measure is defined by

$$M_{L_2} \triangleq \left\| \frac{\partial h}{\partial \boldsymbol{A}} \right\|_2^2 + \left\| \frac{\partial h}{\partial \boldsymbol{b}} \right\|_2^2 + \left\| \frac{\partial h}{\partial \boldsymbol{c}} \right\|_2^2 + \left\| \frac{\partial h}{\partial d} \right\|_2^2 \tag{2}$$

and depends on the realization. It is then natural to define the following problem:

$$\boldsymbol{\mathcal{U}}_{opt} = \underset{\boldsymbol{\mathcal{U}} \text{ invertible}}{\arg\min} M_{L_2}(\boldsymbol{\mathcal{U}}). \tag{3}$$

In [1], it is shown that this problem has one unique solution. Hence, for example, a gradient method can be used to solve it.

In addition to the transfer function sensitivity measure, some other sensitivity-based measures have been developed : the perturbations of the system poles is specially studied [3, 4, 2]. Poles are not only structuring parameters, but also indicators of the stability. Let $(\lambda_k)_{1 \leqslant k \leqslant n}$ denote the poles and zeros of the system (*i.e.* the eigenvalues of $\boldsymbol{A}$).Since the FWL error that can cause a stable system to become unstable is determined by how close the pole are to 1 and how sensitive they are to the parameter perturbations, the following

measure is classically used [5]:

$$\Psi \triangleq \sum_{k=1}^{n} \omega_k \left\| \frac{\partial |\lambda_k|}{\partial \boldsymbol{A}} \right\|_F^2, \tag{4}$$

where $(\omega_k)_{1 \leqslant k \leqslant n}$ are some weighting coefficients. Generally $\omega_k = \frac{1}{1-|\lambda_k|}$ to give more weight for the poles closed to the unit circle [5]. The pole sensitivity measure is also used in closed-loop context, in some stability-related measures [2].

2.2. Roundoff noise

In addition to parametric errors, the numerical noises in the computations are also studied. Indeed, each quantization is equivalent to add an independent white noise to the signal to quantized [6, 7] and these noises propagate through the computation up to the output(s).

The most common used criteria for evaluating these roundoff noises is the Signal to Quantization Noise Ratio (SQNR), defined by

$$SQNR \triangleq \frac{\sigma_{\boldsymbol{y}}^2}{\sigma_{\boldsymbol{\xi}^\dagger}^2} = \frac{E\left\{\boldsymbol{y}^\top(k)\boldsymbol{y}(k)\right\}}{E\left\{\boldsymbol{\xi}^\top(k)\boldsymbol{\xi}(k)\right\}} \tag{5}$$

where $\boldsymbol{\xi}^\dagger(k)$ is the overall noise defined as the difference between the mathematical output(s) $\boldsymbol{y}(k)$ and the fixed-point implemented output $\boldsymbol{y}^\dagger(k)$, $E\{.\}$ is the mean operator, and $\sigma_{\boldsymbol{y}}^2$ and $\sigma_{\boldsymbol{\xi}^\dagger}^2$ denote the second-order moment (*i.e.* the power) of the output y and the noise $\boldsymbol{\xi}^\dagger$, respectively,

In other words, the *implemented* output $\boldsymbol{y}^\dagger$ can be seen as the *mathematical* output $\boldsymbol{y}$, perturbed with an additive noise $\boldsymbol{\xi}^\dagger(k)$. Moreover, if $\boldsymbol{\xi}(k)$ is a vector collecting all the roundoff noise occurring from quantizations in the computations at step k, then $\boldsymbol{\xi}^\dagger(k)$ can be seen as the output of $\boldsymbol{\xi}(k)$ through a given filter $\boldsymbol{H}_{\boldsymbol{\xi}}$ to determine, as shown in figure 1 [8] (this is due to the linearity of the filter).

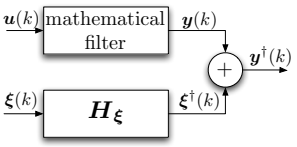

Fig. 1. Implemented filter, with quantization noises extracted

Different methods are used to determined the SQNR (5) or the roundoff noise power $\sigma_{\boldsymbol{\xi}^\dagger}^2$. This could be measured by simulations [9, 10] or in some cases evaluated by analytical approaches [8, 11].

2.3. Different possible structures

An important point in our approach is the set of possible algorithms to realize a given filter/controller. Some of them are quite known and used, like the Direct Form, but they do not often propose good numerical properties. In addition to the state-space realizations, where an infinity of equivalent realizations are possible (parametrized by a transformation matrix $\boldsymbol{\mathcal{U}}$), it exists plenty of other equivalent realizations.

For a n-th order filter, these structured realization can have from $2n+1$ coefficients (direct forms) up to $(n+1)^2$ (fully parametrized state-space realizations) or even more, but all with different robustness to the fixed-point implementation.

Some of the interesting structures that should be considered are:

- the classical forms (such as Direct Form I and II, transposed or not, state-space realization, ...) ;
- realizations 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 [12]. These realizations are similar to the classical realizations (direct form or state-space), but using a δ^{-1} operator instead of a delay q^{-1} operator. They often present superior numerical properties since $\Delta > 1$ [1] ;
- realizations with the ρ-operator, a very promising extension of the δ-operator with extra-parameters. The ρ-Direct Form II transposed (ρDFIIt) [13] and the ρ-modal forms [14] are good examples;
- wave lattice filter, warped filters and some other specific realizations like LGC or LCW-structures [15], ...

Except the direct forms, all these structured realizations are parametrized by several parameters (like the transformation matrix $\boldsymbol{\mathcal{U}}$ for the q-state-space), leading for each structure to an infinite number of realizations. In addition, some values of these tuning parameters can lead to sparse realizations (*i.e.* realizations with trivial coefficients like 0 or ± 1), to decrease the computational cost [16].
Also, it is possible to decompose a filter in cascaded and/or parallel filters, and each one could be realized with one of the previous structures (mixed realizations are also possible), increasing the number of possibilities.

All these equivalent realizations have different FWL properties and different number of coefficients. So they should all be taken in consideration for the research of the *optimal implementation* of a given filter, in order to determine the tradeoff between precision and computational cost.

3. OPTIMAL REALIZATION PROBLEM

These different aspects of the same problem show that it is very important to consider the problem from every angle and to build appropriate tools and methodology for the *search for an optimally implemented realization*.
This is a very difficult multi-objectives optimization problem, and as far as we know, there is no such global approach in the literature[1].

There is two kinds of objectives to consider:

- Some objectives relative to performance under finite precision. They should be able to evaluate
  - that the output roundoff noise induced by the implementation is *quite low* relatively to the output ;
  - that the transfer function is *not too much* modified, specially for certain frequency ranges. If the filter/controller were designed to follow certain specifications like stopband/passband frequency and/or attenuation, its fixed-point version should also respect them ;
  - that the stability is preserved, or that the poles (specially in closed-loop context) do not move *too much*.
- Some objectives relative to computational cost:
  - the number of operations, the number of coefficients;

[1] For example, `fdatool` from Mathworks helps to implement filters by showing characteristics of the implemented version, according to some realization and implementation scheme, but no optimization on the realization or the wordlength are performed.

– the surface or the power consumption if the filter is implemented on FPGA or ASIC ;

– the computational time: it is linked to the number of operations, but also to its parallelization degree (for example, the ρ-modal realization exhibited in [14] is highly parallelisable).

These different objectives can be organized in two categories:

- those that do not consider the concrete HW/SW fixed-point implementation. We can denote them *a priori* measures, because they do not require any information about the fixed-point implementation and only consider the algorithm used to numerically realize the filter/controller. They estimate the *robustness* of the algorithm with respect to the implementation.
- those that are based on a concrete HW/SW fixed-point implementation. We can denote them *a posteriori* because they can only be applied after having completed the implementation process. They can evaluate or measure the impact of a given implementation (depending on the precision of the criteria used).

With these two different levels of objectives, there is two different optimization steps. The first is based on *a priori* implementations and considering the algorithms only (the choice of one realization among the set of equivalent realizations), whereas the second is based on the implementation itself (wordlength, quantization modes, shifts to align the binary-point position for the arithmetic operations, etc.). Both are linked but it is too difficult for the moment to optimize them simultaneously.

4. UNIFYING APPROACH

After having defined the optimal realization problem, we need a general unifying approach in order to formalize the set of equivalent realizations, to formalize a fixed-point implementation that can guaranty no overflows or underflows (depending on wordlengths used for the coefficients, variables and the arithmetic operators) and to design some tractable analytic measures to evaluate the FWL impact[2].

4.1. Specialized Implicit Framework

Many controller/filter forms, such as lattice filters and δ-operator controllers, make use of intermediate variables, and hence cannot be expressed in the traditional state-space form. The SIF has been proposed in [17] in order to model a much wider class of discrete-time linear time-invariant controller/filter realizations.

The model takes the form of an implicit state-space realization specialized according to

$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_n & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \tag{6}$$

where the matrix $\boldsymbol{J}$ is lower triangular with 1's on the main diagonal. Note $\boldsymbol{x}$ is the state-vector and is stored from one step to the next, whilst the vector $\boldsymbol{t}$ plays a particular role as $\boldsymbol{t}(k+1)$ is independent of $\boldsymbol{t}(k)$ and is defined as the vector of intermediary variables. The particular structure of $\boldsymbol{J}$ allows to model how the computations are decomposed with intermediates results that are reused in the same step [17].

It is implicitly assumed throughout the paper that the computations associated with the realization (6) are executed in row order, giving the following algorithm:

$$\begin{aligned} &\text{[i]} \quad \boldsymbol{J}.\boldsymbol{t}(k+1) \leftarrow \boldsymbol{M}.\boldsymbol{x}(k) + \boldsymbol{N}.\boldsymbol{u}(k) \\ &\text{[ii]} \quad \boldsymbol{x}(k+1) \leftarrow \boldsymbol{K}.\boldsymbol{t}(k+1) + \boldsymbol{P}.\boldsymbol{x}(k) + \boldsymbol{Q}.\boldsymbol{u}(k) \\ &\text{[iii]} \quad \boldsymbol{y}(k) \leftarrow \boldsymbol{L}.\boldsymbol{t}(k+1) + \boldsymbol{R}.\boldsymbol{x}(k) + \boldsymbol{S}.\boldsymbol{u}(k) \end{aligned} \tag{7}$$

Note that in practice, steps [ii] and [iii] could be exchanged to reduce the computational delay. Also note that because the computations are executed in row order and since J is lower triangular with 1's on the main diagonal, there is no need to compute J^{-1}.

A **realization** of a transfer matrix H is entirely defined by the matrix $\boldsymbol{Z}$, where $\boldsymbol{Z}$ is partitioned according to[3]

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \tag{8}$$

In order to exploit the potential offered by the specialized implicit form in improving implementations, it is necessary to describe sets of equivalent system realizations.

Let us consider a realization $\boldsymbol{Z}_0$. Then all the realizations $\boldsymbol{Z}_1$ with

$$\boldsymbol{Z}_1 = \begin{pmatrix} \boldsymbol{\mathcal{Y}} & & \\ & \boldsymbol{\mathcal{U}}^{-1} & \\ & & I_p \end{pmatrix} \boldsymbol{Z}_0 \begin{pmatrix} \boldsymbol{\mathcal{W}} & & \\ & \boldsymbol{\mathcal{U}} & \\ & & I_m \end{pmatrix} \tag{9}$$

and $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{W}}$, $\boldsymbol{\mathcal{Y}}$ are non-singular matrices, are equivalent to $\boldsymbol{Z}_0$, and share the same complexity (i.e. generically the same amount of computation). It is also possible to just consider a subset of similarity transformations that preserve a particular structure, by adding specific constraints on $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{W}}$ or $\boldsymbol{\mathcal{Y}}$. Then, the set of equivalent realizations with the same structure will be defined with an initial realization $\boldsymbol{Z}_0$ and some constraints on the transformation matrices.

All the linear realizations (direct forms, state-space, δ-realizations, ρ-realizations, cascade decomposition, etc.) can be easily described with the SIF. For example, a δ-state-space realization [12, 1]defined by

$$\begin{cases} \delta[\boldsymbol{x}(k)] = \boldsymbol{A}_\delta \boldsymbol{x}(k) + \boldsymbol{B}_\delta \boldsymbol{u}(k) \\ \boldsymbol{y}(k) = \boldsymbol{C}_\delta \boldsymbol{x}(k) + \boldsymbol{D}_\delta \boldsymbol{u}(k) \end{cases} \tag{10}$$

with $\delta = \frac{q-1}{\Delta}$ ($\Delta \in \mathbb{R}_{+*}$ and q is the shift operator) can be expressed as

$$\begin{pmatrix} \boldsymbol{I}_n & \boldsymbol{0} & \boldsymbol{0} \\ -\Delta \boldsymbol{I}_n & \boldsymbol{I}_n & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}_\delta & \boldsymbol{B}_\delta \\ \boldsymbol{0} & \boldsymbol{I}_n & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_\delta & \boldsymbol{D}_\delta \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \tag{11}$$

4.2. *A priori* measures

4.2.1. Input-output and pole sensitivity

The classical sensitivity measures (transfer function and pole sensitivity) have been extended to the SIF in [17] (in [18], they are also extended to controllers in closed-loop context, but not presented here

[2] Due to lack of space, their analytical expression are not explicitly given here, but can be found in the mentioned references.

[3] This notation is introduced to make the further developments more compact.

due to a lack of space). In that context, the L_2-sensitivity measure in SISO context[4] is defined by

$$M_{L_2} \triangleq \left\| \frac{\partial h}{\partial \boldsymbol{Z}} \times \boldsymbol{\delta_Z} \right\|_2^2 \tag{12}$$

where $\boldsymbol{\delta_Z}$ is a weighting matrix indicating which coefficient of $\boldsymbol{Z}$ will introduce error:

$$(\delta_{\boldsymbol{Z}})_{ij} \triangleq \begin{cases} 0 & \text{if } \boldsymbol{Z}_{ij} \text{ is exactly implemented} \\ 1 & \text{otherwise.} \end{cases} \tag{13}$$

Moreover, the L_2-norm in (12) can also be weighted by an other transfer function so as to take more in consideration some frequency ranges over the others.

The same extension to the SIF has been made for the pole sensitivity measure. It is now defined by

$$\Psi \triangleq \sum_{k=1}^{n} \omega_k \left\| \frac{\partial |\lambda_k|}{\partial \boldsymbol{Z}} \times \boldsymbol{\delta_Z} \right\|_F^2. \tag{14}$$

These two measure are tractable and their analytical expression can be found in [17].

4.2.2. Roundoff noise gain

By considering the quantization noises after each multiplication, the algorithm in (7) becomes:

$$\begin{aligned} &\text{[i]} \quad \boldsymbol{Jt}(k+1) \leftarrow \boldsymbol{Mx}(k) + \boldsymbol{Nu}(k) + \boldsymbol{\xi_t}(k) \\ &\text{[ii]} \quad \boldsymbol{x}(k+1) \leftarrow \boldsymbol{Kt}(k+1) + \boldsymbol{Px}(k) + \boldsymbol{Qu}(k) + \boldsymbol{\xi_x}(k) \\ &\text{[iii]} \quad \boldsymbol{y}(k) \leftarrow \boldsymbol{Lt}(k+1) + \boldsymbol{Rx}(k) + \boldsymbol{Su}(k) + \boldsymbol{\xi_y}(k) \end{aligned}$$

where $\boldsymbol{\xi_t}$, $\boldsymbol{\xi_x}$ and $\boldsymbol{\xi_y}$ are the noises sources corrupting $\boldsymbol{t}$, $\boldsymbol{x}$ and $\boldsymbol{y}$. These noises are usually modeled as independent white sequences. Denote $\boldsymbol{\xi}$ the vector formed by all these noises:

$$\boldsymbol{\xi}(k) = \begin{pmatrix} \boldsymbol{\xi_t}(k) \\ \boldsymbol{\xi_x}(k) \\ \boldsymbol{\xi_y}(k) \end{pmatrix}. \tag{15}$$

It is possible to aggregate all of them as an additive noise $\boldsymbol{\xi}'(k)$ on the output. This (colored) noise results from the filtering of $\boldsymbol{\xi}(k)$ via a transfer function $\boldsymbol{H_\xi}$ (in Fig.1), directly determined from the matrices $\boldsymbol{J}$, $\boldsymbol{K}$, $\boldsymbol{L}$, $\boldsymbol{M}$, $\boldsymbol{N}$, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{S}$ (*i.e.* the transfer function from the intermediate variables, the states and the output to the output).

Suppose that the roundoff noises $\boldsymbol{\xi}$ have all the same power σ_0^2. The roundoff noise gain (RNG) measure is then defined by the ratio of the power of global noise $\xi'(k)$ and σ_o^2 [19]:

$$RNG \triangleq \frac{E\{\boldsymbol{\xi}'(k)^\top \boldsymbol{\xi}'(k)\}}{\sigma_0^2}. \tag{16}$$

Its analytical expression can be found in [20]. It depends on $\boldsymbol{Z}$, the number of non-trivial parameters in $\boldsymbol{Z}$ and the observability Gramian of the system.

[4] It has also been extended to MIMO case.

4.3. Fixed-point implementation scheme

In order to refine measures like the Roundoff Noise Gain, the exact fixed-point representation should be known, specially the binary-point position of each variable and intermediate computation, and all the required shift necessary to align the binary-point position and perform the additions.

The notation (β, γ) is used for the fixed-point representation of a variable or coefficient (2's complement scheme), according to Figure 2. β is the total wordlength of the representation in bits, whereas γ is the wordlength of the fractional part (it determines the position of the binary-point). They are suffixed by the variable/coefficient they refer to and could be scalars, vectors or matrices.

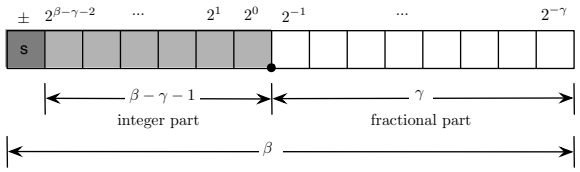

Fig. 2. Fixed-point representation

The operation in algorithm (7) only requires scalar product, as shown on figure 3 (some possible shifts can be added after multiplication, in order to align the binary-point positions).

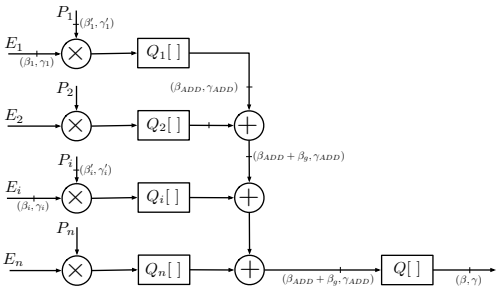

Fig. 3. Scalar product detailled with operations and fixed-point format

$\beta_{\boldsymbol{Z}}$ (the coefficients wordlengths), $\beta_{\boldsymbol{u}}$, $\beta_{\boldsymbol{y}}$, $\beta_{\boldsymbol{t}}$, $\beta_{\boldsymbol{x}}$ (the inputs, outputs, intermediate variables and states coefficients wordlengths) and β_{ADD} (the accumulator operators wordlength) are supposed to be known. $\gamma_{\boldsymbol{u}}$ is also known (or deduced by $\gamma_{\boldsymbol{u}} = \beta_{\boldsymbol{u}} - 1 - \left\lfloor \log_2 \max(\boldsymbol{u}) \right\rfloor$). It is also supposed that the accumulations (in each scalar product, see figure 3) are done on the same fixed-point format (no shift between two consecutive additions).

Then, the binary-point positions $\gamma_{\boldsymbol{t}}$, $\gamma_{\boldsymbol{x}}$, $\gamma_{\boldsymbol{y}}$ are deduced. This could be done with the L_1-norm of the transfer function from the intermediate variables, the state and the output to the output (that can also be expressed from $\boldsymbol{Z}$). The L_2-norm can also be used (less conservative than the L_1-one), but cannot guaranty the overflows.

Afterwards, the common fixed-point format of each accumulator γ_{ADD} can be set in order to represent the dynamic of each product without overflow and the final result without overflow.

Finally, in order to align the results of the products, two computational schemes are possibles:

- Roundoff After Multiplication: the result of the product is

	truncation	best roundoff
μ_e	$2^{-\gamma-1}(1-2^{-d})$	$2^{-\gamma-d-1}$
σ_e^2	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$

Table 1. Right shift of d bits the signal with $(\beta+d, \gamma+d)$ as representation is similar to add a white independent noise e with μ_e and σ_e^2 as first and second-order moments [6, 20].

right shifted so as to meet the accumulator's fixed-point format (as shown in figure 3)

- Roundoff Before Multiplication: the required quantization is moved to the coefficient. In that case, no extra right shifts are required for the scalar product computations, but the coefficients can be represented with less significant bits that they can be. The roundoff noise is changed as a parametric error (see sections 4.2.1 and 4.2.2).

4.4. *A posteriori* measures

Determining the representation (β, γ) of everything in the computations allows to know which quantizations are performed (right shift operators) and then the first and second moment order of each element of $\xi(k)$, according to the table 1.

So the first and second-order moments of the noises $\boldsymbol{\xi}(k)$ can be deduced, and the roundoff noise power $\sigma_{\boldsymbol{\xi}^\dagger}$ can be computed with [20]

$$\sigma^2_{\boldsymbol{\xi}^\dagger} = \|\boldsymbol{\phi}_{\boldsymbol{\xi}} \boldsymbol{H}_{\boldsymbol{\xi}}\|_2^2 \tag{17}$$

where $\boldsymbol{\phi}_{\boldsymbol{\xi}}$ satisfies $\boldsymbol{\psi}_{\boldsymbol{\xi}} = \boldsymbol{\phi}_{\boldsymbol{\xi}} \boldsymbol{\phi}_{\boldsymbol{\xi}}^\top$ and $\boldsymbol{\psi}_{\boldsymbol{\xi}}$ is the covariance matrix of $\boldsymbol{\xi}$.

It is also possible to use the norms $\left\|\boldsymbol{h} - \boldsymbol{h}^\dagger\right\|_2$ and $\left\||\boldsymbol{\lambda}| - \left|\boldsymbol{\lambda}^\dagger\right|\right\|_F$ where $\boldsymbol{h}^\dagger$ and $\boldsymbol{\lambda}^\dagger$ denote the transfer function and the poles with quantized coefficients respectively. But these measures are highly non-smooth, non analytical and cannot be used in the optimization steps.

Finally, hardware measures like area can be built. Basically, they depends linearly from the number of involved additions and quadratically for the multiplications.

5. METHODOLOGY

The different tools proposed on the *Specialized Implicit Framework* allow us to design a complete methodology taking in consideration the equivalent realizations and the Software/Hardware possibilities. It is represented on figure 4, with the following steps:

a) Choose a structure (from the existing ones, such state-space, δ-state-space, ρDFIIt, ρ-modal, cascade decomposition where each sub-filter can be in any structure, etc.) ;

b) Use the SIF formalization and obtain the set of equivalent structured realizations (mainly a realization $\boldsymbol{Z}_0$, plus additive constraints on transformation matrices $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{W}}$ and $\boldsymbol{\mathcal{Y}}$) ;

c) *a priori* measures are used for the optimization of the realization, with a gradient or a global optimization algorithm (like Adaptive Simulated Annealing [21]). Classical multi-objectives techniques can be used, such as defining a weighted criteria $\mathcal{J}$ by

$$\mathcal{J} \triangleq \frac{M_{L_2}}{M_{L_2}^{opt}} + \frac{\Psi}{\Psi^{opt}} + \frac{RNG}{RNG^{opt}} \tag{18}$$

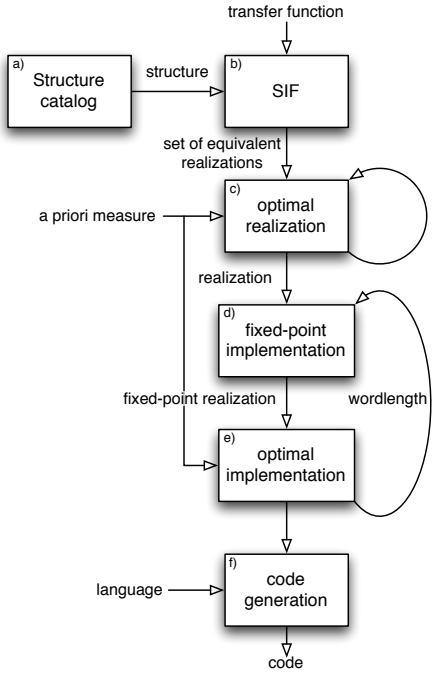

Fig. 4. Methodology based on a unifying approach

where $M_{L_2}^{opt}$, Ψ^{opt}, RNG^{opt} are optimal value for the M_{L_2}, Ψ and RNG measures, respectively (this requires three extra steps of single objective optimization) ;

d) According to the wordlength, the fixed-point implementation is determied ;

e) If the hardware architecture allows multiple wordlength paradigm (FPGA, ASIC, ...), then optimal implementation is obtain by optimizing an *a posteriori* criteria (typically area or wordlength-related measure) under precision constraints given by an *a priori* criteria ;

f) Finally, automatic code generation can be performed, in various languages (VHDL, C, Matlab, ...).

This methodology, based on formal representation and analytical criteria, is an important step to deal with the optimal implementation problem. Methods for points c) and d) are not developed in details here due to lack of space, although tractable with classic optimization methods. More specific optimization algorithms should be now developed and compared for that specific problem.

6. EXAMPLE WITH THE FWR TOOLBOX

A Matlab open-source toolbox, called *Finite Wordlength Realization Toolbox* (FWR Toolbox) have been released, based on the SIF and the unifying approach[5]. It is yet able to deal with most of the methodology and the mentioned measures, even if it is still in development. It is based on tow classes (`FWR` and `FWS`) that can represent realizations and structurations, with methods to compute *a priori*/*a posteriori* measures, perform optimization and export code.

Let us consider the filter obtained by the Matlab command `butter(4,0.125)`. Due to lack of space, it is not possible here

[5]It is freely available at `http://fwrtoolbox.gforge.inria.fr`

to detail every steps, but only the result for the ρ-DFIIt structure (proposed in [13] and described with the SIF in [18]). This structure has n free parameters denoted $\boldsymbol{\gamma}$ that are here used for the optimization of tradeoff measure $\mathcal{J}$. The optimization step c) with criterion $\mathcal{J}$ gives $\boldsymbol{\gamma} = \begin{pmatrix} 0.24998 & 0.80129 & 0.72471 & 0.70086 \end{pmatrix}^\top$. The optimized realization is then implemented in 16 bits (32 bits for the accumulation), with a Roundoff After Multiplication scheme, as shown in algorithm 1. It involves 11 multiplications and 12 additions (plus extra shifts), and proposes low transfer function sensitivity, low pole sensitivity and low roundoff noise gain.

7. CONCLUSION

This paper deals with the optimal implementation problem of linear filters. After having formalized the problem, we have proposed a unifying approach for the description of the equivalent realizations and the different FWL measures to use. Some are applied on the realization level, whereas some consider the implementation level.
A first methodology, combining two optimization steps, is proposed. Our current work is now focused on *ad hoc* constrained optimization algorithms and new specific structures. The development of the FWR Toolbox will be continued, so as to propose a graphical tool able to apply the proposed methodology and generate optimal fixed-point implementations.

8. REFERENCES

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

[2] R. Istepanian and J.F. Whidborne, Eds., *Digital Controller implementation and fragility*, Springer, 2001.

[3] Gang Li, "On pole and zero sensitivity of linear systems," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 44, no. 7, pp. 583 –590, jul 1997.

[4] J.F. Whidborne, J. Wu, and R.H. Istepanian, "Finite word length stability issues in an ℓ_1 framework," *Int. J. Control*, vol. 73, no. 2, pp. 166–176, 2000.

[5] J. Wu, S. Chen, G. Li, R. Istepanian, and J. Chu, "An improved closed-loop stability related measure for finite-precision digital controller realizations," *IEEE Trans. Automatic Control*, vol. 46, no. 7, pp. 1162–1166, 2001.

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

[7] G. Constantinides, P. Cheung, and W. Luk, "Truncation Noise in Fixed-Point SFGs," *IEE Electronics Letters*, vol. 35, no. 23, pp. 2012–2014, November 1999.

[8] D. Ménard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," in *Proceedings of DATE02 (Design Automation and Test in Europe)*, march 2002.

[9] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE : A fixed-point design and simulation environment," *EDAA*, 1998.

[10] S. Kim, KI. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems*, vol. 45, no. 11, pp. 1455–1464, November 1998.

[11] R. Rocher, D. Menard, O. Sentieys, and P. Scalart, "Analytical accuracy evaluation of fixed-point systems," in *Proc. European Signal Processing Conference (EUSIPCO'07)*, 2007.

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

[13] 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.

[14] Y. Feng, P. Chevrel, and T. Hilaire, "A practival strategy of an efficient and sparse FWL implementation of lti filters," in *European Control Conference (ECC'09)*, 2009.

[15] 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.

[16] J. Wu, S. Chen, G. Li, and J. Chu, "Constructing sparse realizations of finite-precision digital controllers based on a closed-loop stability related measure," *IEE Proc. Control Theory and Applications*, vol. 150, no. 1, pp. 61–68, January 2003.

[17] 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.

[18] T. Hilaire, P. Chevrel, and J.F. Whidborne, "Finite wordlength controller realizations using the specialized implicit form," *Int. Journal of Control*, vol. 83, no. 2, pp. 330–346, February 2010.

[19] C. Mullis and R. Roberts, "Synthesis of minimum roundoff noise fixed point digital filters," in *IEEE Transactions on Circuits and Systems*, September 1976, vol. CAS-23.

[20] T. Hilaire, D. Ménard, and O. Sentieys, "Bit accurate roundoff noise analysis of fixed-point linear controllers," in *Proc.* IEEE Int. Symposium on Computer-Aided Control System Design *(CACSD'08)*, September 2008.

[21] L. Ingber, "Adaptive simulated annealing (ASA): Lessons learned," *Control and Cybernetics*, vol. 25, pp. 33–54, 1996.

[22] Kyungtae Han and Brian L. Evans, "Optimum wordlength search using sensitivity information," *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 76, uary.

Input: u: 16 bits integer
Output: y: 16 bits integer
Data: xn: array [1..5] of 16 bits integers
Data: T: 16 bits integers
Data: Acc: 32 bits integer
begin

```
// Intermediate variables
Acc ← xn(1) << 14;
Acc ← Acc + (u * 31323) >> 11;
T ← Acc >> 14;
// States
Acc ← T << 14;
Acc ← Acc + xn(2) << 14;
Acc ← Acc + (xn(1) * 32766) >> 2;
Acc ← Acc + (u * 25359) >> 7;
xn(1) ← Acc >> 15;
Acc ← (T * −26735) >> 2;
Acc ← Acc + xn(3) << 13;
Acc ← Acc + (xn(2) * 26257);
Acc ← Acc + (u * 17831) >> 4;
xn(2) ← Acc >> 15;
Acc ← (T * −32768) >> 5;
Acc ← Acc + xn(4) << 13;
Acc ← Acc + (xn(3) * 23747);
Acc ← Acc + (u * 19675) >> 2;
xn(3) ← Acc >> 15;
Acc ← (T * −21440) >> 4;
Acc ← Acc + (xn(4) * 22966);
Acc ← Acc + u << 13;
xn(4) ← Acc >> 15;
// Outputs
y ← T1;
```

end

Algorithm 1: optimal 16-bit ρDFIIt