Thesis mid-term report.

"Towards unified code generation for libms and filters."

May 4, 2016

PhD student Anastasia Volkova

Thesis director Jean-Claude Bajard

Advisors Thibault Hilaire, Christoph Lauter

1 Introduction

A digital filter, or a general digital signal processing system, operates on a discrete input data to produce a discrete output by means of a computational algorithm. We are interested in such algorithms that are implemented for control and signal processing application in embedded systems. Regardless of the type of embedded system, the data processed by the digital system is ultimately stored in memory registers with finite capacity. Therefore, all the computations are performed in finite-precision arithmetic and thus result in an inherent limitation on the accuracy of the result. In this thesis we investigate the effects of finite-precision implementation of digital filters. We apply a rigorous computer arithmetic approach to evaluate the errors due to those effects in order to provide a reliable implementation.

We consider implementation of discrete Linear Time-Invariant (LTI) systems [1] and more specifically infinite impulse response filters, which means that in response to a brief initial input the filter output does not become exactly zero past a certain point but continues indefinitely.

A discrete-time digital filter is usually characterized by its transfer function. A transfer function is a representation in frequency domain of the relation between the input and output of a LTI system with zero initial conditions. It can be expressed as the $\mathcal{Z}$-transform of the inputs and outputs of the filter [2] and it has the form $H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1+\sum_{i=1}^{n} a_i z^{-i}}$, where n is the order of the filter [3]. Designing a filter consists of designing a transfer function which meets the filter specifications (e.g. degree).

A given transfer function may be evaluated in many different ways, just like polynomials can be evaluated in many different ways [4]. Further in this work we will refer to filter structures as to different ways to evaluate a filter; and to filter realizations as to concrete instances of a filter structure. All filter structures describe different computational algorithms of the same mathematical object, the transfer function. A lot of structures have been developed over time, such as Direct Forms [5], State-Space [6], Wave [7], and every year some new ones appear, for example a new wave structure appeared in March 2016 [8].

Once the filter structure is chosen, software or hardware implementations may be provided. The software implementation is providing a list of formal instructions that are supported by a hardware target (e.g. CPU). The hardware implementation is for instance a generation of a circuit architecture, such as VHDL [9], which will carry out the filter algorithm. We will concentrate on software implementation of digital filters.

In software implementation numbers can be represented in different ways, consequently arithmetic operations are carried out differently. Most of embedded digital signal processing algorithms use the fixed-point arithmetic [10]. With this representation the numbers are stored in a machine as integers (mantissa).

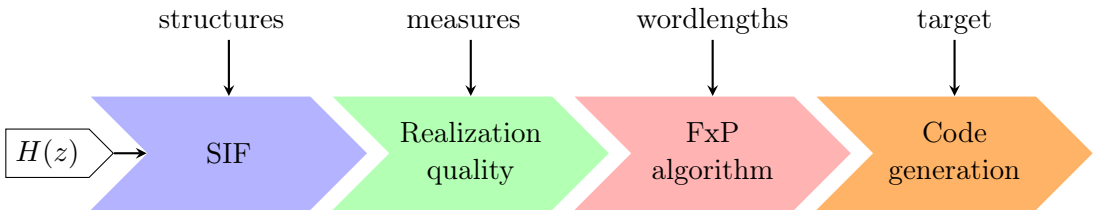


Figure 1: Automatic filter generator flow.

However, on each step of the filter implementation various issues arise. First, the given transfer function is an exact mathematical object, which needs to be quantized for a finite-precision implementation. The quantization of the filter coefficients perturbates the location of the filter poles and zeroes. As a result, the actual filter output differs from the ideal one.

Moreover, similar to the polynomial evaluation [4], filter realizations have different numerical properties in finite-precision arithmetic. Some structures are more efficient in terms of number of operations or storage elements required for their implementation, and others provide advantages such as improved numerical stability and reduced round-off error. In order to choose the best realization according to speed, error, power consumption or some other criteria one must be able to compare them fairly. Various quality measures exist for filters. However, these measures are often structure-specific, so the comparison with other realizations is impossible or rather complicated.

Finally, using finite-precision arithmetic leads to rounding errors within the filter computations. Also, it is possible for internal computations to overflow, which leads to eventual non-linearity of the implemented filter or. Moreover, in some applications, such as avionics, overflows can cause serious system faults.

The transfer function quantization, rounding errors, risk of overflows - all these finite wordlength effects lead to degradation of the actually implemented filter. Therefore, an accurate error analysis of the filter implementation is required. In the signal processing domain this question is usually addressed from a statistical point of view [11]. However, this approach does not give a mathematical guarantee on the error of the filter implementation. In some applications a low probability of filter failure is tolerated. However in others, such as aeronautics, it is a crucial to have a rigorous error-bound. Moreover, the filter analysis techniques vary from structure to structure because of lack of a unified representation.

This thesis is a part of an ANR project "Metalibm", one of goals of which is to provide an automatized filter-to-code generator, which will encapsulate the above described process under a unified work-flow and provide a reliable implementation for any LTI system. The idea is to provide rigorous evaluation of the errors due to a finite-precision implementation. This will allow to determine the implementation parameters (structure, fixed-point formats for number representations, order of arithmetic operations etc.), which guarantee a user-given error bound.

The proposed generator is based on the Specialized Implicit Framework (SIF), a modeling tool, which was introduced in [12]. It allows us to represent any LTI algorithm in a unified form which solves the problem of realizations comparison. Therefore, once rigorous error analysis techniques are developed for the SIF formalism, they can be applied upon any linear realization.

A simplified scheme of the proposed work-flow is given in Fig. 1. The filter-to-code generation can be divided into four stages:

1. given a transfer function, choose a structure and represent the filter realization in the SIF formalism;
2. evaluate the quality of the chosen filter realization;
3. determine the parameters of the fixed-point representations of filter variables which will ensure that no overflow occurs and the user-provided error-bound on the filter implementation is met;

4. given software or hardware specifications, generate the fixed-point code or a VHDL scheme.

Stages 1 to 4 can be repeated for all possible structures. This generator can be also used as a black-box tool which does not require any interference in the decision-making process of filter implementation. Given a transfer function and quality criteria (e.g. output error bound and parallelism requirement), the generator chooses a structure which suits the best the given criteria and generates software and hardware implementations. To achieve this, all the stages of the generator are plugged into various optimization routines (e.g. structure choice or fixed-point formats optimization under given criteria).

Some of the most popular filter structures, such as Direct Forms and state-space have already been represented in the SIF formalism [12], [13], [14]. Various filter quality measures have been adopted for the SIF formalism as extension of all the classical measures defined for state-space realizations. Some new, more realistic, measures have been introduced in [15]. For hardware implementations these measures take into account different impact on the output error of different variables. An approach on bounding the variables of a filter implementation has been proposed in [16]. The final stage of the generator was developed in [17], where an automatic Fixed-Point code generation tool for the SIF has been proposed and is based on particular implementation of Sum-of-Products (SoPs).

However, various challenging questions of filter implementation have not been investigated, yet. The first goal of the thesis is to get deep insight on the whole filter-to-code generation process. Then, to provide a rigorous evaluation of the rounding errors that occur in the implemented filter and, consequently, provide a fixed-point implementation, which guarantees an error-bound criteria and that no overflow occurs.

In order to get familiar with the proposed filter implementation process in general and with SIF in particular, we provided an algorithm for a conversion of new structure, Lattice Wave digital filters (a highly-modular parallel structure), to the SIF formalism. A fair comparison of the Lattice Wave filters to the structures that are already present in the SIF framework was performed using various sensitivity-based measures. This work led to a publication at the 2015 European Signal Processing Conference (EUSIPCO 2015) [18].

The main contribution of the thesis so far is ensuring the Stage 3 of the filter generator, i.e. providing a reliable implementation of digital filters in Fixed-Point arithmetic. To deduce the fixed-point representations of all variables involved in the filter computation we must first deduce their range. As it was described above, in [16] an approach on bounding the output of a linear filter was proposed. This approach is based on a computation of a largest possible peak value of the output of a filter[19]. However, this measure itself cannot be computed exactly and the error analysis of its computation is not trivial. We proposed an algorithm of reliable computation of this measure in arbitrary precision. We provided rigorous numerical analysis of the algorithm as well as its implementation. This work led to a publication at the 22nd IEEE Symposium on Computer Arithmetic (ARITH22) [20].

However, deducing the range of the filter is not sufficient. Due to rounding errors of the internal computations within the filter, the output interval of the implemented filter differs from the ideal one. Moreover, since the filters are recursive, the error propagation is non-linear. Taking this effect into account is crucial to ensure that no overflows occur.

We applied the WCPG measure to provide a rigorous approach on LTI filters error analysis which does not use any simulations but proofs. We take into account the error though filter propagation and provide a fixed-point implementation, which guarantees that no overflow occurs. Moreover, we proved that the determined most significant bit positions are never overestimated more than by one bit. This work led to a publication at the 49th Asilomar Conference on Signals, Systems and Computers (Asilomar 49) [21].

The second part of the "Metalibm" project is providing a code generator for mathematical functions. The code generator provides correctly-rounded implementations for various elementary functions. The second goal of my work is drawing a parallel between digital filters generation and mathematical functions implementation and will be addressed in the second part of the thesis.

2 State of the Art

Notation: Throughout the article matrices are in uppercase boldface, vectors are in lowercase boldface, scalars are in lowercase. All matrix inequalities and absolute values are considered element-by-element.

2.1 Linear Time-Invariant Filters

A Linear Time Invariant (LTI) filter is a numerical application that transforms an input signal $\{\boldsymbol{u}(k)\}_{k\geq 0}$ into an output signal $\{\boldsymbol{y}(k)\}_{k\geq 0}$ ($\boldsymbol{u}(k)$ and $\boldsymbol{y}(k)$ may be vectors or scalars), where $k \in \mathbb{N}$ is the step time. We consider here only stable Infinite Impulse Response (IIR) systems and, for generalization purposes, with Multiple Inputs Multiple Outputs (MIMO), i.e. vectors $\boldsymbol{u}(k)$ and $\boldsymbol{y}(k)$.

In general the input-output relations of such systems are represented by a transfer function. As a network structure, such function is most commonly represented by a Signal Flow Diagram (e.g. see Fig. 3).

An analytical representation of a LTI IIR filter is the state-space representation [1]. It describes the evolution of the state vector $\boldsymbol{x}(k)$ from the previous step and the input, while a new output $\boldsymbol{y}(k)$ is computed out of the current state and the input:

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{1}$$

where $\boldsymbol{u}(k) \in \mathbb{R}^{q\times 1}$ is the input vector, $\boldsymbol{y}(k) \in \mathbb{R}^{p\times 1}$ the output vector, $\boldsymbol{x}(k) \in \mathbb{R}^{n\times 1}$ the state vector and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times q}$, $\boldsymbol{C} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{p\times q}$ are the state-space matrices of the system. Unlike a mathematical function, the output at time k depends not only on in the input at time k but also on the internal state of the filter (generally determined from the previous inputs and outputs). The system (**??**) is said to be stable iif its spectral radius is less than 1 [1].

There exist some methods of filter analysis on signal flow graphs. However, these methods have been developed for particular filter structures and are very difficult to be applied on an arbitrary LTI filter. Though the graph representation itself is not convenient for some mathematical approaches to be applied.

Therefore, we prefer an analytical representation. In our automatized filter-to-code generator we use an extension of the state-space representation, which will be described in the next section.

2.2 Fixed-Point arithmetic

The considered filters are supposed to be implemented using signed Fixed-Point (FxP) arithmetic with two's complement representation [22], [10].

Let t be a FxP number. It can be written as $t = -2^m t_m + \sum_{i=\ell}^{m-1} 2^i t_i$, where $t_i \in \mathbb{B} = \{0, 1\}$ is the i^{th} bit of t and the Fixed-Point Format FxPF (m, ℓ) gives the position of the *most* (MSB) significant and *least* (LSB) significant bits positions respectively, as shown in Fig. 2. The wordlength w is given by $w = m-\ell+1$.

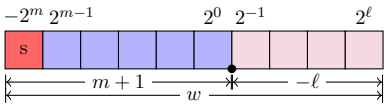


Figure 2: Fixed-point representation (here, $m = 5$ and $\ell = -4$).

The quantization step of the representation is 2^ℓ. Further in the report the LSB position ℓ sometimes will be substituted directly with $m - w + 1$.

The real number t is represented in machine by the integer T, such that $T = t \cdot 2^{-\ell}$, where $T \in [-2^{w-1}; 2^{w-1} - 1] \cap \mathbb{Z}$.

Let $\boldsymbol{y}(k) \in \mathbb{R}$ be an output of a digital filter and suppose that the wordlengths $\boldsymbol{w}_y$ are fixed. Then, determining the Fixed-Point Formats for $\boldsymbol{y}(k)$ means to find a MSB vector $\boldsymbol{m}_y$ such that for all k

$$\boldsymbol{y}_i(k) \in [-2^{\boldsymbol{m}_{y_i}}; 2^{\boldsymbol{m}_{y_i}} - 2^{\boldsymbol{m}_{y_i}-\boldsymbol{w}_{y_i}+1}]. \tag{2}$$

Obviously, we are interested in the least possible MSB, which guarantees (2): any greater MSB would yield to a larger quantization step, hence round-off noise, at constant wordlength.

It should be noted that we will follow the multiple wordlength paradigm, i.e. the formats for each element of the input and output vectors can be different. This permits to take into account different impact that different variables have on the filter behavior.

3 Specialized Implicit Framework

In this section we will state the basic notions of the Specialized Implicit Framework (SIF), which is a basic brick of the automatized filter-to-code generator. This framework permits to represent any LTI filter in a unified form and therefore to unify the analysis approach. Numerous quality measures have been adapted for this framework. In this section, however, only some sensitivity-based measures are reminded.

One of the contributions of the thesis is introducing an algorithm for the conversion of Lattice Wave Digital filters to SIF. It is a simple translation of the lattice wave structure description into analytical representation, which gave an insight of the Stages 1 and 2 of the filter generator. It yielded a deeper understanding of the filter quality measures and of further development of automatized structure choice approach. A fair comparison of lattice wave structures with several other popular realizations is provided as well. The conversion algorithm was implemented in Matlab and tested on random filters.

3.1 State of the Art

In order to encompass all the possible realizations for a given transfer function (direct forms, state-space, cascade or parallel decomposition, or some more exotic realizations like the ρDFIIt ones [23, 24]), the Specialized Implicit Framework (SIF) has been proposed in [12].

SIF is an extension of the state-space realization, modified in order to allow chained Sum-of-Products (SoP) operations. All the input-output relationships with delays, computation order, multiplications by constants and additions can be represented with the SIF.

This macroscopic description is more suited for the analysis than a graph relationship as it gives direct analytical formula for the finite precision error analysis [12].

Denote $\boldsymbol{u}(k)$ and $\boldsymbol{y}(k)$ the input and output respectively. Variables that are stored from one step to the other are in the state vector $\boldsymbol{x}(k)$, while intermediate results are collected in the vector $\boldsymbol{t}(k)$. Then, the SIF is the following system:

$$\mathcal{H}\begin{cases} \boldsymbol{J}\boldsymbol{t}(k+1) = & \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) = \boldsymbol{K}\boldsymbol{t}(k+1) + & \boldsymbol{P}\boldsymbol{x}(k) + \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) = \boldsymbol{L}\boldsymbol{t}(k+1) + & \boldsymbol{R}\boldsymbol{x}(k) + \boldsymbol{S}\boldsymbol{u}(k) \end{cases} \tag{3}$$

Note that $\boldsymbol{J}$ must be lower triangular with 1 on its diagonal, so the first value of $\boldsymbol{t}(k+1)$ is first computed, then the second one is computed using the first and so on (thus,the computation of $\boldsymbol{J}^{-1}$ is not necessary). The *implicit* term $\boldsymbol{J}\boldsymbol{t}(k+1)$ naturally serves for preservation of the computation order specific for each realization. Its transfer function $\boldsymbol{H}$ can be obtained analogously to the state-space realization from the matrices $\boldsymbol{J}, \boldsymbol{K}, \ldots, \boldsymbol{S}$.

Denote $\boldsymbol{Z}$ to be a set of the SIF coefficients:

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix}. \tag{4}$$

Various filter analysis measures [12, 25, 26] have been explicitly introduced for this framework along with rigorous error analysis algorithms. Some of the sensitivity measures are presented further in this section and are extensions of classical measures for the state-space structure [27]. These are called **a priori** measures, because they evaluate the sensitivity of a filter to potential quantization errors, i.e. before we know anything on its software implementation.

3.1.1 Sensitivity-based measures for SIF

Obviously, after implementation in finite-precision the coefficients $\boldsymbol{Z}$ will be modified into $\boldsymbol{Z}+\boldsymbol{\Delta Z}$, where the errors $\boldsymbol{\Delta Z}$ depend on the coefficients values and word-lengths, according to Fixed-Point formats. In order to evaluate how the filter characteristics *may* be modified by the quantization of the coefficients, sensitivity-based measures are usually used [27]. For simplicity, in this section we suppose that the filter is Single Input Single Output.

The sensitivity of the transfer function H with respect to the coefficients $\boldsymbol{Z}$ is given by $\frac{\partial H}{\partial \boldsymbol{Z}}$. Analytical form of this measure is usually developed specifically for each realization. However, once obtained for SIF [12], it can be applied to any realization. Unfortunately, this commonly used measure is not fair and does not reflect how the coefficients' quantization changed the transfer function H into $H+\Delta H$ since the absolute error of the coefficients may not all have the same magnitude order.

For that purpose, a *stochastic* sensitivity-based measure has been proposed and developed with respect to FxP considerations in [25]. Here, quantization error $\boldsymbol{\Delta Z}_{ij}$ of the coefficient $\boldsymbol{Z}_{ij}$ is considered as a random variable, uniformly distributed in $[-2^{\ell_{\boldsymbol{Z}_{ij}}-1}; 2^{\ell_{\boldsymbol{Z}_{ij}}-1}]$ where $\ell_{\boldsymbol{Z}_{ij}}$ is the LSB of the $\boldsymbol{Z}_{ij}$. Then, the error transfer function $\boldsymbol{\Delta H}$ can be seen as a transfer function with random variables as coefficients. From [25] and with first order approximation of $\boldsymbol{\Delta H}$ with the derivative $\frac{\partial H}{\partial \boldsymbol{Z}}$ its second-order moment can be evaluated with

$$\sigma^2_{\Delta H} = \left\| \frac{\partial H}{\partial \boldsymbol{Z}} \times \boldsymbol{\Xi} \right\|_2^2 \tag{5}$$

where

$$\boldsymbol{\Xi}_{ij} := \begin{cases} 0 & \text{if } \boldsymbol{Z}_{ij} \in \{0, \pm 1\} \\ \frac{2^{-\boldsymbol{w}_{\boldsymbol{Z}_{ij}}+1}}{\sqrt{3}} \lfloor \boldsymbol{Z}_{ij} \rfloor_2 & \text{otherwise} \end{cases} \tag{6}$$

and $\boldsymbol{w}_{\boldsymbol{Z}_{ij}}$ is the word-length fixed to represent $\boldsymbol{Z}_{ij}$ and $\lfloor x \rfloor_2$ denotes a nearest power of 2 smaller than x . Thus, $\sigma^2_{\Delta H}$ captures more information on the transfer function error.

If all the coefficients have the same word-length, then a normalized transfer function error measure is defined by

$$\bar{\sigma}^2_{\Delta H} \triangleq \left\| \frac{\partial H}{\partial \boldsymbol{Z}} \times \lfloor \boldsymbol{Z} \rfloor_2 \right\|_2^2. \tag{7}$$

This normalization is useful for a concrete realization analysis on the design stage, when word-lengths are not known.

The same approach was applied for the poles $\{\boldsymbol{\lambda}_i\}$ of the systems, or more interestingly to their moduli $|\boldsymbol{\lambda}_i|$ in order to ensure filter's stability after quantization. Analogously, the stochastic pole error $\sigma^2_{\Delta|\boldsymbol{\lambda}|}$ was introduced in [25]. It can be computed as

$$\sigma^2_{\Delta|\boldsymbol{\lambda}|} = \sum_{i=1}^{n} \left\| \frac{\partial |\boldsymbol{\lambda}_i|}{\partial \boldsymbol{Z}} \times \Xi \right\|_F^2 \tag{8}$$

Likewise to (7) the word-length independent normalized error pole measure $\bar{\sigma}^2_{\Delta|\boldsymbol{\lambda}|}$ may be computed.

Using these **a priori** measures, we can fairly compare various filter realizations of a user-given transfer function. Also, since some of the filter realizations can be optimized [28], these measures can be used as criteria.

3.2 Contribution: representing Lattice Wave Digital Filters with Specialized Implicit Framework

Lattice Wave Digital Filters (LWDF) is a class of recursive Wave Digital Filters that inherit several good properties, such as stability for implementation and possibility of suppression of parasitic oscillations. LWDF can be either derived from analog reference filters [7] or using explicit formulas [29].

The LWDF structure is highly modular and has a high degree of parallelism, which makes them suitable for a VLSI implementation. Their good stability qualities [7] make them good candidates for adaptive filtering

and Hilbert transformers design [30]. LWDFs are considered in numerous different applications, including studies on linear-phase structures [31], design of multiplierless LWDFs [32] and energy-efficient structures [33].

However, all studies on fixed-point implementation of lattice wave structures are performed **a posteriori**, *i.e.* when the implementation parameters are known [34]. These are commonly two- or three-step algorithms that propose coefficients quantization scheme based on solving optimization problems for infinite-precision filter models and then adjustment of finite-precision filter. These are specific approaches to the LWDF analysis and are not really suitable for a fair comparison with other structures.

Providing a LWDF to SIF conversion algorithm leads to several benefits. First of all, it permits us to fairly compare the lattice wave structures with various different realizations using unified measures. Secondly, a direct and reliable filter implementation approach, which does not use any filter simulations but is based on mathematical proofs, can be applied.

3.2.1 Lattice Wave structure to be converted

LWDF are represented by two parallel branches, which realize all-pass filters. These all-pass filters are composed of cascaded first- and second-order symmetric two-port adaptors. Its data-flow diagram (DFG) is shown on Figure 3.

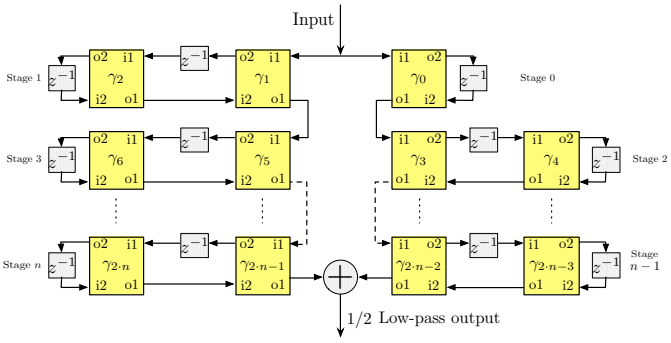


Figure 3: Data-flow diagram of a low-pass LWDF.

Each adaptor contains three adders and one multiplier. According to [29], the adaptor coefficients γ may be guaranteed to fall into the interval $-1 < \gamma < 1$. In [7] it was proposed to use Richards' structures for adaptors, which are shown on Figure 4. The interval $(1; -1)$ is divided into four sub-intervals, therefore there are four types of structures, in which the multiplication is performed by a $0 < \alpha \leq 1/2$.

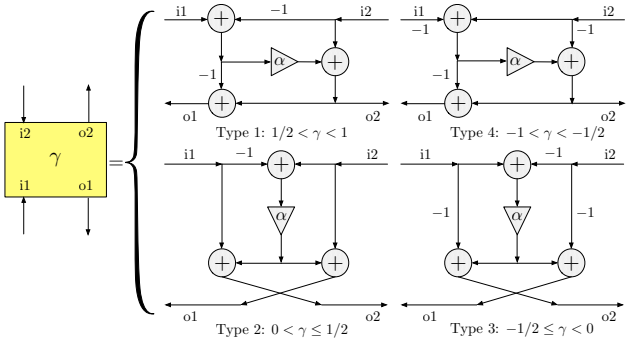


Figure 4: Two-port adaptor structures, for which actual multiplier α is computed out of γ.

It was shown in [7] that in order to make sure that only one passband and only one stop-band occur, the orders of the upper and lower branches must differ by one, such that the overall order n of the filter is odd.

The high-pass filter may be obtained simultaneously by changing the sign of the all-pass lower branch.

In [29] explicit formulas for LWDF transfer function design for several common filters such as Butterworth, Cauer (elliptic) and Chebyshev were presented. However, LWDF structure can realize all transfer functions that can be achieved by the families of reference filters considered so far, although not strictly all. It should be pointed out, that for common applications filter requirements are expressed not in terms of transfer functions but in terms of loss and phase delay that needs to be located in tolerance bounds

3.2.2 LWDF-to-SIF conversion algorithm

Due to the high modularity of the LWDF, the conversion between its DFG and SIF is not difficult. As seen on Figure 3, LWDF consists of two branches, and each branch is a cascade of stages. Each stage in its turn may be considered as a cascade of *subsystems* of two types, as shown on Figure 5.

(a) Type A (b) Type B

Figure 5: A stage is considered as cascade of subsystems of type A (Subfig. a) or type B (Subfig. b).

Therefore, the basic brick for cascade sequence is actually not a simple adaptor but a two-port adaptor with one output delayed (Type A: Figure 5a) and a 1-input/1-output adaptor with delay (Type B: Figure 5b). Then, given filter's coefficients γ, the conversion algorithm can be divided into following steps:

Step 1: Deduce SIF representation (matrix $\boldsymbol{Z}$) for each subsystem according to its γ value;

Step 2: Cascade subsystems into stages;

Step 3: Cascade stages into branches;

Step 4: Regroup two branches SIFs into final filter according to Figure 3.

The first step can be easily done by applying SIF notation to the subsystem DFG. Two subsystems per each adaptor structure must be considered, overall 8 basic bricks. For example, for $-1 < \gamma < 1/2$ the structures to be converted into SIF can be described with DFGs on Figure 6.

(a) (b)

Figure 6: (a) Type A subsystem with adaptor of type 4. (b) Type B subsystem with adaptor of type 4.

For instance, applying SIF definition (3), we can deduce that SIFs $\boldsymbol{Z}_A$ and $\boldsymbol{Z}_B$ for subsystems shown on Figure 6 are:

$$\boldsymbol{Z}_A \triangleq \left(\begin{array}{c|c|c} -\boldsymbol{J}_A & \boldsymbol{M}_A & \boldsymbol{N}_A \\ \hline \boldsymbol{K}_A & \boldsymbol{P}_A & \boldsymbol{Q}_A \\ \hline \boldsymbol{L}_A & \boldsymbol{R}_A & \boldsymbol{S}_A \end{array}\right) = \left(\begin{array}{cc|cc|cc} -1 & 0 & 0 & 1 & 1 \\ \alpha & -1 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 0 \\ \hline -1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \end{array}\right) \qquad \boldsymbol{Z}_B \triangleq \left(\begin{array}{c|c|c} -\boldsymbol{J}_B & \boldsymbol{M}_B & \boldsymbol{N}_B \\ \hline \boldsymbol{K}_B & \boldsymbol{P}_B & \boldsymbol{Q}_B \\ \hline \boldsymbol{L}_B & \boldsymbol{R}_B & \boldsymbol{S}_B \end{array}\right) = \left(\begin{array}{cc|cc} -1 & 0 & 1 & -1 \\ \alpha & -1 & 1 & 0 \\ \hline 0 & -1 & 0 & 0 \\ \hline -1 & 1 & 0 & 0 \end{array}\right) \tag{9}$$

However, cascading for the Step 2 requires a special approach, since it is not an output-to-input link but a "loop": type A subsystem H_A is a MIMO filter, which is cascaded with a SISO filter H_B. If a standard

cascading approach is used, the condition of matrix $\boldsymbol{J}$ being lower triangular is not fulfilled. We provided explicit formulas for the cascading of such SIFs and use topological sort [35] to re-triangularize the matrix $\boldsymbol{J}$. However, due to heaviness, the formulas are not presented in this report.

The sequential cascading for Step 3 is performed using explicit formulas from [13]. Notate, that even on the Step 1 the matrices $\boldsymbol{Z}_A$ and $\boldsymbol{Z}_B$ are sparse. Each cascading (on Step 2 and 3) would produce an even more sparse matrix.

3.2.3 Numerical results

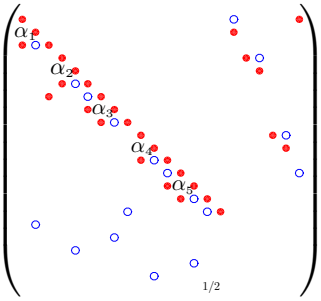
The above algorithm was implemented in Matlab, since the Matlab version of FWR[1] toolbox was used for the SIF analysis.

The following example is based on the LWDF coefficients, which were obtained with Lattice Wave Digital Filters[2] toolbox for Matlab. This toolbox is based on explicit formulas introduced in [29]. However, the LWDF-to-SIF conversion algorithm requires only the γs, therefore does not depend on software providing coefficients. So, any procedure producing the LWDF coefficients via [36] can be used as preliminary step.

A Matlab-generated low-pass 5^{th} order Butterworth filter with cutoff frequency 0.1 was considered. Four realizations of this transfer function were considered: LWDF, balanced state-space, Direct Form I and ρ Direct Form II transposed (ρDFIIt [24]). Only the normalized (*i.e.* input-width independent) versions of measures were used.

The result SIF for the considered LWDF structure is a sparse 22×22 matrix shown below. It has only three types of non-zero elements: adaptor coefficients α_i and plus/minus ones represented by filled and contour circles respectively:

$$\mathbf{Z} = \left(\begin{array}{c} \\ \end{array} \right.$$



As described in [23, 24], the ρDFIIt is parametrized by n extra parameters. It can be a subject of multi-criteria optimization. For this example a tradeoff function minimizing weighted sum of normalized transfer function and pole errors was used. Excellent results may be observed in Table 1.

However, Direct Form I showed to be very ill-conditioned for the considered filter, and its pole error cannot be computed. It is obvious, that implementing such a filter with this structure is not a good idea.

Thanks to the minimal number of coefficients, the LWDF approaches in its normalized transfer function error the optimized ρDFIIt. The specific alternating distribution of LWDF poles [34] leads to good results in pole error measure. Therefore, ρDFIIt may be a good alternative.

The state-space realization chosen is a balanced state-space. It is expectantly sensitive to quantization errors (it has much more coefficients).

To conclude, the conversion from LWDF to SIF permits to apply numerous classical and novel sensitivity measures upon any LWDF realization and any other. However, to provide a consistent implementation comparison a code optimization and generation chain FiPoGen [26] + FloPoCo[3] is required.

[1] `https://gforge.inria.fr/projects/fwrtoolbox/`
[2] `http://ens.ewi.tudelft.nl/~huib/mtbx/index.php`
[3] `http://flopoco.gforge.inria.fr/`

Realization	size(Z)	coeff.	$\bar{\sigma}^2_{\Delta H}$	$\bar{\sigma}^2_{\Delta\|\boldsymbol{\lambda}\|}$
LWDF	22×22	5	0. 3151	0.56
state-space	6× 6	36	1.15	5.75
ρDFIIt	11×11	11	0.09	0.45
DFI	12×12	11	1.42e+6	-

Table 1: Different realizations comparison.

4 Reliable Fixed-Point implementation of Digital Filters

Once the filter realization is chosen, we need to provide Fixed-Point code to compute the filter. Given implementation constraints, such as output error bound, and wordlength constraints for all variables we must choose Fixed-Point Formats for the implementation such that the output code**ensures that no overflow occurs.** It should be noted that different variables can have different wordlengths and, most important, different fixed-point formats according to the impact they have on the filter behavior, which is often neglected in currently used approaches.

A reasonable approach is to first determine the output interval of the filter and then determine the format within the wordlength constraints if possible. However, the propagation of rounding errors induced by finite-precision arithmetic must be taken into account.

In the signal processing error analysis is often performed using a statistic approach, where the coefficient quantification errors are considered as random "noise" with normal or uniform distribution [37]. Moreover the algorithms to compute the output interval of a filter give no guarantee on the result and do not take into account the filter computation error. Usually they consist in performing numerous (non-exhaustive) simulations via software tools such as Matlab. This approach does not give any guarantee on the non-overflow.

In order to compute the largest possible output interval for a filter, we use the result proposed in [16], based on a property of bounded-input bounded output systems [19, 38] where the largest possible peak value of the output is determined by the use of the Worst-Case Peak Gain (WCPG) measure. Error propagation analysis in LTI systems is directly dependent on the reliable evaluation of the WCPG.

This measure is computed as an infinite sum and contains large powers of a matrix, which makes the error analysis non-trivial. We provided a rigorous approach on the WCPG evaluation and implemented it as a C library.

Further, we investigated the error propagation through the filters and provided an algorithm for determining the Fixed-Point Formats with mathematically proven guarantee that no overflows occur. In our approach the computed MSB positions are either exact or overestimated by at most one bit. Therefore, we solved the basic brick problem for the Stage 3 of the filter generator work-flow.

The section is organized as follows. First, we state some previous results on determining the output interval of a filter and give the reader an idea how it can be used in LTI systems error analysis. Then, we present our algorithm for reliable determination of the WCPG measure. Finally, we perform the error analysis of a filter implementation in Fixed-Point and give a rigorous algorithm on determining the Fixed-Point formats.

4.1 State of the Art

Although in our filter generator the SIF representation is used for filters, in this chapter we provide results for the state-space realization, especially since the conversion between SIF and state-space is easy. Indeed, it can be established that the SIF equation (3) is equivalent in infinite precision to a state-space filter:

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k)+\boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k)+\boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{10}$$

with:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{P}, \quad & \boldsymbol{B} &= \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{N} + \boldsymbol{Q}, \\ \boldsymbol{C} &= \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{R}, \quad & \boldsymbol{D} &= \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{N} + \boldsymbol{S}. \end{aligned} \tag{11}$$

It should be remarked that in finite precision the equation (10) corresponds to a different set of coefficients than the one in (3). However, this issue can be resolved by taking into account the computational errors in (11).

In [16] a following theorem giving an output interval of a stable LTI filter was stated.

Proposition 1 (Worst-Case Peak Gain theorem [16])**.** *Let $\mathcal{H}$ be a state-space system. If an input $\{\boldsymbol{u}(k)\}_{k\geq 0}$ is known to be bounded by $\bar{\boldsymbol{u}}$ ($\forall k \geq 0, \quad |\boldsymbol{u}_i(k)| \leq \bar{\boldsymbol{u}}_i, \quad 1 \leq i \leq q$), then the output $\{\boldsymbol{y}(k)\}_{k\geq 0}$ will be bounded iff the spectral radius $\rho(\boldsymbol{A})$ is strictly less than 1. This property is known as the Bounded Input Bounded Output (BIBO) stability [1].*

Moreover, in that case, the output is (component-wise) bounded by

$$\forall k, \quad |\boldsymbol{y}(k)| \leq |\boldsymbol{W}\bar{\boldsymbol{u}}|\,, \tag{12}$$

where $\boldsymbol{W} \in \mathbb{R}^{p\times q}$ is the Worst-Case Peak Gain matrix [19] of the system. It can be computed as

$$\boldsymbol{W} := |\boldsymbol{D}| + \sum_{k=0}^{\infty} \left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right|. \tag{13}$$

This proposition can be completed when considering intervals for the input, instead of bounds (corresponding to symmetric intervals). In that case, the Worst-Case Peak Gain matrix indicates by how much the radius of the input interval is amplified on the output [16] (although this is not valid for the transient phase, i.e. for the few first steps). Moreover, this bound can be achieved (up to any small $\varepsilon > 0$).

However, even in that case, $\boldsymbol{W}\bar{\boldsymbol{u}}$ is a supremum we need to compute in order to deduce the output interval for a BIBO stable filter with bounded inputs. Using the WCPG to bound all the variables involved in the filter computation algorithm, we can determine their fixed-point representation. We will show further in this section that for the rigorous determination of Fixed-Point Formats an arbitrary precision computation algorithm for the WCPG is required.

Also, we can use WCPG to determine the impact on the output of the computational errors. Classical error analysis cannot be used in that context due to the feedback scheme of the computation (Interval Arithmetic or Affine Arithmetic do not provide tight bounds [39]).

Since the filter is linear, the implemented filter $\mathcal{H}^*$ can be seen as the exact filter $\mathcal{H}$ where the output is corrupted by the vector of errors $\boldsymbol{\varepsilon}(k)$ occurring at each sum of product through a given linear filter $\mathcal{H}_e$ (see Fig. 7).

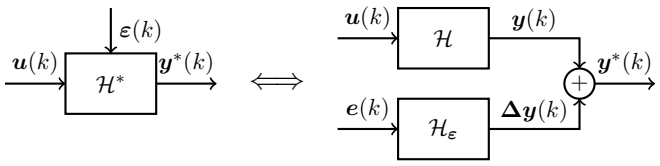


Figure 7: The implemented filter is equivalent to the exact filter where the output is corrupted by the computational errors passing themselves through a filter.

A State-space representation of $\mathcal{H}_e$ can be obtained analytically [16] and Proposition 1 can be used to determine the output error $\boldsymbol{\Delta y}$ due to finite-precision arithmetic, i.e. due to the roundoff errors $\varepsilon(k)$.

For all these reasons, the reliable computation of the Worst-Case Peak Gain matrix is a required step for the accurate error analysis of LTI systems in finite precision.

4.2 Contribution: Reliable evaluation of the Worst-Case Peak Gain measure

4.2.1 Algorithm for WCPG evaluation

Given a BIBO stable LTI filter in state-space realization (**??**) and ε, a desired absolute approximation error, we want to determine the Worst-Case Peak Gain matrix $\boldsymbol{W}$ of this filter, defined in (13). While computing such an approximation, various errors, such as truncation and summation errors, are made.

Instead of directly computing the infinite sum $\left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right|$ for any $k \geq 0$, we will use an approximate eigenvalue decomposition of $\boldsymbol{A}$ (i.e. $\boldsymbol{A} \approx \boldsymbol{V}\boldsymbol{T}\boldsymbol{V}^{-1}$) and compute the FP sum $\left|\boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$ for $0 \leq k \leq N$.

Our approach to compute the approximation $\boldsymbol{S}_N$ of $\boldsymbol{W}$ is summarized in algorithm 1 where all the operations ($\otimes$, $\oplus$, inv, abs, etc.) are FP multiple precision operations done at various precisions to be determined such that the overall error is less than ε:

$$|\boldsymbol{W} - \boldsymbol{S}_N| \leq \varepsilon. \tag{14}$$

The overall error analysis decomposes into 6 steps, where each one expresses the impact of a particular approximation (or truncation), and provides the accuracy requirements for the associated operations such that the result is rigorously bounded by ε.

Algorithm 1: Floating-point evaluation of the WCPG.

Input: $\boldsymbol{A} \in \mathbb{F}^{n\times n}, \boldsymbol{B} \in \mathbb{F}^{n\times q}, \boldsymbol{C} \in \mathbb{F}^{p\times n}, \boldsymbol{D} \in \mathbb{F}^{p\times q}, \varepsilon > 0$
Output: $\boldsymbol{S}_N \in \mathbb{F}^{p\times q}$
Step 1: Compute N
Step 2: Compute $\boldsymbol{V}$ from an eigendecomposition of $\boldsymbol{A}$
$\boldsymbol{T} \leftarrow \mathrm{inv}(\boldsymbol{V}) \otimes \boldsymbol{A} \otimes \boldsymbol{V}$
if $\|\boldsymbol{T}\|_2 > 1$ **then return** $\perp$
Step 3: $\boldsymbol{B}' \leftarrow \mathrm{inv}(\boldsymbol{V}) \otimes \boldsymbol{B}$
$\boldsymbol{C}' \leftarrow \boldsymbol{C} \otimes \boldsymbol{V}$
$\boldsymbol{S}_{-1} \leftarrow |\boldsymbol{D}|, \quad \boldsymbol{P}_{-1} \leftarrow \boldsymbol{I}_n$
for k **from** 0 **to** N **do**
Step 4: $\quad \boldsymbol{P}_k \leftarrow \boldsymbol{T} \otimes \boldsymbol{P}_{k-1}$
Step 5: $\quad \boldsymbol{L}_k \leftarrow \boldsymbol{C}' \otimes \boldsymbol{P}_k \otimes \boldsymbol{B}'$
Step 6: $\quad \boldsymbol{S}_k \leftarrow \boldsymbol{S}_{k-1} \oplus \mathrm{abs}(\boldsymbol{L}_k)$
return $\boldsymbol{S}_N$

Step 1: Let $\boldsymbol{W}_N$ be the truncated sum

$$\boldsymbol{W}_N := \sum_{k=0}^{N} \left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right| + |\boldsymbol{D}|\,. \tag{15}$$

We compute a truncation order N of the infinite sum $\boldsymbol{W}$ such that the truncation error is less than $\varepsilon_1 > 0$:

$$|\boldsymbol{W} - \boldsymbol{W}_N| \leq \varepsilon_1. \tag{16}$$

The idea of the algorithm is described further in this section.

Step 2: Error analysis for computing the powers $\boldsymbol{A}^k$ of a full matrix $\boldsymbol{A}$, when the k reaches several hundreds, is a significant problem, especially when the norm of $\boldsymbol{A}$ is larger than 1 and its eigenvalues are close to 1. However, if $\boldsymbol{A}$ may be represented as $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1}$ with $\boldsymbol{E} \in \mathbb{Z}^{n\times n}$ strictly diagonal and $\boldsymbol{X} \in \mathbb{Z}^{n\times n}$, then powering of $\boldsymbol{A}$ reduces to powering the diagonal matrix $\boldsymbol{E}$, which is more convenient.

Suppose we have a matrix $\boldsymbol{V}$ approximating $\boldsymbol{X}$. We require this approximation to be just quite accurate so that we are able to discern the different associated eigenvalues and be sure their absolute values are less than 1.

We may then consider the matrix $\boldsymbol{V}$ to be exact and compute an approximation $\boldsymbol{T}$ to $\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}$ with sufficient accuracy such that the error of computing $\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}$ instead of matrix $\boldsymbol{A}^k$ is less than $\varepsilon_2 > 0$:

$$\left| \boldsymbol{W}_N - \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right| \right| \leq \varepsilon_2. \tag{17}$$

We require for matrix $\boldsymbol{T}$ to satisfy $\|\boldsymbol{T}\|_2 \leq 1$. This condition is stronger than $\rho(\boldsymbol{A}) < 1$, it naturally means that there exist some margin for computational errors between the spectral radius and 1. We provide an algorithm to test it using an analogue of Gershgorin's circle theorem [40] for the eigenvalues.

Step 3: We compute approximations $\boldsymbol{B'}$ and $\boldsymbol{C'}$ of $\boldsymbol{V}^{-1}\boldsymbol{B}$ and $\boldsymbol{C}\boldsymbol{V}$, respectively. We require that the propagated error committed in using $\boldsymbol{B'}$ instead of $\boldsymbol{V}^{-1}\boldsymbol{B}$ and $\boldsymbol{C'}$ instead of $\boldsymbol{C}\boldsymbol{V}$ be less than $\varepsilon_3 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right| - \sum_{k=0}^{N} \left| \boldsymbol{C'}\boldsymbol{T}^k\boldsymbol{B'} \right| \right| \leq \varepsilon_3. \tag{18}$$

Step 4: We compute in $\boldsymbol{P}_k$ the powers $\boldsymbol{T}^k$ of $\boldsymbol{T}$ with a certain accuracy. It is required that the error be less than $\varepsilon_4 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C'}\boldsymbol{T}^k\boldsymbol{B'} \right| - \sum_{k=0}^{N} \left| \boldsymbol{C'}\boldsymbol{P}_k\boldsymbol{B'} \right| \right| \leq \varepsilon_4. \tag{19}$$

Step 5: We compute in $\boldsymbol{L}_k$ each summand $\boldsymbol{C'}\boldsymbol{P}_k\boldsymbol{B'}$ with a error small enough such that the overall approximation error induced by this step is less than $\varepsilon_5 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C'}\boldsymbol{P}_k\boldsymbol{B'} \right| - \sum_{k=0}^{N} \left| \boldsymbol{L}_k \right| \right| \leq \varepsilon_5. \tag{20}$$

Step 6: Finally, we sum $\boldsymbol{L}_k$ in $\boldsymbol{S}_N$ with enough precision so that the absolute error bound for summation is bounded by $\varepsilon_6 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{L}_k \right| - \boldsymbol{S}_N \right| \leq \varepsilon_6. \tag{21}$$

By ensuring that each step verifies its bound ε_i, and taking $\varepsilon_i = \frac{1}{6}\varepsilon$, we get $\varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4 + \varepsilon_5 + \varepsilon_6 \leq \varepsilon$, hence (14) will be satisfied if inequalities (16) to (21) are.

Our approach hence determines first a truncation order N and then performs summation up to that truncation error, whilst adjusting precision in the different summation steps. A competing approach would be not to start with truncation order determination but to immediately go for summation and to stop when adding more terms does not improve accuracy. However, such an approach would not allow the final error to be bounded in an a priori way. Moreover, the multiple precision FP summation needs to know a bound on the number of terms to be summed, beforehand.

Truncation order In [19] Balakrishnan and Boyd propose "simple" lower and upper bounds on N. However, they describe their algorithm in terms of exact arithmetic, *i.e.* do not propose any error analysis. Their iterative approach has several difficulties which make it unusable in finite-precision algorithms.

We have deduced a direct formula for the truncation order N such that the bound (16) is satisfied. Our algorithm ensures that we do not underestimate the truncation bound. However, it includes solving an instance of an eigendecomposition problem and solving a linear system of equations as well.

There exist many FP libraries, such as LAPACK[4], providing functions for an eigendecomposition and to solve linear systems of equations. They usually deliver good and fast approximations to the solution of a given

[4] `http://www.netlib.org/lapack/`

	Example 1			Example 2			Example 3			Example 4
sizes n, p and q	$n=10,\ p=11,\ q=1$			$n=12,\ p=1,\ q=25$			$n=60,\ p=28,\ q=14$			$n=3,\ p=1,\ q=4$
$1-\rho(\boldsymbol{A})$	1.39×10^{-2}			8.65×10^{-3}			1.46×10^{-2}			2^{-60}
$\max(\boldsymbol{S}_N)$	3.88×10^{1}			5.50×10^{9}			2.64×10^{2}			-
$\min(\boldsymbol{S}_N)$	1.29×10^{0}			1.0×10^{0}			1.82×10^{1}			-
ε	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}	2^{-5}
N	220	2153	29182	308	4141	47811	510	1749	27485	-
Inversion iterations	0	2	4	2	3	5	1	2	4	-
overall max precision (bits)	212	293	1401	254	355	1459	232	306	1416	-
Overall execution time (sec)	0.11	1.53	60.06	0.85	11.54	473.20	45.62	177.90	9376.86	0.00...

Table 2: Numerical results for 3 real-world and 1 constructed example

numerical problem but there is neither verification nor guarantee about the accuracy of that approximation. An approach to obtain trusted error bounds is to use Interval Arithmetic [41]. In Interval Arithmetic real numbers are represented as sets of reals with addition, subtraction, multiplication and division defined [41]. The Theory of Verified Inclusions [42], [43] ,[44], [45] is a set of algorithms computing guaranteed bounds on solutions of various numerical problems, developed by S. Rump [42]. The verification process is performed by means of checking an interval fixed point, i.e. it is verified that the result interval contains an exact solution of given numerical problem.

For these reasons we propose to combine LAPACK FP arithmetic with Interval Arithmetic enhanced with the Theory of Verified Inclusions in order to obtain trusted intervals on the eigensystem and, eventually, a rigorous bound on N without significant impact on algorithm performance, since this computation is done only once. In addition, if the spectral radius of $\boldsymbol{A}$ cannot be shown less than 1, we stop the algorithm.

4.2.2 Numerical examples

The algorithms discussed above were implemented in `C`, using GNU MPFR version 3.1.12, GNU MPFI[5] version 1.5.1 and CLAPACK[6] version 3.2.1. Our implementation was tested on several real-life and random examples:

- The first example comes from Control Theory: the LTI system is extracted from an active controller of vehicle longitudinal oscillation [46], and WCPG matrix is used to determine the fixed-point arithmetic scaling of state and output.

- The second is a 12^{th}-order Butterworth filter, described in ρ-Direct Form II transposed [24] (a particular algorithm, with low complexity and high robustness to quantization and computational errors), where the errors-to-output LTI system $\mathcal{H}_e$ is considered (see Figure 7).

- The third one is a large random BIBO stable filter (obtained from the `drss` command of Matlab), with 60 states, 14 inputs and 28 outputs.

- The last one is built with a companion matrix $\boldsymbol{A}$ with spectral radius equal to $1-2^{-60}$.

Experiments were done on a laptop computer with an Intel Core i5 processor running at 2.8 GHz and 16 GB of RAM.

The numerical results detailed in Table 2 show that our algorithm for Worst-Case Peak Gain matrix evaluation with a priori error bound exhibits reasonable performance on typical examples. Even when the a priori error bound is pushed to compute WCPG results with an accuracy way beyond double precision, the algorithm succeeds in computing a result, even though execution time grows pretty high.

Our algorithm includes checks testing that certain properties of matrices are verified, in particular that $\rho(\boldsymbol{A})<1$ and $\|\boldsymbol{T}\|_2\leq 1$. Our Example 4, not taken from a real-word application but constructed on purpose, shows that the algorithm correctly detects that the conditions are not fulfilled for that example.

[5] `https://gforge.inria.fr/projects/mpfi/`

[6] `http://www.netlib.org/clapack/`

4.3 Contribution: rigorous approach on determining Fixed-Point Formats for a digital filter implementation

Once a reliable implementation of the WCPG evaluation is obtained, we can apply it to bound all the variables of the filter and use it to determine the impact of rounding errors on the output of implemented filter.

4.3.1 Problem statement

The problem of determining the FxPF for a filter $\mathcal{H}$ implementation can be formulated as follows. Let $\mathcal{H}$ be a filter in a state-space representation (**??**). Suppose all the inputs to be in an interval bounded by $\bar{\boldsymbol{u}}$.

Given the wordlength constraints vector $\boldsymbol{w}_x$ for the state and $\boldsymbol{w}_y$ for the output variables we look for a FxPF for $\boldsymbol{x}(k)$ and $\boldsymbol{y}(k)$ such that for any possible input no overflow occurs. We wish to determine the least MSB vectors $\boldsymbol{m}_y$ and $\boldsymbol{m}_x$ such that

$$\forall k, \quad \boldsymbol{y}(k) \in [-2^{-\boldsymbol{m}_y}; 2^{\boldsymbol{m}_y} - 2^{\boldsymbol{m}_y - \boldsymbol{w}_y + 1}], \tag{22}$$

$$\forall k, \quad \boldsymbol{x}(k) \in [-2^{-\boldsymbol{m}_x}; 2^{\boldsymbol{m}_x} - 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1}]. \tag{23}$$

Remark 1. *Since the filter $\mathcal{H}$ is linear and input interval is centered at zero, the output interval is also centered in zero. Therefore, further we will seek to determine the least $\boldsymbol{m}_y$ and $\boldsymbol{m}_x$ such that*

$$\forall k, \quad |\boldsymbol{y}(k)| \leq 2^{\boldsymbol{m}_y} - 2^{\boldsymbol{m}_y - \boldsymbol{w}_y + 1}, \tag{24}$$

$$\forall k, \quad |\boldsymbol{x}(k)| \leq 2^{\boldsymbol{m}_x} - 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1}. \tag{25}$$

4.3.2 Applying the WCPG to compute MSB positions

Applying the WCPG theorem on the filter $\mathcal{H}$ yields a bound on the output interval:

$$|\boldsymbol{y}_i(k)| \leq \left(\langle\langle \mathcal{H} \rangle\rangle \, \bar{\boldsymbol{u}}\right)_i, \quad i = 1, \ldots, p. \tag{26}$$

Denote the bound vector $\bar{\boldsymbol{y}} := \langle\langle \mathcal{H} \rangle\rangle \, \bar{\boldsymbol{u}}$.

We can determine the FxPF for the output of a LTI filter $\mathcal{H}$ utilizing the following lemma.

Lemma 1. *Let $\mathcal{H} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ be a BIBO-stable MIMO LTI filter and $\bar{\boldsymbol{u}}$ be a bound on the input interval. Suppose the wordlengths $\boldsymbol{w}_y$ are known and $\boldsymbol{w}_{y_i} > 1$, $i = 1, \ldots, p$.*

If for $i = 1, \ldots, p$ the MSBs are computed with

$$\boldsymbol{m}_{y_i} = \left\lceil \log_2(\bar{\boldsymbol{y}}_i) - \log_2\left(1 - 2^{1 - \boldsymbol{w}_{y_i}}\right) \right\rceil \tag{27}$$

and the LSBs are computed with $\boldsymbol{\ell}_{y_i} = \boldsymbol{m}_{y_i} + 1 - \boldsymbol{w}_{y_i}$, then for all k $|\boldsymbol{y}_i(k)| \leq 2^{\boldsymbol{m}_{y_i}} - 2^{\boldsymbol{m}_{y_i} - \boldsymbol{w}_{y_i} + 1}$ and $\boldsymbol{m}_{y_i}$ is the least.

Using Lemma 1 we can determine the FxPF for the output of a filter. In order to determine the FxPF for the state variable we modify the filter $\mathcal{H}$ by vertically concatenating the state vector and the output and include necessary changes into the state matrices.

Denote vector $\boldsymbol{\zeta}(k) := \begin{pmatrix} \boldsymbol{x}(k) \\ \boldsymbol{y}(k) \end{pmatrix}$ to be the new output vector. Then the state-space relationship (**??**) takes the form:

$$\mathcal{H}_\zeta \left\{ \begin{array}{rcl} \boldsymbol{x}(k+1) = & \boldsymbol{A}\boldsymbol{x}(k) & + \quad \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{\zeta}(k) = & \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix} \boldsymbol{x}(k) & + \quad \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{u}(k) \end{array} \right. . \tag{28}$$

Hence the problem is to find the least MSB vector $\boldsymbol{m}_\zeta$ such that (element-by-element)

$$\forall k, \quad |\boldsymbol{\zeta}(k)| \leq 2^{\boldsymbol{m}_\zeta} - 2^{\boldsymbol{m}_\zeta - \boldsymbol{w}_\zeta + 1}. \tag{29}$$

Now, applying the WCPG theorem on the filter $\mathcal{H}_\zeta$ and using Lemma 1, we can deduce the MSB positions of the state and output vectors for an implementation of the filter $\mathcal{H}$.

4.3.3 Taking rounding errors into account

However, due to the finite-precision degradation what we actually compute is not the exact filter $\mathcal{H}_\zeta$ but an implemented filter $\mathcal{H}_\zeta^\Diamond$:

$$\mathcal{H}_\zeta^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) = \Diamond_{\boldsymbol{\ell}_x}\left(\boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{B}\boldsymbol{u}(k)\right) \\ \boldsymbol{\zeta}^\Diamond(k) = \Diamond_{\boldsymbol{\ell}_\zeta}\left(\begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{C}\end{pmatrix}\boldsymbol{x}^\Diamond(k) + \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{D}\end{pmatrix}\boldsymbol{u}(k)\right) \end{cases} \tag{30}$$

where the Sums-of-Products (accumulation of scalar products on the right side) are computed with some rounding operator $\Diamond_\ell$. Suppose, this operator ensures faithful rounding [47]:

$$|\Diamond_\ell(x) - x| < 2^\ell \tag{31}$$

In [**?**, **?**] it was shown that such an operator can be implemented using some extra guard bits for the accumulation.

Denote the errors due to operator $\Diamond_\ell$ as $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ for the state and output vectors, respectively. Essentially, the vectors $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ may be associated with the noise which is induced by the filter implementation. Then the implemented filter can be rewritten as

$$\mathcal{H}_\zeta^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) = \boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{B}\boldsymbol{u}(k) + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\zeta}^\Diamond(k) = \begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{C}\end{pmatrix}\boldsymbol{x}^\Diamond(k) + \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{D}\end{pmatrix}\boldsymbol{u}(k) + \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{I}\end{pmatrix}\boldsymbol{\varepsilon}_y(k) \end{cases}, \tag{32}$$

where

$$|\boldsymbol{\varepsilon}_x(k)| < 2^{\boldsymbol{\ell}_x}, \quad |\boldsymbol{\varepsilon}_y(k)| < 2^{\boldsymbol{\ell}_y}.$$

It should be remarked that since the operator $\Diamond_l$ is applied $\boldsymbol{\varepsilon}_x(k) \neq \boldsymbol{x}(k) - \boldsymbol{x}^\Diamond(k)$ and $\boldsymbol{\varepsilon}_y(k) \neq \boldsymbol{y}(k) - \boldsymbol{y}^\Diamond(k)$. As the rounding also affects the filter state, the $\boldsymbol{x}^\Diamond(k)$ drifts away from $\boldsymbol{x}(k)$ over time, whereas with $\boldsymbol{\varepsilon}_x(k)$ we consider the error due to one step only.

It can be observed that at each instance of time the state and output vectors are computed out of $\boldsymbol{u}(k)$ and error-vectors, which can be considered as inputs as well. Thanks to the linearity of the filters, we can decompose the actually implemented filter into a sum of the exact filter and an "error-filter" $\mathcal{H}_\Delta$ as shown in Fig. 8. Note that this "error-filter" is an artificial one; it is not required to be "implemented" by itself and serves exclusively for error-analysis purposes.

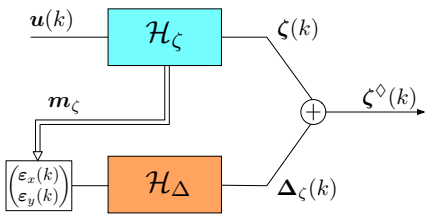


Figure 8: Implemented filter decomposition.

The filter $\mathcal{H}_\Delta$ is obtained by computing the difference between $\mathcal{H}_\zeta^\Diamond$ and $\mathcal{H}_\zeta$. This filter takes the rounding errors $\boldsymbol{\varepsilon}(k) := \begin{pmatrix}\boldsymbol{\varepsilon}_x(k)\\ \boldsymbol{\varepsilon}_y(k)\end{pmatrix}$ as input and returns the result of their propagation through the filter:

$$\mathcal{H}_\Delta \begin{cases} \boldsymbol{\Delta}_x(k+1) = \boldsymbol{A}\boldsymbol{\Delta}_x(k) + \begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{0}\end{pmatrix}\boldsymbol{\varepsilon}(k) \\ \boldsymbol{\Delta}_\zeta(k) = \begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{C}\end{pmatrix}\boldsymbol{\Delta}_x(k) + \begin{pmatrix}\boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{pmatrix}\boldsymbol{\varepsilon}(k) \end{cases}, \tag{33}$$

where, the vector $\boldsymbol{\varepsilon}(k)$ is guaranteed to be in the interval bounded by $\bar{\boldsymbol{\varepsilon}} := 2^{\boldsymbol{\ell}_\zeta}$.

Once the decomposition is done, we can apply the WCPG theorem on the "error-filter" $\mathcal{H}_\Delta$ and deduce the output interval of the computational errors propagated through filter:

$$\forall k, \quad |\boldsymbol{\Delta}_\zeta(k)| \leq \langle\langle \mathcal{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}}. \tag{34}$$

Hence, the output of the implemented filter is bounded with

$$\left|\boldsymbol{\zeta}^\Diamond(k)\right| \leq |\boldsymbol{\zeta}(k)| + |\boldsymbol{\Delta}_\zeta(k)|\,. \tag{35}$$

Applying Lemma 1 on the implemented filter and using (35) we obtain that the MSB vector $\boldsymbol{m}_\zeta^\Diamond$ can be upper bounded by

$$\begin{aligned} \boldsymbol{m}_{\zeta_i}^\Diamond = \big\lceil \log_2 \big((\langle\langle \mathcal{H}_\zeta \rangle\rangle \cdot \bar{\boldsymbol{u}})_i + (\langle\langle \mathcal{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}})_i \big) \\ - \log_2 \big(1 - 2^{1-\boldsymbol{w}_{\zeta_i}}\big) \big\rceil\,. \end{aligned} \tag{36}$$

Therefore, the FxPF $(\boldsymbol{m}_\zeta^\Diamond, \boldsymbol{\ell}_\zeta^\Diamond)$ guarantee that no overflows occur for the implemented filter.

Since the input of the error filter $\mathcal{H}_\Delta$ depends on the FxPF chosen for implementation, we cannot directly use (36). The idea is to first compute the FxPF of the variables in the exact filter $\mathcal{H}$, where computational errors are not taken into account, and use it as an initial guess for implemented filter $\mathcal{H}_\zeta^\Diamond$. Hence, we obtain the following two-step algorithm:

Step 1: Determine the FxPF $(\boldsymbol{m}_\zeta, \boldsymbol{\ell}_\zeta)$ for the exact filter $\mathcal{H}_\zeta$

Step 2: Construct the "error-filter" $\mathcal{H}_\Delta$, which gives the propagation of the computational errors induced by format $(\boldsymbol{m}_\zeta, \boldsymbol{\ell}_\zeta)$; then, compute the FxPF $(\boldsymbol{m}_\zeta^\Diamond, \boldsymbol{\ell}_\zeta^\Diamond)$ of the actually implemented filter $\mathcal{H}_\zeta^\Diamond$ using (36).

The above algorithm takes into account the filter implementation errors. However, the algorithm itself is implemented in finite-precision and can suffer from rounding errors, which influence the output result. All operations in the MSB computation will induce errors, so what we actually compute are only floating-point approximations $\widehat{\boldsymbol{m}_\zeta}$ and $\widehat{\boldsymbol{m}_\zeta}^\Diamond$.

We proposed error-analysis of the floating-point evaluation of the MSB positions via Lemma (1).

Both (27) and (36) contain the WCPG evaluations. Using the result obtained in [20], we can compute them with arbitrary precision. By controlling the accuracy of the WCPG matrices we proved that we can achieve for the approximations on the MSB positions to be off by at most one (to be either an exact MSB or to overestimate by one bit). The proofs are not presented in the current report but can be found in [21].

4.3.4 Complete algorithm

The two-step algorithm, presented in above takes into account accumulation of computational errors in a filter over time and we provided an error-analysis of the MSB position computation procedure. However, one additional fact has not been taken into account.

In most cases the MSB vectors $\widehat{\boldsymbol{m}}_\zeta$ (computed on Step 1) and $\widehat{\boldsymbol{m}}_\zeta^\Diamond$ (computed on Step 2) are the same. However, in some cases they are not, which can happen due to one of the following reasons:

- the accumulated rounding error due to the FxPF $(\widehat{\boldsymbol{m}}_\zeta, \widehat{\boldsymbol{\ell}}_\zeta)$ makes output of the actually implemented filter pass over to the next binade; or
- the floating-point approximation $\widehat{\boldsymbol{m}}_\zeta^\Diamond$ is off by one.

Moreover, if the MSB position after Step 2 of the algorithm is increased, the LSB position moves along and increases the error. Therefore, the modified format must be re-checked. Hence, the FxPF determination algorithm gets transformed into the following three-step procedure:

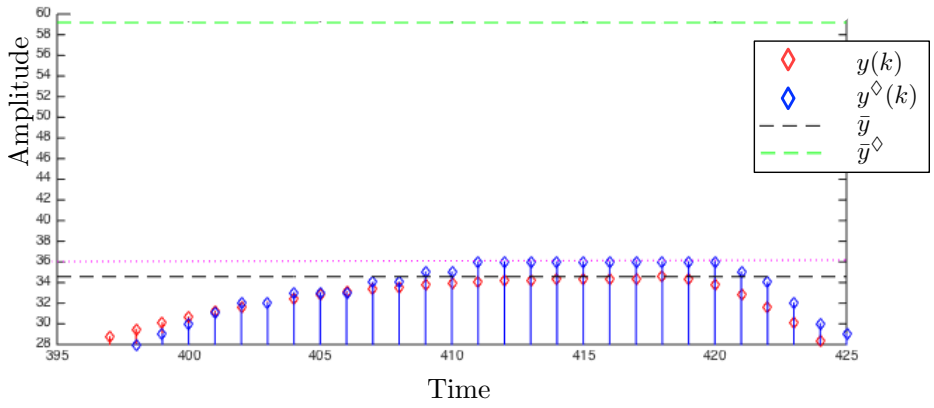


Figure 9: The exact and quantized outputs of the example. Quantized output does not pass over to the next binade.

Step 1: Determine the FxPF $(\widehat{\boldsymbol{m}}_\zeta, \widehat{\boldsymbol{\ell}}_\zeta)$ for the exact filter $\mathcal{H}_\zeta$;

Step 2: Construct the "error-filter" $\mathcal{H}_\Delta$, which shows the propagation of the computational errors induced by format $(\widehat{\boldsymbol{m}}_\zeta, \widehat{\boldsymbol{\ell}}_\zeta)$; then, compute the FxPF $(\widehat{\boldsymbol{m}}^{\Diamond}_\zeta, \widehat{\boldsymbol{\ell}}^{\Diamond}_\zeta)$ of the actually implemented filter $\mathcal{H}^{\Diamond}_\zeta$;

Step 3: If $\widehat{\boldsymbol{m}}^{\Diamond}_{\zeta_i} == \widehat{\boldsymbol{m}}_{\zeta_i}$, then return $(\widehat{\boldsymbol{m}}^{\Diamond}_\zeta, \widehat{\boldsymbol{\ell}}^{\Diamond}_\zeta)$;
otherwise $\widehat{\boldsymbol{m}}_{\zeta_i} \longleftarrow \widehat{\boldsymbol{m}}_{\zeta_i} + 1$ and go to Step 2.

4.3.5 Numerical results

The above described algorithm was implemented as a C library, using GNU MPFR[7] [48] version 3.1.12, GNU MPFI[8] version 1.5.1 and the WCPG library [18].

Consider following example: let $\mathcal{H}$ be a random stable filter with 3 states, 1 input and 1 output. Suppose the inputs are in an interval absolutely bounded by $\bar{\boldsymbol{u}} = 5.125$. In this example we set all the wordlength constraints to 7 bits, however our library supports the multiple wordlength paradigm.

The results of the work of our algorithm can be observed in Table 3. On the Step 1 we deduce the MSB positions for the filter $\mathcal{H}$, which does not take computational errors into account. These MSBs serve as an initial guess for the Step 2. We compute the vector $\bar{\boldsymbol{\varepsilon}}_\zeta$ and construct the "error-filter" $\mathcal{H}_\Delta$. Taking into account the error propagation yields changes in the MSB positions: the rounding errors force the third state $\boldsymbol{x}_3(k)$ to pass over to the next binade. However, moving the MSB yields larger quantization step and an addition step is required. Step 3 verifies that the FxPF deduced on the Step 2 guarantees no overflow.

We can verify the result by tracing the state and output vectors in two cases: the exact vectors $\boldsymbol{y}(k)$ and $\boldsymbol{x}(k)$; and the quantized vectors $\boldsymbol{y}^{\Diamond}(k)$ and $\boldsymbol{x}^{\Diamond}(k)$. The quantized vectors are the result of an implementation of the filter $\mathcal{H}$ with FxPF deduced on the Step 1. For each state and output we take an input sequence which makes the state respectively output attain the theoretical bound. However, this sequence is not guaranteed to maximize the quantized error.

In Fig. 9 we can observe that the exact vector $\boldsymbol{y}(k)$ attains the bound computed with the WCPG theorem. The quantized output stays within the bound $\bar{\boldsymbol{y}}^{\Diamond}$ for the implemented filter but passes over the bound of the exact filter. However, as the bound $\bar{\boldsymbol{y}}$ is far from the next binade, the MSB position is not changed and the computational errors do not result in overflow.

For the third state $\boldsymbol{x}_3(k)$, its bound $\bar{\boldsymbol{x}}_3$ is very close to the $2^{\boldsymbol{m}_{x_3}} - 2^{\boldsymbol{\ell}_{x_3}}$ and taking into account the filter computational errors yields passing to the next binade. It can be clearly observed in Fig. 10 that the

[7] `http://www.mpfr.org/`
[8] `https://gforge.inria.fr/projects/mpfi/`

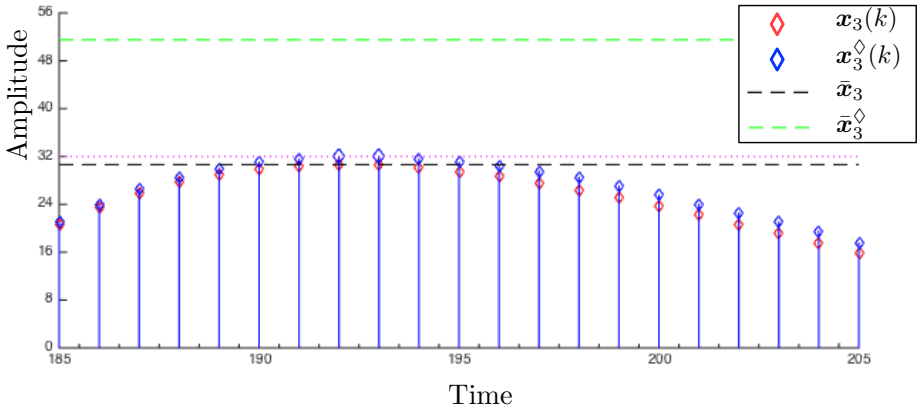


Figure 10: The exact and quantized third state of the example. Quantized output passes over to the next binade.

quantized state passes over the value $2^{m_{x_3}} - 2^{\ell_{x_3}}$. If the computational errors were not taken into account, the initial format (deduced on the Step 1) would result in overflow.

Therefore, we observed that deducing the FxPF without taking into account the computational errors can result in overflow but our approach guarantees a reliable implementation without overestimation.

	states			output
	$\boldsymbol{x}_1(k)$	$\boldsymbol{x}_2(k)$	$\boldsymbol{x}_3(k)$	$\boldsymbol{y}(k)$
Step 1	6	7	5	6
Step 2	6	7	6	6
Step 3	6	7	6	6

Table 3: Evolution of the MSB positions vector through algorithm iterations

5 Perspectives

There are a lot of open questions concerning filter generator. The problem to be addressed in the first place is the off-by-one problem for the MSB computation. The potential overestimation by one bit comes from the implemented filter decomposition as seen on the Fig. 8. When we represent the implemented output as a sum of the exact output and the output of the error-filter, we make a pessimistic assumption. In order to compute the output interval of the implemented filter, the WCPG theorem is applied *separately* on the exact and error filters. An interpretation of this approach is that the input sequence resulting in the largest peak value of the output of the exact filter generates the largest errors as well. However, this is not generally true. he connection between the input interval and rounding errors is known, but its non-linear dependence is still to be investigated.

However, the off-by-one problem can still be dealt with. It can be interpreted as an instance of the Table Maker's Dilemma [47]: the interval approximation on the MSB position contains a power of 2 and we do not know on which side of it the exact value lies. The first approach would be iteratively reducing the interval approximation and performing an adaptation of Ziv's test [47]. However, such an approach does not guarantee an answer in reasonable timings and may require large resources. Another approach can be re-formulating the problem into an integer linear programming problem and combining it with LLL lattice base reduction algorithm [49]. I am very interested in applying the lattice-based approach for the fixed-point error analysis.

One more interesting approach would be investigating the reachability and controllability problem for the discrete-time control systems. Testing if a particular system state is reachable from the initial zero condition can probably be done with the LLL base reduction approach or using a satisfiability modulo theories (SMT) solver [50].

A measure, which can potentially help in understanding the behavior of rounding errors through filter is the probability distribution function (PDF) of the output of the filter. It can be shown that there is a strong connection between the PDF and the WCPG. The distribution of the output of an implemented system, when the computational errors are taken into account, can help to refine the output interval of the implemented filter and therefore solve the off-by-one problem. The PDF can also be exploited in the signal processing applications that accept a probability of overflow, for example in telecommunications, where a small interference of a signal is tolerated.

Apart from the questions of the rounding errors through filter propagation, there is another interesting problem, which concerns all generator flow. As it was mentioned before, all the steps of the filter implementation can be plugged in various optimization routines. In my opinion, it is important to provide a thorough description of the objective functions and also to investigate the of existence of the solutions.

I find challenging and interesting the problem of coefficient quantization during the design of the transfer function out of filter specifications. This problem is similar to rational polynomial approximation, which is not trivial. In the case of finite impulse response filters, a robust design approach using the Parks-McCellan algorithm has been obtained by S. Filip [51]. The same author is currently developing a similar approach for generation of the transfer function of infinite impulse response systems. This approach should be extended to the SIF formalism and, therefore, to be applied to any filter structure and not only to direct forms (those forms use the transfer function's coefficients as coefficients).

Currently in our generator the filter is assumed to be designed outside the scope and the filter coefficients are assumed to be exact. Adding one more preliminary stage of the transfer function generation with control of the coefficients accuracy permits to get a real picture on the filter implementation error. This stage can be plugged in the optimization routines in order to provide just enough accuracy of the transfer function for the final filter error-bound to be met. However, taking into account filter coefficient quantization leads to various changes in the existing algorithms within the generator. This concerns both analysis of the error through filter propagation and the WCPG measure analysis, which were developed during this thesis. The filter error analysis scheme (see Fig. (8)) must be complemented with a special filter, which describes the coefficient quantization error. In case of filter coefficients represented as intervals with small radii, the WCPG measure can not obviously be computed with arbitrary precision. However, a trusted interval for the output WCPG can be obtained, which is an on-going work. Another important filter characteristic of the filters is their ℓ_2-norm. Its computation involves solving an instance of Lyapunov equations [52]. Passing to the case of interval matrices requires solving the Lyapunov equations in multiple-precision interval arithmetic, which is not a trivial task.

Finally, I would like to get a better understanding of the implementation of elementary functions. This would teach me new numerical analysis techniques and permit to draw the parallels between the libm and filter generation. When implementing a mathematical function, we often use its polynomial approximation. At the same time, a filter is characterized by its transfer function, which is a ratio of polynomials. Implementing a transfer function has some similarities with implementation of rational functions . Moreover, there exist some filter structures, which can be interpreted as the common polynomial evaluation schemes, such as Horner's method. Drawing a parallel between filter and function implementation might offer some new insight on the analysis of both processes as well as provide novel combined approaches to both signal processing and computer arithmetic domains.

References

[1] T. Kailath, *Linear Systems*. Prentice-Hall, 1980.

[2] E. Jury, *Theory and application of the z-transform method*. Wiley, 1964.

[3] B. Girod, R. Rabenstein, and A. Stenger, *Signals and systems*. Wiley, 2001.

[4] J. Muller, *Elementary Functions:: Algorithms and Implementation*. Birkhäuser Boston, 2013.

[5] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall,, 1975.

[6] B. Friedland, *Control Systems Design: An Introduction to State-Space Methods*. McGraw-Hill Higher Education, 1985.

[7] A. Fettweiss, "Wave digital filters: Theory and practice," *Proc. of the IEEE*, vol. 74, no. 2, 1986.

[8] R. Barsainya, T. K. Rawat, and R. Mahendra, "A new realization of wave digital filters using {GIC} and fractional bilinear transform," *Engineering Science and Technology, an International Journal*, vol. 19, no. 1, pp. 429 – 437, 2016.

[9] R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*. Springer US, 2012.

[10] T. Finley, "Two's complement." Cornell University lecture notes, 2000.

[11] *Fundamentals of Statistical Signal Processing, Volume 1: Estimation Theory*. Pearson Education.

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

[13] T. Hilaire, *Analyse et synthèse de l'implémentation de lois de contrôle-commande en précision finie (Étude dans le cadre des applications automobiles sur calculateur embarquée)*. PhD thesis, Université de Nantes, June 2006.

[14] J.-C. Bajard, L.-S. Didier, and T. Hilaire, "ρ-direct form transposed and residue number systems for filter implementations," in *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, pp. 1–4, IEEE, 2011.

[15] T. Hilaire, P. Chevrel, and J.-P. Clauzel, "Pole sensitivity stability related measure of FWL realization with the implicit state-space formalism," in *5th IFAC Symposium on Robust Control Design (ROCOND'06)*, July 2006.

[16] T. Hilaire and B. Lopez, "Reliable implementation of linear filters with fixed-point arithmetic," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, 2013.

[17] B. Lopez, *Implémentation optimale de filtres linéaires en arithmétique virgule fixe*. PhD thesis, UPMC, 2015.

[18] A. Volkova and T. Hilaire, "Fixed-point implementation of lattice wave digital filter: Comparison and error analysis," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pp. 1118–1122, Aug 2015.

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

[20] A. Volkova, T. Hilaire, and C. Lauter, "Reliable evaluation of the worst-case peak gain matrix in multiple precision," in *Computer Arithmetic (ARITH), 2015 IEEE 22nd Symposium on*, pp. 96–103, June 2015.

[21] A. Volkova, T. Hilaire, and C. Lauter, "Determining fixed-point formats for a digital filter implementation using the worst-case peak gain measure," in *2015 49th Asilomar Conference on Signals, Systems and Computers*, pp. 737–741, Nov 2015.

[22] J. v. Newmann, "First draft of a report on the edvac," tech. rep., 1945.

[23] G. Li, C. Wan, and G. Bi, "An improved rho-DFIIt structure for digital filters with minimum roundoff noise," *IEEE Trans. on Circuits & Systems*, vol. 52, no. 4, pp. 199–203, 2005.

[24] Z. Zhao and G. Li, "Roundoff noise analysis of two efficient digital filter structures," *IEEE Transactions on Signal Processing*, vol. 54, pp. 790–795, February 2006.

[25] 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.

[26] B. Lopez, T. Hilaire, and L.-S. Didier, "Formatting bits to better implement signal processing algorithms," in *4th Int. conf. PECCS, proceedings of*, 2014.

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

[28] A. Mutambara, *Design and Analysis of Control Systems*. Taylor & Francis, 1999.

[29] L. Gazsi, "Explicit formulas for lattice wave digital filters," *IEEE Trans. Circuits & Systems*, vol. 32, no. 1, 1985.

[30] H. Johansson and L. Wanharomar, "Digital hilbert transformers composed of identical allpass subfilters," in *ISCAS 1998. Proceedings of*, vol. 5, pp. 437–440 vol.5.

[31] H. Johansson and L. Wanharomar, "Design of linear-phase lattice wave digital filters."

[32] T. S. J. Yli-Kaakinen, "A systematic algorithm for the design of multiplierless lattice wave digital filters," in *First International Symposium on Control, Communications and Signal Processing*, pp. 393 – 396, 2004.

[33] H. O. O. G. W. L. L. Wanharomar, "An environment for design and implementation of energy efficient digital filters," in *SSoCC*, apr 2003.

[34] J. Yli-Kaakinen and T. Saramäki, "A systematic algorithm for the design of lattice wave digital filters with short-coefficient wordlength," *IEEE Trans. on Circuits & Systems*, 2007.

[35] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction To Algorithms*. MIT Press, 2001.

[36] A. F. H. Levin and A. Sedlmeyer, "Wave digital lattice filters," *Int. J. Circ. Theor. Appl.*, vol. 2, pp. 203–211, 1974.

[37] *Adaptive Signal Processing*. Pearson Education.

[38] J. Carletta, R. Veillette, F. Krach, and Z. F., "Determining appropriate precisions for signals in fixed-point iir filters," in *Design Automation Conference, 2003. Proceedings*, pp. 656–661, 2003.

[39] J. Lopez, C. Carreras, and O. Nieto-Taladriz, "Improved interval-based characterization of fixed-point LTI systems with feedback loops," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 11, pp. 1923–1933, 2007.

[40] S. Gershgorin, "Über die Abgrenzung der Eigenwerte einer Matrix.," *Bull. Acad. Sci. URSS*, vol. 1931, no. 6, pp. 749–754, 1931.

[41] H. Dawood, *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. LAP Lambert Academic Publishing, 2011.

[42] S. M. Rump, "New results on verified inclusions," in *Accurate Scientific Computations, Symposium, 1985, Proceedings*, pp. 31–69, 1985.

[43] S. M. Rump, "Solution of linear systems with verified accuracy," *Applied numerical mathematics*, vol. 3, no. 3, pp. 233–241, 1987.

[44] S. M. Rump, "Reliability in computing: The role of interval methods in scientific computing," ch. Algorithms for Verified Inclusions Theory and Practice, pp. 109–126, Academic Press, 1988.

[45] S. M. Rump, "Guaranteed inclusions for the complex generalized eigenproblem," *Computing*, vol. 42, pp. 225–238, Sept. 1989.

[46] D. Lefebvre, P. Chevrel, and S. Richard, "An H_∞ based control design methodology dedicated to the active control of longitudinal oscillations," *IEEE Trans. on Control Systems Technology*, vol. 11, no. 6, pp. 948–956, 2003.

[47] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[48] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, pp. 13:1–13:15, 2007.

[49] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[50] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability* (A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, eds.), vol. 185 of *Frontiers in Artificial Intelligence and Applications*, ch. 26, pp. 825–885, IOS Press, Feb. 2009.

[51] S.-I. Filip, "A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters." Preliminary version accepted for publication, Mar. 2015.

[52] A. Barraud, "A numerical algorithm to solve $a^T xa - x = q$," *IEEE Transactions on Automatic Control*, vol. 22, pp. 883–885, Oct 1977.