Optimal Word-Length Allocation for the Fixed-Point Implementation of Linear Filters and Controllers

Thibault Hilaire, Hacène Ouzia, Benoit Lopez

Optimal word-length allocation for the fixed-point implementation of linear filters and controllers

Thibault Hilaire*†, Hacène Ouzia* and Benoit Lopez
*Sorbonne Université, LIP6 UMR 7606, F-75005 Paris, France
†INRIA, Université Paris-Saclay, F-91120 Palaiseau, France

***Abstract*—This article presents a word-length optimization problem under accuracy constraints for the hardware implementation of linear signal processing systems with fixed-point arithmetic. For State-Space systems (describing a linear filter or a controller), a complete error analysis is exhibited, where the final output error bound depends on the word-lengths and the fixed-point formats chosen for each variable. The Most Significant Bit of each one can be determined in order to guarantee that no overflow occurs. Thus, it is possible to obtain a hardware implementation minimizing resource use. This leads to a convex nonlinear integer optimization problem where the resources to minimize and the accuracy constraints depend on the internal word-lengths. This problem can then be solved with appropriate heuristics. Finally, a global approach is proposed and illustrated by some examples.**

I. Introduction

This article addresses the transformation of linear filters or controllers into hardware operators using Fixed-Point arithmetic. Digital filters are ubiquitous algorithms that transform digital signals, and are mainly used in applications based on Signal Processing or Control theories, such as robotics, communications, aeronautic, automotive, etc. The algorithms considered here are the Linear Time Invariant filters/controllers expressed using the common State-Space representation.

The hardware implementation of these algorithms in FPGA logic or ASIC is a difficult task when the accuracy of the output signal is considered: due to the finite precision arithmetic used for the computations (mainly Fixed-Point arithmetic) and the recursive structure of these algorithms, the roundoff error due to the internal computations may accumulate and make the implemented output diverge from the exact result. The main goal of the designer is then to provide an implementation where the output error is guaranteed to be bounded by a given bound, while using the least amount of logic resources possible. Contrary to classical signal processing approach where the errors are modeled as noise [1] (statistical approach), we rather use worst-case bounds in order to provide the guarantee required for some critical systems. Our error analysis provides reliable error bound (bound on the worst-case error), by opposition to the Signal to Quantization Noise Ratio, used classically as an indicator of the output error (its variance) [2], [3], [4]. It will allow to build linear filter/controller architecture that behave as if the computation was performed with infinite accuracy, and then converted to the low precision output format with an error smaller than its least significant bit [5]. And since we want to dedicate the smallest number of bits possible for each operation, our approach leads to a classical optimal word-length allocation problem.

We here consider the first parts of the filter-to-code flow exhibited in Figure 1, that concern the determination of these word-lengths under accuracy constraints. We first determine the Most Significant Bit position in order to guarantee that no overflow ever occurs. Then, using a last bit accurate operator that computes the Sum of Products by real Constants (SoPC) studied in [5], we perform the error analysis of the algorithm, i.e., we bound the error on the output due to the propagation of the roundoff errors into the recursive algorithm. This allows us to define the optimization problem.

This optimization problem has already been studied in the past, but most of the researches [6], [7], [8] consider the roundoff errors as noise and study their propagation through the algorithm with statistical means (mean and variance) that do not give any guarantee on the range of the errors. Some of them also use range analysis that do not recursive algorithms [9] or provide pessimistic range estimations [10].

The main contributions of this work are a reliable error analysis with respect to the internal word-lengths and the formulation of the word-length optimization as a convex nonlinear integer optimization problem with some simple heuristics computing good feasible solutions.

The paper is organized as follows. In section II, the classical State-Space representation and the Worst-Case Peak Gain theorem are reminded. Fixed-Point arithmetic and the determination of the Most Significant Bit are given in section III, based on previous work [11]. In Sections IV and V , the Sum of Products by Constants and the error analysis are discussed. In section VI, we give a formulation of the optimization problem and its solution is discussed. Some real-life examples are given in Section VII., before concluding remarks.

Notations

Throughout this paper, column vectors are in lowercase boldface and matrices in uppercase boldface. The matrix $\boldsymbol{I}_n$ denotes the $n \times n$ identity matrix. The matrix $\mathbb{1}$ and $\mathbf{0}$ are vectors full of 1s and 0s, respectively (their size are given by the context). Moreover, for the briefness of some equations, some functions, like absolute value, expectation, logarithm, etc. and some comparisons can be lifted to vectors and matrices, i.e., they will be applied component-wise. For example, the absolute value $|\boldsymbol{A}|$ of a matrix $\boldsymbol{A}$ is the matrix of the absolute values of $\boldsymbol{A}$ (component-wise absolute value).

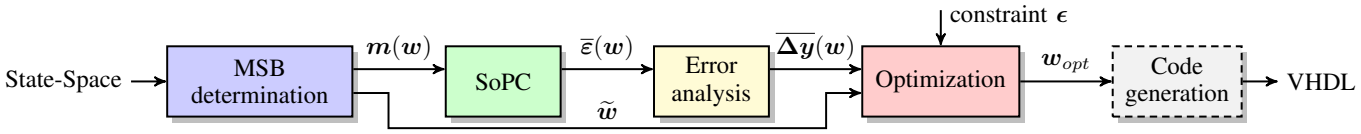


Fig. 1: From state-space to optimized word-lengths, and later VHDL code generation.

II. Prerequisites

A. State-Space

Let $\mathscr{H}$ be a stable, discrete-time Linear Time Invariant (LTI) State-Space system $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$, i.e., a system used as a filter or controller such that the inputs and outputs are linked with the following system of equations:

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

where k is the discrete time ($k \in \mathbb{Z}$), $\boldsymbol{u}(k)$ is the vector of q inputs ($\boldsymbol{u}(k) \in \mathbb{R}^q$), $\boldsymbol{y}(k)$ the vector of p outputs ($\boldsymbol{y}(k) \in \mathbb{R}^p$), $\boldsymbol{x}(k)$ the state vector that stores the internal states of the system ($\boldsymbol{x}(k) \in \mathbb{R}^n$) and $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ are given matrices that entirely define the system. These matrices are designed accordingly to the desired behavior of the controller or filter. The reader should refer to [12], [13] for filter design and [14], [15] for controller design.

In the following, we will consider two kinds of State-Space system: our main system $\mathscr{H}$ to be implemented in finite precision, and some other State-Space systems that will be used to study the effect of finite precision formats over $\mathscr{H}$.

The impact of finite precision on our filter $\mathscr{H}$ is classically divided in two separate effects: the effects of the coefficient quantization (how the quantization of the coefficients impacts the input-output relationship) and the effect of the round-off errors (how these errors due to the finite precision computations accumulate and propagate through the filter to the output). This article is based on multipliers by real constants, in such a way that the error due to the quantization of the coefficients is embedded into the roundoff errors, as shown in Section IV.

B. Worst-Case Peak Gain

The following proposition is a classical result [16], [17] used for the error analysis of linear systems. It bounds the outputs of a stable filter with respect to the inputs. (when Interval Arithmetic or Affine Arithmetic are used directly on eq. (1), they lead to oversized intervals [10])

Proposition 1. *Let $\mathscr{E}$ be a q-input p-output linear filter with State-Space coefficients $(\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}, \boldsymbol{N})$. Let $\boldsymbol{u}(k)$ be a vector signal ($\boldsymbol{u}(k) \in \mathbb{R}^q$) bounded by $\bar{\boldsymbol{u}} \in \mathbb{R}^q$ (component-wise bound, i.e., $|\boldsymbol{u}_i(k)| \leqslant \bar{\boldsymbol{u}}_i, \forall 1 \leqslant i \leqslant q, \forall k \geqslant 0$).*
Then, for null initial conditions, the output $\boldsymbol{y}(k)$ will be bounded over time iff the spectral radius of $\boldsymbol{K}$ (denoted $\rho(\boldsymbol{K})$) is strictly less than 1 (this property is called the Bounded Input Bounded Output stability [12]). In that case, the output is component-wise bounded by $\bar{\boldsymbol{y}} \in \mathbb{R}^p$ with

$$\bar{\boldsymbol{y}} = \langle\langle \mathscr{E} \rangle\rangle \, \bar{\boldsymbol{u}}, \tag{2}$$

where $\langle\langle \mathscr{E} \rangle\rangle$ denotes the Worst-Case Peak-Gain (WCPG) matrix of $\mathscr{E}$. The term $\langle\langle \mathscr{E} \rangle\rangle_{ij}$ corresponds to the maximum possible value on the i^{th} output when only the j^{th} input is considered (the others are null).
The matrix $\langle\langle \mathscr{E} \rangle\rangle \in \mathbb{R}^{p \times q}$ can be computed by

$$\langle\langle \mathscr{E} \rangle\rangle = |\boldsymbol{N}| + \sum_{k=0}^{\infty} \left| \boldsymbol{M}\boldsymbol{K}^k\boldsymbol{L} \right|. \tag{3}$$

Proof: Assuming $\boldsymbol{x}(0) = 0$, from eq. (1), we have

$$\boldsymbol{y}(k) = \sum_{l=0}^{k} \boldsymbol{J}(l)\boldsymbol{u}(k-l), \tag{4}$$

where $\boldsymbol{J}(k) \in \mathbb{R}^{p \times q}$ is defined by:

$$\boldsymbol{J}(k) = \begin{cases} \boldsymbol{N} & \text{if } k = 0 \\ \boldsymbol{M}\boldsymbol{K}^{k-1}\boldsymbol{L} & \text{if } k > 0. \end{cases} \tag{5}$$

$\boldsymbol{J}(k)$ is the impulse response matrix of the system, i.e., $\boldsymbol{J}_{ij}(k)$ is the response on the i^{th} output to the Dirac impulse[1] on the j^{th} input. So, the output $\boldsymbol{y}(k)$ is the result of the convolution of $\boldsymbol{J}(k)$ by $\boldsymbol{u}(k)$.
Then the output is (component-wise) bounded by

$$|\boldsymbol{y}(k)| \leqslant \left(\sum_{l=0}^{k} |\boldsymbol{J}(l)| \right) \bar{\boldsymbol{u}}, \quad \forall k \geqslant 0. \tag{6}$$

Finally

$$\forall k \geqslant 0, \quad |\boldsymbol{y}(k)| < \left(\sum_{l=0}^{\infty} |\boldsymbol{J}(l)| \right) \bar{\boldsymbol{u}}.$$

■

Remark 1. *Suppose we fix i and k, and build an input signal $\boldsymbol{u}(k)$ such that*

$$\boldsymbol{u}_j(l) = \begin{cases} \bar{\boldsymbol{u}}_j \cdot sign\left(\boldsymbol{J}_{ij}(k-l)\right) & for\ 0 \leqslant l \leqslant k \\ 0 & elsewhere \end{cases} \tag{7}$$

where $sign(x)$ is the integer -1, 0 or 1 depending on the sign of x. Then, the i^{th} component of $\boldsymbol{y}(k)$ is equal to

$$\boldsymbol{y}_i(k) = \sum_{l=0}^{k} \sum_{j=1}^{q} \boldsymbol{J}_{ij}(l)\boldsymbol{u}_j(k-l) \tag{8}$$

$$= \sum_{l=0}^{k} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j \left| \boldsymbol{J}_{ij}(l) \right| \tag{9}$$

and the bound of eq. (6) is reached for the i^{th} output.
Since $\rho(\boldsymbol{K}) < 1$, then for any arbitrary small quantity $\varepsilon > 0$,

[1] A Dirac impulse signal is a signal $\delta(k)$ such that $\delta(0) = 1$ and $\delta(k) = 0, \quad \forall k \neq 0$.

there exists $K > 0$ *such that* $\sum_{k=K+1}^{\infty} |\boldsymbol{J}(k)| < \varepsilon$. *So the vector* $\bar{\boldsymbol{y}}$ *is the supremum of any possible inputs (bounded by* $\bar{\boldsymbol{u}}$*) and it is possible to exhibit an input (the one given by* (7)*) to approach it on any given output at any given distance (since* $|\boldsymbol{y}_i(K) - \bar{\boldsymbol{y}}_i| < \varepsilon$*).*

Moreover, this WCPG matrix can be computed at any arbitrary precision, thanks to the multi-precision implementation[2] proposed in [17]. Proposition 1 and the Remark 1 have also been formally proved in the formal proof assistant Coq[3] [18].

III. Determining the Fixed-Point Formats

A. Fixed-Point Arithmetic

The signed Fixed-Point (FxP) arithmetic with two's complement representation is used in this paper to represent numbers and perform the computations. Let x be a w-bit long fixed-point number, with bits $\{x_i\}_{m \geqslant i \geqslant \ell}$:

$$x = -2^m x_m + \sum_{i=\ell}^{m-1} 2^i x_i \tag{10}$$

where m and ℓ are the positions of the most significant bit (MSB) and the least significant bit (LSB) of x, respectively. So the word-length w and the Most and Least Significant Bit positions, m and ℓ are linked with

$$w = m - \ell + 1. \tag{11}$$

The couple (m, ℓ) denotes the Fixed-Point Format of x. Figure 2 exhibits the fixed-point numbers and their binary representation.

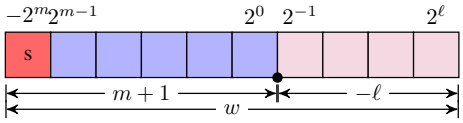


Fig. 2: Fixed-point number in format (m, ℓ).

A variable with (m, ℓ) as FxP format can take any values in the interval $[-2^m; 2^m - 2^\ell]$ with a quantization step of 2^ℓ. Internally, the Fixed-Point numbers are represented with an integer X (denoted *mantissa*) formed by the w-bit $\{x_j\}$ and an implicit scaling factor 2^ℓ, i.e.,

$$X = -2^{w-1} x_m + \sum_{i=0}^{w-2} 2^i x_{i+\ell} \in \mathbb{Z} \tag{12}$$

or equivalently $x = X 2^\ell$. All the operations are then integer operations on the mantissa X.

B. Most Significant Bit Determination

The first step to implement a State-Space system in Fixed-Point arithmetic (or to perform the error analysis) is to determine the Fixed-Point format of all the variables used, i.e., the states and the outputs (the input formats are given by the user). Since our analysis depends on the word-lengths, we will first determine all the Most Significant Bit positions, and then deduce the Least Significant Bit position using (11).
It is necessary to determine the MSB position of each of these variables to ensure that all the possible values, over time, can be represented, i.e., that no overflow ever occurs. And of course, we are interested in the minimum MSB position.

Suppose that all the inputs belong to an interval (componentwise) bounded by $\overline{\boldsymbol{u}} \in \mathbb{R}^q$ (i.e., $\forall k, \quad |\boldsymbol{u}_i(k)| \leqslant \overline{\boldsymbol{u}}_i$ for $i = 1, \ldots, q$), then Proposition 1 can be used to bound the outputs. However, to also get bounds on the state vector, we must build a new State-Space system $\mathscr{H}_{\boldsymbol{\zeta}}$ where the state vector is also on the output (this system is of course only used for the MSB analysis, not for the implementation).

Denote $\boldsymbol{\zeta}(k) \triangleq \begin{pmatrix} \boldsymbol{x}(k) \\ \boldsymbol{y}(k) \end{pmatrix} \in \mathbb{R}^{n+p}$ the aggregation of the state and output vectors, and $\mathscr{H}_{\boldsymbol{\zeta}}$ the following state-space representation:

$$\mathscr{H}_{\boldsymbol{\zeta}} \begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{\zeta}(k) &= \boldsymbol{M}_1 \boldsymbol{x}(k) + \boldsymbol{M}_2 \boldsymbol{u}(k), \end{cases} \tag{13}$$

with

$$\boldsymbol{M}_1 \triangleq \begin{pmatrix} \boldsymbol{I}_n \\ \boldsymbol{C} \end{pmatrix} \in \mathbb{R}^{(n+p)\times n}, \quad \boldsymbol{M}_2 \triangleq \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \in \mathbb{R}^{(n+p)\times q}.$$

The bound $\overline{\boldsymbol{\zeta}}$ on the vector $\boldsymbol{\zeta}(k)$ can be obtained with Proposition 1 applied to $\mathscr{H}_{\boldsymbol{\zeta}}$, i.e.,

$$\overline{\boldsymbol{\zeta}} = \langle\langle \mathscr{H}_{\boldsymbol{\zeta}} \rangle\rangle \, \overline{\boldsymbol{u}}. \tag{14}$$

We denote $\boldsymbol{w}$ the word-length vector of $\boldsymbol{\zeta}$ (i.e., $\boldsymbol{w}_i$ the word-length of the i^{th} element of $\boldsymbol{\zeta}$, a state or an output, according to i), and $\boldsymbol{m}$ the Most Significant Bit position vector of $\boldsymbol{\zeta}$. The MSB position $\boldsymbol{m}_i$ of each element of ζ_i must be the lowest integer such that

$$\forall k, \quad \boldsymbol{\zeta}_i(k) \in [-2^{\boldsymbol{m}_i}, 2^{\boldsymbol{m}_i}(1 - 2^{1-\boldsymbol{w}_i})]. \tag{15}$$

Since the filter is linear and the input interval centered at zero, the output and state intervals are also centered at zero. Therefore, we seek to determine the least integer $\boldsymbol{m}_i$ such that

$$\forall k, \quad |\boldsymbol{\zeta}_i(k)| \leqslant 2^{\boldsymbol{m}_i}(1 - 2^{1-\boldsymbol{w}_i}). \tag{16}$$

By applying the above bound on eq. (14), we obtained a simple formula for the computation of the MSB position of the states and the outputs (for $i = 1, \ldots, n+p$):

$$\boldsymbol{m}_i = \left\lceil \log_2\left(\overline{\boldsymbol{\zeta}}_i\right) - \log_2\left(1 - 2^{1-\boldsymbol{w}_i}\right) \right\rceil \tag{17}$$

or in a vector form:

$$\boldsymbol{m} = \left\lceil \log_2\left(\langle\langle \mathscr{H}_{\boldsymbol{\zeta}} \rangle\rangle \, \overline{\boldsymbol{u}}\right) - \log_2\left(\mathbb{1} - 2^{\mathbb{1}-\boldsymbol{w}}\right) \right\rceil. \tag{18}$$

Remark 2. *It is important to notice that the above reasoning does not consider the roundoff errors that occur in the evaluation of the Sum-of-Products and may make the states or the outputs overpass the bounds in* (14) *and produce overflows. This problem has been fully considered in [11], [19], as well as the error analysis of the MSB computation formula (mainly by controlling the accuracy of the WCPG evaluation [17]).*

In the case where we want to minimize the word-length $\boldsymbol{w}_i$, it is of interest to find an equivalent expression for (17) where the dependency on $\boldsymbol{w}_i$ is easier to handle. First, we suppose that the word-lengths are at least 3 bits, so $0 < -\log_2\left(1 - 2^{1-\boldsymbol{w}_i}\right) \leqslant \frac{1}{2}$. Secondly, if $\overline{\boldsymbol{\zeta}}_i$ is a power of 2,

[2] https://github.com/fixif/WCPG
[3] https://coq.inria.fr

then $\boldsymbol{m}_i = \lceil \log_2(\overline{\zeta}_i) \rceil + 1$ for every word-length $\boldsymbol{w}_i \geqslant 2$. And finally, we can remark that we have $\boldsymbol{m}_i = \lceil \log_2(\overline{\zeta}_i) \rceil$ for a large enough value of $\boldsymbol{w}_i$ and $\boldsymbol{m}_i = \lceil \log_2(\overline{\zeta}_i) \rceil + 1$ if $\boldsymbol{w}_i$ is not great enough, or if $\overline{\zeta}_i$ is a power of 2. This leads to the following proposition:

Proposition 2. *Equivalently to equation* (17)*, the minimum MSB position that guarantees that no overflow occurs can be computed with*

$$\boldsymbol{m} = \lceil \log_2 (\langle\langle \mathscr{H}_\zeta \rangle\rangle \overline{\boldsymbol{u}}) \rceil + \boldsymbol{\delta} \tag{19}$$

where $\boldsymbol{\delta} \in \mathbb{R}^{n+p}$ *depends on* $\boldsymbol{w}$ *and is such that*

$$\boldsymbol{\delta}_i = \begin{cases} 0 & \text{if } \boldsymbol{w}_i \geqslant \widetilde{\boldsymbol{w}}_i \\ 1 & \text{if } \boldsymbol{w}_i < \widetilde{\boldsymbol{w}}_i \end{cases} \tag{20}$$

and

$$\widetilde{\boldsymbol{w}} \triangleq \mathbb{1} + \lceil \log_2(\overline{\zeta}) \rceil - \left\lfloor \log_2 \left(2^{\lceil \log_2(\overline{\zeta}) \rceil} - \overline{\zeta} \right) \right\rfloor \tag{21}$$

Note that $\widetilde{\boldsymbol{w}}_i$ *is set to* $+\infty$ *when* $\overline{\zeta}_i$ *is a power of 2.*

Proof: Let us suppose that $\overline{\zeta}_i$ is not a power of 2. We compute the minimum value of $\boldsymbol{w}_i$ (denoted $\widetilde{\boldsymbol{w}}_i$) for which we have $\lceil \log_2(\overline{\zeta}_i) - \log_2(1 - 2^{1-\boldsymbol{w}_i}) \rceil = \lceil \log_2(\overline{\zeta}_i) \rceil$. This holds iff

$$0 \leqslant \lceil \log_2(\overline{\zeta}_i) \rceil - \log_2(\overline{\zeta}_i) + \log_2(1 - 2^{1-\boldsymbol{w}_i}) < 1 \tag{22}$$

The 2nd term of this inequality (< 1) is always true since $\lceil \log_2(\overline{\zeta}_i) \rceil - \log_2(\overline{\zeta}_i) < 1$ and $\log_2(1 - 2^{1-\boldsymbol{w}_i}) < 0$ for $\boldsymbol{w}_i > 2$. The 1st inequality leads to

$$\overline{\zeta}_i \cdot 2^{-\lceil \log_2(\overline{\zeta}_i) \rceil} \leqslant 1 - 2^{1-\boldsymbol{w}_i} \tag{23}$$

$$2^{1-\boldsymbol{w}_i} \leqslant 2^{-\lceil \log_2(\overline{\zeta}_i) \rceil} \left(2^{\lceil \log_2(\overline{\zeta}_i) \rceil} - \overline{\zeta}_i \right) \tag{24}$$

Since $\overline{\zeta}_i$ is not a power of 2, we have

$$1 - \boldsymbol{w}_i \leqslant -\lceil \log_2(\overline{\zeta}_i) \rceil + \log_2 \left(2^{\lceil \log_2(\overline{\zeta}_i) \rceil} - \overline{\zeta}_i \right) \tag{25}$$

and then $\boldsymbol{w}_i \geqslant \widetilde{\boldsymbol{w}}_i$ with (21). ∎

The vector $\widetilde{\boldsymbol{w}}$ can be seen as a threshold below which the word-length is too close the next binade.

IV. Sum of Products by Constants

A. Problem Statement

For the State-Space algorithm (1), but also for all the algorithms used to implement filters/controllers, the basic operation is the Sum of Products by Constants (SoPC), i.e., the accumulation of multiplications of variables by real constants

$$r = \sum_{i=1}^{N} c_i \cdot v_i = \sum_{i=1}^{N} p_i, \tag{26}$$

where the $\{p_i\}_{1 \leqslant i \leqslant N}$ denotes the products $c_i \cdot v_i$ of the real non-zero constant c_i by the variable v_i. In our context, the real constants are known, and they come from the coefficients of the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ where the null coefficients are removed. The fixed-point format of the variables v_i and the result r are also known (since they are states or outputs of the State-Space system to implement, their MSB position are determined with Section III-B, and their LSB deduced from the associated word-lengths). We denote (m, ℓ) the FxP format of the result r.

B. Bit Guards to Provide Last Bit Accuracy

As done in [5] by Volkova et al., it is possible to compute the Sum of Products by Constants r with *last bit accuracy* (also called faithful rounding), i.e., with an error smaller than the value of the last bit (LSB) of the result, here 2^ℓ.
To achieve this, the main idea is to use a slightly extended precision for the internal computation of the accumulation, using g extra bits, called *guards* bits. We first compute approximations of the products p_i on the format $(m, \ell - g)$, accumulate them on this same format $)m, \ell - g)$ (this addition is then error-free) and then round the partial result $\tilde{r}_{\text{ext}}$ to the desired format (m, ℓ). The final approximated result is denoted $\tilde{r}$, and this process is illustrated in Figure 3.

We assume that we are able to build hardware constant multipliers that compute some approximation

$$\widetilde{p}_i \approx c_i v_i \tag{27}$$

of the mathematical product $c_i v_i$ on the format $(m, \ell - g)$, and we assume that the rounding error of each of these multipliers is bounded by some $\overline{\varepsilon}_i(g)$:

$$|\widetilde{p}_i - p_i| < \overline{\varepsilon}_i(g), \quad \forall v_i. \tag{28}$$

The value of $\overline{\varepsilon}_i(g)$ depends on the constant: multiplication by zero will be exact, as will be, under some conditions, multiplications by powers of two and by other constants that can be written in binary on few bits. In the general case where c_i is real, the multiplier will entail a rounding error which depends on the multiplier technique used (a detailed example will be shown in the next subsection). However, whatever the technique, this error bound can be made as small as needed by increasing g (in other words, by computing more accurately).

We then compute the sum $\tilde{r}_{\text{ext}}$ of the $\widetilde{p}_i$. This summation, as long as it is performed with adders of the proper size (here $m - \ell + 1 + g$ bits), will entail no error. Indeed, fixed-point addition of numbers of the same format may entail overflows (these have been taken care of by choosing appropriate MSB for r), but no rounding error. This enables us to write

$$\tilde{r}_{\text{ext}} = \sum_{i=1}^{N} \widetilde{p}_i, \tag{29}$$

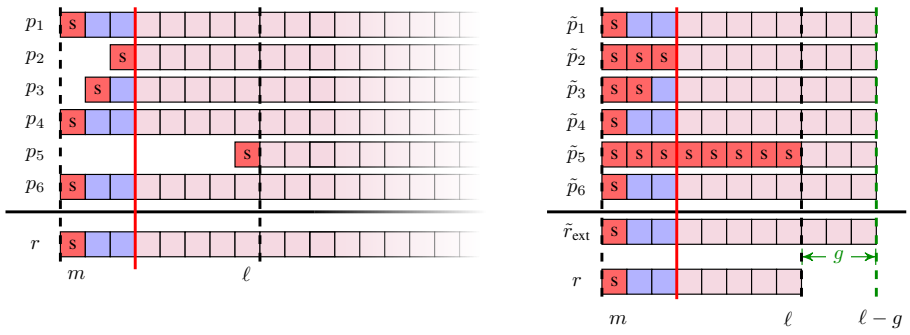
therefore the total rounding error of the sum is bounded by

$$\sum_{i=1}^{N} \overline{\varepsilon}_i(g). \tag{30}$$

As each $\overline{\varepsilon}_i(g)$ can be made arbitrarily small by increasing g, there exists some g such that

$$\sum_{i=1}^{N} \overline{\varepsilon}_i(g) < 2^{\ell-1}. \tag{31}$$

The intermediate result $\tilde{r}_{\text{ext}}$ now has g more bits at its LSB than required (see Figure 3b). It therefore alse needs to be rounded to the target format. Rounding to precision ℓ is obtained by first adding $2^{\ell-1}$ then discarding bits lower than 2^ℓ. In the worst case, this will entail an error of at most $2^{\ell-1}$.



(a) Exact SoPC

(b) Last bit accurate SoPC

Fig. 3: The Sum of Products by Constants is computed with the accumulation of approximated products with g extra bits (so with format $(m, \ell - g)$), and then rounded to the format (m, ℓ). The number of extra bits g is such that $|r - \tilde{r}| < 2^{\ell}$.

To sum up, the overall error of a last-bit accurate SOPC architecture is:

$$\left| \tilde{r} - \sum_{i=1}^{N} c_i \cdot v_i \right| < 2^{\ell-1} + 2^{\ell-1} \quad = \quad 2^{\ell}. \tag{32}$$

The previous discussion is independent of the target technology. However, the actual computation of the optimal g out of constraint (31) will depend on the multiplier technique chosen.

Remark 3. *It may happen that some products p_i have a MSB strictly greater than the result's MSB m. But, since the MSB of r has been computed to guarantee that no overflow ever occurs, we can drop out all the extra MSB bits They do not contribute to the final result, thanks to the modular properties of the two's complement arithmetic [20], [21].*

C. Table-based Constant Multiplier for FPGAs

A table-based constant multiplier for FPGAs can be used to efficiently perform a multiplication by a *real* constant. This has been implemented in the tool FloPoCo[4] and detailed in [5]. It is based on a variation of the KCM technique [22], [23]. The idea is to break down the binary representation of the input v_i to multiply into d_i chunks of α bits (or equivalently to use a radix-2^{α} basis to represent v_i). When α is the LUT input size, we can build the constant multiplier using one look-up table per output bit (see [5] for the details).

The error of such a multiplier is bounded by $d_i 2^{\ell-g}$, and this bound can be reduced accordingly to the coefficient (power of 2, constant that can be written with few bits, etc.). This error is proportional to 2^{-g}, so can made as small as needed by increasing g.

It can be shown that using g extra bits such that $g \leqslant \left\lceil \log_2 \left(\sum_{i=1}^{N} d_i \right) \right\rceil$ is enough to achieve (31), so we will never use more that $\left\lceil \log_2 \left(\frac{W}{\alpha} + N \right) \right\rceil$ extra guard bits for the SoPC, where W is the total number of bits of the variables in the SoPC (since we have $d_i = \left\lceil \frac{w_i}{\alpha} \right\rceil$ where w_i is the word-length of the i$^{\text{th}}$ variable v_i).

[4] http://flopoco.gforge.inria.fr/

Moreover, the main summation that includes the terms required for the bit sign extension and the final rounding can be implemented efficiently using compression techniques [5].

V. Error Analysis

In this Section, we now study how the roundoff errors due to the $n+p$ approximated Sum of Products by Constants impact the outputs of the implemented State-Space.

At each step k, the computation of the state and output vectors (using eq. (1)) rely on sum-of-products, one per state and output. As seen in Section IV each SoPC may include an error. State-Space (1) is then changed into

$$\mathscr{H}^* \begin{cases} \boldsymbol{x}^*(k+1) & = \boldsymbol{A}\boldsymbol{x}^*(k) + \boldsymbol{B}\boldsymbol{u}(k) + \boldsymbol{\varepsilon}_{\boldsymbol{x}}(k) \\ \boldsymbol{y}^*(k) & = \boldsymbol{C}\boldsymbol{x}^*(k) + \boldsymbol{D}\boldsymbol{u}(k) + \boldsymbol{\varepsilon}_{\boldsymbol{y}}(k), \end{cases} \tag{33}$$

where $\boldsymbol{\varepsilon}_{\boldsymbol{x}}(k)$ and $\boldsymbol{\varepsilon}_{\boldsymbol{y}}(k)$ are the vectors of roundoff errors due to the sum-of-products evaluation. Denote $\boldsymbol{\varepsilon}(k)$ the column vector that aggregates those error vectors:

$$\boldsymbol{\varepsilon}(k) \triangleq \begin{pmatrix} \boldsymbol{\varepsilon}_{\boldsymbol{x}}(k) \\ \boldsymbol{\varepsilon}_{\boldsymbol{y}}(k) \end{pmatrix} \in \mathbb{R}^{n+p}. \tag{34}$$

Denoting $\boldsymbol{\Delta x}(k) \triangleq \boldsymbol{x}^*(k) - \boldsymbol{x}(k)$ the computation error on state $\boldsymbol{x}(k)$, and $\boldsymbol{\Delta y}(k) \triangleq \boldsymbol{y}^*(k) - \boldsymbol{y}(k)$ the output error (i.e., the errors on the output $\boldsymbol{y}(k)$), it follows from (1) and (33) that $\boldsymbol{\Delta y}(k)$ and $\boldsymbol{\varepsilon}(k)$ are linked with:

$$\mathscr{H}_{\boldsymbol{\varepsilon}} \begin{cases} \boldsymbol{\Delta x}(k+1) & = \boldsymbol{A}\boldsymbol{\Delta x}(k) + \boldsymbol{M}_3\boldsymbol{\varepsilon}(k) \\ \boldsymbol{\Delta y}(k) & = \boldsymbol{C}\boldsymbol{\Delta x}(k) + \boldsymbol{M}_4\boldsymbol{\varepsilon}(k). \end{cases} \tag{35}$$

and $\boldsymbol{M}_3 = (\boldsymbol{I}_n \quad \boldsymbol{0}) \in \mathbb{R}^{n \times (n+p)}$, $\boldsymbol{M}_4 = (\boldsymbol{0} \quad \boldsymbol{I}_n) \in \mathbb{R}^{p \times (n+p)}$.

These equations describe a State-Space system $\mathscr{H}_{\boldsymbol{\varepsilon}}$ with matrices $(\boldsymbol{A}, \boldsymbol{M}_3, \boldsymbol{C}, \boldsymbol{M}_4)$. It computes the output error $\boldsymbol{\Delta y}(k)$ from the computational errors $\boldsymbol{\varepsilon}(k)$.

The error analysis can be summarized with Figure 4, where the implemented filter $\mathscr{H}^*$ can be seen as the exact filter $\mathscr{H}$ perturbed by the error filter $\mathscr{H}_{\boldsymbol{\varepsilon}}$ that amplifies the error $\boldsymbol{\varepsilon}$.

The system $\mathscr{H}_{\boldsymbol{\varepsilon}}$ expresses how the errors $\boldsymbol{\Delta x}(k)$ propagate through the filter $\mathscr{H}$ and modify the outputs $\boldsymbol{y}(k)$.

Using the WCPG theorem (Proposition 1), we can compute the bound of the output error, denoted $\overline{\boldsymbol{\Delta y}}$ by:

$$\overline{\boldsymbol{\Delta y}} = \langle\langle \mathscr{H}_{\boldsymbol{\varepsilon}} \rangle\rangle \, \overline{\boldsymbol{\varepsilon}} \tag{36}$$

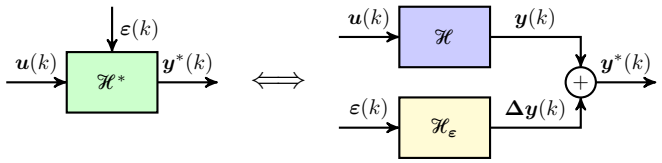


Fig. 4: The internal roundoff errors are equivalent to extern perturabations on the ouput, with the error filter $\mathscr{H}_{\boldsymbol{\varepsilon}}$.

where $\overline{\varepsilon}$ is the bound on the roundoff error. This bound can be determined using the SoPC error analysis of Section IV, where we have shown how to achieve last bit accurate Sum of Products by Constants, leading to

$$\overline{\varepsilon} = 2^{\boldsymbol{m}-\boldsymbol{w}+\mathbb{1}}. \tag{37}$$

Finally, with $\boldsymbol{m}$ determined by Proposition 2, we have:

$$\overline{\boldsymbol{\Delta y}} = \langle\langle \mathscr{H}_{\boldsymbol{\varepsilon}} \rangle\rangle \, 2^{\left\lceil \log_2\left(\langle\langle \mathscr{H}_{\boldsymbol{\zeta}} \rangle\rangle \overline{\boldsymbol{u}}\right)\right\rceil + \mathbb{1} - \boldsymbol{w} + \boldsymbol{\delta}}. \tag{38}$$

The bound $\overline{\boldsymbol{\Delta y}}$ depends on $\boldsymbol{w}$ ($\boldsymbol{\delta}$ only depends whether $\boldsymbol{w}$ is greater or not than $\widetilde{\boldsymbol{w}}$).

As an element of comparison, notice that, when the errors are modeled as uniformly distributed random variables, their variance are $\sigma^2_{\boldsymbol{\varepsilon}_i} = \frac{2^{2\overline{\boldsymbol{\varepsilon}}_i}}{12}$, and the output error variance is $\sigma_{\boldsymbol{y}} = \frac{1}{12}\sum_i \|\mathscr{H}_{\boldsymbol{\varepsilon}_i}\|_2^2 \, 2^{2\overline{\boldsymbol{\varepsilon}}_i}$, where $\|\cdot\|_2$ is the filter ℓ_2 norm.

VI. Word-length Optimization

A. The Optimization Problem

Since we want to find hardware implementations that minimize hardware resource use or impact (area, number of lookup tables (LUTs), power consumption, etc.) but still guarantee a given accuracy of the outputs, we can formulate this as an optimization problem with respect to the word-lengths.
The optimization problem to be solved has objective function $f(\boldsymbol{w}) = \sum_{j=1}^{n+p} \boldsymbol{w}_j$ (see Section VI-D for a discussion about the objective function to minimize) and the following accuracy constraints on the output error:

$$\overline{\boldsymbol{\Delta y}}_i \leqslant \boldsymbol{\epsilon}_i, \quad \text{for } i = 1, \ldots, p \tag{39}$$

where $\boldsymbol{\epsilon}_i$ is the bound allowed on each component of the output error. From (38), these constraints can be rewritten as

$$\sum_{j=1}^{n+p} \boldsymbol{E}_{ij} 2^{-\boldsymbol{w}_j + \boldsymbol{\delta}_j} \leqslant \boldsymbol{\epsilon}_i, \quad \text{for } i = 1, \ldots, p, \tag{40}$$

where $\boldsymbol{E} \in \mathbb{R}^{p \times (n+p)}$ is a constant matrix defined by

$$\boldsymbol{E}_{ij} \triangleq \langle\langle \mathscr{H}_{\boldsymbol{\varepsilon}} \rangle\rangle_{ij} \, 2^{\left\lceil \log_2\left(\langle\langle \mathscr{H}_{\boldsymbol{\zeta}} \rangle\rangle \overline{\boldsymbol{u}}\right)_j \right\rceil}. \tag{41}$$

Thus, the optimization variables are the integer values $\{\boldsymbol{w}_j\}_{1 \leqslant j \leqslant n+p}$ and $\{\boldsymbol{\delta}_j\}_{1 \leqslant j \leqslant n+p}$.

In order to solve this optimization problem, the dependency between $\boldsymbol{\delta}$ and $\boldsymbol{w}$ should be converted into linear constraints.

Proposition 3. *Let w and $\tilde{w}$ be two integers belonging to the the interval $[2, u]$, such that the value of $\tilde{w}$ is fixed. Let δ a binary variable such that*

$$\delta = \begin{cases} 0 & \text{if } w \geqslant \tilde{w} \\ 1 & \text{if } w < \tilde{w} \end{cases} \tag{42}$$

Then, this last condition is equivalent to the following linear constraints:

$$(2 - \tilde{w})\,\delta + 1 \leq w - \tilde{w} + 1 \leq (1 - \delta)\,(u - \tilde{w} + 1)\,. \tag{43}$$

Proof: Let w, $\tilde{w}$ and δ as stated in the proposition above. On one hand, if δ equals 1 then the right constraint in (43) implies that $w < \tilde{w}$ and the left constraint, $w \geq 2$, remains valid. On the other hand, if δ equals 0 then the left constraint if (43) implies that $w \geq \tilde{w}$ and the right constrain, $w \leq u$, remains also valid. Thus $\delta = 1$ iff $w < \tilde{w}$. ∎

Remark 4. *Notice that linearizing equations (40) is not as simple as linearizing (20) since it will complicate the objective function that will become non-linear.*

Finally, our optimization problem reads

$$\boldsymbol{w}_{\text{opt}} = \arg\min \sum_{j=1}^{n+p} \boldsymbol{w}_j \tag{44}$$

subject to

$$\begin{array}{ll}
\displaystyle\sum_{j=1}^{n+p} \boldsymbol{E}_{ij} 2^{-\boldsymbol{w}_j + \boldsymbol{\delta}_j} \leqslant \boldsymbol{\epsilon}_i, & 1 \leqslant j \leqslant p, \\
(2 - \widetilde{\boldsymbol{w}}_j)\,\boldsymbol{\delta}_j + 1 \leqslant \boldsymbol{w}_j - \widetilde{\boldsymbol{w}}_j + 1, & 1 \leqslant j \leqslant n+p, \\
\boldsymbol{w}_j - \widetilde{\boldsymbol{w}}_j + 1 \leqslant (1 - \boldsymbol{\delta}_j)\,(\boldsymbol{u}_j - \widetilde{\boldsymbol{w}}_j + 1)\,, & 1 \leqslant j \leqslant n+p, \\
2 \leqslant \boldsymbol{w}_j \leqslant \boldsymbol{u}_j, & 1 \leqslant j \leqslant n+p, \\
\boldsymbol{w}_j \in \mathbb{Z},\ \boldsymbol{\delta}_j \in \{0, 1\}\,, & 1 \leqslant j \leqslant n+p,
\end{array}$$

The optimization problem (44) is a separable convex nonlinear integer optimization problem. It is an NP-hard optimization problem [24] meaning that the best known exact algorithm to solve it has exponential time complexity. The two main approaches used to solve exactly (44) are the branch-and-bound approach and the outer approximation approaches [24]. Regarding software, one can use the IBM-open-source software Bonmin[5] or the commercial solver Artylis-Knitro[6].

In the following, we also propose two other heuristics.

B. Uniform Word-lengths

In the case, where a uniform word-length is used, i.e., when the same word-length is used for all the states and the outputs ($\boldsymbol{w} = \mathbb{1}w$), we can directly find the minimal word-length w_{uni} that satisfies the constraints (40).
Suppose that w is such that $\delta_j = 0$ for all $j = 1, \ldots, n+p$ (it is the case when $w \geqslant \max(\widetilde{\boldsymbol{w}})$, where $\max(\boldsymbol{x})$ is the maximum element of the vector $\boldsymbol{x}$). Then looking for the lowest integer w_{uni} such that the constraints

$$\sum_{j=1}^{n+p} \boldsymbol{E}_{ij} 2^{-w} 2^{\boldsymbol{\delta}_j} \leqslant \boldsymbol{\epsilon}_i, \quad \text{for } i = 1, \ldots, p, \tag{45}$$

hold, leads to

$$w_{\text{uni}} = \max\left(\left\lceil \log_2\left(\boldsymbol{E}\mathbb{1}\right) - \log_2(\boldsymbol{\epsilon})\right\rceil\right). \tag{46}$$

In the other hand, if we suppose that $\delta_j = 1$ for all $j = 1, \ldots, n+p$ (it is the case when $w < \min(\widetilde{\boldsymbol{w}})$), then equation (45) leads to

$$w_{\text{uni}} = 1 + \max\left(\left\lceil \log_2\left(\boldsymbol{E}\mathbb{1}\right) - \log_2(\boldsymbol{\epsilon})\right\rceil\right). \tag{47}$$

[5] https://www.coin-or.org/Bonmin/
[6] https://www.artelys.com/

The value of w_{uni} differs in the two cases by only one bit. So, in the general case, we use the following algorithm:

1) Compute $w_{\text{guess}} \triangleq \max\left(\lceil \log_2(\boldsymbol{E}\mathbb{1}) - \log_2(\boldsymbol{\epsilon})\rceil\right)$ a first guess for the minimum word-length ;
2) Then check if w_{guess} satisfies the constraints (45). If so, $w_{\text{uni}} = w_{\text{guess}}$, otherwise $w_{\text{uni}} = w_{\text{guess}} + 1$.

In practice, the uniform word-length is used for software-based implementation (the word-lengths are often set to 8, 16, 32 or 64 bits) and also for comparison with the result of the general problem in the multiple word-length setting.

C. Equitably Distributed Budget Error

As a different heuristic for computing a feasible solution of the problem (44), one can split the i^{th} total budget error constraint $\sum_{j=1}^{n+p} \boldsymbol{E}_{ij} 2^{-\boldsymbol{w}_j+\boldsymbol{\delta}_j} \leqslant \boldsymbol{\epsilon}_i$ in $n+p$ terms, each of them concerning the budget error of the j^{th} parameter. Since in the cost function all the parameters $\boldsymbol{w}_j$ have the same weight, it is natural to split the budget error constraints (40) into $n+p$ equal terms in order to provide stricter and simpler constraints:

$$i = 1, \ldots, p,\ j = 1, \ldots, n+p : \ \boldsymbol{E}_{ij} 2^{-\boldsymbol{w}_j+\boldsymbol{\delta}_j} \leqslant \frac{\boldsymbol{\epsilon}_i}{n+p}. \quad (48)$$

Providing $\boldsymbol{w}_j$ such that (48) holds implies that the constraints (40) hold too. They lead to $\boldsymbol{w}_j - \boldsymbol{\delta}_j \geqslant \boldsymbol{w}_{\text{guess},j}$ with

$$\boldsymbol{w}_{\text{guess},j} \triangleq \max_i \left\lceil \log_2(\boldsymbol{E}_{ij}(n+p)) - \log_2(\boldsymbol{\epsilon}_i) \right\rceil. \quad (49)$$

Then, as done in Section VI-B, we check if $\boldsymbol{w}_{\text{guess},j}$ satisfies the constraints (48) (or equivalently if $\boldsymbol{w}_{\text{guess},j}$ is greater than $\widetilde{\boldsymbol{w}}_j$). If so, the chosen value $\boldsymbol{w}_{\text{eq},j}$ is equal to $\boldsymbol{w}_{\text{guess},j}$, otherwise $\boldsymbol{w}_{\text{eq},j} = \boldsymbol{w}_{\text{guess},j} + 1$.

As shown in the examples, this heuristic gives a good approximation of $\boldsymbol{w}_{\text{opt}}$ when $p = 1$.

D. Discussion About the Cost Function

In the word-length optimization problem previously defined in Section VI-A, the cost function to be minimized was the sum of the word-lengths of the states and the outputs. This cost function has been chosen as a first order approximation of a real cost function which can reflect, for instance, the power consumption, the area or the number of LUTs.

Another cost function could be to sum the number of bits used in all the additions. For a N-term addition over M bits, we can consider that $N.M$ bits are involved (see Figure 3b). Applied to the Sum of Products by Constants detailed in Section IV, the computation of the term $\boldsymbol{\zeta}_j$ involves $(n+p)(\boldsymbol{w}_j+g_j)$ bits where g_j is the number of guard bits. This number cannot be given a priori, because it is highly dependent on the coefficients involved. But we have seen that is bounded by $\left\lceil \log_2\left(\frac{1}{\alpha}\sum_{k=1}^{n+p}(\boldsymbol{w}_k + n + p)\right)\right\rceil$. So the bound on the total number of bits is roughly proportional to $g(\boldsymbol{w})$ with :

$$g(\boldsymbol{w}) \triangleq (n+p)\log_2\left(\frac{1}{\alpha}\sum_{j=1}^{n+p}(\boldsymbol{w}_j + n + p)\right) + \sum_{j=1}^{n+p}\boldsymbol{w}_j. \quad (50)$$

With $h(x) = x + (n+p)\log_2(\frac{x+(n+p)^2}{\alpha})$, we have $g(\boldsymbol{w}) = h(f(\boldsymbol{w}))$ and g is a strictly increasing function and continuous, minimizing $g(\boldsymbol{w})$ is equivalent to minimizing $f(\boldsymbol{w})$. The simpler objective function $f(\boldsymbol{w})$ is then not so simplistic.

Finally, using a much more realistic cost function such as a more accurate approximation of the number of LUTs used in the FPGAs or the power consumption or one that measures directly on the hardware generated (like real number of LUTs used) could give better results but also increase the complexity of the optimization problem. As future work, we will consider using such realistic cost functions.

VII. Real-life Examples

In the following, we consider the three real-life examples[7] from the Signal Processing and the Control contexts. For all of them, the input bound $\overline{\boldsymbol{u}}$ is set arbitrary to 10, and $\boldsymbol{\epsilon}$ to 2^{-6}. These values of course depend on the context and needs of the designer, and will not be discussed here.

The first example is an active control of longitudinal oscillations studied in [25]. It has been designed to remove the unpleasant oscillations of the vehicle by means of a controller active on the engine torque. It results in a 10^{th} order single-input single-output state-space ($n = 10$, $p = q = 1$).

The second example is a multiple-input multiple-output controller from the automotive context ($n = 4$, $p = 5$, $q = 7$). It is not published, so it can be considered as a random stable state-space controller.

The last example comes from Signal Processing. It is a 5^{th} order single-input single-output low pass filter ($n = 5$, $p = q = 1$) designed to keep low frequency signals (0 to 50Hz) and reject with at least -40dB of gain the signals above 90Hz. This elliptic filter [26] has been designed with a sampling frequency of 50kHz and its coefficients were found using the *Filter Designer* tool from Matlab[8]. These parameters make the pass band very narrow and close to the stop band; the filter is difficult to implement because it may require high internal precision to achieve reasonably accurate results.

For each example, the results of the word-length allocation problem are shown in Table I. Due to a small number of parameters (up to 22, only), the optimal solution has been computed using `Artelys-Knitro` solver. For the single-output examples ($p = 1$), the sub-optimal word-lengths $\boldsymbol{w}_{\text{eq}}$ are good approximations close to the optimal. It is not the case when $p > 1$, since we need to take the maximum word-length satisfying the constraints for each output. We can also remark that $\widetilde{\boldsymbol{w}}$ is very often low (except for $\boldsymbol{x}_9$ of the first example, where $\overline{\boldsymbol{\zeta}}_9 \approx 63.412$ that is relatively close to a power of 2). The $\boldsymbol{\delta}$ parameters are always equal to 0 in our setup, except for $\boldsymbol{y}_7$ and $\boldsymbol{y}_3$ (second example) where $\boldsymbol{w}_{\text{opt}} < \widetilde{\boldsymbol{w}}$.

Even if the size (number of variables and constraints) of the instances we considered is relatively small, the word-length optimization problem still difficult to solve. Solving instances featuring more than 50 variables and 10 constraints is a very challenging task.

Notice that a comparison with classical statistic approach [6], [7], [8] is not really possible since they do not provide error bound, thus their constraints are less restrictive.

[7] The real coefficients of these examples can be found in our code: `https://github.com/fixif/examples/blob/master/ARITH26/ARITH26.py`.

[8] https://www.mathworks.com

	oscillation controller												random controller												low-pass filter						
	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$	$\boldsymbol{x}_6$	$\boldsymbol{x}_7$	$\boldsymbol{x}_8$	$\boldsymbol{x}_9$	$\boldsymbol{x}_{10}$	y	$f(\boldsymbol{w})$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{y}_1$	$\boldsymbol{y}_2$	$\boldsymbol{y}_3$	$\boldsymbol{y}_4$	$\boldsymbol{y}_5$	$\boldsymbol{y}_6$	$\boldsymbol{y}_7$	$f(\boldsymbol{w})$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_4$	y	$f(\boldsymbol{w})$
$\boldsymbol{m}$	9	8	9	9	8	8	8	6	6	2	4		6	6	5	6	16	15	16	16	16	15	16		43	43	43	43	43	5	
$\widetilde{\boldsymbol{w}}$	3	3	3	3	3	3	3	4	8	3	3		3	4	3	6	4	3	6	3	4	3	6		3	3	3	3	3	3	
$\boldsymbol{w}_{\text{opt}}$	17	15	17	17	15	15	15	11	11	3	13	**149**	13	11	14	12	4	3	5	4	4	3	5	**78**	52	55	55	55	53	14	**284**
$\boldsymbol{w}_{\text{uni}}$	16	16	16	16	16	16	16	16	16	16	16	**176**	31	31	31	31	31	31	31	31	31	31	31	**341**	54	54	54	54	54	54	**324**
$\boldsymbol{w}_{\text{eq}}$	17	15	18	17	15	15	15	12	12	4	14	**154**	33	32	34	32	26	25	26	26	26	25	26	**311**	53	55	56	55	53	14	**286**

TABLE I: Optimal word-length allocation for the three examples: $\boldsymbol{w}_{\text{opt}}$ is the result of the general optimization problem, $\boldsymbol{w}_{\text{uni}}$ the result of the uniform word-length problem and $\boldsymbol{w}_{\text{eq}}$ of the uniformly distributed budget error problem.

VIII. Conclusion

In this article, we have shown how to reliably implement State-Space filters or controllers with Fixed-Point arithmetic. The Most Significant Bit position of each variable has been computed with respect to the word-lengths used in order to prevent any overflow. Then, using a Sum of Products by constants that provides last bit accurate, we have performed a full error analysis of the implementation. Finally, the optimal word-length allocation problem has been defined, and solved using appropriate heuristics.

This work did not yet consider the last part of the filter-to-code flow described in see Figure 1. But thanks to code generators like FloPoCo, this can be done in order to provide last bit accuracy FPGA implementations. From them, it will be possible to measure interesting values, such as the number of LUTs or the power consumption, and then use them as black-box objective function values for the optimization process (the current result will then be used as initial value for the solver).

Moreover, we also plan to extend this work to the full class of linear time invariant algorithms, using the unifying framework proposed in [27] and then finally be able to compare all the possible implementations under the same accuracy constraint.

References

[1] B. Widrow, I. Kollár, and M. Liu, "Statistical analysis of amplitude quantized sampled-data systems," in *IEEE Trans. on Instrumentation and Measurement*, vol. 45, no. 6, 1995, pp. 353–361.

[2] L. Jackson, "On the interaction of roundoff noise and dynamic range in digital filters," *Bell System Technical Journal*, vol. 49, 01 1970.

[3] A. Fettweis, "Roundoff noise and attenuation sensitivity in digital filters with fixed-point arithmetic," *Circuit Theory, IEEE Transactions on*, vol. 20, pp. 174 – 175, 04 1973.

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

[5] A. Volkova, M. Istoan, F. De Dinechin, and T. Hilaire, "Towards hardware iir filters computing just right: Direct form i case study," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 597–608, April 2019.

[6] O. Sarbishei, K. Radecka, and Z. Zilic, "Analytical optimization of bit-widths in fixed-point LTI systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, 2012.

[7] D. Boland and G. Constantinides, "Word-length optimization beyond straight line code," in *ACM Field Programmable Gate Arrays*, 2013.

[8] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, pp. 1432–1442, 2003.

[9] S. Vakili, J. M. P. Langlois, and G. Bois, "Enhanced precision analysis for accuracy-aware bit-width optimization using affine arithmetic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1853–1865, 2013.

[10] 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, November 2007.

[11] A. Volkova, T. Hilaire, and C. Lauter, "Determining fixed-point formats for a digital filter implementation using the worst-case peak-gain measure," in *Asilomar Conf. Signals, Systems and Computers*, 2015.

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

[13] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing (2Nd Ed.)*. Prentice-Hall, Inc., 1999.

[14] K. Ogata, *Discrete-time Control Systems*, 3rd ed. Prentice Hall International, 2006.

[15] P. J. Antsaklis and A. N. Michel, *Linear systems*. Birkhauser,, 2006.

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

[17] A. Volkova, T. Hilaire, and C. Q. Lauter, "Reliable evaluation of the Worst-Case Peak Gain matrix in multiple precision," in *22nd IEEE Symposium on Computer Arithmetic, Lyon, France.*, 2015.

[18] D. Gallois-Wong, S. Boldo, and T. Hilaire, "A coq formalization of digital filters," in *11th Conference on Intelligent Computer Mathematics (CICM)*, Aug. 2018, pp. 87–103.

[19] A. Volkova, T. Hilaire, and C. Lauter, "Arithmetic approaches for rigorous design of reliable Fixed-Point LTI filters," Nov. 2018, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01918650

[20] L. Jackson, J. Kaiser, and H. McDonald, "An approach to the implementation of digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 16, no. 3, pp. 413–421, Sep. 1968.

[21] B. Lopez, T. Hilaire, and L.-S. Didier, "Formatting bits to better implement signal processing algorithms," in *4th int. Conf. on Pervasive and Embedded Computing and Communication Systems*, 2014.

[22] K. Chapman, "Fast integer multipliers fit in FPGAs," *EDN magazine*, no. 10, p. 80, 1993.

[23] M. Wirthlin, "Constant coefficient multiplication using look-up tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, 2004.

[24] C. A. Floudas, *Nonlinear and Mixed-Integer Optimization*. Oxford University Press,, 1995.

[25] 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, November 2003.

[26] D. Schlichthärle, *Digital Filters: Basics and Design, 2nd edition*. Springer, 2011.

[27] 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, August 2007.