Reliable fixed-point implementation of linear data-flows

Thibault Hilaire, Anastasia Volkova, Maminionja Ravoson
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
Email: first_name.last_name@lip6.fr

***Abstract*—In this article, we propose a complete methodology to implement a signal processing or control-engineering algorithm described with a linear data-flow into numerical code using fixed-point arithmetic. Our approach is based on a reliable determination of the Worst-Case Peak gain of a filter as well as on rigorous error analysis of roundoff error propagation. It guarantees that no overflow will occur and that the output error due to the finite precision implementation is less than a given bound.**

Without loss of generality, we consider the linear data-flows given in the form of Simulink block diagram. It is first transposed into an internal matrix-based representation and then the reliable evaluation of the magnitudes of each internal variable is performed. Our approach allows determining the minimum word-length required to achieve a given accuracy. Finally, the methodology is illustrated with numerical examples.

I. Introduction

The implementation of embedded algorithms is a tedious task for many reasons. Due to the limited resources available (power, memory, computing units, processing time, etc.), the transformation from a mathematical model (whatever its form: equations, transfer function, data-flow graph, etc.) into embedded code is not straightforward. One principal pitfall concerns the accuracy part: generally finite precision arithmetic, like the Fixed-Point (FxP) arithmetic is used, and the computational errors induced by limited bit-width may be significant.

Moreover, in FxP arithmetic, the developer has to deal with the binary-point alignment, fixed-point conversion, etc., while in floating-point arithmetic this is done by the hardware.

The two main bottlenecks for the Fixed-Point implementation are the setting of the FxP formats (they depend on the magnitude of each variable) and the roundoff error analysis. We are interested in determining the impact of roundoff errors on the output(s) of the implemented system, therefore we need to rigorously analyse the difference between the exact system and the implemented one. These two issues may be not so easy to deal reliably with, especially in the cases where the system has an internal feedback, like for the Infinite Impulse Response (IIR) filters.

In practice, the FxP conversion is done using simulations, typically using Matlab/Simulink tools[1]:

a) run the system with some inputs (using floating-point double precision arithmetic) and determine the magnitudes of all variables;

This work has been partially sponsored by french ANR agency under grant No ANR-13-INSE-0007-04 MetaLibm.

[1]http://www.mathworks.com

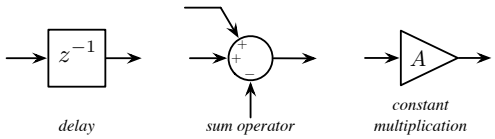

Fig. 1. Blocks of linear data-flows.

b) deduce the FxP formats and the FxP operations;
c) run the FxP implementation of the system and compare it with the simulations from step a);
d) if the result is not convincing, increase the word-length, and return to step b).

This process is unsatisfactory for various reasons. First, a finite-precision (floating-point) simulation is taken as a reference for comparison which can be inaccurate. Secondly, the input sequences to be tested are often not representative. Moreover, these non-exhaustive and time-consuming simulations do not give any **reliable** guarantee on the FxP implementation. The saturation technique is widely used to deal with overflows, however it does not ensure the correct value of the output.

In our work we consider only **linear** algorithms, *i.e.* data-flow systems built only with the sum operators ($+$ or $-$ operations), constant multiplications and delays (see Fig. 1). The examples of such algorithms are the IIR and FIR (Finite Impulse Response) filters, Linear Time Invariant (LTI) controllers, etc.

In order to overcome the drawbacks of the simulation-based techniques and provide mathematical guarantees on the FxP implementation, we use an analytical data-flow representation called Specialized Implicit Framework (SIF) [1].

We propose a technique on representation of the linear data-flows with SIF formalism. Then, we can easily determine the inputs-outputs relationship as transfer functions, and then establish analytically the impact of the finite precision implementation. The implementation flow is encapsulated into an automatized code generator. A rigorous approach on determining the magnitudes and the Fixed-Point formats has been developed for the generator [2], [3]. Furthermore, without using any simulations we can optimize the word-lengths within the algorithm evaluation to satisfy the output precision constraints.

The paper is organized as follows. Our complete flow from data-flow system to fixed-point code is described in Section II with some necessary prerequisites. Section III focuses on the transformation of Simulink linear data-flow systems to our

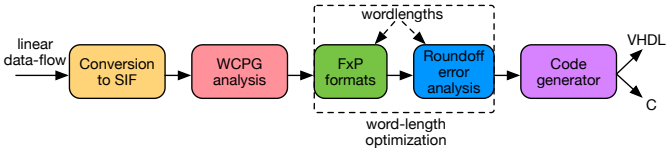

Fig. 2. Data-flow implementation steps.

Specialized Implicit Framework. The fixed-point conversion and the complete roundoff analysis are done in Section IV. Finally, some illustrative examples of fixed-point implementation under precision constraints are given in Section V.

Notation: throughout this paper, real numbers are in lowercase, column vectors in lowercase boldface and matrices in uppercase boldface; $\boldsymbol{I}_n$ is a $n \times n$ identity matrix.

II. Methodology and prerequisites

A. Methodology

In order to provide rigorous error analysis of finite-precision implementation effects, we propose to use an analytical representation of a linear system, SIF, which is described in details in Section II-B. This is a matrix representation, which can be obtained for any data-flow graph. It preserves all the properties of the system but is easier to operate with, in our context.

Our approach consists of following. First, we represent the data-flow with SIF formalism. Then, we deduce the magnitudes of all variables within the system. In comparison to the commonly used approach, we do not use any simulations to compute them. Considering that the inputs are bounded, our approach is based on reliable evaluation of the maximum amplification of the output of a system using the Worst-Case Peak Gain theorem [2]. This gives us a theoretical guarantee on the computed magnitudes and does not depend on a specific choice of inputs. See Section II-D for detailed description.

Given an output error bound and word-length constraints for all variables we choose FxP formats for the implementation (using **reliably** computed magnitudes) such that the output code ensures that no overflow occurs.

Finally, we deduce the parameters for algorithm evaluation in FxP such that the computed output satisfies an *a priori* given error bound for each iteration of the system. See Section IV-B.

However, the propagation of the roundoff errors through recursive systems is non-linear and can have a significant impact on the output. We take into account the roundoff errors propagation and provide a rigorous bound on the difference between the implemented and the ideal systems.

Therefore, our approach can be represented as following steps, as shown in Fig. 2:

a) transform a data-flow to the SIF formalism;
b) **rigorously** compute magnitudes using the Worst-Case Peak-Gain analysis;
c) given word-length constraints determine the FxP formats based on rigorous magnitudes, while taking into account the propagation of the computational errors;
d) generate the software or hardware code.

Moreover, if the word-lengths are not fixed, the Fixed-Point Formats computed on step c) can be optimized to meet an output error-bound criteria without using any simulations.

B. Specialized Implicit Framework

The Specialized Implicit Framework has been first proposed in [1] as a unifying tool in order to describe and encompass all the possible realizations of a given transfer function (like the direct forms, state-spaces, cascade or parallel decompositions, lattice filters, etc.). It allows the study and comparison of their finite precision effects. SIF is an extension of the state-space realization, modified in order to allow chained Sum-of-Products (SoP) operations.

In fact, all linear data-flows (those with delays, 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 [1]. Both the Single Input Single Output (SISO) and the Multiple-Inputs Multiple Outputs (MIMO) can be described with SIF.

Denote $\boldsymbol{u}(k)$ and $\boldsymbol{y}(k)$ the vector of q inputs and the vector of p outputs respectively. The n variables that are stored from one step to the other are in the state vector $\boldsymbol{x}(k)$, while the l intermediate results are collected in the vector $\boldsymbol{t}(k)$. Then, the SIF is the following set of equations:

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

The vector $\boldsymbol{t}(k)$ is not used for computations at step k, which characterizes the concept of intermediate variables.

The matrix $\boldsymbol{J}$ is 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 describing the specific order of the computation. Therefore, the general algorithm to compute the SIF is:

$$\begin{aligned} \boldsymbol{J}\boldsymbol{t}(k+1) &\leftarrow \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) &\leftarrow \boldsymbol{K}\boldsymbol{t}(k+1) + \boldsymbol{P}\boldsymbol{x}(k) + \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &\leftarrow \boldsymbol{L}\boldsymbol{t}(k+1) + \boldsymbol{R}\boldsymbol{x}(k) + \boldsymbol{S}\boldsymbol{u}(k) \end{aligned} \quad (2)$$

The matrices $\boldsymbol{J}$, $\boldsymbol{K}$ and $\boldsymbol{L}$ allow us to describe the sequence of computations. For example, $\boldsymbol{y} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{x}$ can be computed as $\boldsymbol{y} = (\boldsymbol{M}_2\boldsymbol{M}_1)\boldsymbol{x}$ or as $\boldsymbol{y} = \boldsymbol{M}_2(\boldsymbol{M}_1\boldsymbol{x})$. The latter expression will be described as

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{M}_2 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{t} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_1 \\ \boldsymbol{0} \end{pmatrix} \boldsymbol{x}, \quad (3)$$

with $\boldsymbol{t}$ holding the intermediate value $\boldsymbol{M}_1\boldsymbol{x}$.

The same approach will be used to transform any linear data-flow into a SIF in Section III.

Throughout the paper we consider that the computations associated to the equations (1) are ordered from top to bottom[2], associated in a one to one manner to the Algorithm (2).

[2]Of course, there is no need to consider the null coefficients. The associated computations may be removed.

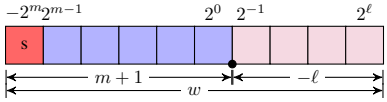

Fig. 3. Representation of a fixed-point number in format (m, ℓ).

It can be easily established that equation (1) is equivalent in infinite precision to the state-space filter $(\boldsymbol{A_Z}, \boldsymbol{B_Z}, \boldsymbol{C_Z}, \boldsymbol{D_Z})$:

$$\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A_Z x}(k) + \boldsymbol{B_Z u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C_Z x}(k) + \boldsymbol{D_Z u}(k) \end{cases} \tag{4}$$

with:

$$\begin{aligned} \boldsymbol{A_Z} &= \boldsymbol{K J^{-1} M} + \boldsymbol{P}, \quad & \boldsymbol{B_Z} &= \boldsymbol{K J^{-1} N} + \boldsymbol{Q}, \\ \boldsymbol{C_Z} &= \boldsymbol{L J^{-1} M} + \boldsymbol{R}, \quad & \boldsymbol{D_Z} &= \boldsymbol{L J^{-1} N} + \boldsymbol{S}. \end{aligned} \tag{5}$$

However, equation (4) corresponds to a different set of coefficients than the one in (1). Therefore, while in infinite precision (1) and (4) are equivalent, in finite precision they have different numerical properties. Thus, the SIF not only describes the input/output relationship but also fully captures properties of computational algorithm.

The main interest of such a framework is that it allows expressing various algorithms of the same filter in a unifying form, in order to study and compare the effect of the finite precision arithmetic on them, and then to choose the best FxP algorithm to implement this filter. All the classical measures used to evaluate the impact of the quantization of the coefficients (sensitivity-based measure like in [4], [5], [6]) and the impact of the roundoff errors ([5], [7], [8], [9], [10]) have been extended to the SIF [11], [12].

C. Fixed-Point Arithmetic

In this paper, only the signed fixed-point arithmetic with 2's complement representation is used. Let z be a w-bit fixed-point number:

$$z = -2^m z_m + \sum_{i=\ell}^{m-1} 2^i z_i \tag{6}$$

where m and ℓ are the positions of the most significant bit (MSB) and the least significant bit (LSB) of z, respectively. Let the couple (m, ℓ) denote the Fixed-Point format of z (see Fig. 3). The word-length w can be obtained with:

$$w = m - \ell + 1, \tag{7}$$

In order to convert a non-zero real number r into a fixed-point number $\tilde{r}$, the first step is to evaluate the binade in which r resides, and more precisely the position of the MSB of $\tilde{r}$. The main formula to compute m_r is given by:

$$m_r = \begin{cases} \lceil \log_2(r) \rceil & \text{if } r > 0 \\ \lfloor \log_2(-r) \rfloor + 1 & \text{if } r < 0 \end{cases} \tag{8}$$

For some very special cases (where r rounded to $\tilde{r} \in [-2^{m_r-1}; 2^{m_r-1} - 2^{\ell_r}]$ while r is outside this interval), (8) needs to be adjusted (see [13]). Then the fractional part ℓ_r is obtained from (7) and we have:

$$\tilde{r} = \lfloor r \cdot 2^{-\ell_r} \rceil \cdot 2^{\ell_r}, \tag{9}$$

where $\lfloor \cdot \rceil$ is the round-to-nearest-integer operator. In the machine r is represented as a w-bit signed integer $R = \lfloor r \cdot 2^{-\ell_r} \rceil$.

D. The Worst-Case Peak Gain theorem

The following theorem is used to provide the maximum possible value for the outputs of the algorithm, and thus define their FxP formats (using (8)) with a guarantee that no overflow will occur. It is also used to determine the impact of the internal roundoff error on the output (see Section IV-C).

Let $\mathcal{H} := (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ be a q-input p-output Bounded-Input Bounded-Output stable MIMO LTI filter in state-space representation:

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

with $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{B} \in \mathbb{R}^{n \times q}$, $\boldsymbol{C} \in \mathbb{R}^{p \times n}$ and $\boldsymbol{D} \in \mathbb{R}^{p \times q}$.

Theorem 1 (Worst-Case Peak Gain theorem) Let $\mathcal{H}$ be a state-space system. If an input $\{\boldsymbol{u}(k)\}_{k \geqslant 0}$ is known to be bounded by $\bar{\boldsymbol{u}}$ ($\forall k \geqslant 0, \quad |\boldsymbol{u}_i(k)| \leqslant \bar{\boldsymbol{u}}_i, \quad 1 \leqslant i \leqslant q$), then the output $\{\boldsymbol{y}(k)\}_{k \geqslant 0}$ will be bounded[3] iff the spectral radius $\rho(\boldsymbol{A})$ is strictly less than 1.

In that case, the output is (element-by-element) bounded by

$$\forall k, \quad |\boldsymbol{y}(k)| \leqslant |\langle\!\langle \mathcal{H} \rangle\!\rangle \bar{\boldsymbol{u}}|, \tag{11}$$

where $\langle\!\langle \mathcal{H} \rangle\!\rangle$ is the Worst-Case Peak Gain (WCPG) matrix of the system [15], [16]. It can be computed as

$$\langle\!\langle \mathcal{H} \rangle\!\rangle \triangleq |\boldsymbol{D}| + \sum_{k=0}^{\infty} |\boldsymbol{C A^k B}|. \tag{12}$$

Moreover, for any $\varepsilon > 0$, there always exists a finite input sequence $\{\boldsymbol{u}(k)\}_{0 \leqslant k \leqslant N}$ such that the inequality (11) is almost attained for one output i, *i.e.*

$$\left| \boldsymbol{y}_i(N) - \left(\langle\!\langle \mathcal{H} \rangle\!\rangle \bar{\boldsymbol{u}} \right)_i \right| < \varepsilon. \tag{13}$$

There exist numerous other approaches to determine the output interval, such as involving Affine Arithmetic [17], which may lead to overestimations, or simulations [18], which in turn are inefficient and not rigorous. It can be shown that the WCPG approach gives the smallest interval containing all possible values of $\boldsymbol{y}(k)$. Moreover, in [2] an algorithm to evaluate the WCPG at arbitrary precision was proposed.

III. Simulink-to-SIF conversion

A. Simulink block diagram

Simulink, by Mathworks, is a widely used environment for model-based design of dynamic systems. It uses mainly a graphical block diagram to describe the model.

Internally, the block diagram is stored in an xml file (more specifically in `.slx` format). In this format the `<System>` tag contains the model description, and the `<Block>` and `<Line>` tags inside hold blocks of elements and their interconnections.

Consider a very simple data-flow diagram given on Fig. 4. It consists of standard gain, delay and sum blocks. All the

[3]This property is known as the Bounded Input Bounded Output (BIBO) stability [14].

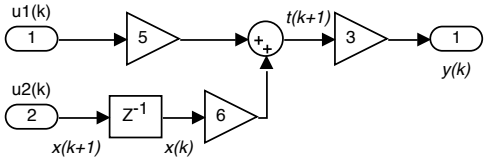

Fig. 4. A very simple Simulink block diagram.

information on the block parameters can be easily retrieved by parsing the xml file.

B. The conversion algorithm

The SIF representation is a set of equations which compute the output, the states and the intermediate variables from the value of inputs, current states and the intermediate variables, and where each equation is a Sum-of-Products (SoPs).

In order to systematically transform any Simulink linear data-flow model to the SIF representation, we assume following rules. Since each delay element represents a computation that is saved from one time instance to another, in SIF we note it as a state variable. The gain elements are naturally interpreted in SIF as coefficients of the variables in corresponding SoPs. The outputs of gain and sum operators are naturally considered as temporary variables in SIF (unless they feed a delay or system output) since they are not stored from one system iteration to another. After noting all the variables of the system, their corresponding SoPs can be easily formed. Therefore, a system of equations corresponding to the data-flow can be obtained.

By design, most of the time there is a chain of sum blocks in a data-flow. For a finite-precision implementation the order of computations has a significant impact on the precision of the result. To apply our SoP computation technique presented in IV-B, merging all directly cascading sum blocks is necessary. Moreover, in case of a subsystem present in the diagram, a care was taken to flatten the design before getting blocks equations. For instance, the simple example of a data-flow diagram given in Fig. 4 corresponds to the following system of equations:

$$\begin{cases} t(k+1) &= 6 \cdot x(k) + 5 \cdot u_1(k) \\ x(k+1) &= 1 \cdot u_2(k) \\ y &= 3 \cdot t(k+1) \end{cases}$$

From this system of equations, it is then straightforward to identify matrices $\boldsymbol{J}$, $\boldsymbol{K}$, $\boldsymbol{L}$, $\boldsymbol{M}$, $\boldsymbol{N}$, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{S}$ of the SIF. An important detail is to preserve the order in which the variables appear in SoPs in order to match exactly the data-flow diagram. In our toy example, we have $\boldsymbol{J} = \begin{pmatrix}1\end{pmatrix}$, $\boldsymbol{L} = \begin{pmatrix}3\end{pmatrix}$, $\boldsymbol{M} = \begin{pmatrix}6\end{pmatrix}$, $\boldsymbol{N} = \begin{pmatrix}0 & 5\end{pmatrix}$, $\boldsymbol{P} = \begin{pmatrix}0\end{pmatrix}$, $\boldsymbol{Q} = \begin{pmatrix}0 & 1\end{pmatrix}$, and the other matrices are null.

However, the order of labeling the temporary variables during the conversion can compromise the lower triangularity of the matrix $\boldsymbol{J}$. Since $\boldsymbol{J}$ represents the order of computation of the intermediate variables within the data-flow and dependencies between them, it can be interpreted as an adjacency matrix for a directed acyclic graph. Therefore, a topological sort algorithm can be applied to determine the correct order of computations. Here the depth-first search algorithm was used to ensure the lower triangular form of $\boldsymbol{J}$ along with necessary reorganizations of matrices $\boldsymbol{N}$, $\boldsymbol{K}$ and $\boldsymbol{L}$.

IV. Fixed-point implementation and roundoff error analysis

A. Fixed-Point formats

The first step of the fixed-point implementation is the determination of the FxP format of every internal variable within algorithm computation. Applying the WCPG theorem we can obtain the magnitudes for the outputs of the system. However, we can obtain the intervals for the intermediate and state variables as well [13]. To achieve this we concatenate the output vector $\boldsymbol{y}$ with vectors $\boldsymbol{t}$ and $\boldsymbol{x}$, and make necessary concatenations in the right side of (10). Thus, instead of considering the filter $\mathcal{H}$ we consider the filter $\mathcal{H}_u = (\boldsymbol{A_Z}, \boldsymbol{B_Z}, \boldsymbol{N}_1, \boldsymbol{N}_2)$ with

$$\boldsymbol{N}_1 \triangleq \begin{pmatrix} \boldsymbol{J}^{-1}\boldsymbol{M} \\ \boldsymbol{A_Z} \\ \boldsymbol{C_Z} \end{pmatrix}, \boldsymbol{N}_2 \triangleq \begin{pmatrix} \boldsymbol{J}^{-1}\boldsymbol{N} \\ \boldsymbol{B_Z} \\ \boldsymbol{D_Z} \end{pmatrix}. \tag{14}$$

Then, applying the WCPG theorem upon the filter $\mathcal{H}_u$ we obtain the magnitudes $\bar{\boldsymbol{t}}, \bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$ by

$$\begin{pmatrix} \bar{\boldsymbol{t}} \\ \bar{\boldsymbol{x}} \\ \bar{\boldsymbol{y}} \end{pmatrix} = \langle\!\langle \boldsymbol{\mathcal{H}}_u \rangle\!\rangle \, \bar{\boldsymbol{u}}. \tag{15}$$

The MSB positions $\boldsymbol{m_t}$, $\boldsymbol{m_x}$ and $\boldsymbol{m_y}$ are obtained with (8).

The evaluation of the magnitude can be improved by taking into account the propagation of roundoff errors that may, in some rare cases, yield an overflow. An algorithm for determining the FxP formats with a non-overflow guarantee was presented in [3].

B. Sum-of-Products

As shown in Section II-B, the evaluation of linear data flow involves the evaluation of some sum-of-products, *i.e.* sums of variables (inputs, states and intermediate variables) multiplied by constant coefficients (those of the gain blocks). Let us consider here one sum-of-product (SoP):

$$s = \sum_{i=1}^{N} \boldsymbol{c}_i \cdot \boldsymbol{v}_i = \sum_{i=1}^{N} \boldsymbol{p}_i, \tag{16}$$

where $\boldsymbol{p}_i$ denotes the product $\boldsymbol{c}_i \times \boldsymbol{v}_i$, $\boldsymbol{c}_i$ is a constant and $\boldsymbol{v}_i$ a variable, for $1 \leqslant i \leqslant N$.

In our context, the MSB positions of the variables are known (deduced using the reliably computed magnitudes) and the MSB of the constants are computed using (8). In the SIF the sum-of-product results are stored either in the intermediate, state or output variables, so their MSB are also known. Thus, it is easy to deduce the FxP format of the accumulation in order to guarantee a roundoff error less than a given bound.

Denote (m_s, ℓ_s) the required FxP format for the result. Then from [12], in order to guarantee a faithful rounding (*i.e.* the SoP evaluation error bounded by 2^{ℓ_s}) it is sufficient to perform the multiplication and accumulation with the format $(m_s, \ell_s - g)$ with $g = \lceil \log_2 N \rceil$. The g extra guard bits are discarded after the accumulation is performed, as shown in Fig. 5.

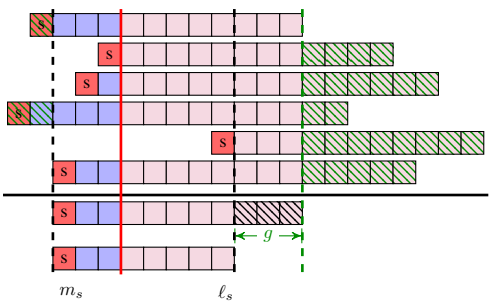

Fig. 5. Sum-of-Products performed with FxP format $(m_s, \ell_s - g)$.

C. Roundoff error analysis

The evaluation of each SoP in fixed-point arithmetic may provide an additional error. When implemented, the Algorithm (2) is changed to

$$\begin{aligned} \boldsymbol{Jt}^*(k{+}1) &\leftarrow \boldsymbol{M}\boldsymbol{x}^*(k) + \boldsymbol{N}\boldsymbol{u}(k) &&+\boldsymbol{\varepsilon_t}(k) \\ \boldsymbol{x}^*(k{+}1) &\leftarrow \boldsymbol{K}\boldsymbol{t}^*(k{+}1) + \boldsymbol{P}\boldsymbol{x}^*(k) + \boldsymbol{Q}\boldsymbol{u}(k) &&+\boldsymbol{\varepsilon_x}(k) \\ \boldsymbol{y}^*(k) &\leftarrow \boldsymbol{L}\boldsymbol{t}^*(k{+}1) + \boldsymbol{R}\boldsymbol{x}^*(k) + \boldsymbol{S}\boldsymbol{u}(k) &&+\boldsymbol{\varepsilon_y}(k) \end{aligned} \quad (17)$$

where $\boldsymbol{\varepsilon_t}(k)$, $\boldsymbol{\varepsilon_x}(k)$ and $\boldsymbol{\varepsilon_y}(k)$ 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; and $\boldsymbol{\ell}$ and $\boldsymbol{w}$ the vectors aggregating the LSB positions and word-lengths of the intermediate variables, states and outputs (in that order), respectively.

From the Section IV-B it follows that we can compute SoPs with faithful rounding. Using equations (7) and (15) we obtain that the errors $\boldsymbol{\varepsilon}(k)$ are element-by-element bounded by $\bar{\boldsymbol{\varepsilon}}$:

$$\bar{\boldsymbol{\varepsilon}} \triangleq 2^{\boldsymbol{\ell}} = 2 \left\lceil \langle\!\langle \boldsymbol{\mathcal{H}}_u \rangle\!\rangle \, \bar{\boldsymbol{u}} \right\rceil_2 \times 2^{-\boldsymbol{w}}, \quad (18)$$

where $\lceil x \rceil_2$ is the operator that gives the power-of-2 immediately greater than x, $\times$ is the entrywise product and assuming the power of 2 of a vector is done element-by-element.

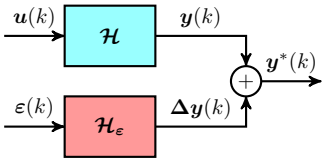

Fig. 6. Equivalent system, with errors separated.

In order to capture the effects of the FxP implementation we must take into account the propagation of the roundoff errors through the data-flow.

Due to the linearity of the system $\boldsymbol{\mathcal{H}}$, we can express the implemented system as the initial system with an error $\boldsymbol{\Delta y}(k)$ added to the outputs, as shown in Fig. 6. The vector $\boldsymbol{\Delta y}(k)$ shows the propagation of the error $\boldsymbol{\varepsilon}(k)$ through the special error-system $\boldsymbol{\mathcal{H}_\varepsilon}$. It is obtained by performing a difference between the implemented system (17) and the exact one. Its state-space representation is $(\boldsymbol{A_Z}, \boldsymbol{M}_1, \boldsymbol{C_Z}, \boldsymbol{M}_2)$, with:

$$\boldsymbol{M}_1 \triangleq \begin{pmatrix} \boldsymbol{K}\boldsymbol{J}^{-1} & \boldsymbol{I}_n & \boldsymbol{0} \end{pmatrix}, \quad \boldsymbol{M}_2 \triangleq \begin{pmatrix} \boldsymbol{L}\boldsymbol{J}^{-1} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix}. \quad (19)$$

Using such a decomposition and the bound (18) on the roundoff errors, we can apply the WCPG theorem upon the system $\mathcal{H}_\varepsilon$ to obtain the maximum amplification of the errors to the output. Then, for all time instances k the difference between the exact output $\boldsymbol{y}(k)$ and the implemented output $\boldsymbol{y}^*(k)$ is bounded by $\overline{\boldsymbol{\Delta y}}$:

$$\overline{\boldsymbol{\Delta y}} \triangleq 2 \, \langle\!\langle \boldsymbol{\mathcal{H}_\varepsilon} \rangle\!\rangle \left(\left\lceil \langle\!\langle \boldsymbol{\mathcal{H}}_u \rangle\!\rangle \, \bar{\boldsymbol{u}} \right\rceil_2 \times 2^{-\boldsymbol{w}} \right). \quad (20)$$

Therefore, we have a reliable analytical bound on the data-flow implementation error, which takes into account the (possible) non-linear propagation of roundoff errors. As we can see, our approach does not involve any simulations.

D. Word-length optimization

We can note that the output error bound $\overline{\boldsymbol{\Delta y}}$ depends only on the word-lengths vector $\boldsymbol{w}$. For hardware implementation, where it is possible to have different word-lengths for each variable and operator (*multiple word-length paradigm*), it should be possible to find the word-lengths that minimize a criteria, like area, power consumption, etc., while satisfying a constraint on the output error.

Although this is out of the scope of this article, such optimization problem can be solved when the cost function is not too complicated: for linear cost function (like the sum of the word-lengths), it can be solved with some Mixed-Integer Non Linear Programming solver, like `Bonmin`[4].

When considering homogeneous word-lengths, the minimum word-length $w_{\min}$ that guarantees that the output error is less than a given $\boldsymbol{\varepsilon}$ is

$$w_{\min} = \left\lceil \log_2 \left(\frac{\langle\!\langle \boldsymbol{\mathcal{H}_\varepsilon} \rangle\!\rangle \left\lceil \langle\!\langle \boldsymbol{\mathcal{H}}_u \rangle\!\rangle \, \bar{\boldsymbol{u}} \right\rceil_2}{\boldsymbol{\varepsilon}} \right) \right\rceil + 1. \quad (21)$$

It can be used to compare realizations.

V. Illustrative Examples

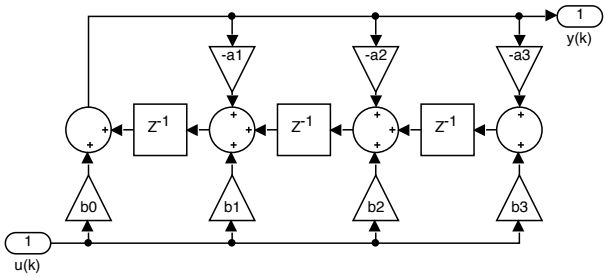

Fig. 7. Direct Form II transposed data-flow.

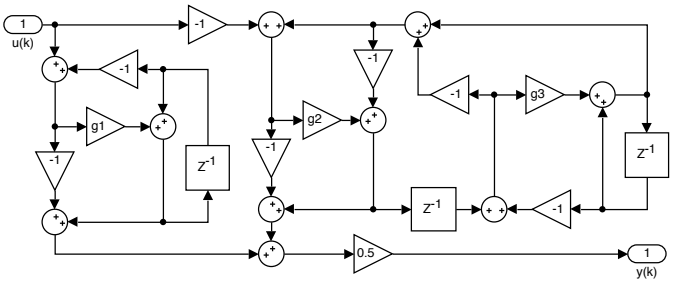

Fig. 8. Lattice Wave Digital Filter data-flow.

[4]Basic Open-source Nonlinear Mixed INteger programming, https://projects.coin-or.org/Bonmin

To illustrate our flow from a linear data-flow diagram to reliable fixed-point implementation, we use two realizations of the same 3rd order Butterworth filter with cutoff frequency 0.2. The transfer function was designed with Matlab. The first realization is a Direct Form II transposed, which requires 7 coefficients to be stored. The second one is a Lattice Wave Digital Filter [19], which requires only 3 coefficients. The corresponding data-flows are given in Fig. 7 and Fig. 8.

The two data-flow graphs were translated into our internal framework using the Simulink-to-SIF converter described in the Section III. Then, for a given $\bar{u}$, equal here to 5.25, the MSBs of the intermediate variables, states and output were deduced using (15).

For a given homogeneous word-length w (*i.e.* the same w for all variables), the error for each sum-of-products was deduced along with the final output error, which takes into account the propagation of the computational errors.

Finally, instead of exhibiting the output error with respect to the variable word-lengths, we prefer to use a more realistic indicator, like the total number of bits used for the computations as shown in Fig. 9. We can observe that Lattice Wave realization requires less coefficients to be stored and, consequently, slightly smaller total amount of bits than the Direct Form II transposed to attain the same output error. In our example, the Direct Form II transposed realization requires 17 bits per variable to ensure an output error less than $\varepsilon = 10^{-2}$, whereas 16 bits per variable are enough for the Lattice Wave one.

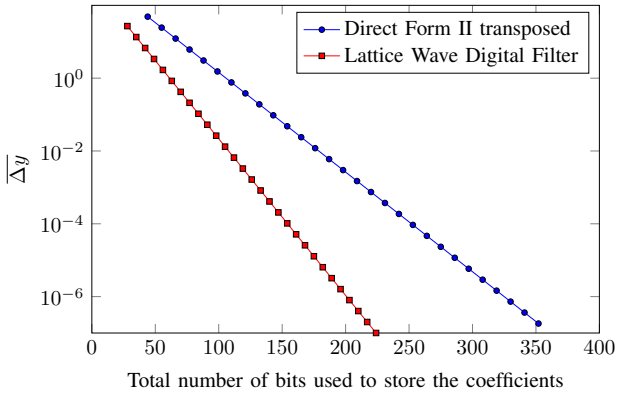

Fig. 9. Output error when the word-length w goes from 4 to 32.

VI. Conclusion

We proposed a reliable method to implement linear data-flows with fixed-point arithmetic, while ensuring that no overflow occurs and controlling the output error. It is based on an analytical description that describes all the computations of a data-flow, and allows performing analytical roundoff error analysis, thanks to the Worst-Case Peak Gain theorem.

The output error depends exclusively on the word-lengths used to store the intermediate variables and states, and to perform the sum-of-products. However, these variables have different impact on the overall output error. Therefore, as a perspective, we should focus on solving the word-length optimization problem, in order to find the word-lengths that minimize a surface or power consumption criteria, while guaranteeing an output error constraint. Finally, the corresponding code (C or VHDL) will be generated with third-party tools, like FloPoCo[5] [20] or Stratus[6] [21].

References

[1] 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.

[2] 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*, June 2015, pp. 96–103.

[3] ——, "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*, Nov 2015, pp. 737–741.

[4] G. Li, "On the structure of digital controllers with finite word length consideration," *IEEE Trans. on Autom. Control*, vol. 43, May 1998.

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

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

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

[8] 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.

[9] T. Hinamoto, H. Ohnishi, and W. Lu, "Roundoff noise minimization of state-space digital filters using separate and joint error feedback/coordinate transformation optimization," in *IEEE Transactions on Circuits and Systems, Fundamental Theory and Applications*, vol. 50, 2003.

[10] D. Menard and O. Sentieys, "A methodology for evaluating the precision of fixed-point systems," in *ICASSP*, 2002, pp. 3152–3155.

[11] 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*, vol. special issue on Quantization of VLSI Digital Signal Processing Systems, January 2011.

[12] B. Lopez, T. Hilaire, and L.-S. Didier, "Formatting bits to better implement signal processing algorithms," in *4th international Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, Lisbon, Portugal, Jan. 2014.

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

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

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

[16] S. P. Boyd and J. Doyle, "Comparison of peak and rms gains for discrete-time systems," *Syst. Control Lett.*, vol. 9, no. 1, pp. 1–6, June 1987.

[17] 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.

[18] S. Kim, K. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems*, vol. 45, November 1998.

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

[20] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[21] S. Belloeil, D. Dupuis, C. Masson, J.-P. Chaput, and H. Mehrez, "Stratus: A procedural circuit description language based upon Python," in *International Conference on Microelectronics*, 2007, pp. 275–278.

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

[6] https://soc-extras.lip6.fr/en/coriolis/stratus/