Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation

Benoit Lopez, Thibault Hilaire, Laurent-Stéphane Didier

Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation

Benoit Lopez, Thibault Hilaire and Laurent-Stéphane Didier
LIP6, University Pierre et Marie Curie (UPMC), Paris, France

Abstract— **The signal processing and control algorithms are widely based on sum-of-products evaluation. In fixed-point arithmetic, the roundoff errors and coefficient quantization may have an important effect on the application's performance and characteristics.**
As part of a global methodology on optimal fixed-point implementation of filters/controllers, this paper formalizes the various implementation schemes for sum-of-products in fixed-point arithmetic and automates the fixed-point code production. The order of the operations are considered, as their bit-width and the fixed-point representation of the coefficients, variables and partial results. Applied to linear filters, the output roundoff noise error is then evaluated and used as a criteria to find out interesting evaluation scheme. An example illustrates the approach.

I. Introduction

The great majority of embedded signal processing or control algorithms is implemented using digital devices such as general purpose micro-controllers, DSP, ASIC or FPGA. The computation on these devices is mainly based on integer arithmetic (rather than floating-point arithmetic) for cost, size and power consumption reasons. The fixed-point arithmetic is used as an approximation of real numbers. Unfortunately the numerical implementation of such algorithms suffers from a deterioration in performance and characteristics, due to the quantization of the embedded coefficients and the roundoff errors occurring in the computations [1], [2], [3].

Classically, the filter implementation problem is tackled in several steps [4]. First, for a given filter or controller, a *good* realization is chosen, such as a direct form, a state-space realization, a cascade decomposition, etc. Criteria based on transfer function sensitivity, or pole sensitivity [1], [5] can be used as indicators of the Finite Word-Length (FWL) effects, in order to compare the realizations. Then, the associated algorithm is realized in fixed-point arithmetic on the dedicated device.

Since the considered algorithms (Linear Time Invariant systems, such as FIR[1] or IIR[2] filters, linear controllers, etc.) are highly regular, we focus in this article on the study of the sum-of-products implementation in fixed-point arithmetic, and the evaluation of the associated roundoff errors. These errors are highly dependent on the order of the additions [6], on each binary-point position and the bit-width of each operator. So, a complete formalization of sum-of-products in fixed-point arithmetic is proposed here. It is much more detailed and accurate than regular approaches used, in the sense that it provides various possibilities, such as a dedicated fixed-point representation for each coefficient, variable and partial result and it supports the multiple word-length paradigm [7].

[1]Finite Impulse Response
[2]Infinite Impulse Response

We do not focus here on the optimal realization problem (see [8]) nor on hardware consideration (*i.e.* how to map the algorithm on a given architecture), but mainly on arithmetic and algorithmic organization of the computations and its effects on precision. The main objective is to formalize all the possible fixed-point implementations of a given sum-of-products in order to exhibit the *interesting* ones (in a sense to define). This part will be used in the global implementation flow from filter/controller to code [4] and is common to hardware and software implementation. It will be even important for FPGA implementation, where fine-grained computing should be exploited to obtain good tradeoff between implementation cost and precision.

The paper is organized as follows. Section II reminds the FWL effects on IIR filter and the dynamic range evaluation. Then, after reminding the fixed-point arithmetic and its associated operations, section III considers the ordered sum-of-products and the propagation rules of the fixed-point format, with different variations. Finally, an illustrative example is given in section IV before conclusion in section V.

II. Finite Word-Length effects and dynamic range evaluation

Let us consider a n-th order IIR filter with h as transfer function:

$$h(z) = \frac{\boldsymbol{b}_0 + \boldsymbol{b}_1 z^{-1} + \cdots + \boldsymbol{b}_n z^{-n}}{1 + \boldsymbol{a}_1 z^{-1} + \cdots + \boldsymbol{a}_n z^{-n}}, \quad \forall z \in \mathbb{C}. \tag{1}$$

This filter is usually realized with the following algorithm

$$y(k) = \sum_{i=0}^{n} \boldsymbol{b}_i u(k-i) - \sum_{i=1}^{n} \boldsymbol{a}_i y(k-i) \tag{2}$$

where $u(k)$ and $y(k)$ are the input and output at step k, respectively.

Of course, this direct form (2) is not the only one possible realization. Some other interesting realizations can be considered, such as state-space realizations [1], realizations with the δ-operator (defined by $\delta \triangleq \frac{q-1}{\Delta}$, where q is the classical

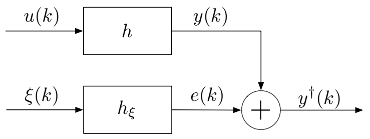


Fig. 1. Equivalent system, with noise extracted

shift operator, and Δ a strictly positive constant [9]), the ρ-Direct Form II transposed (ρDFIIt) [10], the ρ-modal forms, the wave lattice filter, warped filter, and a lot of other specific realizations (LGC, LCW-structures [11], etc.).
The Specialized Implicit Framework (SIF) proposed in [8] can be used as a unifying framework to describe all these realizations. In that context, this work on sum-of-products evaluation scheme will be extended to that framework, in order to be applied on various realizations.

A. Roundoff Noise Analysis

When implemented in finite precision, equation (2) is modified by the addition of some noise $\xi(k)$, and only $y^\dagger$ (the output contaminated with roundoff error) can be computed:

$$y^\dagger(k) = \sum_{i=0}^{n} \boldsymbol{b}_i u(k-i) - \sum_{i=1}^{n} \boldsymbol{a}_i y^\dagger(k-i) + \xi(k) \qquad (3)$$

This added noise depends on:

- the way the computations are organized and realized (the order of the sums, etc.);
- the fixed-point representation of all the signals used in the computations (inputs, outputs, coefficients)
- and the fixed-point representation of each step of the operations (inner signals)

In fixed-point arithmetic, it can be modeled as independent white noise with given first and second order moments (see section III-F).

Proposition 1 *It is possible to express the implemented system as the initial system with a noise $e(k)$ added on the output, as shown in Figure 1.*
Moreover $e(k)$ is the result of the noise $\xi(k)$ through the filter h_ξ defined by:

$$h_\xi(z) = \frac{1}{1+\sum\limits_{i=1}^{n} \boldsymbol{a}_i z^{-i}}, \quad \forall z \in \mathbb{C}. \qquad (4)$$

Proof: $e(k)$ is defined by $e(k) \triangleq y^\dagger(k) - y(k)$, so

$$\begin{aligned} e(k) &= -\sum_{i=1}^{n} \boldsymbol{a}_i \left(y^\dagger(k-i) - y(k-i)\right) + \xi(k) & (5)\\ &= \xi(k) - \sum_{i=1}^{n} \boldsymbol{a}_i e(k-i) & (6)\end{aligned}$$

and then $e(k)$ is the result of noise $\xi(k)$ through the transfer function h_ξ. ■

The first (μ) and second (σ^2) order moments of a noise ξ are defined and denoted by:

$$\begin{aligned} \mu_\xi &\triangleq E\left\{\xi(k)\right\} & (7)\\ \sigma_\xi^2 &\triangleq E\left\{(\xi-\mu_\xi)^2(k)\right\}, & (8)\end{aligned}$$

where $E\{.\}$ is the *mean* operator.

One classical criteria used to measure the roundoff noise effect is the output noise power:

Definition 1 (Output noise power) *The output noise power is defined as the power of the noise added on the output:*

$$p \triangleq E\left\{e^2(k)\right\}. \qquad (9)$$

Proposition 2 *The output noise power is obtained from the first and second-order moment of ξ by:*

$$p = \|h_\xi\|_2^2 \sigma_\xi^2 + (h_\xi(1)\mu_\xi)^2 \qquad (10)$$

where $\|h_\xi\|_2$ is the ℓ_2-norm of the filter h_ξ and $h_\xi(1)$ its DC-gain:

$$h_\xi(1) = \frac{1}{1+\sum\limits_{i=1}^{n} \boldsymbol{a}_i}. \qquad (11)$$

Proof: Eq. (9) leads to $p = \sigma_e^2 + \mu_e^2$. Moreover, the basic properties of noise transmission through a linear system gives [12], [13]

$$\mu_e = h_\xi(1)\mu_\xi, \quad \sigma_e^2 = \|h_\xi\|_2^2 \sigma_\xi^2. \qquad (12)$$

■

B. Quantization of the coefficients

In addition to roundoff noise, the coefficients of the implemented filter are different from the ideal filter. Indeed, the coefficients are approximated by their fixed-point best approximation because of the finite precision of the representation of real numbers (see section III-A).
We denote $h^\dagger$ the implemented transfer function h, *i.e.* the transfer function with quantized coefficients:

$$h^\dagger(z) = \frac{\sum_{i=0}^{n} \boldsymbol{b}_i^\dagger z^{-i}}{1+\sum_{i=1}^{n} \boldsymbol{a}_i^\dagger z^{-i}}, \quad \forall z \in \mathbb{C}. \qquad (13)$$

where the $(\boldsymbol{a}_i^\dagger)$ and $(\boldsymbol{b}_i^\dagger)$ are the quantized approximations of $(\boldsymbol{a}_i)$ and $(\boldsymbol{b}_i)$, respectively. These approximations depend on the fixed-point representation used.

The criterion used to measure the transfer function deviation is the following:

Definition 2 (Transfer function error) *The transfer function error Δh due to the quantization is defined as the ℓ_2-norm of the gap between the ideal transfer function and the implemented one,* i.e.*:*

$$\Delta h \triangleq \left\|h - h^\dagger\right\|_2. \qquad (14)$$

Remark 1 This measure depends explicitly on the exact fixed-point representation (bit-width, binary-point position) used for each coefficient. Since this information is not always available when the FWL effects are studied, a more interesting criterion is preferred: the ℓ_2-sensitivity measure. A lot of work have been done in order to evaluate this sensitivity with respect to the coefficients and evaluate how much the coefficients' quantization changes the transfer function and the poles or zeros. For further explanations, see [14], [1], [5].

C. Dynamic Range analysis

If $x(k)$ is a scalar signal, we denote $\|x\|_\infty$ the L_∞-norm of this signal, *i.e.*

$$\|x\|_\infty \triangleq \sup_k |x(k)| \,. \tag{15}$$

The following lemma reminds the basic properties of a Bounded Input Bounded Output (BIBO) system, and is useful to determine the fixed-point format of the output of the considered filter.

Lemma 1 *The filter h is said to be BIBO if the ℓ_1-norm of h is finite, with*

$$\|h\|_{\ell_1} \triangleq \sum_{k=0}^{\infty} |h(k)| \tag{16}$$

and $\{h(k)\}_{k\geq 0}$ is the impulse response of h.
Moreover, in that case, the peak value of the output y can be bounded by [15], [16]

$$\|y\|_\infty \leq \|h\|_{\ell_1} \|u\|_\infty \,. \tag{17}$$

where u is the input.

Remark 2 The ℓ_1-norm can be very conservative, specially if the input u is known to have some particularities. For example, if u has a known frequency distribution, it can be interesting to use a weighted ℓ_1-norm, in order to avoid over-estimation of $\|y\|_\infty$.

III. Fixed-point sum-of-products

The sum-of-products (SoP) considered in this paper is a dot-product between a constant vector $\boldsymbol{c} = (\boldsymbol{c}_i)_{1\leqslant i\leqslant n}$ and a variable vector $\boldsymbol{x} = (\boldsymbol{x}_i)_{1\leqslant i\leqslant n}$:

$$r = \boldsymbol{c}.\boldsymbol{x} = \sum_{i=1}^{n} \boldsymbol{c}_i \boldsymbol{x}_i. \tag{18}$$

In this section, a lot of possible schemes for evaluate r will be shown, so the problem can be stated as follow :

Problem 1 *Given a sum-of-products as in eq.* (18) *where the constants $\boldsymbol{c}_i$'s, and their wordlength, the fixed-point representation of the variable $\boldsymbol{x}_i$'s and the fixed-point representation of the result r are known, determine all the possible evaluation schemes for* (18) *and parametrize them.*

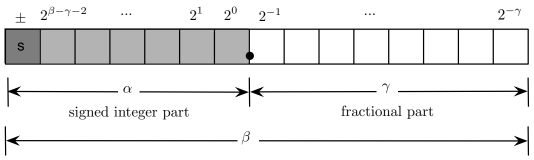


Fig. 2. FPR scheme

A. Fixed-point arithmetic and operations

Compared to floating-point arithmetic, fixed-point arithmetic is a number representation widely used in embedded system because of its ratio precision/lightness.

Fixed-Point Representation (FPR) can be formalized by the tuple (β, α, γ), where β is the bit-width of a number x in fixed-point arithmetic, α the number of bits of the integer part and γ the number of bits of its fractional part (see Figure 2). Number representation used in this paper is signed number with 2's complement, so the sign bit is included in α.

Let $x \in \mathbb{R}^*$ be a real number we want to convert in a signed-fixed-point number. The bit-width of x (β_x) is a chosen value, generally a multiple of 8 bits. Then, the length of its integer part is determined as the exact number of bits necessary to write its integer part, so:

$$\alpha_x \triangleq \lfloor \log_2 |x| \rfloor + 2 \tag{19}$$

with $\lfloor x \rfloor$ the operation that rounds x to the nearest integer lower than or equal to x.

Note that if an unsigned representation were used, we should have $\alpha_x = \lfloor \log_2 |x| \rfloor + 1$.

Finally the fractional part is evaluated as $\gamma_x = \beta_x - \alpha_x$, and the FPR of x is defined as $FPR_x = (\beta_x, \alpha_x, \gamma_x)$.

Moreover, in fixed-point arithmetic, x is represented by the integer N_x determined by the first β_x significant bits of its binary representation:

$$N_x \triangleq \lfloor x.2^{\gamma_x} \rceil \tag{20}$$

where $\lfloor x \rceil$ is the operation that rounds x to the nearest integer. So, x is approximated by $x^\dagger = N_x.2^{-\gamma_x}$.

After those recalls, the only two types of operations needed for the sum-of-products, $i.e.$ additions and multiplications, can be described.

For both of multiplication and addition, FPR's of operands and bit-width of the result are given data, and complete FPR of the result (*i.e.* its α and γ values) need to be determined.

1) Multiplication: Let $z = x \times y$ be a multiplication, with $(\beta_x, \alpha_x, \gamma_x)$ and $(\beta_y, \alpha_y, \gamma_y)$ the respective FPR of x and y. FPR_z is given by:

$$FPR_z = (\beta_x + \beta_y, \alpha_x + \alpha_y, \gamma_x + \gamma_y) \tag{21}$$

and the operation is realized by $N_z \leftarrow N_x \times N_y$.

In fact, this is the optimal case, because generally the bit-width of the hardware operation is fixed, and possibly different from $\beta_x + \beta_y$.

Proposition 3 (Fixed-Point Multiplication) *Let β_{op} be the fixed bit-width of the result. Then, in the general case, the result FPR of the operation $z = x \times y$,* i.e. *FPR_z, is deduced from the operands' FPR, from β_{op}, FPR_x and FPR_y, by:*

$$FPR_z = (\beta_{op}, \alpha_x + \alpha_y, \beta_{op} - (\alpha_x + \alpha_y)) \tag{22}$$

and the operation is realized by

$$N_z \leftarrow (N_x \gg s_x) \times (N_y \gg s_y) \tag{23}$$

where $\gg$ is the right-shift operation (with troncation or round-to-the-nearest), s_x and s_y are signed right bit-shifts to apply on N_x and N_y such that $s_x + s_y = \beta_x + \beta_y - \beta_{op}$ (if $\beta_op = \beta_x + \beta_y$, then $s_x = s_y = 0$ is choosen).
See Figures 5(a) and 5(b) for an illustrative example.

Remark 3 As said in Proposition 3, β_{op} is a given data, so determine FPR_z means determine α_z or γ_z (the other one is given by completion with β_{op}).

2) Addition: The other operation required, for which we have a constraint. Indeed, for adding two numbers in fixed-point arithmetic, they need to share the same FPR. Let $z = x + y$ be an addition, with FPR_x and FPR_y as defined in previous subsection. $FPR_z = (\beta_z, \alpha_z, \gamma_z)$ is determined in the optimal case by:

$$\left\{ \begin{array}{lcl} \alpha_z & = & \max(\alpha_x, \alpha_y) + 1 \\ \gamma_z & = & \max(\gamma_x, \gamma_y) \\ \beta_z & = & \alpha_z + \gamma_z \end{array} \right. \tag{24}$$

Once again, generally the bit-width of the operator, *i.e.* β_{op} may be different than the optimal one. The following proposition gives the fixed-point representation for the general case:

Proposition 4 (Fixed-Point Addition) *The result FPR $FPR_z = (\beta_z, \alpha_z, \gamma_z)$ is inferred from the input's FPR in the general case, from β_{op}, FPR_x and FPR_y, by:*

$$\left\{ \begin{array}{lcl} \alpha_z & = & \max(\alpha_x, \alpha_y) + 1 \\ \gamma_z & = & \beta_{op} - \max(\alpha_x, \alpha_y) - 1 \\ \beta_z & = & \beta_{op} \end{array} \right. \tag{25}$$

Moreover, $z = x + y$ is realized by:

$$N_z \leftarrow (N_x \gg s_x) + (N_y \gg s_y) \tag{26}$$

with $s_x = \gamma_x - \gamma_z$ and $s_y = \gamma_y - \gamma_z$.

B. Ordered Sum-of-products

A consequence of proposition 4 is that the addition in fixed-point arithmetic is commutative but not necessarily associative. Indeed, before an addition, one of the two operands will be quantized. If $\oplus$ represents the fixed-point addition, $(a \oplus b) \oplus c$ could be different from $a \oplus (b \oplus c)$, because the fixed-point representation of $(a \oplus b)$ and $(b \oplus c)$ can be different, leading to different quantizations of a, b and c in the two cases.

Consequently, a given dot-product can be realized in fixed-point arithmetic with several sum-of-products with particular order, depending on the parenthesizings of the sum. We denote them *ordered*-SoP (oSoP).

Moreover, for a n^{th}-order SoP, there are $\prod_{i=1}^{n-1} (2i - 1)$ different oSoPs to consider (the proof can be obtained by induction) [17].

For instance, for $n = 6$, there are 945 possibilities. Here are two of them, with $\boldsymbol{z}_i = \boldsymbol{c}_i.\boldsymbol{x}_i$ (see Figures 3(a) and 3(b)):

$$r = ((\boldsymbol{z}_0 + \boldsymbol{z}_3) + \boldsymbol{z}_2) + ((\boldsymbol{z}_4 + \boldsymbol{z}_5) + \boldsymbol{z}_1) \tag{27}$$

and

$$r = (((\boldsymbol{z}_3 + \boldsymbol{z}_5) + \boldsymbol{z}_2) + \boldsymbol{z}_4) + (\boldsymbol{z}_0 + \boldsymbol{z}_1) \tag{28}$$

Obviously, they can lead to different result, when implemented in fixed-point arithmetic.

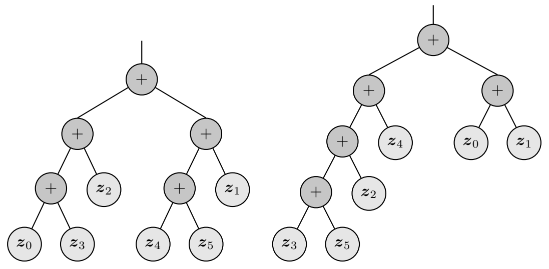


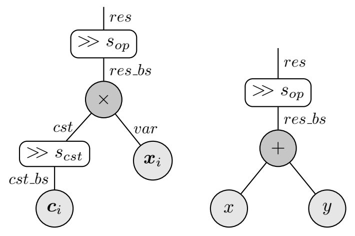
Fig. 3. Two 6th-order oSoPs, among the 945 possibilities

Thus, it is of interest to consider all of them and find the most interesting ones, with respect to roundoff noise analysis (and for our further work with respect to architecture adequacy).
It is quite easy to generate them all, except that it is not reasonable from $n \geq 10$.

C. FPR propagation in an oSoP

The FPR propagation is the way to determine the FPR among the various operations of the oSoP.

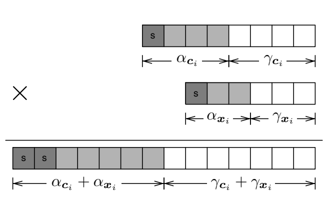
In our approach, oSoP are binary trees, where internal nodes are adders and multipliers and leaves are the couples $(\boldsymbol{c}_i, \boldsymbol{x}_i)_{1 \leqslant i \leqslant n}$. In fact, if we consider a multiplier as a cell composed by the operator and its operands, then all leaves of the trees are these multiplier cells and internal nodes are adders.



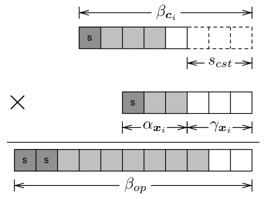
(a) A Multiplier cell (b) An Adder cell

Fig. 4. Graphical representations of adder and multiplier cells

The two operators used in our oSoP and their associated rules of propagation can be now described .



(a) FPR scheme for optimal multiplication



(b) FPR scheme for multiplication with a fixed β_{op}

Fig. 5. Optimal and general multiplication

1) Multiplier cell: As shown in Figure 4(a), the multiplier cell is composed by a multiplier (β_{op} as resulting bit-width), followed by a potential signed right bit-shift of s_{op} bits, used to align the result. The result of the multiplication, before the shift, is denoted res_bs (*result before shift*), whereas the shifted result is denoted res.

The first operand of the multiplier is the variable $\boldsymbol{x}_i$ (with FPR_{var} as FPR) and the second operator is the constant $\boldsymbol{c}_i$ potentially followed by a signed right-shift of s_{cst} bits. cst_bs and cst denote the constant before shift and the shifted constant, respectively.

The following parameters are supposed to be known: β_{op} the bit-width of the multiplier, $FPR_{var} = (\beta_{\boldsymbol{x}_i}, \alpha_{\boldsymbol{x}_i}, \gamma_{\boldsymbol{x}_i})$ the fixed-point representation of the variable $\boldsymbol{x}_i$ and $FPR_{cst\_bs} = (\beta_{\boldsymbol{c}_i}, \alpha_{\boldsymbol{c}_i}, \gamma_{\boldsymbol{c}_i})$ the fixed-point representation of the constant $\boldsymbol{c}_i$ (obtained with equation (19)).

Then the propagation is made by determining FPR_{res_bs} and s_{op} from these three values and proposition 3:

- The right shift is only applied here on the constant, so

$$s_{cst} = \beta_{\boldsymbol{c}_i} + \beta_{\boldsymbol{x}_i} - \beta_{op}, \tag{29}$$

- the FPR of the constant operand is then

$$FPR_{cst} = (\beta_{op} - \beta_{\boldsymbol{x}_i}, \alpha_{\boldsymbol{c}_i}, \gamma_{\boldsymbol{c}_i} - s_{cst}), \tag{30}$$

- and finally the FPR of the result (before a potential shift) is

$$FPR_{res\_bs} = (\beta_{op}, \alpha_{\boldsymbol{c}_i} + \alpha_{\boldsymbol{x}_i}, \beta_{op} - \alpha_{\boldsymbol{c}_i} - \alpha_{\boldsymbol{x}_i}). \tag{31}$$

Figures 5(a) and 5(b) illustrate the operation.

2) Adder cell: The adder cell, as shown in Figure 4(b) is composed by an adder (with β_{op} as bit-width), followed by a potential signed right bit-shift of s_{op} bits. The same notation res_bs (result before shift) and res is used for consistency. The two operands are denoted x and y.

In our approach, propagation algorithm is a recursive algorithm, *i.e.* it is applied on the final adder of the oSoP (the tree root) which calls the algorithm for its two operands, and so on.

This algorithm, applied on an adder cell, determines the FPR of the result, before the potential shift. It is composed of three steps:

a) Apply the propagation algorithm to its two operand x and y in order to obtain their respective FPR_{res_bs}. Note that if x or y are a multiplier cell, then equations (29) to (31) are used.

b) A common FPR is determined for the addition, by applying proposition 4:

$$FPR_{res\_bs} = (\beta_{op}, \max(\alpha_x, \alpha_y){+}1, \beta_{op}{-}\max(\alpha_x, \alpha_y){-}1) \tag{32}$$

Moreover,since the FPR of the final result of the sum-of-products is supposed to be known, we can limit the integer part of the partial results to the integer part of the final result. So the FPR_{res_bs} is given by $FPR_{res\_bs} = (\beta_{op}, \alpha_{op}, \beta_{op} - \alpha_{op})$ with

$$\alpha_{op} = \min(\max(\alpha_x, \alpha_y) + 1, \alpha_r) \tag{33}$$

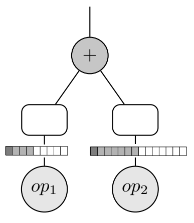
where α_r is the integer part of the final result r (even if an intermediate overflow occurs here, it will be compensated in a next addition, thanks to Jackson's rule[3][18]).

c) Then, we know what must be the FPR of the two operands. They are the FPR_{res} of the operands x and y. Since their FPR_{res_bs} is known, the right shift s to be applied can be evaluated with

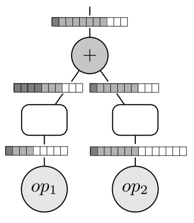
$$s_x = \gamma_x - \gamma_{res\_bs}, \quad s_y = \gamma_y - \gamma_{res\_bs}. \tag{34}$$

It corresponds to the s_{op} of the two cells x and y.

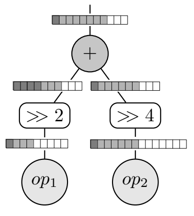
Figures 6(a), 6(b) and 6(c) illustrate respectively the steps a), b) and c) of the propagation algorithm.



(a) Step a)



(b) Step b)



(c) Step c)

Fig. 6. Propagation algorithm for the addition (with $\beta_{op} = 10$)

In addition, some further rules can be added for best roundoff or propagation, with respect to material architectures or extra optimization. We are going to discuss these options in the next parts.

D. Roundoff Before and/or After Multiplication

Inside the multiplication cell, it has been seen that there could be a shift on the constant when the multiplier bit-width is smaller than the optimal bit-width. Moreover, an adder could yield to a shift on its operands, like a multiplier.

So a multiplier cell may have two different shifts, one on the constant and the other on the result (see Figure 7(a)). Of course, the first one is a *virtual* shift, *i.e.* there is no real shift to perform, the constant is only replaced by its shifted value.

[3]Jackson's Rule says that in any number of consecutive additions and/or subtractions, some intermediate results and operands may overflow. As long as the final result representation can handle the final result without overflow, then the result is valid. This is due to the fact that fixed-point arithmetic is a modulo arithmetic.

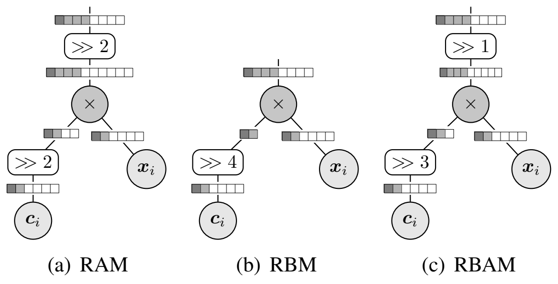


Fig. 7. Roundoff Before and/or After Multiplication

Depending on the architecture, there is not always the possibility to proceed to a shift after the multiplication (for example, most DSP doesn't have a barrel shifter in their Multiplier-and-Accumulator (MAC) operator, and do a shift after the multiplication implies a lot of extra cycles). In that case, a Roundoff After Multiplication (RAM) scheme is not possible, so we have to group the two shifts onto the constant shift. These scheme is called Roundoff Before Multiplication (RBM): the bit-shift s_{op} is then deferred onto the bit-shift s_{cst} (see Figure 7(b)).
RBM doesn't produce any roundoff noise after the multiplication, but there is a loss of precision into the coefficients used, leading to a greater transfer function error. This effect mainly depends on the sensitivity of the application (here a filter or controller) with respect to its coefficients.

Another option is to let the roundoff after multiplication. Indeed, as it said below, and if material architecture permits it, RAM saves constant informations for multiplication. One can note that we can't defer s_{cst} onto s_{op} in this case because s_{cst} is due to a bit adjustment compared to operator width.

A third and last option is a roundoff before and after multiplication (RBAM), where we choose to defer a part of the bit-shift s_{op} onto the bit-shift s_{cst} and let the other part (see Figure 7(c)). With this option we can control the loss of information and the noise appearing in oSoP after multipliers, as a tradeoff.

E. Shifts removal

As seen in the previous part, RBM scheme removes the shifts directly after multiplication in our oSoP (*i.e.* between multipliers and adders), but what if we want no shift at all in the oSoP? A good solution to this problem is to consider a single common FPR for all additions. Indeed, once we have calculated final fixed-point representation $FPR_r = (\beta_r, \alpha_r, \gamma_r)$, we can choose a single FPR with α_r as integer part for all additions in our oSoP.

Then, in the propagation algorithm, the integer part of each adder is not determined by equation (33), but by α_r. Once again, some overflows may occurs in intermediate additions, but the final result will still be valid thanks to Jackson's rule.

F. Roundoff noise model

In fixed-point arithmetic, any positive shift can be considered as equivalent to the add of noise, called *roundoff noise* (see Figure 8). These noises are modeled as white noise uniformly distributed and statistically independent [2].

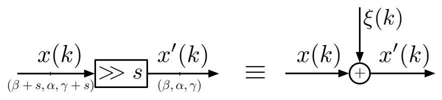


Fig. 8. Right shift can be considered as equivalent to add roundoff noise.

The following proposition reminds the noise produced during shift:

Proposition 5 *Let $x(k)$ be a signal with $(\beta+s, \alpha, \gamma+s)$ as FPR. Right shifting $x(k)$ of s bits is similar to add to $x(k)$ the independent white noise $\xi(k)$.*
If $s > 0$, the moments of $\xi(k)$ are given by:

$$\mu_\xi = 2^{-\gamma-1}(1-2^{-s}) \tag{35}$$

$$\sigma_\xi^2 = \frac{2^{-2\gamma}}{12}(1-2^{-2s}) \tag{36}$$

otherwise $\xi(k)$ is null.
Proof: See [2], [19]. ■

Remark 4 We consider here right-shift with troncation, but propostion 5 can also be generalized for round-to-the-nearest quantization.

So, when FPR have been propagated through the whole oSoP, all shift values are known and so we can evaluated final noise. Indeed, according to the roundoff noise model, every noises are independents, so we can sum noises through the oSoP as well as we propagate FPR.

Finally, a sum-of-products implemented in fixed-point arithmetic (with a given oSoP and implementation scheme) is equivalent to the original infinite precision sum-of-products, corrupted by the addition of a roundoff noise, that is the sum of all the roundoff noises occurring during the fixed-point evaluation.
So the moments of the final noise $\xi(k)$ added by the implementation of the IIR in equation (3) can be known and used with proposition 2 to evaluate the roundoff noise power of a given implementation.

IV. ILLUSTRATIVE EXAMPLE

A 3-th order Butterworth filter is used as illustrative example. Its coefficients $(\boldsymbol{a}_i)$ and $(\boldsymbol{b}_i)$ are given by the Matlab command `butter(3,0.166)`:

$$\begin{array}{rclrcl} & & & \boldsymbol{b}_0 & = & 0.0112074993 \\ \boldsymbol{a}_1 & = & -1.9670683622 & \boldsymbol{b}_1 & = & 0.0336224979 \\ \boldsymbol{a}_2 & = & 1.4046495057 & \boldsymbol{b}_2 & = & 0.0336224979 \\ \boldsymbol{a}_3 & = & -0.3479211490 & \boldsymbol{b}_3 & = & 0.0112074993 \end{array} \tag{37}$$

Its ℓ_1-norm is $\|h\|_{\ell_1} = 1.21820$.
We will consider 8 and 16-bit implementation, with different evaluation schemes, and compare them according to their respective roundoff noise error p and transfer function error Δh. We will also consider the theoretical latency of the

computations (assuming infinite ressources), *i.e.* the height of the oSoP, given by the number of successive additions required.
The oSoPs will be displayed as a tree, with all the FPR visible (coefficients, variables and partial results), and as fixed-point algorithm (see Figures 9(a) and 9(b), and equations (38) to (39)).

We suppose that the input belongs to $[-10, 10]$ ($\|u\|_\infty \leq 10$), so with lemma 1 the output peak value is bounded by $\|y\|_\infty \leq 12.19$. Then u and y will have the same fixed-point representation,namely $FPR_x = FPR_y = (8, 5, 3)$ for 8-bit implementation and $FPR_x = FPR_y = (16, 5, 11)$ for 16-bit.

1) Basic evaluation (oSoP #1): As said in introduction, classic evaluation (like those proposed by the usual tools from Mathworks and FPGA vendors) considers the same fixed-point representation for all the coefficients. In that case, the common format for all the coefficients is $(8, 3, 5)$ and $(16, 3, 13)$ for 8 and 16-bit implementation, respectively. One can directly note that 5 bits for the fractional part are not enough to represent the $(\boldsymbol{b}_i)$ coefficients that will be rounded to zero (underflow). So 8-bit implementation is not possible in that context (at least 11 bits are necessary to represent each coefficient without underflow).

For 16-bit implementation, the additions/accumulations will be performed on 32 bits, with $FPR_{op} = (32, 8, 24)$. The associated fixed-point algorithm is given by equation (38). Note that any evaluation scheme can be used (any parenthesis of the sum) can be used, since the additions are all done on the same FPR and are thus associative.

2) Optimal oSoPs: We also consider various optimal schemes, according to the different possible options presented in section III.

- For oSoP #2 (see Figure 9(a) and equation (38)), we consider all the possible oSoPs, with Roundoff After Multiplication and allowing bit-shifts in the computations. Among the oSoPs with the smallest roundoff error p, we consider the oSoPs with minimal height (in that case, there is only one).
- oSoP #3 is similar to oSoP #2 but with Roundoff Before Multiplication. The associated algorithm is given by eq. (40).
- We finally also look at the oSoPs involving accumulations only, with Roundoff After Multiplication. Four oSoPs satisfy the lowest roundoff error p, and they are equivalent (only permutations of multiplications with same FPR). One of them is given by (41), and denoted as oSoP #4.

All the results are summed up in table I.
We have considered a 7-term sum-of-products, with various schemes. In 16 bits, there is not much differences between the four realizations, since the final quantization (to bring back the 32-bit result in the 16-bit variable $y(n)$) is the predominant source of degradation (the roundoff noise error p is not displayed here with enough precision to notice the differences). However, the differences are much more important in 8 bits. First, the basic approach is not possible, due to underflows of the coefficients. Then, one can remark that oSoP #3 produces less noise than the other oSoPs, but the transfer function is much more changed. This is due to the RBM, since some of the bit-shifts are moved into the coefficients. The two others oSoP are finally quite similar, and will be chosen according to the hardware target used for the implementation.

	8-bit		16-bit		
	p	Δh	p	Δh	height
oSoP #1	-	-	$1.4688e^{-4}$	$4.60875e^{-4}$	6
oSoP #2	$3.77955e^{-2}$	$3.22968e^{-4}$	$1.4688e^{-4}$	$1.98361e^{-4}$	4
oSoP #3	$1.21495e^{-2}$	$1.53966e^{-2}$	$1.4688e^{-4}$	$1.39950e^{-4}$	4
oSoP #4	$3.78499e^{-2}$	$3.22968e^{-4}$	$1.4688e^{-4}$	$1.98361e^{-4}$	6

TABLE I
ROUNDOFF NOISE POWER AND TRANSFER FUNCTION ERROR FOR THE VARIOUS IMPLEMENTATIONS

V. CONCLUSION

In this paper, new evaluation schemes for sum-of-products in fixed-point arithmetic have been proposed. Those schemes have been formalized according to fixed-point representation of constants and variables, and severals options. Not only it automates the fixed-point code generation, but also it provides simultaneously a roundoff noise evaluation. Moreover, this result has been applied to IIR filter, for those we have described FWL effects analysis. Of course, this work presents a first step for the optimal fixed-point implementation. In hardware, some other important steps need to be performed, such as word-length optimization and hardware mapping.

It will be also interesting, in further work, to apply these results to the Specialized Implicit Framework [8], in order to generate and analyze fine fixed-point code for various filter structures. Combined to techniques such as residue feedback error [20] or ρ-operator, it will provides an interesting tool for optimized fixed-point implementation of filters.
Finally, the optimal implementation problem [4], based on the tradeoff precision vs implementation cost will be tackled.

ACKNOWLEDGMENT

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

REFERENCES

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

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

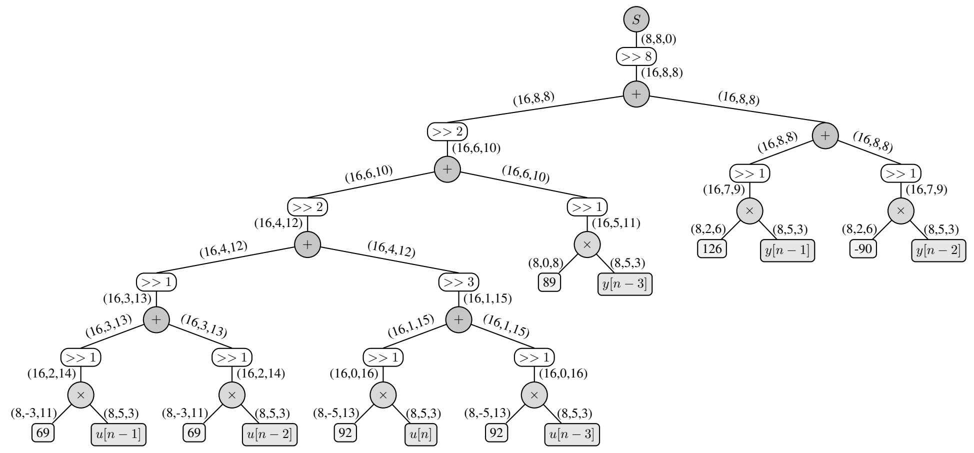
[3] H. Hanselmann, "Implementation of digital controllers - a survey," *Automatica*, vol. 23, no. 1, pp. 7–32, January 1987.

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

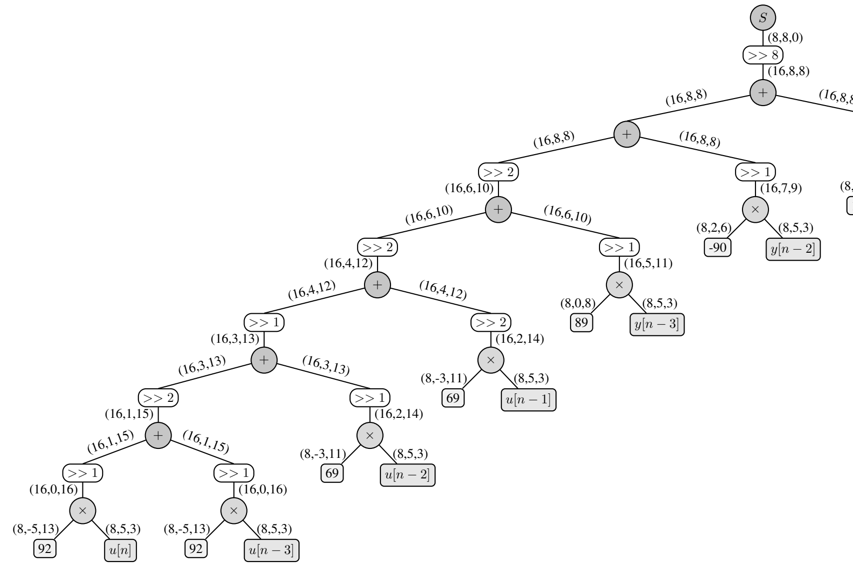
[5] T. Hilaire and P. Chevrel, "Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context," *EURASIP Journal on Advances in Signal Processing*, vol. special issue on Quantization of VLSI Digital Signal Processing Systems, January 2011.

[6] P. Langlois, M. Martel, and L. Thévenoux, "Accuracy versus time: a case study with summation algorithms," in *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, ser. PASCO '10. New York, NY, USA: ACM, 2010, pp. 121–130.

[7] G. Constantinides, P. Cheung, and W. Luk, "The multiple wordlength paradigm," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.



(a) 8-bit implementation of oSoP #2



(b) 8-bit implementation of oSoP #4

$$\begin{aligned} N_{y(n)} \;&\leftarrow\; (((((((91 * N_{u(n)}) + (275 * N_{u(n-1)})) + (275 * N_{u(n-2)})) + (91 * N_{u(n-3)})) \\ &\quad +(16114 * N_{y(n-1)})) + (-11507 * N_{y(n-2)})) + (2850 * N_{y(n-3)})) \qquad (38) \end{aligned}$$

$$\begin{aligned} N_{y(n)} \;&\leftarrow\; (((126 * N_{y(n-1)}) \gg 1 + (-90 * N_{y(n-2)}) \gg 1) + ((((69 * N_{u(n-1)}) \gg 1 + (69 * N_{u(n-2)}) \gg 1) \gg 1 \\ &\quad +((92 * N_{u(n)}) \gg 1 + (92 * N_{u(n-3)}) \gg 1) >> 3) \gg 2 + (89 * N_{y(n-3)}) \gg 1) \gg 2) \qquad (39) \end{aligned}$$

$$\begin{aligned} N_{y(n)} \;&\leftarrow\; (((16114 * N_{y(n-1)}) + (-11507 * N_{y(n-2)})) + ((((8814 * N_{u(n-1)}) + (8814 * N_{u(n-2)})) >> 1 \\ &\quad +((11752 * N_{u(n)}) + (11752 * N_{u(n-3)})) >> 3) >> 2 + (11400 * N_{y(n-3)})) >> 2) \qquad (40) \end{aligned}$$

$$\begin{aligned} N_{y(n)} \;&\leftarrow\; (((((((92 * N_{u(n)}) >> 1 + (92 * N_{u(n-3)}) >> 1) >> 2 + (69 * N_{u(n-2)}) >> 1) >> 1 + (69 * N_{u(n-1)}) >> 2) >> 2 \\ &\quad +(89 * N_{y(n-3)}) >> 1) >> 2 + (-90 * N_{y(n-2)}) >> 1) + (126 * N_{y(n-1)}) >> 1) \qquad (41) \end{aligned}$$

[8] 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.

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

[10] G. Li and Z. Zhao, "On the generalized DFIIt structure and its state-space realization in digital filter implementation," *IEEE Trans. on Circuits and Systems*, vol. 51, no. 4, pp. 769–778, April 2004.

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

[12] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. Mc Graw Hill, 1991.

[13] T. Hilaire, D. Ménard, and O. Sentieys, "Bit accurate roundoff noise analysis of fixed-point linear controllers," in *Computer-Aided Control Systems, 2008. CACSD 2008. IEEE International Conference on*, September 2008, pp. 607–612.

[14] V. Tavşanoğlu and L. Thiele, "Optimal design of state-space digital filters by simultaneous minimization of sensibility and roundoff noise," in *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. CAS-31, October 1984.

[15] 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.

[16] V.-S. Chellaboina, W. M. Haddad, D. S. Bernstein, and D. A. Wilson, "Induced convolution norms of linear dynamical systems," in *Proc. Amer. Contr. Conf.*, June 1999, pp. 3805–3810.

[17] "The on-line encyclopedia of integer sequences, Sequence A001147." [Online]. Available: http://oeis.org/A001147

[18] L. B. Jackson, *Digital Filters and Signal Processing*, 3rd ed. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[19] G. Constantinides, P. Cheung, and W. Luk, "Truncation Noise in Fixed-Point SFGs," *IEE Electronics Letters*, vol. 35, no. 23, Nov. 1999.

[20] D. Williamson and S. Sridharan, "Residue feedback in digital filters using fractional feedback coefficients," in *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 33, April 1985, pp. 477–483.