Formatting bits to better implement signal processing algorithms

Benoit Lopez[1], Thibault Hilaire[1] and Laurent-Stéphane Didier[2]
[1]*LIP6, Pierre and Marie Curie University (UPMC Univ Paris 06), Paris, France*
[2]*IMATH, University of the South, Toulon-Var (USTV), Toulon, France*
{*benoit.lopez, thibault.hilaire*}*@lip6.fr, laurent-stephane.didier@univ-tln.fr*

Keywords: Fixed-point arithmetic, accurate sum-of-products, bit formatting, digital signal processing implementation

Abstract: This article deals with the fixed-point computation of the sum-of-products, necessary for the implementation of several algorithms, including linear filters. Fixed-point arithmetic implies output errors to be controlled. So, a new method is proposed to perform accurate computation of the filter and minimize the word-lengths of the operations. This is done by removing bits from operands that don't impact the final result under a given limit. Then, the final output of linear filter is guaranteed to be a faithful rounding of the real output.

1 INTRODUCTION

Usually, embedded digital signal processing algorithms are specified using floating-point arithmetic and next implemented using fixed-point (FxP) arithmetic (Padgett and Anderson, 2009) for cost, size and power consumption reasons. FxP arithmetic is used as an approximation of real numbers based on integers and implicit fixed scaling by a power of 2. Of course, the quantization of coefficients and the rounding errors due to FxP computations lead to a degraded numerical accuracy of the implemented algorithm. Therefore, it is a great interest for the designer of embedded system to determine and control the implementation error while maintaining low computational effort.

In fixed-point arithmetic, a main current problem is to minimize the word-lengths of operands under constraints of precision in order to minimize area and/or power consumption (Constantinides et al., 2004). In this paper, a new method to reduce the number of bits to consider in each sum-of-products (SoP, also called Multiply-And-Accumulate) is proposed. The SoPs are one of the elementary operations of DSP algorithms. The main point of our approach is that if the final fixed-point format is known, then the bits having no impact in the final result can be detected and therefore discarded. Each term of the sum-of-products can be then reformatted into a new fixed-point format having less bits.

Some fixed-point arithmetic definitions and notations are reminded in section 2. Section 3 formalizes the proposed approach, which is decomposed into two formatting, for most significant bits and least significant bits, respectively. Section 4 describes the error analysis for Direct Form I filters implemented with the bit formatting technique. Finally, an illustrative example is given with a 4^{th} order Butterworth filter, before conclusion in section 6.

2 Fixed-Point arithmetic and Sum-of-Products

In this article we consider *signed* FxP arithmetic in two's complement representation. Let x be such a FxP number with w bits as word-length:

$$x = -2^m x_m + \sum_{i=\ell}^{m-1} 2^i x_i \qquad (1)$$

where $x_i \in \mathbb{B} \triangleq \{0,1\}$ is the i^{th} bit of x, m and ℓ are the position of the *most* significant bit (MSB) and *least* significant bits (LSB), respectively (Fig. 1). It can be noted that $m > \ell$ and

$$w = m - \ell + 1. \qquad (2)$$

In a digital system, x is represented by an integer X, composed by the w bits $\{x_i\}_{\ell \leqslant i \leqslant m}$. In other words, $X = x.2^{-\ell}$, or equivalently

$$X = -2^{m-\ell} x_m + \sum_{i=0}^{m-\ell-1} 2^i x_{i+\ell}. \qquad (3)$$

Through this paper the notation (m,ℓ) is used to denote the Fixed-Point Format (FPF) of such a fixed-point number and m, ℓ and w will be suffixed by the variable or constant they refer to.

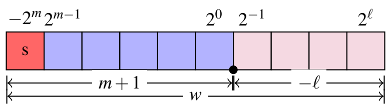


Figure 1: Fixed-point representation. m and ℓ are the position of the MSB and LSB respectively (in this figure, $m=5$ and $\ell=-4$).

Remark 1. *In FxP arithmetic, there is no restriction on the the position of the MSB and LSB. The FPF is often chosen with $m \geqslant 0$ and $\ell \leqslant 0$. FPF with $\ell > 0$ are also possible (the quantization step is greater than 1) or $m < 0$ (the largest represented number is lower than $\frac{1}{2}$).*

2.1 Conversion from real to fixed-point

Many SoP-based DSP algorithms involve real coefficients that have to be converted into FxP arithmetic. Let consider a real constant $c \in \mathbb{R}^*$. The position of the most significant bit m of its w-bit wide FxP representation in binary two's complement is:

$$m = \begin{cases} \lceil \log_2 |c| \rceil & \text{if } c < 0 \\ \lfloor \log_2 |c| \rfloor + 1 & \text{if } c > 0 \end{cases} \tag{4}$$

where $\lfloor \cdot \rfloor$and $\lceil \cdot \rceil$ are the *round to the integer towards minus infinity* and *round towards plus infinity* operators, respectively.

For some very special cases, eq. (4) should be adapted (Hilaire and Lopez, 2013). The position of the least significant bit ℓ is deduced from eq. (2) and the w-bit integer C representing c is computed:

$$C = \left\lfloor c.2^\ell \right\rceil \tag{5}$$

where $\lfloor \cdot \rceil$ is the *round to the nearest integer* operator.

2.2 Sum-of-Products

In digital signal processing, the computation of filter or controller algorithms requires the evaluation of one or several SoP. Their type and number depend on the algorithm chosen (Hanselmann, 1987; Istepanian and Whidborne, 2001; Gevers and Li, 1993). For instance, the direct forms require only 1 SoP, whereas the n-th order state-space require $n+1$ SoPs.

The products considered in such a SoP are products of real constants and real variables. But, in the context of fixed-point design, only fixed-point variables and fixed-point constants are considered. In this article, we consider SoPs whose constants have already been converted in FxP format.

More formally, we consider SoPs

$$s = \sum_{i=1}^{n} c_i \cdot v_i, \tag{6}$$

where $\{c_i\}_{1 \leqslant i \leqslant n}$ are given non-null FxP constants and $\{v_i\}_{1 \leqslant i \leqslant n}$ FxP variables only known to be in known intervals $[\underline{v}_i; \overline{v}_i]$. We focus on the best way (*i.e.* employing the minimum word-lengths) to obtain a rounding of the exact sum s at a given format.

Remark 2. *It is also possible to consider the $\{c_i\}$ to be real constants instead of FxP constants, so as to analyze the impact of their quantization that is not considered here.*

However, this impact is well studied with sensitivity measures such as the transfer function sensitivity (Tavşanoğlu and Thiele, 1984; Gevers and Li, 1993; Hinamoto et al., 2006), the pole/zero sensitivity (Gevers and Li, 1993; Li, 1998) or IIR stability (Lu and Hinamoto, 2003).

3 BITS FORMATTING

The main point of the proposed approach is that if the final fixed-point format of a sum $s = \sum p_i$, denoted $FPF_f = (m_f, \ell_f)$ is known, then it is probably possible to discard some useless bits.

More formally, this paper is focused on bits of p_is with positions lower than ℓ_f (section 3.2) and greater than m_f (section 3.3) in order to determine their impact on the result. We determine the useless bits and remove them from p_is before the sum is computed. The p_is are rounded into an intermediate format $(m_i, \ell_f - \delta)$, where δ is the number of non-useless bits with position lower than ℓ_f. Then, the sum of these modified p_is is computed and rounded into the final format FPF_f in order to obtain the final result (see Figure 2(b)).

3.1 Definitions and notations

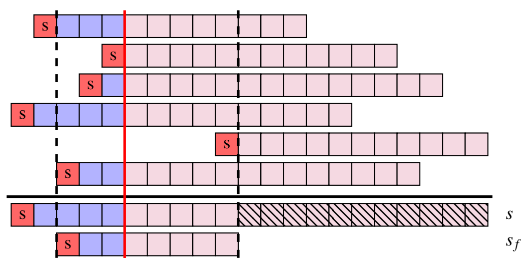
In this section the fixed-point rounding modes used in this article are defined.

Definition 1 (Fixed-point rounding modes)**.** *Let x be a real value. The notations $\circ_d(x)$, $\nabla_d(x)$ and $\triangle_d\ (x)$ express the* rounding to the nearest, *the* rounding down *(*i.e. *truncation) and the* rounding up *of x according to the d^{th} bit, respectively. These operators are defined by:*

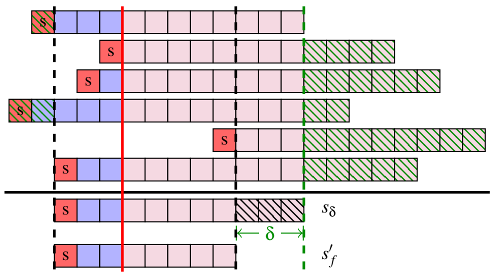
$$\circ_d(x) \triangleq 2^d \cdot \left\lfloor \frac{x}{2^d} \right\rceil, \tag{7}$$

$$\nabla_d(x) \triangleq 2^d \cdot \left\lfloor \frac{x}{2^d} \right\rfloor, \tag{8}$$

$$\triangle_d\ (x) \triangleq 2^d \cdot \left\lceil \frac{x}{2^d} \right\rceil. \tag{9}$$



(a) The exact sum is performed and then rounded to (m_f, ℓ_f)



(b) The sum is performed on the format $(m_f, \ell_f + \delta)$ and then rounded to (m_f, ℓ_f)

Figure 2: Two different ways to perform the FxP accumulation

The operator $\star_d(x)$ is the faithful rounding of x at the d^{th} bit, i.e.

$$\star_d(x) \in \{\nabla_d(x), \Delta_d\,(x)\}. \tag{10}$$

The round-to-the-nearest operation always returns the nearest representable point of the real exact value, while the faithful rounding operation produces either the nearest or next-nearest point.

Notations Some notations need to be explicitly defined before explaining the proposed approach:

- $p_i \triangleq c_i \times v_i$ denotes the result of the fixed-point product of c_i and v_i. According to the fixed-point multiplication rule (Lopez et al., 2012), the fixed-point format of p_i is defined as $FPF_{p_i} \triangleq (m_i, \ell_i) = (m_{c_i} + m_{v_i} + 1, \ell_{c_i} + \ell_{v_i})$.
- $p_{i,j}$ is the j^{th} bit of p_i, for $1 \leqslant i \leqslant n$ and $\ell_i \leqslant j \leqslant m_i$.
- (M, L) is the FPF of the exact sum $s = \sum_{i=1}^{n} p_i$, where

  $$M \triangleq \max_i(m_{p_i}) + \lceil \log_2(n) \rceil \tag{11}$$

  and

  $$L \triangleq \min_i(\ell_{p_i}). \tag{12}$$

  $\lceil \log_2(n) \rceil$ corresponds to the number of carry bits to consider for the sum of n terms.

Moreover, three different sums are also considered, where $\diamond_d$ is a given common rounding mode (round-to-nearest or truncate): $\diamond_d \in \{\circ_d, \nabla_d\}$:

- $s_f \triangleq \diamond_{\ell_f}(s)$ is the rounding of the exact sum s into the final format FPF_f.
- $s_\delta \triangleq \sum_{i=1}^{n} \diamond_{\ell_f - \delta}(p_i)$ is the sum of the products p_is rounded into format $(m_i, \ell_f - \delta)$ where δ is a given positive constant to be discussed later.
- $s'_f \triangleq \diamond_{\ell_f}(s_\delta)$ is the rounding of the sum s_δ into the final format FPF_f.

Figures 2(a) and 2(b) illustrate these different approximation of the sum s.

Modular fixed-point sum As reminded in equation (3), a fixed-point number x is coded in computer with a w-bit signed integer X. As a consequence, all the operations are done modulo 2^w on this integer. Proposition 1 specifies the modular fixed-point sum as an extension of the modular sum on integers.

Proposition 1 (Modular fixed-point sum)**.** *The sum modulo 2^d of two fixed-point numbers x and y sharing the same FPF (m, ℓ), is noted $x \overset{d}{\oplus} y$ and is computed as:*

$$x \overset{d}{\oplus} y \triangleq \left(\left((X + Y + 2^{d-\ell}) \mod 2^{d-\ell+1} \right) - 2^{d-\ell} \right) 2^{\ell}. \tag{13}$$

Moreover, the modular fixed-point sum of n fixed-point numbers x_i is noted and computed as:

$$\overset{d}{\bigoplus_{1 \leqslant i \leqslant n}} x_i \triangleq x_1 \overset{d}{\oplus} x_2 \overset{d}{\oplus} \dots \overset{d}{\oplus} x_n. \tag{14}$$

Proof: The fixed-point sum modulo d, $x \overset{d}{\oplus} y$, corresponds to the $d - \ell$-bits sum of the integers $X = x.2^{-\ell}$ and $Y = y.2^{-\ell}$.

Therefore, adding two signed fixed-point numbers requires to convert them in positive integers, add them modulo $2^{d-\ell+1}$, and convert the result back into signed fixed-point number. ■

Example 1. *Adding* 12.5 *and* 3.75 *in FPF* $(4, -3)$ *(two's complement with 8 bits) is given by* $12.5 \overset{4}{\oplus} 3.75$ *and leads to* -15.75 *according to eq.* (13) *because of the overflow.* *12.5 is coded by* $01100.100_{\mathbb{B}}$*,* 3.75 *is coded by* $00011.110_{\mathbb{B}}$*, so the modular sum leads to* $10000.010_{\mathbb{B}}$ *into format* $(4, -3)$*, that is interpreted as* -15.75.

3.2 LSBs formatting

Let consider the final FxP format (m_f, ℓ_f) of a SoP. It appears that not all the least significant bits are usefull in order to correctly round the result of a SoP to (m_f, ℓ_f). The value δ is the position such that all bits with a position lower than $\ell_f - \delta$ are insignificant and can be removed from p_is (Fig. 2(b)). Therefore, only p_is such that $\ell_i < \ell_f - \delta$ are rounded, the other remain unchanged. The sum s_δ of rounded p_is is computed on format $(m_f, \ell_f - \delta)$ and finally rounded onto the final format (m_f, ℓ_f).

The following proposition formalizes the choice of δ.

Proposition 2. *For both rounding mode ($\diamond_{l_f} = \circ_{l_f}$ round-to-nearest or $\diamond_{l_f} = \nabla_{l_f}$ truncation), the integer δ that provides $s'_f = \star_{l_f}(s_f)$ is given by:*

$$\delta = \lceil \log_2(n_f) \rceil \qquad (15)$$

with $n_f = Card(I_f)$ and $I_f \triangleq \{i \mid \ell_i < \ell_f\}$.

Proof: This proof is done for truncation rounding mode.The same reasoning can be established for round-to-nearest mode.

Computation of s involves all bits $p_{i,j}$ whereas s_δ requires only bits $p_{i,j}$ for $j \geqslant 2^{l_f - \delta}$, so:

$$s \geqslant s_\delta \qquad (16)$$

The trivial case $s = s_\delta$ implies $s_f = s'_f$, so thereafter only the case $s - s_\delta > 0$ is considered and the difference $s - s_\delta$ is evaluated precisely as the sum of bits $p_{i,j}$ for $1 \leqslant i \leqslant n$ and $L \leqslant j \leqslant \ell_f - \delta - 1$:

$$s - s_\delta = \sum_{i=1}^{n} \sum_{j=L}^{\ell_f - \delta - 1} 2^j p_{i,j} \qquad (17)$$

Since $\sum_{j=L}^{\ell_f-\delta-1} 2^j p_{i,j} < 2^{\ell_f - \delta}$ for $1 \leqslant i \leqslant n$, $s - s_\delta$ can be bounded as follows:

$$s - s_\delta < n_f \cdot 2^{\ell_f - \delta} \qquad (18)$$

Now, the difference $s_f - s'_f$ corresponds to the rounding of the difference $s - s_\delta$ according to the ${\ell_f}^{th}$ bit:

$$s_f - s'_f = \nabla_{\ell_f}(s - s_\delta) \qquad (19)$$

It also can be viewed as the carry bits greater than 2^{ℓ_f} implied by the difference $s - s_\delta$.

Using equation (18), the difference $s_f - s'_f$ can also be bounded:

$$\lfloor (s - s_\delta) \cdot 2^{-\ell_f} \rfloor \leqslant (s - s_\delta) \cdot 2^{-\ell_f} < n_f \cdot 2^{-\delta} \qquad (20)$$

$$s_f - s'_f < n_f \cdot 2^{\ell_f - \delta} \qquad (21)$$

Since δ needs to be determined in order to verify $|s_f - s'_f| \leqslant 2^{\ell_f}$, the following inequality comes from equation (21):

$$n_f \cdot 2^{\ell_f - \delta} \leqslant 2^{\ell_f} \qquad (22)$$

The smallest integer solving inequality (22) is $\delta = \lceil \log_2(n_f) \rceil$.

■

Remark 3. *With Proposition 2, it may happened that one p_i (or more) has a MSB lesser than $\ell_f - \delta$, and so all bits of this p_i will be removed by applying this technique. Therefore, a good idea will be to redetermine δ with n_f minus the number of removed p_i. So equation* (15) *can be replaced by Algorithm 1.*

Algorithm 1: Evaluation of the integer δ

```
Input: Operands $p_i$s in format $(m_i, \ell_i)$
       The final format $FPF_f = (m_f, \ell_f)$
Output: $\delta \in \mathbb{N}$
1 $n_\delta \leftarrow n$;
2 repeat
3     $n' \leftarrow n_\delta$;
4     $\delta \leftarrow \lceil \log_2(n') \rceil$;
5     $n_\delta \leftarrow Card(\{i \mid 1 \leqslant i \leqslant n$ and $m_i < \ell_f - \delta\})$;
6 until $n' = n_\delta$;
7 return $\delta$
```

Formating method The first step of LSB formatting is a direct application of proposition 2: it removes useless bits. After this step $\ell_i \geqslant \ell_f - \delta, \forall 1 \leqslant i \leqslant n$.

The second step involves having $\ell_i = \ell_f - \delta, \forall 1 \leqslant i \leqslant n$. To do this, either FPF_{p_i} can be changed from (m_i, ℓ_i) to $(m_i, \ell_f - \delta)$ for p_is such that $\ell_i > \ell_f - \delta$ (consisting to add $\ell_i - \ell_f + \delta$ zeros to the right of these p_is), or multipliers can be rewritten to perform operation into a given word-length. Let M_i be the multiplier computing $p_i = c_i \times v_i$. Then, w_{M_i}, the word-length of the result of M_i is given by:

$$w_{M_i} = m_{M_i} + \ell_{M_i} + 1 \qquad (23)$$

with

$$m_{M_i} = m_{c_i} + m_{v_i} + 1 \qquad (24)$$

$$\ell_{M_i} = \ell_f - \delta \qquad (25)$$

where m_{c_i} and m_{v_i} are MSBs of c_i and v_i respectively.

Remark 4. *For a better accuracy, m_{M_i} can be evaluated using formulas from section 2.1, it avoids double sign bit in general case (given by the +1 in eq.* (24)*). Moreover, if $\ell_{c_i} + \ell_{v_i} > \ell_{M_i}$, then $\ell_{c_i} + \ell_{v_i} - \ell_{M_i}$ zeros are added to the right of p_is to ensure $\ell_i = \ell_f - \delta$ for these p_is.*

Error evaluation Adding two numbers in FxP arithmetic requires to align them onto the same LSB using right-shifts. A rounding error may occur, which introduces a numerical error. After the second step of LSB formatting, where rounding errors may be introduced, all p_is have the same LSB, *i.e.* $\ell_f - \delta$. There is no need of right-shift to align operands of additions and therefore no additional rounding errors are introduced by the global sum of p_is.

The total number of right-shifts involved in the first step of our method can be bounded as follows.

Proposition 3. *With this LSB formatting technique and for a n^{th}-order SoP, the number of right-shifts is bounded by $n+1$, at most one right-shift by multiplier (denoted d_i for multiplier M_i) and exactly one final right-shift (denoted d_f). Their values are:*

$$d_i \triangleq \ell_f - \delta - \ell_{c_i} - \ell_{v_i} \;\forall i \in I \tag{26}$$

and

$$d_f \triangleq \delta \tag{27}$$

where $I = \{i \mid 1 \leqslant i \leqslant n \text{ and } \ell_f - \delta > \ell_{c_i} + \ell_{v_i}\}$.

Proof: d_i is the right-shift in multiplier M_i if a right-shift is necessary, *i.e.* if $\ell_f - \delta > \ell_{c_i} + \ell_{v_i}$ so the number of bits to remove to ensure $\ell_i = \ell_f - \delta$ is $\ell_f - \delta - \ell_{c_i} - \ell_{v_i}$. All the additions are computed on $w_f + \delta$ bits, and the result is w_f bits long, so the final right-shift value is δ. ■

Remark 5. *In Proposition 3 only non-zero right-shifts are considered. If d_i is defined as $\max(\ell_f - \delta - \ell_{c_i} - \ell_{v_i}, 0)$ rather than just $\ell_f - \delta - \ell_{c_i} - \ell_{v_i}$, all multipliers have a right-shift, possibly null, and so the exact number of right-shifts is $n+1$.*

Finally, it is possible to bound the error introduced by our method. As seen in (Hilaire and Lopez, 2013), the right shifting of d bits of a variable x (with (m, ℓ) as FPF) is equivalent to add an interval error $[e] = [\underline{e}; \overline{e}]$ with

	Truncation	Round to the nearest
$[\underline{e}, \overline{e}]$	$[-2^{\ell+d} + 2^{\ell}; 0]$	$[-2^{\ell+d-1} + 2^{\ell}; 2^{\ell+d-1}]$

(28)

So the global interval error for the LSB technique can be evaluated with the following properties.

Proposition 4. *The global interval error using LSB formatting technique is $[e] = [\underline{e}; \overline{e}]$ with:*
Truncation:

$$\underline{e} = \sum_{i \in I} (-2^{\ell_i + d_i} + 2^{\ell_i}) - 2^{\ell_f} + 2^{\ell_f - \delta} \tag{29}$$

$$\overline{e} = 0 \tag{30}$$

Round to nearest:

$$\underline{e} = \sum_{i \in I} (-2^{\ell_i + d_i - 1} + 2^{\ell_i}) - 2^{\ell_f - 1} + 2^{\ell_f - \delta} \tag{31}$$

$$\overline{e} = \sum_{i \in I} (2^{\ell_i + d_i - 1}) + 2^{\ell_f - 1} \tag{32}$$

with $I = \{i \mid 1 \leqslant i \leqslant n \text{ and } \ell_f - \delta > \ell_{c_i} + \ell_{v_i}\}$ and $\ell_i = \ell_{c_i} + \ell_{v_i}$, where ℓ_{c_i} and ℓ_{v_i} are positions of LSBs of c_i and v_i respectively.

Proof: By using (28) on a multiplier i, $\underline{e}$ equals to $-2^{\ell_i + d_i - 1} + 2^{\ell_i}$ where d_i is the right-shift value given by Proposition 3 and p_i is the initial LSB of the multiplier result, *i.e.* the optimal LSB which is the sum of LSBs of product operands c_i and v_i. For the final right-shift, the initial LSB equals to $\ell_f - \delta$ and the final result is δ bits right-shifted. ■

Remark 6. *The precise bounds of the global interval error shown in Proposition 4 can be bounded by a power of 2. Indeed, for truncation rounding mode the global interval error is included in $]-2^{\ell_f+1}; 0]$, whereas for round-to-nearest rounding mode it is included in $]-2^{\ell_f}; 2^{\ell_f}[$.*

In (Lopez et al., 2012), all p_is have different LSBs, and therefore the global error depends on the order of the additions. Consequently, all the different evaluation schemes (ES), *i.e.* all the different possible orders of the additions, are generated and the choice is made meanly for ES with a minimal error. Here, all ES have the same global error value (Proposition 4), so error can not be a criteria to choose the best ES representing the sum. The criteria chosen in the section 5 is the *infinite parallelism* criteria, *i.e.* the most parallelizable ES.

3.3 MSBs formatting

The MSBs of p_is having a greater positions than the final MSB, m_f can be removed using a new formalization of the Jackson's Rule (Jackson, 1970). This Rule states that in consecutive additions and/or subtractions in two's complement arithmetic, 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.

Example 2. *Let us consider a sum S of three 8-bit integers with two's complement arithmetic, for example $104 + 82 - 94$. The result $S = 92$ is in the range of 8-bit signed numbers, but the intermediate sum $104 + 82$ produces an overflow and equals to -70 into this format (instead of 186 that cannot be represented). The final sum $-70 - 94$ also produces an overflow and equals to 92 into the final format, that is the correct result.*

With this paper's notations, it means that bits with a greater position than m_f can be removed from concerned p_is.

Proposition 5 (Fixed-Point Jackson's Rule)**.** *Let s be a sum of n fixed-point number p_is, in format (M, L).*

If s is known to have a final MSB equals to m_f with $m_f < M$, then:

$$s = \bigoplus_{1\leqslant i\leqslant n}^{m_f+1} \left(\sum_{j=L}^{m_f} 2^j p_{i,j} \right) \tag{33}$$

Proof: $s = \sum_{i=1}^{n} p_i$, so, from (1):

$$s = \sum_{i=1}^{n} \left(-2^M p_{i,M} + \sum_{j=L}^{M-1} 2^j p_{i,j} \right) \tag{34}$$

All bits of s greater than 2^{m_f} (from $p_{i,j}$ with $j \geqslant m_f$ and from the carry bits produced by $p_{i,j}$ with $j < m_f$) are repetitions of the sign bits, since $-2^{m_f} \leqslant s < 2^{m_f}$ (by definition of the final FPF). ■

Thus, in our method, the MSB formatting is an application of the propostion to a sum s_δ previously *LSB-formatted* p_is, *i.e.* with $L = \ell_f - \delta$. Therefore, s_δ can be computed only using bits $p_{i,j}$ with $1 \leqslant i \leqslant n$, $\ell_f - \delta \leqslant j \leqslant m_f$ without considering intermediate overflows.

4 OUTPUT ERROR ANALYSIS

Let us consider a n-th order IIR[1] filter having H as a transfer function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}, \quad \forall z \in \mathbb{C}. \tag{35}$$

This filter is usually realized with the following algorithm

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i) \tag{36}$$

where $u(k)$ is the input at step k and $y(k)$ the output at step k.

So the evaluation of the filter relies on the evaluation of a SoP. As seen in previous sections, the fixed-point evaluation of eq. (36) implies the add of an error $e(k)$ at time k, and only $y^\dagger$ (the output contaminated with roundoff error) can be computed:

$$y^\dagger(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y^\dagger(k-i) + e(k). \tag{37}$$

In (Lopez et al., 2012), it has been shown that the implemented system eq. (37) can be seen as the initial system (36) with an error added on the output, as shown in Figure 3: by subtracting equations (37) and (36), it comes

[1]Infinite Impulse Response

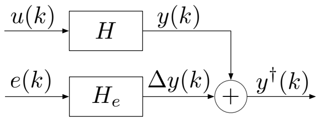


Figure 3: Equivalent system, with output error extracted

$$y^\dagger(k) - y(k) = e(k) - \sum_{i=1}^{n} a_i \left(y^\dagger(k-i) - y(k-i) \right) \tag{38}$$

So the output error $\Delta y(k) \triangleq y^\dagger(k) - y(k)$ can be seen as the result of the error $e(k)$ through the filter H_e defined by

$$H_e(z) = \frac{1}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}, \quad \forall z \in \mathbb{C}. \tag{39}$$

Since the error $e(k)$ done in the evaluation of the SoP is known to be in a given interval $[\underline{e};\overline{e}]$ (see Proposition 4), then the following proposition (Hilaire and Lopez, 2013) gives the output error interval:

Proposition 6 (Output error interval)**.** *$\Delta y(k)$ is the output of the error $e(k)$ through the filter H_e. If the error $e(k)$ is in $[\underline{e};\overline{e}]$, then $\Delta y(k)$ is in $[\underline{\Delta y};\overline{\Delta y}]$ with:*

$$\underline{\Delta y} = \frac{\overline{e}+\underline{e}}{2} |H_e|_{DC} - \frac{\overline{e}-\underline{e}}{2} \|H_e\|_{\ell^\infty} \tag{40}$$

$$\overline{\Delta y} = \frac{\overline{e}+\underline{e}}{2} |H_e|_{DC} + \frac{\overline{e}-\underline{e}}{2} \|H_e\|_{\ell^\infty} \tag{41}$$

and $|H_e|_{DC}$ is the DC-gain (low-frequency gain) of H_e and $\|H_e\|_{\ell^\infty}$ its worst-case peak gain:

$$\|H_e\|_{\ell^\infty} \triangleq \frac{\sup_{k\geqslant 0}|y(k)|}{\sup_{k\geqslant 0}|u(k)|} \forall u \text{ and } y \text{ input and output of } H_e. \tag{42}$$

They can be computed by:

$$|H_e|_{DC} = H_e(1), \quad \|H_e\|_{\ell^\infty} = \sum_{k\geqslant 0} |h_e(k)| \tag{43}$$

where $h_e(k)$ is impulse response of the filter H_e.

Proof: Since H_e is linear, $\Delta y(k)$ can be seen as the sum of a constant term $\frac{\overline{e}+\underline{e}}{2}$ through the filter H_e and a variable term bounded by $\frac{\overline{e}-\underline{e}}{2}$. The constant term is amplified by the low-frequency gain $|H_e|_{DC}$, whereas the bound of the variable term is amplified by $\|H_e\|_{\ell^\infty}$ (eq. (42)). ■

5 RESULTS AND COMPARISONS

A 4-th order Butterworth filter is used as an illustrative example. The chosen realization to compute

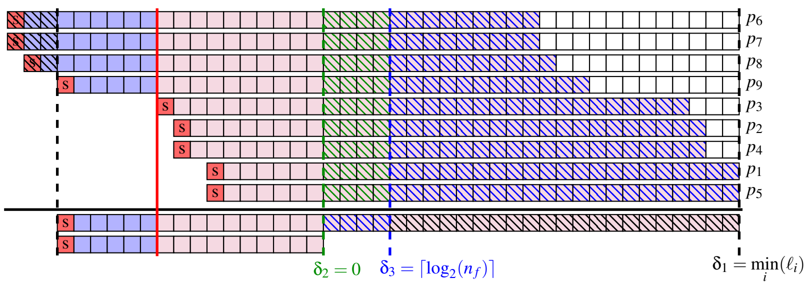

p_6
p_7
p_8
p_9
p_3
p_2
p_4
p_1
p_5
S
$\delta_2 = 0$
$\delta_3 = \lceil \log_2(n_f) \rceil$
$\delta_1 = \min_i(\ell_i)$

Figure 4: Bits representation of the sum of the example.

this filter is the Direct Form I:

$$y(k) = \sum_{i=0}^{4} b_i u(k-i) - \sum_{i=1}^{4} a_i y(k-i). \tag{44}$$

Its coefficients a_is and b_is are given by the Matlab command `butter(4,0.136)`:

			b_0	=	0.001328017792779
a_1	=	−2.871116228316502	b_1	=	0.005312071171115
a_2	=	3.208250066295749	b_2	=	0.007968106756673
a_3	=	−1.634594881084453	b_3	=	0.005312071171115
a_4	=	0.318709327789667	b_4	=	0.001328017792779

For implementations, the output variables $y(i)$s, input variables $u(i)$s, constants a_is and b_is are 16-bit words.

Moreover, variables $u(i)$s are considered in this example to be in the interval $[-13;13]$ (the corresponding FPF is $(4,-11)$), and variables $y(i)$s (including result output $y(k)$) are known to be in the interval $[-17.123541221107534;17.123541221107534]$ (corresponding to the final FPF $(m_f,\ell_f) = (5,-10)$).

From these informations, the operands to be summed, p_is, can be obtained with their respective FPF:

$$p_i \triangleq \begin{cases} b_{i-1}u(k-(i-1)) & \text{if} \quad 1 \leqslant i \leqslant 5 \\ a_{i-5}y(k-(i-5)) & \text{if} \quad 6 \leqslant i \leqslant 9 \end{cases}$$

FPF_{p_1}	=	$(-4,-35)$			
FPF_{p_2}	=	$(-2,-33)$	FPF_{p_6}	=	$(8,-23)$
FPF_{p_3}	=	$(-1,-32)$	FPF_{p_7}	=	$(8,-23)$
FPF_{p_4}	=	$(-2,-33)$	FPF_{p_8}	=	$(7,-24)$
FPF_{p_5}	=	$(-4,-35)$	FPF_{p_9}	=	$(5,-26)$

Four implementations are compared: a double precision implementation and three fixed-point implementations using bit formatting approach using different values of δ. In the first FxP implementation (denoted Fix_1), all bits are considered. This means that δ_1 is chosen such that $\ell_f - \delta_1 = \min_i(\ell_i)$. In other word, we have $\delta_1 = \ell_f - \min_i(\ell_i) = 25$. The Fix_1 implementation corresponds to the computation of the sum s with no LSB formatting.

In the second FxP implementation Fix_2, no additional bits are considered. The intermediate format is the final format, (m_f,ℓ_f), which corresponds to $\delta_2 = 0$. A large LSB reduction is performed.

The third FxP implementation Fix_3 is the faithful implementation, with δ_3 determined from Proposition 2, *i.e.* $\delta_3 = \lceil \log_2(n_f) \rceil$ with $n_f = n = 9$, so $\delta_3 = 4$. Only 4 guards bits are used in the LSB formatting.

Figure 4 illustrates this example. For Fix_1, since the intermediate format is $(5,-35)$, additional bits equal to 0 are considered to align all p_is onto this format. For Fix_2 and Fix_3, the intermediate formats, $(5,-10)$ and $(5,-14)$ respectively, permit to remove bits, blue and green hatched bits and blue hatched bits respectively. Finally, the intermediate sums are rounded to the final format $(5,-10)$, except s_{δ_2} which is already in the final format.

The global interval error $[\underline{e};\overline{e}]$ is computed, for the implementation Fix_3, using Proposition 4 :

$$\underline{e} = -1.4645302 \times 10^{-3}, \quad \overline{e} = 0. \tag{45}$$

From equation (43), DC-gain and worst-case peak gain of H_e are obtained :

$$|H_e|_{DC} = 49.5647, \quad \|H_e\|_{\ell^\infty} = 66.8474. \tag{46}$$

Finally, the output error interval (Proposition 6) $[\underline{\Delta y};\overline{\Delta y}]$ is computed from equations (45) and (46) :

$$\underline{\Delta y} = -8.52445240 \times 10^{-2}, \; \overline{\Delta y} = 1.26555189 \times 10^{-2}. \tag{47}$$

As illustration (but not proof), of the theoretical result (46), a simulation in FxP and floating point arithmetic has been done with a white noise input $u(k)$ in $[-13;13]$. The error between the double floating result and each of the FxP implementations is shown in Figure 5 The number of additional bits considered in Fix_3 is small compared with Fix_1 which considers all bits, but it is good enough to have errors measures far better than Fix_2 and really close to Fix_1. The plotted error for implementation Fix_3 is in the bound predicted by the theory.

The fixed-point implementation Fix_3 is given by algorithm 2. In this implementation, sum modulo

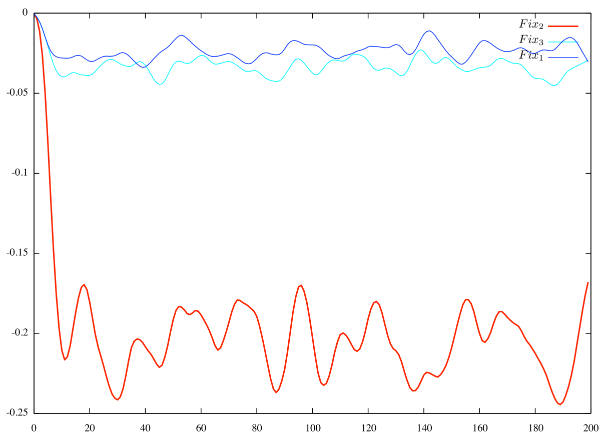


Figure 5: Computed error between double implementation and the three different fixed-point implementations.

2^{m_f+1} (*i.e.* 2^6) is performed, but since algorithm considers integer computations, the sum is performed modulo $2^{m_f+1-\ell_f+\delta}$ (*i.e.* 2^{20}).

Algorithm 2: Fixed-point algorithm

```
Input:
$U0$ to $U4$: 16-bit input $(4,-11)$
$Y1$ to $Y4$: 16-bit input $(5,-10)$
Output: $Y$: 16-bit output $(5,-10)$
Data: $Rx$: 20-bit registers
$\oplus$: the 20-bit sum
$R0 \leftarrow (23520 * Y1) \gg 9$;
$R1 \leftarrow (-26282 * Y2) \gg 9$;
$R2 \leftarrow R0 \oplus R1$;
$R0 \leftarrow (22280 * U0) \gg 21$;
$R1 \leftarrow R0 \oplus R2$;
$R0 \leftarrow (22280 * U3) \gg 19$;
$R2 \leftarrow (-20887 * Y4) \gg 12$;
$R3 \leftarrow R0 \oplus R2$;
$R0 \leftarrow R1 \oplus R3$;
$R1 \leftarrow (22280 * U4) \gg 21$;
$R2 \leftarrow (26781 * Y3) \gg 10$;
$R3 \leftarrow R1 \oplus R2$;
$R1 \leftarrow (16710 * U2) \gg 18$;
$R2 \leftarrow R3 \oplus R1$;
$R1 \leftarrow (22280 * U1) \gg 19$;
$R3 \leftarrow R2 \oplus R1$;
$R1 \leftarrow R0 \oplus R3$;
// Output computation
$Y \leftarrow R1 \gg 4$;
```

6 CONCLUSIONS

Throughout this paper, a new method of formatting bits has been described, in order to design fixed-point sum-of-products and then linear filters. This method allows to remove some bits and keep only the bits that impact the final result. The computed result is a faithful rounding of the final result considering all the bits. The example has shown the utility of applying this method to a linear filter expressed in a very common form, and the gain in term of number of bits is significant.

Future work will consist of a word-length optimization step that will consider the bit formatting method, and a code generation for algorithm-to-code mapping.

ACKNOWLEDGEMENTS

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

The authors would like to thank Florent de Dinechin for the instructive discussions about fixed-point implementation.

REFERENCES

Constantinides, G., Cheung, P., and Luk, W. (2004). *Synthesis and Optimization of DSP Algorithms*. Kluwer Academic Publishers.

Gevers, M. and Li, G. (1993). *Parametrizations in Control, Estimation and Filtering Probems*. Springer-Verlag.

Hanselmann, H. (1987). Implementation of digital controllers - a survey. *Automatica*, 23(1):7–32.

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

Hinamoto, T., Omoifo, O., and Lu, W.-S. (2006). L2-sensitivity minimization for mimo linear discrete-time systems subject to l2-scaling constraints. In *Proc. ISCCSP 2006*.

Istepanian, R. and Whidborne, J., editors (2001). *Digital Controller implementation and fragility*. Springer.

Jackson, L. (1970). Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form. *Audio and Electroacoustics, IEEE Transactions on*, 18(2):107–122.

Li, G. (1998). On the structure of digital controllers with finite word length consideration. *IEEE Trans. on Autom. Control*, 43(5):689–693.

Lopez, B., Hilaire, T., and Didier, L.-S. (2012). Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)*.

Lu, W.-S. and Hinamoto, T. (2003). Optimal design of iir digital filters with robust stability using conic-quadratic-programming updates. In *IEEE Trans. Signal Processing*, volume 51, pages 1581–1592.

Padgett, W. T. and Anderson, D. V. (2009). Fixed-point signal processing. *Synthesis Lectures on Signal Processing*, 4(1):1–133.

Tavşanoğlu, V. and Thiele, L. (1984). Optimal design of state-space digital filters by simultaneous minimization of sensibility and roundoff noise. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, volume CAS-31.