Fast Fixed-Point Optimization of DSP Algorithms

Gabriel Caffarena, Ángel Fernández-Herrero, Juan López, Carlos Carreras

Fast Fixed-Point Optimization of DSP Algorithms

Gabriel Caffarena[1], Ángel Fernández-Herrero[2], Juan A. López[2] and Carlos Carreras[2]

[1] Universidad CEU San Pablo,
Dep. Ingeniería de Sistemas de Información y Telecomunicación, Madrid 28668, Spain
gabriel.caffarenafernandez@ceu.es
[2] Universidad Politécnica de Madrid, Dep. Ingeniería Electrónica,
Madrid 28040, Spain
{angelfh,juanant,carreras}@die.upm.es

Abstract. In this chapter, the fast fixed-point optimization of Digital Signal Processing (DSP) algorithms is addressed. A fast quantization noise estimator is presented. The estimator enables a significant reduction in the computation time required to perform complex fixed-point optimizations, while providing a high accuracy. Also, a methodology to perform fixed-point optimization is developed.
Affine Arithmetic (AA) is used to provide a fast Signal-to-Quantization Noise-Ratio (SQNR) estimation that can be used during the fixed-point optimization stage. The fast estimator covers differentiable non-linear algorithms with and without feedbacks. The estimation is based on the parameterization of the statistical properties of the noise at the output of fixed-point algorithms. This parameterization allows relating the fixed-point formats of the signals to the output noise distribution by means of fast matrix operations. Thus, a fast estimation is achieved and the computation time of the fixed-point optimization process is significantly reduced.
The proposed estimator and the fixed-point optimization methodology are tested using a subset of non-linear algorithms, such as vector operations, IIR filter for mean power computation, adaptive filters – for both linear and non-linear system identification – and a channel equalizer. The computation time of fixed-point optimization is boosted by three orders of magnitude while keeping the average estimation error down to 6% in most cases.

Keywords: Fixed-Point Optimization, Digital Signal Processing, Quantization, Word-Length, Affine Arithmetic, Error Estimation, Signal-to-Quantization-Noise Ratio

1 Introduction

The use of fixed-point (FxP) arithmetic has proved to provide low-cost hardware implementations [1–3]. The selection of the FxP formats of the variables of an

algorithm is a time-consuming task that involves an optimization process whose goal is to find the set of word-lengths (WLs) that reduces cost most. The FxP optimization (FPO) process is a slow process, since the complexity of the optimization problem has been shown to be very complex (NP-hard [4]) and, also, because of the necessity of continuously assessing the accuracy of the algorithm which involves time-consuming simulations.

The estimation of the algorithm accuracy is normally performed adopting a simulation-based approach [1], which leads to long design times. However, in the last few years there have been attempts to provide fast estimation methods based on analytical techniques. These approaches can be applied to Linear Time-Invariant (LTI) systems [5, 2, 6] and to differentiable non-linear systems [7–10]. As for the noise metric used, they are based on the peak value [10] and on the computation of SQNR [2, 7, 5, 9, 8]. Since SQNR is a very popular error metric within DSP systems and because LTI systems have been extensively studied, our work aims at fast SQNR estimation techniques for differentiable non-linear systems.

The chapter contains the following contributions:

- A novel Affine-Arithmetic based SQNR estimator.
- An efficient methodology to perform fast and accurate SQNR estimates.
- Performance results using a set of non-linear benchmarks with and without feedbacks: adaptive filters, matrix operations, a mean power IIR filter, and a MIMO equalizer.

The chapter is organized as follows: In Section 2, some related works are discussed. Section 3 deals with fixed-point optimization. Section 4 introduces AA. Section 5 explains the fast estimation method. The software implementation of a FPO tool is addressed in Section 6. The performance results are presented in Section 7. And finally, Section 8 draws the conclusions.

2 Related work

Only those approaches aiming at the automatic SQNR estimation of non-differentiable algorithms are tackled here. This also excludes approaches exclusively based on simulations, as the automation of these approaches does not translate into a significant reduction of computation times since these methods are inherently very slow. In the approaches being considered, non-linearities are addressed in terms of the perturbation theory, where the effect of the quantization of each signal on the output signals is supposed to be *very small*. This allows the application of first-order Taylor expansions to each non-linear operation in order to characterize the quantization effect. Thus, the set of algorithms is constrained to those containing differentiable operations. Existent methods enable to obtain an expression that relates the WLs of signals to the power – also mean and variance – of the quantization noise at the output. This will be further explained through eqn. 22 in subsection 5.2.

The work in [7] proposes a hybrid method that combines simulations and analytical techniques to estimate the variance of the noise. The estimator is suitable for non-recursive and recursive algorithms. The parameterization phase is relatively fast, since it requires $|S|$ simulations for an algorithm with $|S|$ variables (or signals). The noise model is based on [11] and second order effects are neglected by applying first order Taylor expansions. The paper seems to suggest that the contributions of the signal quantization noises at the output can be added, assuming that the noises are independent. The accuracy of the method is not supported with any empirical data, so the quality of the method cannot be inferred.

The method in [9] makes use of a more time-consuming method, since $|S|^2/2$ simulations as well as a curve fitting technique with $|S|^2/2$ coefficients are required. On the one hand, the noise produced by each signal is described in terms of the traditional quantization noise model from [12], which is less accurate than [11], and, again, second order statistics are neglected. On the other hand, the expression of the estimated noise power accounts for noise interdependencies, which is a better approach than [7]. The method is tested with an LMS adaptive filter and the accuracy is evaluated graphically. There is no information about computation times.

Finally, in [8] the parameterization is performed by means of $|S|$ simulations and the estimator is suitable only for non-recursive systems. The accuracy of this approach seems to be the highest of all presented methods since it uses the model from [11] and it accounts for noise interdependencies. Although the information provided about accuracy is more complete, it is still not sufficient, since the estimator is only tested in a few SQNR scenarios. This approach was successfully extended to recursive systems in [13], with reasonably short parameterization times due to the use of linear-prediction techniques.

The approach explained in this chapter (Section 5) tries to overcome most of the drawbacks of the works presented above by considering:

- Both non-recursive and recursive algorithms
- An accurate noise model [11]
- Noise interdependencies

3 Fixed-Point Optimization

The starting point of FPO is a graph $G(V, S)$ describing a FxP algorithm. Set V contains the operations of the algorithm, and set S its signals. The FxP format of a number is defined by means of the pair (p, n), where p represents the number of bits required to represent the integer part, and n is the total number of bits (see left side of Figure 1). In fact, the complete FxP format of a signal requires more information: the format before quantization – (p_{pre}, n_{pre}) – and the format after quantization – (p, n) (see [2]). The error introduced by each quantization operation (i.e. truncation) is directly related to the number of least significant bits removed (see subsection 5.1 for a description of an error model that makes

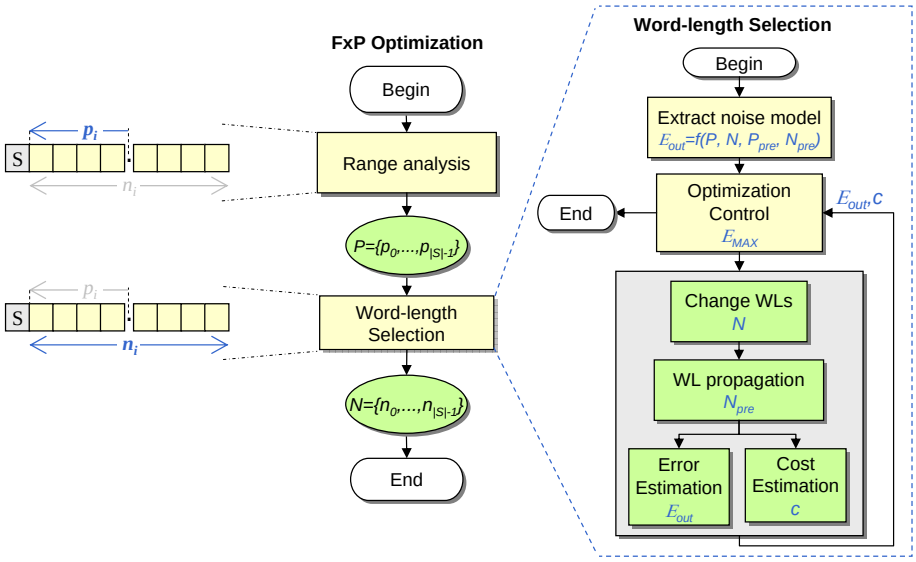

Fig. 1. Fixed-point optimization diagram.

use of pairs (p_{pre}, n_{pre}) and (p, n)). Figure 2 depicts the quantization of the output of a multiplication. Here, 16-bit signals a and b are multiplied, and the result c is quantized to 8 bits (i.e. 16-8=8 bits).

Initially, the FxP formats of signals are unknown and it is the task of FPO to find a suitable set of these that minimizes cost. The FxP format determines the quantization error generated by a quantized signal. This error is propagated to the output of the algorithm. Also, the FxP format determines the number of bits of a signal, and therefore the size of the hardware resources required to

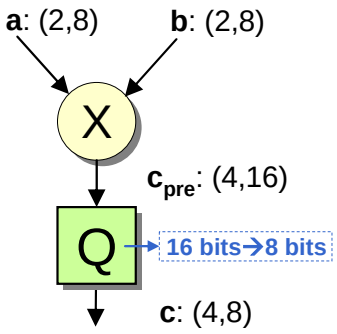

Fig. 2. Quantization of a multiplier.

process it. The size of a resource ultimately determines its area, delay and power costs. For instance, going back to Figure 2, the FxP format of signals a and b is determining the cost of the multiplier, while the truncation of output c is affecting the mathematical precision of the operation, which is ultimately affecting the overall mathematical precision of the algorithm. During FPO, the optimization is guided by the cost and the output error obtained from the different FxP formats tried through successive iterations.

Figure 1 depicts the FPO approach adopted in this work. FPO is composed of the stages of ranges analysis or *scaling*, which determines the set of p ($P = \{p_0, \ldots, p_{|S|-1}\}$), and *word-length selection*, which determines the set of n ($N = \{n_0, \ldots, n_{|S|-1}\}$). This separation allows simplifying FPO while still providing significant cost reductions.

A wrap-around scaling strategy is adopted since it requires less hardware resources than other approaches (i.e. saturation techniques). After scaling, the values of p are the minimum possible values that avoid the overflow of signals or, at least, those that reduce the likelihood of overflow to a negligible value. A simulation-based approach is used to carry out scaling [1]. During the simulation, the dynamic range of each signal is obtained and the value of p is computed by means of $p_i = \lfloor log_2(max_i) \rfloor + 1$.

Once scaling is performed, the values of p can be fixed during word-length selection. The right side of Fig. 1 shows a diagram of the basic steps during word-length selection. Basically, word-length selection iterates trying different word-lengths for the variables of the algorithm (i.e. n) until cost is minimized. Any time the WL of a signal or a group of signals is changed, the WLs must be propagated throughout the graph, task referred to as graph *conditioning* [2]. The *optimization control* block selects the size of the new WLs using the values of the previous error and cost estimations and decides when the optimization procedure has finished. The first task in the diagram is the extraction of the quantization noise model. The role of this operation is to generate a model of the quantization noise at the output, related to the FxP format of each signal (i.e. (p_i, n_i) and $(p_{i_{pre}}, n_{i_{pre}})$). This is the key to avoid the use of time-consuming simulations. The implications of using a quick error estimator within the optimization loop are twofold: i) the optimization process can be faster, or, ii) it is possible to perform a wider design space exploration. During the optimization process, the control block makes decisions about the signals that change their WLs according to the error and cost estimations.

4 Affine Arithmetic

Affine Arithmetic (AA) [14] is aimed at the fast and accurate computation of the ranges of the signals of an algorithm. Its main feature is that it automatically cancels the linear dependencies of the included uncertainties along the computation path, thus avoiding the oversizing produced by Interval Arithmetic (IA) approaches [15]. Regarding fixed-point optimization, it has been applied to both scaling computation [16, 17, 10], and word-length selection [5, 16, 10, 18]. Also, a

modification, called Quantized Affine Arithmetic (QAA), has been applied to the computation of limit cycles [19] and dynamic range analysis of quantized LTI algorithms [17].

4.1 Description

The mathematical expression of an affine form is

$$\hat{x} = x_0 + \sum_{i=1}^{N_x} x_i \epsilon_i \tag{1}$$

where x_0 is the central value of $\hat{x}$, and ϵ_i and x_i are its i-th noise term identifier and amplitude, respectively. In fact, $x_i\epsilon_i$ represents the interval $[-x_i, +x_i]$, so an affine form describes a numerical domain in terms of a central value and a sum of intervals with different identifiers. Affine operations are those which operate affine forms and produce an affine form as a result. Given the affine forms $\hat{x}$, $\hat{y}$ and $\hat{c} = c_0$, the affine operations are

$$\hat{x} \pm \hat{c} = x_0 \pm c_0 + \sum_{i=1}^{N_x} x_i \epsilon_i \tag{2}$$

$$\hat{x} \pm \hat{y} = x_0 \pm y_0 + \sum_{i=1}^{\max(N_x, N_y)} (x_i \pm y_i)\epsilon_i \tag{3}$$

$$\hat{c} \cdot \hat{x} = c_0 x_0 + \sum_{i=1}^{N_x} c_0 x_i \epsilon_i \tag{4}$$

These operations suffice to model any LTI algorithm. Differentiable operations can be approximated using a first-order Taylor expansion:

$$f(\hat{x}, \hat{y}) \approx f(x_0, y_0) + \sum_{i=1}^{\max(N_x, N_y)} \left(\frac{\delta f(x_0, y_0)}{\delta \hat{x}} \cdot x_i + \frac{\delta f(x_0, y_0)}{\delta \hat{y}} \cdot y_i \right) \epsilon_i \tag{5}$$

4.2 Example of application

This section describes an example of application of AA. Let us consider the standard Red, Green and Blue (RGB) to Luma, Red and Blue Chroma (YCrCb) converter shown in Figure 3 [2, 17], whose sequence of operations is given in the first column of Table 1. The second column shows the computation of the affine form associated to each algorithm's signal. The last column shows the interval associated to the dynamic range of each signal. Without loss of generality, in this

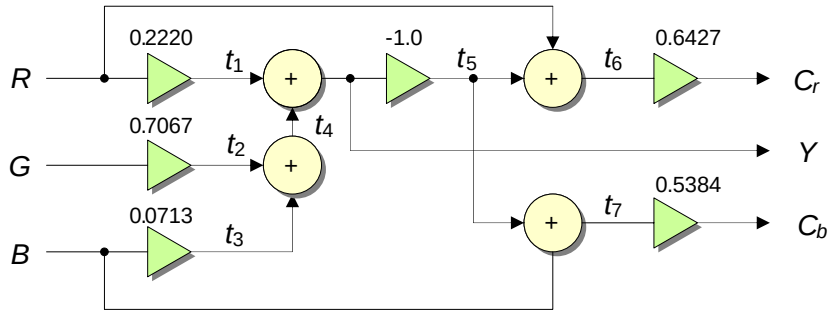

Fig. 3. ITU RGB to YCrCb converter.

Table 1. Propagation and computation of the affine forms for the ITU RGB-YCrCb converter.

Operations	AA-based computations	Signal ranges
$t_1 = 0.22R$	$\hat{t}_1 = 0.22 \cdot (96 + 32\epsilon_1) = 21.31 + 7.10\epsilon_1$	$[14.208, 28.416]$
$t_2 = 0.71G$	$\hat{t}_2 = 0.71 \cdot (96 + 32\epsilon_2) = 67.84 + 22.61\epsilon_2$	$[45.228, 90.458]$
$t_2 = 0.071B$	$\hat{t}_3 = 0.071 \cdot (96 + 32\epsilon_3) = 6.842.28\epsilon_3$	$[4.563, 9.126]$
$t_4 = t_2 + t_3$	$\hat{t}_4 = (67.84 + 22.61\epsilon_2) + (6.84 + 2.28\epsilon_3)$ $= 74.69 + 22.61\epsilon_2 + 2.28\epsilon_3$	$[49.792, 99.584]$
$t_4 = t_2 + t_3$	$\hat{t}_4 = (67.84 + 22.61\epsilon_2) + (6.84 + 2.28\epsilon_3)$ $= 74.69 + 22.61\epsilon_2 + 2.28\epsilon_3$	$[49.792, 99.584]$
$Y = t_1 + t_4$	$\hat{Y} = (21.31 + 7.1\epsilon_1) + (74.69 + 22.61\epsilon_2 + 2.28\epsilon_3)$ $= 96 + 7.1\epsilon_1 + 22.61\epsilon_2 + 2.28\epsilon_3$	$[64, 128]$
$t_5 = -Y$	$\hat{Y} = -(96 + 7.1\epsilon_2 + 22.61\epsilon_2 + 2.28\epsilon_3)$ $= -96 - 7.1\epsilon_1 - 22.61\epsilon_2 - 2.28\epsilon_3$	$[-128, -64]$
$t_6 = t_5 + R$	$\hat{t}_6 = (-96 - 7.1\epsilon_1 - 22.61\epsilon_2 - 2.28\epsilon_3) + (96 + 32\epsilon_1)$ $= 24.9\epsilon_1 - 22.61\epsilon_2 - 2.28\epsilon_3$	$[-49.792, 49.792]$
$C_r = 0.64t_6$	$\hat{C}_r = 0.64 \cdot (24.9\epsilon_1 - 22.61\epsilon 2 - 2.28\epsilon 3)$ $= 16\epsilon_1 - 14.53\epsilon_2 - 1.47\epsilon_3$	$[-32.001, 32.001]$
$t_7 = t_5 + B$	$\hat{t}_7 = (-96 - 7.1\epsilon_1 - 22.61\epsilon_2 - 2.28\epsilon_3) + (96 + 32\epsilon_3)$ $= -7.1\epsilon_1 - 22.61\epsilon_2 + 29.72\epsilon_3$	$[-59.437, 59.437]$
$C_b = 0.54t_7$	$\hat{C}_b = 0.54 \cdot (-7.1\epsilon_1 - 22.61\epsilon_2 + 29.72\epsilon_3)$ $= -3.82\epsilon_1 - 12.18\epsilon_2 + 16\epsilon_3$	$[-32.0008, 32.0008]$

example it is considered that the three input values are contained in the range $[64, 128]$, and that the computations are performed using infinite precision.

Assuming that the RGB values are independent from each other, the affine forms that represent the signal ranges are modeled using one distinct noise term per uncertainty source, i.e.,

$$\hat{R} = 96 + 32\epsilon_1 \tag{6}$$
$$\hat{G} = 96 + 32\epsilon_2 \tag{7}$$
$$\hat{B} = 96 + 32\epsilon_3. \tag{8}$$

By applying the operation definitions described in subsection 4.1, the affine form that represents the values of t_1 is

$$\hat{t}_1 = 0.2220 \cdot \hat{R} = 0.2220(96 + 32\epsilon_1) = 21.3120 + 7.1040\epsilon_1 \tag{9}$$

and the interval that specifies its range is

$$range(\hat{t}_1) = t_{1,0} \pm \sum_{i=1}^{\max(N_x,N_y)} range(t_{1_i} \cdot \epsilon_i) \tag{10}$$
$$= range(21.3127 \pm 7.1040) \tag{11}$$
$$= [14.2080, 28.4160] \tag{12}$$

The same procedure can be applied to the rest of the signals to obtain results found in Table 1.

This example illustrates the application of AA to compute the dynamic range of signals. However, AA is used is this work as a means to propagate the quantization error of each signal to the output. For this purposes, affine forms are used to represent quantization errors that are added to signals. The propagation of these errors follows the same rules that have been applied in the previous example. However, as shown in the next section, affine forms will be interpreted from an statistical point of view, since their error terms will be assigned probability density functions (PDFs). This will lead to the estimation of SQNR by means of AA.

5 SQNR Estimation

In this section, an AA-based method able to estimate the SQNR of non-linear algorithms with and without feedback loops is presented. The method is able to extract an estimation of the power of the quantization noise of a system from an AA-based simulation. This would not be of much use for a fast FPO if AA simulations must be repeated during the WL selection phase (see Figure 1). The ultimate goal of the method is to use a single AA simulation to extract a model of the quantization noise that enables fast SQNR estimation, so it supports fast FPO.

5.1 Affine arithmetic applied to error propagation analysis

Noise estimation is based on the assumption that the quantization of a signal i from n_{pre} bits to n bits can be modeled by the addition of a uniformly distributed white noise with the following statistical parameters [11]:

$$\sigma_i^2 = \frac{2^{2p_i}}{12}\left(2^{-2n_i} - 2^{-2n_i^{pre}}\right) \tag{13}$$

$$\mu_i = -2^{p_i-1}\left(2^{-n_i} - 2^{-n_i^{pre}}\right). \tag{14}$$

This noise model is a refinement of the traditional modeling of the quantization error as an additive white noise [12] and, therefore, it is more accurate. The values of p are obtained during the scaling phase and the values of n and n_{pre} are computed during word-length selection by means of WL propagation (see the optimization flow in Figure 1).

The deviation from the original behavior of an algorithm with feedback loops caused by quantized signals can be modeled by adding an affine form $\hat{n}_i[k]$ to each signal i at each simulation time instant k (i.e. loop iteration index) [5]. These affine forms can properly model the quantization noise of each signal if the error term ϵ is assigned a uniform distribution:

$$\hat{n}_i[k] = \mu_i + \sqrt{12\sigma_i^2}\epsilon_{i,k} = \epsilon'_{i,k} \tag{15}$$

Thus, an AA simulation automatically identifies the origin of any particular error term (i) and the moment when it was generated (k). Error term ϵ' encapsulates the mean value and the variance of the error term ϵ, that now can be seen as a random variable with variance σ_i^2 and mean μ_i. Thus, the AA-based simulation can be made independent on the particular statistical parameters of each quantization. This is highly desirable in order to obtain a parameterizable noise model. In fact, this is a reinterpretation of AA, since error terms are not only intervals, but they also have an associated probability distribution. Once the simulation is finished, it is possible to compute the impact of the any quantization noise produced by signal s_i on the output of the algorithm by extracting the value of the error amplitude (given by $x_{i,k}$ in eqn. (1)). This enables the parameterization of the noise. Once the parameterization is performed, the quantization error produced by any combination of (p, n) can be easily assessed by replacing all $\epsilon'_{i,k}$ by the original expression that accounts for the mean and variance ($\mu_i + \sqrt{12\sigma^2}\epsilon_{i,k}$), thus enabling a fast estimation of the quantization error. We present in detail the complete process in the next paragraphs.

Given an algorithm with $|S|$ signals, the expression of output $\hat{Y}$ is

$$\hat{Y}[k] = Y_0[k] + \sum_{i=0}^{|S|-1}\sum_{j=0}^{k} Y_{i,j}[k]\epsilon'_{i,j}, \tag{16}$$

where $Y_0[k]$ is the value of the output of the algorithm using floating-point arithmetic and the summation is the contribution of the quantization noise sources.

Note that the error term amplitude $Y_{i,j}[k]$ is a function that depends on the inputs of the algorithm for non-linear systems.

The error $\hat{Err}_Y$ at the output is

$$\hat{Err}_Y[k] = Y_0[k] - \hat{Y}[k] = -\sum_{i=0}^{|S|-1}\sum_{j=0}^{k} Y_{i,j}[k]\epsilon'_{i,j}, \tag{17}$$

and it is formed by a collection of affine forms at each time step n. This expression of the error can be used to estimate the error peak-value, using the traditional interpretation of AA [20, 10], and, also, to obtain the PDF of the error [5, 20]. However, in the next subsection we focus on obtaining the value of the power of the error considering the actual PDF of each error term (i.e. a uniform density function).

5.2 Analytical SQNR estimation

The power of the quantization noise is computed as the Mean Square Error (MSE), that is, the mean value of the expectancy of the power of the summations of $\epsilon_{i,j}$ during K time steps.

$$\begin{aligned} P\left(\hat{Err}_Y\right) &= \frac{1}{K}\sum_{m=0}^{K-1} E\left[\left(\hat{Err}_Y[m]\right)^2\right] \\ &= \frac{1}{K}\sum_{m=0}^{K-1}\Big(Var(\hat{Err}_Y[m]) + \\ &\qquad E\left[\hat{Err}_Y[m]\right]^2\Big) \end{aligned} \tag{18}$$

The two main terms in eqn. (18) are developed in (19) and (20). The former makes use of the fact that it can be assumed that error terms $\epsilon'_{i,k}$ are uncorrelated to each other [12].

$$\begin{aligned} Var(\hat{Err}_Y[m]) &= Var(-\sum_{i=0}^{|S|-1}\sum_{j=0}^{m} Y_{i,j}[m]\epsilon'_{i,j}) \\ &= \sum_{i=0}^{|S|-1}\sum_{j=0}^{m} Var(-Y_{i,j}[m]\epsilon'_{i,j}) \\ &= \sum_{i=0}^{|S|-1}\sigma_i^2\sum_{j=0}^{m} Y_{i,j}^2[m] \end{aligned} \tag{19}$$

$$E[\hat{Err}_Y[m]] = E[-\sum_{i=0}^{|S|-1}\sum_{j=0}^{m} Y_{i,j}[m]\epsilon'_{i,j}]$$
$$= -\sum_{i=0}^{|S|-1}\mu_i \sum_{j=0}^{m} Y_{i,j}[m] \tag{20}$$

Combining (18), (19) and (20):

$$P\left(\hat{Err}_Y[k]\right) = \frac{1}{K}\sum_{m=0}^{K-1}\left(\sum_{i=0}^{|S|-1}\sigma_i^2\sum_{j=0}^{m} Y_{i,j}^2[m] + \left(\sum_{i=0}^{|S|-1}\mu_i\sum_{j=0}^{m} Y_{i,j}[m]\right)^2\right) \tag{21}$$

Equation (21) can be expressed in vectorial form (eqns. (22-24)). The statistical parameters of the quantized signals are in vectors $\boldsymbol{s} = \left\langle \sigma_0^2 \dots \sigma_{|S|-1}^2 \right\rangle$ and $\boldsymbol{\mu} = \left\langle \mu_0 \dots \mu_{|S|-1} \right\rangle$. Once vector $\boldsymbol{v}$ and matrix M are computed –during the noise parameterization phase– the estimation of the quantization noise does not require any further AA simulations, but the computation of (22), which is a much faster process. Also, note that we are not actually computing the SQNR, but the power of the output signal. The SQNR can be easily computed from the power of the output signal using infinite precision (i.e. floating-point double precision) and applying the formula $SQNR = 20 \cdot log(P_{Y_0}/P_E)$).

$$P_E = \frac{1}{K}\left(\boldsymbol{s}\cdot\boldsymbol{v}^T + \boldsymbol{\mu}\cdot M\boldsymbol{\mu}^T\right) \tag{22}$$

$$\boldsymbol{v} \equiv \left\langle \sum_{k=0}^{K-1}\sum_{j=0}^{k} Y_{0,j}^2[k], \dots, \sum_{k=0}^{K-1}\sum_{j=0}^{k} Y_{|S|-1,j}^2[k] \right\rangle \tag{23}$$

$$M \equiv \begin{bmatrix} m_{0,0} & \cdots & m_{|S|-1,0} \\ & \ddots & \\ m_{0,|S|-1} & \cdots & m_{|S|-1,|S|-1} \end{bmatrix} \tag{24}$$

$$m_{i_1,i_2} = \sum_{k=0}^{K-1}\left(\sum_{j_1=0}^{k} Y_{i_1,j_1}[k]\sum_{j_2=0}^{k} Y_{i_2,j_2}[k]\right) \tag{25}$$

Algorithm 1 Gradient-descent optimization

Input: $G_{SFG}(V, S)$ with no fixed-point information, error constraint E_{max}
Output: word-length optimized $G_{SFG}(V, S)$

Perform *scaling* and initialize $P = \{p_0, \ldots, p_{|S|-1}\}$
Find minimum n that complies with noise constraint E_{max}
when $\forall n_i \in N = \{n_0, \ldots, n_{|S|-1}\}, n_i = n$
Compute output error E
$C_{min} = \infty$
repeat
  $s_{candidate} = undefined$
  for all $s_i \in S$ **do**
    $n_i = n_i - 1$
    Compute output error E
    if $E < E_{max}$ **then**
      Compute cost C
      if $C < C_{min}$ **then**
        $C_{min} = C$
        $s_{candidate} = s_i$
      end if
    end if
    restore n_i
  end for
  if $s_{candidate}\neg =$ undefined **then**
    $n_{candidate} = n_{candidate} - 1$
  end if
until $s_{candidate}\neg =$ undefined

5.3 Accuracy for LTI systems

The purpose of the proposed fast estimation method is to estimate the quantization noise produced by non-linear algorithms. However, it is important to verify that eqn. (22) matches the well-known expression that is used to compute the output noise power of a quantized LTI system in steady state. This verification can be found in [21], where it is analytically proven that if eqn. (22) is used for LTI systems by removing the first J samples (to remove the transient) then

$$\begin{aligned} P_{LTI}\left(\hat{E}rr_Y\right) &= \frac{1}{K-J}\sum_{m=J}^{K-1} E\left[\left(\hat{E}rr_Y[m]\right)^2\right] \\ &\approx \sum_{i=0}^{|S|-1} \sigma_i^2 \cdot \frac{1}{2\pi}\int_{-\pi}^{\pi} \left|G_i(e^{j\Omega})\right|^2 d\Omega \\ &\quad + \left(\sum_{i=0}^{|S|-1} \mu_i \cdot G_i(1)\right)^2 . \end{aligned} \tag{26}$$

The basis of this demonstration is that it is possible to relate the error terms amplitudes ($Y_{i,j}[k]$) to the transfer function from signal i to the output (G_i).

This proves that the accuracy of the presented estimation is very high for LTI systems. Nonetheless, the method presented here is not intended for LTI systems, since there are already more efficient ways to compute their output noise [2, 5, 21]. The accuracy for non-linear algorithms is presented in the next section.

6 Fixed-point optimization tool

FPO can now be accelerated by means of the estimator presented in the previous section. Here, we outline how the FPO process must be performed and we present the implementation of the process as an automatic design tool.

6.1 Estimation-based optimization

Before carrying out the task of WL selection, it is necessary to parameterize the quantization noise at the output of the system. The parameterization process is carried out by means of the following steps:

1. Perform a $K-$step AA simulation adding an affine form $\hat{n}_i$ to each signal i.
2. Compute eqns. (23-25) using the previously collected $Y_{i,j}[k]$.

Once vector $\boldsymbol{s}$ and matrix $\boldsymbol{M}$ are available, expression (22) can be used during the optimization process to assess the system quality.

Many optimization techniques can be applied to the problem of finding the appropriate word-lengths. Here, we present a gradient-descent optimization [22] that provides a trade-off between low complexity and optimality. Its behavior is described using pseudo-code in Algorithm 1. Given an algorithm with $|S|$ signals, it performs $|S|$ independent tests, where the WL of each signal is reduced one bit and the resulting error (E) and cost (C) are recorded. The test producing the largest cost reduction among those tests that comply with the error constraint ($E < E_{MAX}$) is selected and the WL change performed in that test is made permanent. The algorithm goes on until it is not possible to reduce the WLs without violating the error constraint.

Regarding the cost function, some authors use area [22, 2, 23, 21], others the error itself [24] and others a linear combination of both [25]. The reader can extend the information on FPO techniques consulting [26, 24, 27, 28].

6.2 Software implementation

The proposed fast estimator, as well as the FPO methodology, were implemented using C++. Operator overloading was used in order to enable the execution of the same algorithm description using different data types. For instance, during *scaling*, a floating-point simulation that collects the dynamic range information for each signal must be performed. Also, during the *error modeling* phase, an AA simulation is used to extract parameters $\boldsymbol{v}$ and $\boldsymbol{M}$. During WL selection, it is necessary to know the set N_{pre} that is obtained through *WL propagation*.

Square.h

```
class Square::QAlgorithm{
...
set_signals(){
   x.set_input(_UNQUANTIZED, _ROUND);
   y.set_output(_QUANTIZED, _TRUNC);
}
set_inputs(){
for (int i=0; i<x.data.size(); i++)
   x.data[i] = rand()/(RAND_MAX+1);
}
exe(){   c=a*a; }
...
Quant_signal x;
Quant_signal y;
...
}
```

FPO.cpp

```
#include "FPO.h"
#include "Square.h"
main(){
 FPO fpo;
 Square sqr(fpo);

 fpo.set_simulation(10000);
 sqr.init();    // call set_signals,set_inputs,etc.
 fpo.scaling();
 fpo.error_model();
 fpo.set_sqnr(80.0); // SQNR=80 dB
 fpo.optim(_GRAD_DESCENT, _AA);
 double pow_AA = fpo.get_pow(_AA)
 double sqnr_AA = fpo.get_sqnr(_AA)
 double pow_FX = fpo.get_pow(_FXP) // FxP simulation
 double sqnr_FX = fpo.get_sqnr(_FXP) // FxP simulation
 double pow_err = 100.*(pow_FX-pow_AA)/pow_FX;
 double sqnr_err = sqnr_FX-sqnr_AA;
 cout<<"Estimation error: "<<pow_err<<"%; ";
 cout<<sqnr_err<<" dB" << endl;
}
```

Fig. 4. Example of use of the software framework.

During this phase, the FxP format of each variable was used to perform WL propagation. And, finally, the last FxP simulation used to assess the estimator quality requires the use of a FxP data type.

A framework to coordinate the different FPO phases, and to allow the designer to configure the optimization process to quantize the algorithm, was also developed. Figure 4 displays a simple example. File `Square.h` contains the description of the algorithm. The designer must specify the way that the signals (e.g. inputs, outputs and signals) are treated during FPO (see function `set_signals`). They can be quantized or not, and also different rounding schemes can be applied (i.e. rounding or truncation). The designer must also specify the original WL of inputs. Also, the input values must be fed through `set_inputs`. The operation sequence is described in `exe`. The next step for the designer is to declare an object for the algorithm and also to declare an object `FPO` that encapsulates all the FPO features (see `main.cpp`). The FPO process is straight forward, since there are methods for the main FPO tasks: `scaling`, `error_model`, `optim` (see Figure 1). Note that the parameters of method `fpo.optim()` are the type of optimization technique (i.e. `_GRAD_DESCENT` is for a gradient-descent optimization such as Algorithm 1) and the type of error computation (i.e. `_AA` for AA-based estimation, and `_FXP` for a FxP simulation-based estimation). Needless to say that the use of simulations requires much longer processing times than AA-based optimizations. In the last few lines of the code, the quantization noise power and the SQNR are computed using the AA estimations (i.e.

`get_pow(_AA)` and `get_sqnr(_AA)`) and FxP simulations (i.e. `get_pow(_FXP)` and `get_sqnr(_FXP)`), and the estimation error is assessed (see subsection 7.2).

The implemented software framework has been used to quantize different benchmarks, presented in subsection 7.1, proving the validity of the techniques proposed in this chapter.

7 Results

In this section, the benchmarks used to test our fast estimator, as well as the performance results are presented.

7.1 Benchmarks

The benchmarks are the following:

- 3×3 vector scalar multiplication ($VEC_{3\times3}$)
- 8×8 vector scalar multiplication ($VEC_{8\times8}$)
- Mean power estimator based on a 1^{st}-order IIR filter (POW)
- Channel equalizer for MIMO receiver (EQ) [29]
- 1^{st}-order LMS filter (LMS_1) [30]
- 5^{th}-order LMS filter (LMS_5) [30]
- 3^{rd}-order Volterra filter (VOL_3) [31]

The main features of the benchmarks are summarized in Table 2, which contains the type of algorithm (LTI or non-linear, with or without loops), the number of inputs/outputs, the number and type of operations involved (z^{-1} representing delays and $*K$ constant multiplications), and the total number of signals ($|S|$). The set of benchmarks covers non-linear algorithms, both recursive (with feedback loops) and non-recursive. It must be noted that the set of operations is quite complete since it includes additions, multiplications, and also divisions, usually neglected in similar research studies. In addition to that, it is interesting to highlight that the algorithms cover channel equalization for 4G MIMO communications, vector multiplications and adaptive filtering with both linear and non-linear systems.

All benchmarks are fed with 16-bit inputs and 12-bit constants and the noise constraint is specifed as an SQNR ranging from 40 to 120 dB. The inputs used to perform the noise parameterization as well as steps of the fixed-point simulation are summarized in the last column of the table.

Vector Scalar Multiplication. Due to the importance of vector and matrix operations in the development of scientific applications, an $N \times N$ vector scalar multiplication is included as a benchmark. Given two N-element vectors a and b, the scalar product is defined as:

$$\begin{pmatrix} a_0 \dots a_{N-1} \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ \vdots \\ b_{N-1} \end{pmatrix} = \sum_{i=0}^{N-1} a_i \cdot b_i \ . \tag{27}$$

Table 2. Properties of benchmarks.

Benchmark	LTI	Cyclic	Inputs	Outputs	z^{-1}	+/-	*	$*K$	$\div$	$\|S\|$	Input signals
$VEC_{3\times3}$	NO	NO	6	1	0	2	3	0	0	12	Uniform noise
$VEC_{8\times8}$	NO	NO	16	1	0	7	8	0	0	32	Uniform noise
POW	NO	YES	1	1	1	1	1	2	0	7	Synthetic tone
EQ	NO	YES*	3	2	64	2	7	3	3	85	MIMO channel Tx [29]
LMS_1	NO	YES	2	1	3	4	4	2	0	16	Synthetic tone
LMS_5	NO	YES	2	1	11	12	12	6	0	44	Synthetic tone
VOL_3	NO	YES	2	1	2	4	6	4	0	19	Gaussian noise

* MAC operations applied to chuncks of 32 data

The arithmetic operations involved are: N multiplications and $N-1$ additions. The inputs used to extract the noise model are uniformly distributed noises.

IIR Mean Power Computation. The mean power computation is based in the use of an IIR filter. The equation describing its behavior is:

$$y[k] = x[k]^2 \cdot \alpha + y[k-1] \cdot \beta \tag{28}$$

where $x[k]$ is the input to the filter and $y[k]$ is the output. The constants α and $\beta = 1-\alpha$ fix the time length of the mean calculation. The operations used in the algorithm are one multiplication, two constant multiplications and one addition. It must be stressed that the filter contains a delay that conforms a loop.

The input used for this benchmark is a phase modulated tone signal with an uniform noise added.

LMS Adaptive Filtering. Adaptive filtering is widely used in many DSP applications. In particular, the Least Mean Squares (LMS) adaptive filter is a common solution due to its low computational load in comparison to other adaptive approaches. A reference signal d is estimated by means of output y:

$$y[k] = \sum_{i=0}^{N} x[k-i] \cdot w_i, \tag{29}$$

where x is the input to the filter and w_i are its coefficients. The coefficients are updated in every time step by means of constant μ:

$$w_i^{next} = w_i + \mu \cdot x[k-i](d[k] - y[k]). \tag{30}$$

An Nth-order LMS filter requires $2N+2$ multiplications, $N+1$ constant multiplications, $2N+1$ additions, 1 subtraction and $2N+1$ delays. It also contains loops.

Two input signals must be specified here: the reference d is a synthetic tone signal with phase and amplituted noises, and the input x is signal d with N added echoes.

Equalizer for Alamouti MIMO Communications System. The selected equalizer aims at 4G communications and it is embedded in a Multi-Carrier Code-Division Multiple-Access (MC-CDMA) radio system [29], which is able to handle up to 32 users and provides transmission bit-rates up to 125 Mbps. The transmitter sends complex data Y_i using different subcarriers (i is the subcarrier index) and the receiver combines the data received by the different antennas to produce complex Z_i, which is the input of the equalizer. This signal is related to Y_i by means of the real signal H_i, which is contains information about the communication channel, and N_i is a noise term signal.

$$Z_i = H_i \cdot Y_i + N_i \tag{31}$$

The output of the equalizer is an estimate of Y_i:

$$Y_i = G_i \cdot Z_i. \tag{32}$$

The coefficients G_i can be extracted using H_i and constant λ, which is related to the Signal to Noise Ratio conditions.

$$G_i = \frac{1}{H_i + \lambda} \cdot \frac{S_F}{\sum_{j=i \mod S_F}^{(i \mod S_F)+S_F-1} \frac{H_j}{H_j+\lambda}} \tag{33}$$

Notice that the second factor is constant for every group of S_F consecutive subcarriers. For this particular application $S_F = 32$.

The equalizer can be implemented using seven multiplications, three constant multiplications, three divisions and two additions. It is interesting to highlight that the presence of multiplications and divisions make this algorithm highly non-linear.

The input signals H and Z are generated using a realistic MIMO channel model [29].

Volterra Adaptive Filtering. Volterra adaptive filters are used when the non-linearities present in the system that is being approximated cannot be neglected. The behavior of the filter is similar to that of LMS, but now, the computation of output y involves a non-linear equation. In particular, a Hermite polynomial series is used for the estimation:

$$y[k] = \sum_{i=0}^{K} H_i(\bar{x}) \cdot w_i, \tag{34}$$

where H_i is the Hermite polynomial of order i, $\bar{x} = \frac{x}{P_x}$ is the normalized input to the filter, P_x is the power of signal x and w_i are its coefficients. The coefficients are updated each time step by means of constant μ:

$$w_i^{next} = w_i + \mu \cdot H_i(\bar{x})(d[k] - y[k]). \tag{35}$$

Here, we address the estimation of the function $arctan(x)$ by means of a 3^{rd}-order Hermite expansion. The expression of $y[k]$ for such a Volterra adaptive

filter is:

$$y[k] = \left(\frac{x[k]}{P_x}\right) \cdot w_1 + \left(\left(\frac{x[k]}{P_x}\right) - 3\left(\frac{x[k]}{P_x}\right)^3\right) \cdot w_3 \tag{36}$$

Table 2 displays the number of operations required for the 3^{rd}-order Volterra filter. The inputs used are as follows: signal x is a gaussian noise and d is $arctan(x)$.

7.2 Experimental setup

All the benchmarks were used to test both the accuracy and computation performance of the proposed FPO method. The following procedure was carried out:

- For *all benchmarks* do
  1. Compute scaling by means of a floating-point simulation.
  2. Extract noise parameters (eqns. 23-25) performing an AA-based simulation.
  3. Record computation time $T^{Param-AA}$.
  4. For $SQNR = [3, \ldots, 120]$ do
     (a) Perform a WL selection based on the fast estimator (eqn. 22) using a gradient-descent approach.
     (b) Record computation time $T_{SQNR}^{Optim-AA}$, estimation P_{SQNR}^{AA} and the number of optimization iterations I_{SQNR}.
     (c) Perform a single FxP bit-true simulation of the quantized algorithm and use it as reference to compute the performance and accuracy of the estimator.
     (d) Record computation time of a single fixed-point simulation (T_{SQNR}^{FxP}).
     (e) Record an estimate of a simulation-based FPO ($T_{SQNR}^{Optim-FxP} = T_{SQNR}^{FxP} \times I_{SQNR}$) and the simulation-based SQNR (P_{SQNR}^{FxP}).
  5. Compute the average values $\bar{T}^{Optim-AA}$ and $\bar{T}^{Optim-FxP}$.

7.3 Accuracy results

The accuracy obtained by means of a gradient-descent FPO [2] under different SQNR constraints for the different benchmarks is presented in Table 3. A total of 80 different SQNR constraints were used, ranging from 40 dB to 120 dB. The first column indicates the benchmark used. The remaining columns show the accuracy of the estimations measured in terms of the maximum absolute values of the relative errors in dB, and the average of the absolute values of the percentage errors, for four SQNR ranges: [120,100] dB, [100,80] dB, [80, 60] dB and [60,40] dB (see the expressions of the metrics at the bottom of the table). The last row contains the maximum and average value using the information from all benchmarks

Table 3. Performance of the estimation method: precision.

Benchmark	Estimation error							
	[120,100][1] dB		[100,80] dB		[80,60] dB		[60,40] dB	
	(dB)[2]	(%)[3]	(dB)	(%)[3]	(dB)[2]	(%)[3]	(dB)[2]	(%)[3]
$VEC_{3\times3}$	0.07	0.54	0.07	0.11	0.06	0.50	0.09	0.72
$VEC_{8\times8}$	0.05	0.57	0.04	0.40	0.04	0.57	0.13	1.19
POW^*	0.27	0.98	0.24	0.71	0.29	0.17	0.18	1.52
EQ^*	0.39	5.00	0.17	1.55	0.76	5.96	1.12	12.12
LMS_1^*	0.09	0.41	0.14	0.90	0.16	1.74	0.82	6.96
LMS_5^*	0.09	0.46	0.08	0.07	0.13	1.08	1.09	5.51
VOL_3^*	1.14	3.33	0.49	1.84	0.81	6.70	1.43	16.67
All	0.39	1.27	0.24	0.05	0.76	1.48	1.12	4.21

* Recursive

[1] Error constraint

[2] $|10log(\frac{P_{ref}}{P_{est}})|$ (max)

[3] $|100(\frac{P_{ref}-P_{est}}{P_{ref}})|$ (average)

Results show that the estimator is very accurate. The mean percentage error (see last row) is smaller than 4.3 %, and the maximum relative error is smaller than 1.12 dB. Note that the accuracy decreases as long as the error constraints get looser. This is due to the amplification of the Taylor error terms (specially in the presence of loops) and also to the fact that the uniformly distributed model for the quantization noise does not remain valid for small SQNRs. Anyway, the quality of the estimates is still very high, thus confirming the excellent accuracy of the estimator. The accuracy of recursive algorithms is slightly reduced, since the estimation errors are somehow amplified by the feedback loops. The estimation errors for benchmark EQ appear to be greater that the rest, probably due to the presence of both divisions and feedback loops.

Figure 5 displays the mean estimation error vs. the target SQNR for the benchmarks $VEC_{3\times3}$ and POW. Note that the target SQNR range has been extended to $[3, 120]$ dB. These two algorithms present similar non-linearities. The former performs the summation of the multiplication of three pairs of numbers, while the latter performs the accumulation of the square of a signal. The main difference between them is the presence of a feedback in POW. First, it can be seen in the figure that as long as the SQNR decreases the error in the estimation increases. This is expected since the quantization model relies on the fact that the quantization error is much smaller than the dynamic range of the signal. $VEC_{3\times3}$ presents an error smaller than 20% for the whole SQNR range. However, POW achieves similar errors for SQNR values smaller than 65 dB, but for SQNR values smaller than 30 dB, the error reaches values close to 100%. As aforementioned, the error introduced by the 1_{st}-order Taylor approximation becomes magnified in the pressence of feedbacks.

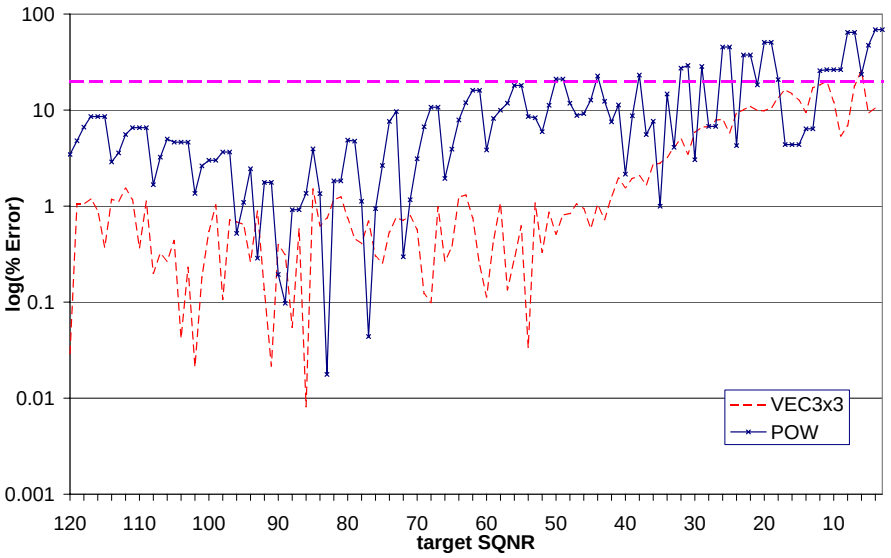

Fig. 5. Estimation error (%) vs. target SQNR for $VEC_{3\times3}$ and POW.

Figure 6 shows a similar graph for benchmarks LMS_5, VOL_3 and EQ. LMS presents a strong feedback with a smooth non-linearity (i.e. multiplication). VOL_3 has a similar feedback (somehow smaller, since the order is smaller), but the non-linearities are stronger (i.e. x^3). Finally, EQ presents the more abrupt non-linearity (i.e. several divisions) but the feedback is not as prominent, since the accumulations are reset every 32 clock cycles. The three of them perform correctly for $SQNR = [65, 120]$. When $SQNR = [40, 65]$, approximately, the error is greater than 20%. For SQNR values smaller than 30 the performance is really poor, presenting VOL_3 and EQ the worst errors (i.e. larger than 100%).

It is interesting to see that the *amount* of non-linearity in the algorithm clearly impacts on the quality of the error estimation. VOL_3 performs very bad for small SQNR values in comparison to LMS_5. However, EQ performs well for most of the SQNR range, since the feedback effect is limited, but for very small SQNRs the non-linearity of division shows off, causing estimation errors up to 10^5. Since the quantization model is expected to fail for small SQNRs, this situation is not seen as an anomaly of the estimator, but as an intrinsic characteristic that it not possible to overcome. As shown in Table 3 the method works properly in the range $[40, 120]$.

7.4 Computation time results

Table 4 holds the computation times required for both noise parameterization and word-length selection. The first column shows the names of the benchmarks. The second one shows the length of the input vectors required for a fixed-point

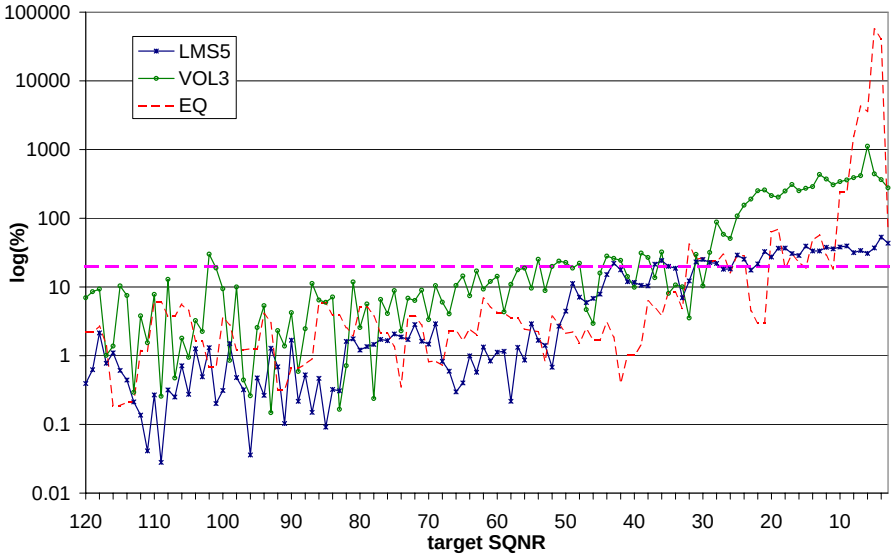

Fig. 6. Estimation error (%) vs. target SQNR: LMS_5, VOL_3 and EQ.

simulation and for the parameterization process. The parameterization time is in the third column. The fourth column characterizes the complexity of the process through the number of estimates obtained during the optimization. Each iteration implies a noise estimation (using a simulation or our fast estimator). The next two columns present the computation time required to perform the gradient-descent optimization using our estimation-based proposal and using a classical simulation-based approach. The computation time for the simulation-based approach is an estimation obtained from multiplying the average number of optimization iterations by the computation time of a single fixed-point simulation. The speed-up obtained by our estimation-based approach is in the last column. The last row contains the average speedup considering all experiments.

The parameterization time goes from 59.66 μsecs. to 28 mins. (1646 secs.) and it depends on the size of the input dataset, the complexity of the algorithm (i.e. number and types of operations) and the presence of feedback loops. The cases including multiplications show how an increase in complexity implies an increase in parameterization time. This situation is more acute in the presence of loops (see LMS_1 and LMS_5). However, the effect of algorithm complexity in FPO time is negligible. These times might seem quite long, but it must be kept in mind that the parameterization process is performed only once, and after that the algorithm can be evaluated for different fixed-point formats as many times as desired using the fast estimator.

The mean number of estimates in the fifth column is shown to give an idea of the complexity of the optimization process. A simulation-based optimization approach would require that very same number of simulations, thus taking a very

Table 4. Performance of the estimation method: computation time.

Bench.	FxP Samples	$T^{Param-AA}$ (secs)$^+$	No. of estimates (mean)	$\bar{T}^{Optim-AA}$ (secs)$^+$	$\bar{T}^{Optim-FxP}$ (secs)$^+$	Speed-up
$VEC_{3\times3}$	$2 \cdot 10^4$	59.66	150.14	0.03	66.86	$\times$2122
$VEC_{8\times8}$	$2 \cdot 10^4$	330.67	1739.96	1.72	2331	$\times$1377
POW^*	$2 \cdot 10^4$	546.14	97.15	0.02	21.93	$\times$1048
EQ^*	$16 \cdot 10^3$	61.64	231.98	0.12	105.78	$\times$904
LMS_1^*	$5 \cdot 10^3$	908.02	712.28	0.42	163.73	$\times$394
LMS_5^*	$5 \cdot 10^3$	1646.38	2547.48	7.26	1611.46	$\times$221
VOL_3^*	$5 \cdot 10^3$	212.72	673.38	0.29	151.13	$\times$526
All	-	-	-	-	-	$\times$942

* With feedback

$^+$ On a 1.66 GHz Intel Core Duo, 1 GB of RAM

long time. For instance, the optimization of LMS_5 would approximately require 2500 FxP simulations of 5000 input data. Considering the number of estimations required, the optimization times are extremely fast, ranging from 0.02 secs to 7.26 secs. The speedups obtained in comparison to a simulation-based approach are staggering: boosts from $\times$221 to $\times$2122 are obtained. The average boost is $\times$942 which proves the advantage of our approach, not only in terms of accuracy but also in terms of computation time. Therefore, our approach enables fast and accurate FxP of non-linear DSP algorithms.

8 Conclusions

A fast and accurate SQNR estimation method based on the use of Affine Arithmetic has been presented. The estimator is used within a fixed-point optimization framework and fast quantization is achieved. Affine-arithmetic is used during the noise parameterization phase. The estimator can be used to perform complex FPO in reduced times, leading to significant hardware cost reductions. The method can be applied to differentiable non-linear DSP algorithms with and without feedbacks.

Summarizing, the main contributions of the chapter are:

- The proposal of a fast quantization noise estimation, based on affine arithmetic, for non-linear algorithms with and without feedbacks.
- The introduction of a methodology and an automatic design tool to perform fast FPO.
- The average estimation error for non-linear systems is smaller than 17% for all examples, and smaller than 7% for most cases.
- The computation time of FPO is boosted up to $\times$2122 (average of $\times$942).

Future research will pursue higher accuracy in the estimation of non-linear operations, probably by extending the presented approach to include additional terms of the Taylor series expansion. The goal is not only to improve further the accuracy of the method as presented, but to enable its application to algorithms with strong non-linearities.

9 Acknowledgments

This work was supported in part by Research Projects USP-BS PPC05/2010 (Banco Santander and University CEU San Pablo) and TEC2009-14219-C03 (Spanish Ministry of Science and Innovation).

References

1. W. Sung and K.-I. Kum, "Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems," *IEEE Trans. Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.
2. G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Wordlength Optimization for Linear Digital Signal Processing," *IEEE Trans. Computer-Aided Design*, vol. 22, no. 10, pp. 1432–1442, 2003.
3. G. Caffarena, G. Constantinides, P. Cheung, C. Carreras, and O. Nieto-Taladriz, "Optimal Combined Word-Length Allocation and Architectural Synthesis of Digital Signal Processing Circuits," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 5, pp. 339–343, 2006.
4. G. Constantinides and G. Woeginger, "The Complexity of Multiple Wordlength Assignment," *Applied Mathematics Letters*, vol. 15, pp. 137–140, 2002.
5. J. López, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "Fast and accurate computation of the roundoff noise of linear time-invariant systems," *IET Circuits, Devices & Systems*, vol. 2, no. 4, pp. 393–408, 2008.
6. D. Menard and O. Sentieys, "A Methodology for Evaluating the Precision of Fixed-Point Systems," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2002. Proceedings.*, vol. 3, pp. 3152–3155, 2002.
7. G. Constantinides, "Perturbation Analysis for Word-Length Optimization," in *Proc. FCCM*, pp. 81–90, 2003.
8. D. Menard, R. Rocher, P. Scalart, and O. Sentieys, "SQNR Determination in Non-Linear and Non-Recursive Fixed-Point Systems," in *Proc. EUSIPCO*, pp. 1349–1352, 2004.
9. C. Shi and R. Brodersen, "A Perturbation Theory on Statistical Quantization Effects in Fixed-Point DSP with Non-Stationary Inputs," in *Proc. ISCAS*, vol. 3, pp. 373–376 Vol.3, 2004.
10. D.-U. Lee, A. Gaffar, R. Cheung, W. Mencer, O. Luk, and G. Constantinides, "Accuracy-Guaranteed Bit-Width Optimization," *IEEE Trans. Computer-Aided Design*, vol. 25, no. 10, pp. 1990–2000, 2006.
11. G. Constantinides, P. Cheung, and W. Luk, "Truncation Noise in Fixed-Point SFGs," *IEE Electronics Letters*, vol. 35, no. 23, pp. 2012–2014, 1999.
12. L. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," *IEEE Trans. Audio Electroacoust.*, vol. 18, pp. 107–122, Jun 1970.

13. R. Rocher, D. Menard, O. Sentieys, and P. Scalart, "Analytical Accuracy Evaluation of Fixed-Point Systems," in *Proc. EUSIPCO*, pp. 999–1003, 2007.
14. J. Stolfi and L. H. Figueiredo, "Self-Validated Numerical Methods and Applications," in *Brazilian Mathematics Colloquium: IMPA*, 1997.
15. B. Hayes, "A Lucid Interval," *American Scientist*, vol. 91, no. 6, pp. 484–488, 2003.
16. J. López, *Evaluación de los Efectos de Cuantificación en las Estructuras de Filtros Digitales Mediante Técnicas de Simulación Basadas en Extensiones de Intervalos*. PhD thesis, Universidad Politécnica de Madrid, 2004.
17. J. López, C. Carreras, and O. Nieto-Taladriz, "Improved Interval-Based Characterization of Fixed-Point LTI Systems With Feedback Loops," *IEEE Trans. Computer-Aided Design*, vol. 26, pp. 1923–1933, Nov. 2007.
18. C. Fang, T. Chen, and R. Rutenbar, "Floating-Point Error Analysis Based on Affine Arithmetic," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 561–564, 2003.
19. J. López, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "Analysis of Limit Cycles by Means of Affine Arithmetic Computer-aided Tests," in *Proc. EUSIPCO*, pp. 991–994, 2004.
20. C. Fang, R. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," *Int. Conf. on Computer Aided Design.*, pp. 275–282, Nov. 2003.
21. G. Caffarena, *Combined Word-Length Allocation and High-Level Synthesis of Digital Signal Processing Circuits*. PhD thesis, Universidad Politécnica de Madrid, 2008.
22. H. Choi and W. Burleson, "Search-based Wordlength Optimization for VLSI/DSP Synthesis," in *IEEE Workshop VLSI Signal Processing*, pp. 198 –207, 1994.
23. G. Caffarena and C. Carreras, "Architectural synthesis of DSP circuits under simultaneous error and time constraints," pp. 322 –327, sept. 2010.
24. M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, "An Automatic Word Length Determination Method," in *IEEE Int. Symp. on Circuits and Systems*, vol. 5, (Sydney, Australia), pp. 53–56 vol. 5, 2001.
25. K. Han, B. Evans, and E. Swartzlander, "Data Wordlength Reduction for Low-Power Signal Processing Software," in *IEEE Workshop on Signal Processing Systems*, pp. 343–348, 2004.
26. F. Catthoor, J. Vandewalle, and H. De Man, "Simulated Annealing Based Optimization of Coefficient and Data Word-Lengths in Digital Filters," *J. Circuit Theory Applications*, vol. 16, pp. 371–390, Sept. 1988.
27. T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: architectures and design methods," *Computers and Digital Techniques, IEE Proceedings* -, vol. 152, pp. 193 – 207, mar 2005.
28. K. Han and B. Evans, "Optimum Wordlength Search Using Sensivity Information," *EURASIP Journal of Applied Signal Processing*, vol. 2006, pp. 1–14, 2006. doi:10.1155/ASP/2006/92849.
29. A. Fernández, A. Jimenez, G. Caffarena, and J. Casajús, "Design and Implementation of a Hardware Module for Equalisation in a 4G MIMO Receiver," in *Proc. FPL*, pp. 765–768, 2006.
30. S. Haykin, *Adaptive Filter Theory*. Upper Saddle River: Prentice-Hall, 2002.
31. T. Ogunfunmi, *Adaptive Nonlinear System Identification: The Volterra and Wiener Approaches*. Springer, 2007.