Efficient Implementation
of Systems With Adaptive Filters

Marta Filipa Pirão Freire

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. Teresa Maria Canavarro Menéres Mendes de Almeida

Examination Committee

Chairperson: Prof. Gonçalo Nuno Gomes Tavares
Supervisor: Prof. Teresa Maria Canavarro Menéres Mendes de Almeida
Member of the Committee: Prof. Paulo Ferreira Godinho Flores

November 2015
Acknowledgments

First and foremost I would like to thank my supervisor Professor Teresa Mendes de Almeida for giving me the opportunity to do this project and for always giving me support and motivation in every stage of this work.

I would also like to thank my parents for always helping and support me and for giving me the opportunity to proceed my academic path. Finally but not least, I would thank my brother for always challenging me to do better and keeping me focused on what is really important.
Abstract

Digital filters are commonplace and an essential element of everyday life devices, such as mobile phones, radios, hearing aids, etc. The digital filter is designed to fulfill the required specifications either in the frequency domain or time domain of stationary input signals. Nonetheless, in most cases these input signals do not remain stationary, and thus is necessary to consider an adaptive filter, whose coefficients are adjusted according to the signal's variations and characteristics using an adaptive algorithm. Nevertheless, the design of these filters are more complex than in digital filters with fixed coefficients. The implementation of a digital signal processing system is always with a finite number of bits thus will be introduced errors due to the quantizations. The sources of these errors can be the analogue-to-digital converters and the arithmetic operations performed. If the system implemented has a very large number of arithmetic operations, as an adaptive system, these errors increase further more. Nevertheless, these quantization errors can be reduced if increased the resolution used in the conversion and in the arithmetic operations, but it will also increase the price of the system and it cannot reduce significantly the error. Thus it is of great importance to find solutions to reduce the quantization error from the arithmetic operations in fixed-point of an adaptive filter independent of the type of hardware chosen to implement.

Keywords

adaptive filter, LMS algorithm, fixed-point arithmetic, quantization error
Resumo

Filtros digitais estão presentes em variados dispositivos electrónicos do dia-a-dia, tais como telemóveis, rádios, aparelhos auditivos, etc. O filtro digital é projectado para cumprir as especificações requeridas quer seja no domínio da frequência ou do tempo relativamente a sinais de entrada estacionários. Porém, em muitos casos o sinal de entrada não se mantém estacionário, e é então necessário usar um filtro adaptativo, cujos coeficientes são reajustados de acordo com as variações e características do sinal através de um algoritmo adaptativo. Contudo, o projecto deste tipo de filtros é mais complexo do que os filtros digitais com coeficientes fixos. A implementação de um sistema de processamento de sinal digital é sempre realizada com um número finito de bits que irá introduzir erros de quantificação. As fontes de erro são os conversores analógico-digital e as operações aritméticas realizadas. Se o sistema digital implementado tiver que realizar muitas operações aritméticas, como por exemplo um filtro adaptativo, estes erros irão aumentar consideravelmente. No entanto, estes erros de quantificação podem ser reduzidos se aumentar a resolução na conversão ou nas operações aritméticas, porém também irá aumentar o preço do sistema e poderá não reduzir significativamente o erro. Assim, é de grande importância encontrar soluções para reduzir os erros de quantificação das operações aritméticas em vírgula fixa de um filtro adaptativo independentemente do hardware escolhido para implementar.

Palavras Chave

filtro adaptativo, algoritmo LMS, aritmética de vírgula fixa, erro de quantificação
Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
  1.1 Motivation ........................................... 1
  1.2 Outline ........................................... 2

2 Digital Filters 3
  2.1 Digital Filters with Fixed Coefficients ................. 3
    2.1.1 Recursive and Non-recursive Filters ................ 3
    2.1.2 Filter Structures ................................ 4
  2.2 Digital Signal Processing System ....................... 5
    2.2.1 Digital Signal Processing System of a Digital Filter 5
    2.2.2 Finite Precision Arithmetic ....................... 6
    2.2.3 Quantization Noise ................................ 7
  2.3 Finite Number Representation .......................... 9
    2.3.1 Floating-Point Representation ..................... 9
    2.3.2 Fixed-Point Representation ....................... 10
  2.4 Round-off Operation ................................ 11
    2.4.1 Truncation ..................................... 11
    2.4.2 Rounding ...................................... 12
  2.5 Coefficient Quantization ................................ 13
  2.6 Product Quantization ................................ 16
  2.7 Effects of Finite Precision Arithmetic in a Digital Filter 20
  2.8 Conclusion ........................................ 24

3 Adaptive Filters 25
  3.1 The Wiener Filter .................................... 26
  3.2 The Gradient Descent Algorithm ......................... 27
    3.2.1 Convergence .................................... 28
  3.3 The Least Mean Squares Algorithm ....................... 29
    3.3.1 Convergence .................................... 30
  3.4 Applications of Adaptive Processing .................... 31
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Direct form structure of a $N^{th}$-order FIR filter.</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Direct form transposed structure of a $N^{th}$-order FIR filter.</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Cascade structure of a $N^{th}$-order FIR filter with $L$ second-order sections.</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Simplified block diagram of the digital signal processing system of a digital filter.</td>
<td>6</td>
</tr>
<tr>
<td>2.5</td>
<td>Simplified models of a quantizer: (a) block diagram; (b) statistical additive model.</td>
<td>7</td>
</tr>
<tr>
<td>2.6</td>
<td>Probability density function of the truncation error.</td>
<td>7</td>
</tr>
<tr>
<td>2.7</td>
<td>Probability density function of the rounding error.</td>
<td>7</td>
</tr>
<tr>
<td>2.8</td>
<td>Two's complement number circles for 3 bits: (a) for integer numbers; (b) for fractional numbers ($Q_2$-format).</td>
<td>10</td>
</tr>
<tr>
<td>2.9</td>
<td>Example of quantization by truncation for $B = 255$ and $T = 2$: (a) quantized values $\hat{x}$ represented in bold; (b) quantization error.</td>
<td>11</td>
</tr>
<tr>
<td>2.10</td>
<td>Example of quantization by rounding for $B = 255$ and $R = 2$: (a) quantized values $\hat{x}$ represented by the solid line; (b) quantization error due to rounding.</td>
<td>12</td>
</tr>
<tr>
<td>2.11</td>
<td>Parallel equivalent model of a digital filter with quantized coefficients.</td>
<td>13</td>
</tr>
<tr>
<td>2.12</td>
<td>Effects of coefficient quantization using rounding in the gain response of a bandpass FIR filter of $118^{th}$-order, for 8 and 16 bits resolution.</td>
<td>15</td>
</tr>
<tr>
<td>2.13</td>
<td>Effects of coefficient quantization using truncation in the gain response of a bandpass FIR filter of $118^{th}$-order, for 8 and 16 bits resolution.</td>
<td>15</td>
</tr>
<tr>
<td>2.14</td>
<td>Additive noise model of quantization error of a multiplication operation of two constants.</td>
<td>16</td>
</tr>
<tr>
<td>2.15</td>
<td>Relative error of the quantized result of $c = a \times b$, for all values of $a$ and $b = 1/3$, using 8 bits for $a$, $b$ and $c$.</td>
<td>19</td>
</tr>
<tr>
<td>2.16</td>
<td>Relative error of the quantized result of $c = a \times b$, for all values of $a$ and $b = 1/3$, using 16 bits for $a$, $b$ and $c$.</td>
<td>19</td>
</tr>
<tr>
<td>2.17</td>
<td>Additive noise model of quantization error on a filtering operation of a non-recursive filter of a $0^{th}$-order.</td>
<td>20</td>
</tr>
<tr>
<td>2.18</td>
<td>Additive noise model of quantization error on a filtering operation of a $1^{st}$-order non-recursive filter.</td>
<td>21</td>
</tr>
<tr>
<td>2.19</td>
<td>Input signal of the digital filter and respective input signal quantized for 8 and 16 bits.</td>
<td>22</td>
</tr>
<tr>
<td>2.20</td>
<td>Output signal of the digital filter and respective output signal quantized for 8 and 16 bits.</td>
<td>22</td>
</tr>
<tr>
<td>2.21</td>
<td>Power spectral density of the output signal of the digital filter: (a) ideal ($THD = -63.78$ dB); (b) quantized for 8 bits ($THD = -31.02$ dB); (c) quantized for 16 bits ($THD = -63.71$ dB).</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>Adaptive filter with recursive structure in the direct form.</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Block diagram of a system with a Wiener filter $W$.</td>
<td>26</td>
</tr>
</tbody>
</table>
6.4 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of adaptive noise cancellation with scaled coefficients. ................. 63

6.5 Results of the implementation of adaptive noise cancellation with both scaled coefficients and using an accumulator with extended resolution. ......................... 63

6.6 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of adaptive noise cancellation with scaled coefficients and an accumulator with extended resolution. ........ 64

6.7 Results of the implementation of adaptive system identification with accumulator with extended resolution for different resolutions. .................. 66

6.8 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with accumulator with extended resolution. .... 66

6.9 Results of the implementation of adaptive system identification with scaled coefficients for the different resolutions. ........................................... 67

6.10 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with scaled coefficients. ............... 67

6.11 Results of the implementation of adaptive system identification with both scaled coefficients and using an accumulator with extended resolution. .................. 68

6.12 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with scaled coefficients and an accumulator with extended resolution. .................. 68

6.13 Recent models of noise cancelling headphones commercialized by Bose: (a) around-ear headphone, (b) in-ear headphone. ........................................... 72

6.14 Chart of the market prices of normal and noise cancelling headphones for different brands. ............. 73

6.15 Simplified scheme of a noise cancelling headphones. ........................................... 74

6.16 Block diagram of the equivalent noise cancellation system included in headphones. .................. 74

6.17 Results of the full precision implementation of noise cancellation system using a music signal and white noise. ........................................... 76

6.18 Results of the fixed-point implementation of noise cancellation system using a music signal and white noise. ........................................... 76

6.19 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of noise cancellation system. ........................................... 77

6.20 Results of the fixed-point implementation of noise cancellation system with the combined solution to cancel noise in a music signal. ........................................... 77

6.21 Quantization error of the error signal \( e(n) \) for 8, 16 and 32 bits resolution for the implementation of noise cancellation system with the combined solution. ........................................... 78
List of Tables

2.I Characteristics of standard floating-point representations. 9
2.II Examples of coefficient quantization for 4 and 8 bits using rounding or truncation. 14
2.III Example 1 of a multiplication of \( a = 0.087 \) and \( b = 0.183 \) with result quantized to 8 bits using truncation or rounding. 17
2.IV Example 2 of a multiplication of \( a = 0.936 \) and \( b = 0.61 \) with result quantized to 8 bits using truncation or rounding. 18
2.V Relative error for quantized values of \( c = a \times b \), for \( a = 1/5 \) and \( b = 1/3 \), using either truncation or rounding. 18
2.VI Relative error for quantized values of \( c = a \times b \), for \( a = 1/50 \) and \( b = 1/3 \), using either truncation or rounding. 18
2.VII Relative error for quantized values of \( c = a \times b \), for \( a = 0.98 \) and \( b = 1/3 \), using either truncation or rounding. 19
6.I Signal-to-quantization-noise ratio (dB) in each implementation and resolution for the adaptive noise cancellation system. 61
6.II Signal-to-quantization-noise ratio (dB) in each implementation and resolution for adaptive system identification. 69
6.III Characteristics and prices of suitable processors to implement noise cancellation system. 70
6.IV Signal-to-quantization-noise ratio (dB) in each implementation and resolution for noise cancellation system for noise cancelling headphones. 75
Chapter 1

Introduction

1.1 Motivation

Nowadays, digital filters are commonplace and an essential element of everyday life devices, such as mobile phones, radios, etc. In this way, digital signal processing systems are of great importance, and in some applications digital filters may provide better results than their counterpart analogue filters. The digital filter is designed to fulfil the required specifications either in the frequency domain or time domain, as for example, a notch filter to remove an undesired specific interference or noise. Generally this noise or interference signal is merged with the desired signal. To remove this noise with an ordinary FIR or IIR filter, the noise must be stationary, i.e. must have the same characteristics over time, so the designed filter meet the noise specifications in order to cancel it. Besides the interfering noise signal should not overlap in the frequency domain with the desired signal, in order for the signal no to be cancelled by the FIR/IIR filter.

Nonetheless, in most cases these signals do not remain stationary, and thus it is necessary to consider an adaptive filter, whose coefficients are adjusted according to the variations and characteristics of the signal using an adaptation algorithm. The ability of digital systems based on adaptive filters to update their characteristics may be applicable in many different areas, such as noise cancellation, system identification or prediction of signals. Nevertheless, the design of these filters is more complex than in digital filters with fixed coefficients. This processing need to be in real-time, and therefore the system have to be fast to adapt to the changes of the input signals. Hence the implemented algorithm must be stable and have a minimum convergence time. The filter implementation and the chosen algorithm will determine the performance of the system.

Whenever filtering a signal through the implementation of a digital filter, with fixed or variable coefficients, quantization errors will always be present either from analogue-to-digital conversion or from the arithmetic operations with finite word-length numeric representation. These quantizations errors affect the performance of the filter. Since the adaptation algorithm consists in more arithmetic operations besides those present in a digital filter with fixed coefficients, there will be more sources of quantization error and reduce drastically the performance of the adaptive filter. Thus, the objective of this work is to study and envisage efficient implementations for systems with adaptive filters in order to reduce the effects of finite precision arithmetic. The concept of efficient implementation imply a fast and flexible implementation in low resolution with fixed-point representation.
From a practical point of view an efficient digital implementation with reduced word-length arithmetic is of most importance such as in the case of common people listening to music on their way to work. Since portable audio player devices have been commercially available, they have become very popular among users. The ability to hear any recorded audio track, anywhere, changed somehow the concept of "listening to music". In daily tasks or routines, such as the way from work to home, seating in a coffee shop or in the park, travelling inside an aircraft, etc., anyone could hear and enjoy own favourite music. However, with these portable audio players a problem arises: all the environment noise interferes with the audio. The background traffic noise, machinery noise, engine aircraft noise, and others, are always present in the user’s ears. The design of headphones changed along the way to oppose to this problem, with sound absorbing materials and different types of ear cups.

Notwithstanding, these techniques are not entirely efficient, and still a part of the noise reaches the user’s ears. Along with the evolution of technology, it was possible to design headphones that reduce the noise in an adaptive way, to acceptable levels, leading to better sound quality. Despite this advance and the good performance of these products, these headphones are very expensive, and they have not become the first solution to common users that just want to hear their favourite music in their way to work. Using common models of integrated circuits to process digital signals, it is possible to find a solution that allows a similar performance as the one of commercially available professional headphones, yet cheaper, so that an everyday music enthusiastic can appreciate own music at its full quality at a reasonable or even reduced cost.

1.2 Outline

The thesis is organized into seven chapters. After this introductory chapter, Chapter 2 establishes the basis on digital filters, finite word-length arithmetic and their effect on digital filter implementation. On Chapter 3 the basis on adaptive filter theory is presented, as well as some illustrative practical applications based on this type of filters. Chapter 4 presents a review of the existing possibilities for the implementation of digital systems based on adaptive filters. On Chapter 5 the specific effect of finite precision arithmetic on adaptive filters is addressed. Chapter 6 presents results about different implementation solutions. Finally, in Chapter 7, the conclusions of the work are presented. The thesis is completed with the bibliographic references.
Chapter 2

Digital Filters

A filter can be seen as a linear time invariant system, which changes amplitude, frequency and/or phase characteristics of an input signal according to the filter specifications [1], establishing a relationship between the input and output signals. A digital filter corresponds to the filtering algorithm implemented in a digital domain, such as in hardware or software [1]. There are two classes of digital filters: recursive and non-recursive, which are named according to their structure.

Digital filters are in most cases advantageous comparing with their counterpart analogue filters, as for example the digital filter is not affected by environment changes as the analogue filters, and in some applications may even provide better results. However when implementing a digital filter there is always a finite number of bits for numeric representation, meaning that every value is an approximation of the real number representing, for example, a sample of a real signal. This will produce errors in the filtering algorithm which can then affect the performance of the filter [1].

In this chapter is presented both classes of digital filters with fixed coefficients, characteristics of a digital signal processing system, binary numeric representations and respective arithmetic operations and how the finite wordlength will affect the performance of a filter.

2.1 Digital Filters with Fixed Coefficients

2.1.1 Recursive and Non-recursive Filters

A linear digital filter with fixed coefficients can be characterized by its impulse response, defined as [2]:

$$ h(n) = \sum_{k=0}^{\infty} h_k \delta(n - k) , \quad n = 0, 1, ... , $$

(2.1)

where $\delta(n - k)$ represents an unit impulse at $n = k$, or by a linear difference equation, where the filter output $y(n)$ is defined as [1]:

$$ y(n) = \sum_{k=0}^{N} a_k x(n - k) + \sum_{k=1}^{M} b_k y(n - k) , \quad n = 0, 1, ... , $$

(2.2)
which means that the filter output depends on the last \((N + 1)\) input samples and the last \(M\) output samples. In this perspective, the transfer function of the filter in the \(z\) domain can also be expressed as [1]:

\[
T(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} a_k z^{-k}}{1 - \sum_{k=1}^{M} b_k z^{-k}},
\]

with \(N \leq M\) for a causal filter. Based on this definition, a filter can be classified as non-recursive or recursive. A non-recursive filter has all \(b_k = 0\), its output depending only in the input samples, and its impulse response always has finite duration. These filters are then referred as finite impulse response (FIR) filters [1]. A recursive filter has, at least, one of the \(b_k\) coefficients different from zero. Most of these filters have an impulse response with infinite duration, and by this reason are referred as infinite impulse response (IIR) filters [1], although sometimes they may have a finite impulse response.

An IIR filter has an infinite impulse response as defined in Equation 2.1 and its output depends on both input and output samples, and its transfer function is defined as in Equation 2.3 [1]. Analysing the transfer function, an IIR filter has \(N\) zeros and \(M\) poles. A digital filter to be stable must have all the poles inside the unit circle in \(z\) domain \((|z| = 1)\), and since the IIR filter has a recurring structure with output feedback it may lead to instability if the filter poles lie outside the unit circle.

A FIR filter output only depends on its input samples, then the transfer function of a \(N^{th}\)-order filter is [1]:

\[
T(z) = \sum_{k=0}^{N} a_k z^{-k} = \frac{\sum_{k=0}^{N} a_k z^{N-k}}{z^N},
\]

and has a finite impulse response, defined as:

\[
h(n) = \sum_{k=0}^{N} h_k \delta(n - k) \quad , \quad n = 0, 1, ... .
\]

Equation 2.4 shows that a FIR filter has \(N\) zeros and \(N\) poles at the origin \((z = 0)\). Since the stability of a filter depends on the position of the poles inside the unit circle, any \(N^{th}\)-order FIR filter is always stable.

### 2.1.2 Filter Structures

To implement a filter it is important to understand its structure. FIR filters are characterized for having a transversal structure [1]. In Figure 2.1 it is represented a generic \(N^{th}\)-order FIR filter structure in direct form, which represents one of the most used structures in signal processing, where each box \(T\) denotes a delay of one sample. In this structure, to compute each output sample, \(y(n)\), it is needed space in memory to store all \((N + 1)\) coefficients and \((N + 1)\) delayed input samples, and \((N + 1)\) multiplications and \(N\) additions of parcels of two numbers will be computed, establishing the computational complexity for the filter implementation.

However, there are other types of structures to implement a FIR filter. One is the direct form transposed [1], as presented in Figure 2.2. To calculate the output sample it has the same computation complexity as the direct form. On the other hand, the cascade structure is composed by a product of \(L\) second-order
sections which is equivalent to a $N^{th}$-order FIR filter, as shown in Figure 2.3, defined as [1]:

$$ T(z) = h(0) \prod_{k=1}^{L} 1 + \beta_{1k}z^{-1} + \beta_{2k}z^{-2} . $$

(2.6)

Although in infinite precision arithmetic the $L$ second-order sections would be equivalent to the $N^{th}$-order FIR filter, with finite precision arithmetic different filter structures will produce different results [1].

IIR filters have recursive structures, which are traditionally classified as direct form I and direct form II [1]. The direct form I is obtained directly from the difference equation (Equation 2.2). On the other hand, the direct form II (a canonic form) is obtained by transposing and rearranging the direct form I in order to have a minimum number of delays. Both structures are presented in [1], [4] and [3]. The structures of IIR filters are not presented in detail here because further ahead the adaptive filters presented are of type FIR due to guarantee stability of the system.

### 2.2 Digital Signal Processing System

#### 2.2.1 Digital Signal Processing System of a Digital Filter

To implement a digital filter is needed a digital signal processing system, as exemplified in Figure 2.4, where is presented a simplified block diagram. Most often the signals to be processed are analogue
signals, so there is necessary to convert the signal to the digital domain. Thus, it is always present in a digital signal processing system an analog-to-digital converter (ADC) [5]. After the input signal is converter it is computed through arithmetic operations in finite numeric representations in the block digital filter. After finished the processing, the output signal needs to be transmitted into the analogue domain, a digital-to-analogue converter (DAC) is used [5].

In the conversion process from the analogue to the digital domain it is carried out a sampling process, in which, at a sampling rate \( f_s = 1/T_s \), a sample of the continuous time signal it is taken. In this process it is important to convey the Sampling Theorem [5], which implies the sampling rate to be higher than twice the maximum frequency of the signal, \( f_s = 2f_M \), in order to prevent aliasing.

According to Figure 2.4, initially the continuous input signal, \( x(t) \), must go through a lowpass filter with cut-off frequency at \( f_s/2 \), to remove all frequency components higher than \( f_s/2 \) to prevent aliasing, having then \( x_A(t) \) [2]. Then the sampling process is performed by a sample-and-hold (S&H), which takes a sample of \( x_A(t) \) at each \( nT_s \) seconds and hold it until \( (n+1)T_s \), producing the signal \( x_A(nT_s) \) [2]. After the discretization in time, the signal goes through an analogue-to-digital converter (ADC) mapping the amplitudes of the signal into the quantization levels, \( \hat{x}_A(n) \) [2]. After the filtering process, when it is available the output signal, \( \hat{y}(n) \), this signal is then converted to continuous time by the digital-to-analogue converter (DAC), producing the signal \( \hat{y}(t) \) [2]. At a last step, the signal goes through a lowpass reconstructing filter with cut-off frequency at \( f_s/2 \) to remove high frequencies and "smoothing" the signal in order to obtain the analogue output signal, \( y(t) \) [2].

The amplitude of the discrete time signal is also quantized according to the resolution of the converter, i.e., the number of bits. The number of bits will determine the levels of quantization to which the amplitudes of the signal will be mapped [3]. The resolution of the converter is finite and therefore limits the precision of the amplitudes representation. The levels of quantization are determined by the number of bits \( B+1 \) of the converter, and if the input signals have amplitudes between \( -A \) and \( A \), the quantization step, \( \Delta \), is defined as [4]:

\[
\Delta = \frac{2A}{2^{B+1}-1} \approx \frac{2A}{2^{B+1}},
\]

which, for the specific case of amplitudes with \( |A| = 1 \), the quantization step will be equal to:

\[
\Delta = 2^{-B}.
\]

### 2.2.2 Finite Precision Arithmetic

All the data in the digital domain is processed in a binary representation with a finite number of bits. No matter the number of bits used, this fact will always translate in approximations of real values, however
Figure 2.5
Simplified models of a quantizer (adapted from [1]): (a) block diagram; (b) statistical additive model.

Figure 2.6
Probability density function of the truncation error.

Figure 2.7
Probability density function of the rounding error.

a higher number of bits means less difference between the digital representation and the real value [1]. Every computation, as in a digital filter implementation, will not have infinite precision, therefore will introduce noise or error through the digital system. The first step where this happen is in the converters from the analogue to digital domain, which in signal processing is inevitable [1]. Moreover, during arithmetic operations in a digital system it can occur other types of errors which then can cause [1]: (1) changes in the filter frequency response because the coefficients are quantized; (2) noise in the filter output due to the quantization of multiplications outputs, and (3) accumulator overflow (when the result is greater than the dynamic range).

2.2.3 Quantization Noise

Any quantization (Figure 2.5(a)), either from converters or in arithmetic operations, introduces an error to the signal. The process of quantization can be seen as an addition of noise or error to the real number or with higher precision, as presented in Figure 2.5(b), and can be expressed as [3]:

\[
Q[a] = a + \varepsilon = \hat{a}.
\]

(2.9)

The quantization error can have multiple sources but is according to the round-off operation used. Consider the following assumptions for the quantization error \(\varepsilon\) from Figure 2.5(b) [4]: (1) it can be seen as
sequence of a stationary random process; (2) is uncorrelated with the input $a$; (3) the random variables
of the error process are uncorrelated and (4) has an uniform probability distribution. Regarding this, the
quantization error $\epsilon$ is uniformly distributed in the interval $[-\Delta, 0]$, as shown in Figure 2.6, if performed
truncation [4]. From the probability density function of the truncation error the expected value and the
variance are defined as [4]:

$$\mu_T = -\frac{\Delta}{2} , \quad (2.10)$$

$$\sigma_T^2 = \frac{1}{\Delta} \int_{-\Delta}^{0} \epsilon^2 = \frac{\Delta^2}{12} . \quad (2.11)$$

When using rounding as round-off, the assumptions taken in the truncation operation apply in the same
way to the rounding quantization error. Therefore, the quantization error has a probability density func-
tion uniform between $[-1/2\Delta, 1/2\Delta]$ [4] (see Figure 2.7). So, the expected value and the variance of the
rounding operation are [4]:

$$\mu_R = 0 , \quad (2.12)$$

$$\sigma_R^2 = \frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} \epsilon^2 = \frac{\Delta^2}{12} . \quad (2.13)$$

This quantization noise affects the digital signal quality when compared with the original analogue signal
from a similar analogue system. The signal-to-quantization-noise ratio (SQNR) provides information to
analyse the effects of the quantization noise in the performance of the digital system. The signal-to-
quantization-noise ratio is defined as [3]:

$$SQNR = 10 \log \left( \frac{P_S}{P_Q} \right) , \quad (2.14)$$

where $P_S$ is the power of the input signal and $P_Q$ the power of the quantization noise.

In any type of digital system is always present quantizations due to ADC’s and it is possible to understand
the effect of these converters in the signals if not considered all the other sources of quantization noise,
as arithmetic operations. So if considered the case where the converter is ideal and perform rounding
when mapping the input samples, as seen previously, the power of the quantization error is then [3]:

$$P_Q = \frac{\Delta^2}{12} , \quad (2.15)$$

which if replaced in Equation 2.14, the signal-to-quantization-noise ratio becomes:

$$SQNR = 4.77 + 6.02B + 10 \log P_S \ [\text{dB}] , \quad (2.16)$$

for a normalized $A = 1$. With this definition it is easily concluded that for each one more bit in the
converter resolution adds approximately $6 \text{ dB}$ to the signal-to-quantization-noise ratio.
2.3 Finite Number Representation

2.3.1 Floating-Point Representation

Every value and arithmetic operation in a digital signal processing system is represented with a finite number of bits. The most common finite number representations used are fixed-point and floating-point. Floating-point numbers have variable step size and have a very high dynamic range [2]. The floating-point number has a sign bit, an exponent $E$ and a mantissa $M$. The floating-point format for signed numbers is expressed as:

$$x = s e_{N_e-1} \ldots e_1 e_0 \cdot m_1 m_2 \ldots m_{N_m},$$

(2.17)

where $s$ represents the sign bit, $N_e$ the number of bits of the exponent, $e_0$ the binary point and $N_m$ the number of bits of the mantissa. The exponent is expressed by:

$$E = \sum_{k=0}^{N_e-1} e_k 2^k,$$

(2.18)

and the mantissa can be determined as:

$$M = \sum_{k=1}^{N_m} m_k 2^{-k}.$$

(2.19)

The most used floating-point format was set by the Institute of Electrical and Electronic Engineers (IEEE), which consists of a 32 bits format, where the exponent have 8 bits and the mantissa 23 bits. If the exponent $E = 2^8 - 1 = 255$ and $M \neq 0$ then $x$ is NaN (Not a Number), without regard to the sign bit, but if $M = 0$ then $x = (-1)^s \times \infty$. However if $E = 0$ and $M = 0$ then $x = (-1)^s \times 0$. If the exponent is $0 \leq E \leq 255$ then $x$ is calculated by:

$$x = (-1)^s 2^{E-127}(1 + M).$$

(2.20)

Besides this 32 bits standard format, there are devices using other floating-point formats referred as double precision and double extended, whose characteristics are presented in Table 2.I.

<table>
<thead>
<tr>
<th></th>
<th>Single</th>
<th>Double</th>
<th>Double Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bits</td>
<td>32</td>
<td>64</td>
<td>80</td>
</tr>
<tr>
<td>Exponent bits</td>
<td>8</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Mantissa bits</td>
<td>23</td>
<td>52</td>
<td>64</td>
</tr>
<tr>
<td>Maximum Positive</td>
<td>$\approx 2^{128}$</td>
<td>$\approx 2^{1024}$</td>
<td>$\approx 2^{16384}$</td>
</tr>
<tr>
<td>Minimum Positive</td>
<td>$\approx 2^{-126}$</td>
<td>$\approx 2^{-1022}$</td>
<td>$\approx 2^{-16382}$</td>
</tr>
</tbody>
</table>

Table 2.I
Characteristics of standard floating-point representations [2].
2.3.2 Fixed-Point Representation

In the digital domain, data can be represented with the fixed-point representation, where the binary point is fixed and the quantization step is constant. Fixed-point numbers can be expressed in several formats such as sign-and-magnitude, one’s complement or two’s complement [3]. The most used format is the two’s complement in which hereafter will be the only format considered.

In a two’s complement representation, an integer number with \((B + 1)\) bits can be expressed as [3]:

\[
x = b_B b_{B-1} ... b_1 b_0 = -b_B 2^B + \sum_{k=0}^{B-1} b_k 2^k ,
\]

(2.21)

where the most significant bit \(b_B\) represents the sign of the number. In this case the range of numbers to be represented is \([-2^B, 2^B - 1]\). To represent a real number in fixed-point representation can be used \(I\) bits for the integer part and with \(F\) bits for the fractional part:

\[
x = b_I b_{I-1} ... b_1 b_0 . b_{-1} b_{-2} ... b_{-F} ,
\]

(2.22)

where once again, the most significant bit \(b_{I-1}\) corresponds to the sign of the number and \(\cdot\) represents the binary point. In most digital filter implementations, coefficients and signals are scaled to the range \([-1, 1]\], and it can be used a fixed-point fractional format with \((B + 1)\) bits represented as [4]:

\[
x = b_0 . b_{-1} b_{-2} ... b_{-B} = -b_0 + \sum_{k=1}^{B} b_{-k} 2^{-k} ,
\]

(2.23)

where bit \(b_0\) is the sign bit. The range of values to be represented in this format is \([-1, 1 - 2^{-B}]\), as easily deducted from Equation 2.23. This fractional fixed-point representation is usually referred as the format-\(Q_B\). From now on, when referred fixed-point representation with \((B + 1)\) bits it means that it is used the format-\(Q_B\).

In Figure 2.8 is an example of 3 bit representation \((B = 3)\) in two’s complement, for both integer numbers and in the format-\(Q_2\), rearranged in a circle of numbers. This graphic representation maybe used to explain that, in some cases, when computing multiple additions, in intermediate results it can occur overflow, where the result goes from one side of the circle to the other (from positive to negative values or vice-versa) but, nevertheless the final result of the additions may be already inside the representation range and so, correct regardless intermediate overflows.

![Figure 2.8](image)

**Figure 2.8**

Two’s complement number circles for 3 bits (adapted from [2]): (a) for integer numbers; (b) for fractional numbers (\(Q_2\)-format).
2.4 Round-off Operation

Besides the converter quantization, while performing arithmetic operations the result may have to be quantized to the resolution of the hardware, including then errors throughout the processing [4]. To quantize a number from higher to lower resolution a round-off operation is used, which can be either truncation or rounding [4].

2.4.1 Truncation

One way to represent a number with less resolution is by truncation. Considering a number with $(B + 1)$ bits, $x$, that is then truncated to $(T + 1)$ bits, $\hat{x}$, discarding then the least significant $(B - T)$ bits, can be represented as [4]:

\[
x = b_0 . b_{-1}b_{-2}...b_{-(T-1)}b_{-T}b_{-(T+1)}...b_{-(B-1)}b_{-B},
\]

\[
\hat{x} = b_0 . b_{-1}b_{-2}...b_{-(T-1)}b_{-T} = -b_0 + \sum_{k=1}^{T} b_{-k}2^{-k}.
\]

When a number is truncated to $(T + 1)$ bits, each value between $n\Delta$ and $(n + 1)\Delta$ is mapped to $n\Delta$ [4], where $n$ identifies a level of quantization between $-2^T$ and $2^T - 1$, and:

\[
\Delta = 2^{-T}.
\]

In Figure 2.9 is shown an example on how the values are mapped to the quantization levels and the quantized values are always below the values with higher resolution. From here it is easy to deduce the quantization error from truncation, defined as [4]:

\[
\varepsilon_T = \hat{x} - x = - \sum_{k=T+1}^{B} b_{-k}2^{-k},
\]

![Figure 2.9](image)

**Figure 2.9**

Example of quantization by truncation for $B = 255$ and $T = 2$: (a) quantized values $\hat{x}$ represented in bold; (b) quantization error.
and the range of the quantization error behind truncation as [4]:

\[-(2^{-T} - 2^{-B}) \leq \epsilon_T \leq 0\]  \hspace{1cm} (2.28)

which, if \( B \gg T \) can be approximated to:

\[-2^{-T} \leq \epsilon_T \leq 0\]  \hspace{1cm} (2.29)

### 2.4.2 Rounding

If a number of \((B+1)\) bits, \(x\), has to be quantized to \((R+1)\) bits, \(\hat{x}\), with \(R < B\), the procedure is different but, in a certain sense, similar to truncation, because the least significant \((B-R)\) bits are discarded [4].

\[x = b_0 \cdot b_{-1} b_{-2} \ldots b_{-(R-1)} b_{-(R+1)} \ldots b_{-(B-1)} b_{-B} \]  \hspace{1cm} (2.30)

\[\hat{x} = b_0 \cdot b_{-1} b_{-2} \ldots b_{-(R-1)} b_{-R} \]  \hspace{1cm} (2.31)

However, in the case of rounding, the bit \((R+1)\) will define if the quantized number will be rounded to the quantization level above or to the quantization level under the number with higher resolution. Based on this, a value between \((n-1/2)\Delta\) and \((n+1/2)\Delta\) is mapped to \(n\Delta\) [4], where \(n\) identifies a level of quantization between \(-2^R\) and \(2^R - 1\), and:

\[\Delta = 2^{-R} \]  \hspace{1cm} (2.32)

In opposition to the truncation case, the rounded quantized values can be above or under the number with higher resolution (see Figure 2.10(a)). The range of the quantization error (see Figure 2.10(b)) relative to rounding is [4]:

\[-\frac{1}{2} \left(2^{-R} - 2^{-B}\right) < \epsilon_R \leq \frac{1}{2} \left(2^{-R} - 2^{-B}\right) \]  \hspace{1cm} (2.33)

**Figure 2.10**

Example of quantization by rounding for \(B = 255\) and \(R = 2\): (a) quantized values \(\hat{x}\) represented by the solid line; (b) quantization error due to rounding.
and if $B >> R$ the range of the quantization error due to rounding can be approximated to:

$$-\frac{1}{2}2^{-R} < \varepsilon_R \leq \frac{1}{2}2^{-R}.$$  \hfill (2.34)

## 2.5 Coefficient Quantization

If a digital filter is to be implemented, initially the filter has to be designed to meet the desired specifications. Usually the design is performed mathematically, so the coefficients of the filter have very high precision. However, when the filter is to be implemented in a digital processor the coefficients must be quantized to the hardware resolution [1], meeting a finite word-length representation.

Considering the project of a $N^{th}$-order filter with $(N + 1)$ coefficients:

$$H(z) = \sum_{k=0}^{N} h_k z^{-k}, \quad (2.35)$$

where $h_k$ are the filter coefficients, mathematically obtained at the design stage with high precision. When the coefficients are quantized to a binary word of finite word-length, the filter becomes then defined as [1]:

$$\hat{H}(z) = \sum_{k=0}^{N} \hat{h}_k z^{-k}, \quad (2.36)$$

and, as analysed before, the quantization can be seen as an addition of error or noise, thus [1]:

$$\hat{H}(z) = \sum_{k=0}^{N} (h_k + \varepsilon_k) z^{-k}, \quad (2.37)$$

where each $\varepsilon_k$ is the quantization error of the coefficient $h_k$. With these errors, besides having the desired filter, it is possible to consider the presence of another one, $E(z)$, in parallel with the designed one [1], as shown in Figure 2.11:

$$\hat{H}(z) = \sum_{k=0}^{N} h_k z^{-k} + \sum_{k=0}^{N} \varepsilon_k z^{-k} = H(z) + E(z). \quad (2.38)$$

This will imply that if the coefficients are now different, so the filter frequency response will also be different from the designed one, even before any computation of signals take place. Coefficient quantization can create problems if the specifications of the quantized filter response no longer meet the desired specifications. In Figure 2.12 and Figure 2.13 are shown an example of a designed digital filter and the same digital filter with quantized coefficients, using rounding or truncation, respectively. In the example it is shown the gain response of a digital bandpass FIR filter of 118th-order with passband between 2 kHz and 4 kHz, 0.1 dB of passband ripple and 60 dB of attenuation in both stopbands, for a sampling

![Figure 2.11](image-url)

Parallel equivalent model of a digital filter with quantized coefficients (adapted from [1]).
frequency of 40 kHz. The quantized coefficients of the filter were determined according to the binary representation in two’s complement with a resolution of 8 bits, \( \hat{H}_8(z) \), and 16 bits, \( \hat{H}_{16}(z) \). For 8 bits resolution and using rounding, the quantized coefficients produce a frequency response which does not met the specifications for the filter gain response, neither in the passband or stopbands. With 16 bits resolution, the passband of the quantized filter is very close to the designed one, but have small changes in the stopbands, where in some frequencies the attenuation specified it is not met. If using truncation instead of rounding, as presented in Figure 2.13, the quantized filter with 8 bits resolution has more ripple in the passband and lower attenuation in both stopbands, proving that truncation generates more quantization error than using rounding. Therefore, it is important to understand the effect of these errors in the performance of the filter. To reduce the impact on the filter frequency response rounding can be used as round-off for the coefficients, since it will originate smaller quantization error.

If the filter with quantized coefficients does not meet the desired specifications it will be necessary an implementation with more resolution or increase the filter order to guarantee the desired specifications. The effect of the coefficient quantization by truncation or rounding introduces an error whose range of values is the one considered in the previous section (Equation 2.29 and Equation 2.34). The quantized coefficients can be determined based in the binary representation in two’s complement according with the available binary word \((B + 1)\) bits. For each coefficient of the filter \( a_k \) \((-1 \leq a_k < 1)\):

\[
\hat{a}_k^R = \text{round}\left\{ a_k \times 2^B \right\} \quad \frac{2^B}{2^B},
\]

and:

\[
\hat{a}_k^T = \text{floor}\left\{ a_k \times 2^B \right\} \quad \frac{2^B}{2^B}.
\]

The same result can be obtained if considered a binary representation in two’s complement with \((N + 1)\) bits of resolution, which \(N > B\):

\[
a_k = -b_0 + \sum_{k=1}^{N} b_k 2^{-k},
\]

and then perform the round-off operation to \(B\) bits at bit level.

In Table 2.11 are presented three illustrative examples of coefficient quantization for 4 and 8 bits \((B + 1)\) bits), using truncation, \(\hat{a}_k^T\), or using rounding, \(\hat{a}_k^R\), with their respective relative error, \(\delta_T = (a_k^T - a_k)/a_k\) and \(\delta_R = (a_k^R - a_k)/a_k\). It is shown the reduction of the error introduced when it is increased the number of bits, specially for coefficients with small values, as well as the difference between rounding and truncation.

### Table 2.11
Examples of coefficient quantization for 4 and 8 bits using rounding or truncation.

<table>
<thead>
<tr>
<th>(a_k)</th>
<th>(B + 1)</th>
<th>(a_k^T)</th>
<th>(\delta_T)</th>
<th>(a_k^R)</th>
<th>(\delta_R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>4</td>
<td>0 0000</td>
<td>-100%</td>
<td>0.125 0001</td>
<td>25%</td>
</tr>
<tr>
<td>8</td>
<td>0.09375 0001100</td>
<td>-6%</td>
<td>0.101563 0001101</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>4</td>
<td>0.25 0010</td>
<td>-17%</td>
<td>0.25 0010</td>
<td>-17%</td>
</tr>
<tr>
<td>8</td>
<td>0.297875 00100110</td>
<td>-1%</td>
<td>0.296875 00100110</td>
<td>-1%</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>4</td>
<td>0.875 0111</td>
<td>-3%</td>
<td>0.875 0111</td>
<td>-3%</td>
</tr>
<tr>
<td>8</td>
<td>0.898438 01110011</td>
<td>-0.2%</td>
<td>0.898438 01110011</td>
<td>-0.2%</td>
<td></td>
</tr>
</tbody>
</table>
Figure 2.12
Effects of coefficient quantization using rounding in the gain response of a bandpass FIR filter of 118th-order, for 8 and 16 bits resolution.

Figure 2.13
Effects of coefficient quantization using truncation in the gain response of a bandpass FIR filter of 118th-order, for 8 and 16 bits resolution.
2.6 Product Quantization

When multiplying two numbers in fixed-point representation in two’s complement, where each operand is represented with \((B + 1)\) bits with the \(Q_B\)-format \((B\) bits for fractional part), the result will be a number with a fractional part with \(2B\) bits and the sign bit will be represented twice, having then a word with \((2B + 2)\) bits due to the sign extension of the representation of the binary multiplications. However in most cases this result need to be quantized, i.e. the number of bits must be reduced to the format of the hardware which is, for instance, again the \(Q_B\)-format. So, this bit reduction will introduce error in the result of the multiplication [1].

For example, consider a multiplication of two constants, \(a\) and \(b\):

\[
c = a \times b ,
\]

whose values are represented in fixed-point representation in the processor to perform the multiplication. The quantization can be modelled as a linear model and the multiplication operation can also be reduced to a model, as shown in Figure 2.14 [6]. Each multiplication operand is equal to its quantized value added with the introduced quantization error:

\[
\hat{a} = a + \varepsilon_a , \quad \hat{b} = b + \varepsilon_b ,
\]

and the result of the multiplication will then be:

\[
\hat{c} = \hat{a}\hat{b} + \varepsilon_c = ab + a\varepsilon_b + b\varepsilon_a + \varepsilon_a\varepsilon_b + \varepsilon_c ,
\]

leading to a total quantization error for the multiplication of values \(a\) and \(b\):

\[
\varepsilon_M = a\varepsilon_b + b\varepsilon_a + \varepsilon_a\varepsilon_b + \varepsilon_c ,
\]

where each error is the quantization error either from truncation or rounding. The quantization error due to multiplication, not only depends on the number of bits and the round-off operation used, but it also depends on the operands themselves. If the relevant information of the result is held in the lower part of the digital word (a small number), after quantization these least significant bits will be discarded and the difference between the original result and the quantized one will be large. On the other hand, if most of the information is present on the left side of the digital word (a larger number), the bits discarded after quantization will not introduce a significant error.
Let's consider an example (Example 1) of a multiplication of two numbers quantized to 8 bits ($Q_7$-format) using truncation or rounding as round-off operation shown in Table 2.III. In this example with either truncation or rounding, the multiplication originates a result in which the bits ‘1’ are on the right side of the digital word, and after truncation the only bit ‘1’ remaining is the one with weight $2^{-7}$. The original result should be $c \approx 0.015921$ and the quantized result is $\hat{c} = 0.0078125$, producing then a quantization error of $\varepsilon_M = \hat{c} - c = -0.008125$, corresponding to a very high relative error of $\delta = \varepsilon_M / c \approx -51\%$. With rounding, it will carry one bit and the quantized result is $\hat{c} = 0.015625$, resulting in a lower quantization error of $\varepsilon_M = \hat{c} - c \approx -0.0003$ and a relative error of $\delta \approx -1.9\%$. This example shows how large the error can be if multiplied two small values using truncation as round-off operation, and how can be reduced significantly if using rounding.

In the Example 2, shown in Table 2.IV, the binary representation of $a$ using rounding is different from the one using truncation, thus originating in a different result in the multiplication. For opposition to the previous example, both operands are large values, which produces smaller errors even if using truncation. Using truncation, the quantization error is $\varepsilon_M = -0.00846$ and the relative error is $\delta \approx -1.5\%$, and using rounding the quantization error is $\varepsilon_M \approx -0.00065$ and the relative error is $\delta \approx -0.1\%$.

To further understand how the error from multiplication is affected, in Table 2.V, Table 2.VI and Table 2.VII are presented the relative errors of the quantized multiplication $c = a \times b$, for different resolutions for the operands and for the result of the multiplication. In each table the value of $b$ is fixed at $1/3$ while the value of $a$ is $1/5$, $1/50$ and $0.98$. The first conclusion taken from these tables is that small operands will generate in more error in the result than large operands, specially for low resolutions where the relative error can reach -100%. From these examples can also be seen that rounding introduce less error than truncation in most cases, as seen in previous sections. The error of the multiplication is affected by both quantization of the operands and quantization of the result, according to Equation 2.45. However, from these examples, it is seen that the quantization of the result affects more the error than the quantization of the operands, since there is a more significant reduction of the error by column than by row, whether for small or large values of operands. In Figure 2.15 is illustrated the relative error for truncation and rounding when using 8 bits for $a$, $b$ and $c$, and it shows how large the error can be when $a$ is very small. This happen because with only 8 bits is not possible to represent with accuracy $a$ or $c$, however when $a$ increases the relative error decreases, as expected. In Figure 2.16 is illustrated for the same values of $a$ and $b$ but with 16 bits resolution for $a$, $b$ and $c$. The behaviour of the relative error is identical to Figure 2.15 however with lower magnitude since there is more precision with 16 bits. Once again, it is shown in both figures how truncation will introduce more error than rounding.

### Table 2.III

Example 1 of a multiplication of $a = 0.087$ and $b = 0.183$ with result quantized to 8 bits using truncation or rounding.

<table>
<thead>
<tr>
<th>Truncation</th>
<th>Rounding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{a}$</td>
<td>00001011</td>
</tr>
<tr>
<td>$\hat{b}$</td>
<td>00010111</td>
</tr>
<tr>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>$\hat{a} \times \hat{b}$</td>
<td>0000000011111101</td>
</tr>
<tr>
<td>$\hat{c} = 0.0078125$</td>
<td>00000001</td>
</tr>
<tr>
<td>$\varepsilon_M \approx -0.0081$</td>
<td></td>
</tr>
<tr>
<td>$\delta \approx -51%$</td>
<td></td>
</tr>
<tr>
<td>$\hat{a}$</td>
<td>00001011</td>
</tr>
<tr>
<td>$\hat{b}$</td>
<td>00010111</td>
</tr>
<tr>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>$\hat{a} \times \hat{b}$</td>
<td>0000000011111101</td>
</tr>
<tr>
<td>$\hat{c} = 0.015625$</td>
<td>00000010</td>
</tr>
<tr>
<td>$\varepsilon_M \approx -0.0003$</td>
<td></td>
</tr>
<tr>
<td>$\delta \approx -1.9%$</td>
<td></td>
</tr>
</tbody>
</table>
Table 2.IV
Example 2 of a multiplication of $a = 0.936$ and $b = 0.61$ with result quantized to 8 bits using truncation or rounding.

<table>
<thead>
<tr>
<th></th>
<th>Truncation</th>
<th>Rounding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{a}$</td>
<td>01110111</td>
<td>$\hat{a}$</td>
</tr>
<tr>
<td>$\hat{b}$</td>
<td>01001110</td>
<td>$\hat{b}$</td>
</tr>
<tr>
<td>$\hat{a} \times \hat{b}$</td>
<td>0010010001000010</td>
<td>$\hat{a} \times \hat{b}$</td>
</tr>
<tr>
<td>$\hat{c} = 0.5625$</td>
<td>01001000</td>
<td>$\hat{c} = 0.5703125$</td>
</tr>
<tr>
<td>$\varepsilon_M = -0.00846$</td>
<td></td>
<td>$\varepsilon_M \approx -0.00065$</td>
</tr>
<tr>
<td>$\delta \approx -1.5%$</td>
<td></td>
<td>$\delta \approx -0.1%$</td>
</tr>
</tbody>
</table>

Table 2.V
Relative error for quantized values of $c = a \times b$, for $a = 1/5$ and $b = 1/3$, using either truncation or rounding.

<table>
<thead>
<tr>
<th></th>
<th>(B + 1) bits for $a$ and $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>(B + 1) bits for $c$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>16</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>32</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>

Table 2.VI
Relative error for quantized values of $c = a \times b$, for $a = 1/50$ and $b = 1/3$, using either truncation or rounding.

<table>
<thead>
<tr>
<th></th>
<th>(B + 1) bits for $a$ and $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>(B + 1) bits for $c$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>16</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>32</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>
Table 2.VII
Relative error for quantized values of $c = a \times b$, for $a = 0.98$ and $b = 1/3$, using either truncation or rounding.

<table>
<thead>
<tr>
<th>$(B + 1)$ bits for $a$ and $b$</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(B + 1)$ bits for $c$</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>T</td>
<td>-1.9%</td>
<td>-1.9%</td>
<td>-1.9%</td>
</tr>
<tr>
<td>R</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.4%</td>
</tr>
<tr>
<td>16</td>
<td>-1.9%</td>
<td>-0.01%</td>
<td>-0.002%</td>
</tr>
<tr>
<td>T</td>
<td>0.4%</td>
<td>0.007%</td>
<td>-0.002%</td>
</tr>
<tr>
<td>R</td>
<td>-1.9%</td>
<td>-0.008%</td>
<td>≈0%</td>
</tr>
<tr>
<td>32</td>
<td>0.4%</td>
<td>0.004%</td>
<td>≈0%</td>
</tr>
</tbody>
</table>

Figure 2.15
Relative error of the quantized result of $c = a \times b$, for all values of $a$ and $b = 1/3$, using 8 bits for $a$, $b$ and $c$.

Figure 2.16
Relative error of the quantized result of $c = a \times b$, for all values of $a$ and $b = 1/3$, using 16 bits for $a$, $b$ and $c$. 
2.7 Effects of Finite Precision Arithmetic in a Digital Filter

After analysing the type of errors or noise that will be present in the digital domain along the signal processing path, it is important to see its effects when implementing a digital filter. As an illustrative example, the implementation of non-recursive FIR filters is now considered. Let’s consider a first case, where the filter just has one coefficient:

\[ y(n) = h_0 x(n), \quad n = 0, 1, \ldots \]  

(2.46)

Since the input sample is acquired from an analogue signal, the conversion process at the ADC and all involved hardware will introduce quantization error due to the limit number of bits, assuming that all input samples are scaled within the range of representation to prevent overflow. The coefficient of the filter would be calculated \textit{a priori}, while designing the digital filter, hence need to be represented in fixed-point representation having then associated a quantization error. The filter output will then be:

\[ \hat{y} = \hat{h}_0 \hat{x}_A, \]  

(2.47)

where the time-index \( n \) was dropped to simplify notation. The filter output is affected by the quantization errors of the coefficient and the multiplication (see Figure 2.17):

\[ \hat{y} = \hat{x}_A \times (h_0 + \varepsilon_0) + \varepsilon_{0,A} = \hat{x}_A h_0 + \hat{x}_A \varepsilon_0 + \varepsilon_{0,A}, \]  

(2.48)

and the error due to fixed-point arithmetic operations will be:

\[ \epsilon_y = \hat{x}_A \varepsilon_0 + \varepsilon_{0,A}. \]  

(2.49)

This error does not consider the ADC quantization error that depends on its resolution. This is the error at the digital domain level due to finite word-length. Similarly to the multiplication of two constants, analysed in the previous section, the error of the quantized output sample depends on the quantization error of the coefficient, \( \varepsilon_0 \), the value itself of the input sample and the quantization error introduced by the multiplication operation, \( \varepsilon_{0,A} \), while converting the result again to the hardware resolution, and does not depend on the value of the coefficient. Quantization error at the output sample can be reduced by using rounding, as shown before.

Let’s consider now that the filter to be implemented is of 1st-order. In this case the quantization error will be affected by two multiplications since the result of an addition does not require quantization (if not

![Additive noise model of quantization error on a filtering operation of a non-recursive filter of a 0th-order (adapted from [6]).](image)

**Figure 2.17**
Additive noise model of quantization error on a filtering operation of a non-recursive filter of a 0th-order (adapted from [6]).
Additive noise model of quantization error on a filtering operation of a 1st-order non-recursive filter (adapted from [6]).

Figure 2.18

Additive noise model of quantization error on a filtering operation of a 1st-order non-recursive filter (adapted from [6]).

considering overflow). The output of the filter is:

\[ y(n) = h_0 x(n) + h_1 x(n-1) \quad , \quad n = 0, 1, \ldots , \]  

(2.50)

where \( h_0 \) and \( h_1 \) are the coefficients of the filter, and \( x(n) \) and \( x(n-1) \) samples of the input signal. As before, each predefined coefficient has to be represented in fixed-point hence having an error introduced in the filtering process, \( \varepsilon_0 \) and \( \varepsilon_1 \), respectively. In a simplified notation, as before, \( x(n) \) will be denoted as \( \hat{x}_A \), \( x(n-1) \) as \( \hat{x}_B \), and \( y(n) \) as \( \hat{y} \). The quantized filter output is defined as (see Figure 2.18):

\[ \hat{y} = [\hat{x}_A \times (h_0 + \varepsilon_0) + \varepsilon_{0A}] + [\hat{x}_B \times (h_1 + \varepsilon_1) + \varepsilon_{1B}] \quad , \]

(2.51)

and the error of the quantized filter output it:

\[ \varepsilon_y = \hat{x}_A \varepsilon_0 + \varepsilon_{0A} + \hat{x}_B \varepsilon_1 + \varepsilon_{1B} \quad . \]

(2.52)

Comparing with the previous case, here the quantization error has more additive parcels, because, as expected, it also depends on the input samples which doubled, the quantization error from each coefficient and the error from two product quantizations. In the case of a high order filter, the computational complexity will be higher, meaning that more multiplications and additions will be computed. Although generic, these conclusions may be exceptions in certain particular cases. For instance a lowpass filter of lower order may introduce more error than a higher order one, as in:

\[ y(n) = \frac{1}{M} \sum_{k=0}^{M-1} x(n-k) \quad , \quad n = 0, 1, \ldots , \]

(2.53)

for \( M = 16 \) and \( M = 13 \), because \( h_k = 2^{-4} \) does not introduce quantization error and this filter does not need multiplications. With more computations more errors will affect the output of the filter, and it is necessary to remember the always present coefficient quantization changes the frequency response of the filter even before any computation takes place.

Another thing to take into account when implementing a filter is its structure because it will also affect the quantization errors introduced during computation. All these previous simple examples are digital filters with transversal structure, but a digital filter can also be implemented in a direct form transpose or cascaded structure (as seen in previous sections). The direct form transposed structure has the same computational complexity as the direct form, however is more sensible to quantization noise than the direct form [3]. The cascaded structure is referred to be less sensible to quantization noise than
the other two structures thus desirable for noise reduction [1]. This assumption is applied since most coefficients of filter are usually very small values and when converted to second-order sections its new coefficients, $b_{1k}$ and $b_{2k}$, are of higher magnitude, introducing then less noise into the filtering process.

To understand how all the quantization errors affect the filtering process, let's consider the 118th-order bandpass filter of the previous section has a sinusoidal signal with frequency equal to 2 kHz and amplitude equal to 0.8 as the input signal, $x(n)$. The input signal is quantized for 8 bits, $\hat{x}_8(n)$, and for 16 bits, $\hat{x}_{16}(n)$, while using rounding, and in the filtering process it is performed quantizations using rounding after each product and then accumulated. The frequency of the input signal corresponds to the first frequency of the passband of the filter, thus ideally the output signal should have the same amplitude as the input signal. Analysing first the Figure 2.19, the quantized samples of the input signal practically overlap the ideal input signal, which does not introduce significant errors. If seeing the Figure 2.12, at the frequency 2 kHz the quantized coefficients with 8 bits produces a magnitude lower than 0 dB, which is the expected gain in the passband. Therefore, the output signal quantized for 8 bits should have smaller amplitude than the input signal, as illustrated in Figure 2.20. In Figure 2.21 are shown the power spectral density for the ideal output signal and quantized for 8 and 16 bits. Here can be seen how these errors affect the output signal, which for 8 bits, the harmonics already have higher amplitude than in the
Figure 2.21
Power spectral density of the output signal of the digital filter: (a) ideal ($THD = -63.78$ dB); (b) quantized for 8 bits ($THD = -31.02$ dB); (c) quantized for 16 bits ($THD = -63.71$ dB).
ideal case resulting in a distorted output signal, as the value of the total harmonic distortion decreased significantly. However, using 16 bits resolution there is practically no problems in accuracy, since the input and output signals matches the ideal ones and the total harmonic distortion is very close to the ideal one.

2.8 Conclusion

The representation of numbers in the digital domain will always imply an introduction of errors along the signal processing. Despite the inevitable quantization errors from the conversions between analogue-to-digital domain and vice-versa, while running an algorithm or digital system implemented is necessary to quantize coefficients and during multiple arithmetic operations can also be necessary to quantize the result, adding then more error to the signals, which can be significant specially in multiplications. All these errors combined originate in a different filter gain response and can also lead to instability of the filter.

These errors can be generated from using truncation or rounding to quantize the values, where the latter imply a smaller error. All these quantizations are inevitable, however it is important to understand its consequences into the filtering process and how it can be reduced. From the examples shown in this chapter it can be noted that the best choice is to use rounding in all the quantizations, either from converters or arithmetic operations, translating to a smaller error. Should also be taken into consideration that the error depends on the magnitude of the operands in the case of multiplications, and the error can be more significant for small values than when using large values. When implementing a digital filter the quantizations errors are affected by the multiplications performed while filtering the input signal, excluding the coefficients quantization. These errors will accumulate along the processing, originating in a filter and an output signal far for the desired specifications.
Chapter 3

Adaptive Filters

In signal processing applications the digital filters most used are designed accordingly to the desired output. If the input signal is stationary over time, i.e. the signal characteristics do not change over time, the digital filter is designed according to those characteristics, and the parameters of the filter will remain unchanged.

However if the characteristics of the input signal change the digital filter cease to become effective because its parameters no longer correspond to the attributes of the input signal. In these cases it is necessary to use a digital filter that readjusts its parameters at each time instant based on the input signal. These filters are called adaptive filters, in which the digital filter produces an outcome relative to an input signal and an adaptive algorithm adjust the coefficients of the filter [7]. In Figure 3.1 it is shown an example of an $N^{th}$-order adaptive filter whose coefficients, $w_k(n)$, change over time, and the filter output is defined as:

$$y(n) = \sum_{k=0}^{N} w_k(n)x(n-k), \quad n = 0, 1, ...$$

(3.1)

The adaptive algorithm is designed to minimize a cost function established from an error signal, which is defined as the difference between a desired response and the filter output.

It is necessary to establish two important aspects concerning the adaptive filter when using an algorithm to find the suitable filter coefficients: convergence and stability. As in any other algorithm it is desirable to have the minimum convergence time possible without compromise the results. If the input signal characteristics change between time instants, it is essential to reach convergence before the signal characteristics change again. Stability obviously plays an important role in the path to find the suited coefficients. For this reason, in adaptive filtering the most commonly used filters are the FIR type as they are always stable regardless the filter order [4].

![Figure 3.1](image)

Adaptive filter with recursive structure in the direct form.
3.1 The Wiener Filter

A useful approach to the filter-optimization problem is to minimize the mean-square value of the error signal \( [2] \). If the input signal is stationary, a solution to this problem is the Wiener filter, which is optimal in terms of the mean square error. In Figure 3.2 is represented the block diagram of a system with a Wiener filter, whose impulse response coefficients allow the expected value of the squared error to be minimized. The output of the filter is defined by:

\[
y(n) = w_0 x(n) + w_1 x(n-1) + \ldots + w_N x(n-N) = W^T X(n) , \quad n = 0,1,\ldots
\]  

(3.2)

where \( W \) is the vector of the coefficients of the filter with \((N + 1)\) coefficients and \( X(n) \) the vector of input values. The function to be minimize is referred as a cost function, \( J(w) \), which in this case is equal to the expected value of the squared error \([2]\):

\[
J(w) = E\{e^2(n)\} ,
\]

(3.5)

where \( E\{\} \) is the expected value operator. Considering the error is equal to the difference between the desired signal and the filter output, the previous expression can be simplified to:

\[
J(w) = E\{[d(n) - y(n)]^2\}
\]

\[
= E\{d^2(n) - 2d(n)y(n) + y^2(n)\}
\]

\[
= E\{d^2(n)\} - 2E\{d(n)W^T X(n)\} + E\{W^T X(n)X^T(n)W\} ,
\]

(3.6)

and since the filter weight vector is not a random variable, the cost function becomes:

\[
J(w) = E\{d^2(n)\} - 2W^T E\{d(n)X(n)\} + W^T E\{X(n)X^T(n)\} W .
\]

(3.7)

Defining the cross correlation vector between the desired signal \( d(n) \) and the input signal \( x(n) \) as \([2]\):

\[
p \triangleq E\{d(n)X(n)\} = \begin{bmatrix} p(0) & p(1) & \ldots & p(N) \end{bmatrix}^T ,
\]

(3.8)

the autocorrelation matrix of the input signal as \([2]\):

\[
R = E\{X(n)X^T(n)\} ,
\]

(3.9)
and assuming that $d(n)$ has zero mean and $\sigma_d^2$ is the variance of $d(n)$, it is possible to rewrite the cost function $J(w)$ in terms of the autocorrelation and cross correlation matrices [2]:

$$J(w) = \sigma_d^2 - 2W^Tp + W^TRW.$$  \hspace{1cm} (3.10)

The previous equation represents a $(N + 2)^{th}$ dimensional convex surface with a unique minimum point (minimum expected value of the squared error) (as example see Figure 3.3) which can be calculated with the operation gradient with respect to the filter weight vector. Thus, the gradient of the cost function becomes [2]:

$$\nabla J(w) = -2p + 2RW.$$  \hspace{1cm} (3.11)

Setting this gradient to zero is then possible to calculate the minimum of the cost function which corresponds to the optimum filter coefficients, $W_{opt}$:

$$\nabla J(w) = 0 \iff W_{opt} = R^{-1}p.$$  \hspace{1cm} (3.12)

In conclusion, with the knowledge of a set of input signal $x(n)$ and desired signal $d(n)$ samples, it is possible to determine the optimum filter coefficients, or Wiener filter coefficients, which will take the error signal to a minimum. This is the optimal filter solution that the adaptive filtering algorithms attempt to achieve [2]. Despite there is an explicit equation to calculate the coefficients of the Wiener filter, this equation is not implemented in practice for two reasons: the computational complexity is high due to the calculation of the inverse of matrix $R$ and therefore not suitable for real time applications; and in reality we come across signals for which the values of $R$ and $p$ are not available and must be estimated. Thereby the objective of an adaptive algorithm is to surpass these disadvantages. In the following subsection the gradient descent algorithm and the least mean squares algorithm are then considered.

### 3.2 The Gradient Descent Algorithm

The gradient descent method is a way to find a local minimum of a cost function [2]. Initially it starts at any solution of the function and calculates the gradient of the function at that point. Then with the negative direction of the gradient finds the new solution of the function. This process is repeated over and over and the algorithm will eventually converge where the gradient is zero (which correspond to a
Figure 3.4
Three dimensional simplified representation of how the gradient descent algorithm finds the minimum of a convex cost function for $N = 1$.

local minimum), as exemplified in Figure 3.4.

In this case the objective is to find the minimum of the cost function $J(w)$. Given some initial value $W(0)$ for the filter weights, $W$, one can change its value in many directions. To find what is the best direction to minimize $J(w)$, the gradient $\nabla J(w)$ is calculated at the initial point. Intuitively, the gradient will give the slope of the curve at that $W$ and its direction will point to an increase in the function, but since the goal is to find the minimum, the algorithm uses the opposite direction of the gradient. The gradient descent algorithm calculates the weight for the next iteration, $W(n+1)$, of the digital filter based on the gradient of the cost function, $J(w)$ (Equation 3.13), and converges to the minimum of the cost function which corresponds to the optimum weight filter vector [2].

$$W(n+1) = W(n) - \mu \frac{\partial}{\partial w} J(w) = W(n) + \mu [p - RW(n)] \quad , \quad n = 0, 1, ... \quad (3.13)$$

The $\mu$ parameter in Equation 3.13 represents the step size at which the algorithm jumps from point to point in the cost function. Is easy to conclude the larger the step size the fastest is the convergence of the algorithm, however a large value can also lead to instability [8]. In Equation 3.13 the gradient of the cost function is substituted by Equation 3.11. When compared with the Wiener solution, this algorithm does not need to calculate the inverse of matrix $R$ to find the optimum weight filter vector, $W_{opt}$, saving computation time and decreasing computational complexity.

3.2.1 Convergence

The convergence of the algorithm is an important aspect to consider when choosing an algorithm to implement in the adaptive signal processing system. To find the convergence of the gradient descent algorithm let first consider the definition of a weight error vector as [2]:

$$E(n) = W(n) - W_{opt} \quad , \quad n = 0, 1, ... \quad (3.14)$$

The gradient descent algorithm weight update equation may be written as:

$$W(n+1) - W_{opt} = W(n) - W_{opt} + \mu [p - RW(n)] \quad ,$$
$$E(n+1) = E(n) + \mu [p - RW(n)] \quad . \quad (3.15)$$
Using the equation of the optimum weight vector, or Wiener weight vector $W_{\text{opt}} = R^{-1}p$ [2], it is possible to calculate the weight error vector based on the autocorrelation matrix $R$:

$$E(n+1) = E(n) + \mu [RW_{\text{opt}} - RW(n)] = [I - \mu R] E(n) . \quad (3.16)$$

Using the propriety that the autocorrelation matrix $R$ can be represented by the unitary matrix with its eigenvectors $Q (Q^T Q = I)$ and the diagonal matrix $D$ of its eigenvalues ($\lambda_k, k = 0, 1, ..., N$), decomposed as [2]:

$$R = QDQ^T , \quad (3.17)$$

and defining a rotated weight misalignment vector $\tilde{E}(n) = Q^T E(n)$ [2]:

$$Q^T E(n+1) = Q^T E(n) - \mu Q^T DQ^T E(n) = Q^T E(n) - \mu DQ^T E(n) , \quad (3.18)$$

$$\tilde{E}(n+1) = [I - \mu D] \tilde{E}(n) , \quad (3.19)$$

$$\tilde{\varepsilon}_k(n+1) = [1 - \mu \lambda_k] \tilde{\varepsilon}_k(n) , \quad k = 0, 1, ..., N \quad ,$$

where $\tilde{E}_k$ are the components of the $\tilde{E}(n+1)$ vector. The gradient descent algorithm tries to achieve the optimum weight vector, i.e. the point when the weight error vector components are zero,

$$\lim_{n \to \infty} \tilde{\varepsilon}_k(n) = 0 , \quad k = 0, 1, ..., N \quad , \quad (3.20)$$

which requires that, for each $\lambda_k$, $|1 - \mu \lambda_k| < 1$. And since $\lambda_k \geq 0$ and $\mu > 0$, thereby the condition for convergence of the algorithm is that, for all $k$ [2]:

$$0 < \mu < \frac{2}{\lambda_k} . \quad (3.21)$$

Since all the eigenvalues of the autocorrelation matrix $R$ are equal or smaller than the largest eigenvalue, $\lambda_{\text{max}}$, the necessary condition for convergence of the gradient descent algorithm is [2]:

$$0 < \mu < \frac{2}{\lambda_{\text{max}}} . \quad (3.22)$$

With this condition it is possible to infer the bigger the eigenvalue spread, $\xi(R) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$, the slower will be the convergence.

### 3.3 The Least Mean Squares Algorithm

The gradient descent algorithm surpass the problem of the calculation of the inverse matrix. However, in real time applications the values of the autocorrelation matrix $R$ and the cross correlation vector $p$ are not available and must be estimated [2]. This is the foundation of the least mean squares (LMS) algorithm, where it is considered an estimation based on the expected values [2]:

$$R = E \{X(n)X^T(n)\} \approx X(n)X^T(n) \quad , \quad (3.23)$$

$$p = E \{d(n)X(n)\} \approx d(n)X(n) \quad .$$
With these estimations the equation to calculate the next weights of the filter is:

\[
W(n + 1) = W(n) + \mu[p - RW(n)] = W(n) + \mu[d(n)X(n) - X(n)X^T(n)W(n)]
= W(n) + \mu[X(n)d(n) - X^T(n)W(n)] = W(n) + \mu[X(n)d(n) - y(n)]
= W(n) + \mu X(n)e(n), \quad n = 0, 1, \ldots .
\] (3.24)

In conclusion, the equations of the LMS algorithm resume to (see Figure 3.5) [2]:

\[
y(n) = W^T(n)X(n), \quad e(n) = d(n) - y(n), \quad W(n + 1) = W(n) + \mu X(n)e(n).
\] (3.25)

In the perspective of computational complexity, a generic adaptive filter of \(N\)-th-order using the LMS algorithm (see Equation 3.25) need space memory for \((N+1)\) coefficients, which are constantly updated, for \((N + 1)\) input samples, for the filter output sample \(y(n)\) and for the error sample \(e(n)\), and compute \((2N + 3)\) multiplications and \((2N + 2)\) additions of two values.

3.3.1 Convergence

The convergence of the LMS algorithm is based on the following assumptions: (1) the input reference signal vectors \(X(1), \ldots, X(n)\) are statistically independent of each other; (2) the filter weight vector \(W(n)\) is independent of the input vector \(X(n)\); and (3) the input vector \(X(n)\) is independent of the previous samples of the desired signal \([d(n - 1), d(n - 2), \ldots]\).

As in the gradient descent algorithm, the weight update equation of the LMS algorithm can be represented on a weight error vector as [2]:

\[
E(n + 1) = E(n) + \mu X(n)[d(n) - X^T(n)W(n)] ,
\] (3.26)

which, if the expected value operator, \(\mathbb{E}\{\}\), is applied to both sides of the equation, it becomes:

\[
\mathbb{E}\{E(n + 1)\} = \mathbb{E}\{E(n)\} + \mu \mathbb{E}\{X(n)d(n)\} - \mu \mathbb{E}\{X(n)X^T(n)W(n)\} .
\] (3.27)

According with assumption (2) the last expected value can be simplified to [2]:

\[
\mathbb{E}\{X(n)X^T(n)W(n)\} = \mathbb{E}\{X(n)X^T(n)\} \mathbb{E}\{W(n)\} .
\] (3.28)

Figure 3.5
Block diagram of a system with an adaptive filter \(W(n)\), adjusted by the error signal \(e(n)\) (adapted from [2]).
Similarly to the gradient descent algorithm, it is possible to obtain [2]:

\[
E\{E(n+1)\} = E\{E(n)\} + \mu [p - RE\{W(n)\}] = E\{E(n)\} + \mu [RW_{opt} - RE\{W(n)\}] = [I - \mu R] E\{E(n)\} .
\]

The result from the previous equation is equal to the gradient descent method except it has an expectation operator on the weight error vector. Thus the condition for convergence is equal to:

\[
0 < \mu < \frac{2}{\lambda_{max}} ,
\]

where \(\lambda_{max}\) is the maximum eigenvalue of the autocorrelation matrix \(R\). In practice the eigenvalues of the autocorrelation matrix are not available, and \(\lambda_{max}\) cannot be greater than the trace of the matrix \(D\), which is the sum of all the elements of the main diagonal of \(D\). Since the autocorrelation matrix \(R\) is equal to the matrix \(D\), result then the following condition [2]:

\[
\lambda_{max} \leq tr\{D\} = tr\{R\} = \sum_{k=0}^{N} \lambda_k ,
\]

where \(tr\{\}\) is the matrix trace operator. Furthermore, in the LMS algorithm it is considered that \(R \approx X(n)X^T(n)\), then the trace of \(R\) is given by [2]:

\[
tr\{R\} = \sum_{k=0}^{N} x^2(n-k) \triangleq (N + 1) \|X(n)\|^2 ,
\]

in which \(\|X(n)\|^2\) denotes the power of \(x(n)\). Therefore, the convergence of the LMS algorithm depends on the power of the input reference signal and the order of the adaptive filter [2]:

\[
0 < \mu < \frac{2}{(N + 1)\|X(n)\|^2} .
\]

Since the power of the input signal is always greater or equal to the maximum eigenvalue of the autocorrelation matrix, the LMS algorithm condition of convergence is stricter than the condition of convergence of the gradient descent algorithm.

### 3.4 Applications of Adaptive Processing

Adaptive signal processing applications may be divided into four main classes [8]: (1) system identification (Figure 3.6(a)); (2) inverse system modelling (Figure 3.6(b)); (3) prediction of signals (Figure 3.6(c)); and (4) interference cancellation (Figure 3.6(d)).

If there is an unknown system (plant) with an input and output signal, with an adaptive filter it is possible to find the unknown system transfer function. With the same input signal \(x(n)\) as the plant, the adaptive filter tries to be equal to the unknown system through the error signal which readjusts the coefficients of the adaptive filter. When the error signal \(e(n)\) starts to be zero, i.e. the output of the adaptive filter is equal to the output of the plant, the coefficients of the adaptive filter are already adjusted and the algorithm converged to a transfer function that corresponds to the unknown system.

The second application class, inverse system identification, is similar to the first one. However instead
of the adaptive filter to be equal to the plant it will have the inverse transfer function of the plant. The input of the adaptive filter is the output of the plant and, since any system with transfer function \( H(z) \) in \( z \) domain is possible to have \( H(z)H(z)^{-1} = 1 \), the error signal \( e(n) \) will tend to be zero when \( y(n) \) is equal to the input signal \( x(n) \).

With an adaptive filter it can be predicted samples of a signal at a future time. If the prediction is perfect, the adaptive filter represents a model for the input signal, where the input signal is the one to be estimated. The error signal \( e(n) \) will tend to zero as in the output of the adaptive filter \( y(n) \) will be equal to the future samples of \( x(n) \).

Whenever there is noise corrupting a desired signal, with an adaptive filter the noise can be cancelled or reduce it. The input signal of the adaptive filter is a noise signal \( x(n) \) correlated with the noise incorporated in the signal \( d(n) \), and ideally the output signal of the adaptive filter \( y(n) \) tends to be equal to the noise in \( d(n) \), in order to the error signal \( e(n) \) be simply the desired signal free of noise.

### 3.5 Example of an Adaptive Filter

There are two important aspects to evaluate the performance of an adaptive system, such as time of convergence and mean squared error (MSE). Time of convergence gives us information on how long it takes the algorithm to converge, which according to the application of adaptive filters is when the output signal of the system is zero or a signal without noise. The MSE is according to a residual error signal, which again differs from application to application. The ideal is to have a very small convergence time and the lowest MSE possible. However, it is very difficult to combine both these characteristics in the same adaptive system, therefore is necessary to compromise either the convergence time or MSE in some implementations in order to have the best performance according to the expected specifications.

In this section is presented an example of an adaptive system identification, as illustrated in Figure 3.6(a). Consider that the unknown system is a FIR filter, whose output signal is \( d(n) \) and is characterized by:

\[
d(n) = 0.1x(n) + 0.2x(n - 1) + 0.3x(n - 2) + 0.4x(n - 3) \quad , \quad n = 0, 1, ...
\]  

(3.34)
The input signal, \( x(n) \), is white noise with Gaussian distribution and variance equal to one, generated in the computational environment MATLAB. The adaptive filter is a FIR filter of 3rd-order, whose coefficients were initialized at zero. From Equation 3.33 it is shown that the step size depends on the adaptive filter order and power of the input signal. If the variance of the input signal is one, then the upper bound of the step size becomes:

\[
\mu < \frac{2}{N + 1} = 0.5,
\]

so in that way the step size chosen is \( \mu = 0.01 \), validating the convergence condition of the LMS algorithm. This experiment was repeated 100 times in order to have an ensemble of data to calculate the MSE.

In this type of application, the adaptive filter should have a transfer function equal to the unknown plant, or this case, the 3rd-order FIR filter with difference equation in Equation 3.34. Therefore, the error signal \( e(n) \) of the adaptive system should be zero when both systems are equal, as in illustrated in Figure 3.7 after approximately 600 iterations.

The time of convergence can also be analysed in Figure 3.8, where it is shown that each coefficient of the adaptive filter tend to the value of the coefficient of the considered unknown plant. From Figure 3.8, the time of convergence is also approximately 600 iterations.

In Figure 3.9 it is illustrated the instantaneous squared error \( e^2(n) \) of the adaptive system and the MSE of the ensemble, defined as:

\[
MSE(n) = \frac{1}{T} \sum_{i=1}^{T} e^2_i(n),
\]

where \( T \) is the number of tests, equal to 100, and \( e_i(n) \) the sample of the error signal in test \( i \). The instantaneous squared error varies more along the iterations, while the MSE represents the average of the squared error, illustrating an error with less noise, according to the data of the ensemble. Both these signals after the convergence time tend for an amplitude of \( 10^{-5} \), representing at the output of the adaptive system an error with amplitude around 1/1000. With these results it is possible to conclude that the adaptive filter is a very good approximation of the unknown plant.

As referred before, the step size predict the time of convergence of the algorithm. The value of the step size should always be according to Equation 3.33, however different values will mean a faster or
Figure 3.8
Evolution of the coefficients of the adaptive filter.

Figure 3.9
Mean squared error and instantaneous squared error of the adaptive system identification.

Figure 3.10
Mean squared error for different values of the step size for the adaptive system identification example.
slower convergence, as illustrated in Figure 3.10. It is presented the MSE for three different values of the step size: 0.05, 0.03 and 0.01. For the case of \( \mu = 0.01 \), as seen before, the time of convergence is approximately 600 iterations. If the step value is increased to 0.03 the time of convergence decreases to approximately 225 iterations, and if \( \mu = 0.05 \) the algorithm converges even faster, after only 150 iterations, approximately.

3.6 Conclusion

In a digital system with the purpose of filtering a certain input signal whose characteristics are static, it is ideal to have a well designed and optimized filter in order to have the expected output signal. This optimum filter is referred as Wiener filter and it is optimum in the sense of producing the minimum mean square value of the error signal. Nevertheless, if using the Wiener filter, when the characteristics of the input signal start changing the filter no longer corresponds to an optimum filter. Since there is available an equation to determine the optimum coefficients, they could be recalculated at each iteration. However this brings a disadvantage, because to recalculate the optimum coefficients is necessary to have the auto-correlation and cross-correlation matrices at each iteration, where in most cases these are unknown and unpredictable, thus preventing the update the coefficients. Besides, the equation to update the coefficients to be the optimum coefficients has high computational complexity becoming inadequate for real time processing.

One solution to surpass the high computational complexity is to use the gradient descent algorithm. Instead of calculating the optimum coefficients at each iteration, the algorithm will update the coefficients of the filter at each iteration according to the auto-correlation and cross-correlation matrices so it would converge to the optimum Wiener filter. The computational complexity is reduced since is no longer needed to calculate the inverse matrix of the autocorrelation as in the equation of the optimum Wiener filter.

Despite decreasing the computational complexity, the gradient descent algorithm is still based on the auto-correlation and cross-correlation matrices, thus it was created the LMS algorithm, which gives an estimation for those matrices based only on the samples of the input signals. In each iteration the algorithm calculates the filter output and readjust the coefficients in order to converge to the optimum Wiener filter.

Adaptive filtering has multiple applications which can be generically organized into four different classes: system identification; inverse system identification; prediction of signals and noise cancellation.
Chapter 4

Integrated Circuits for Implementation of Digital Systems

Nowadays there are available numerous options of integrated circuits (IC) to implement a digital circuit or system, from general purpose processors (GPP) to special purpose processors (SPP) [9]. To have an efficient implementation of an adaptive system, which is a digital circuit, it has to be chosen the right IC for implementation. The principal point is the architecture of the IC, which should guarantee the good performance of the adaptive algorithm, mathematically and computationally.

In the next sections are presented an overview of possible ICs able to run an efficient adaptive system, considering the performance of the algorithm, computation power and cost, such as: microprocessors, as in micro-controller (MCU) and digital signal processor (DSP), and field programmable gate array (FPGA).

4.1 Microprocessors

A microprocessor is present in almost every electronic device used everyday, such as mobile phones, audio players, televisions, washing machines, hearing aids, cars, personal computers, and so on. This processor is an IC with its own central processing unit (CPU) operating at a certain frequency, internal memory(ies) and input/output channels for data transfer [9], and are used for a very specific goal, whose architecture differ from the GPP, being then referred as SPP.

Usually a microprocessor is used for data manipulation, as word processing and database management, or for mathematical calculations. A microprocessor is designed to perform only basic arithmetic calculations as addition, subtraction, multiplication and division, hence any other mathematical operation, such as logarithm, cosine, etc., must be performed as a series of those basic arithmetic operations [9]. The data to be processed can either come from the memory embedded in the microprocessor or through a receiver acquiring an analogue signal which then has to be converted to the digital domain to be processed, having then off-line and real time processing, respectively.

These digital processors are programmed according to the type of data processing. Nowadays the digital domain is the most used to process signals because it brings more long-term advantages when compared with analogue signal processing. For example, after designed an analogue filter, it is more
difficult to calibrate and modify the filter parameters than in the digital domain with a microprocessor, which is software based.

To implement an algorithm or digital circuit the microprocessor is programmed through a software tool using an programming language, which is usually C or assembly. Assembly is usually referred as a low-level programming language, executing then faster than a program written in C language, which is a high-level programming language [9]. However, C language is more largely used because is easier to develop and maintain than assembly. Every software tool has a compiler which translates the high-level programming language into machine code for the microprocessor to understand, mapping according to the microprocessor architecture. In most compilers, are implemented algorithms to optimize the code in terms of computation speed or code size for a specific microprocessor target.

The first models of microprocessors had Von Neumann architecture which include a single memory and a single bus for data transfer between the CPU and memory [9]. This implies that the CPU could either read an program instruction or read/write data from memory separately, being always required one clock cycle for each reading/writing, where one clock cycle corresponds to the inverse of the operating frequency of the CPU.

Later, the Harvard architecture was introduced to microprocessors, which corresponds to the basic architecture of the microprocessors available nowadays. The main difference between the Von Neumann and the Harvard architecture is that the latter has different memory systems for both instructions and data. The Harvard architecture uses a separate memory for data and another to program instructions, each one with independent buses for data transfers and with different addresses spaces [9]. This allow the CPU to fetch data and a program instruction at the same time in just one clock cycle. In microprocessors there can also be available peripheral units such as analogue-to-digital or digital-to-analogue converters, timers, modulators, input/output (I/O) ports, etc.

All the different types of microprocessors are designed with variations in their architecture that then influence the computation speed (throughput), the size of data that can be processed and stored, size of the chip, power consumption and performance [10]. In the following sections are presented different SPP which can be suited to perform adaptive filtering in real time: digital signal processor and microcontroller. The following analysis will fall on the architecture of each device, related to performance and computation speed.

4.2 Digital Signal Processor

A DSP is a specialized microprocessor with an architecture optimized to do calculations of mathematical functions for real time signal processing in order to reduce computation time [9], as for example filtering. All DSPs are based on Harvard architecture, but over the years they have been modified to have better performance and to be faster while processing digital signals. With these changes most of the architectures of current DSPs are referred as Modified Harvard architectures or non-strict Harvard architectures [9].

The objective of all these modifications is to increase computation speed, as decreasing the time to fetch information from both data and program memories and performing the maximum number of operations concurrently. One of the optimizations was to include an instruction cache in the CPU [9], which is a very small memory, where is saved the last instructions performed, and have a smaller access time than the program memory. Considering that filtering is largely used in digital signal processing, there is performed
a loop to calculate the output sample of the filter with the input samples and coefficients of the filter. After the first iteration, in the instruction cache is already stored the program instructions corresponding to the loop, eliminating then the need to access the program memory to perform the rest of the iterations of the loop, saving computation time, especially if the length of the filter is large. It was also included an I/O controller where is possible to store the input data directly in the data memory without going through the CPU, referred as direct memory access (DMA) [9]. There are also available both serial and parallel high speed communications ports, because in real time processing the objective is to move the data in, perform calculations and then put the result out before the next sample is available. Besides these additions to the hardware, the recent versions of DSPs are designed to perform instructions in pipeline, reducing the computation time.

A simplified diagram of the architecture of a DSP is presented in Figure 4.1, including the CPU, memories and peripheral units, and considering the bold arrows as internal buses for data transfers. In this simplified description inside the CPU is usually a datapath, for data processing, and a control unit which controls the processing unit [10]. In the control unit are usually data address generators (DAG) to control the addresses for both data and program memories, defining the location of the information to be read or written, and the instruction decoder with an instruction cache inside, responsible to decode the program instruction from program memory and also communicates with both DAGs and data registers ([9], [10]). The data registers belong to the datapath and can be used to keep intermediate results of calculations, prepare data for calculations, play as buffer for data transfers, keep the program counter, and so on on [10]. The calculations are performed inside the datapath in either the multiplier, the arithmetic and logic unit (ALU) or the barrel shifter [10]. The multiplier multiplies two numbers fetched from registers and store the result in the same or other register. In ALU can be executed addition, subtraction, modulo, AND, OR, XOR and NOT functions. In most cases, both multiplier and ALU can be accessed at the same time.

For last, in the barrel shifter is executed shifts, rotations and extract and deposit segments. Besides the CPU, as referred before, there are also memories for data and program instructions and the I/O controller (DMA).

There are available DSPs with two different number representations: fixed-point and floating-point. As implicit in the name, in the first representation the numbers have fixed binary point while in the second the position of the binary point can vary. Nowadays, there are DSPs with 16 or 32 bits for fixed-point and 32 bits for floating-point. The differences between these two representations rely on cost and precision. The fixed-point representation is cheaper than the floating-point, however this one has more precision, dynamic range and development time [9]. The choice between either representations should focus on the complexity of the algorithm to be implemented, for example, if the goal is to have a FIR filter it can be used fixed-point representation, but if there is a need to perform frequency domain operations, such as fast Fourier transform (FFT), is beneficial to use floating-point representation.

When using fixed-point representation, it should be taken into consideration the overflows or underflows along the mathematical operations. Since one of the most used functions in signal processing is filtering, it was later included in the DSP an instruction called multiply and accumulate (MAC) with an accumulator with double precision [9]. With this addition to the architecture is possible to reduce the quantization error from fixed-point multiplications, since it is only quantized the result of the accumulation with double precision of all the products of the filtering loop instead of the accumulation of each quantized product. In more recent models of DSPs the accumulator has extended precision, as for example, the DSP model of Texas Instruments TMS320C5532 has an accumulator in the multiplier with 40 bits of resolution [11], where 32 bits are for the product plus extra 8 bits to prevent overflows or underflows while accumulat- ing. With these improvements in a recent DSP it is possible to perform a set of operations, such as
A MCU is a microprocessor designed and optimized for control, timing or supervising tasks of other devices [12]. Opposing to the DSP, a MCU is characterized by being less complex with a reduced cost and small size due to high production levels [10]. Initially the MCU were known for having various external memories and a large amount of peripherals, where the DSP had more specialized hardware for mathematical functions and less peripheral units. They both share the same basic Harvard architecture, however the common MCU is not specialized in signal processing thus does not share the same improvements as the DSP related to the Modified Harvard architecture [10], but more recently there is already some models of MCU that have Modified Harvard architecture. In Figure 4.2 is presented a simplified block diagram of a MCU with Harvard architecture, including its CPU, memories and the common peripherals units, considering the bold arrows as internal buses for data transfers. Comparing with the common DSP architecture, the ALU of the MCU is simpler and does not have both DGA for data and program memories. However, there is usually some external memories as well as oscillators present in a MCU and not in DSPs.

A MCU only have hardware for integer representation or fixed-point representation (in the recent versions) with usually 8, 16 or 32 bits of resolution [10]. And with a less complex CPU architecture, a MCU has a lower clock frequency than a DSP. With all these differences the MCU is also characterized for having less computational power than a DSP. With the evolution of electronics, these two types of microprocessors have became more and more close to each other in cost and performance especially for particulars applications where the computational need is not very complex, such as a simple algorithm with a few lines of code without complex mathematical operations.
4.4 Field Programmable Gate Array

A FPGA is composed by an array of programmable logic blocks which are connected to each other by a programmable interconnect network, which can be electrically programmable to every kind of digital circuit [13]. The designation of programmable or reconfigurable express the ability to implement any new function on the IC even after its fabrication, opposing to microprocessors. The other main attraction of a FPGA is the flexibility, where for example, parts of the FPGA can be partially reconfigured while simultaneously the other parts can be running whatever is already programmed in them, and any new update can be easily implemented by upgrading the bitstream code (machine code). Nevertheless, this hardware flexibility also cause the FPGA to be bigger and more power consumer than others ICs, since most part of the architecture of a FPGA is composed by the programmable routing interconnect.

The basic structure of a FPGA (see Figure 4.3) include programmable logic blocks to implement logic functions, programmable routing to connect those logic functions and I/O blocks connected to logic blocks through routing interconnect allowing exterior communications. In a simplified and generic description, the configurable logic blocks (CLB) are placed in a two dimensional grid and are connected between them using the programmable routing resources. All the I/O blocks are located around the grid of the CLBs, being also connected to the programmable routing network [13].

The most basic component of a FPGA is the CLB, contributing to the basic logic and storage functionalities for the implemented digital circuit, which can be as simple as an transistor or complex as a processor. The most simple component requires more programmable interconnections occupying more area of the FPGA, causing low performance and higher power consumption. However, in the other extreme, the basic logic block of a processor cannot support small functions on it, contributing in a waste of resources. Between these two extremes, it is possible to find basic logic blocks as NAND gates, multiplexers, look-up tables (LUT) and programmable array logic (PAL) style wide input gates [13]. Most known manufactures of FPGA (as Xilinx [14] and Altera [15]) produce LUT-based CLBs to supply a good...
basic logic and storage functionality. Each CLB can comprise one basic logic element (BLE) or a cluster of locally interconnected BLEs [13]. On other hand, each BLE contain a k-input LUT, a Flip-Flop and a multiplexer to select the BLE output coming from either the LUT or the Flip-Flop. In a CLB with a cluster of BLEs, each one can access each other output through the local routing network, and each BLE has its own output pin. In the recent models of FPGAs, besides the basic logic blocks, there is already included some specific purpose blocks, as memories, multipliers, adders, DSP blocks, etc. [13]. These specific purpose blocks, opposing to the basic logic blocks, are efficiently designed to implement specific functions, as the microprocessors referred in previous sections. Despite this, these purpose blocks consume statically a large amount of logic and routing network, and if not used these resources cannot be reconfigurable and used for something else, wasting then an unused part of the FPGA.

To connect all the blocks of the FPGA, the routing network uses wires and programmable switches [13]. This network should be designed optimally to the characteristics of the digital circuit to be implemented, being at the same time flexible and efficient. For example, using short connections can lead to a huge amount of wires, consuming more area, or distant connections can lead to a sparse long wire, causing delays in the connections. Thus, the arrangement of all routing network influences the efficiency of the architecture of the implemented digital circuit.

To map an application to a FPGA is used a software tool that converts the circuit description in hardware description language (HDL) into a stream of bits to program in the FPGA. The process to convert the circuit from HDL to bits is composed by four steps: synthesis, technology mapping, placement and routing [13]. The first step transforms the HDL description of the circuit into a set of boolean gates and Flip-Flops, including the wiring connections between them. The technology mapping step takes the boolean network and converts into a k-bounded cells (k-input LUTs and Flip-Flop), and can also form the clusters of k logic blocks. In this step the circuit can also be optimized in depth, area or power. The third step consists in connecting and placing the logic blocks close together to minimize the wiring (wire length-driven placement), to place blocks to balance wiring density (routability-driven placement) or to maximize circuit speed (timing-driven placement). For last, the routing step assigns nets to the routing
resources, in order to prevent one routing resource sharing more than one net.

4.5 Conclusion

Microprocessors are SPP designed and optimized for specific functions, in order to have low computation time and a good performance. In these processors is usually used a high-level programming language, as C, to implement the digital circuit and then through a compiler the program is converted to machine code, according to the model of the microprocessor.

A DSP is based on Modified Harvard architecture, and thus is optimized for mathematical functions specially for real-time signal processing. The architecture of the DSP allows in just one clock cycle to perform multiple instructions such as multiplication, accumulation, shift, data moves, etc.. This is a big advantage when implementing a digital filter, where in the filtering process it takes less time to perform the loop to calculate the filter output with smaller quantization errors.

On other hand, the MCU is specialized for control tasks thus characterized for being a more hardware limited microprocessor than a DSP relatively to mathematical functions for signal processing and lower CPU clock frequency. Despite this disadvantage, for the same resolution, a MCU is usually cheaper than a DSP, and some of the limitations can be solved using computational strategies.

The big advantage of a FPGA towards any kind of microprocessor is the functionality and flexibility to program all the hardware of the circuit to be implemented, which if combining with parallel computing architecture, as pipeline architecture or multiple multipliers, it is possible to achieve higher CPU clock frequency than a DSP. Besides these techniques there is already available some models of FPGA with DSP blocks integrated allowing a better performance in signal processing. However, comparing to a DSP or MCU the price of a FPGA is higher and can, according to the system implemented, consume more power.
Chapter 5

Finite Precision Arithmetic in Adaptive Filters

In Chapter 2 it was presented an analysis of the effects of finite word-length in digital filters, and it was concluded that the quantization errors can affect the performance of the filter, as for example, having a different filter frequency response from the one designed. When implementing an adaptive filter the effects of the quantization errors become even more significant. In this chapter it is presented an analysis of the effects of finite word-length in adaptive filters and an evaluation of the minimum number of bits of word-length to choose to implement an adaptive filter.

5.1 Effects of Finite Word-Length Arithmetic

Quantization errors may be originated from the ADC conversion, after acquiring the input samples into the system, from coefficient quantizations or from multiplications, considering that overflows in the accumulator never occur. All these effects combined can jeopardize the performance of a simple digital filter with fixed coefficients, mostly due to the filtering operation. However, while implementing an adaptive filter, besides the issues in the filtering operation, more quantization noise will be generated when updating coefficients values, since this operation is based on multiplications.

First on, consider a simplified example of an adaptive filter of 1st-order to understand the effects of finite word-length. The operation of an adaptive filter using the LMS algorithm is defined as:

\[
\begin{align*}
    y(n) &= w_0(n)x(n) + w_1(n)x(n-1) , \\
    e(n) &= d(n) - y(n) , \\
    w_0(n+1) &= \mu e(n)x(n) , \\
    w_1(n+1) &= \mu e(n)x(n-1) , \\
    n &= 0,1,...
\end{align*}
\]

(5.1)

where the coefficients of the filter \(w_0(n)\) and \(w_1(n)\) change at each iteration, depending on the step size \(\mu\), the error signal \(e(n)\) and the respective input sample. The system input signals are \(x(n)\) and \(d(n)\) and as output signals there are the filter output \(y(n)\) and the error signal \(e(n)\). Similar to the analysis in Chapter 2 for digital filters with fixed coefficients, the filter operation and the LMS algorithm can be
modelled with an additive noise structure due to quantization as:

\[
\hat{y} = \hat{w}_0 \hat{x}_A + \hat{w}_1 \hat{x}_B + \varepsilon_{0A} + \varepsilon_{1B} ,
\]

\[
\hat{e} = \hat{d} - \hat{y} ,
\]

\[
\hat{w}_0 = \hat{w}_0 + \left( (\mu + \varepsilon_{\mu}) \hat{e} + \varepsilon_M \right) \hat{x}_A + \varepsilon_{MA} ,
\]

\[
\hat{w}_1 = \hat{w}_1 + \left( (\mu + \varepsilon_{\mu}) \hat{e} + \varepsilon_M \right) \hat{x}_B + \varepsilon_{MB} ,
\]

where \( \hat{y} \) corresponds to the filter output quantized, \( \hat{w}_0 \) and \( \hat{w}_1 \) are the quantized filter coefficients, \( \varepsilon_{0A} \) and \( \varepsilon_{1B} \) are the quantization errors of the multiplications in the filtering equation, \( \hat{d} \) is the quantized sample of \( d(n) \), \( \hat{e} \) is the quantized error signal, \( \varepsilon_{\mu} \) is the quantization error from the step size (which is a constant), \( \varepsilon_M \) is the quantization error after multiplying the step size by error sample and, for last, \( \varepsilon_{MA} \) and \( \varepsilon_{MB} \) are the quantization errors for multiplying the respective sample of \( x(n) \) by the last multiplication result when updating the coefficients \( w_0 \) and \( w_1 \), respectively. Figure 5.1 shows the model for the introduction of quantization errors for this 1st-order filter example.

In this type of filter it is not present a ‘real’ value of the coefficients, since they are not defined beforehand as in digital filters with fixed coefficients. However this does not imply the non-existence of coefficient quantization, on the contrary, the quantization noise inherent from the coefficient becomes more complex, as can be seen in Equation 5.2. Whereas in digital filters with fixed coefficients, the coefficient quantization only depends on the resolution used and the round-off operation, in adaptive filters the coefficient quantization will depend on the quantization of the step size \( (\varepsilon_{\mu}) \), the quantization of the error signal sample \( (\hat{e}) \), the quantization of the input sample (either \( \hat{x}_A \) or \( \hat{x}_B \)) and the quantization noise from both multiplications \( (\varepsilon_M \) and either \( \varepsilon_{MA} \) or \( \varepsilon_{MB} \)). All these quantization noises will be included in the filtering process, thus influencing the error of the filter output \( \hat{y} \). After this, the filter output sample it will be used to calculate the error signal \( \hat{e} \), where the quantization noise of \( \hat{y} \) will also affect the quantization noise of \( \hat{e} \). This means the quantization errors will propagate throughout all steps of the algorithm and along the iterations, having a recursive nature.
Considering now a generic adaptive filter of $N^{th}$-order, the filter output $\hat{y}(n)$ is:

$$\hat{y}(n) = \sum_{k=0}^{N} Q[\hat{w}_k(n)\hat{x}(n-k)], \quad n = 0, 1, \ldots$$

(5.3)

where $\hat{w}_k(n)$ is the coefficient of index-$k$ and $\hat{x}(n-k)$ the respective input sample already quantized (from the ADC). This traduces in $(N + 1)$ quantizations performed in the filtering process related to the multiplications. The quantization errors of $\hat{y}(n)$ can be reduced if rounding is used after each multiplication, however this quantization error still can be significant if the filter order, $N$, is large due to the successive quantizations from multiplications.

Regarding the update of the coefficients, the critical steps are the two multiplications to calculate the variation of each coefficient, as previously seen in the example of the Equation 5.2. For each coefficient, initially it is performed the multiplication of the quantized step size by the error signal sample, converting then the intermediate result to the available hardware resolution. This intermediate result is then multiplied by the corresponding input sample and only then the final result is quantized to less bits. Simplifying, each coefficient in the LMS algorithm is updated as:

$$w_k(n+1) = w_k(n) + [\mu e(n)]\hat{x}(n-k), \quad k = 0, 1, \ldots, N, \quad n = 0, 1, \ldots$$

(5.4)

and, taking into account quantization error, coefficient update becomes:

$$\hat{w}_k = \hat{w}_k + [(\mu + \epsilon_\mu)\hat{e} + \epsilon_M]\hat{x}_k + \epsilon_{M_k}, \quad k = 0, 1, \ldots, N,$$

(5.5)

where $\hat{w}_k$ is the quantized coefficient of index-$k$, $\epsilon_\mu$ is the quantization error of representation of the step size in fixed-point, $\epsilon_M$ is the quantization error of the first multiplication, $\hat{x}_k$ is the input signal delayed by $k$ samples and, lastly, $\epsilon_{M_k}$ the quantization error of the final multiplication. These quantization errors will form a recursive nature through all the steps of the adaptive system. Due to a higher number of arithmetic operations, namely multiplications, can then jeopardize deeply the accuracy of the adaptive filter. It is then important to address this problem and try to find solutions to reduce it.

In Chapter 2 it was referred the filter structures also affect the quantization error present in the output signal, with emphasis on the cascade structure for being less sensitive to coefficient quantization. This statement present in [3] or [1] is valid for digital filters with fixed coefficients, however the same is not directly applicable to adaptive filters, since this type of structure will imply a very slow convergence and/or issues with non-convergence of the LMS algorithm.

### 5.2 Examples of Fixed-Point Adaptive Systems

The problems that can arise from using fixed-point representation with finite number of bits in any digital system may jeopardize its performance. For the case of adaptive filters, there are more factors contributing to the effects of finite word-length than in a simple digital filter, as already analysed. Quantization errors are affected by the round-off operation and even depend on the values to be processed, largely on the resolution used. If implementing an adaptive filter is essential to guarantee good performance and stability of the LMS algorithm, and therefore these quantizations errors must be taken into consideration.

For this reason, in the following two subsections are presented results of simulations performed in MATLAB environment for two cases of adaptive systems. Results are compared for different implementations
with finite number of bits and with full precision, to evaluate the effects in an adaptive filter in both performance and cost perspective. It was chosen to test for 8, 16 and 32 bits of resolution (fixed-point in $Q_B$-format for words of $(B + 1)$ bits), since these are the most common resolutions used in processors and for terms of comparison, despite for example in a FPGA it can be chosen any number of bits as word-length. From the analysis of the LMS algorithm the update of the coefficients is the most problematic, because it has two multiplications and it is a very important step of the LMS algorithm to converge and to be stable. One of the first conclusions in a previous section is that the quantization error of multiplications depends on the operands themselves, where small operands can lead to bigger quantization errors than larger operands. Thus, to test the performance of the whole LMS algorithm using fixed-point representation two different kind of applications of adaptive systems were chosen: noise cancellation and system identification. The reason to choose these two applications is due to the fact that for the first case the error signal supposedly tend to a desired signal without any noise, which again is not predictable but most definitely not zero (considering then objectively large values), and for the second case the error signal supposedly tends to zero if the adaptive filter is equal to the unknown plant (considering then very small values).

5.2.1 Noise Cancellation

One of the applications of adaptive filters is noise cancellation, where it is possible to remove noise embedded in a signal. The system implemented for noise cancellation is illustrated by a block diagram in Figure 5.2. In most cases there is only available the signal corrupted with noise, $d(n)$, and the noise signal, $x(n)$. However, since this implementation is performed in a computational environment for simulation, both signal and noise sources were created. Thus, the signal $d(n)$ is composed by the signal $m(n)$ and the noise signal $x'(n)$, as:

$$d(n) = m(n) + x'(n), \quad n = 0, 1, \ldots$$

(5.6)

The input signal of the adaptive filter, whose transfer function is represented by $W(z)$, is the noise signal $x(n)$. Noise signal $x'(n)$ is not the same as $x(n)$, but they are correlated since they are both generated by the same source. In this case it reflects a delay of 0.45 ms and an attenuation factor:

$$x'(n) = 0.85 \times x(n-20), \quad n = 0, 1, \ldots$$

(5.7)

The adaptive filter output is referred as $y(n)$, and the system output is the error signal $e(n)$. The steps of the LMS algorithm for this system for every time instant $n = 0, 1, \ldots$ are:

$$y(n) = \sum_{k=0}^{N} w_k(n)x(n-k),$$

$$e(n) = y(n) - d(n),$$

$$w_k(n+1) = w_k(n) + \mu e(n)x(n-k), \quad k = 0, 1, \ldots, N.$$  

(5.8)

The order of the adaptive filter has to be chosen appropriately with the application requirements, whereas a small order filter will have a frequency response less selective and a high order filter will meet the ideal requirements. The step size of the algorithm, $\mu$, has also to be chosen according to the specific application, which will allow the convergence of the system, adapting to the variations of the input signal, contributing for the time of convergence and, for example in noise cancellation, can imply different levels
of cancellation. Therefore, according to this specific application of noise cancellation and input signals, was chosen a 31\textsuperscript{th}-order FIR filter ($N = 31$) as the adaptive filter, whose coefficients were initialized at zero, and a step size as $\mu = 0.01$.

For simulation purpose the signal source generates a square wave with frequency of 85 Hz when the sampling frequency is $F_S = 44.1$ kHz:

$$m(n) = 0.8 \text{ square}(2\pi \times (85/F_S) \times n), \quad n = 0, 1, ...$$

(5.9)

The noise signal, $x(n)$, is generated as a sinusoidal wave where at a certain point in time changes its frequency, from 1676 Hz to 2875 Hz:

$$x(n) = \begin{cases} 
0.2 \sin(2\pi \times (1676/F_S) \times n), & 0 \leq n \leq \alpha \\
0.2 \sin(2\pi \times (2875/F_S) \times n), & n > \alpha 
\end{cases}$$

(5.10)

where $\alpha = 2^{12} = 4096$ samples, in order to see the reaction of the LMS algorithm to a abrupt change in the noise signal. It is important to note that the characteristics of the noise signal change only after the LMS algorithm has converged.

A first simulation of this system was conducted with the default precision of MATLAB, which is 64 bits using floating-point representation. Its results will be used to compare the performance of the other quantized systems results, since the quantization error in this case can be seen as zero due to the very high resolution, thus hereafter it is referred as the full precision result. In all the performed simulations it is used rounding instead of truncation as round-off operation for the sake of reducing the quantization error and because it is an operation with an easy implementation on any processor.

Figure 5.3 illustrates signals $m(n)$, $d(n)$ and the system output $e(n)$. It can be seen in the signal $d(n)$ the noise signal frequency changing after 4096 samples. Focusing on the left half of the sample interval in Figure 5.3, the error signal $e(n)$ after approximately 500 iterations becomes very similar to the original signal $m(n)$, yet still showing a small amplitude noise. When the frequency of the noise signal increases, the LMS algorithm tries to follow that change, and after approximately 500 iterations the algorithm converges again, having an error signal $e(n)$ a lot similar to the original signal $m(n)$. Comparing this cancellation with the left half, in this case the amplitude of the noise that remains in the signal is smaller, resulting in a better performance in terms of cancellation. Besides the difference in terms of cancellation of the noise, in either cases the algorithm adapted to the different characteristics of the noise signal and was able to converge.

In Figure 5.4 is illustrated the output signal of the system for each implementation: with 8 bits, $e_{A8}(n)$, with 16 bits, $e_{A16}(n)$ and with 32 bits, $e_{A32}(n)$. The superscript $A$ may be ignored in this chapter. It is present here because in the next chapter different implemented solutions results (with superscript $B$, $C$ and $D$) will be compared with these results. The first remark is that when implementing with 8 bits, the
Figure 5.3
Results of the full precision implementation of the noise cancellation system, featuring the signal $m(n)$, the signal $d(n)$ and the signal $e(n)$.

Figure 5.4
Representation of the error signal $e(n)$ of the noise cancellation system for each implementation with finite word-length.
Figure 5.5
Quantization error of the error signal $e(n)$ in each fixed-point representation of the noise cancellation system.

LMS algorithm diverges and the output of the system is practically the same as the input signal $d(n)$. For the other two implementations, at first sight it seems that the error signal is much similar to the error signal in the full precision implementation. To evaluate if the finite word-length jeopardize the system, in Figure 5.5 is illustrated the quantization error for all the implementations. In each case the quantization error is defined as:

$$
\varepsilon_{B+1}^A(n) = e_{B+1}^A(n) - e(n), \quad n = 0, 1, \ldots
$$

(5.11)

where the index $B+1$ corresponds to the number of bits used with fixed-point representation, which can be 8, 16 or 32. With this example, it is proven that for 8 bits practically the whole noise signal is still present at the output and the noise cancellation did not occurred. For 16 bits the quantization error has a magnitude of $10^{-4}$ order, and for 32 bits the quantization error is even much smaller, with a magnitude of $10^{-9}$ order. Despite that the magnitude of the quantization error when using 16 bits is not as small as with 32 bits, this can be considered insignificant since it does not jeopardize the performance of the adaptive filter. Therefore, with this evaluation, when implementing a noise cancellation system the minimum required resolution seems to be 16 bits in order to have an acceptable performance and a cheaper system than using 32 bits.

5.2.2 System Identification

System identification with an adaptive filter is used when there is unknown system, like a black box, without any information of its characteristics, having only available its input and output signals. Using these signals and with the LMS algorithm, the adaptive filter will try to converge to a transfer function that can represent the unknown system. This means that the relation between the output and input signal that the black box imposes must be replicated by the adaptive filter. Figure 5.6 shows the block diagram of this application and identifies the signals involved. Since this simulation is performed in a computational environment, as the unknown system it was chosen a lowpass IIR filter of $4^{th}$-order, whose transfer function is represented as $P(z)$ (Equation 5.12). When the adaptive filter has replicated the unknown system, the error signal $e(n)$ is zero, since both filter output $y(n)$ and the output of the unknown system $d(n)$ would be equal.
The lowpass IIR filter chosen to represent the unknown system is characterized by:

\[ P(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3} - b_4 z^{-4}} , \]  
(5.12)

with coefficients:

\[
\begin{align*}
a_0 &= 0.0013 , & a_1 &= 0.0051 , & a_2 &= 0.0076 , & a_3 &= 0.0051 , & a_4 &= 0.0013 , \\
b_1 &= 2.8869 , & b_2 &= -3.2397 , & b_3 &= 1.6565 , & b_4 &= -0.3240 .
\end{align*}
\]

To find the transfer function of the unknown system, it is necessary to consider as the input signal of the adaptive filter the same as the input signal of the unknown system, \( x(n) \). The signal source generates a sequence of random samples at a sampling frequency of 44.1 kHz, originating a random signal, with amplitude inside the range of fixed-point representation, between \([-0.9, 0.9]\). The unknown system output signal is defined by the following recurrent equation:

\[ d(n) = \sum_{k=0}^{4} a_k x(n-k) + \sum_{k=1}^{4} b_k y(n-k) , \quad n = 0, 1, ... . \]  
(5.13)

Relative to the adaptive filter, the filter output signal, \( y(n) \), the error signal, \( e(n) \), and each coefficient of the filter, \( w_k \), are for every \( n = 0, 1, ... \) defined as:

\[
\begin{align*}
y(n) &= \sum_{k=0}^{N} w_k(n)x(n-k) , \\
e(n) &= d(n) - y(n) , \\
w_k(n+1) &= w_k(n) + \mu e(n)x(n-k) , \quad k = 0, 1, ..., N .
\end{align*}
\]  
(5.14)

The adaptive filter order and the step size were once again chosen according to this specific application, in which the adaptive filter is a FIR filter of 31th-order (\( N = 31 \)), whose coefficients were initialized at zero, and the step size used is \( \mu = 0.01 \), as in the first application of noise cancellation.

Also as in the previous example, it was performed initially a simulation of this adaptive filter with the default MATLAB precision, which is 64 bits in floating-point, to be considered as the full precision result for comparison purposes with the results of implementations with 8, 16 and 32 bits with fixed-point. Once again, in every simulation executed it is used rounding as round-off operation in order to reduce beforehand the quantization error.

Full precision results are illustrated in Figure 5.7, showing at the first sub-plot the input signal \( x(n) \) for the unknown system and for the adaptive filter. In the second sub-plot it is shown the unknown system output \( d(n) \), and for last, in the third sub-plot the error signal \( e(n) \). In this last sub-plot, it is visible the error signal \( e(n) \) reducing until is approximately zero after 1000 iterations. Thus, in this case the

**Figure 5.6**

Block diagram of the system identification application.
LMS algorithm converges approximately after 1000 iterations, where the adaptive filter has a frequency response equivalent to the filter $P(z)$.

In Figure 5.8 is illustrated the error signal for each implementation with 8 ($e_{8}^{A}(n)$), 16 ($e_{16}^{A}(n)$) and 32 ($e_{32}^{A}(n)$) bits. With the lowest resolution, the adaptive filter cannot perform the system identification, having an error signal $e_{8}^{A}(n)$ very close to the input signal $x(n)$. Thus, as happened with noise cancellation, an implementation with 8 bits resolution suggests that it should be discarded for this kind of implementation. In the implementations with 16 and 32 bits (see Figure 5.8), the time of convergence is approximately equal to the time of convergence in the full precision implementation. Besides, both error signals, $e_{16}^{A}(n)$ and $e_{32}^{A}(n)$, are very similar to the error signal $e(n)$ from Figure 5.7. For a more detailed evaluation of the quantization error with these implementations in Figure 5.9 it is illustrated the quantization error for each tested resolution, defined as:

$$e_{B+1}^{A}(n) = e_{B+1}^{A}(n) - e(n) \quad , \quad n = 0, 1, ... \quad (5.15)$$

Results in Figure 5.9 corroborate that for a resolution of 8 bits the performance of the adaptive filter is very bad and should not be used. For the resolution with 16 bits, the quantization error has a magnitude of $10^{-3}$ order, while for 32 bits the quantization error has a magnitude of $10^{-8}$ order. With no doubt that the implementation with 32 bits obtains in a smaller quantization error, which is totally insignificant. The quantization error originated from the implementation with 16 bits is much larger than with 32 bits, however it can be considered insignificant when compared with the magnitude of the signals involved in this system. Thus, as in the case of noise cancellation, for this implementation the minimum resolution required to have an acceptable performance of a system identification application is 16 bits.
Figure 5.8
Representation of the error signal $e(n)$ of the system identification for each implementation with finite word-length.

Figure 5.9
Quantization error of the error signal $e(n)$ in each fixed-point representation of the system identification.
5.3 Solutions to Reduce Quantization Noise

The simplest way to reduce the quantization noise from arithmetic operations is increasing resolution, however it will also imply probably a more expensive hardware. Thus, since the goal is to have the most cheap hardware possible, another solution is to implement the arithmetic operations of the algorithm in different way. The first step, which is to choose rounding as round-off operation in any quantization, which reduces the quantization error was already considered. Besides this, other alternatives to reduce more significantly the quantization errors can be considered in order to increase performance for a 16 bits implementation and even to achieve the possibility of having an 8 bits system working properly. In the following subsections, three solutions are considered: perform quantization only after accumulation, using a power-of-two step size and using scaled coefficients.

5.3.1 Accumulation with Extended Resolution

Since there are available some models of microprocessors that have a built-in multiplier and accumulator with extended resolution, allowing to accumulate the results of the multiplications with double precision, and only after finishing the accumulation the result would be quantized for the hardware resolution. Using this type of implementation will enable a reduction of the quantization error in one of the most used operations in signal processing, filtering [2].

Using again the previous example of a 1\textsuperscript{st}-order adaptive filter (defined in Equation 5.1), Figure 5.10(a) shows the additive noise model when the filter output is calculated using an accumulator with extended resolution. Using this model the quantization error can be reduced since it is only performed one quantization instead of two (not including quantization of the coefficients), and the output of the filter is defined as:

\[
\hat{y} = \hat{x}_A \times \hat{w}_0 + \hat{x}_B \times \hat{w}_1 + \varepsilon_2 .
\]  

The model to update the coefficients is shown in Figure 5.10(b), which apply to both coefficients, being represented inside parentheses the corresponding variables regarding the update of the coefficient \( \hat{w}_1 \).
Now, considering once again a generic adaptive filter of \( N^{th} \)-order, with this implementation the filter output, \( y(n) \), is:

\[
\hat{y}(n) = Q\left[\sum_{k=0}^{N} \hat{w}_k(n)\hat{e}(n-k)\right], \quad n = 0, 1, ...
\]

(5.17)

Without extended accumulation, for each iteration of the LMS algorithm would be performed approximately \((2N + 3)\) multiplications, which translates into \((2N + 3)\) quantizations, and for \( N \) large results in a very large quantization error in overall. However if used the implementation with an accumulator with extended precision the quantization error can be drastically reduced in the filtering process, especially for large \( N \), because in this case \((2N + 3)\) multiplications will only originate \((N + 3)\) operations of quantization.

### 5.3.2 Power-of-two Step Size

One of the sources of noise introduction in the LMS algorithm, when implemented with fixed-point arithmetic, is its need for a double multiplication for the coefficients update. An intermediate result of the step size multiplication by the error sample should be stored and then used for the subsequent multiplication by each signal sample. One way to avoid this is to consider a step size whose value is a power-of-two, because in that case there is no need for the first multiplication because it can be substituted by an accumulation with a proper left shift of the result of the multiplication:

\[
w_k(n+1) = w_k(n) + 2^{-b}(e(n)x(n-k)) \quad , \quad k = 0, 1, ..., N \quad , \quad n = 0, 1, ...
\]

(5.18)

Although this avoids a multiplication, which is a time consuming operation, the most important is to avoid a multiplication quantization. Of course the efficiency of this form of implementation depends on the desired step size value and the implementation resolution (8, 16 or 32 bits). For instance, for the case just considered in the adaptive systems simulations, where a step size \( \mu = 0.01 \) was considered, an implementation with just 8 bits resolution would not produce a good solution because the step size quantization will be quantized to \(2^{-7}\) leading to a relative error of -22%. Of course higher resolution would reduce this significantly this error (0.1% for 16 bits and \( \approx 0\% \) for 32 bits).

But in the case that the step size would be limited to be a power-of-two, for \( \mu = 0.01 \) the relative error that would be obtained is the same for all three resolutions, -22%. On the other hand if a representation based on two power-of-two parcels would be possible (meaning two shifted accumulations of the multiplication output), in the case of 16 and 32 bits resolutions, the relative error of \( \mu \) will be decreased to -2.3% with \( Q[\mu] = 2^{-7} + 2^{-9} \). This quite simple example shows that, depending on the desired value for the step size and the hardware resolution it is possible to find solutions that avoid a second multiplication in the coefficients updating process.

Depending on the specific hardware available, this concept of power-of-two, or several power-of-two parcels constituting the quantized step size may take advantage of a processor particular ALU structure. These solutions will then be hardware dependent but may help to improve an adaptive system performance.
5.3.3 Scaled Coefficients

One of the problems in updating the coefficients in the LMS algorithm is that the step size is usually a small number, which can, after multiplied by \( x(n - k)e(n) \) (for each coefficient), result in a very small value that may be truncated or rounded to zero after quantization. This would imply not updating the coefficients and contribute to the algorithm diverging or even becoming unstable, as in the examples shown in the previous section for both noise cancellation and system identification when using 8 bits resolution. One solution that has become usual in digital signal processing is scaling when the processed signals have small amplitudes. For adaptive filters this technique can be applied in the coefficients of the filter [16], considering scaled coefficients, referred as \( w'_k(n) = w_k(n)/\mu \). Using this type of implementation it is suppressed the multiplication of the step size in Equation 5.4. However, these scaled coefficients do not correspond to the real coefficients of the filter and therefore the filter output does not correspond to the correct one according to the LMS algorithm, so the step size must be included in the calculation of the filter output. Using this implementation, the filtering operation and the LMS algorithm become:

\[
\begin{align*}
    y'(n) &= \sum_{k=0}^{N} w'_k(n)x(n - k) \quad , \quad n = 0, 1, \ldots \\
    y(n) &= \mu y'(n) \quad , \\
    e(n) &= d(n) - y(n) \quad , \\
    w'_k(n + 1) &= w'_k(n) + x(n - k)e(n) \quad , \quad k = 0, 1, \ldots, N.
\end{align*}
\] (5.19)

Since in most cases now the filter output sample \( y'(n) \) is not a very small value, when multiplied by the step size it produces a much smaller quantization error then when multiplying the step size by \( x(n - k)e(n) \) in the update of the coefficients. This has particular importance in applications of adaptive filters where the error signal sample \( e(n) \) tends to be a very small value and, as explained before, after multiplied by the input sample \( x(n - k) \) and the step size can originate zero after all quantizations, which can corrupt the convergence of the filter. It is also important to note that the scaled coefficients and the intermediate filter output \( y'(n) \) have magnitudes larger than one, thus can no longer be represented in the \( Q_{B,}\)-format with \( B \) bits for fractional part, since part of the available bits must be used for integer part.

So for implementing this solution it is necessary to assure that all fixed-point arithmetic operations are performed correctly between different types of formats of the values to be processed.

5.4 Conclusion

When implementing a digital system the effects of using a finite number of bits will always be present, either in a digital filter with fixed coefficients or an adaptive filter. Regarding the LMS algorithm, the quantization errors are mostly originated from the arithmetic operations, and since the coefficients of the filter are updated according to the output and input samples, the errors are more complex and form a recursive loop when comparing with a digital filter with fixed coefficients, propagating through all the steps of the algorithm and throughout the iterations. In digital filters with fixed coefficients it is usually used a cascaded structure for being the less sensitive to quantization errors, reducing then the effects of finite word-length in the filtering process. However, the same does not apply to adaptive filters due to problems of instability. Therefore, it is only considered the direct form structure for adaptive filters.

It was performed a comparison for implementations with 8, 16 and 32 bits since these are the word-
lengths of digital words most used in DSP and MCU. A original LMS algorithm structure was implemented and it is concluded that the minimum required resolution for an acceptable performance of the adaptive filter is 16 bits, according to the examples shown in this section, for both critical cases where the error signal of the adaptive system tends to be either a number different of zero or to be a number very close to zero. For 8 bits resolution in either adaptive systems simulated the algorithm did not converge and the adaptation was not performed, thus is not recommended to use. For 32 bits resolution the quantization error is of a very small amplitude, and therefore exhibits a very good performance for the adaptive filter, for both noise cancellation and system identification.

To reduce the effects of quantization errors in the adaptive systems it can be used in some processors an accumulator with extended resolution, allowing to only perform one quantization at the end of the accumulation, or using scaled coefficients, allowing the accumulation of bigger amplitude values, generating in smaller quantization errors. Depending on the processor architecture and ALU structure, specific solutions can also be considered as in the case of a power-of-two step size.
Chapter 6

Improved Implementation of Adaptive Systems

Results from the previous chapter suggested that an adaptive system should only be implemented with at least 16 bits resolution to have an acceptable performance. Three solutions to reduce the quantization errors, accumulation with extended resolution, power-of-two step size and scaled coefficients, were considered in order to try to increase system performance and reduce resolution. In this chapter are presented the results of the simulations of implementation of the two already considered adaptive systems (noise cancellation and system identification) with the proposed solutions that will be hardware independent: extended accumulation and scaled coefficients.

6.1 Applications with Reduced Quantization Noise

The cost of an implementation of an adaptive system is affected by the type of processor chosen, which then imply the type of hardware available. In that way, the choice must be careful in order to not jeopardize the performance of the implemented system. Aspects as resolution, which derive from the hardware of the processor, is an important point to take into consideration when choosing the processor, as seen before. The objective of this dissertation is to have an efficient system with an adaptive filter, combining good performance and low cost. In digital signal processing a way to reduce the cost of the system is to use a processor with a minimum number of bits resolution.

Nevertheless, the number of bits chosen should not compromise the performance of the algorithm, and with less bits more error will be introduced in the algorithm and it can lead to instability. Most adaptive systems are based in the LMS algorithm and its structure imply a significant introduction of quantization errors, specially for low resolution, as seen in previous chapters. However, these quantization errors can be reduced if using different implementations of the original LMS algorithm. In the following subsections the effect of the solutions presented in the previous chapter on the performance of adaptive systems (noise cancellation and system identification) are considered.
6.1.1 Adaptive Noise Cancellation

To evaluate if both solutions, accumulation with extended resolution and scaled coefficients, are beneficial to adaptive noise cancellation, are presented the results of the implementation of these solutions for the particular example of adaptive noise cancellation used in the previous chapter considering also the use of rounding as round-off operation. There is an input signal, \( x(n) \), that is a sinusoidal noise signal, which at a certain point changes frequency. The other input signal, \( d(n) \), is composed by a correlated noise signal \( x'(n) \) and a desired signal \( d(n) \), which is a square wave. The adaptive filter has 32 coefficients and the step size is \( \mu = 0.01 \), as before.

In Figure 6.1 are illustrated the results of the implementation of the noise cancellation system using an accumulator with extended resolution for all the three resolutions defined before, 8, 16 and 32 bits. At first sight this implementation does not bring significant differences to the performance of the system comparing with the respective \( \varepsilon_{B+1}^A(n) \) presented previously for all three resolutions. However, this implementation reduces very slightly the quantization error, and the first conclusion seem to be that this solution is not very beneficial for neither resolutions. This small difference can be verified in Figure 6.2, where the quantization errors are illustrated:

\[
\varepsilon_{B+1}^B(n) = \varepsilon_{B+1}^B(n) - e(n) \quad , \quad n = 0, 1, ... \tag{6.1}
\]

where \( B + 1 \) represents the number of bits used, \( \varepsilon_{B+1}^B(n) \) corresponds to the error signal, or system output, for this type of implementation and \( e(n) \) the error signal of the original implementation considered full precision. Despite the small reduction of the quantization error, for the implementation with 8 bits the adaptive system still cannot converge and cannot cancel noise. For the other two resolutions, the performance of the adaptive filter remains practically the same as in the first simulation, since both have the same time of convergence of the full precision version, but with a little more noise presented in the system output. This imply that using extended resolution accumulator does not reduce significantly the quantization error as usually does in an implementation of digital filter with fixed coefficients [2].

For the second solution, an adaptive filter for noise cancellation using scaled coefficients was implemented. As referred before is important to understand that the values of the scaled coefficients have higher amplitude than the normal coefficients, and thus it is necessary to adjust the format of representation of the variables in the LMS algorithm. In this case, it is used a step size \( \mu = 0.01 \), which then after dividing in the coefficients, \( w_k'(n) = w_k(n)/\mu \), the coefficients instead of being values between \([-1, 1]\) due to the format-Q\(B\), are now in the range of \([-100, 100]\]. Thus to represent these values it is necessary to have 8 bits for the integer part for the coefficients. The same happen for the filter output, \( y'(n) \), which now has value in the range of \([-100, 100]\], thus it must be represented with 8 bits for integer part. In Figure 6.3 is illustrated the error signal, \( \varepsilon_{B+1}^C(n) \), for the three different resolutions, and it is specially visible the big difference in the implementation for 8 bits, when compared with the previous implementations. For this implementation, the performance of the noise cancellation system is much better than for the original or when using accumulator with extended resolution, nevertheless and as expected with inferior performance than for 16 and 32 bits resolution. The reduction of the quantization errors using scaled coefficients, \( \varepsilon_{B+1}^C(n) \), is also visible when comparing both Figure 6.4 with Figure 6.2, where the quantization error \( \varepsilon_k^C(n) \) shows a drastically reduction and both quantization errors for 16 and 32 bits, \( \varepsilon_{16}^C(n) \) and \( \varepsilon_{12}^C(n) \), respectively, have also decreased. With these results, it is possible to conclude that the solution of scaled coefficients is very beneficial to use in an adaptive system, which for some cases can guarantee the convergence of the algorithm, which was impossible with the original implementation.
Since both solutions are focused on reducing the quantization error and, according to the results, both bring advantages for the performance of the adaptive system, it is also presented a fourth implementation which combines both, accumulation with extended resolution for the filter output and scaled coefficients. Using the considerations already stated previously about the implementation of each solution, in Figure 6.5 it is illustrated the results of the adaptive noise cancellation system when using the combination of scaled coefficients and accumulation with extended resolution for the filtering and update of the coefficients. And once again, the more visible difference is in the lowest resolution, where the amplitude of the noise in the output signal is smaller than in Figure 6.3. In Figure 6.6 are presented the quantization errors, $e_{B+1}(n)$, for this implementation, and if comparing with the original one (see Figure 5.5) there is a very huge reduction of the quantization error, especially for the case where $B + 1 = 8$. For resolutions of 16 and 32 bits, the original implementation already produces a very small quantization error (see Figure 5.5), but it can be even more reduced using this combined solution.

In Table 6.1 it is presented the signal-to-quantization-noise ratio for each different implementation and resolution. Each tag represents an implementation: $NC_A$ represents the implementation with the original structure of the LMS algorithm, whose results were shown in the last chapter; $NC_B$ represents the implementation with the accumulation with extended resolution; $NC_C$ represents the implementation with scaled coefficients; and $NC_D$ represents the combined implementation of both solutions. As already seen in the results, the implementation using accumulation with extended resolution, $NC_B$, does not bring many advantages, especially for 8 bits resolution. If using scaled coefficients, $NC_C$, the vari-

<table>
<thead>
<tr>
<th>$B + 1$</th>
<th>$NC_A$</th>
<th>$NC_B$</th>
<th>$NC_C$</th>
<th>$NC_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>17.0</td>
<td>17.0</td>
<td>34.4</td>
<td>38.9</td>
</tr>
<tr>
<td>16</td>
<td>73.2</td>
<td>73.6</td>
<td>81.1</td>
<td>87.7</td>
</tr>
<tr>
<td>32</td>
<td>172.7</td>
<td>173.1</td>
<td>176.6</td>
<td>181.2</td>
</tr>
</tbody>
</table>
Figure 6.2
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive noise cancellation with accumulator with extended resolution.

Figure 6.3
Results of the implementation of adaptive noise cancellation with scaled coefficients for the different resolutions.
Figure 6.4
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive noise cancellation with scaled coefficients.

Figure 6.5
Results of the implementation of adaptive noise cancellation with both scaled coefficients and using an accumulator with extended resolution.
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive noise cancellation with scaled coefficients and an accumulator with extended resolution.

Figure 6.6

The quantization error has a more predominant reduction than with the previous implementation. Despite the first solution proposed here alone is not very beneficial to the reduction of the quantization error, when combined with scaled coefficient has a more significant reduction. As seen before, the biggest reduction of the quantization error is in resolution of 8 bits, where the SQNR increases approximately $22\, \text{dB}$ in $\text{NC}_D$ relative to the original $\text{NC}_A$. For the other two resolutions, 16 and 32 bits, the SQNR increases approximately $15\, \text{dB}$ and $8\, \text{dB}$ from $\text{NC}_A$ to the $\text{NC}_D$. Despite the reduction of the quantization error in these two resolutions not being very significant as with 8 bits resolution, the SQNR is already very high in the original implementation. Thus, for this type of adaptive system, these solutions have more impact in low resolutions implementations, not only meaning a much smaller quantization error but also guaranteeing the convergence of the adaptive filter.

6.1.2 Adaptive System Identification

When using an adaptive filter for system identification, the system output should tend to zero when the adaptive filter has converged to the characteristics of the unknown system. As seen before, the multiplication of very small values can produce higher quantization error than with larger values and in the update of the coefficients there is the multiplication of three values where two of them are very small numbers for this type of application. Therefore, this application can originate high quantization error when implemented in a digital processor with fixed-point representation. So, it is most relevant to evaluate the performance of the adaptive system for the solutions proposed to reduce the quantization errors. The adaptive system identification used to test is the same presented in the last chapter, in Figure 5.6, whose input signal $x(n)$ is generated by a signal source as a random signal and the unknown system is defined with the frequency response $P(z)$, representing a lowpass IIR filter. The adaptive filter used is of 31th-order and the step size is $\mu = 0.01$.

In Figure 6.7 is illustrated the result of simulating the implementation of the adaptive system identification using accumulation with extended resolution. Comparing these results with the original implementation in fixed-point illustrated in Figure 5.8, the difference is not much evident, resulting in a very small reduc-
tion of the quantization error:
\[ \varepsilon_{B+1}^B = e_{B+1}^B - e(n) \]  
\[ \text{(6.2)} \]
where \( B + 1 \) represents the number of bits of the resolution, \( e_{B+1}^B \) the error signal of the system for this implementation and \( e(n) \) the error signal for the full precision result presented in the last section for this type of application of adaptive filters. This conclusion can also be justified by the quantization errors \( \varepsilon_{B+1}^B(n) \) illustrated in Figure 6.8, which are very similar to the quantization errors in Figure 5.9. This solution seems not to bring any advantage in terms of the performance of the adaptive system for the resolution of 8 bits, since the LMS algorithm did not converged as in the original implementation presented before. For 16 and 32 bits resolution, the performance of the algorithm is slightly improved by a small reduction of the quantization error.

Despite the very small differences to the first solution, when implementing the LMS algorithm with scaled coefficients, there is improvement of the performance of the adaptive system, as can be analysed in Figure 6.9. The implementation of scaled coefficients is identical to the one implemented in the adaptive noise cancellation. Since the step size defined here is also \( \mu = 0.01 \), it will translate in scaled coefficients and filter output with amplitudes between \([-100, 100]\), since originally every signal had values between \([-1, 1]\), being necessary to guarantee 8 bits for integer part. Once again, this solution has a bigger impact when implementing with 8 bits resolution, because now the LMS algorithm is able to converge and the error signal \( e^C(n) \) tends approximately to a very small number. However, as expected, the performance for 8 bits is worse than the one with 16 or 32 bits. For these last resolutions, the quantization errors were also reduced significantly, increasing the performance of the adaptive system, but without any changes in the time of convergence, when compared with the full precision implementation (see Figure 5.7). The quantization errors are illustrated in Figure 6.10.

Both solutions proposed can be combined to have an implementation to reduce even more the quantization error produced by the finite word-length arithmetic in the LMS algorithm. Combining scaled coefficients with accumulation with extended resolution for the filtering process, produces results (illustrated in Figure 6.11) that when compared with the previous solution (Figure 6.9) is not very evident the different between the combined solution and the solution with only scaled coefficients. However if analysing the quantization error (Figure 6.12 and the Figure 6.10) is visible the reduction of the quantization error for all three resolutions. As in the last solution, the implementation for 8 bits resolution is able to converge, resulting in a good performance of the adaptive system with such low resolution. Comparing the three resolutions, it is evident that 8 bits resolution represents the worst performance, having a longer time of convergence and bigger quantization error. Despite that, for 16 and 32 bits, originally the quantization error is very small, with this solution the quantization can be even more reduced, also improving the performance of the adaptive filter.

To analyse in more detail the reduction of the quantization error in each solution, Table 6.11 presents the SQNR for each implementation and resolution. Each tag represents an implementation of the adaptive system identification: \( SI_A \) represents the implementation with the original structure of the LMS algorithm, whose results are shown in the last section; \( SI_B \) represents the implementation with the accumulation with extended resolution; \( SI_C \) represents the implementation with scaled coefficients; and \( SI_D \) represents the combined implementation of both solutions. The first remark to note is the comparison of the SQNR for \( SI_A \) and for \( NC_A \), where it can be seen that the quantization errors are bigger for this application of adaptive systems, since the error signal tends to very small values. Validating the results illustrated, the more significant reduction of quantization error is for resolution with 8 bits, where the SQNR increased approximately 27 dB from \( SI_A \) to \( SI_D \). For both implementations with 16 bits and 32 bits, the SQNR from the combined solution increased about 22 dB comparatively to the original im-
Figure 6.7
Results of the implementation of adaptive system identification with accumulator with extended resolution for different resolutions.

Figure 6.8
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with accumulator with extended resolution.
Figure 6.9
Results of the implementation of adaptive system identification with scaled coefficients for the different resolutions.

Figure 6.10
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with scaled coefficients.
Figure 6.11
Results of the implementation of adaptive system identification with both scaled coefficients and using an accumulator with extended resolution.

Figure 6.12
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of adaptive system identification with scaled coefficients and an accumulator with extended resolution.
Table 6.11
Signal-to-quantization-noise ratio (dB) in each implementation and resolution for adaptive system identification.

<table>
<thead>
<tr>
<th>B + 1</th>
<th>SI_A</th>
<th>SI_B</th>
<th>SI_C</th>
<th>SI_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10.8</td>
<td>11.0</td>
<td>31.7</td>
<td>37.6</td>
</tr>
<tr>
<td>16</td>
<td>63.7</td>
<td>63.8</td>
<td>77.5</td>
<td>86.0</td>
</tr>
<tr>
<td>32</td>
<td>157.8</td>
<td>158.0</td>
<td>173.2</td>
<td>179.1</td>
</tr>
</tbody>
</table>

Thus, albeit the very small quantization error for 16 and 32 bits resolution, these solutions can even reduce more the quantization error. So in conclusion, the combined solution proposed has a more significant improvement on the performance of the LMS algorithm, specially for lower resolution, guaranteeing the convergence for the implementation with 8 bits resolution, opposing with the original implementation, and a very large reduction of the quantization error.

6.1.3 Performance Analysis

Analysing the results of the simulations performed for the adaptive noise cancellation and the adaptive system identification to evaluate if the solutions proposed are beneficial to reduce the quantization errors, it was concluded that the solution using scaled coefficients allows more reduction than using only accumulation with extended resolution. This proves that the most critical step in the LMS algorithm is the update of the coefficients, having a bigger contribution to quantization error than the filtering process.

However, if combining both these solutions, it is possible to have an implementation with a very significant reduction of the quantization error. This proposed solution is very suitable for a low resolution implementation, because the original implementation does not guarantee convergence and stability of the LMS algorithm and also results in a very high quantization error when using fixed-point arithmetic. The solution combining scaled coefficients and accumulation with extended resolution allows a major improvement of the performance of the algorithm for 8 bits resolution, by not only reducing the quantization error but most importantly guaranteeing the convergance of the system. For the resolutions of 16 and 32 bits the original implementation already translates in good performance having a small quantization error but with this solution the quantization errors become even smaller.

In conclusion, for a implementation of 8 bits it is imperative to use this solution in order to have an efficient adaptive system according to the resolution. For the other resolutions it depends on the final application of the system if there is a need to reduce more the quantization error of the original implementation, since it is already a very small value. This decision will remain to the user, whom will evaluate if the performance of the original implementation is enough to meet the specifications for the performance of the system.

6.2 Integrated Circuits for Implementation

With the simulations performed it was possible to understand the limitations in the performance of adaptive systems relative to finite precision arithmetic and quantization from ADC. To surpass these problems, it is presented solutions to reduce the quantization errors that are generated in a digital signal processing system, using an accumulator with extended resolution for the filtering process and/or using
Table 6.III  
Characteristics and prices of suitable processors to implement noise cancellation system.

<table>
<thead>
<tr>
<th>Model</th>
<th>Bits</th>
<th>CPU Clock</th>
<th>Features</th>
<th>Area [mm²]</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC12F1571</td>
<td>8</td>
<td>32 MHz</td>
<td>RISC architecture</td>
<td>29.4</td>
<td>$0.39 @ 5ku</td>
</tr>
<tr>
<td>PIC24F04KA200</td>
<td>16</td>
<td>32 MHz</td>
<td>17x17 single-cycle multiplier</td>
<td>32</td>
<td>$1.00 @ 5ku</td>
</tr>
<tr>
<td>dsPIC33EP32MC202</td>
<td>16</td>
<td>70 MIPS</td>
<td>Single-cycle MAC 17x17</td>
<td>80</td>
<td>$1.86 @ 5ku</td>
</tr>
<tr>
<td>PIC32MX330F064H</td>
<td>32</td>
<td>100 MHz</td>
<td>Single-cycle MAC 32x16 and two-cycle 32x32 multiply</td>
<td>81</td>
<td>$2.45 @ 5ku</td>
</tr>
<tr>
<td>DSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMS320C5532</td>
<td>32</td>
<td>50 MHz</td>
<td>Dual MAC 16x16 and single MAC 32x32</td>
<td>144</td>
<td>$1.95 @ 1ku</td>
</tr>
<tr>
<td>TMS320VC5501</td>
<td>32</td>
<td>300 MHz</td>
<td>Dual MAC 16x16 and single MAC 32x32</td>
<td>225</td>
<td>$4.95 @ 1ku</td>
</tr>
<tr>
<td>FPGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XC6SLX4-2TQG144C</td>
<td>–</td>
<td>———</td>
<td>3840 Logic Cells/Elements and 300 CLBs</td>
<td>484</td>
<td>$11.48 @ 1ku</td>
</tr>
<tr>
<td>EP4CE6E22C8</td>
<td>–</td>
<td>———</td>
<td>6272 Logic Cells/Elements and 392 CLBs</td>
<td>484</td>
<td>$13.15 @ 1ku</td>
</tr>
</tbody>
</table>

scaled coefficients. Both these solutions were validated in the previous section, proving that can reduce significantly the quantization error and improve the performance of the adaptive system.

In a previous chapter, it is presented a description of possible integrated circuits where can be implemented an adaptive system. Since the objective of this work is to envisage implementation of adaptive systems to be effective and if possible cheap without jeopardize the performance of the LMS algorithm, it was chosen the following three types of integrated circuits: MCU, DSP and FPGA. Each one of them has different architectures which will imply in different areas, power consumption and prices. In Table 6.III is shown some models for each type of integrated circuit and its characteristics, that can be able to support an adaptive system.

In the MCU category, there is available models with 8, 16 and 32 bits resolution. The lowest resolution model offers more limitations in terms of hardware available to implement an adaptive system, however it has the smallest area and it is the cheapest IC. For the model of 16 bits, there is available hardware to perform a multiplication of 17x17 bits in just one clock cycle, which is an improvement from the previous model, but the price is higher and the area is bigger. There is available some models of 16 bits resolution MCU that contain some digital signal processing blocks, which then are able to perform in just one cycle an 17x17 bits MAC instruction. This new feature is reflected in both price and area of the MCU. Nowadays there is already available MCU with 32 bits resolution with a considerable CPU clock frequency and allows to perform in just one cycle 32x16 MAC instruction.

DSP are IC that are specially designed for digital signal processing calculations as filtering, and therefore there is available is most of the recent models two arithmetic and logic units (ALU) which can perform in one clock cycle either two 16x16 MAC instruction or one 32x32 MAC instruction. With these features, the filtering process of the system implemented in the DSP will be faster if using 16 bits resolution. Comparing with the models of MCUs, the DSP have higher CPU clock frequency and have Modified Harvard Architecture, representing in a larger area and expensive ICs.

The FPGA models are characterized technically by its capacity of logic elements and CLBs, which translate to the available area to program the hardware needed. Since for this type of IC all the hardware is reprogrammable, aspects as CPU clock frequency or resolution of the system will be only defined after programmed the hardware, however in most cases is possible to achieve higher clock frequencies in FPGA than in DSP. Both area of the chip and price are usually higher comparatively to the MCU and DSP.

In the simulations performed it was used 8, 16 and 32 bits of resolution. For a implementation with 8 bits it can be used the MCU model PIC12F1571, and it can only be used the solution with scaled coefficients, since this model does not have an accumulator with extended resolution. This will translate in a very small and very cheap implementation of an adaptive system, with an acceptable performance, as seen
in the simulations. One of the limitations of the MCU is the low CPU clock frequency which means that this type of IC can take more time to perform all the computations of the LMS algorithm than any other model of IC. However, it is important to assure that the output sample of the system be available before a new input sample. This implementation can be used for applications whose the most important aspect is the cost and there is no need for a very high performance.

For a implementation with 16 or 32 bits there are available various models of ICs. If using a model that does not have an accumulator with extended resolution, it can be used the scaled coefficient solution only, otherwise it is recommended to use the combined solution in order to have less quantization error and thus improving the performance of the adaptive system. The choice of the IC to implement an adaptive system with 16 or 32 bits resolution will be depend exclusively on aspects as the price, area or power consumption. For example, there are available the 16 bits MCU models, PIC24F04KA200 and dsPIC33EP32MC202, where both are able to perform a single MAC instruction per clock cycle however the last model is a little more expensive and has considerable higher area than the first model. There is available a MCU model, PIC32MX330F064H, with a CPU clock of 100 MHz which is the smaller IC presented if interested on an implementation with 32 bits, although it performs one 32x32 multiplication in two clock cycles. Nevertheless, with higher area and smaller price it can be considered the DSP model TMS320C5532 with a CPU clock of 50 MHz which can perform one MAC instruction per clock cycle. It can be even chosen a DSP model with higher CPU clock frequency at the cost of higher price and larger area. Due to high flexibility in programming the hardware of an adaptive system on a FPGA, can be taken into account only the capacity of logic cells/CLBs regarding the complexity of the system to implement. To improve the efficiency of the implementation, it can also take into account a model of FPGA with a higher number of multipliers or having DSP blocks, at the expense of a larger area and higher price.

6.3 Headphones with Noise Cancellation

6.3.1 Noise Cancelling Headphones

In the class of interference cancellation by adaptive signal processing, one of the countless applications is noise cancelling headphones or headsets. Any type of headphone can provide passive noise reduction because of the sound absorbing materials used, especially high frequency noises. However, some of the noise signals have lower frequencies components and the passive noise reduction become limited in these situations [22]. The active noise reduction technique tries to compensate the limitations of the passive one. Headphones with active noise reduction are designed to produce an anti-noise signal so the environmental noise is cancelled (ideally) and the desired signal, such as music, speech, etc., can reach the ear of the user cleaned from any interference. Whereas the environmental noise is unpredictable and can have multiple sources, a non-adaptive system cannot be efficient to cancel the noise signal merged with the desired signal. Thus, an adaptive system is capable of tracking changes in the noise signal and therefore can reach a higher noise attenuation [22].

The first type of noise cancelling headphones designed was to be used in helicopters and aircraft cockpits in the late 1980s. Only in the beginning of the 21st century appeared the first noise cancelling headphones as a consumer product patented by Bose [23]. Until some years ago, the noise cancelling headphones were only circum-aural or around-ear headphones, such as illustrated in Figure 6.13(a),
which due to the sound absorbing materials and the almost perfect fit in the ear, were considered to provide the best passive noise reduction, so that a combination with active noise reduction was able to maximize the achieved noise reduction. However, nowadays some of the companies that produce these headphones are already focusing on in-ear noise cancelling headphones (Figure 6.13(b)). This type of headphones are much lighter and are the best suited for travelling and, most important, can reach almost the same or higher noise attenuation when compared with around-ear noise cancelling headphones [24].

Albeit most of these products are designed with state of art technology and can guarantee high noise reduction, they have not become very popular among users. One of the reasons is the price of the noise cancelling headphones, which the biggest percentage represents the circuitry used by the adaptive noise cancellation system. In Figure 6.14 it is presented a chart showing the market prices of nowadays available noise cancelling headphones comparing with the market prices of generic headphones. Despite one model, the other noise cancelling headphones have high prices for a type of headphone that can be used everyday. It is also shown the price difference in percentage for each brand, in order to understand the weight of the feature of noise cancellation in the final price of headphones. The difference of price is more accentuated in some brands than in others, however the difference is still significant from the normal type of headphones.

So, if an adaptive system could be used to cancel noise in a signal and then included in a headphone circuitry, it is interesting to find an implementation that can reduce the final price of noise cancellation headphones, but without compromising the performance of the system. As seen before, when implementing a digital system in a processor, the fixed-point arithmetic will produce quantization errors decreasing the performance of the system. Nevertheless, results from the envisaged solutions show that it is possible to reach an implementation with reduced number of bits with high precision.

### 6.3.2 Active Noise Cancellation Headphones

If the objective is to implement a noise cancellation system in headphones, consider the simplified representation of what could be a noise cancelling headphone using the LMS algorithm illustrated in Figure 6.15. By the internal circuitry of the headphone, the music signal is projected through a speaker to our ears. If exists environment noise, it will interfere with music we want to hear. So, to implemented a noise cancellation system, would be necessary to have one microphone redirected to the external part of the earcup of the headphone, to only capture the environment noise, and another microphone
in the internal part of the earcup to capture both music signal and a correlated environment noise, as represented by the microphones A and B, respectively, in Figure 6.15. The noise captured by the microphone B is not exactly the same as the one captured in A, but they are correlated since they are both from the same source. Microphone B is located in order to capture exactly what the user listens.

The equivalent block diagram from this simplified scheme of noise cancelling headphones is shown in Figure 6.16, where it is also represented the placement of both microphones. Microphone A captures only noise signal, \(x(n)\), and microphone B captures the correlated noise signal, \(x'(n)\), the music signal, \(m(n)\) and the adaptive filter output or anti-noise signal, \(y(n)\). This means that the ear of the user hear the error signal \(e(n)\), and that is why the objective of the noise cancellation system is to have \(e(n) = m(n)\), meaning that the filter was able to estimate \(x'(n)\).

However since the simulation is performed in a computational environment, all the signals are generated. For simulation purposes noise source produces white noise at a sampling frequency of \(F_s = 44.1\, \text{kHz}\), which will be the input signal of the adaptive filter. The correlated noise signal is considered to be the noise signal \(x(n)\) attenuated and delayed by 68 µs (3 samples), in order to simulate the isolation of the headphone and the time that takes the noise to travel to the ear, relative to the position of microphone A. Thus, the correlated noise signal is defined as:

\[
x'(n) = 0.7 \times x(n-3) \quad , \quad n = 0, 1, ...
\]

(6.3)

The music signal used is an excerpt of the \textit{Prelude in C major} of BWV 846 by Johann Sebastian Bach, also captured with a frequency sampling of 44.1 kHz. The adaptive filter order and step size were adjusted according to this application of noise cancellation since it differs from the one presented before due to the different types of signals being processed. Therefore, it was chosen an adaptive filter with 64 coefficients and a step size of \(\mu = 0.005\).

In Figure 6.17 it is illustrated the result of the implementation of this system with 64 bits in floating-point in \textit{MATLAB}, which is to be considered as the full precision result, since the resolution and format used by the program produces an insignificant quantization error. In the first sub-plot is illustrated the music signal and in the second sub-plot is represented the music signal in grey and the signal \(d(n)\) which comprises both music signal and the correlated noise signal, in black. From the overlap of these two
signals it is possible to see the correlated noise signal. The third sub-plot it is shown in grey the system output, the error signal \( e(n) \), which should match the signal in the first sub-plot. Since this evaluation is not so easy with these type of signals, it is also shown in the third sub-plot (in black) the residual noise \( r(n) \), which is defined as:

\[
    r(n) = e(n) - m(n), \quad n = 0, 1, \ldots
\]  

(6.4)

which represents the remained noise included in the system output. With this signal is already possible to evaluate the performance of the noise cancellation. It can be seen that the LMS algorithm converges after approximately 12 000 iterations (0.27 s) and that the cancellation is not perfect, still existing some noise components over time.

Next, it was simulated the same system and signals for 8, 16 and 32 bits resolution for the original implementation and using the proposed implementation, combining scaled coefficients and accumulation with extended resolution. The results for the original implementation are presented in Figure 6.18 and the quantization error in Figure 6.19:

\[
    \epsilon_{B+1}^A(n) = e_{B+1}^A(n) - e(n),
\]  

(6.5)

where \( B + 1 \) represents the resolution of the system, \( e_{B+1}^A(n) \) the error signal for each resolution implemented and \( e(n) \) the error signal of the full precision implementation for this noise cancellation system. As in the results from the two examples (noise cancellation and system identification) considered in
Table 6.IV
Signal-to-quantization-noise ratio (dB) in each implementation and resolution for noise cancellation system for noise cancelling headphones.

<table>
<thead>
<tr>
<th>$B + 1$</th>
<th>NCH_A</th>
<th>NCH_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.6</td>
<td>25.8</td>
</tr>
<tr>
<td>16</td>
<td>47.4</td>
<td>77.2</td>
</tr>
<tr>
<td>32</td>
<td>143.7</td>
<td>170.8</td>
</tr>
</tbody>
</table>

the previous chapter, when using the original implementation and 8 bits resolution the adaptive system does not converge and all the noise signal is present in the output signal. For 16 bits resolution the performance of the system is acceptable, since there is small residual noise at the output however the quantization error is not very small. However, for 32 bits resolution there is a very good performance of the adaptive noise cancellation system, exhibiting also a small quantization error.

So, to reduce the quantization noise it is used the proposed combined implementation, whose results are illustrated in Figure 6.20. Initially comparing the result of full precision with these results, there seems to be only difference in the implementation with 8 bits, having more noise present in the music signal. The time of convergence is approximately equal for 16 and 32 bits relative to the full precision case, and a little higher for 8 bits. In Figure 6.21 it is presented the quantization error for each resolution relative to the full precision case:

$$
\varepsilon_{B+1}^D(n) = e_{B+1}^D(n) - e(n),
$$

(6.6)

where $B + 1$ represents the resolution tested, $e_{B+1}^D(n)$ the error signal for each resolution implemented and $e(n)$ the error signal of the full precision implementation for this noise cancellation system. As expected, for 8 bits resolution the quantization error has bigger magnitude than for the other resolutions, but after convergence it remains with a magnitude of $10^{-2}$. With this implementation, the adaptive system is already able to reduce the noise signal at the output, having now a small residual noise, albeit not so small as for the other resolutions. For 16 and 32 bits the performance of the noise cancellation is very good, and the quantization error after convergence remains with a magnitude of $10^{-4}$ and $10^{-9}$, respectively. Another way to evaluate the quantization error for these implementations is through the SQNR, which is presented in Table 6.IV. For the lowest resolution, there is an improvement of 25 dB of the SQNR, for 16 bits an increase of 30 dB and for 32 bits an increase of 27 dB. For this system, the improvement of the SQNR for each resolution has different meanings. For 8 bits resolution, besides decreasing the quantization error, the LMS algorithm is able to adapt and converge, despite the reduction of noise not being high. Opposing to the example shown before, for 16 bits the original implementation represented a quantization error not very small, therefore for this case the proposed solution translates in a reduction of the quantization error to a more acceptable magnitude, according to the resolution. For 32 bits, the SQNR is already very high for the original implementation, and despite the increase produced by the implementation of the proposed solution, it does not have a very significant effect. All these results validate the simulations presented previously in the examples of adaptive noise cancellation and the improvement of the performance of the adaptive system with the proposed solution. Once again, it is possible to have an efficient noise cancellation system with acceptable performance with only 8 bits resolution. For the other two resolutions, the quantization errors are so small that may be ignored.
Figure 6.17
Results of the full precision implementation of noise cancellation system using a music signal and white noise.

Figure 6.18
Results of the fixed-point implementation of noise cancellation system using a music signal and white noise.
Figure 6.19
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of noise cancellation system.

Figure 6.20
Results of the fixed-point implementation of noise cancellation system with the combined solution to cancel noise in a music signal.
6.3.3 Integrated Circuits Selection for Headphones with Noise Cancellation

The concept of noise cancelling headphones was initially focused to help pilots in aircraft to communicate without much interference with the control tower. But over time the need to listen a clear sound signal without noise reached the interest of common users more exclusively for their own leisure. Despite this type of headphones have a good performance they have not become a very popular product yet, mostly due to its price.

The system behind the noise cancellation in the headphones can be an adaptive noise cancellation system. When implementing a digital system the finite numeric representation will affect the performance of the system by introducing quantization errors, as exemplified in previous sections. The price of a digital system depends on the IC chosen to implement and its characteristics as the resolution, nevertheless these effects of quantization error are more significant if the resolution of the system is low. Therefore, the proposed solutions reduce the quantization error and improve the performance of the adaptive system. With these implementations it is necessary then to chose an IC to implement according to the specifications of the system.

Concerning the theme of noise cancelling headphones, multiple models are available with different specifications and performances. The requisites for performance of common noise cancelling headphones are less rigorous than the requisites of performance for a headset for a aircraft pilot or even for a music producer. There is people that would like to have noise cancelling headphone to use every day to listen to music on the way to work or when travelling, and maybe after a year would like to change the headphones for a new one. Thus, for this type of customers it is important to have a not so expensive product but also guaranteeing an acceptable performance, so it can be chosen the cheapest MCU for either 8 bits or 16 bits resolution, thus decreasing the final price of those headphones. However, for a pilot or music producer the performance probably is more important than the cost, so will be beneficial to use a DSP with 32 bits resolution or a FPGA which can be programmable to be even more fast than the DSP if there are any restrictions according to the processing time.

Figure 6.21
Quantization error of the error signal $e(n)$ for 8, 16 and 32 bits resolution for the implementation of noise cancellation system with the combined solution.
6.4 Conclusion

Using the example of adaptive noise cancellation and adaptive system identification it is proven that the proposed solutions in the previous chapter, scaled coefficients and/or accumulation with extended resolution, reduce the quantization error and improves the performance of the respective adaptive system. In every resolution simulated there is reduction of the quantization error, increasing the SQNR approximately around 20 dB and 25 dB in each resolution. Besides, these solutions for the implementation with 8 bits guarantees the stability and convergence of the LMS algorithm that is not possible in the original implementation.

The requisites to choose a IC to implement either these solutions should be according to the resolution, price and area. Some of the IC presented earlier have limitations in the hardware, for example, does not have accumulator with extended resolution, and for those cases it must be used the solution applying only the scaled coefficients. The choice between a MCU, DSP or FPGA should be in the perspective of the requirements of the final application of the system, which can be more directed to a low cost system with an acceptable performance or for a more demanding performance without considering much the price of the system, however in either way it is important to have low quantization noise.

Nowadays there is available a practical application of noise cancellation which are the noise cancelling headphones. There is already available multiple models and brands from a wide range of prices however this product has not yet caught the attention of users due to their higher prices when compared with normal headphones. Therefore, if using an implementation of the proposed solution with a cheap IC is possible to reduce the final price of noise cancelling headphones. Nevertheless, some noise cancelling headphones are specially designed for very good performance and the cost is not a very important aspect, as the headset for aircraft pilots, but does not mean that this solution cannot be applied, since it improves even more the performance of the system.
Chapter 7

Conclusions

Digital filters are very common in digital devices and are used for a wide variety of applications. They are divided into two classes: recursive and non-recursive. When implementing a digital filter into a digital signal processing system it is necessary to convert the signals to a finite number representation. The first step is the analogue-to-digital conversion, and since there is a finite number of bits to represent the sample values this will introduce quantization errors. During the filtering process, due to all the arithmetic operations, quantization errors will also be introduced, due to coefficient quantizations, product quantizations and/or overflows. In the case of a digital filter with fixed coefficients even before any arithmetic operation is performed, the frequency response of the filter is already different from the one initially designed, because the coefficients are approximations of the desired values of the coefficients, due to its quantization.

A digital filter with fixed coefficients is designed according to some specifications depending on its application, in order to produce a desired output signal when the input signal has certain characteristics. However if those characteristics change over time, a digital filter with fixed coefficients no longer produces the desired output. Thus in these cases it is necessary to use an adaptive filter, whose coefficients are adjusted at each time instant through an adaptive algorithm. The most used adaptive algorithm is the least mean squares, whose objective is to converge to optimum coefficients in terms of the minimum square value of the error signal, which corresponds to the Wiener filter. There are multiple applications of adaptive filtering, such as: system identification; inverse system identification; signal prediction; and noise cancellation.

To implement a system with an adaptive filter an integrated circuit able to compute all the arithmetic operations of the algorithm in an efficient way should be considered. Different types of integrated circuits are available, such as micro-controllers, digital signal processors and field programmable gate arrays. However, the limitation of finite number representation in the digital domain will affect the performance of the adaptive system. This effect is particularly important when on try to reduce the word-length of the binary words and to use an arithmetic and logic unit that with the least possible complexity. Besides the quantization error from the analogue-to-digital conversion, the effects of finite precision arithmetic in the adaptive filter are more significant than in a digital filter with fixed coefficients. The easy way to reduce these quantization errors is to increase the resolution of the hardware, nevertheless this will represent a more expensive system. This work analysed the effects of finite word-length and fixed-point arithmetic on filters with fixed-coefficients and on adaptive filters performance, based on a model of additive noise at the quantization stage. Quantization is present while quantifying constant coefficients (both coefficients of a constant coefficients filter and the step size of an adaptive filter) and at the end of all binary
multiplications. A minimum hardware resolution was established based on two application examples of adaptive systems, noise cancellation and system identification. Solutions allowing reducing quantization errors, without increasing the hardware resolution were considered, namely an implementation with extended resolution accumulation, the possibility of power-of-two step size and, finally, a scaled coefficients implementation.

Results from simulation of a noise cancellation system and an identification system with 8, 16 and 32 resolution showed that a minimum resolution of 16 bits should be considered in order for the system to work properly. Using 8 bits resolution and a regular implementation, based on quantization after each multiplication, showed that it is not possible to have an effective adaptive system, because the least-mean-square algorithm never converges.

In order to seek an adaptive system implementation with the lowest possible resolution, results were obtained considering two solutions: accumulation of products with extended resolution; and a scaled coefficients filter implementation. Obtained results show that for both solutions, or the combined solution of scaled coefficients and accumulation with extended resolution, the quantization error can be reduced, especially for lower resolution, as in the case of 8 bits, where without these improvements the algorithm showed problems on converging. Results from the two different adaptive systems proved that in this way it is now possible to have an efficient adaptive system with only 8 bits resolution.

A third solution, considering a power-of-two step size, was also considered in order to reduce quantization errors on the least-mean-square algorithm, but implementation of efficient systems based on this would be hardware dependent, so results were not presented.

Since the solution of scaled coefficients does not depend on the type of hardware, it gives the possibility to implement an effective adaptive system with very good performance in any type of integrated circuit, from a very simple 8 bits micro-controller to a more complex 32 bits digital signal processor, or even programming the entire hardware in a field programmable gate array. Quantization errors are effectively reduced when scaled coefficients and accumulation of products with extended resolution are considered.

This work showed that with these solutions it is possible to implement an efficient adaptive system with good performance with a reduced resolution in terms of number of bits word-length in order to achieve a cheaper system. If the project demands an outstanding performance it is usually chosen hardware with more resolution and without any more improvements in the hardware, however the considered solutions can also be used in this case, reducing even more the quantization errors effect, thus improving system efficiency.

Nowadays headphones and noise cancelling headphones are already available in market. Nonetheless there is a considerable difference of price between these two options. Depending on the brand, difference in price can reach, for example, 202%. This huge difference in price means that, at this time, noise cancelling headphones are mostly used by sound professionals, that need a good performance even if it is an expensive solution. Regular music listeners, like each one of us during our daily way to work, need a cheaper solution but would like to have the possibility of obtaining good environmental noise reduction as well.

Results were obtained and show that the proposed solutions for this practical application are effective even for systems implemented with lower resolution, as in the case of 8 bits. This means that it would be possible to consider the implementation of noise cancelling headphones based on a low resolution integrated circuit, reducing considerably the price while having a good performance. On the other hand, the proposed solutions may also be considered in a more expensive solution for professional use, because results showed that improved performance can be reached.
In a future development, the study of efficient implementation of systems with adaptive filters can be further pursued into another perspectives. For instance, it can be performed a study focused on how the step size influence the performance of the algorithm when implemented with fixed-point arithmetic. In this work was performed a comparison between implementations with three different resolutions since they are the most used in DSP and MCU, however it would also be interesting to evaluate which is the minimum resolution in a FPGA for an efficient implementation of an adaptive system whereas a FPGA can be programmed with any number of bits of resolution. Finally, after focusing in an efficient implementation in more theoretical and simulation realities it can be considered an experimental implementation on a FPGA, where it can be explored all the advantages inherent of the hardware programmability existent in a FPGA.
Bibliography


