AU : 2009-2010

Université de Sousse

Ecole Nationale d'Ingénieurs de Sousse

A thesis submitted in conformity

with the requirements for the degree of Master in

« Systèmes Intelligents et Communicants »

« Option : Microélectronique des Systèmes Embarqués»

FPGA implementation of a frequency domain equalizer

by

Anis SOUARI

Defended in **15/12/2010** in front of the jury

President	:	Bouraoui MAHMOUD, ENISo
Members of jury	:	Faten BENABDALLAH, ENISo
		Nejmeddine JOUIDA, ENISo
Supervisors	:	Mohamed Lassaad AMMARI, ENISo
		Sofiène TAHAR, Concordia University

Résumé

L'égalisation du canal est une technique qui permet de réduire les interférences entres symboles causées par le canal radio-mobile. Généralement, un égaliseur consiste en des filtres numériques dont les coefficients sont mis à jour selon le critère de forçage à zéro ou le critère de la minimisation de l'erreur quadratique moyenne. La taille du filtre égaliseur est un paramètre important qui influence les performances globales du système. Quand cette taille est élevée, la complexité numérique de l'égalisation augmente considérablement à cause de la convolution numérique.

Pour réduire la complexité des égaliseurs de tailles élevées, nous pouvons implémenter l'égaliseur dans le domaine fréquentiel. Le principe de ces derniers est de remplacer la convolution temporelle (filtrage) par une multiplication fréquentielle moins complexe. Toutefois, le passage au domaine fréquentiel nécessite l'utilisation des modules de la transformée de Fourier discrète (TFD).

Dans ce travail nous nous intéressons aux égaliseurs fréquentiels. Dans un premier volet, nous analysons, à l'aide des simulations Monte-Carlo sous Matlab, les performances d'un égaliseur fréquentiel linéaire sous différentes conditions de propagation. Le deuxième volet du travail consiste en la conception et l'implémentation d'un égaliseur fréquentiel sur un FPGA. Après la définition des besoins matériels et logiciels, nous présentons une méthodologie de conception et d'implémentation sur FPGA et nous discutons l'optimisation de l'architecture proposée.

Abstract

The channel equalization is a technique allowing reducing the inter-symbol interference caused by the radio-mobile channel. Generally, an equalizer consists in digital filters where the coefficients are updated according to the zero forcing criteria or the mean square error minimization criteria. The equalizer size is an important parameter that influences the global performances of the system. When that size increases, the equalizer computational complexity augments considerably because of the convolution.

To reduce the complexity of the long size equalizers, we can implement them in the frequency domain. The idea consists in replacing the convolution in the time domain by a less complex multiplication in the frequency domain. However, the passage to the frequency domain requires the use of the discrete Fourier transform (DFT) modules.

In this work, we were interested to the frequency domain equalizers. First, based on Monte-Carlo simulations on Matlab, we analyzed the performances of a linear frequency domain equalizer put under different propagation conditions. Secondly, we designed and we implemented the frequency domain equalizer on FPGA. After defining the required tools, we presented an FPGA design and implementation methodology and we discussed the optimization of the proposed architecture.

To mo loving family

Acknowledgements

I would like to take this opportunity to express my gratitude to all people who contribute in making this work possible.

First and foremost, I wish to thank my supervisors Dr. Mohamed Lassaad AMMARI and Dr. Sofiène TAHAR for their assistance, guidance and patience throughout this project.

I am grateful to all HVG members for their encouragement, help and also the family environment that they offer me. I specially would like to acknowledge the help of my dear friends Naeem and Fariborz.

I thank all the faculty members of the Ecole Nationale d'Ingénieurs de Sousse and the staff of the department of Electrical and Computer Engineering of Concordia University for their support and hard work.

Finally, thank you Taoufik for your support and your help that you gave me in Canada.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AWGN	Additive White Gaussian Noise
BLMS	Block Least Mean Square
DFT	Discrete Fourier Transform
Fast LMS	Fast Least Mean Square
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IFFT	Inverse Fast Fourier Transform
ISI	Inter-Symbol Interference
LMS	Least Mean Square
QAM	Quadrature Amplitude Modulation
SNR	Signal-To-Noise Ratio

Chapter 1

Introduction

1.1 Motivation

Nowadays, due to the radio-mobile development, the digital sound and image diffusion and the multimedia services increase, we attend a true explosion in terms of the request of the digital transmission techniques. These new services usually require transmitting a huge quantity of information in the narrowest frequency band if it is possible and also they require reducing the transmitted power or transmitting the information through some severe channels. So, it is obvious that achieving such objectives will cause some problems for the transmission system designers; in fact, in the case of severe channels such as multipath channels, we need some techniques to avoid the inter-symbol interference (ISI) created by the frequency selectivity of the channels.

To steer clear of the channel frequency selectivity, we can refer to a lot of techniques among them we can mention the equalization; time domain equalization has been the most used method of eliminating ISI, the most used architectures are the linear equalizer and the decision feedback equalizer. Those different equalizers consist in digital filters where the coefficients are generally updated using adaptive algorithms. The size of the filter is an important parameter which influences the global performances of the system. In fact, for a "better" equalization, the equalizer size must be bigger than the impulse response of the channel. So, when the equalizer size increases, the equalizer complexity increases considerably because of the digital convolution. Using common time domain techniques to compensate for distortion introduced by such channels may be very computationally intense. To decrease the filtering complexity, the equalizer can be implemented in the frequency domain. We talk in this case about the frequency domain equalizer. The idea of this technique consists in replacing the timing convolution with a frequency multiplication less, so this method offers a low complexity growth in comparison with the time domain approach. This technique requires the use of the Fast Fourier Transform (FFT) modules.

The objective of this master project is the implementation of the frequency domain equalizer on FPGA and the performance analysis of the designed system. We have first to identify the tools and the software required to achieve this project, then after implementing the equalizer, we have to compare the performance of what we implement with the theoretical performances of the frequency domain equalizers.

1.2 Related work

In 1965, Lucky led the way to the adaptive equalization by developing the zero forcing algorithm which was the corner stone for the high speed modems. Meanwhile, Widrow was working on an other algorithm called the Least Mean Square (LMS) algorithm which became very important for many applications. In 1973, Waltzman and Schwartz introduced the first frequency domain implementation of the LMS algorithm where the coefficients were updated using an isolated training sequence. Since then, researchers never stop developing the adaptive algorithms thanks to its efficiency in many domains.

Due to its efficiency, the LMS algorithm knows a great use in various applications; among them we find the FPGA-signal processing based applications. The papers dealing with this topic are uncountable. We will try in this section to present some works dealing with the implementation of the LMS algorithm in FPGA which have been done.

The paper describing the most similar work to our project is that written by Dr. Chris Dick and Dr. Fred Harris. It is named FPGA QAM Demodulator Design [3]. This paper examines the FPGA implementation of an adaptive equalizer and carrier recovery loop for a 50 Mbps and 16-QAM receiver. To minimize inter-symbol interference in their project, Dr. Dick and Dr. Harris use an adaptive equalizer based on the LMS algorithm which is connected in cascade to a matched filter. To build their system, they employ a fully parallel design consisting of 8 FIR processing elements (PE) and 8 LMS processors. For the implementation on the Virtex II FPGA, the system level design tool System Generator was used.

Some researchers working on FPGA implementation of adaptive equalizers try to modify the LMS algorithm in a way that the complexity when implementing in the hardware is reduced. This is the case of the project described in the paper named “algorithm and architecture design for a low-complexity adaptive equalizer” [1] where researchers implemented the GSPT LMS (Grouped Signed Power-of-Two LMS) algorithm to test the capability of an adaptive

equalizer in equalizing 8PSK signals in several practical channels. To ensure the efficiency of their algorithm, they implemented three adaptive equalizers based on the LMS algorithm, Chen's scheme, and the proposed GSPT LMS algorithm. After comparison of the results, they found that the proposed GSPT LMS algorithm can be implemented with much lower complexity, which is about 30% of the hardware resource required by the conventional LMS algorithm, or about 50% of that in Chen's scheme.

Another paper dealing with adaptive filtering using LMS algorithm is called FPGA Implementation of a Single Channel GPS Interference Mitigation Algorithm [2]. In this project the paper's writers implemented an adaptive filter for narrow band interference excision in Global Positioning Systems (GPS). The Adaptive Filter was implemented in schematic VHDL using four main blocks which are the Coefficient Update Block, the Input Signal Delay Block, the FIR Algorithm Block and the Tree Adder Block. The adaptive filter using 33 taps and a sampling rate of 8 MHz was tested with simulink and it converges in 20μs. In this paper the writers conclude that "The single channel delayed LMS adaptive algorithm is an effective technique for removing narrow-band interfering signals from GPS receivers. It can be effectively implemented using FPGA technology that can be seamlessly inserted between a GPS antenna and receiver."

As we mentioned in the beginning of the section, there are a lot of papers dealing with the hardware implementation of the LMS algorithm applications. So, we tried to present the closest related work.

1.3 Thesis outline

The thesis is structured in the following way; chapter 2 discusses the notion of digital filtering and it focuses on the adaptive filtering by explaining its principle. Some applications of adaptive filtering are also presented in this chapter. Chapter 3 deals with equalization. Time domain and frequency domain equalizers are defined in this section as well as the algorithms used by each equalizer. The implementation of the equalizers, the tools used and a comparison analysis are presented in chapter 4. The fifth chapter opens new horizons by presenting new ideas that can be applied on this project as a further work.

Chapter 2

Adaptive filtering

2.1 Introduction

Adaptive filters [22] are mainly digital filters using adaptive algorithms to update their coefficients. In this section, we will introduce the adaptive filter and we will enumerate their applications.

2.2 Adaptive filtering

Digital filters are mathematical algorithms which are used to modify the digital signals. It is the same procedure followed by the analogue filters to operate on the analogue signals. Equation 2.1 represents the output of a linear digital filter.

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \qquad 2.1$$

Where *h(k), k = 0,1…N-1* are the filter coefficients, *x(n)* is the filter input and *y(n)* is the filter output.

Digital filtering has a lot of advantages among them we can mention:

- Easy implementation of adaptive filtering in case of the use of a programmable processor.
- The reuse of the filter output and input can be guaranteed.
- Analogue filters have limitations.
- Can be improved as the VLSI technology is improved.
- Performance is stable.
- The filter can operate on more than one filter at the same time.
- Precision.
- It is beneficial for the biomedical applications since it can operate at lower frequencies.

As it has many advantages, it has also some drawbacks. The following points summarize the most remarkable disadvantages:

- The hardware resource on which the filter is implemented can limit its speed
- Building a digital filter is much more difficult than building an analogue one.

The adaptive filter coefficients are changing when they are operating in a changing environment. So, recursive algorithms are necessary to put the filter coefficients up to date. The most accurate example that reveals the importance of the adaptive filters is the telephone system where the echo caused by the impedance assorts represents a considerable source of annoyance for this system users. So, the adaptive filter has to eliminate this echo by imitating the echo response of the echo path.

Although they do great job in most cases, implementing adaptive filters in the time domain can not be efficient for some applications where we have an increasing complexity and as a result a very long impulse response. For these special cases, we have resort to implement the adaptive filter in the frequency domain. From here, the importance of the Fast Fourier Transform (FFT) and the Inverse Fast Fourier Transform (IFFT) is obvious since they are the means allowing the passage from the time domain to the frequency domain and from the time domain to the frequency domain. The conversion complexity is much less than the complexity of the time domain algorithm when the filter is long enough in a way that its length is equal or bigger than the crossover point where the computational cost of the frequency domain filter is less than the time domain one.

2.3 Applications of adaptive filters

Adaptive filters are very useful in case of transmitted signals with the minimum amount of information, that is why they are widely exploited in telecommunications, control systems, radar systems…

Due to its flexibility and easy implementation, adaptive filters design and adaptive algorithms development are defined by the applications themselves. The most important applications for the adaptive filters are described in the following sections.

2.3.1 System Identification

System identification consists in modeling an unknown system and estimating its parameters by applying an experimental data. This technique is very efficient in case of dynamic system

with complex behavior, since it is not easy to model it and consequently to control it. So, we refer to collect some experimental data about the system responses by applying some specified excitations.

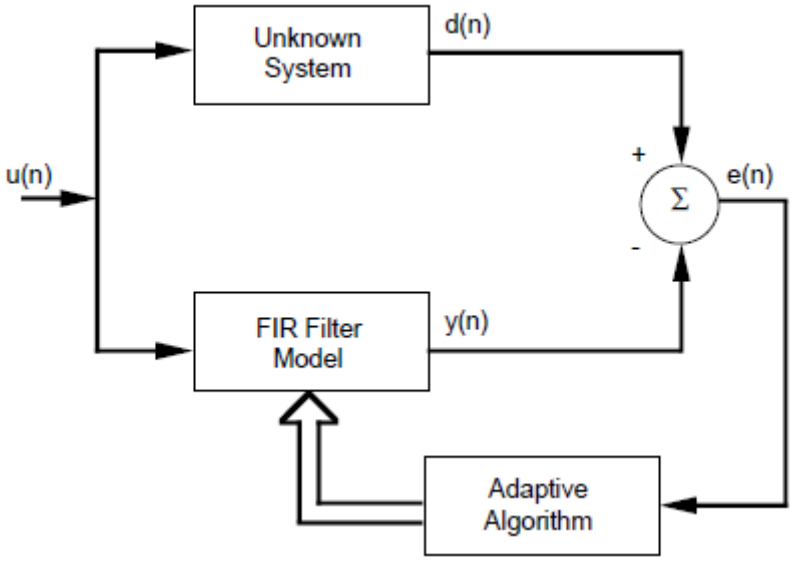

Figure 2.1: System identification using adaptive filter [22]

Figure 2.1 shows a block diagram of the system identification model. We have as a model for the unknown system an FIR filter with adjustable coefficients.

We give the unknown system and the FIR filter model the same input u(n). An estimation error e(n) is produced by doing the difference between the outputs y(n) and d(n) of the adaptive filter which is the system model and the unknown system respectively.

Now, using the estimation error e(n), the adaptive algorithm is updating the tap weights of the filter. This process is repeated until the estimation error e(n) reaches a given limit and then the unknown system response is deduced from the adaptive filter one.

2.3.2 Adaptive equalization

In telecommunications, transmitting data through wide channels requires the use of modems and accordingly the use of equalizers. Equalization is widely using the adaptive algorithms; we are talking about adaptive equalization. The purpose from using an adaptive equalizer is to compensate for the distortion and eliminate the inter-symbol interference caused by the channel noise.

Initially, a desired signal d(n) is defined for the adaptive equalizer. Then, the error estimation is calculated by making the difference between the equalizer output y(n) and the desired signal. The produced error is used to update the coefficients of the equalizer to reach its optimum values. The received signal is obtained by applying the equalizer output y(n) to a decision device.

2.3.3 Echo Cancellation

In telephony, full-duplex operation, transmitting and receiving channels from a two-wire telephone line are provided by a device called “hybrid”. Figure 2.2 shows the layout of the system.

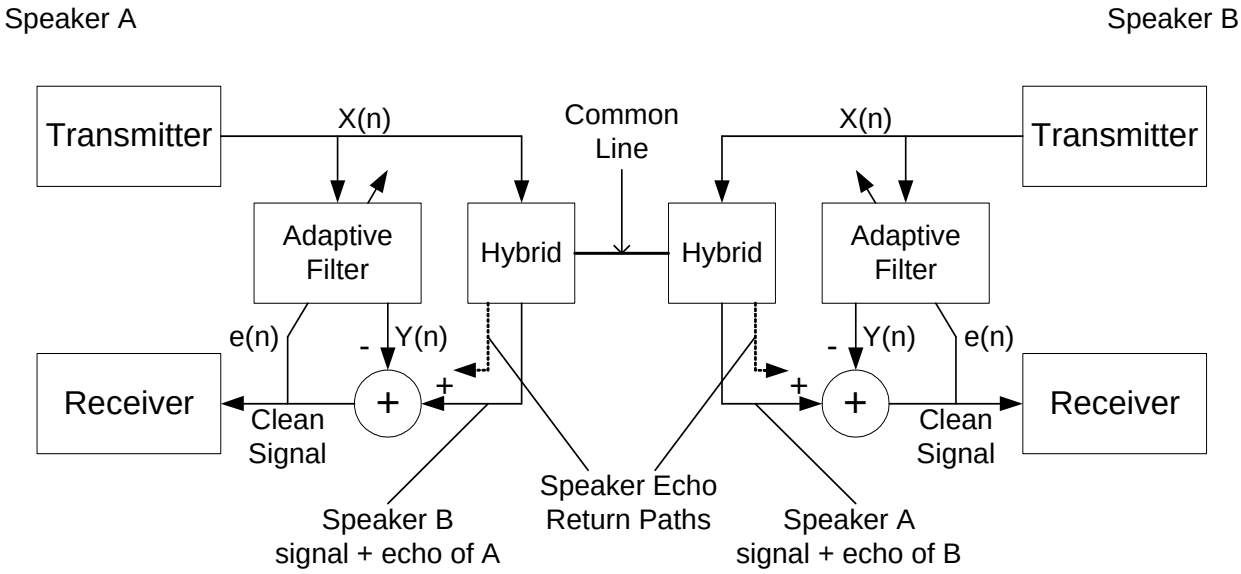

Figure 2.2: Echo cancellation using adaptive filter [22]

An echo is generated because of the impedance mismatch between the hybrid and the telephone channel. I can be eliminated by using adaptive echo cancellers, based on adaptive algorithms, in the network. Updating the optimum coefficients of the echo canceller depends on calculating the error signal resulting from the difference between the estimate and the received signal.

2.3.4 Linear Predictive Coding of Speech Signals

The speech signals can be digitally represented using the method of linear predictive coding (LPC). In LPC, the source vocal tract is modeled as a linear all-pole filter whose parameters are determined adaptively from speech samples by means of linear prediction. The speech samples u(n) are, in this case, the desired response, while u(n-1) forms the inputs to the adaptive FIR filter known as a prediction error filter. The error signal between u(n) and the output of the FIR filter, y(n), is then minimized in the least-squares sense to estimate the model parameters. At the receiver, using the model parameters and the error signal, the speech signal is synthesized.

2.3.5 Array Processing

Adaptive antenna arrays are behaving the same way as adaptive filters in terms of the processing techniques. Those arrays are generating a parallel set of signal samples by the mean of the spatial separation between the antenna elements instead of using the time-delayed or partly processed versions of a one-dimensional input signal. The adaptive antenna arrays are applied for bearing estimation and adaptive beam-forming.

2.4 Conclusion

The adaptive equalization is one of the adaptive filtering applications. They are used to eliminate the inter-symbol interference caused by the noise in the transmission channel. The next chapter will deal with the equalization in both frequency and time domain and it will compare between their performances.

Chapter 3

Equalization

3.1 Introduction

As it is mentioned in the previous chapter, equalization is one of the applications of adaptive filtering. Its role consists in eliminating the ISI caused by the noise in the transmission environment. This chapter will present in details the time domain and the frequency domain equalizers by introducing their structures as well as the algorithms they use.

3.2 Transmission chain presentation

The whole transmission chain is represented by the figure below.

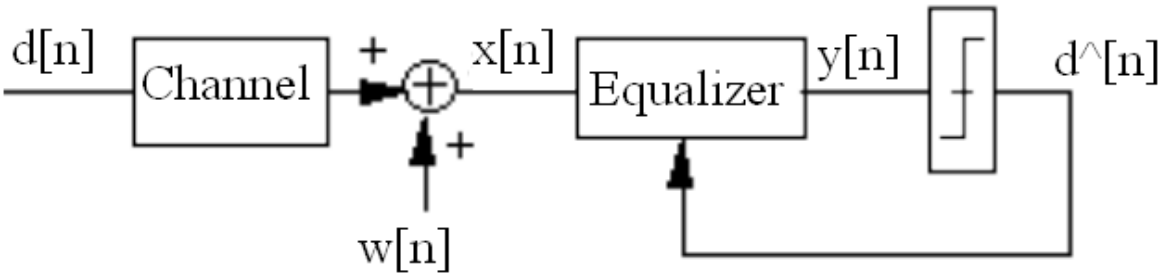

Figure 3.1: Transmission chain [24]

This communication chain is composed of a channel (transmission environment), a modulator, a demodulator and an equalizer.

The channel is noisy by an Additive White Gaussian Noise (AWGN).

The equalizer has to eliminate the inter-symbol interference presented in the received signal x[n] to get in its output the transmitted signal d[n].

3.3 Time domain equalization

To get an output matching as much as possible the desired response, uncountable adaptive

algorithms are used to regulate the filter coefficients. One of the most used algorithms is the LMS algorithm. Its success is explained by its efficiency especially in terms of storage requirements and computational complexity. Adjusting filter coefficients using the LMS algorithm means that these coefficients are updated after every sample.

The following scheme is explaining clearly the notion of the LMS algorithm using a coherent mathematical equation:

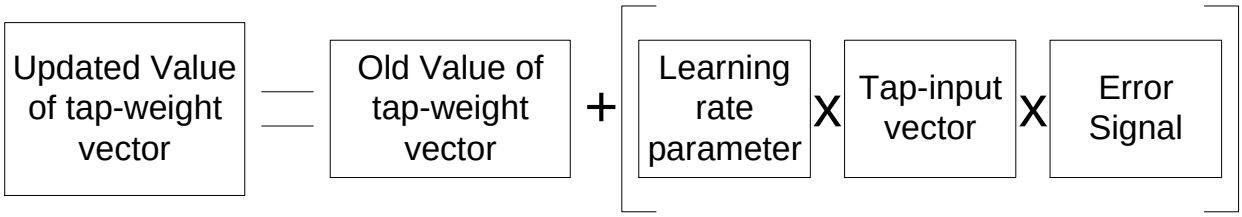

Figure 3.2: The LMS Algorithm [22]

LMS algorithm is very simple and very easy to implement that's why it is adopted by most of real time systems.

The LMS algorithm is described by the following steps:

1. Choosing the desired response. The filter coefficient must be set to zero.

$$h\,(i)\;=\;0, i\;=\;1, 2, 3, \dots, N \qquad 3.0$$

Steps (2) to (4) are repeated every sampling instant (k):

2. The output of the adaptive equalizer y(k) is get by multiplying element by element the input vector by the filter coefficients vector then the sum of all multiplications is done.

$$y(k) = \sum_{i=0}^{N-1} h\,(i)\,x\,(i) \qquad 3.1$$

3. Next, the error is defined by doing the subtraction between the desired signal and the equalizer output y(k) just before updating the coefficients of the equalizer.

$$e\,(k) = y\,(k) - \,d\,(k) \qquad 3.2$$

4. Finally, the equalizer coefficients must be updated by multiplying the error calculated in the previous step by the step-size μ then by the equalizer input x(k). After that, the result is added to the previous value of the equalizer coefficients.

$$h\,(k+1) = h\,(k) + 2\,\mu\,e\,(k)x\,(k) \qquad 3.3$$

Updating the equalizer coefficients using the LMS algorithm requires that the parameters are defined accurately. The step size parameter μ and the number of filter coefficients N are the most important parameters that ensure the convergence of the equalizer to an output matching the best to the desired signal.

Changing the values of μ and N has an impact on the equalizer behavior, that's why we were testing the equalizer with a lot of values of the step size parameter to choose the best one which gives the optimum result. But, concerning the filter length we were limited by the hardware resources that's why we adopted for this project small length filters.

Although it is widely used due its efficiency, the LMS algorithm suffers from one major problem. It mainly deals with its complexity since its computational cost augments exponentially when the length of the filter augments. This problem is actually generated because the LMS algorithm is implemented in the time domain. That's why another version of the LMS algorithm was found to compensate that defect which is the BLMS algorithm or Block LMS algorithm. But, the efficiency of that algorithm in terms of computational cost can not be compared to implementing the adaptive equalizer in the frequency domain.

3.4 Frequency domain equalization

Data alteration between the frequency domain and the time domain requires the use of some tools ensuring the preservation of data during this transition. The most useful tool enabling representing a signal in the frequency domain is the discrete transform that helps to decrease the computational complexity related to signal processing just like convolution. One of the most efficient transforms allowing the transition between the two domains is the Fast Fourier Transform (FFT). It is the most used transform because of its advantages comparing to the other transforms:

- It is efficient
- It can represent data for even short data lengths
- It is accurate in representing data

3.4.1 Discrete Fourier Transform

The Discrete Fourier Transform is one of the structures of the Fourier Transforms allowing to the conversion of a discrete signal from the time domain to the frequency domain.

The expression below represents the equation calculating the Discrete Fourier Transform.

$$X\,(k) = \sum_{n=0}^{N-1} \mathrm{x}\,(\mathrm{nT})\mathrm{e}^{-\mathrm{jkwnT}} \qquad 3.4$$

It is obvious that this expression is similar to the equation allowing calculating the Fourier Transform for continuous signals in time domain.

The DFT has two important properties which are:

- **Symmetry**

  That means that the two elements *X(k)* and *X(k+N)* resulting by applying the Discrete Fourier Transform on the signal x are the same. As a result, it can be deduced that there is a periodicity with period *N*.

- **Convolution**

  The circular convolution as well as the linear convolution can be deduced by the use of the Discrete Fourier Transform. The time convolution theorem declares that a convolution in time domain is transformed into a simple multiplication in the frequency domain. This is summarized by the following expression.

$$x(n) = x_1\,(\mathrm{m}) * x_2\,(\mathrm{n}) = \mathrm{F}^{-1}\{X_1\,(\mathrm{k})\,X_2\,(\mathrm{k})\} \qquad 3.5$$

  where *x*, x_1 and x_2 are finite periodic signals having the length and * expresses circular convolution F^{-1} expresses the Inverse Discrete Fourier Transform.

  This property of the Discrete Fourier Transform has great importance for this project because instead of using convolution in time domain which can increase the computational cost, we can use the multiplication in the frequency domain and as a result we can get the same results with less complexity by just applying the DFT.

3.4.2 The Fast Fourier Transform

One of the most useful and efficient algorithms used to put in work the Discrete Fourier Transform is the Fast Fourier Transform (FFT). Its role consists in transforming a vector *x* expressed in time domain to its equivalent *X* in the frequency domain. To reduce the computational cost and to make the algorithm more efficient, The FFT is based on the

principle of the built in redundancy used by the DFT. Besides the FFT permitting the transformation of a signal from the time domain to the frequency domain, another algorithm allowing the conversion of a signal in the other direction that means from the frequency domain to the time domain; it is the Inverse Fourier Transform (IFFT).

The computational complexity for the Discrete Fourier Transform is equal to N2 complex multiplies, but when we are talking about the FFT this number is decreased and its complexity is equal to (N/2) log2 (2N) complex multiplies + N complex adds. To get much better results, the block length N must be an integer power of 2. By doing this, we enhance the performance of the FFT algorithm. The FFT block length must be the same as the input signal block length.

3.4.3 The Overlap-Save Algorithm

As it is mentioned previously, the convolution property is the key to pass from the time domain to the frequency domain. The most important algorithms implementing convolution are the Overlap-Add and the Overlap-Save. These two algorithms are based on the convolution theorem stating that a convolution in the time domain is equivalent to a multiplication in the frequency domain. I terms of complexity, the Overlap-Save algorithm is more efficient than Overlap-Add algorithm. The earlier algorithm is overlapping the input blocks and to maximize its efficiency a 50% overlap is required; that means that the current input values are composed of a concatenation of the current input block and the previous one. By applying the IFFT, the last N samples must be discarded because of the circular convolution and then we concatenate the output samples to the others to finally form the Overlap-Save algorithm output. The figure below expresses the notion of the Overlap-Save algorithm.

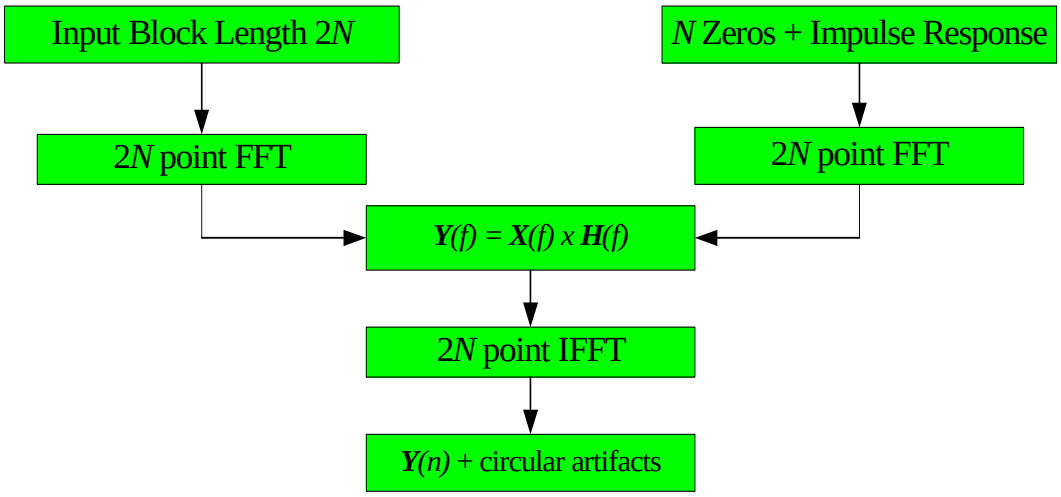

Figure 3.3: The overlap-save algorithm scheme [22]

The length of the input block is 2*N*.

1. The impulse response must have the same length as the FFT block that's why *N* are added to it.

2. The FFT is applied to the impulse response.

$$W\ (k) = \ FFT\ \{h(k)\} \qquad 3.6$$

3. The FFT is applied to the input block. Since a 50% overlap is adopted then every current block is concatenated to the previous block. For the first block, *N* zeros must precede it.

$$X(k) = FFT\{x(k)\} \qquad 3.7$$

4. The product of the FFT results in the steps 2 and 3 is now calculated. This multiplication must be done element by element. This multiplication in the frequency domain is equivalent to the convolution in the time domain.

$$Y(k) = X(k).W(k) \qquad 3.8$$

5. To pass to the time domain, the IFFT must be applied to *Y(k)*.

$$y(n) = IFFT\{Y(k)\} \qquad 3.9$$

6. The last *N* samples of *y(n)* are discarded. However, the first *N* samples are added to the previous output samples

7. The input block is updated and the steps 3 to 7 are repeated.

3.4.4 The Fast LMS Algorithm

For this master project, implementing an adaptive frequency domain equalizer [22, 23] is based on an adaptive frequency domain algorithm called Fast LMS. This version of Fast LMS algorithm is based on the overlap-save convolution algorithm. Updating the equalizer coefficients in the frequency domain using the Fast LMS algorithm is similar to the process in the time domain using the LMS algorithm. One difference between the two procedures in terms of updating coefficients consists in that the Fast LMS algorithm updates the coefficients block by block not sample by sample. The following figure illustrates the principle of the Fast LMS algorithm.

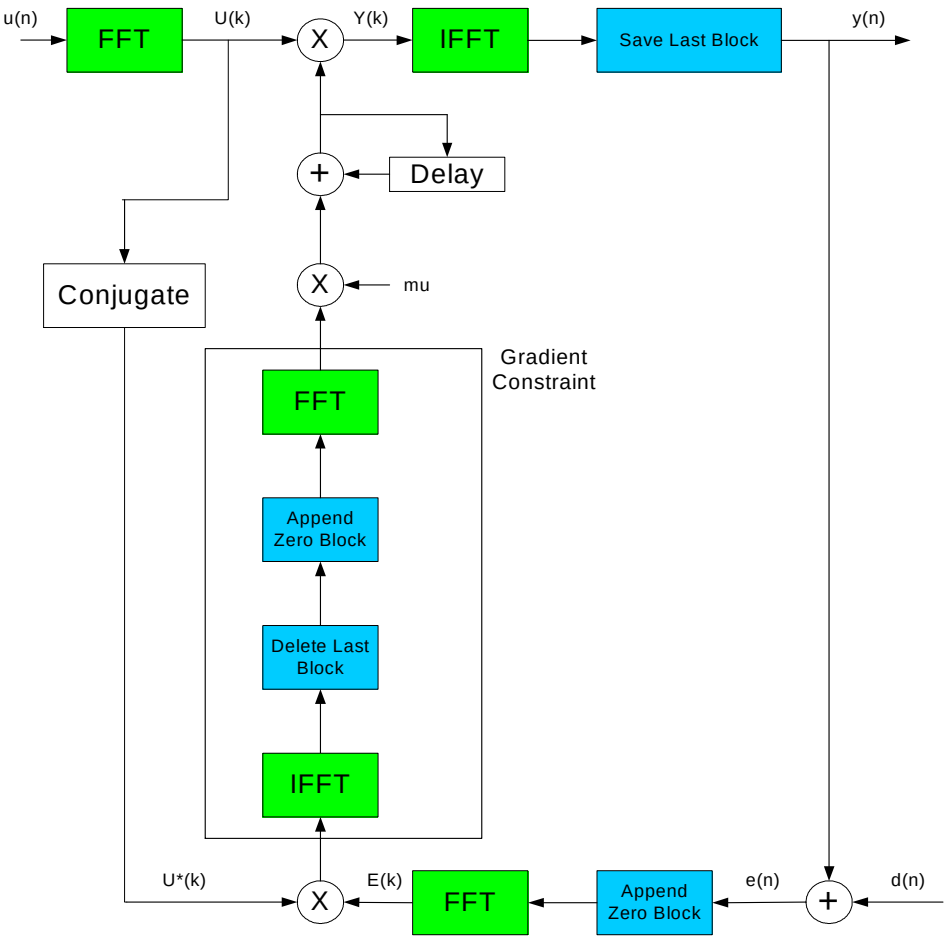

Figure 3.4: the Fast LMS algorithm [22]

The 50% overlap is applied to the input blocks of size *2N* and the filter coefficients are set to zero.

The Fast LMS algorithm is described as follows:

1. The FFT is applied to a 2N input block got from the input signal.

$$U(k) = FFT\{u(n)\} \quad 3.9$$

2. Multiplying ***U(k)*** by the filter coefficients gives the equalizer output in the frequency domain. The coefficients are adjusted before.

$$Y(k) = U(k).W(k) \quad 3.10$$

Then an IFFT is applied to Y(K) to get the result in the time domain.

$$y(n) = IFFT\{Y(k)\} \quad 3.11$$

Because of the circular convolution, only the last N samples must be kept and they will represent the output of the equalizer.

$$y(n) = y(N + 1 \rightarrow 2N) \quad 3.12$$

3. Next, a subtraction between the desired signal and the current equalizer output must be calculated to calculate the error signal.

$$e(n) = d(n) - y(n) \quad 3.13$$

Where *e(n)* is the error and *d(n)* is the desired signal.

After that the error must be transformed to the frequency domain, that's why an FFT is applied to *e(n)* after adding *N* zeros to its start.

$$E(k) = FFT\{zeros, e(n)\} \quad 3.14$$

4. After calculating the conjugate of the *U(k)*, it is multiplied by the error in the frequency domain. Then, an IFFT is applied to the result. Only the first *N* samples of this result are kept because of the circular convolution.

$$g(n) = IFFT\{E(k).U'(k)\} \quad 3.15$$

$$g(n) = g(1 \rightarrow N) \quad 3.16$$

5. A 2*N* point FFT is now applied on g(n) after adding N zeros to its end then the result is multiplied by the step size parameter μ.

$$g(n) = g(n) followed\ by\ N\ zeros \quad 3.17$$

$$W_1(\mathrm{k}) = \mu . FFT\{g(n)\} \quad 3.18$$

The obtained result consists in the update factor for the equalizer coefficients, that's why it is added to the previous value of the filter coefficients.

$$W_1(\mathrm{k}+1) = \mathrm{W}(\mathrm{k}) + W_1(\mathrm{k}) \quad 3.19$$

6. The updated equalizer coefficients are set and ready to be used with the next input block. From one iteration to another the error is decreasing since the coefficient are updated progressively.

3.5 Conclusion

In this chapter, a theoretical background about frequency domain and time domain equalization was presented by describing the algorithms used by both equalizers as well as their structures and the techniques they use such as FFT and IFFT. The preference of the frequency domain equalizer was also explained in this section. In the next chapter, an implementation of the two equalizers will be discussed and a comparison between the theoretical and practical results will be done.

Chapter 4

Implementation

4.1 Introduction:

The theoretical background was introduced by the previous chapter. In this chapter, the practical work will be discussed. First, a tool exploration must be done to define the appropriate method that allows achieving this project objective. Then, the FPGA implementation flow will be described to finally comment the obtained results.

4.2 Tools exploration:

4.2.1 Introduction:

The evolution known by the high-level tools translates the increase of the FPGA technology adoption, since the reprogrammable silicon delivers a lot of benefits to engineers, researchers and scientists of all domains.

4.2.2 Catapult C Synthesis:

Catapult C Synthesis is a *Mentor Graphics* [21] product; it produces an RTL implementations from abstract specifications written in C, C++ or SystemC. The Catapult C flow consists in modeling, synthesizing, and verifying complex ASICs and FPGAs architectures as it is shown in the figure 4.1.

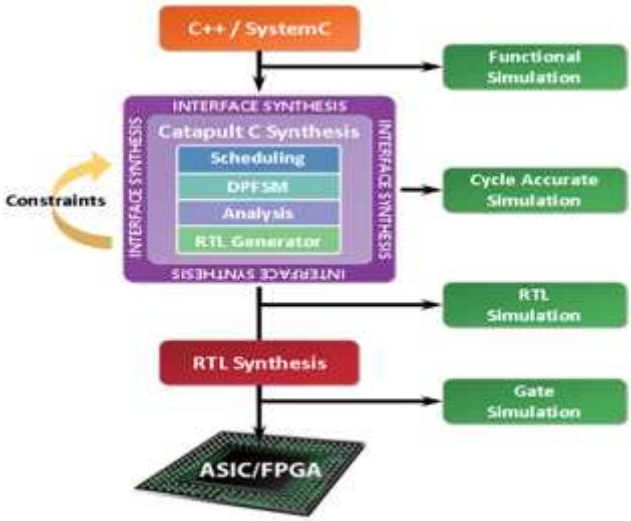

Figure 4.1: The Catapult design flow [21]

4.2.3 AutoPilot FPGA:

AutoPilot FPGA is *AutoESL*'s [19] high level synthesis tool for FPGAs. *AutoPilot FPGA* takes a complex algorithm in the form of C, C++ or SystemC description or a combination of these languages and automatically generates an equivalent RTL that is ready for synthesis into an FPGA device. *AutoPilot FPGA* supports both *Xilinx* and *Altera* devices.

For *Xilinx*:

- ✓ Comprehensive Device Support

  Virtex-6, Virtex-5, Virtex-4, Virtex-II Pro, Virtex-II, Spartan-6, Spartan-3
- ✓ Automatically generates all files required for FPGA implementation using Xilinx XST, ISE, EDK, and Synplify tools
- ✓ Simulation and debugging flow works with ModelSim and Aldec simulators

For *Altera :*

- ✓ Comprehensive Device Support

  Stratix IV, Stratix III, Stratix II, Stratix, Cyclone III

- ✓ Automatically generates all files required for FPGA implementation using Altera Quartus II, SOPC Builder, and Synplify tools
- ✓ Simulation and debugging flow works with ModelSim and Aldec simulators

4.2.4 PICO for FPGA:

PICO for FPGA has two products allowing an FPGA implementation from a C code specification; *PICO Express* and *Pico Extreme*. They are a *Synphora* [20] product, they take a C algorithm and a set of design requirements (clock frequency, throughput target and technology file) and create a series of implementation models (RTL, SystemC). Figure 2 summarizes all the steps in the *PICO Extreme* design flow.

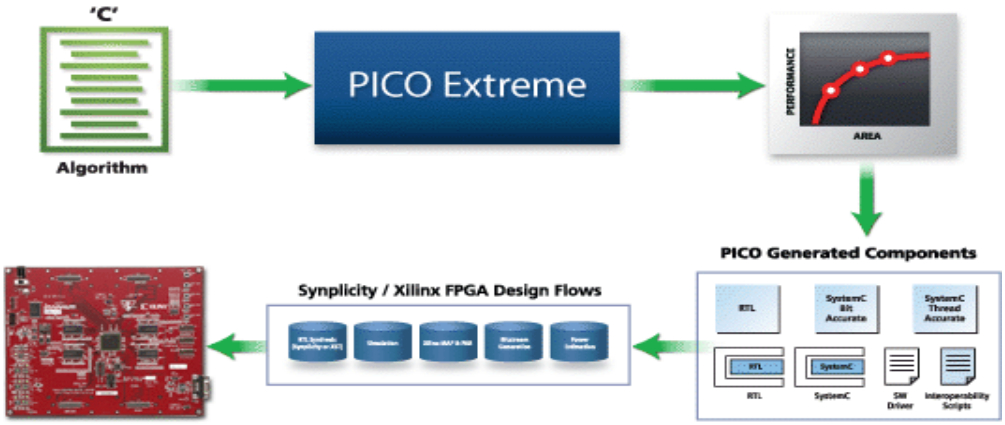

Figure 4.2: Pico Extreme design flow [20]

4.2.5 System Generator for DSP:

System Generator is a DSP design tool from Xilinx [18] that enables the use of The Mathworks model-based design environment Simulink for FPGA design. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file.

System Generator provides a system integration platform for the design of DSP FPGAs that allows the RTL, Simulink, MATLAB and C/C++ components of a DSP system to come

together in a single simulation and implementation environment. System Generator supports a black box block that allows RTL to be imported into Simulink and co-simulated with either ModelSim or Xilinx ISE Simulator. Figure 3 explains the operation principle of System Generator.

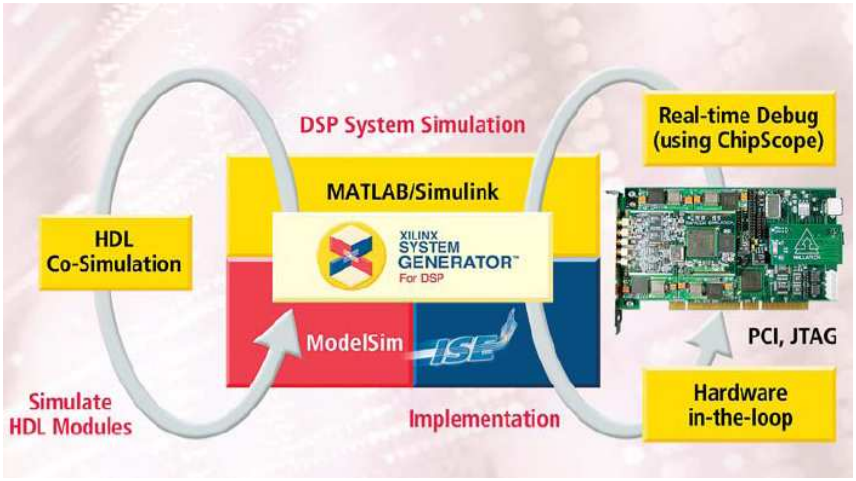

Figure 4.3: System Generator design flow [18]

4.2.6 Conclusion:

Regarding to the different tools and its characteristics, we adopt for this project the system generator for DSP tool. In one hand, it is possible to implement the design described with similink directly in FPGA, in the other hand; we can download it for free from the Xilinx website.

4.3 Implementation methodology:

The specific objectives for this master project are the following:

1. Simulation of the frequency domain equalizer using Matlab.
2. Defining the required tools allowing the implementation of the frequency domain equalizer on FPGA.
3. FPGA implementation of the frequency domain equalizer.
4. Performance analysis.

As it is mentioned in the section 4.2, System Generator for DSP was chosen as the tool allowing the FPGA implementation of the frequency domain equalizer.

System Generator is created to implement DSP applications on FPGA using the Mathworks model-based design tool Simulink. This tool is very easy to work with since it doesn't require a previous knowledge of hardware design methodologies. Designing using System Generator only needs a DSP simulink modeling environment but based on a specific block set from Xilinx. All of the flow of the FPGA implementation steps is done automatically starting form synthesis, passing by place and route and arriving to generating the programming file.

The first step in the design flow using System Generator is describing the specification using the Similink block sets. Then, System Generator defines the design hardware devices using the specific DSP Xilinx block set. After that, Xilinx Core Generator generates an optimized netlitst for the DSP blocks. The programming file, the bitstream, is automatically generated by the System Generator. This latter can also create a testbench based on the vectors used in the simulink specification and which can be run on Modelsim or Xilinx ISE Simulator. Figure 4.4 summarizes the steps of the System Generator design flow.

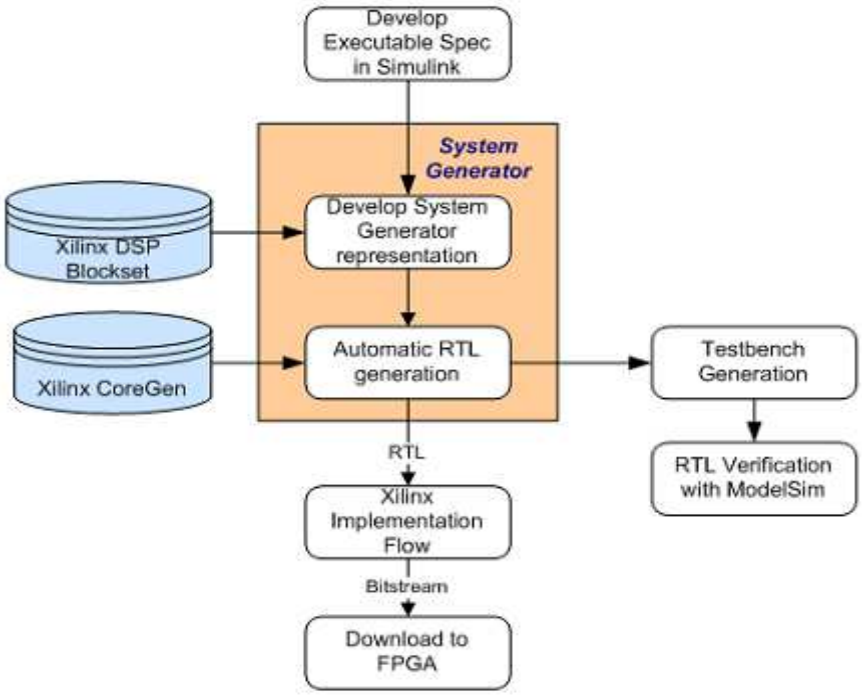

Figure 4.4: System Generator design flow [20]

For this project, the FPGA used for implementation is a Spartan 3 with one million gates.

4.4 Time domain equalizer implementation:

To test its functionality, the time domain equalizer was simulated using Matlab. To get reliable results, the equalizer must be tested in a well defined transmission environment; first we need a symbol generator which generates, in our case, random integer symbols with the range [0..15]. Second, the symbols go through a 16-QAM modulator and after that through the transmission channel. The channel used for matlab simulation is a Proakis A [4] channel. Its coefficients are [0.04, - 0.05, 0.07, - 0.21, -0.5, 0.72, 0.36, 0, 0.21, 0.03, 0.07]. The channel is also noisy by an Additive White Gaussian Noise (AWGN). The equalizer is, then, required to eliminate ISI in the transmitted signal after going through that channel. Finally, the output of the equalizer is demodulated by a 16-QAM demodulator. Figure 4.5 shows the transmission chain components.

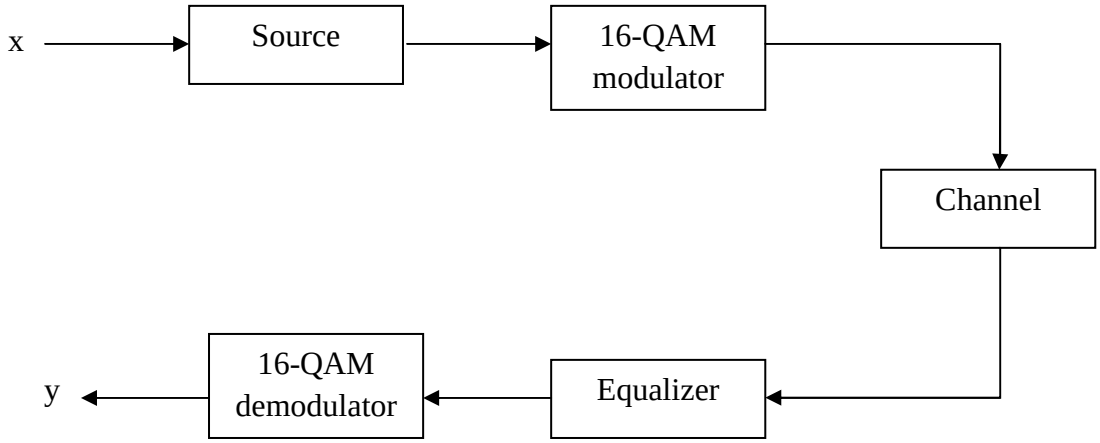

Figure 4.5: Transmission chain scheme

Using matlab, we simulate two types of time domain equalizers, one based on the LMS algorithm and the other one is based on the BLMS algorithm. The difference between the two equalizers is that the first is updating its coefficients after each sample and the second is updating it coefficients after an input block of samples.

The simulation parameters are set as following:

- SNR = 40 dB
- Transmitted symbols = 10 000
- μ = 0.002

The following figure presents the constellation of the signal that enters to the equalizer as well as the constellation of the equalizer output signal.

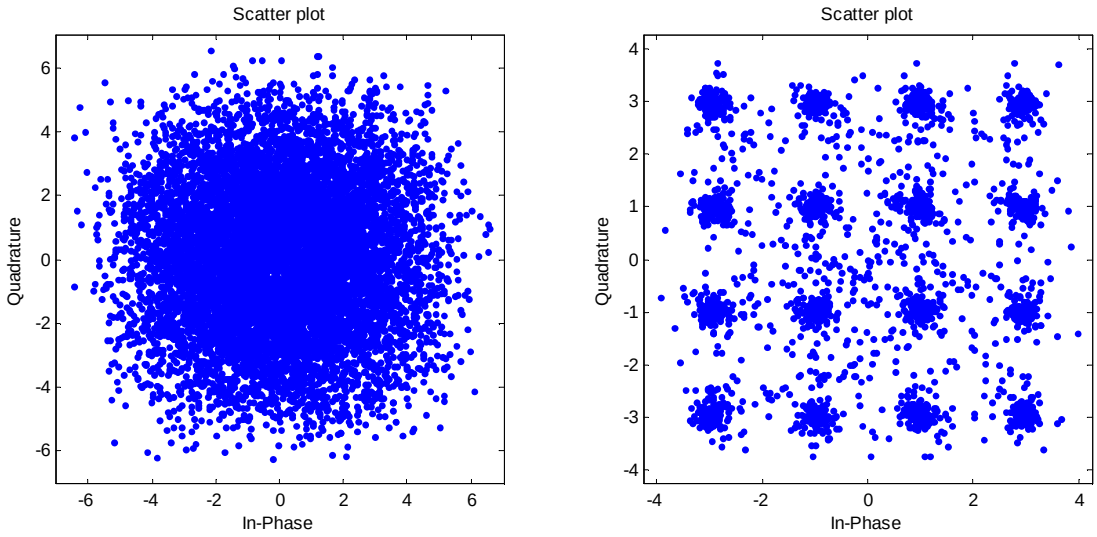

Figure 4.6: (a) Signal constellation before LMS-based equalization, (b) Signal constellation after LMS-based equalization

As it is shown in figure 4.6.a, the signal entering to the equalizer is very noisy because it passes through an AWGN channel. But after going through the equalizer, we obtain a 16-QAM constellation, presented by the figure 4.6.b, which means that the time domain equalizer eliminates the ISI and gives a signal that can be easily demodulated as an output.

The error estimation is explicit in the figure 4.7.

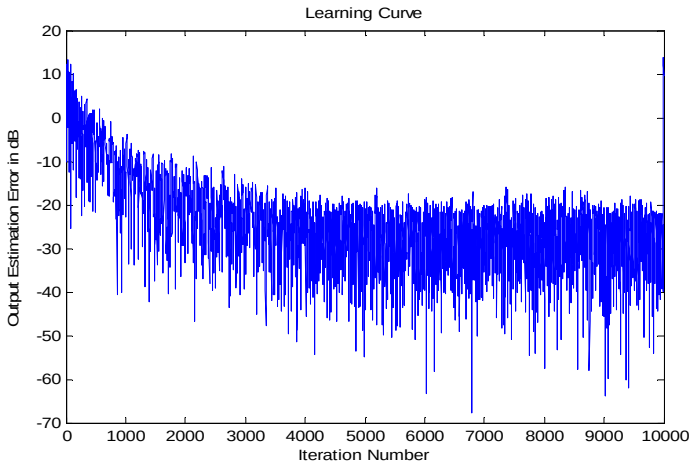

Figure 4.7: LMS equalizer error estimation

The figure above shows that the error is very high in the beginning since the equalizer is still updating its coefficients. It reaches a stable status after almost 2000 symbols where the value of the error rate is equal to –25 dB such that the SNR is equal to 40 dB.

Using the LMS algorithm as the adaptive algorithm for the equalizer gives an error rate equal to 1.91% which means that by sending 10000 symbols, we only loose 191 symbols.

We also simulate a BLMS based equalizer using Matlab using the same LMS equalizer parameters. We get the constellations of the equalizer input and output signal which are represented by the figure below.

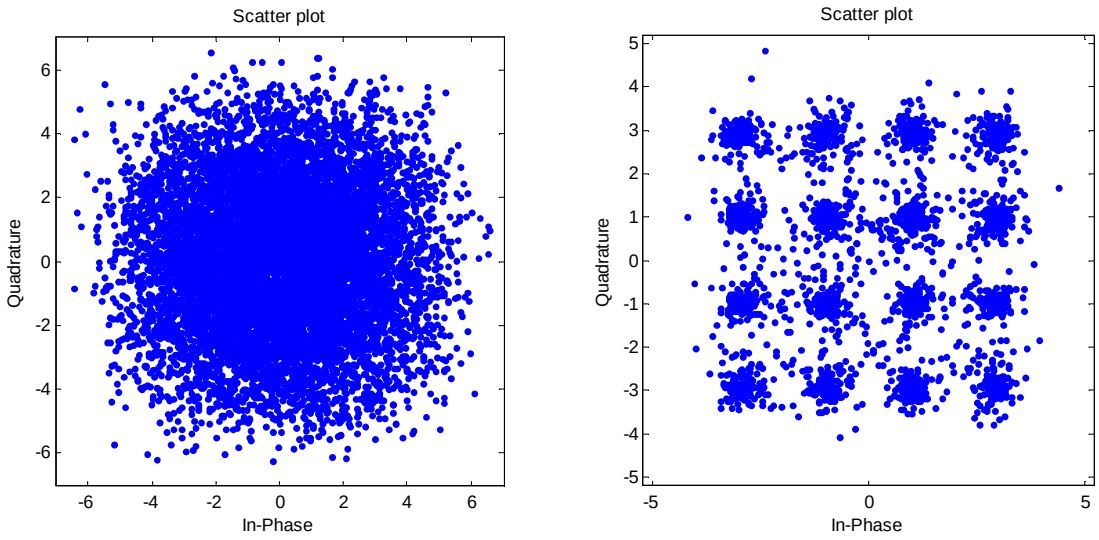

Figure 4.8: (a) Signal constellation before BLMS-based equalization, (b) Signal constellation after BLMS-based equalization

The figure 4.8 shows that the BLMS-based equalizer gives the same results as the LMS-based equalizer. The BLMS equalizer is a time domain equalizer that succeeded to eliminates the ISI and gives almost the same performance as the LMS equalizer since the error rate given by the BLMS equalizer is 2.03% that means that only 203 symbols are lost from the whole 10000 sent symbols. The only difference between the two time domain equalizers is that the LMS one is updating its coefficients after every sample and the BLMS one is updating its coefficients after every block of samples. The error estimation is given by the following curve.

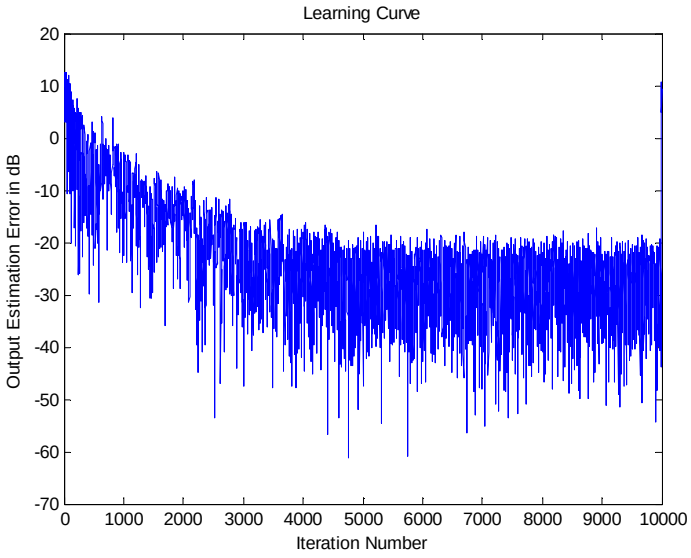

Figure 4.9: BLMS equalizer error estimation

From this curve, we can deduce that the error stable status for the BLMS equalizer is almost the same for the LMS one; it is equal to -25 dB. But, the BLMS algorithm converges slower than the LMS algorithm. It requires 3000 symbols to converge.

Now, as a first step of the FPGA implementation flow, the LMS-based equalizer is described using the blocks of the simulink environment. The whole transmission chain is represented using the simulink block sets as it is shown in figure 4.10.

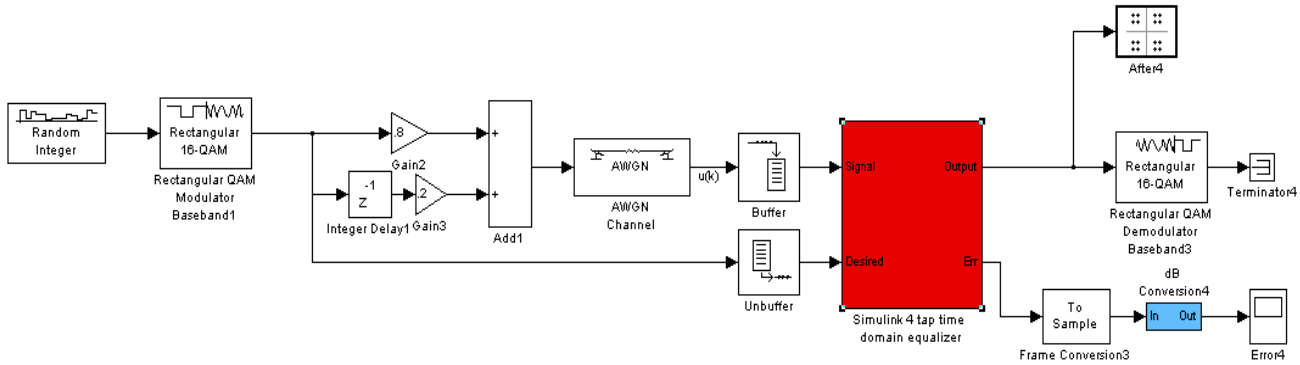

Figure 4.10: Transmission chain with a 4-tap LMS equalizer in Simulink

The figure above shows all the components of the transmission chain. First, we have a random integer generator. It generates random integer symbols belonging to the interval [0..15]. Then, a 16-QAM modulator is installed. After that, the signal must go through an AWGN multipath

channel, in our case, it is a 2-path channel after which a 4-tap time domain equalizer based on the LMS algorithm is set up to eliminate the ISI. Finally, we add a 16-QAM demodulator to get the received signal.

The architecture of the LMS-based equalizer is given by the figure below.

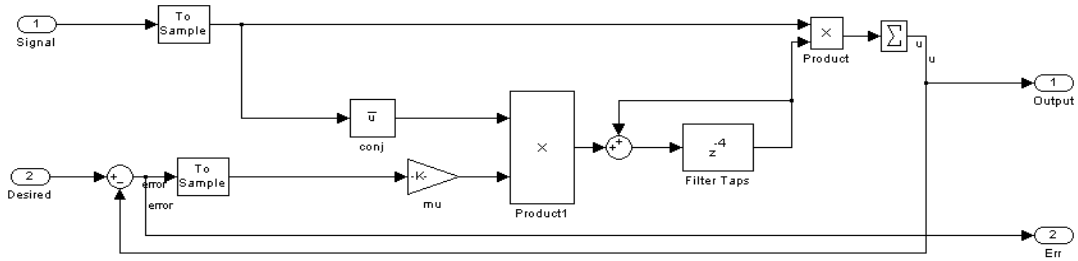

Figure 4.11: LMS equalizer in Simulink

The equalizer has as input the noisy signal coming from the channel as well as the desired signal which is in our case the output signal of the modulator. The output signals of the equalizer are the equalized signal and the error estimation. From the architecture of the equalizer, we can easily notice that the equalizer is based on the LMS algorithm as it is described in section 3.

Next, a simulation must be done to test the functionality of the time domain equalizer. That's why the simulation parameters were set as following:

- SNR = 40 dB
- $\mu = 0.002$
- Filter taps = 4

The figure 4.12 shows the constellation of the equalizer output signal. We can deduce from this constellation the efficiency of the equalizer.

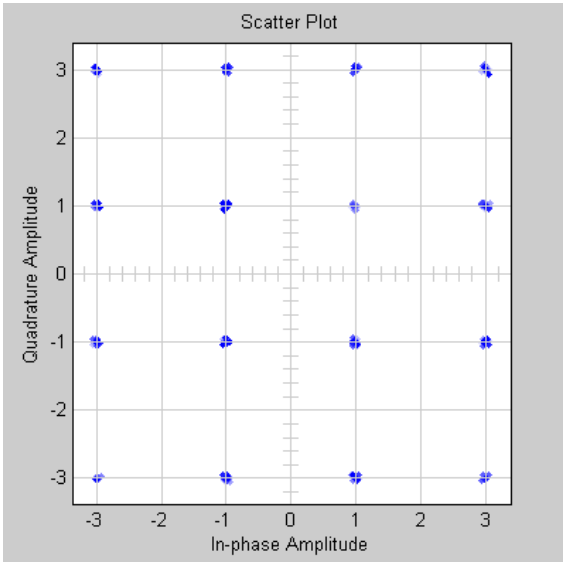

Figure 4.12: Signal constellation after time domain equalization in Simulink

In terms of error estimation, it is shown that the 4-tap time domain equalizer converges after almost 800 symbols to reach the value of -35 dB. The error estimation curve is represented by the figure 4.13 below.

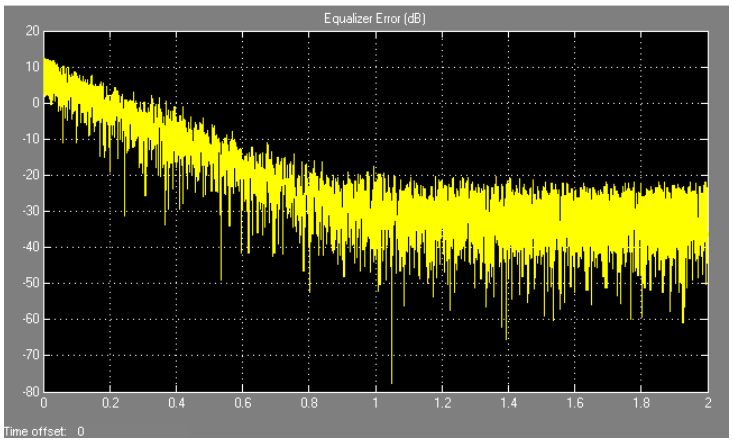

Figure 4.13: Error estimation

To get a synthesizable version of the time domain equalizer, it must be described using the specific Xilinx block set in the simulink environment. The design must be put between two specific blocks called Gateway-in and Gateway-out. Their role consists in limiting the design that will be implemented in the FPGA from the other simulink blocks. The figure below shows the 4-tap time domain equalizer described using the Xilinx block set.

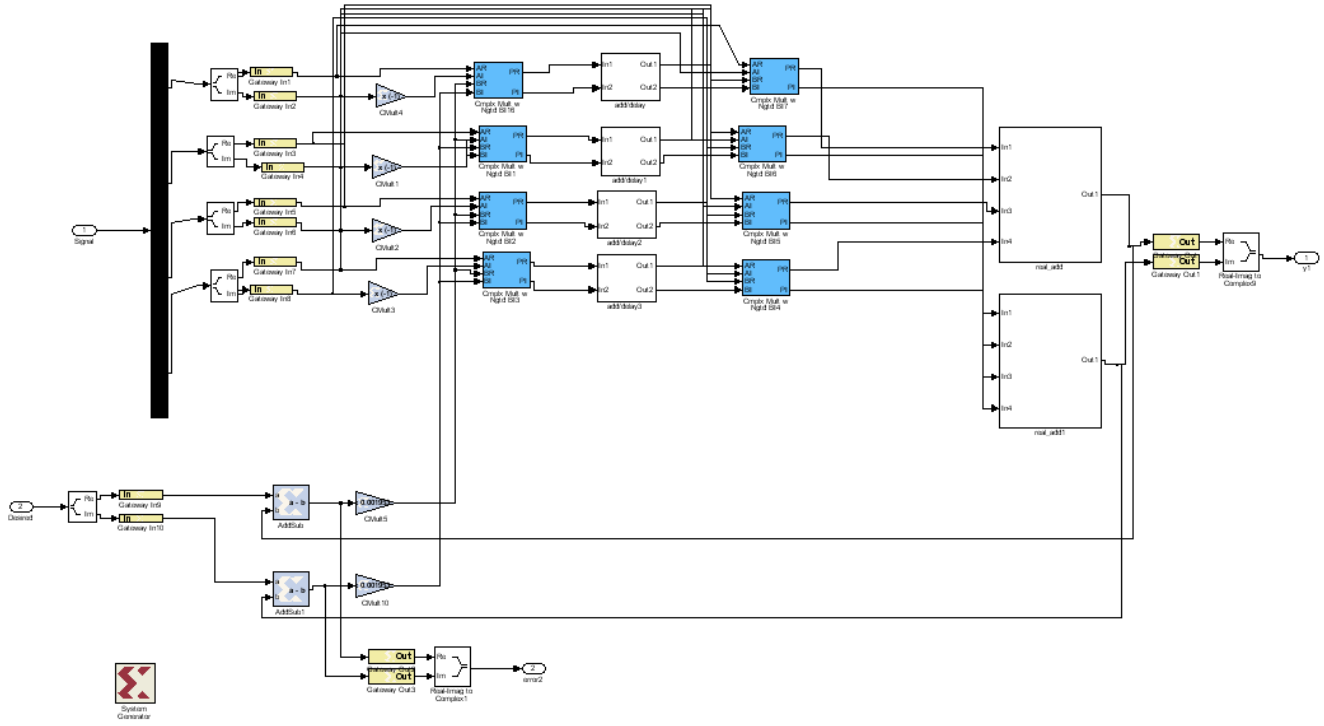

Figure 4.14: A System Generator description of the 4-tap time domain equalizer

The yellow blocks represent the Gateway-in and the Gateway-out blocks. The other blocks are elementary blocks from the Xilinx block set which are essentially multipliers and adders and also some sub-systems doing complex addition and complex multiplication.

To make sure of its functionality, we did a simulation for this design with the same parameters used with the standard simulink design. The signal constellations as well as the error estimation curve are given by the figures 4.15 and 4.16 respectively.

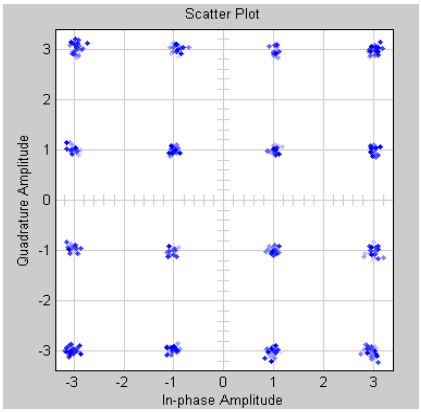

Figure 4.15: Signal constellation after time domain equalization in Sys Gen

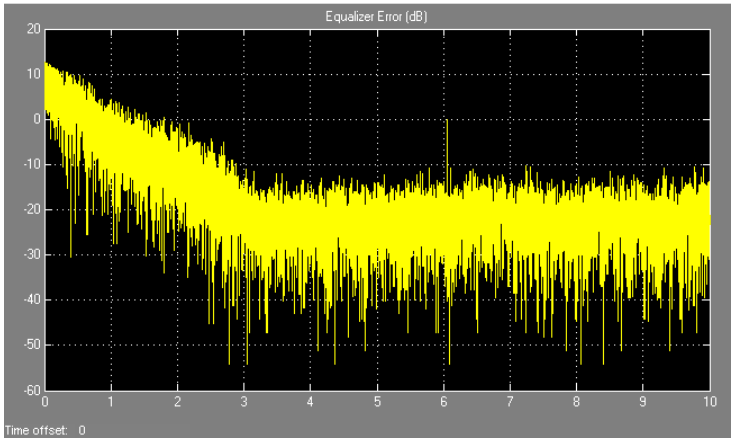

Figure 4.16: Error estimation curve

Comparing the results given by the System Generator description and the standard Simulink blocks description of the 4-tap time domain equalizer, we can deduce that the Simulink design gives better results than the System generator one. In simulink, the symbols are described using the floating point but in System Generator only the fixed point is used. That's why the simulink design gives more accurate results than the System Generator design.

After designing the time domain model, we proceed to the FPGA implementation which is automatically done by the System Generator for DSP tool. It uses the Xilinx ISE 11.1 version to do all the flow. The design will be implemented in the Spartan 3 FPGA board with one million gates. The table 4.1 gives and clear idea about the FPGA logic blocks consumption by the design.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	144	15,360	1%	
Number of 4 input LUTs	12,230	15,360	79%	
Number of occupied Slices	6,278	7,680	81%	
Number of Slices containing only related logic	6,278	6,278	100%	
Number of Slices containing unrelated logic	0	6,278	0%	
Total Number of 4 input LUTs	12,240	15,360	79%	
Number used as logic	12,086			
Number used as a route-thru	10			
Number used as Shift registers	144			
Number of bonded IOBs	233	391	59%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	2.39			

Table 4.1: Time domain equalizer synthesis report

The table shows that the 4-tap LMS equalizer consumes almost 80% of the FPGA resources.

For the frequency domain equalizer, we will go through the same steps as the time domain equalizer to finally compare between the two equalizers.

4.5 Frequency domain equalizer implementation:

The frequency domain equalizer is based on the Fast LMS algorithm to update its coefficients. To test its performances, we describe it and simulate it using Matlab

The simulation parameters used for the frequency domain equalizer are the same used for the time domain equalizer. They are set as following:

- SNR = 40 dB
- Transmitted symbols = 10 000
- $\mu = 0.002$

The two pictures below present the constellation of the input signal to the equalizer and the constellation of its output signal.

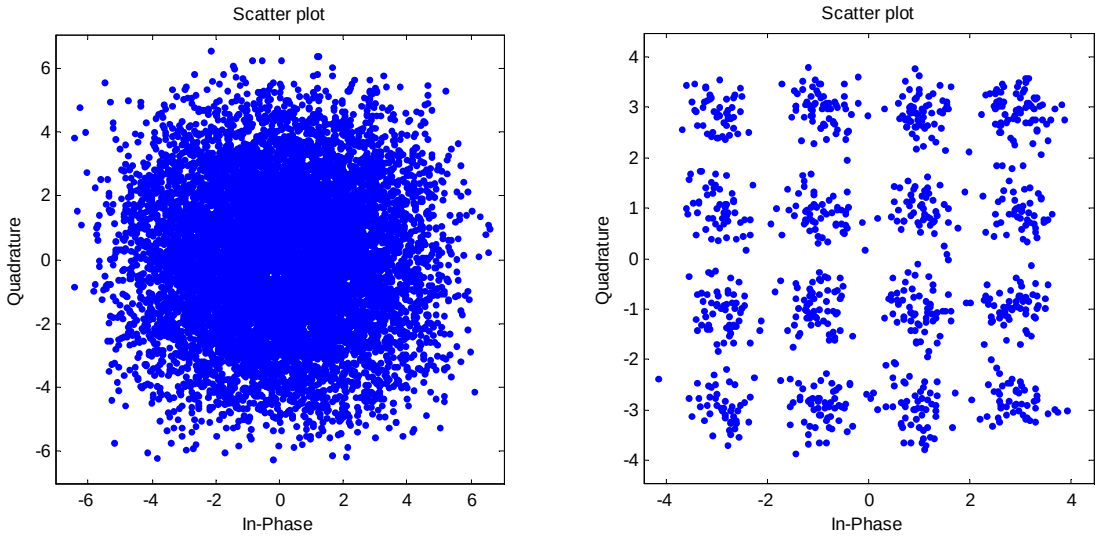

Figure 4.17: (a) Signal constellation before Fast LMS-based equalization, (b) Signal constellation after Fast LMS-based equalization

The figure 4.17.a shows that the input signal to the equalizer is very noisy, and after equalization, a 16-QAM constellation was obtained which means that the frequency domain equalizer eliminates the ISI.

The error estimation curve is given by the figure 4.18.

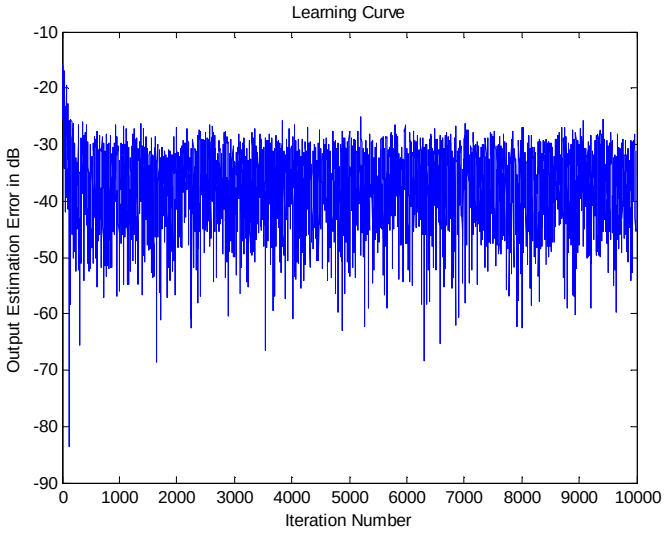

Figure 4.18: Fast LMS equalizer error estimation

The error estimation curve for the fast LMS equalizer shows that it converges within only 200 symbols almost to reach an error rate equal to -40 db for an SNR equal to 40 dB. These results imply that the frequency domain equalizer is much more efficient in terms performance than the time domain equalizer.

Using the Fast LMS algorithm as the adaptive algorithm for the equalizer gives an error rate equal to 1.83% which means that by sending 10000 symbols, we only loose 183 symbols.

Now, the transmission chain must be described using the blocks of the simulink environment. The figure 4.19 shows the communication environment where the 4-tap frequency domain equalizer is tested.

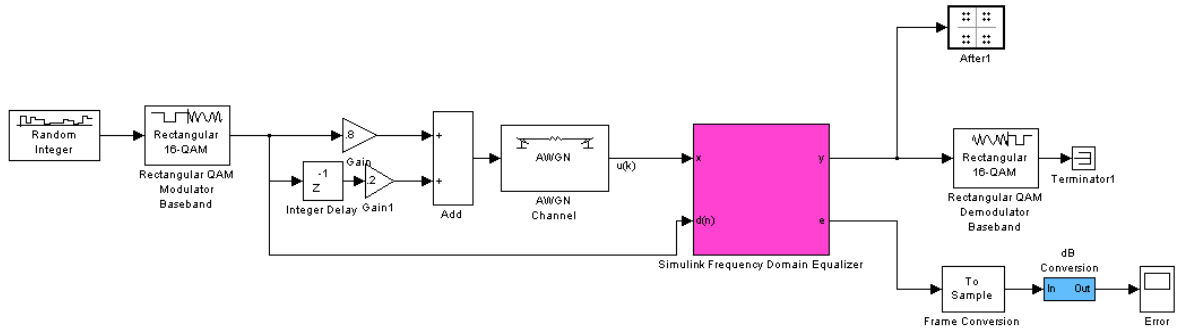

Figure 4.19: Transmission chain with a 4-tap Fast LMS equalizer in Simulink

It is obvious from the figure above that the transmission chain enabling to test he frequency domain equalizer is the same as the chain used with time domain equalizer.

The architecture of the Fast LMS-based equalizer is given by the figure below.

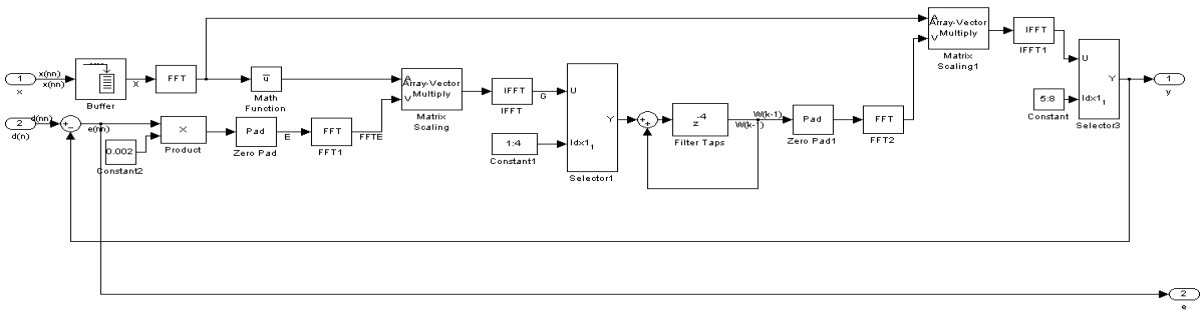

Figure 4.20: Fast LMS equalizer in Simulink

Just like the time domain equalizer, the frequency domain equalizer has as input the noisy signal coming from the channel as well as the desired signal which is in our case the output signal of the modulator. The output signals of the equalizer are the equalized signal and the error estimation. In this equalizer architecture, we notice the use of three FFT blocks and two IFFT blocks allowing the alternation between the time domain and the frequency domain.

Next, a simulation must be done to test the functionality of the frequency domain equalizer. The simulation parameters were kept the same as the time domain simulation parameters where:

- SNR = 40 dB
- μ = 0.002
- Filter taps = 4

The figure 4.12 shows the constellation of the equalizer output signal. The constellation shows no noise which implies that the equalizer is working perfectly.

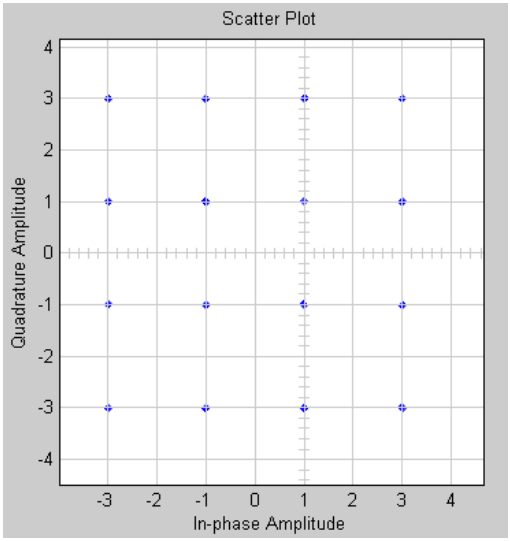

Figure 4.21: Signal constellation after time domain equalization in Simulink

In terms of error estimation, it is shown that the 4-tap frequency domain equalizer converges after almost 200 symbols to reach the value of -40 dB. These results prove that the frequency domain equalizer is better than the time domain equalizer in terms of performance. The error estimation curve is represented by the figure 4.22 below.

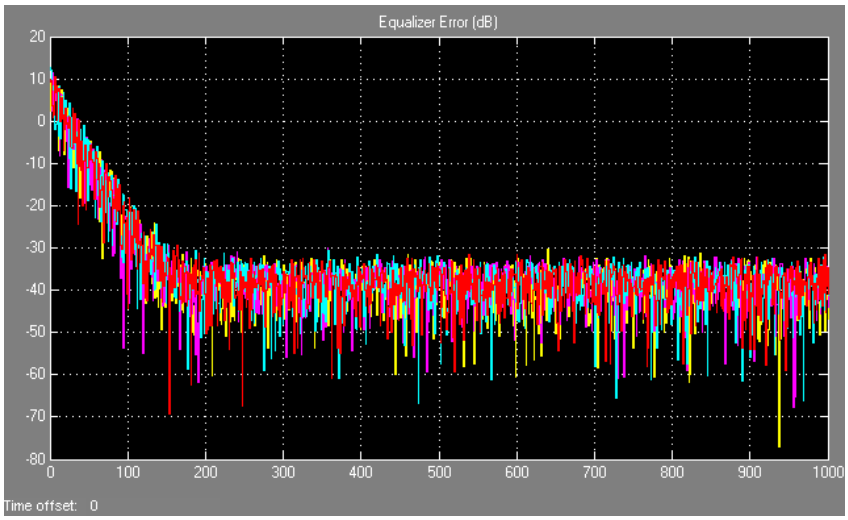

Figure 4.22: Error estimation

The frequency domain equalizer is described using the specific Xilinx block set in the simulink environment. We try to implement a 4-tap frequency domain equalizer on the one million gate Spartan 3 FPGA board but the design was very big that's why we implemented a 2-tap equalizer. The figure below shows the 2-tap frequency domain equalizer described using the Xilinx block set.

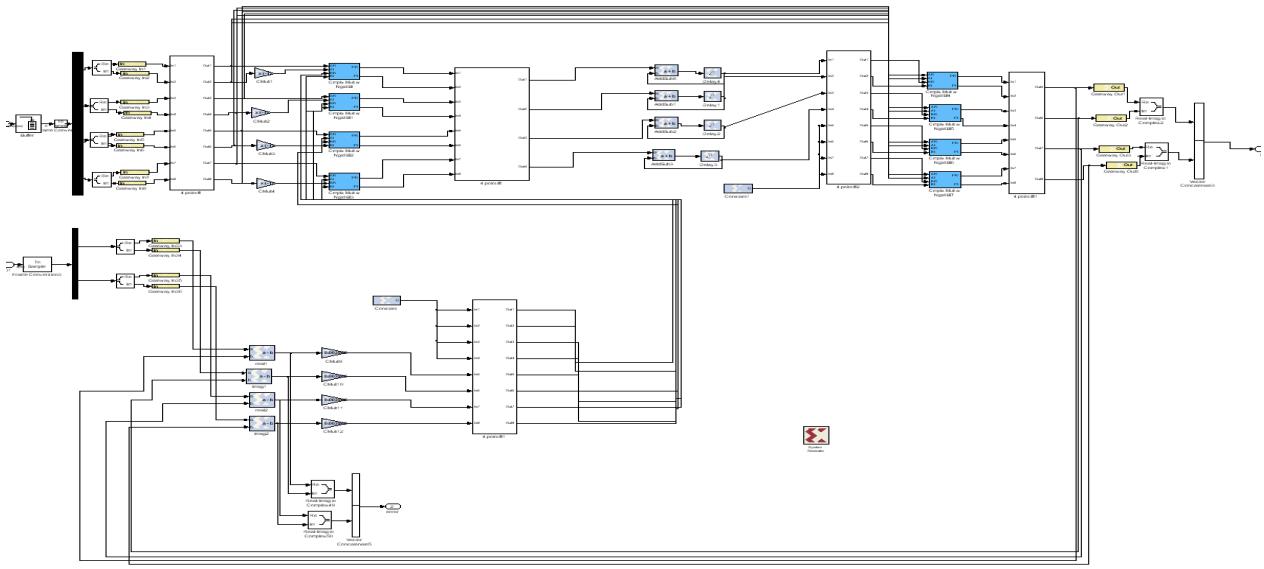

Figure 4.23: A System Generator description of the 2-tap frequency domain equalizer

The design contains elementary blocks from the Xilinx block set which are essentially multipliers and adders and also some sub-systems doing complex multiplication, FFT and IFFT.

The signal constellations as well as the error estimation curve given by the figures 4.24 and 4.25 respectively show that the equalizer eliminates the ISI from the noisy input signal.

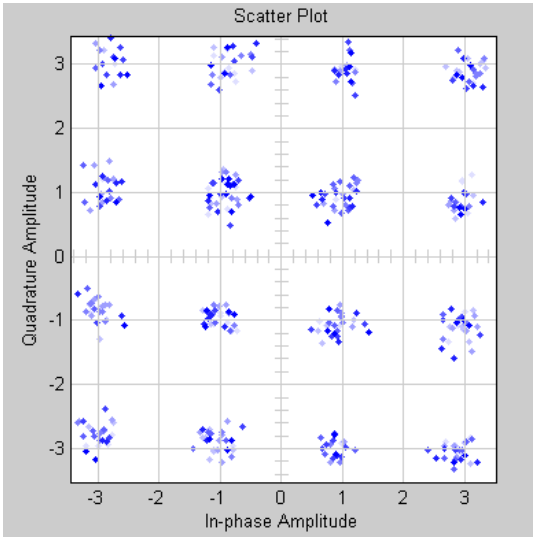

Figure 4.24: Signal constellation after time domain equalization in Sys Gen

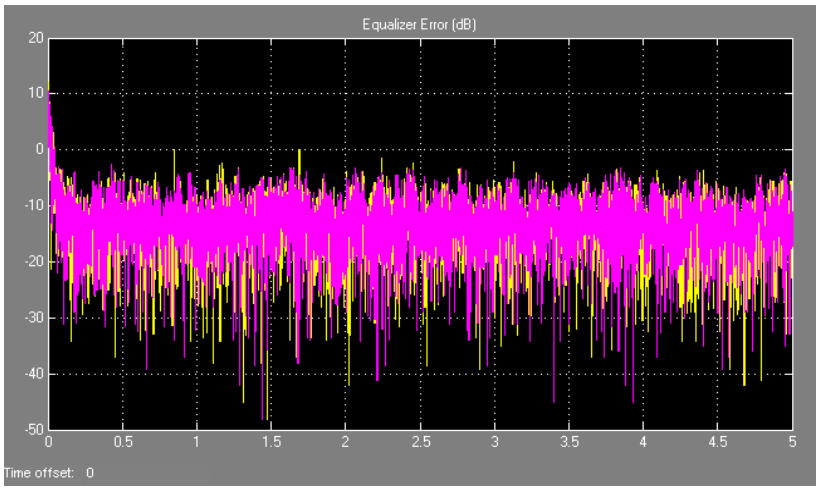

Figure 4.25: Error estimation curve

As it is mentioned for the time domain equalizer, the results got from the simulink model are better than those got from the System Generator model because for simulink symbol are described using floating point and for System Generator symbols are described using fixed point.

For the synthesis, place and route and FPGA implementation were done for the same one million gate Spartan 3 FPGA board and the table 4.2 gives the statistics of the used logic to implement the frequency domain equalizer design

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	56	15,360	1%	
Number of 4 input LUTs	9,157	15,360	59%	
Number of occupied Slices	4,818	7,680	62%	
Number of Slices containing only related logic	4,818	4,818	100%	
Number of Slices containing unrelated logic	0	4,818	0%	
Total Number of 4 input LUTs	9,173	15,360	59%	
Number used as logic	9,157			
Number used as a route-thru	16			
Number of bonded IOBs	201	391	51%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	2.49			

Table 4.2: Frequency domain equalizer synthesis report

The table shows that the 2-tap Fast LMS equalizer consumes almost 60% of the FPGA resources.

4.6 Comparison:

Now that both time and frequency domain equalizers are implemented it is time to compare between the two domains in terms of performance and computational complexity.

4.6.1 Computational complexity:

For the LMS algorithm, getting an output sample requires N complex multiplications which mean that generating an N sample output block requires N^2 complex multiplications. Also, updating the filter coefficients require N^2 complex multiplications. In summary, the LMS algorithm complexity is equal to $2\,N^2$ complex multiplications which is equivalent to $8\,N^2$ real multiplications.

Concerning the Fast LMS algorithm, the whole complexity results of the FFT and IFFT blocks. Since the computational complexity of an $2N$ FFT block is equal to $(N/2)\log_2(2N)$ in terms of complex multiplications then the 5 FFT and IFFT blocks need $(5N/2)\log_2(2N)$

complex multiplications which is equivalent to $10N \log_2 (2N)$ real multiplications. The Fast LMS algorithm requires also two *2N* complex vector products adding *16N* real multiplication to its computational complexity. Finally, the cost of the Fast LMS algorithm in terms of computational complexity is equal to $10N \log_2 (2N)+16N$.

The number of the real multiplications required by the two algorithms for *N* symbol block is tabulated in the table 4.3 where N is equal to 2, 4, 8, 16 and 32.

	N=2	*N=4*	*N=8*	*N=16*	*N=32*
LMS	32	128	512	2048	8192
Fast LMS	72	184	448	1056	2432

Table 4.3: Equalizer complexity comparison

From the table above, we deduce that the frequency domain equalizer is much better than the time domain equalizer in terms of computational complexity when N is equal or greater than 8. These statistics confirm the FPGA implementation results that we got of the time domain and frequency domain equalizers. As it is declared in section 4.5, a 4-tap time domain equalizer consumes almost 80% of the FPGA resources. In another hand, the 4-tap frequency domain equalizer needs over than 200% of the FPGA resources to be implemented as it is shown in the table 4.4, which confirms the theoretical results confirming that when N is less than 8 the LMS algorithm offers significant savings over the Fast LMS algorithm.

Device Utilization Summary (estimated values)				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	
Number of Slices	16069	7680	209%	
Number of Slice Flip Flops	161	15360		1%
Number of 4 input LUTs	31551	15360	205%	
Number of bonded IOBs	2401	391	614%	
Number of GCLKs	1	8		12%

Table 4.4: Synthesis report of 4-tap frequency domain equalizer

4.6.2 Performance:

In terms of performance, all the simulation results according to Matlab, Simulink or SysGen/Simulink confirm that the frequency domain equalizer is more efficient than the time

domain equalizer. First, because it converges faster than the time domain equalizer and second because it reaches a lower error rate.

4.7 Conclusion:

In this section, we went through the whole hardware design flow to implement the time domain and the frequency domain equalizers on FPGA. Although, we were limited by the hardware resources, we succeed to deal with what we have to get reasonable results allowing us to compare between the two designs. Since this is a research work as it is a master project and also it is done in the HVG lab where formal verification techniques are subject for research, we will present in the next section some future ideas related to formal verification that can be applied for this project.

Chapter 5

Conclusion & further work

5.1 Conclusion:

This master thesis was dealing with the implementation of a frequency domain equalizer on FPGA. Unlike the implementations previously described in the related work section which are based on improving the LMS algorithm to implement less complex and more efficient time domain equalizers on FPGA, the thesis contribution consists on implementing a frequency domain equalizer based on the Fast LMS algorithm on FPGA. It is proven by the signal processing researchers that the frequency domain equalizers are more efficient than the time domain equalizers, but in terms of hardware it is not proven yet.

Therefore, we proceed first to a co-simulation of the transmission chain where all the blocks are implemented using simulink except the equalizer which is implemented on FPGA. Then, a comparison was made in this thesis between the time domain and the frequency domain equalizers implemented on FPGA and put under the same conditions. The convergence characteristics and the computational complexity of the LMS and the Fast LMS algorithm were examined. The results of this study are as follows:

1. The Fast LMS algorithm based on the Overlap-Save algorithm gives better convergence than the LMS algorithm. The error estimation curve given by the frequency domain equalizer converges faster and reaches a lower error rate than the error curve given by the time domain equalizer.
2. In terms of computational complexity, it is proven that the 4-tap frequency domain equalizer is more complex than the 4-tap time domain one in terms of hardware consumption. This result ensures the computational complexity comparison analysis of the two algorithms.

The frequency domain algorithm offers significant savings in terms of computational complexity for long size equalizers. However, implementing those long size equalizers on FPGA requires a lot of hardware resources. It is proven in this master project that, using a one

million gate FPGA and the System Generator for DSP as a design tool, only a 2-tap frequency domain equalizer and a 4-tap time domain equalizer can be implemented.

This thesis presented a hardware implementation project. Therefore, as a further work, formal verification techniques can be applied on the frequency domain equalizer design as explained in next section.

5.2 Further work

5.2.1 Introduction

Designing consists in transforming a set of specifications into implementations. However, it is a long process; going from a specification described in high level language to an implementation of that specification on a hardware target implies the passage by many levels of abstractions: functional level, register transfer level, gate level…The impact of faulty design can be catastrophic since it can delays the time-to-market and as a result it can affects the company revenues. Therefore, the design verification would be necessary to avoid those problems. This process is the reverse process of design and it answers the question: does the implementation meet its specifications? In this chapter, we will present some ideas that can be subject of a PhD thesis. All of them are dealing with the verification techniques since this project was held in the Hardware Verification Group; the most famous lab of formal verification in North America.

5.2.2 Simulation based and formal based verification techniques

- ✓ **Simulation-Based Verification**

Simulation-based verification [5] is the most used verification technique. Verifying a design using simulation consists in putting that design under a test bench where input stimuli are applied. The output of the test bench is compared to a reference one. The test bench is a program that supports the operations described in the design and also it has the ability to engender the input stimuli which can be generated before the simulation process or when simulation is running similarly to the output reference generation. Figure 5.1 summarizes the simulation based verification principle.

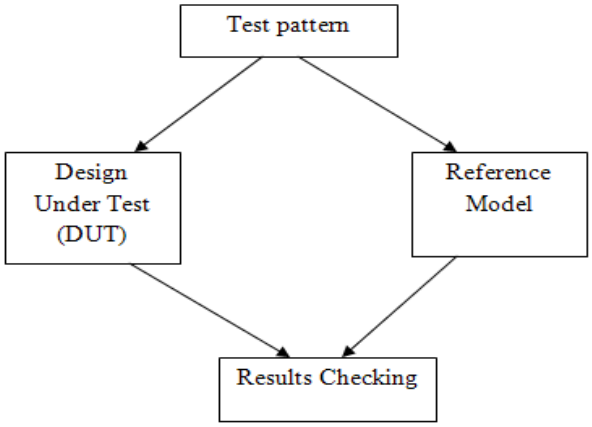

Figure 5.1: Simulation-based verification [25]

Simulation is functional verification since it is based on what is called directed tests. It consists in generating input test vectors aiming to test a specific functionality of the design. To get a full simulation-based verification, simulation must not be done on large pieces of the design. Therefore, the simulation must be done on the lowest levels. There are many types of simulation depending on the different levels of the design; the designer level simulation where a macro is verified, the unit level simulation where a group of macros is verified, the element level simulation where an entire logical function like a processor is verified and the system level simulation where multiple chips are verified. Figure 5.2 shows the different levels of a design called the Hierarchical design.

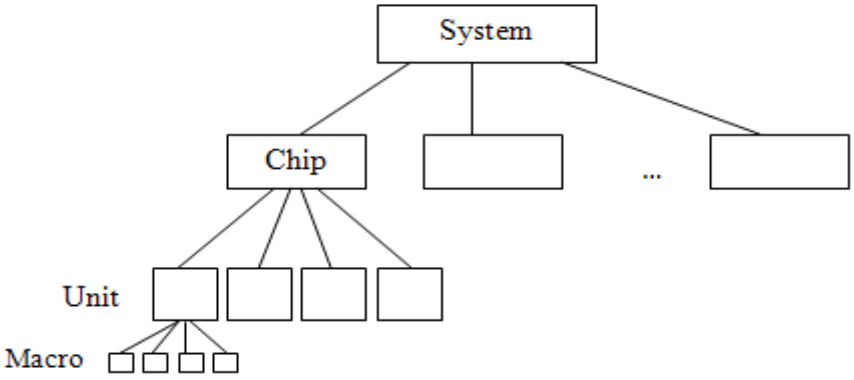

Figure 5.2: Hierarchical design [25]

✓ **Formal Method-Based Verification**

The formal method-based verification [5] differs from the simulation-based verification in that it is not based on test patterns generation. It consists in proving that the implementation meets its specification using a mathematical reasoning just like a mathematical proof. Formal verification techniques are more efficient than simulation because the consideration of all cases is implicit and consequently a full verification is guaranteed.

To get formally verify a design; we need to have a formal specification as well as a formal description of the implementation. Figure 5.3 summarizes the principle of the formal verification.

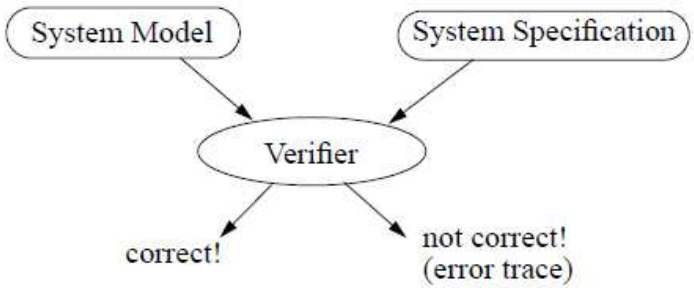

Figure 5.3: Formal method-based verification [25]

There are three techniques in formal verification which are equivalence checking, model checking and the theorem proving.

5.2.3 Further work

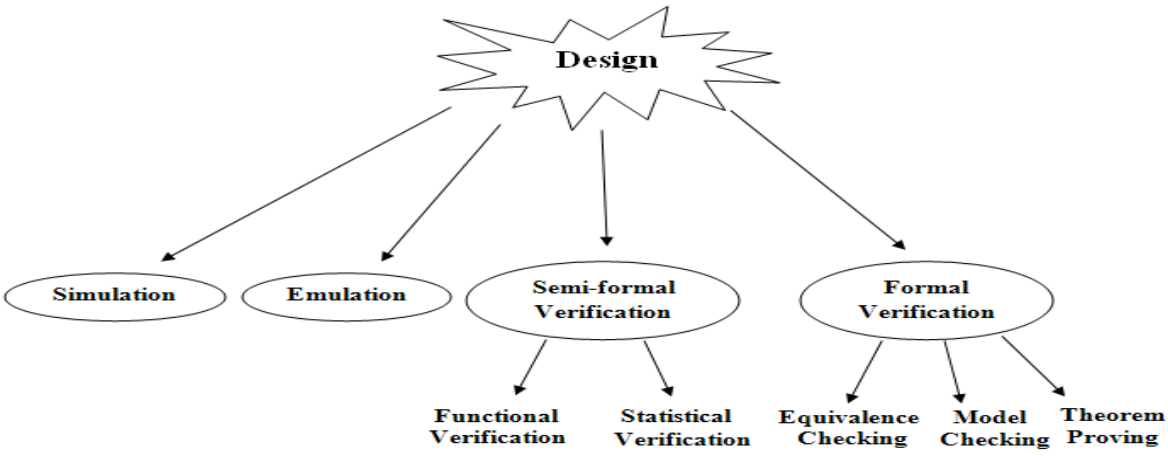

Figure 5.4: Verification techniques

Verification techniques can be broadly classified into three main categories: (1) formal methods based techniques; (2) simulation based techniques; and (3) test based techniques [5]. The formal verification techniques can be further classified into three main categories. They are (1) theorem proving, (2) model checking, and (3) equivalence checking. A class of verification techniques called the semi-formal verification techniques which combine both the formal and simulation based techniques are also commonly used.
Semiformal techniques were developed to take advantage the best of the two techniques (formal techniques and simulation). Emulation is a technique in which the design and the properties are automatically synthesized in to hardware, for example on FPGA [6], thereby increasing the simulation speed by several orders of magnitude. This technique is useful when simulations have to be repeated several thousands of times, for example, in statistical simulations, or when access to the design environment is not easy. For example emulation can be used to create a realistic environment or just to speed up the simulation by off loading the computationally complex parts of the simulation to actual hardware.

Figure 5.4 above shows a high level classification of the verification techniques used these days. Rest of this section briefly describes the verification techniques and explains how some of these techniques can be used to verify the functionality and performance of the Adaptive Filters, such as the implementation of the Fast Least Mean Squared (LMS) Adaptive Equalizer [7].

- ✓ **Semi-formal verification**

Applying functional or statistical verification [8] in our project is feasible. We have first to define the properties to be verified, in our case, we can take as property the algorithm adaptation rate, number of samples required for convergence, average error rate or average mean squared error noise floor once the algorithm has converged. All of these properties provide some measure of performance of adaptive equalizer implementation. Then, instead of using testbench based on simple test vectors defined by the verifier, we formalize the property constraints in a formal way so we get the assertion; it is a kind of conditions which must be satisfied by the design. By formalizing the property in a formal way, we mean that the property be described in way such that it is clear what are the conditions which will make the property true, and what would happen if the property is true and when it false.

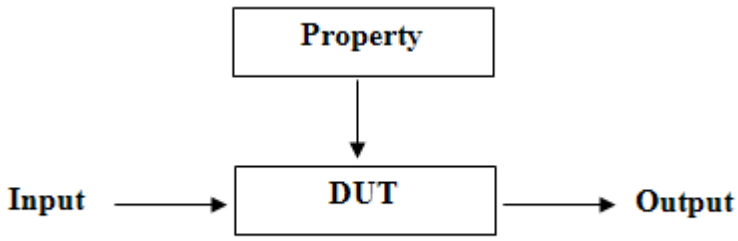

Figure 5.5: Assertion based verification

We have to describe expected or unexpected conditions in device under test in a specialized language then we have to check to make sure that conditions are satisfied dynamically during simulation (functional verification) or statically. In case of conditions failure, then a bug is detected in the device.

In the case of the functional assertion based simulation, we run simulation just once and then we check if the properties are valid. Most designs have random components in it or operate in an uncertain environment. Statistical assertion based techniques are used in such cases. In the case of statistical assertion based verification we run simulation several times and then by applying the Monte Carlo method [9] we extract automatically some of the simulation results on which we can do statistical studies and then we see if the property constraints are satisfied or not.

The advantages of using assertion based verification are that this technique is automatic and quick in debugging a device since it is a Combination of simulation and formal verification. It is automatic unlike many formal methods based techniques such as higher order logic theorem proving and does not have a state explosion problem such as model checking. Similar to simulation based techniques it also cannot not completely verify a design.

What is required to get good results are methodologies for choosing good assertions, intelligent debug systems to understand and analyze assertions and their results and a standard assertion format and good tool support.

- ✓ **Theorem proving**

For this project, the performance of the adaptive filters [7] can be formally analyzed and compared using theorem proving. We can verify minimum and maximum error bounds and expectation and variance of error between the outputs of the two implementation of the

adaptive filter.

First, we have to define the System Generator [10] and the Matlab [11] designs as models written in High-Order-Logic [12] representing, respectively, the fixed-point and the floating-point implementations. Then, based on the outputs of those two models, y and y', we calculate the error which is given by the formula e=y-y'. After that, we can verify, given an adaptive filter algorithm and its fixed-point implementation what are the maximum and the minimum bounds on the error using a theorem prover [12, 13, 14, 15, 16].

We can also formally determine the statistical properties of the error. For example, its expectation, E[e], and its variance, VAR[e] [17]. Both the error bounds and the statistical properties will be generalized expressions in terms of the parameters of the adaptive algorithm and its implementation.
Figure 5.6 summarizes the performance analysis methodology using theorem proving.

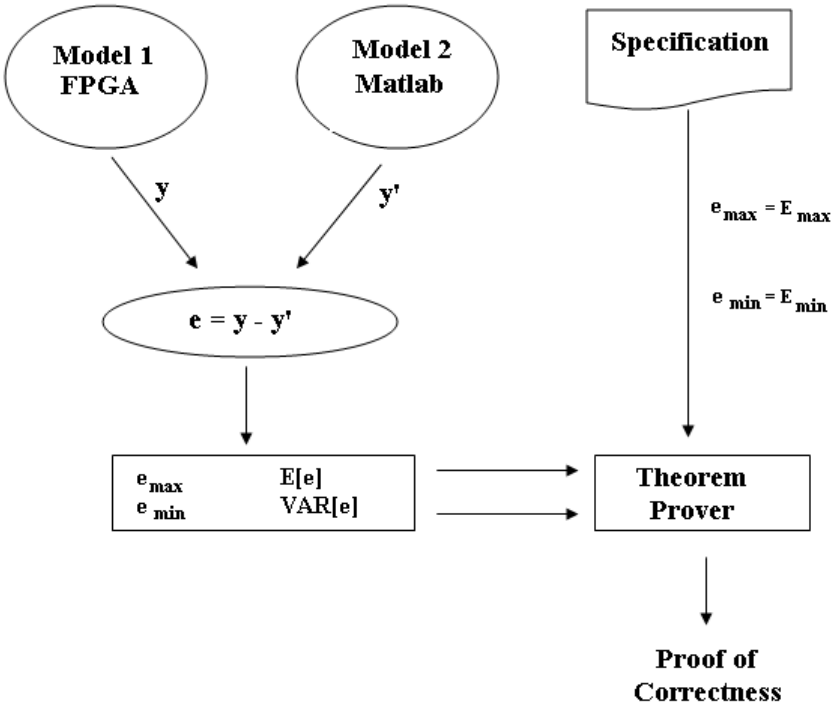

Figure 5.6: Performance analysis methodology with theorem proving

The use of the theorem proving as a formal performance analysis and verification technique, even though it needs some expertise, but it is much more useful and efficient than the run time

verification or simulation since it provides results as general expressions from which we can do verification for all possible cases which is not the case for simulation because to verify for all possible cases we may need an infinite amount of time which is impossible.

Bibliography

1. Chun-Nan Chen, Kuan-Hung Chen, and Tzi-Dar Chiueh "*Algorithm And Architecture Design For A Low-Complexity Adaptive Equalizer*" IEEE, 2003, PP II304-II307.

2. Gabriel Bucco, Matthew Trinkle, Doug Gray and Wai-Ching Cheuk "*FPGA Implementation of a Single Channel GPS Interference Mitigation Algorithm*" *Journal of Global Positioning Systems (2004) Vol. 3, No. 1-2: 106-114.*

3. Chris Dick and Fred Harris "*FPGA QAM Demodulator Design*" Springer-Verlag Berlin Heidelberg 2002, PP. 112-121.

4. J. G. Proakis, *Digital Communications,* McGraw-Hill, New York, NY, USA, 4th edition, 2001.

5. William K. Lam. *Hardware Design Verification: Simulation and Formal Method-based Approaches.* 2005: 1-23.
6. Wiley. *FPGA-based implementation of signal processing systems*. 2008
7. Simon Haykin. *Communication Systems, 5th edition*. 2009.
8. Foster, Harry, Krolnik, Adam, Lacey, David. *Assertion-Based Design, 2nd edition*. 2004.
9. Rubenstein, Reuven Y, Kroese, Dirk P, Botev, Zdravko I, Taimre, Thomas. *Simulation and Monte Carlo Method, 2nd edition*. 2008.
10. http://www.xilinx.com/support/sw_manuals/sysgen_gs.pdf3
11. http://www.mathworks.com/
12. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A theorem proving environment for higher order logic*. 1993
13. Matt Kaufmann, Panagiotis Manolios and J Strother Moore. *Computer-Aided Reasoning ACL2 Case Studies*. 2000.
14. http://pvs.csl.sri.com
15. http://isabelle.in.tum.de/
16. http://coq.inria.fr/
17. Richard A. Johnson, *Miller & Freund's probability and statistics for engineers, 8th edition*.
18. Xilinx Inc.:
    http://www.xilinx.com/tools/sysgen.htm
    http://china.xilinx.com/publications/prod_mktg/pn0010676-2.pdf
19. Auto ESL Inc.:

http://www.autoesl.com/

20. Synphora Inc. :
    http://www.synphora.com/

21. Mentor Graphics :
    http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/
    http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/upload/Catapult_DS.pdf

22. Myles O Fril, *Frequency Domain Adaptive Filtering*. Master's thesis, National University of Ireland, 2005.

23. P. A. Dmochowski. *Frequency domain equalization for high data rate multipath channels*. Master's thesis, Queen's University, 2001

24. Christophe Laot. *Égalisation autodidacte et turbo-égalisation, Application aux canaux sélectifs en fréquence.* PhD thesis, Université de Rennes 1, 1997

25. Sofiène Tahar, *Formal verification*. 2000