POLITECNICO DI TORINO

Faculty of Engineering

Master degree in

Communications and Computer Networks Engineering

MASTER THESIS

Software Defined Radio Implementation of a Satellite Radio System

Supervisor

Prof. Roberto Garello

Candidate

Luca Vigna

April 2020

To you who are the most beautiful

gift life could ever give me

Abstract

This thesis is focused on the design and implementation of a Telecommand Satellite Radio System and it has been carried out with the Italian aerospace engineering company *Argotec*.

A design of the digital signal processing part of the system is proposed, having as guideline for the definition of the specifications the Consultative Committee for Space Data Systems (CCSDS) Standard.

The focus is mainly on the Physical and Coding layers of the architecture proposed by CCSDS. Simulations and performance analysis are carried out through the MATLAB Software.

A research on an experimental coding scheme, able to provide significant coding gain, is presented and the theoretical results are verified in practice.

The document ends with the implementation of the designed system, in C language, on a development board (equipped with a Xilinx Zynq-7000) used to test and validate the prototype.

Contents

List of Figures

List of Tables

Acronyms

ADC Analog to Digital Converter

AGC Automatic Gain Control

AWGN Additive White Gaussian Noise

BCH Bose–Chaudhuri–Hocquenghem

BER Bit Error Rate

BR Buffer Register

CCSDS Consultative Committee for Space Data Systems

CER Codeword Error Rate

CLTU Communications Link Transmission Unit

CMM Carrier Modulation Modes

DAC Digital to Analog Converter

DLL Delay Locked Loop

DSP Digital Signal Processing

EOD Even Odd Detector

FDD Frequency Discriminator Device

FIR Finite Impulse Response

FSM Finite State Machine

ISI Intersymbol Interference

LDPC Low-Density Parity-Check

LFSR Linear Feedback Shift Register

LLR Log-Likelihood Ratio

LO Local Oscillator

LUT Lookup Table

MA Moving Average

MS Min-Sum

NCO Numerically Controlled Oscillator

NMS Normalized Min-Sum

PL Programmable Logic

PLOP Physical Layer Operations Procedure

PS Processing System

RF Radio Frequency

RMS Root Mean Square

Rx Receiver

SDR Software Defined Radio

SNR Signal-to-Noise Ratio

SPA Sum-Product Algorithm

SR Syndrome Register

TC Telecommand

TM Telemetry

Tx Transmitter

CHAPTER 1

1 Introduction

1.1 BACKGROUND AND MOTIVATION

One of the basic functions of any spacecraft is the ability to establish a connection with ground stations or with other spacecrafts to exchange information, commands, control data, etc. To this purpose, the spacecraft uses a transceiver that can be designed for a fixed type of communication (using analog components) guaranteeing no flexibility to the user, being incapable of adapting to new configurations. For this reason, the trend is to move most of the functionality of a conventional radio to the digital domain (using re-programmable hardware logic) in which some of them are fully implemented in software. This new technology is the so-called Software Defined Radio (SDR). Usually, an SDR-based transceiver results in a compact, low power and flexible communication system at the expense of not exactly cheap high-performance components. However, in a satellite system the possibility to have a flexible device is very appealing, using a generic hardware design that can be used to address different communication needs, with varying characteristics.
Applying this concept to small satellites can increase data throughput, add the possibility to perform software updates over-the-air and make it possible to reuse the hardware platform for multiple missions with different requirements [1].

The focus of the aerospace company *Argotec* in the production of nanosatellites for Deep Space applications led to interest in creating a collaboration for an all in-house design and implementation of an SDR.

Argotec is an aerospace engineering company based in Turin and founded by David Avino in 2008. Actually, Argotec is on the verge of integrate two platforms (ArgoMoon and LICIACube) which will fly with two different NASA missions in the years 2020 and 2021.
As anticipated, the platform conceived by the company is based on the all in-house concept. Starting from the ideas of Argotec's engineers, a first design of the project is made. Then, the process of the realization and testing of all the subsystems up to their integration on the comprehensive platform is done in the internal laboratories.

The main pillars of this thesis work are the needs of the company to expand its Portfolio with a deep-space radio for nanosatellites, the opportunity to implement an SDR that can be ported on a real flight system hardware and the will to research and develop new technologies that can improve the actual performance of a system which is experiencing an important period of growth.

In a typical radio communication system, the basic data flow is made by Telemetry (TM) and Telecommand (TC) data, in downlink and uplink respectively.

The work presented in this thesis, is focused on the design and implementation of the Telecommand System. It must reliably and transparently convey control information from an originating source (located on the Earth) to a remotely located physical device (i.e. satellite).

1.2 OBJECTIVES

The goal of this master's thesis work is to make a preliminary analysis on the feasibility of the project starting from a basic design of the Sub-System and to build a prototype of the system able to work as close as possible to realistic conditions.

The object is to find the best solution that fits all the requirements. The first one is that the system must be compliant with the CCSDS Standard. Starting from it, the main objectives of the work are reported below:

- Literature study of a Satellite Radio System focusing on the Receiver (Rx) structure;
- Study of a Base Station Transmitter (Tx) to generate an Uplink signal, able to send realistic data;
- Feasibility analysis of the proposed architecture through MATLAB simulations;
- Implementation on a Development Board able to work in real-time;
- Characterization of each block in order to evaluate which of them could benefit from a hardware implementation (e.g. in terms of performance, resource utilization etc.);
- Test of the prototype simulating a real environment condition.

1.3 THESIS OUTLINE

The thesis is organized as follow:

- **Chapter 2,** Telecommand System Design, reports the study and the definition of the system requirements in order to have a guideline for the development. A functional block diagram of the receiver is proposed;
- **Chapter 3**, Physical Layer Design, describes the functionalities of the physical layer. The receiver block diagram, defined in the previous Chapter, is implemented and simulated in MATLAB in order to study the feasibility of each block (understanding the functionalities and the various parameters) fitting them to the generated Tx signal and analyzing the trade-offs;
- **Chapter 4**, Coding Layer Design, gives an overview of the Synchronization and Channel Coding Sublayer proposed by CCSDS is reported. The Frame Synchronizer and the Decoder implementations are shown with particular attention on their performance;
- **Chapter 5**, LDPC Code, presents, theoretically and practically, a new coding scheme that can work with very low value of SNR;
- **Chapter 6**, Prototype Testing, presents a real-time prototype. After the simulation model is validated, the MATLAB code is ported on a Development Board (C language) and the system is tested;
- **Chapter 7**, Final Remarks, draws the conclusion, highlighting the final result obtained and giving a hint about future works;

CHAPTER 2

2 Telecommand System Design

This chapter provides an overview of the project highlighting the characteristics of the transmitted signal, how it is simulated and what are the effects of the channel and of the non-ideal RF front ends on it. A high-level block diagram of the receiver adopted and the test set-up used are presented.

2.1 PROJECT DESCRIPTION

A space link is a communications link between a spacecraft and its associated ground system or between two spacecrafts. In the figure below is reported the typical configuration where the ground station sends telecommands to a spacecraft

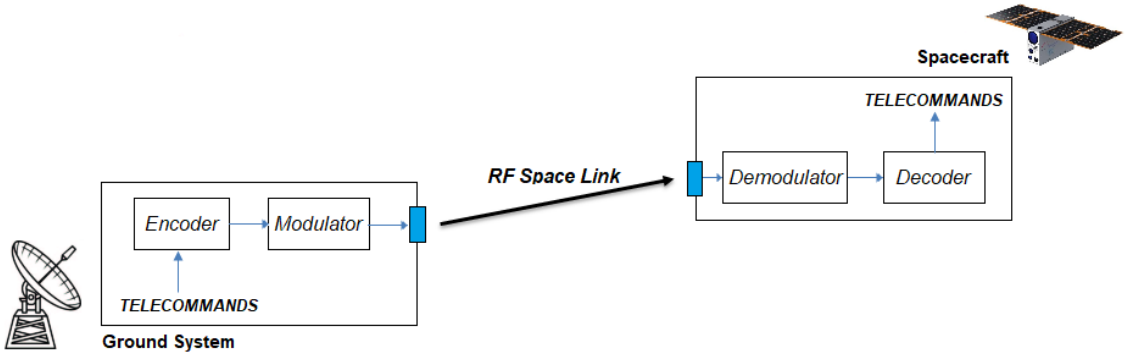

Figure 2.1 – Basic Scheme of Telecommand System

The telecommands are encoded and modulated from a ground system and sent over the Radio Frequency (RF) Space Link. The task of the Spacecraft's transponder is to receive the signal, demodulate it and extract the telecommand through a decoder.

The transponder on board the spacecraft is an SDR. It is a radio communication system that uses a minimum amount of analog/Radio Frequency components to down convert the RF signal from a digital format (and vice versa). Once analog data is digitized, all processing is performed using hardware logic. Typically, this processing includes filtering, modulation, up/down converting and demodulation.

Concerning the architecture of the satellite subsystem the Recommendations proposed by the CCSDS, that has the purpose to promote interoperability and cross support among cooperating space agencies, are taken as a reference point. In Figure 2.2 are represented the CCSDS protocols

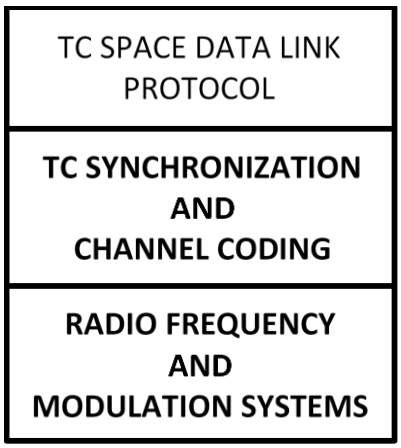

Figure 2.2 – CCSDS Protocols [1]

The focus will be mainly on:

- Physical Layer: provides the RF channel and the techniques required to operate it (e.g. modulation, demodulation, bit synchronization);
- Synchronization and Channel Coding Sublayer: provides functions necessary for transferring frames over a space link (e.g. error-control coding, synchronization, pseudo-randomizing, repeated transmissions).

2.2 PROJECT SPECIFICATION

The typical structure of the transponder is reported in Figure 2.3

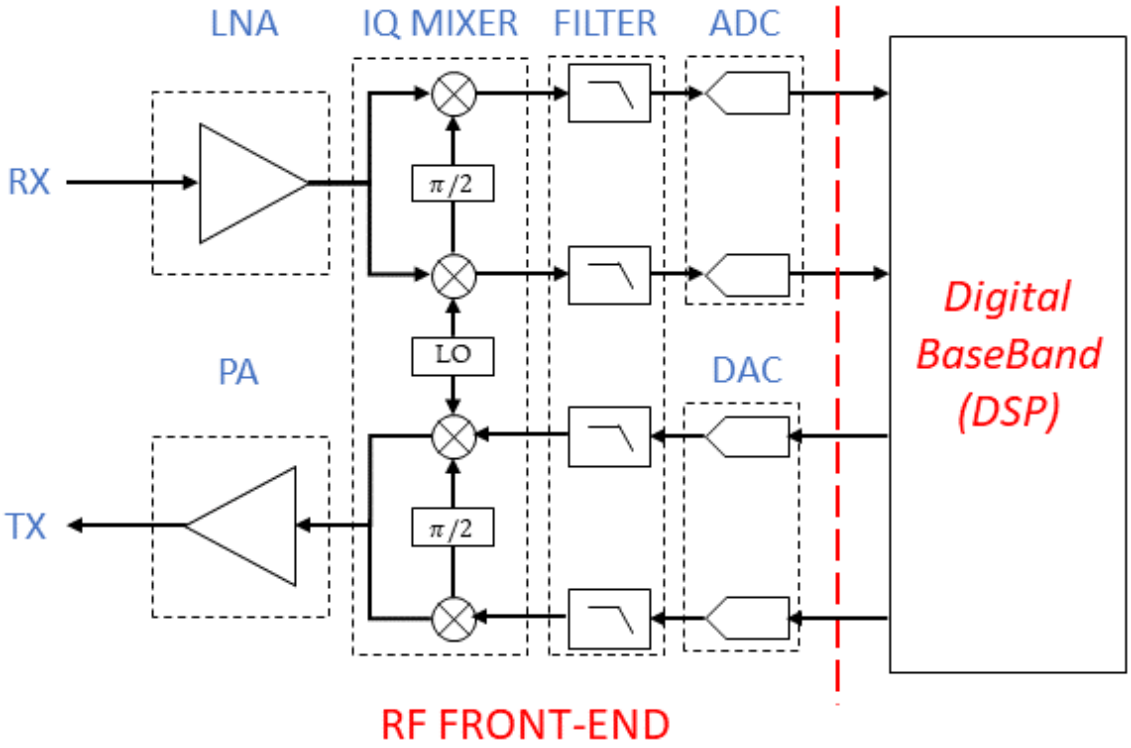

Figure 2.3 – Typical Spacecraft Transponder

This work does not deal with the RF Front-End block of the SDR but only the Digital baseband Signal Processing (DSP) of the signal. However, the impairments introduced by Tx and Rx front-ends are studied in detail.

The choice to be compliant with CCSDS standards leads to an easier definition of initial specification about the Radio Frequency signal. In the following table are reported the characteristics of the transmitted signal. The scope of these choices is that the result of the project must be an SDR prototype that is compatible with real space application. The specifications are explained, more in details, in the Section 2.4

Table 2.1 System Specifications

Specification	Value
Modulation Waveform	BPSK
Sub-carrier Waveform	*Sine Wave*
Sub-carrier frequency	$f_{sc} = 16\ kHz$
Sample Rate	$f_s = 48\ kHz$
Data Rate	$R_s = 1\ kbit/s$
Decoding Scheme	*BCH / SEC Mode*
Physical Layer	*PLOP-2*

2.3 FUNCTIONAL BLOCK DIAGRAM

The receiver designed in this Thesis is a coherent receiver. Coherent transmissions are transmissions where the receiver knows what type of data is being sent. Coherency implies a strict timing mechanism, because even a data signal may look like noise if the receiver looks at the wrong part of it [1].

More in general, a receiver must be able to demodulate and decode the received signal, but also it must recover as much as possible the impairments introduced by the channel and by the non-ideal components. A functional block diagram, taken as reference point, is reported in Figure 2.4

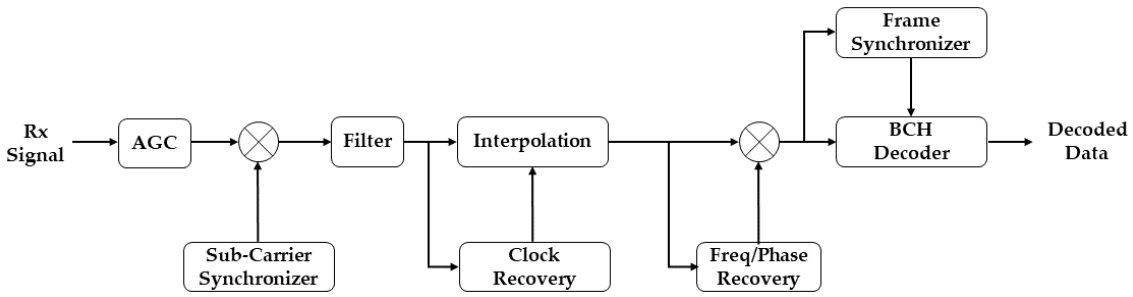

Figure 2.4 – Block Diagram of Telecommand System

2.4 TRANSMITTER SETUP

The transmitted signal, used as input for the receiver developed within the thesis context, is generated by a Workstation exploiting the software GNU Radio. It is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment [4]. Taking advantage of the blocks provided by the software, the signal is generated and sent to a binary file for MATLAB simulation. The generated flow will be then sent to the development board through a socket exploiting an Ethernet connection for the real-time test.

In Figure 2.5 is reported a high-level GNU Radio's block diagram

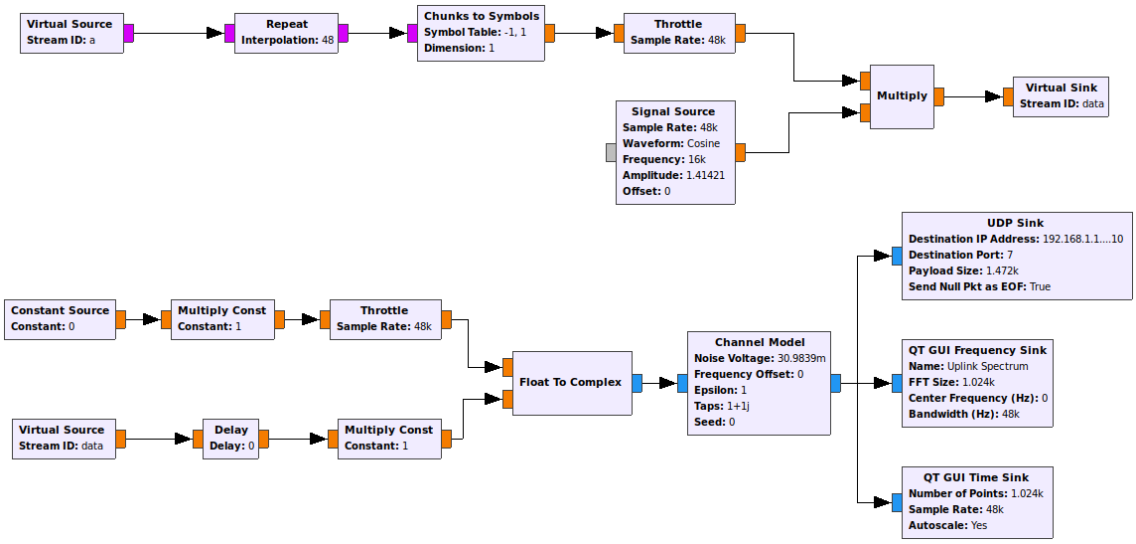

Figure 2.5 – GNU Radio flow graph for Tx signal generation

The specifications reported in Table 2.1 derive from CCSDS Recommendations. BPSK modulation is used since, it is considered the most robust modulation scheme in terms of noise immunity and it allows the highest level of distortion that can still successfully be modulated. A sinewave sub-carrier for telecommand

is chosen with a frequency of 16 kHz (PSK modulated). This allows to separate the data's transmitted spectrum from the RF carrier. The subcarrier is totally suppressed.

The Tx signal equation is reported below:

$$t_{signal}(t) = \sqrt{2P_t}\, e^{j2\pi f_c t} \sum_n a_n s(t - nT)\, e^{j2\pi f_{sc} t}$$

where P_t is the total available transmitter power, a_n are the sequence of constellation points, $s(t)$ is the shaping pulse, f_c is the carrier frequency and f_{sc} is the sub-carrier frequency. T is the data interval ($R_b = {}^1/_T$ is the data rate).

The response in time domain of the signal (no noise is present) is reported in Figure 2.6

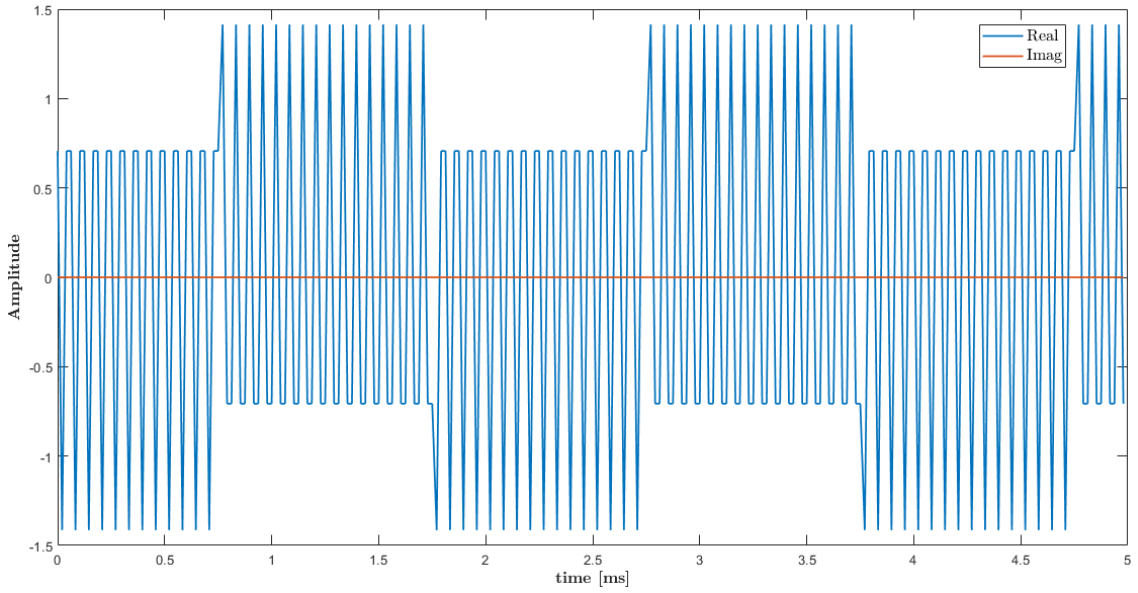

Figure 2.6 – Tx Signal in time domain (No Noise)

The choice to work at *48* samples per symbol derives from the requirement to process the signal at the minimum possible frequency for simulation purpose. Since the CCSDS does not give any specification about the Carrier Tx frequency, is taken as reference *768 kHz*. In order to satisfy Nyquist Theorem (in the RF front-

end) the signal should be sampled at *32 kHz* (twice the frequency of the Sub-carrier). When data are present this sampling is not enough to assure anti-aliasing, so the choice goes to the next integer divider of *768 kHz*. Summarizing it is supposed that the Analog to Digital Converter (ADC) at the RF front-end, samples the received signal at *48 kHz* in order to have an integer number of samples per symbol.

When data are present, the BPSK modulation at the sub-carrier frequency is visible in the spectrum of the signal

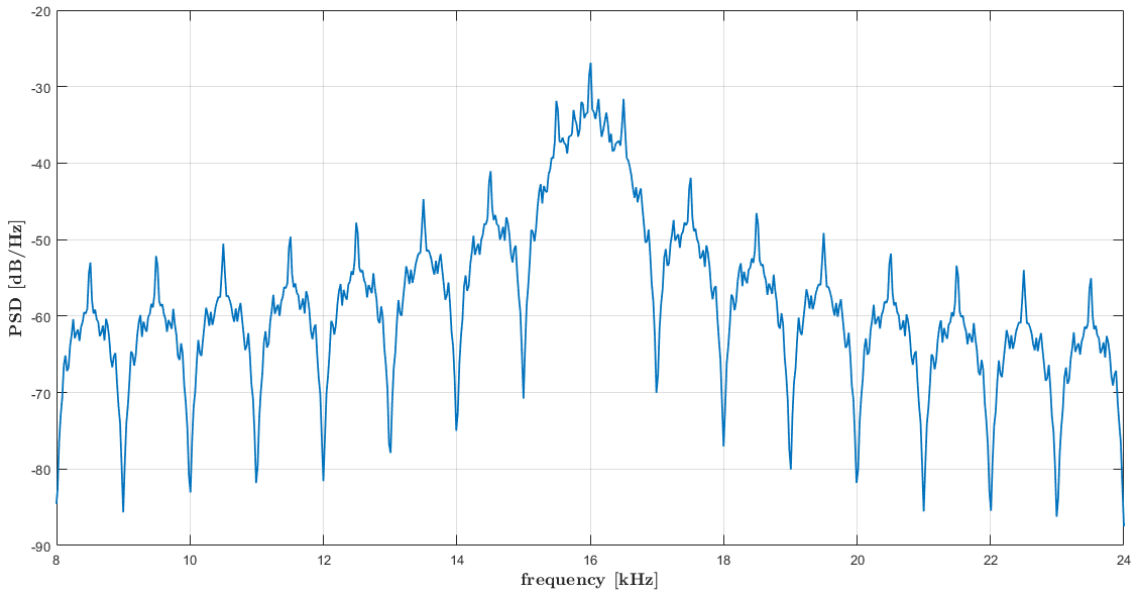

Figure 2.7 – Spectrum of the signal with data (No Noise)

The Physical Layer Operations Procedures (PLOPs) specify the sequence of operations performed during a communications session. For recent missions, CCSDS requires that PLOP-2 is used. It defines three data formats: Acquisition Sequence, Communications Link Transmission Unit (CLTU) and Idle Sequence. The Acquisition Sequence provides initial symbol synchronization within the incoming stream of detected symbols and its pattern consists in alternating "ones" and "zeros". The CLTU is a data structure that contains the data symbols that must be transmitted to the receiving end. The Idle Sequence is the data

structure which provides for maintenance of symbol synchronization in the absence of CLTUs and its pattern is the same of the Acquisition Sequence. In PLOP-2 the channel is not deactivated after each transmitted CLTU and for this reason, when there is a connection, the Acquisition Sequence is always present to ensures that the bit synchronization is maintained. Moreover, a minimum Idle Sequence of one octet is systematically inserted between each CLTU to eliminate the small but finite possibility of synchronization lockout [1].

2.5 RECEIVED SIGNAL

Transmitter and Receiver RF Front-Ends introduce impairments to the Received signal due to their non-idealities. The objective of the digital baseband block is also to recover as much as possible these impairments.

Considering only an Additive White Gaussian Noise (AWGN) introduced by the channel, the Rx signal at the RF Front End is in the form:

$$\mathrm{r}(t) = \sqrt{2P_t}\, e^{j2\pi f_c t} \sum_n a_n p(t - nT) e^{j2\pi f_{sc} t} + n(t)$$

where a_n are the sequence of constellation points used at the Tx, $p(t)$ is the cascade of the Tx filter and the impulse response of the channel ($s(t) * c(t)$) and $n(t)$ is the noise introduced by the channel.

Typical quantities used to describe the relative power of noise in an AWGN channel are the Signal-to-Noise Ratio (*SNR*), the ratio of bit energy to noise power spectral density $\left(\frac{E_b}{N_0}\right)$ and the ratio of symbol energy to noise power spectral density $\left(\frac{E_s}{N_0}\right)$. The relation between the last two quantities (expressed in *dB*) is:

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} + 10 \cdot log_{10}(k)$$

where *k* is the number of information bits per symbol.

In the considered system, *k* is influenced by the size of the modulation alphabet (*M*) and by the code rate (R_c) and it is:

$$k = R_c \cdot log_2(M)$$

Instead, the relationship between $\frac{E_s}{N_0}$ and the *SNR* (in dB), for a complex input signal, is:

$$\frac{E_s}{N_0} = 10 \cdot log_{10}\left(\frac{T_{sym}}{T_{samp}}\right) + SNR$$

where T_{sym} is the symbol time $\left(\frac{1}{R_s}\right)$ and T_{samp} is the sampling time $\left(\frac{1}{f_s}\right)$ [5].

The block that models the channel in GNU Radio requires the noise voltage as parameter for the AWGN noise. Knowing that the SNR is the ratio between the input signal power (S) and the noise power (N) it is possible to derives the noise voltage as the square root of N exploiting the formulas above. The noise voltage is equal to:

$$\sqrt{N} = \sqrt{\frac{S}{\frac{E_b}{N_0} + k - \frac{T_{sym}}{T_{samp}}}}$$

where all the quantities are considered in linear scale. Choosing a value of $\frac{E_b}{N_0}$ a corresponding noise voltage to model the AWGN channel is provided.

In the received signal, also non-idealities are present and they are originated by amplifiers, mixers, A/D and D/A converters.

Amplifiers introduce *third-order distortion* and *additive thermal noise* (that is AWGN).

Mixers (up and down convert the complex baseband signal that contains the useful information) with Local Oscillator (LO) introduce *carrier frequency offset* $\upsilon(t)$, *phase noise* $\theta(t)$ and *IQ imbalance*.

ADC and Digital to Analog Converter (DAC) carry other impairments as *sampling clock offset* (τ) and *sampling jitter* ($\tau(t)$).

After the analog RF front-end (at the digital baseband) the signal is affected by different types of impairments. Rx signal, with the RF impairments, is:

$$r_{BB}(t) = \sqrt{2P_t}\, e^{j(2\pi\upsilon(t)t+\theta(t))} \sum_n a_n p\big((t-nT)-\tau(t)\big) + w(t)$$

where $w(t)$ takes into account the noise introduced by the channel plus other thermal interferences.

The presence of the subcarrier is ignored in the equation, but its contributions are still present and must be removed by the digital baseband block through the Sub-Carrier Synchronization block.

2.6 TEST SET-UP

As already anticipated, once the system is validated through MATLAB simulation, a porting of the project, in C language, is done and it is executed on a Development Board. The Set-up is shown below

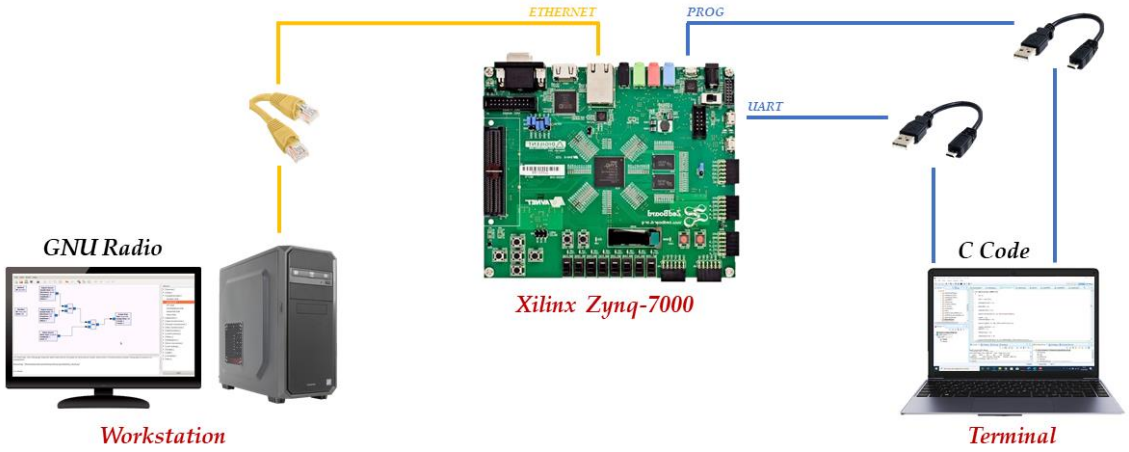

Figure 2.8 – Block Scheme of the test set-up

In the Figure above the test Set-up is shown. It is made by a transmitter that is a Workstation that generates the signal exploiting the software GNU Radio (see Chapter 2.4). In details, the Figure represents the connections to make the virtual link between the Workstation and the development board (ETHERNET), to program the Xilinx (PROG) and to print on terminal some control variables (UART).

The signal is sent through a Gigabit Ethernet connection to a ZedBoard that is an evaluation and development board based on the Xilinx Zynq-7000 All Programmable SoC (Zynq-7000 SoC XC7Z020). It is composed of two major functional blocks: the Processing System (PS) and the Programmable Logic (PL). The ARM Cortex-A9 CPUs are the heart of the processing system and also include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces [6]

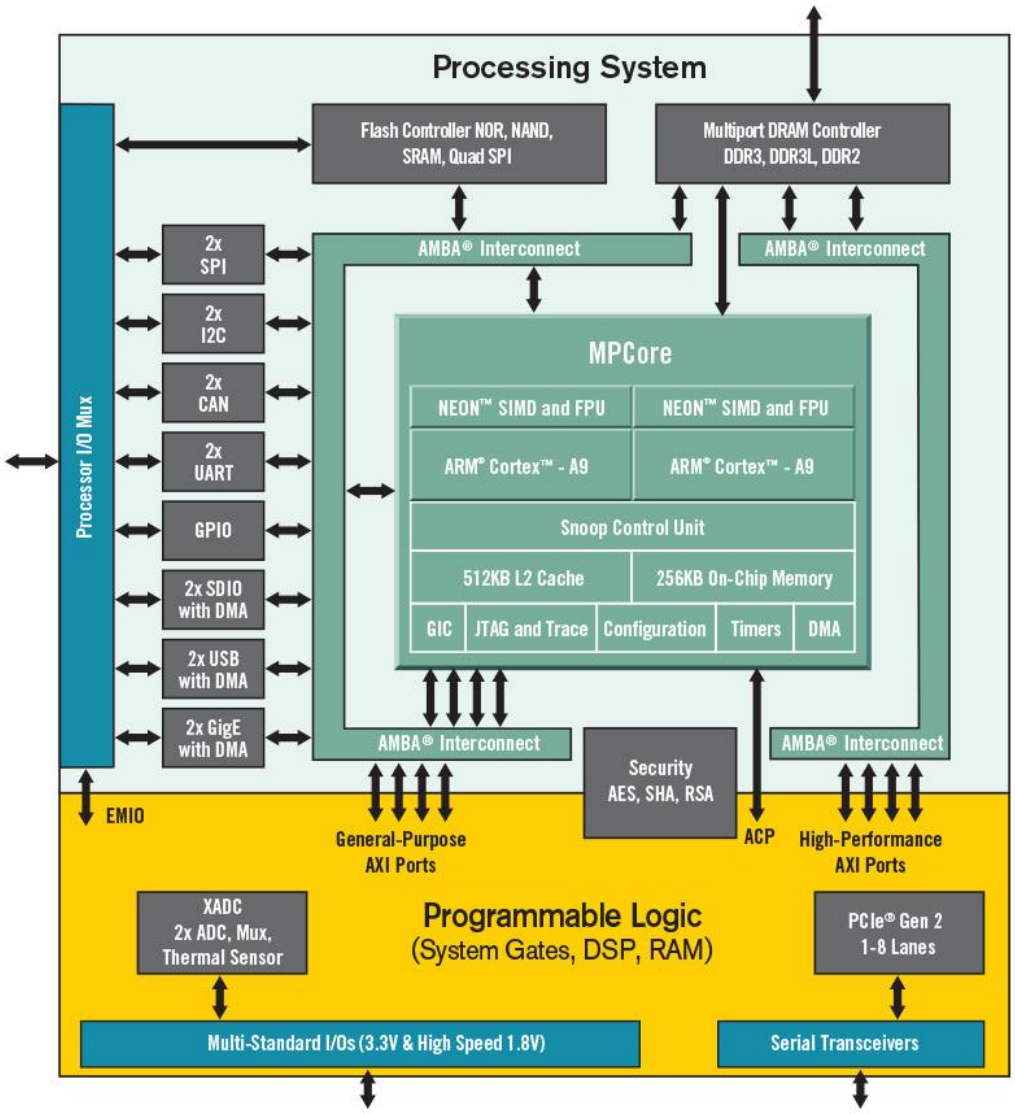

Figure 2.9 – Zynq-7000 SoC Architectural Overview

From the main characteristics it is known that the CPU maximum frequency is 667 MHz and the total amount of DDR3 memory is 512 Mb. The processor also has two USB 2.0, Gigabit Ethernet and other interfaces as the UART that allows the ZedBoard to interact with the Workstation.

The PL offers the designer the ability to implement their own custom logic which can work alongside the software running on the processor cores. The two areas of the device, PS and PL, are linked by a series of interfaces which adhere to the AXI4 interconnect standard. These interfaces allow the designer to implement

custom logic in the PL which can be connected to the PS and extend the range of peripherals which are available and visible in the processor's memory map.

This technology is used to create a block of custom logic in the PL, and then add control and status monitoring capabilities by using memory mapped registers which the processors can access via the AXI4 interconnect [7].

During the study of each receiver blocks will be evaluated the possibility to exploit this functionality and, eventually, exporting the block in hardware logic with the purpose to improve and speed up the implementation and the software.

CHAPTER 3

3 Physical Layer Design

The following chapter will explain in detail the functionalities of the physical layer. For each block, different possible designs are discussed evaluating the trade-off focusing on the implementation of the chosen one. Then simulation and performance results are presented.

3.1 RF AND MODULATION SYSTEMS

The Physical Layer is responsible to activate and deactivate the physical connection between the transmitter and the receiver. The subsystem modulates the CLTUs onto the RF carrier. As discussed in Chapter 2.4, PLOP-2 is chosen and the Carrier Modulation Modes (CMM) are reported in the following diagram

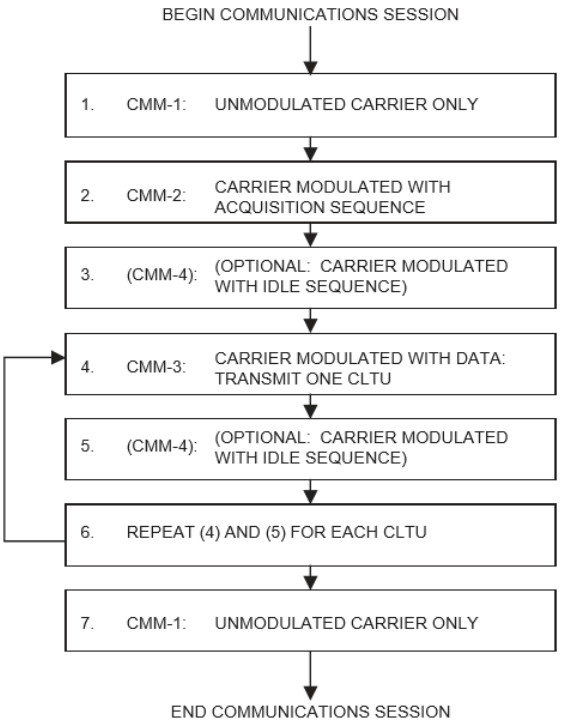

Figure 3.1 – Carrier Modulation Modes (PLOP-2)

The CMMs consist of different states of data modulation which creates the physical telecommand channel.

With PLOP-2, CMM-1 is in charge of establishing and maintaining the RF connection so, in this state no telecommand modulation is present. CMM-2 is concerned to transmit the Acquisition Sequence to enable the receiving end to acquire bit synchronization. CMM-3 deals with the transmission of one CLTU while CMM-4 takes care to send the Idle Sequence between CLTUs in order to maintain the modulation part of the physical connection.

In the next Sections are explained the blocks that makes up the physical layer (Figure 2.4). For each block is shown the possible designs evaluating the trade-off focusing on the implementation of the chosen one. Then simulation and performance results are presented.

3.2 AUTOMATIC GAIN CONTROL

In order to adjust the receiver signal strength to a certain desired value an Automatic Gain Control (AGC) block is implemented. The AGC should process the signal and provide a feedback to the RF chain. If the input power is high, it can decide to reduce the signal strength to minimize the distortion due to nonlinearity or, if the input power is low, it can increase the gain to reduce the noise figure and boost the SNR.

Since this project concerns the digital baseband processing part, the AGC is implemented as first block in the chain with the purpose to bring the signal to a reference value.

3.2.1 Block Design and Trade-off analysis

The response time plays an important role when designing the AGC. There is usually a compromise between having the loop that replies quickly to undesired input level fluctuations vs. having it undesirably modify amplitude modulation on the signal. The reason is to be sought in the fact that large and abrupt changes in the input level of the signal may lead to unacceptable recovery behavior.

In some transponders, the AGC is designed to keep constant the average of the absolute value of the voltage at the AGC output. In other transponders, the AGC is designed to keep constant the Root Mean Square (RMS) voltage at the AGC output [8].

The following block diagram reports the first of the two possible solutions

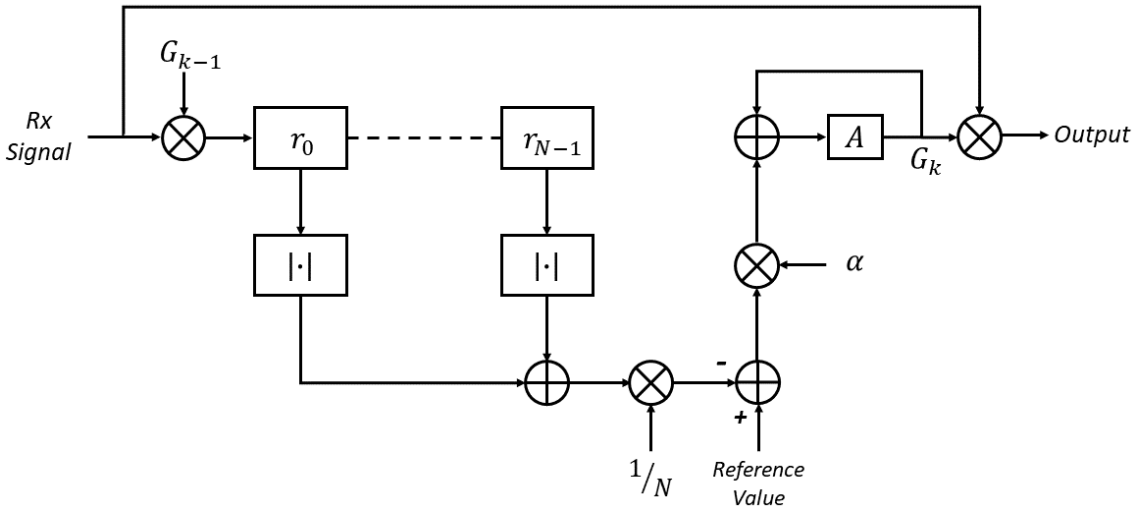

Figure 3.2 – AGC Absolute value block diagram

In the diagram above a recursive algorithm for updating the AGC value (G) is applied:

$$G_{k+1} = G_k + \alpha\,(ref\ - E\{|r|\})$$

where α is the updating step size, *ref* is the reference value to which the average absolute value of the signal must be brought and $E\{|r|\}$ is the average over N samples of the Rx signal. The *output* is the multiplication of each new received sample by the AGC Gain value. For both the AGC it is supposed that the Gain is equal for both real and imaginary part of the signal.

The second AGC computes the average RMS level of the signal and then divides the signal by this value. The result is obtained using a single-pole IIR filter:

$$p_{k+1} = (1-\alpha)p_k + \alpha|x_k|^2 \ , \qquad r_k = \sqrt{p_k}$$

where x_k is the input signal, p_k is the averaged power and r_k is the RMS. The diagram in Figure 3.3 summarizing the block implementation

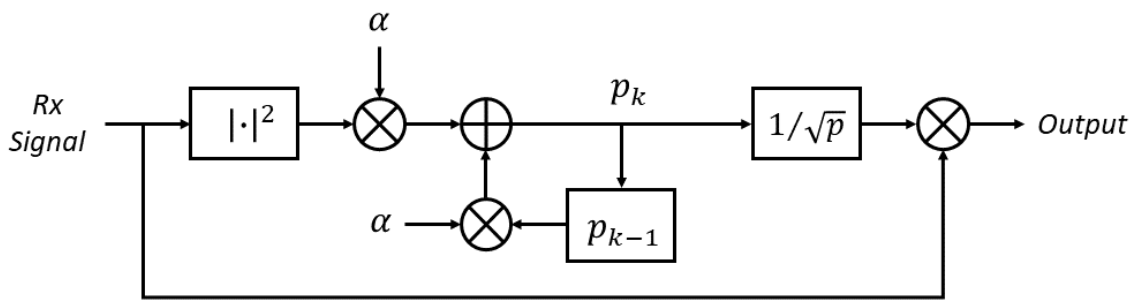

Figure 3.3 – AGC RMS block diagram

The filter has a single design parameter, which is the decay value α and the response is analogous to the response of an electronic low-pass filter. The decay value α is related to the time constant τ. For example, with $\alpha = 0.01$, the time constant $\tau = -\frac{1}{\ln(1-\alpha)} \approx 99.5$. Since the sampling frequency $f_s = 48\ kHz$, the time constant is around 2 ms as it is possible to see in the figure below

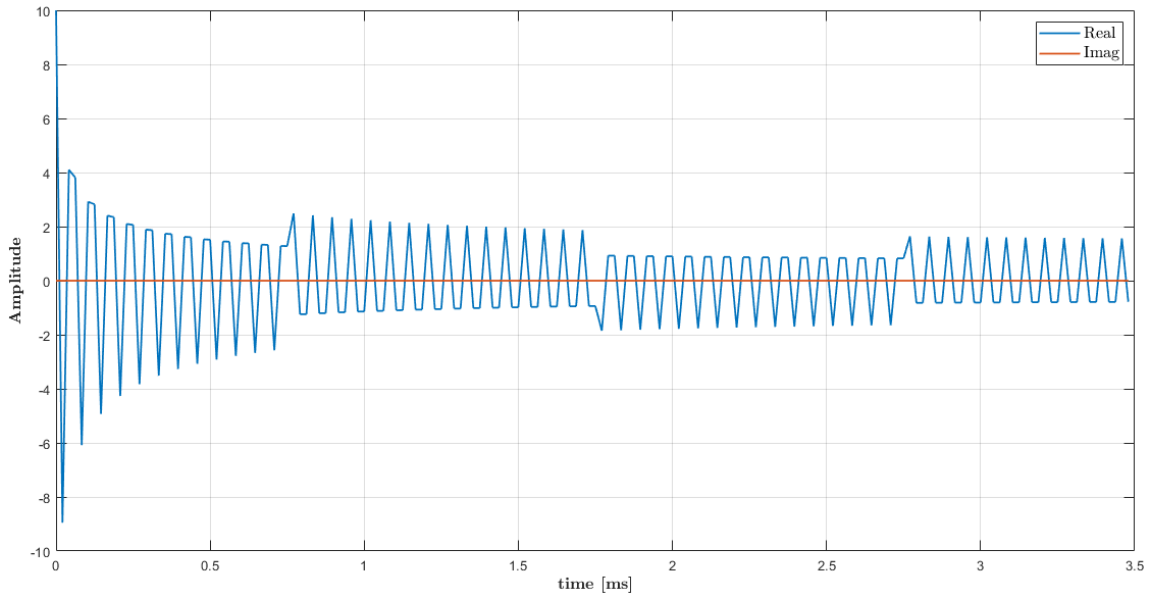

Figure 3.4 – Effect of AGC RMS to Rx Signal

In the next subsection, a choice on the ACG type is done evaluating the performance in a real time implementation of the blocks.

3.2.2 Simulation results

Exploiting a real signal, simulations of the two different implementations are done.

For what concern the first design, it is necessary to reach a compromise on how many samples are taken for the average. Since, from specification, the samples per symbol are 48, for the received signal mean are taken $N = 256$ samples that results in a reasonable trade-off because it means that the signal is observed on an interval of around *0.5 seconds*.

In Figure 3.5 is reported the trend of the G parameter at different step size α. The simulations are done simulating once the AWGN samples and adding them each time to the same portion of the Rx signal. α is taken from 10^{-6} up to 10^{-2} since other values do not give interesting results.

As it is easy to see, the variance of the AGC gain parameter arises as α increases. However, higher values of α lead to a faster convergence of the results, in fact, for $\alpha = 10^{-6}$ it is hard to see where the value converges after 1 second. When the parameter increases so much, the system diverges (for $\alpha = 10^{-6}$ AGC gain goes to $-\infty$).

The good compromise is to take the value around 10^{-4} where the variance of the result is small and the convergence is quite fast

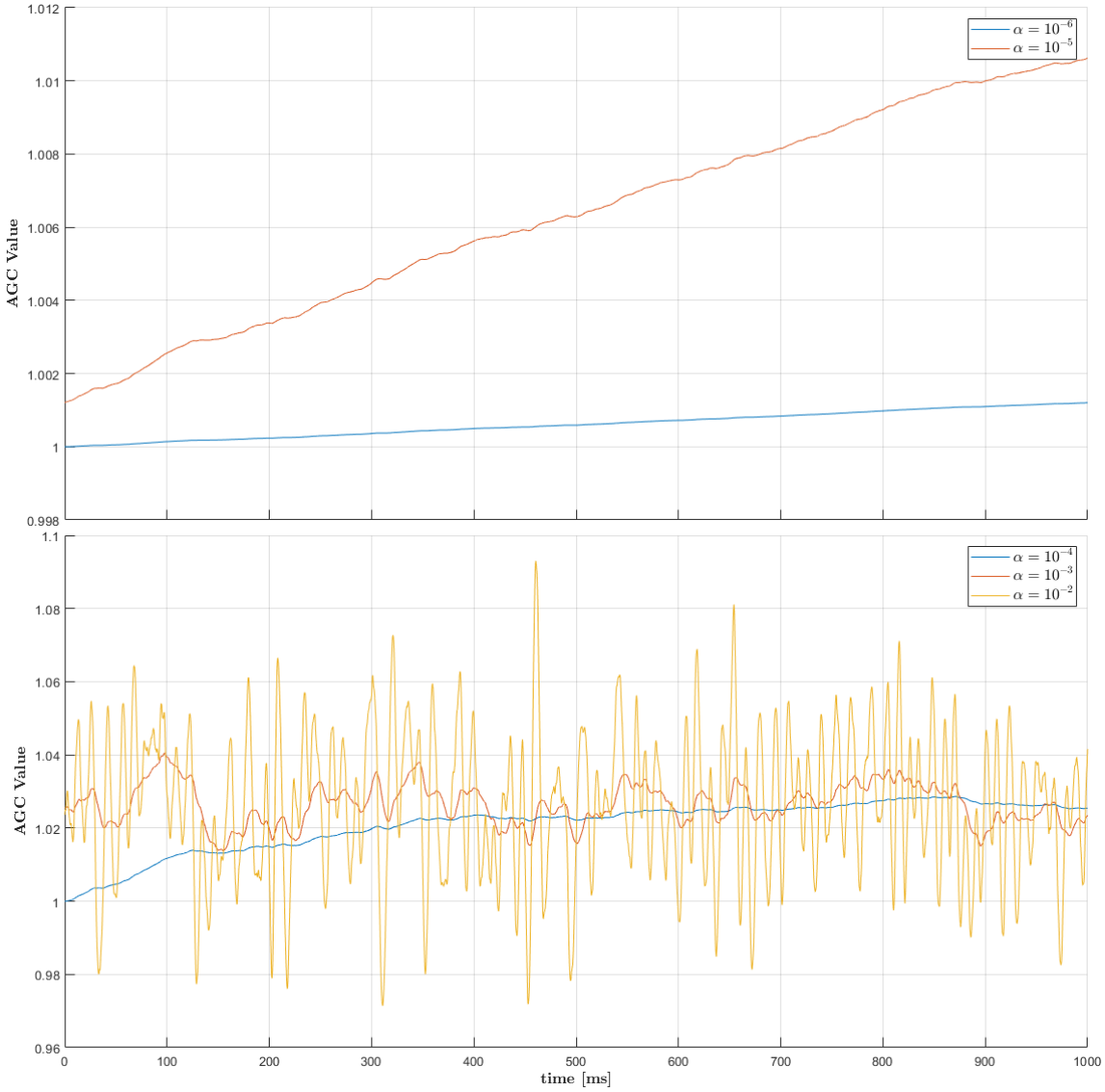

Figure 3.5 – AGC Gain Value for different α (AGC absolute value)

The same simulations made for the first AGC are performed for the second one (same conditions hold).

Also in this case is possible to see the trend of the Gain at the various step size. The variance of the curves is almost equal to the previous case and the choice of the α parameter does not change

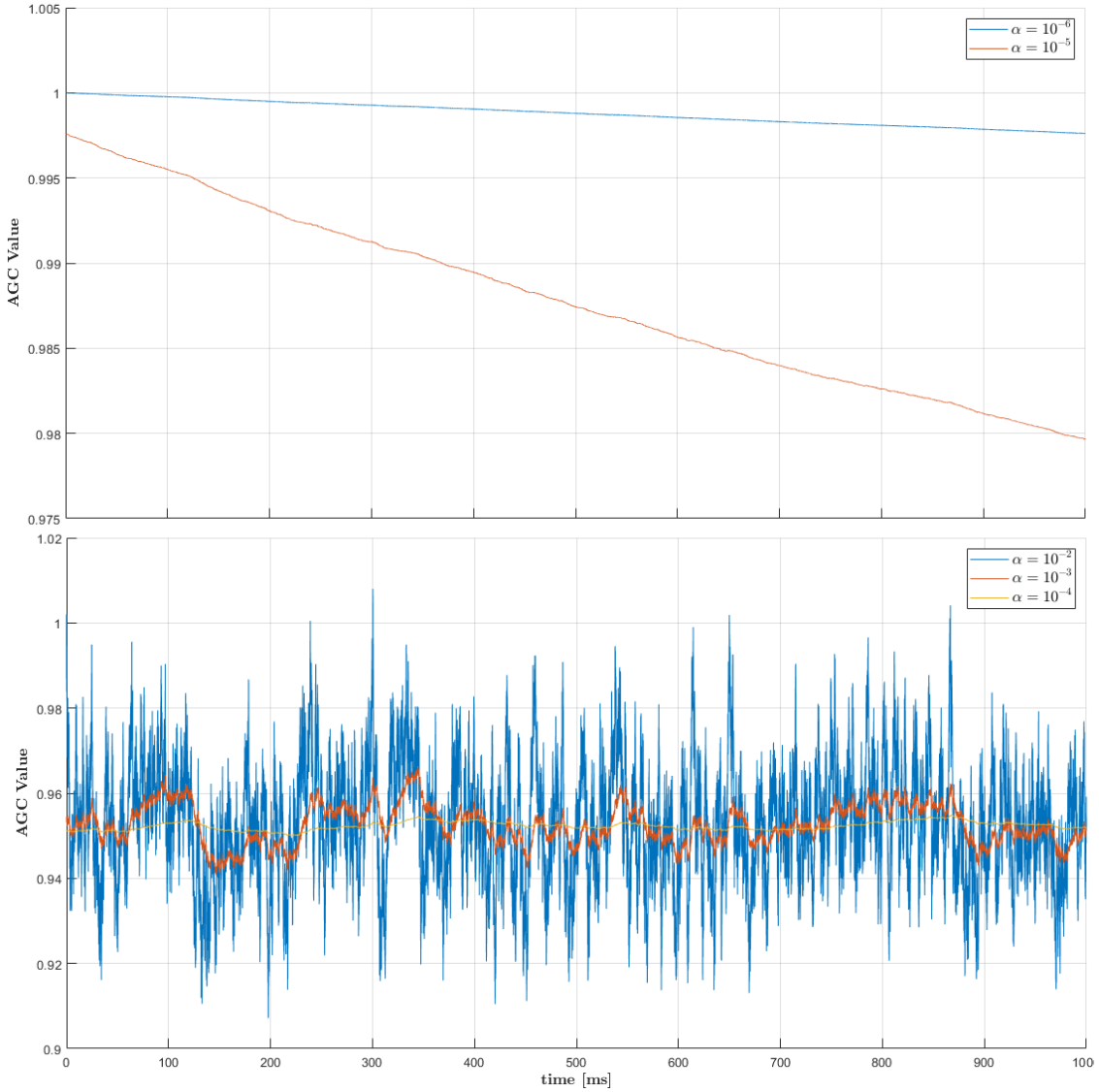

Figure 3.6 – AGC Gain Value for different α (AGC RMS)

Both AGCs are implemented in C language and executed, as already anticipated, on the Xilinx board.

The results, changing the parameter α, are as expected. For what concerned the performance in terms of clock cycles needed to perform the operation of updating the gain parameter, obviously the first AGC performs worse.

3.2.3 Optimization and Performance results

An optimization is adopted regarding the calculation of averaging N samples. Instead of iterating every time on all the considered samples, an accumulator that contains the average value is initialized and, when a new sample enters, it is added to the accumulator (averaged on N samples) while the oldest one is removed from it. This avoid looping always on the vector containing the incoming signal and the clock cycles needed to perform the operation is consistently reduced. The performances are reported below

Table 3.1 – AGC Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
AGC 1	*647*	*1.94*	*48000*	*9.31*
AGC 1 mod.	*131*	*0.39*	*48000*	*1.87*
AGC 2	*80*	*0.24*	*48000*	*1.15*

In the table are reported, for each implemented AGC, the clock cycles needed from the instant in which a new sample enters in the loop to the time in which the AGC gain value is given. For all the performance analysis, from now on, a value of CPU utilization (in percentage), of the Zynq-7000, is calculated, considering the number of time that the operation is done in one second. Since the sampling frequency is *48 kHz*, and the AGC calculates the gain every new sample, the operation is repeated *48000* times per second.

As it can be noticed from Table 3.1, the performance of the modified version of the first AGC (*AGC 1 mod.*) is comparable with the second AGC implementation (*AGC 2*), but it is still worse in terms of computational needs. For this reason,

AGC 2 is chosen for the final implementation. Just one note on the choice of α must be done since, in the real implementation, it is derived that a slightly different value must be used to reach the same result as in simulation environment.

3.3 SUB-CARRIER SYNCHRONIZER

In order to remove the presence of the sub-carrier, the baseband signal is simply multiplied with a local oscillator that (ideally) generates a pure single frequency sine wave.

The PSD of the signal, in which the presence of the subcarrier is evident, is reported in the following picture

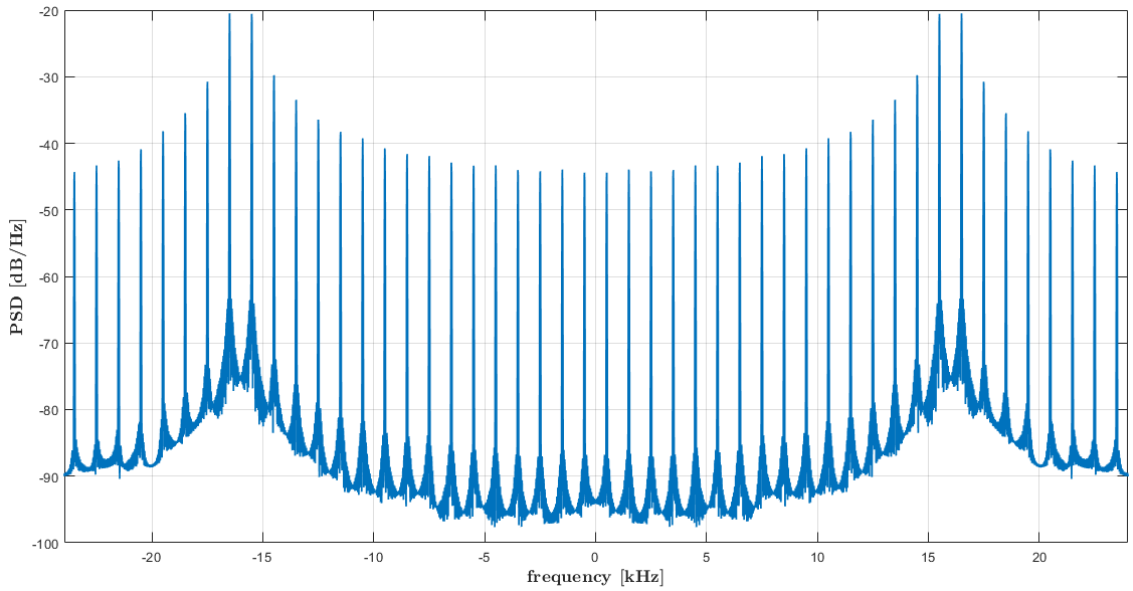

Figure 3.7 – PSD of Rx Signal

The real process leads to the impairments described before that must be recovered. The phase of the sub-carrier generated by the local oscillator must be aligned to the phase of the incoming signal and the phase noise makes the job more difficult.

If the oscillator was ideal, its frequency response could be represented with a Dirac function at the carrier frequency f_c, but the real one presents a kind of bell due to the phase noise profile

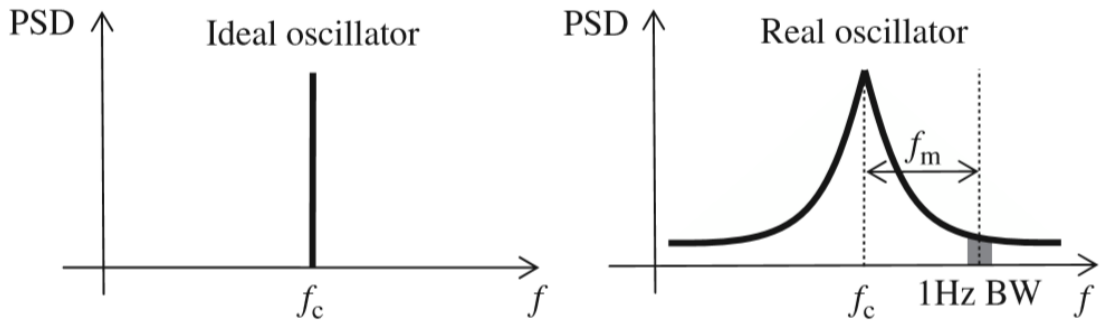

Figure 3.8 – Ideal (Left) vs Real (Right) Oscillator frequency response

The phase noise $\theta(t)$ is described in the frequency domain by its PSD in dBc/Hz. It is the ratio between the noise power measured in 1 Hz bandwidth, at a frequency offset f_m, and the power of the carrier [9].

3.3.1 Design and Trade-off analysis

The best estimate of the unknown phase θ based on the observation, over an interval, is the phase maximizing the log-likelihood function (ML). The use of the open loop ML estimator suffers from two main drawbacks. The first one is that the phase estimate is available only at the end of the observation interval. This leads to the second drawbacks where the data received in this interval have to be stored in order to postpone every decision about them.

The solution is to use an iterative procedure. Assume that at the end of the *k*-th carrier period an estimation $\hat{\theta}_k$ of the carrier phase is available, so it can be used as starting value to be updated according to the received signal observation in the next carrier period. Since maximizing the log-likelihood ratio is equivalent to require that its derivative be zero, it can be conditioned by the value $\hat{\theta}_k$ obtained

in the previous interval. At the end of the training phase it is reasonable to assume that $\hat{\theta}_k \approx \theta$. Its sign gives information on whether the estimation is smaller or greater than the real phase value.

The recursive algorithm adopted to update the estimation is:

$$\hat{\theta}_{k+1} = \hat{\theta}_k - \alpha_k E\{e(\hat{\theta}_k)\}$$

The new phase estimation is equal to the old one minus a coefficient that multiply the average of the error. The ensemble average $E\{\cdot\}$ can be substituted by a time average.

A practical implementation can be obtained through the Phase Locked Loop (PLL). The adopted solution is reported in Figure 3.9

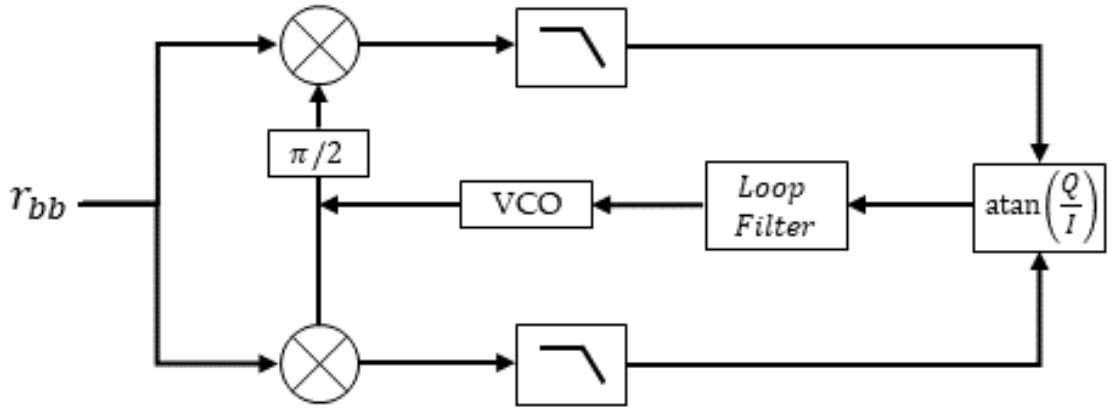

Figure 3.9 – PLL Block Diagram

What is represented in the block diagram is that the received signal in baseband is multiplied by a signal at the subcarrier frequency, generated by the Virtual Controlled Oscillator (VCO), plus a phase offset given by the feedback loop. The loop calculates the instantaneous phase of the signal x_k as:

$$\theta_k = tan^{-1}\left(\frac{Im(x_k)}{Re(x_k)}\right)$$

where *Re* and *Im* are the real and imaginary components of the signal respectively. This value is used like an error $e(\hat{\theta}_k)$ to change the frequency of the VCO and, if the Loop Filter is not present, the PLL is of the First Order (the error enters directly in the VCO). After the multiplication by the subcarrier, the components are filtered in order to remove residual high frequency terms and for the reasons reported in Section 3.4.

The choice of the discriminator is a trade-off between performance of the PLL and required hardware resources, as well as complexity [10].

The optimum compromise is to use as discriminator the error signal reported in the formula above. Looking at the Figure 3.10 (left), considering a constellation point in the complex domain, the angle $\hat{\theta}$ can be estimated as described

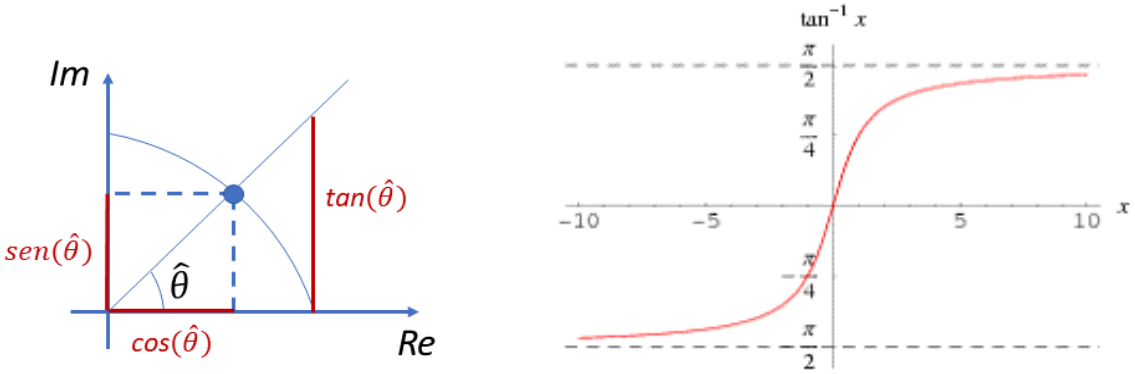

Figure 3.10 – PLL Discriminator

The ratio between Real and Imaginary part is the $tan\,(\hat{\theta})$ so the angle is represented by the $tan^{-1}(\cdot)$. Moreover, the ratio allows to remove the data dependency since they are present in both parts. The output of the discriminator is a limited function, Figure 3.10 (right), between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, so it is possible to have the control of the VCO input.

The PLL can be of different orders. As already anticipated, if the error signal is input to the VCO without further filtering, the PLL is of the first order. The block diagrams are shown below

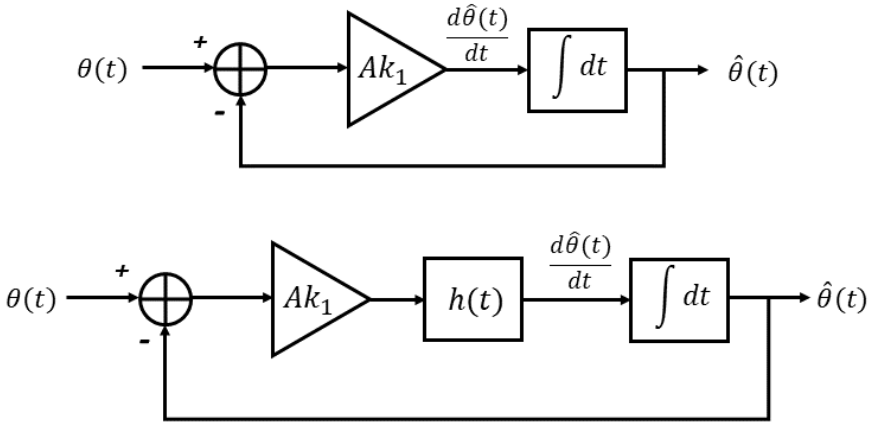

Figure 3.11 – First Order PLL (up) – Second Order PLL (down)

The transfer function of the first order PLL is:

$$H_{eq}(s) = \frac{Ak_1}{s + Ak_1}$$

where A and k_1 are the gains of the loop.

The drawback of the first order PLL is that residual offset in phase will be a function of the offset in frequency

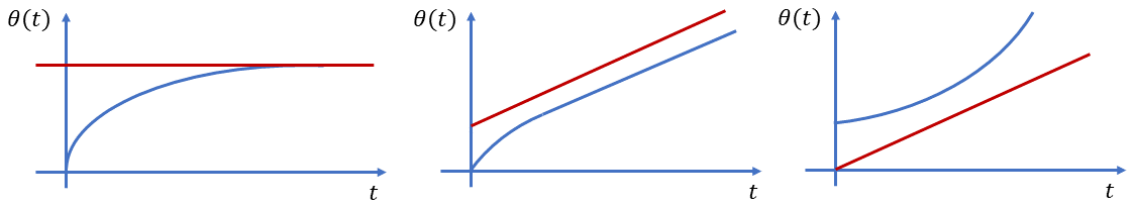

Figure 3.12 – Steady-state errors of first order PLL

As is possible to see, the PLL is able to completely recover the phase offset with the response in Figure 3.12 (left). If it is present a constant frequency offset, it

translates in a constant phase offset (center). Finally, if a non-constant frequency offset is present, the PLL phase error goes to infinite (right).

For this reason in this project is implemented a second order PLL that is not only able to track the phase, but also to track the frequency offset. If the frequency offset is non-constant, the phase is recovered with a constant residual error.

In the following table are summarized the steady-state errors of a PLL

Table 3.2 – Steady-state errors of a PLL

Errors	1-st order PLL	2-nd order PLL	3-rd order PLL
Phase Offset	0	0	0
Constant Freq. Offset	Constant	0	0
Non-Constant Freq. Offset	∞	Constant	0

The case of a third order PLL is reported, but it is not implemented since the complexity of the system grows. Moreover, increasing the tracking capability of the PLL lead to put more noise in the system since the equivalent bandwidth of the PLL will increase. Lastly, a fine phase recovery loop of the first order is inserted in the block diagram of the receiver and this translates in increasing the order loop bandwidth.

From Figure 3.11 is possible to describe the system with the following transfer function:

$$H_{eq}(s) = \frac{4\varsigma\pi f_n s + (2\pi f_n)^2}{s^2 + 4\varsigma\pi f_{ns} + (2\pi f_n)^2}$$

where $f_n = \frac{1}{2\pi}\sqrt{\frac{Ak_1}{\tau_1}}$ is the natural frequency, $\varsigma = \frac{\tau_2}{2}\sqrt{\frac{Ak_1}{\tau_1}}$ is the damping factor and $H(s) = \frac{1+s\tau_2}{s\tau_2}$ is the frequency response of the filter.

These are the three most important parameters in a PLL design and they will be analyzed in detail later.

In order to convert the model from continuos to discrete time is possible to use the Bilinear Transform [11] that allows to easily substitute to the variable *s* the following approximation:

$$s \leftarrow \frac{2}{T}\frac{z-1}{z+1}$$

where *T* is the sampling frequency. The result, after some algebric operations, is:

$$H_{eq}[z] = \frac{\left(\frac{4}{T}\varsigma\omega_n + \omega_n^2\right)z^2 + 2\omega_n^2 z + \omega_n^2 - \frac{4}{T}\varsigma\omega_n}{\left(\frac{4}{T^2} + \frac{4}{T}\varsigma\omega_n + \omega_n^2\right)z^2 + \left(2\omega_n^2 - \frac{8}{T^2}\right)z + \frac{4}{T^2} - \frac{4}{T}\varsigma\omega_n + \omega_n^2}$$

Consequently, because $H_{eq}(s)$ is a second-order analog filter, $H_{eq}[z]$ is a second-order digital filter. From the formula above it is possible to represent the system in the following form:

$$H_{eq}[z] = \frac{b_0 + b_1 z^{-1} + b_0 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

where b_0, b_1 and b_2 are the feed-forward coefficients and a_1, a_2 are the feedback coefficients. The functional block diagram is reported in the figure below (z^{-1} represents a single sample delay)

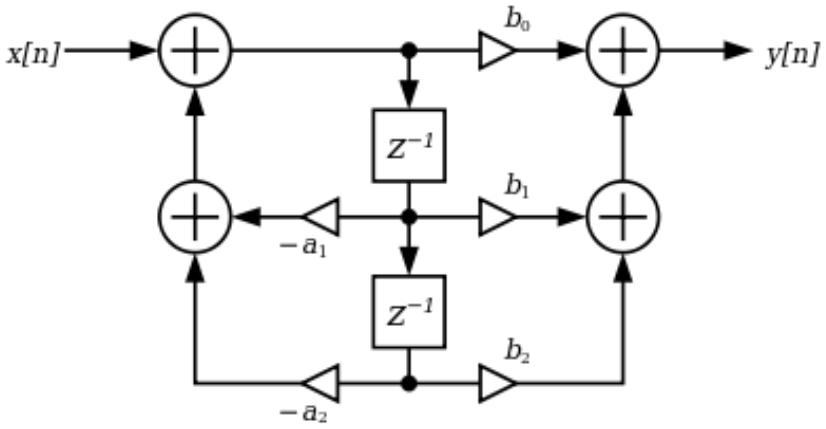

Figure 3.13 – IIR Biquad Filter

A final observation should be done on the design of the VCO. The signal, in practice, can be generated using the '*math.h*' library. The generated *sin* and *cos* have a high accuracy, but the process is quite expensive in terms of clock cycles (see 3.3.2). An alternative method is to use a Lookup Table (LUT) with a Numerically Controlled Oscillator (NCO). Internally, the NCO keeps track of the phase of the sine wave it produces, and it increments this phase at each sample point [12]. Obviously, losses in precision of the result are presents, but they can be alleviated by exploiting some tricks (saving the error computed at each iteration) and by choosing an acceptable dimension for the LUT. For each value of phase, three different values are sufficient to represent the sinusoid since the sampling frequency is three times the subcarrier frequency.

3.3.2 Simulation and Performance results

Considering the second order PLL (IIR Biquad Filter) and fixing the gain loop parameters, τ_1 and τ_2 is possible to verify the step response and the stability of the system exploiting MATLAB

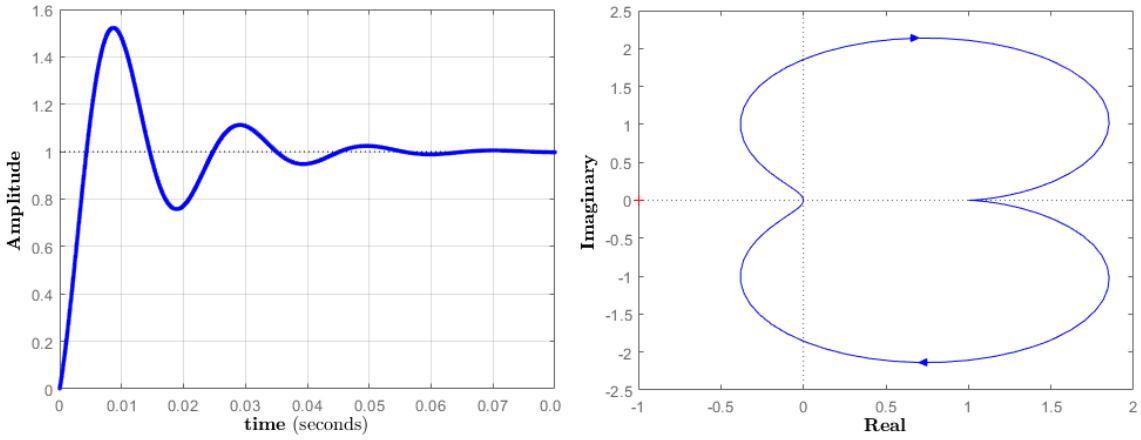

Figure 3.14 – Step response (Left) – Nyquist diagram (Right) with $\varsigma = 0.237$ and $\omega_n = 316.23$

In the first case, fixing the mentioned values, the result is the one reported in Figure 3.14. From the Nyquist diagram is possible to see that the system is stable

since the plot is far from the critical value *-1*. The overshoot in the step response is due to the low value of ς. Implementing the solution in the MATLAB simulation of the Receiver, the result reflects the study just done. The output phase in time is shown below

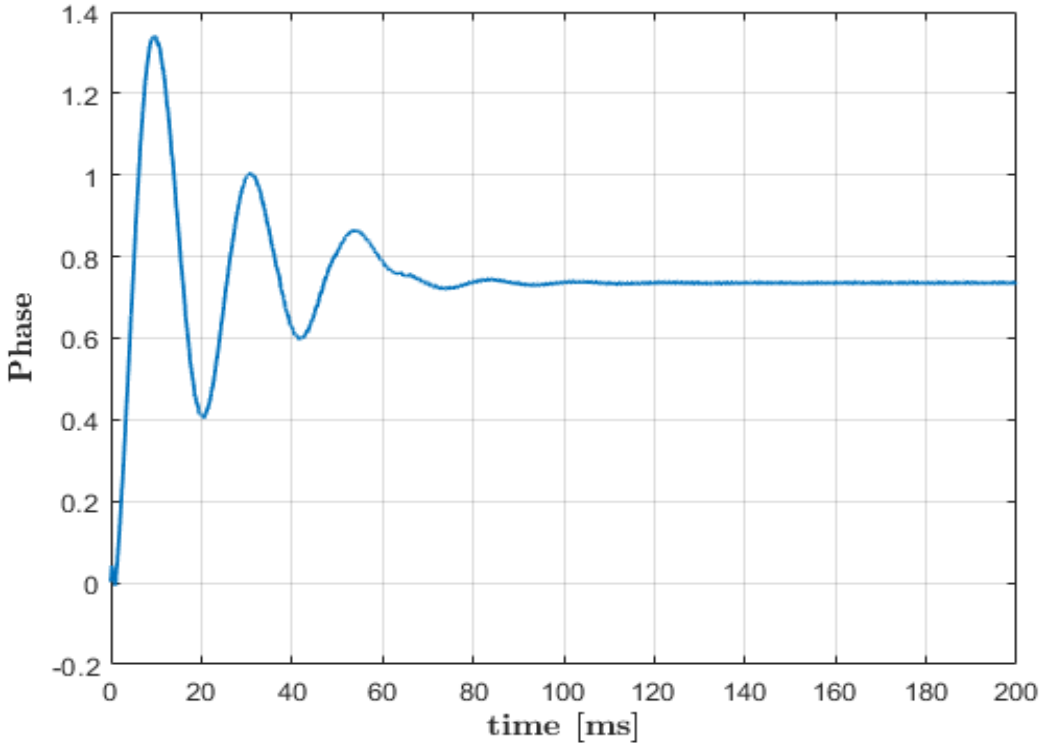

Figure 3.15 – Estimated Phase in Rx simulation – First test

The system can be brought to a slower convergence modifying the parameters. In the following test is possible to see the difference with the first case. The overshoot is smaller and the step response is slower since the dumping factor is slightly higher and the cutoff frequency is low

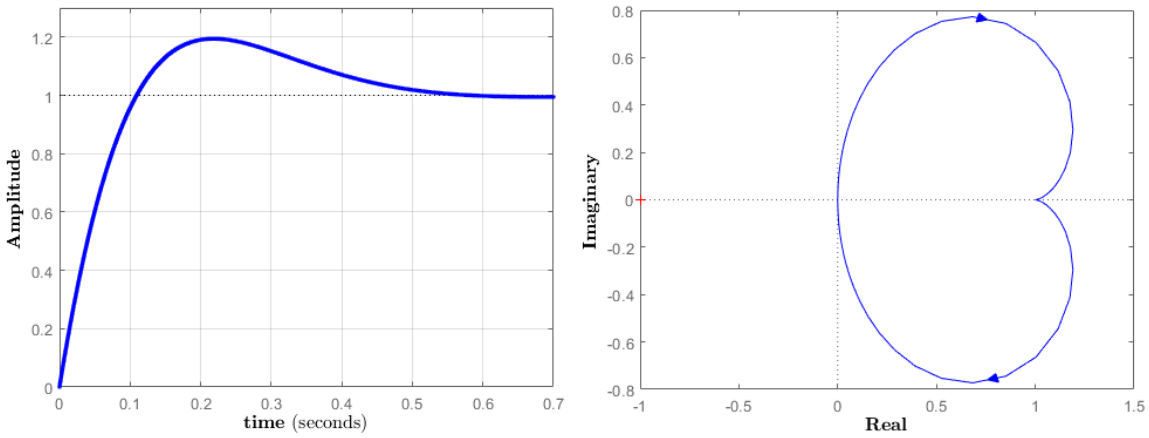

Figure 3.16 – Step response (Left) – Nyquist diagram (Right) with $\varsigma = 0.75$ and $\omega_n = 10$

Also in this case, verifying the solution in the Rx simulation, the phase has the following trend

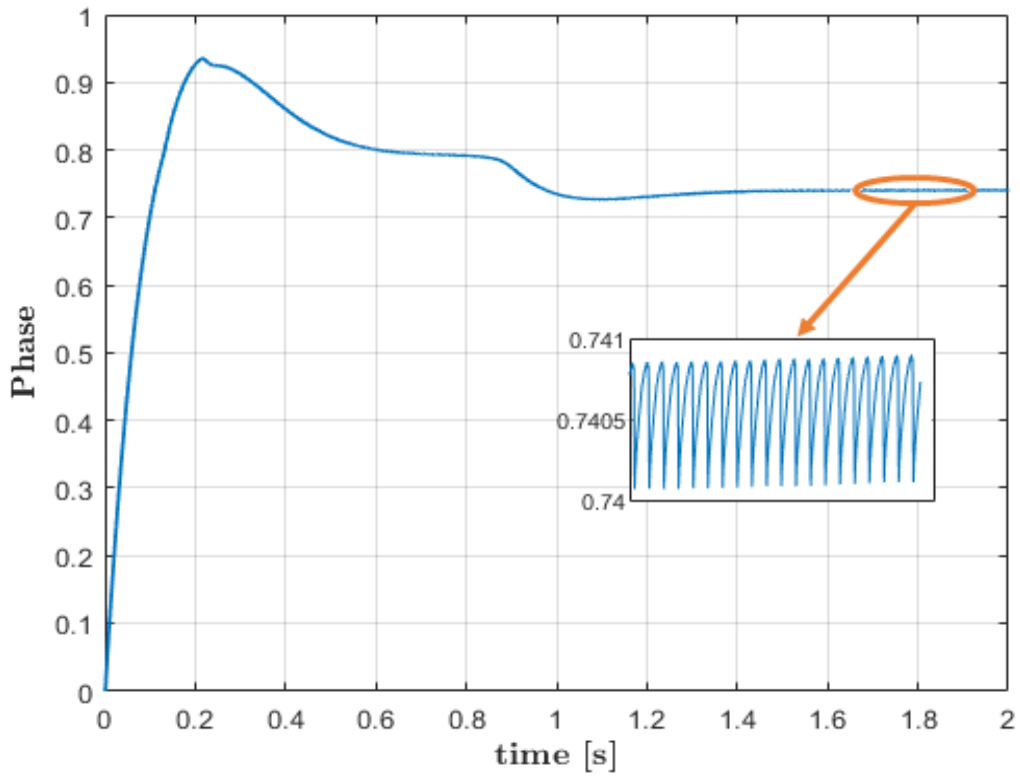

Figure 3.17 – Estimated Phase in Rx simulation – Second test

It is possible to notice that now, the system response is slower to converge to a constant value and it is only present one overshoot. In the box is reported a zoom of a portion of the plot where it can be seen that the phase value oscillates. This effect is due to the fact that the phase is recovered and the PLL is working correctly.

Porting the simulation in C language, the result is not the expected one since the error oscillates between its limit values and so, the PLL is not able to track the phase. The reason is that, when matched filtering is used, the subcarrier synchronizer must work at the optimum sampling instant, since it must know the period on which it must integrate the result. Moreover, for what concerns the choice of the loop bandwidth ω_n, the smaller it is, the smaller is the tracking error but the smaller is the frequency offset that the PLL can follow. For this reason, a good compromise is reached by increasing its value. The coefficients of the IIR filter are calculated in MATLAB fixing the sampling time to *1 ms* (duration of one symbol), the damping factor to $\frac{\sqrt{2}}{2}$ (that is the typical value) and $\omega_n = 110\ rad/s$. In Figure 3.18 is reported the corresponding Bode Diagram, that shows the frequency response of the system in magnitude and phase

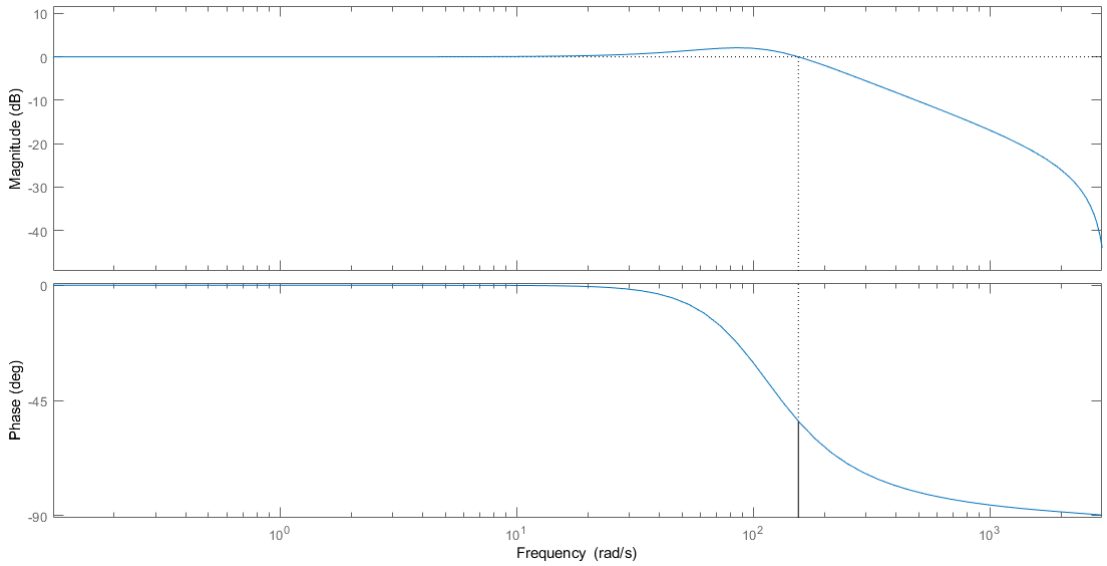

Figure 3.18 – Bode Diagram with $\varsigma = 0.707$, $\omega_n = 110$ and $T_s = 1\ ms$

The step response has an overshoot and the system converges to the final value quite fast

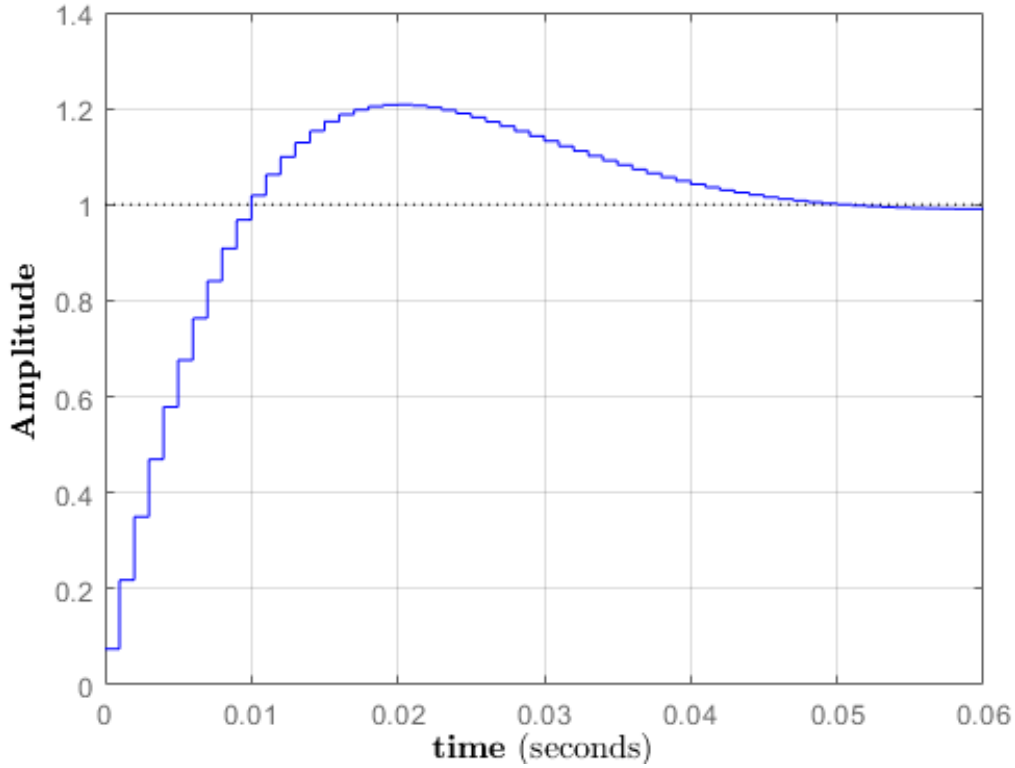

Figure 3.19 – Step response with $\varsigma = 0.707$, $\omega_n = 110$ and $T_s = 1\ ms$

An important parameter is the noise equivalent bandwidth of the PLL that, for a second order PLL and for this system, is:

$$B_{eq} = \pi f_n \sqrt{\varsigma + \frac{1}{4\varsigma}} = 56.64\ Hz$$

From experimental results it is derived that, as in theory, the larger is the noise equivalent bandwidth, the more is the additive noise in the system. However, if the bandwidth is large, more is the frequency offset that the loop can track.

A problem arises in the real implementation of the IIR filter. It is demonstrable that, systems containing non-linearities and feedback loops are always prone to autonomous oscillations or limit cycles also if floating-point numbers are used

[13]. To avoid the problem, the estimated phase is clipped from *0* to 2π, but another undesired effect is shown in Figure 3.20

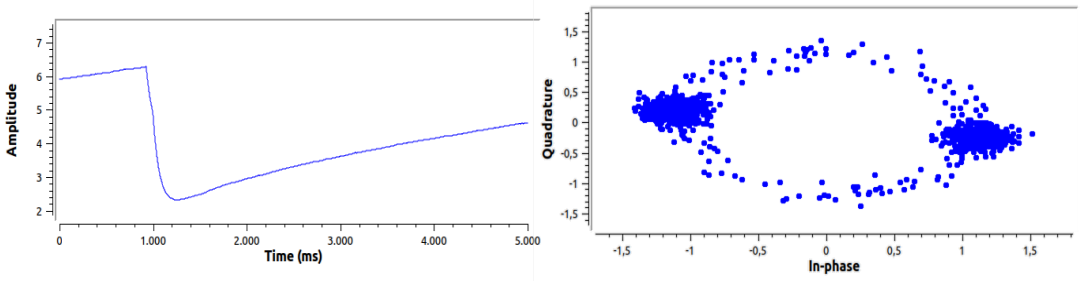

Figure 3.20 – Estimated Phase (Left) – Scattering Diagram after PLL (Right)

For these reasons, the final implementation of the PLL is slightly different from Figure 3.13, but the same study, just done, holds.

As already anticipated in the design subsection, an NCO is implemented. In the following table is shown the difference between removing the presence of the subcarrier multiplying the incoming signal by a signal generated by the '*math.h*' library and using a LUT

Table 3.3 – Subcarrier removal Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
math.h	*501*	*1.50*	*48000*	*7.20*
LUT	*134*	*0.40*	*48000*	*1.92*

As expected, the gain in using the Lookup Table is consistent. The trade-off to be reached is in the dimension of the LUT since, larger it is, better is the accuracy of the result, but higher is the occupied memory. Some tricks to reduce the size of the LUTs can be used as exploiting the same table for the sine and the cosine since they are related in some way.

The CPU utilization of the Sub-Carrier Synchronizer (to be summed with the value on Table 3.3) is very low. It is executed once per symbol and the performance is reported in the following table

Table 3.4 – Sub-Carrier Synchronizer Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Sub-Carrier Synchronizer	*212*	*0.64*	*1000*	*0.06*

Summarizing, the stability study plays the important role in the design of a PLL. More in general, some of the key pieces that must to be known are reported in the following list:

- The convergence's speed depends mainly on the damping factor, smaller it is, faster is the convergence of the PLL, but more overshoots are present;
- Smaller is the Loop Bandwidth, smaller is the tracking error, but also smaller is the frequency offset that the PLL can follow;
- Larger is the Noise equivalent bandwidth (it depends on the damping factor and on the PLL loop bandwidth), higher is the additive noise in the system;
- When matched filtering is used, the PLL must know the period on which integrates the result;
- When the PLL is locked, the estimated phase continuously oscillates in a small range;
- In the implementation of a PLL, cycle sleep are present and they must be solved changing a little bit the structure of the block.

3.4 FILTER

The Rx filter is a filter perfectly matching the one used at the Tx side. The filter used at the Tx is called shaping pulse, instead the one used at the Rx is called matched filter. They have the purpose to shape the spectrum by limiting the effective bandwidth of the signal, to reduce the effect of the noise. More in general, the system should not have Intersymbol Interference (ISI), so it must satisfy the Nyquist criterion. When multiple symbols are transmitted, the impulse response of the channel causes a transmitted symbol to be spread in the time domain and at the receiving end they result overlapped. Nyquist theorem translates the time domain condition in a frequency domain condition providing a method to construct band-limited functions.

Two different kind of filter are provided theoretically and they are implemented in practice showing in what they differ.

3.4.1 Design and Trade-off analysis

The first filter used at the Tx is a rectangular pulse with length equal to the symbol length (Figure 3.21)

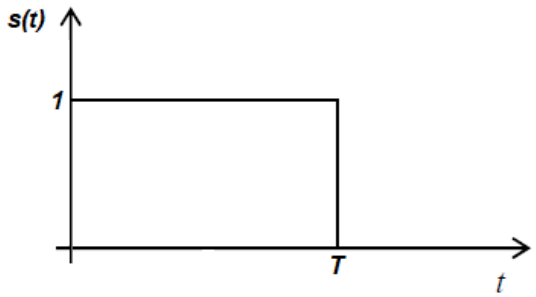

Figure 3.21 – Shaping pulse at the Tx in time

In order to match the filter at the Tx, a Moving Average (MA) filter is used at the Rx. It is a Low Pass Finite Impulse Response (FIR) filter that takes *N* samples of input at time, takes the average and produce a single output point

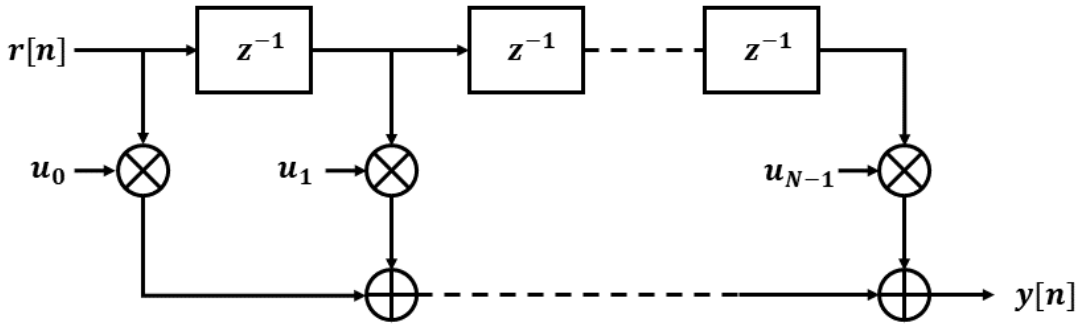

Figure 3.22 – FIR filter Block Diagram

In Figure 3.22, it is reported the typical representation of a FIR filter where u_n are the taps of the filter, $r[n]$ is the signal to be filter, z^{-1} represent a single delay line and $y[n]$ is the filtered signal. The output is described as the convolution between the input signal and the impulse response of the filter that, in the discrete domain, is:

$$y[n] = \sum_{k=0}^{k=N-1} r[n-k] - u[k]$$

Neglecting the noise, the output to the filter is reported in the following plot

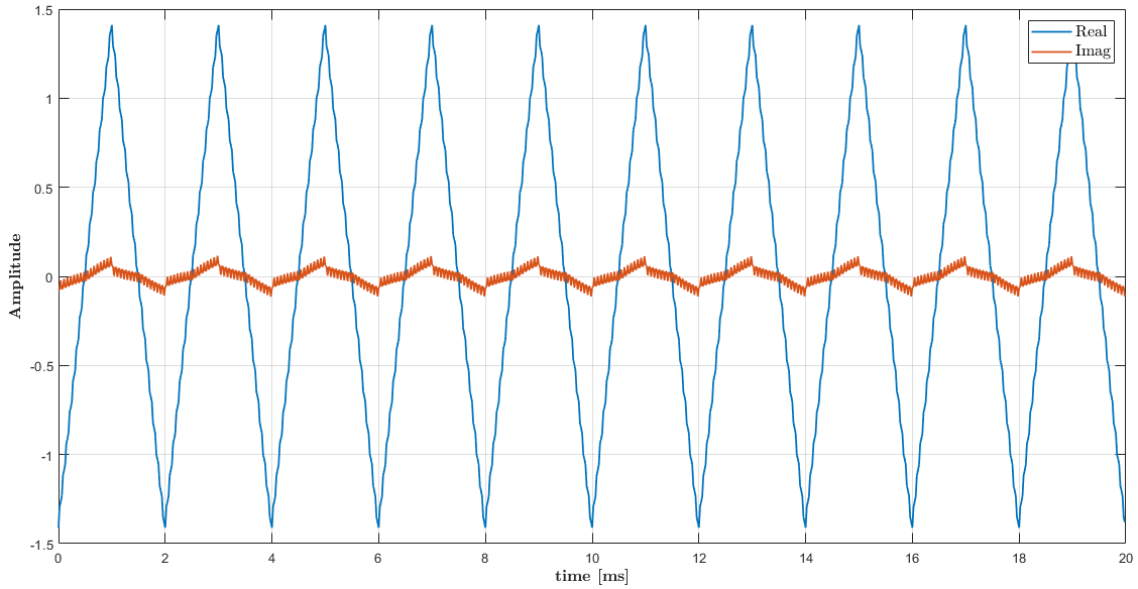

Figure 3.23 – Rx Signal filtered (MA) – No Noise

The second type of filter is the Square Root Raised Cosine (SRRC). It is a particular case of Nyquist filter and its impulse response is defined as follow:

$$h(t) = \frac{2\beta}{\pi\sqrt{T}} \frac{\cos\left[\frac{(1+\beta)\pi t}{T}\right] + \frac{\sin\left[(1-\beta)\pi t/T\right]}{4\beta t/T}}{1-(4\beta t/T)^2}$$

where T is the symbol time and β is the rolloff factor that is a measure of the excess bandwidth of the filter, i.e., the bandwidth occupied beyond the Nyquist bandwidth of $1/2T$ [14].

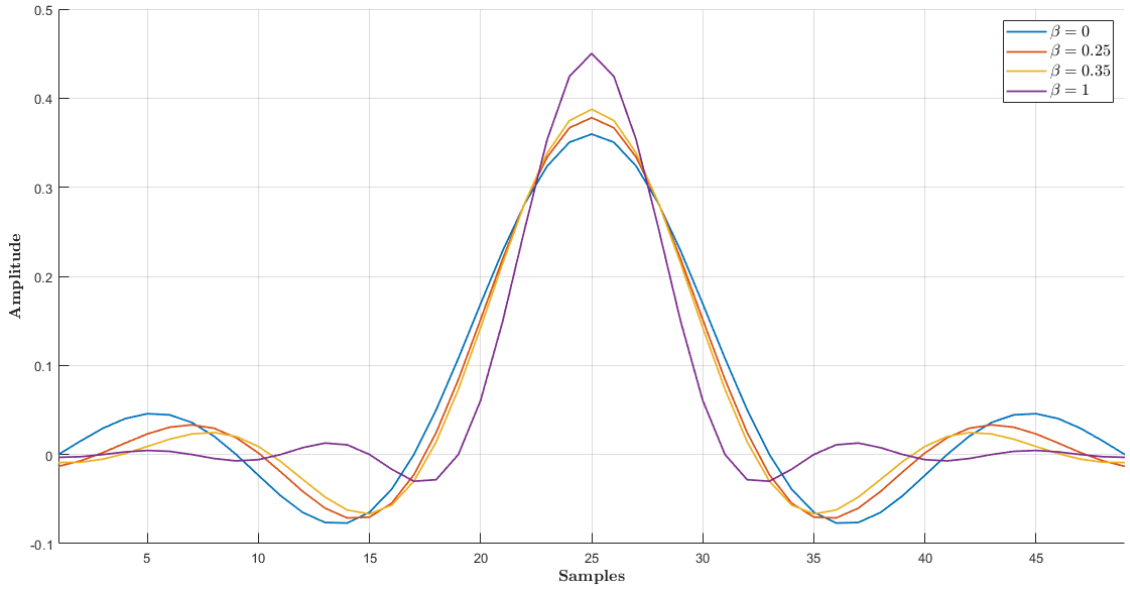

Figure 3.24 – SRRC Impulse Response at different roll-off

The impulse response at the different rolloff factor is reported in the plot above. The filter, theoretically, has an infinite number of taps so it has infinite attenuation in the stop band. However, in implementation, its length should be reduced to a finite value.

Smaller rolloff gives narrower bandwidth, however, its side lobes increase so attenuation in stop band is reduced. In other words, low values of β allow for a more efficient use of the spectrum but increase the ISI.

In Figure 3.24, the filter is represented using *8* sample per symbol and its response is cut at *6* symbols. This trade off is reached in order to compare the MA filter with the SRRC ones taking into account the same number of taps. In this way the complexity of the filtering stage is not increased and a reasonable number of side lobes is considered. From now on, all the following blocks must work at lower samples per symbol, so the resolution of the result is lower and a trade-off must be reached.

The downsampling of the Rx signal, at this stage, is simply done by taking one sample every six:

$$downsampling_{factor} = \frac{f_s}{f_{s,down}} = \frac{48000}{8000} = 6$$

Performance analysis are presented in the next subsection.

3.4.2 Simulation and Performance results

A useful tool for the evaluation of the combined effects of channel noise and ISI on the performance of a baseband pulse-transmission system is the Eye Diagram. It is an oscilloscope display in which a digital signal from a receiver is repetitively sampled and applied to the vertical input, while the data rate is used to trigger the horizontal sweep.

Several system performance measures can be derived by analyzing the display. An open eye pattern corresponds to minimal signal distortion. Distortion of the signal waveform due to ISI and noise appears as closure of the eye pattern [15]

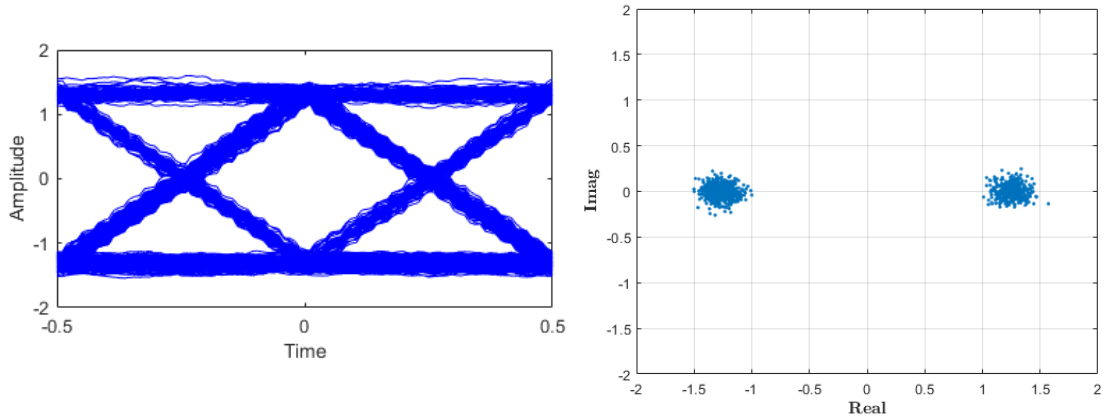

Figure 3.25 – Eye diagram with MA filter (Left) – Scattering Diagram (Right) - $E_b/N_0 = 25$

In the figure above is reported the eye diagram after the MA filter with a value of $E_b/N_0 = 25$. Also the scattering diagram at the same E_b/N_0 is shown in the figure (right).

The filtered signal with the SRRC filter is shown in Figure 3.26

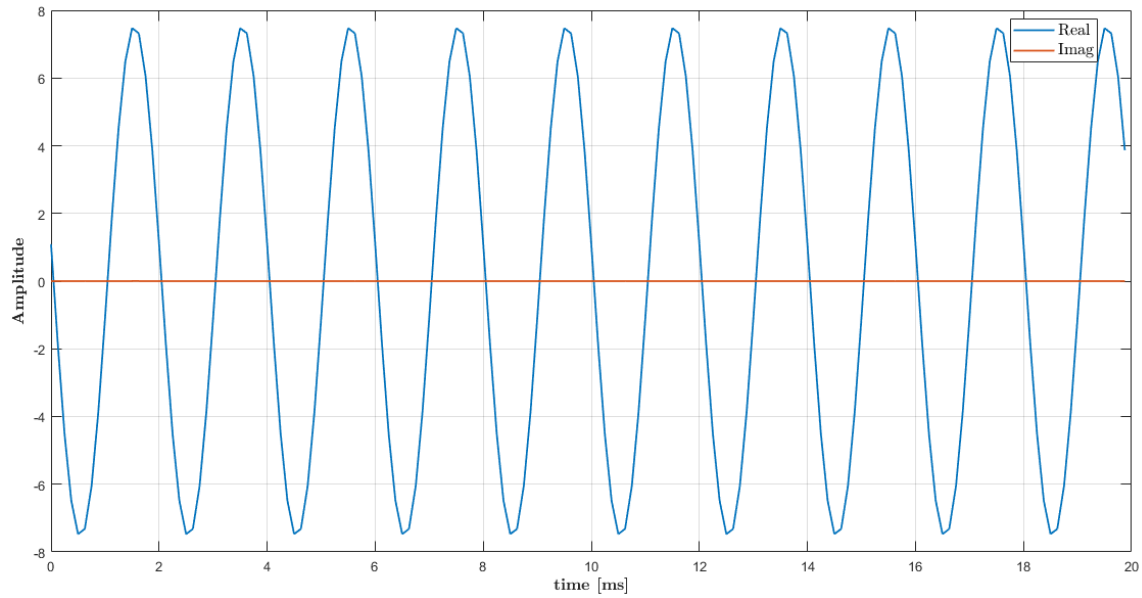

Figure 3.26 - Rx Signal filtered (SRRC, rolloff = *0.25*) – No Noise

A rolloff of *0.25* is used at the Tx and Rx sides. The difference between the MA filter is that the signal is rounded and the Subcarrier Synchronizer was able to completely recover the phase of the signal.

Using a large rolloff, the eye diagram is completely open, this means, if the sampling instant is the perfect one there is not ISI and a small sampling instant time error produce small ISI. Instead, if the rolloff used is small, a small sampling instant time error produces a large ISI.

In the implementation on the development board, the MA filter is implemented (SRRC filter implementation is demanded as Future Work) and the taps chosen are all equal in order to generate the impulse response of the filter in Figure 3.21. As already anticipated, one of the objectives of the project is to find some blocks that can be exported in hardware implementation. From performance analysis results that the FIR filter is very expensive in terms of CPU utilization since, every time one new sample of the signal enters in the system, a loop must be performed to multiply all the taps of the filter by the incoming signal.

The following table shows the gaining in CPU utilization between the two implementations

Table 3.5 – Filter Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Filter 1	*1888*	*5.67*	*48000*	*27.22*
Filter 2	603	1.81	48000	8.69

Filter 1 represents the software implementation of the filter, instead, Filter 2 is its hardware implementation that results in an improvement of about three times in terms of clock cycles needed to perform the filtering of one sample. The hardware implementation of the block will be discussed in detail in Chapter 6.5.

3.5 CLOCK RECOVERY

The ADC delimits the boundary between the analog and digital parts of the signal processing chain and it provides to the DSP block a sequence of samples at a certain rate (~*N* samples per symbol). Clock recovery is required to determine the optimum sampling instant in order to down-sample the signal to 1 sample per symbol.

Moreover, the clocks used to sample the signal are not ideal and introduce error on the sampling time which manifests itself as noise and can limit the system performance. This timing error is known as jitter and is expressed in seconds.

3.5.1 Design and Trade-off analysis

The optimum sampling instant is the one that maximizes the power of the discrete signal (independently from the data). This translates in finding the likelihood of the sampling time offset (τ):

$$\Lambda(\tau) = E[|r(nT + \tau)|^2]$$

The likelihood of τ is the average value of the power, of the received signal, sampled at time τ. In other words, the best possible sampling instant is when the output sequence, after sampling, has the maximum possible power.

This kind of Clock recovery is an Open Loop Estimation. In order to calculate τ it is needed a window of length *L* samples and so this block introduces delay in the processing.

Once the optimum sampling instant is found, it is needed an interpolation because it can be among two samples. This increases the complexity of the system, but it could not be necessary if there is not downsampling after the filtering stage.

There are different kind of interpolator, the less complex is the linear one that provides a very rough interpolation between samples

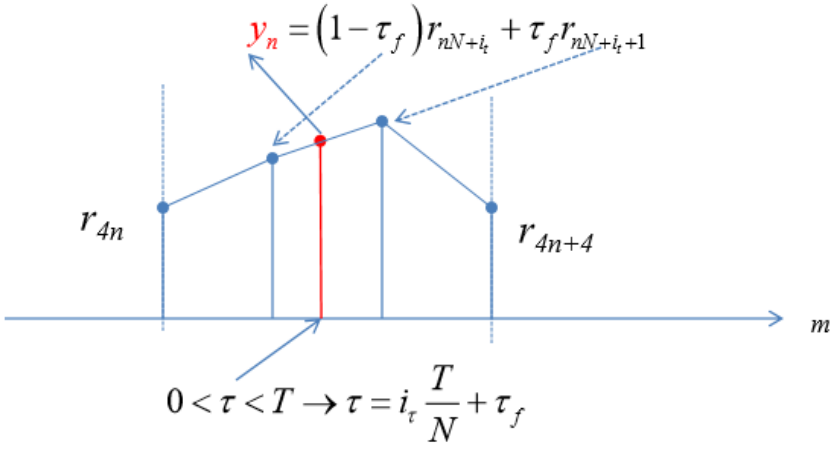

Figure 3.27 – Linear interpolation between 4 samples

In the figure above there is an example of the linear interpolator where 4 samples per symbols are taken into account.

About that, if the filter used at the receiver is the MA (it works with 48 samples per symbol) the error computed in using the nearest sample to τ is negligible.

In this project, the Open Loop Timing estimation is performed once in the receiver chain and when the symbol clock synchronization is lost. Every time the symbol lock is acquired, another kind of estimation is done and it is a Closed Loop Timing estimation.

This technique takes the derivative of the likelihood (derivative of the power of the observation with respect to τ):

$$\frac{d\Lambda(\tau)}{d\tau} = Re\left(r^*(kT+\tau)r'(kT+\tau)\right)$$

The derivative can be calculated replacing it with a finite difference:

$$r'(kT+\tau) = r(kT+\tau+T/2) - r(kT+\tau-T/2)$$

The loop is called Delay Locked Loop (DLL) and, in particular, the error calculated as above, that takes the sample at the actual time (z_{2k}), at time $\tau + \frac{T}{2}$ (z_{2k+1}) and $\tau - \frac{T}{2}$ (z_{2k-1}), is called Early late correlator.

The block diagram is reported in Figure 3.28

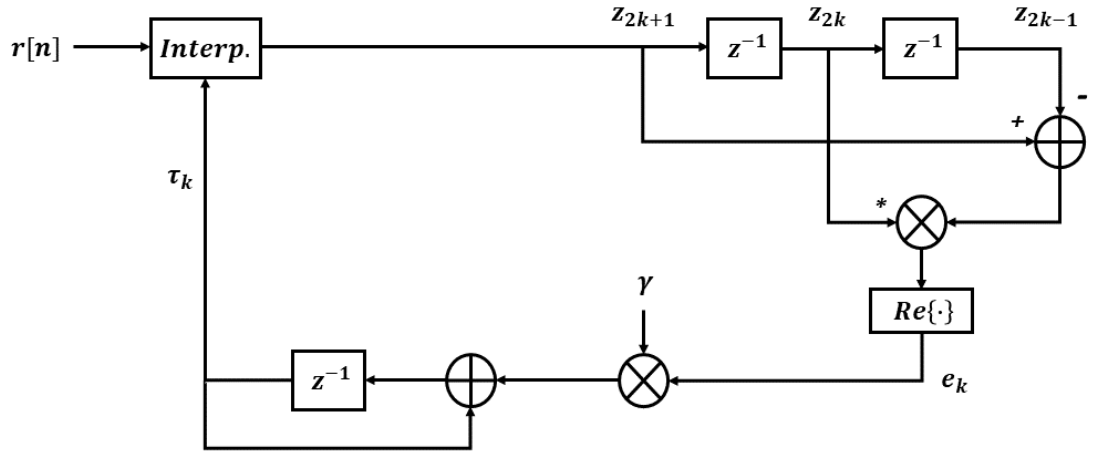

Figure 3.28 – Delay Locked Loop for Timing estimation

The circuit updates the error made in the calculation of the clock timing offset and only needs *2* samples per symbol. The initialization is done using the optimum sampling instant derived from the open loop. The error is computed by taking the product of the middle tap (complex conjugate) times the difference of the previous and the next. The error is used to update τ with a gradient algorithm:

$$\tau_{k+1} = \tau_k + \gamma e_k$$

This is a first order loop and the drawback is the error propagation.

Since it is not possible to have the sample at the time $\tau + \frac{T}{2}$, the derivative can be calculated delaying the correlator by $T/2$, obtaining the Gardner correlator that is the one implemented in the project.

3.5.2 Simulation and Performance results

There are not particular problems in the implementation of the Open Loop Timing on the Xilinx board. The only trade-off to be reached here is finding the right number of symbols that allows to find the optimum sampling instant.

It can be demonstrated that, with a number of symbols less than 50, some errors can appear if the signal-to-noise ratio is low. Taking 100 (or more) symbols gives a good estimation of the optimum sampling instant at which the incoming signal must be sampled. Obviously, the higher is the number of symbols taken into account, the higher is the delay introduced in the processing chain.

The prove that the Open Loop Timing provides the correct sampling instant is shown below

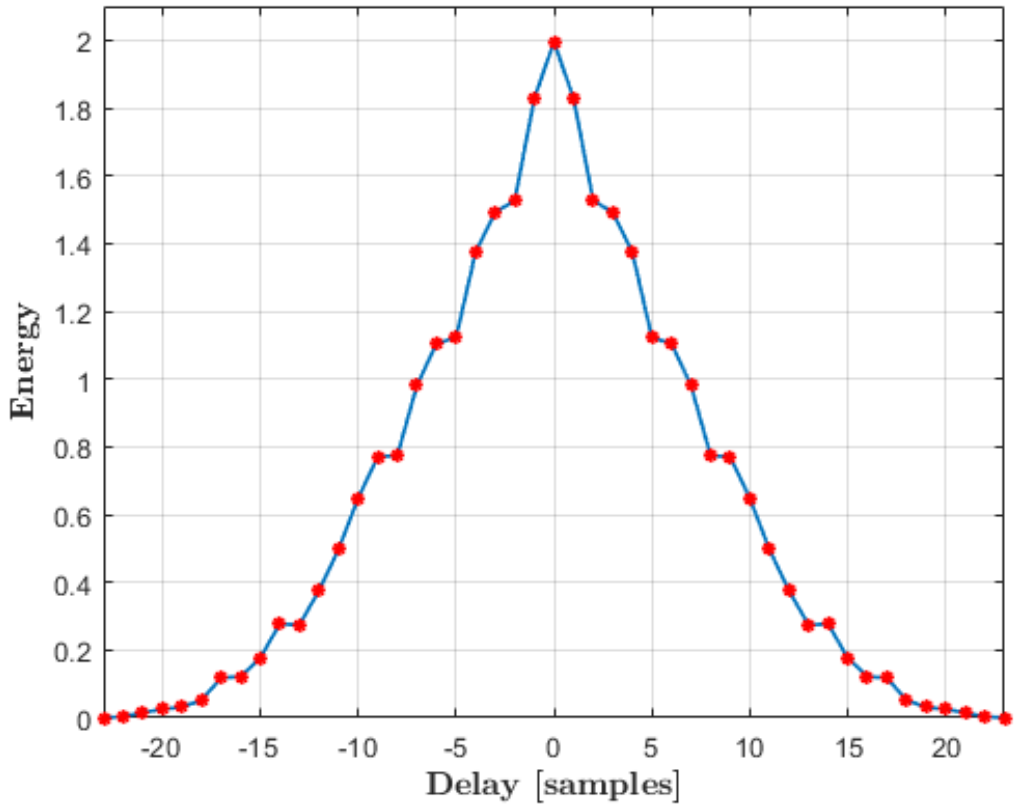

Figure 3.29 – Signal's Energy sampling at different τ

Figure 3.29 illustrates the signal's energy calculated sampling the signals at different sampling instants (shifting the signal by one sample each time).

The energy is calculated by squaring the real part of the filtered signal (since on the imaginary part, after the PLL, there is no more information) over a fixed period of time. The highest value of the energy is in the optimum sampling instant, when no further delay is added in the signal. Farther more, a bad effect caused by sampling at not the right τ can be seen in the scattering diagram. The two constellation points coming closer to each other gradually the delay increases and they collapse in a unique one when the signal is sampled with $\tau = \frac{N}{2} = 24\ samples$ (the energy is equal to zero).

In real implementation, after the Open Loop Timing estimation there is a Closed Loop implementation that is able to follow some variation in the incoming signal due to the different rates between the sample clocks of the transmitter and receiver.

The Channel Model in GNU Radio is able to simulate this jitter in the timing in order to allow the user to be in a more real environment. The problem of implementing a Closed Loop timing is the tuning of the parameter γ.

Its value is dependent from the SNR: the lower the Signal-to-noise ratio is, the lower should be the step size, since the noise reduces the stability of the loop. However, from real simulation it can be seen that noise can help the loop to get out from some minima (since it is a gradient algorithm). When SNR is high and γ is too small, variations in the timing are not followed by the loop.

The acceptable trade-off can be reached by tuning the parameter starting from a very low value ($\gamma = 10^{-6}$) and increasing it bringing the loop near the instability. A value similar to the one extract from MATLAB simulation is chosen.

The performance of the two kinds of loop, above described, are reported here

Table 3.6 – Clock Recovery Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Open Loop	*135429*	*406.69*	*1*	*0.04*
Closed Loop	140	0.42	1000	0.04

In the table above it is supposed that the Open Loop Timing is done once in a second. In reality, it is performed every time the Symbol Lock is lost. The reason to have a Symbol Lock detector is that, when the clock recovery is not able to find the optimum sampling instant (e.g. SNR is too low, timing jitter is too high, etc.), the receiver should be able to signal the event.

In literature there are different kind of symbol lock detectors that performs quite well if they are implemented in this project. The main problem is to make the detector independent from the SNR and from all the blocks of the receiver (e.g. independent from AGC).

One detector is reported briefly here showing some optimization in the real implementation. The detector described here, requires only two samples per symbol (the same used by the Gardner timing recovery), it has good performance with BPSK and it works well in AWGN [16]. The algorithm is:

- Calculate the error in the optimum sampling instant (the same of Gardner): $u(k) = r(kT + \tau - T/2) \cdot [r(kT + \tau) - r(kT + \tau - T)]$;
- Calculate the error moving by half a symbol (at midway sample): $w(k) = r(kT + \tau) \cdot [r(kT + \tau + T/2) - r(kT + \tau - T/2)]$;
- Take the expectation of both errors and make the difference: $Diff = E\{|w(k)|\} - E\{|u(k)|\}$.

The working principle can be understood better by tracing the S-curves of the two errors. Shifting τ, moving away from the optimum sampling instant, the two expectations are equal with opposite sign

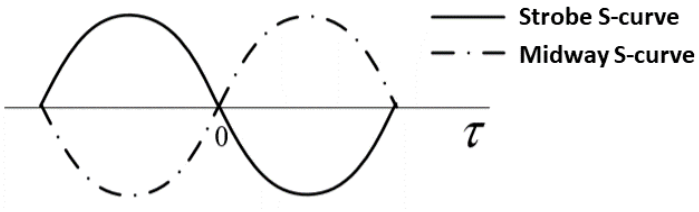

Figure 3.30 – Timing Error Detector S-curves for Strobe and Midway samples

From the plot is easy to see that when the *Diff* goes to zero the symbol lock can be definitely declared. Obviously, this method works well when the Acquisition Sequence is present, but when data are sent the errors increase. Also, if the expectation of the error (over a reasonable interval) is taken, when the data sequence is too long, the error increases too much. So, the practical implementation is not very appeal because, also, it results to be very sensitive to the SNR.

The detector can be slightly modified taking also into account the sign of the sample in the optimum sampling instant [17]. Only the error calculated by the Gardner is needed and it seems to perform well in the condition of the presented project.

In order to declare the symbol lock, a lookup table can be deduced, at different SNR, to set a threshold to be looked by the detector at each optimum sampling instant. The performances are reported in Table 3.7

Table 3.7 – Symbol Lock Detector Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Symbol Lock Detector	*176*	*0.53*	*1000*	*0.05*
Get threshold from LUT	113	0.34	1	0.00

3.6 PHASE RECOVERY

Tx and Rx carrier frequencies are slightly different so, after frequency down-conversion, at the receiving end, the signal will have a residual frequency offset. Distinct estimations between frequency and phase offsets are necessary when the first one is very high. However, since a frequency offset translates into a linear phase variation (in time), only a phase recovery loop can be implemented when the frequency offset is small. In the Subcarrier Synchronizer block a PLL is already implemented, so here a similar one is explained that works directly on the symbol making a decision on it.

3.6.1 Design and Trade-off analysis

The block implemented here is a Costas Loop. It has a Frequency Discriminator Device (FDD) that is a non-linear device returning an error signal proportional to the frequency of the input. The error is calculated by setting to zero the derivative of the log-likelihood of the estimated phase. The result is:

$$e_k = Im\{a_k^* x_k e^{-j\hat{\theta}_k}\}$$

where a_k is the hard decision taken on the entering symbol x_k. At this point it is assumed that all the information about the bit is on the real part of the signal, so the decision is taken only looking at it. Hard decision means that the value of the symbol is chosen comparing it with a threshold, if the symbol is above the threshold it is demodulated as 1, otherwise as 0. The symbols are passed to the layer above (Coding Layer).

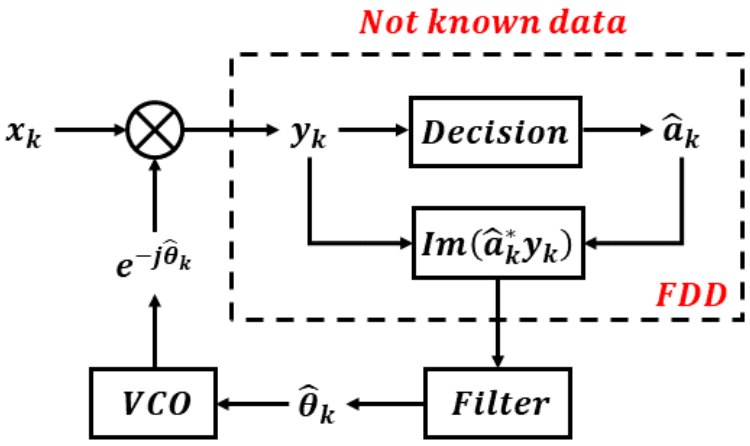

Figure 3.31 – Costas Loop Block Diagram

The updated value of the phase is given by a gradient algorithm and is equal to:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \gamma e_k$$

where γ is the VCO sensitivity.

The Costas Loop is a PLL able to track doppler shift of the carrier. The Doppler effect is a phenomenon that modifies the received signal frequency as a function of the relative motion between transmitter and receiver. If constant Doppler shift is present, in absence of the Costas Loop, the scattering diagram looks as follow

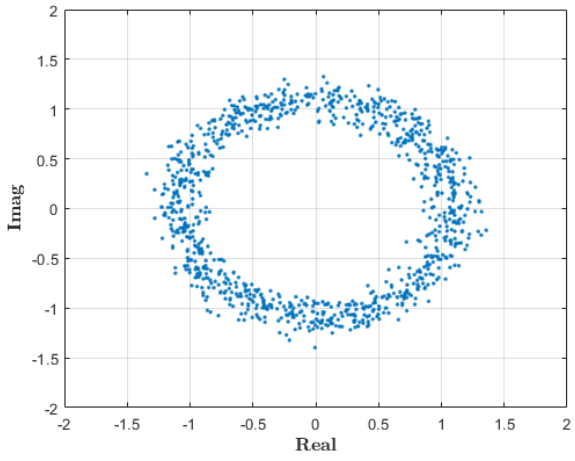

Figure 3.32 – Scattering Diagram in presence of Doppler Shift

Since the Costas Loop is a first order PLL, the study done in the Section 3.3 is valid here as well. Varying the step size γ, the estimated phase at the output of the block is reported here

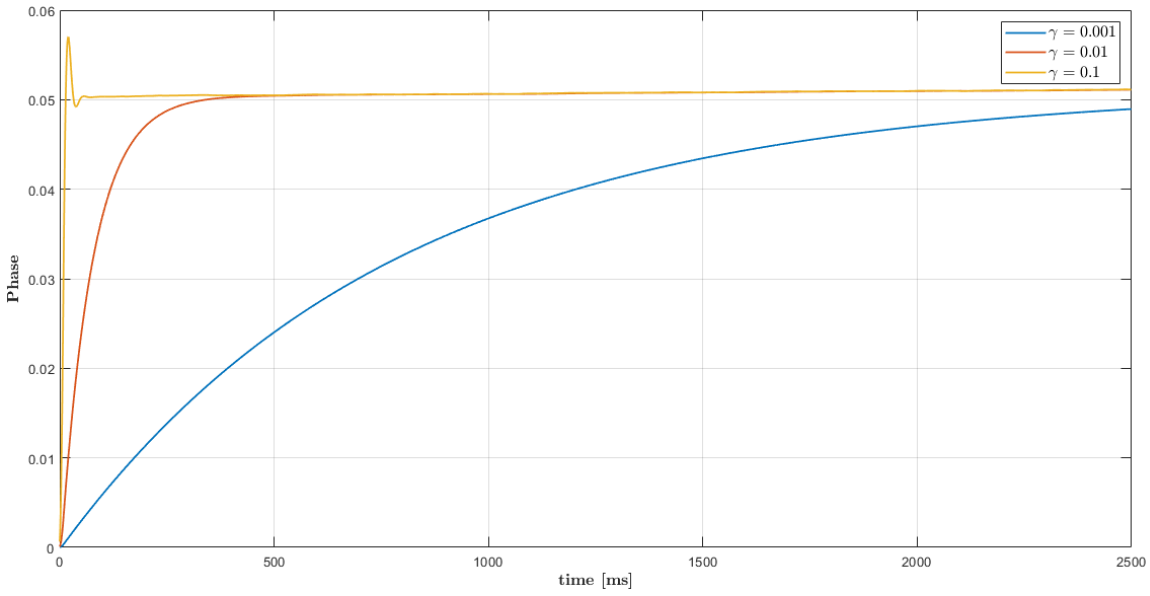

Figure 3.33 – Phase Estimated by Costas Loop with different γ

As expected, increasing the VCO sensitivity, the response of the loop is faster, but it becomes more unstable.

3.6.2 Simulation and Performance results

As anticipated, the Costas Loop works at one sample per symbol, that is the one taken at the optimum sampling instant (given by Clock Recovery block). The performance in terms of clock cycles needed by the loop are reported here

Table 3.8 – Costas Loop Performance

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Costas Loop	*410*	*1.23*	*1000*	*0.12*

The loop processes *1000* symbols in one second and the CPU utilization is 0.12%. Also in this case, in the C implementation a little modification on the γ parameter is needed in order to obtain the same results of the simulation.

CHAPTER 4

4 Coding Layer Design

This Chapter describes the Synchronization and Channel Coding Schemes used with the Telecommand space data link protocol. The data format, the coding scheme and the frame synchronizer block used are explained with particular attention to their performances.

4.1 TC SYNCHRONIZATION AND CHANNEL CODING

The CCSDS specifies a method for synchronizing codewords using a data unit called CLTU (already mentioned in the previous Chapter), which consists in a Start Sequence, a Bose–Chaudhuri–Hocquenghem (BCH) codeword, and a Tail Sequence.

The CLTU is made by two fixed data patterns at the start and at the end. The Start Sequence provides the synchronization pattern and delimits the beginning of the first BCH Codeblock. After *N* BCH Codeblocks, the Tail Sequence is placed and it marks the end of the CLTU

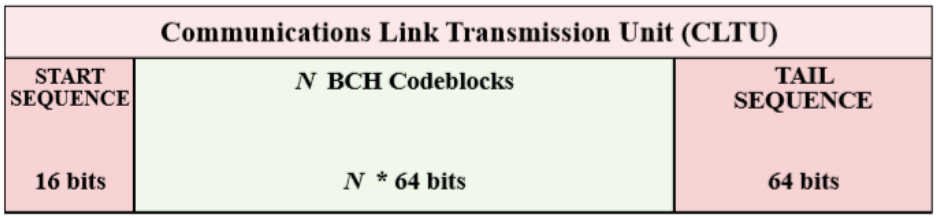

Figure 4.1 – Format of a CLTU [18]

The Reception Logic adopted, at the receiving end, can be represented by the following state diagram

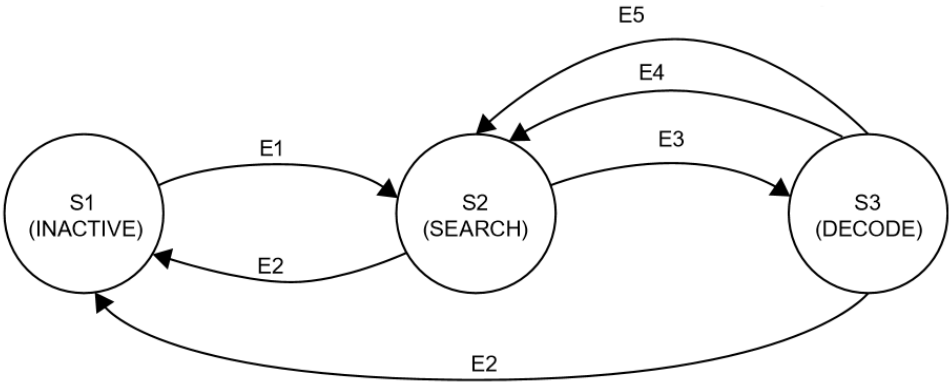

Figure 4.2 – CLTU Reception State Diagram [18]

The Inactive state (S1) is the first state in which the Physical Layer is not sending any bits to the Coding Layer because it does not have bit synchronization of an RF signal. When it achieves the synchronization, after an event signaling, it sends the demodulated bits to the Coding Layer. The Coding Layer switches to the Search state (S2) in which it looks for the Start sequence. When the sequence is found, the Decode state (S3) is enabled and the next received bits are decoded until an End sequence or an uncorrectable word is found. All the events of the state machine are described in the following table

Table 4.1 – Events and States of Coding Layer

	Inactive (S1)	**Search (S2)**	**Decode (S3)**
Channel Activation (E1)	*Go to Search*	/	/
Channel Deactivation (E2)	/	*Go to Inactive*	*Go to Inactive*
Start Sequence Found (E3)	/	*Go to Decode*	/
Codeblock Rejection (E4)	/	/	*Go to Search*

In next Sections, the two main blocks of the Sublayer are described in detail, showing, in particular, the characteristics of the CLTU and of the BCH coding scheme.

4.2 FRAME SYNCHRONIZER

When Coding Layer is in Search state, it is looking for the Start Sequence in order to synchronize the receiver with the CLTU.

To this purpose, the Start sequence is designed in order to provide low autocorrelation side lobes, thereby reducing the probability that the receiving end will mistakenly identify the pattern. In fact, it is clearly distinguishable from the Idle Sequence.

The sequence is composed by 16 bits and the pattern is *EB 90* (in hexadecimal). Its autocorrelation is reported here

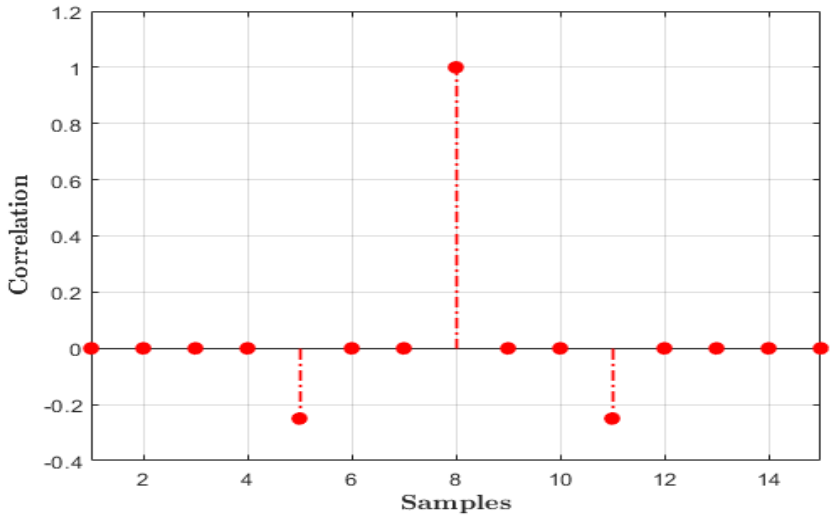

Figure 4.3 – BCH Start Sequence Circular Autocorrelation

The autocorrelation is the correlation of a signal with a delayed copy of itself as a function of delay. In the plot above, the autocorrelation starts with a delay of half of the sequence length (*shift = 8 samples* means that the sequence is perfectly aligned with itself).

4.2.1 Design and Trade-off analysis

Considering the properties of the start sequence, the easiest and most intuitive way to implement the CLTU Synchronizer is to perform the correlation with the incoming symbols bit by bit. The incoming signal is considered in chunk of length equal to the length of the Start sequence and the cross correlation is defined as:

$$(s * x)[n] \triangleq \sum_{m=-\infty}^{m=\infty} s[m]x[m+n]$$

where s is the start sequence, x is the incoming signal and n is the shift in samples.

For example, the cross correlation of the Start sequence with the Idle sequence is shown in the following figure

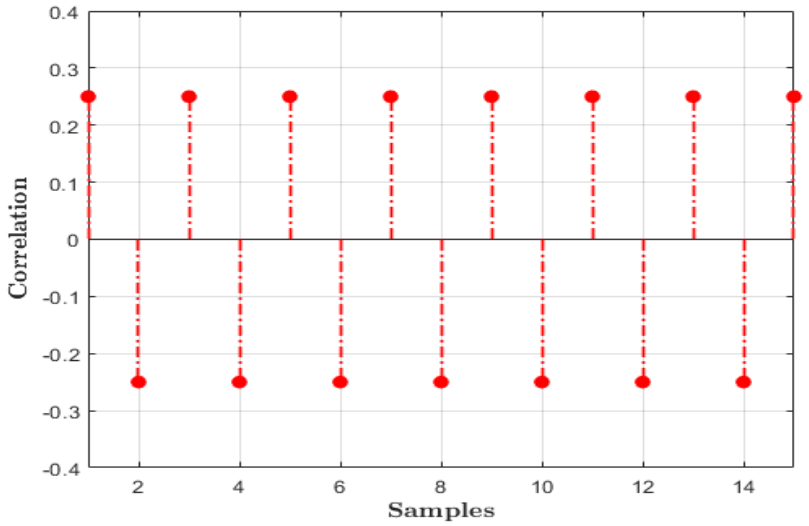

Figure 4.4 – Cross Correlation between BCH Start Sequence and Idle Sequence

For the Frame synchronizer just described, it is supposed that the decision on the symbol at the Physical layer, is a hard decision (Section 3.6.1). However, the decision can be a Soft Decision, meaning that the soft information coming from the channel is exploited. Soft-decision decoders are often used in Viterbi decoders that are used for decoding convolutional codes, so with the BCH coding scheme

this system is not used. This kind of correlation is described in Chapter 5 when a convolutional code is presented.

Since the Physical layer (due to the modulation scheme used) is not able to resolve the data ambiguity, the value of the cross correlation is also used to this purpose. Data ambiguity means that there is a shift in phase of $180°$ of the incoming signal. When the Frame Synchronizer is looking for the Start sequence, it tries to find it or its inverse (the correlation has the opposite value). If the Start of the CLTU is found and it is reversed, all the next bits are inverted until the Coding layer does not go again to the state S2.

Lastly, the Recommended Standard [1] specifies two options for the reception of the CLTU regarding the Start Sequence. The first one is that the CLTU is accepted only if the Start Sequence perfectly match the expected bit pattern (no errors are allowed in the sequence). The second one is that the CLTU is accepted also if the Start Sequence differs of 1 bit from the expected pattern (1 error is allowed). As will be presented in the next subsection, since a decoder in a single error correcting mode is used, the Recommendation suggests to use the second option since it results in a better performance.

4.3 DECODER

The Synchronization and Channel Coding Sublayer encodes the Frames to be transmitted with a BCH block code and generates a set of BCH Codeblocks. The format of a BCH Codeblock is reported in the following Figure

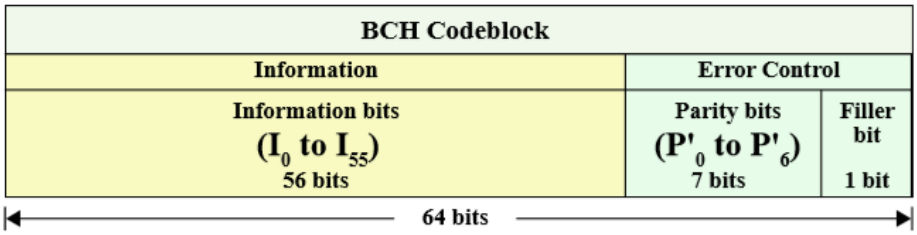

Figure 4.5 – Format of a BCH Codeblock [18]

The Information bits are used as input to the encoding process that generates seven Parity bits. The Filler bit is always set to zero.

This code is called systematic since the information word is unchanged by the encoding process. The error control field is characterized by a generating polynomial and each information word is mapped into it (multiplying them). For systematic codes, all the possible error control fields have an even number of bits equal to one (Parity code). This property is exploited at the receiver side since it is possible to detect any odd number of errors.

The code is an expurgated Hamming code, derived from a basic (63,57) Hamming code. The basic Hamming code can be improved by limiting it to the even-parity codewords. The resulting code has only 2^{56} codewords so it has an information word of 56 bits and 7 check bits. The generator polynomial is:

$$g(x) = x^7 + x^6 + x^2 + 1$$

The modified BCH code has a minimum distance of four meaning that each codeword differs from every other codeword in at least four bits position.

In Figure 4.6, is shown the BCH encoder, and it is briefly explained, since it is implemented for the performance measurements in Chapter 4.4

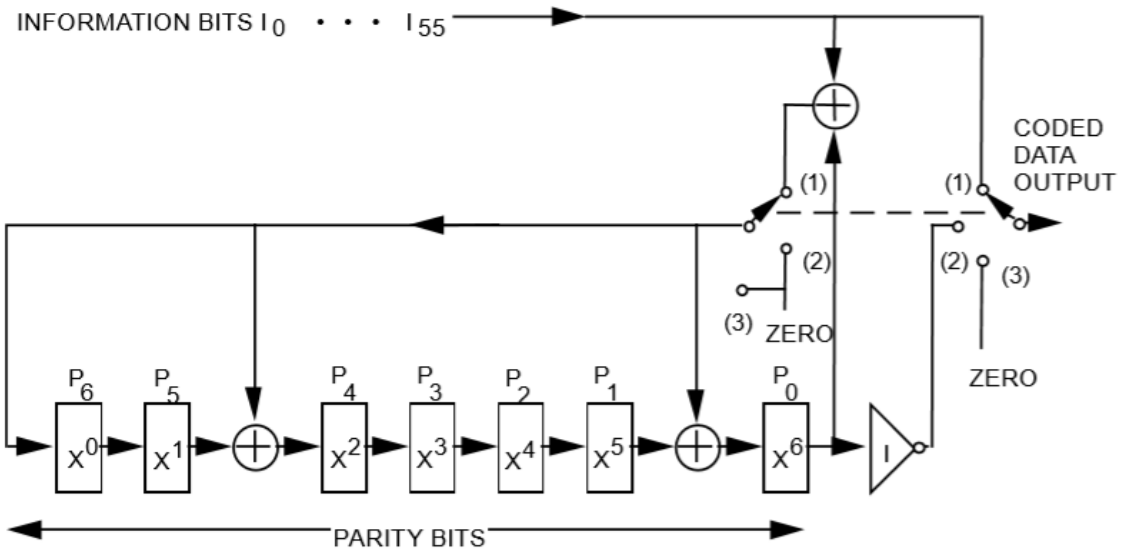

Figure 4.6 – BCH(*63,56*) Encoder [1]

The shift registers are initialized to zero and the switches are in position one while the 56 Information bits enter to the encoder. The Syndrome is updated at each new bit and, after the information bits, the switches move in position two. The seven parity bits are calculated and changed in sign and, for the 64-th bit (filler bit), the switches are in position three.

4.3.1 Design and Trade-off analysis

The decoder can be used either in *error-detecting* or *error-correcting* mode. In the first case, it can detect up to three errors and it can also detect any odd number of errors. In the second case, it can detect up to two errors and correct a single error.

The solution adopted at the Rx, as reported in the Specification, is the Decoder in Single Error Correction (SEC) mode

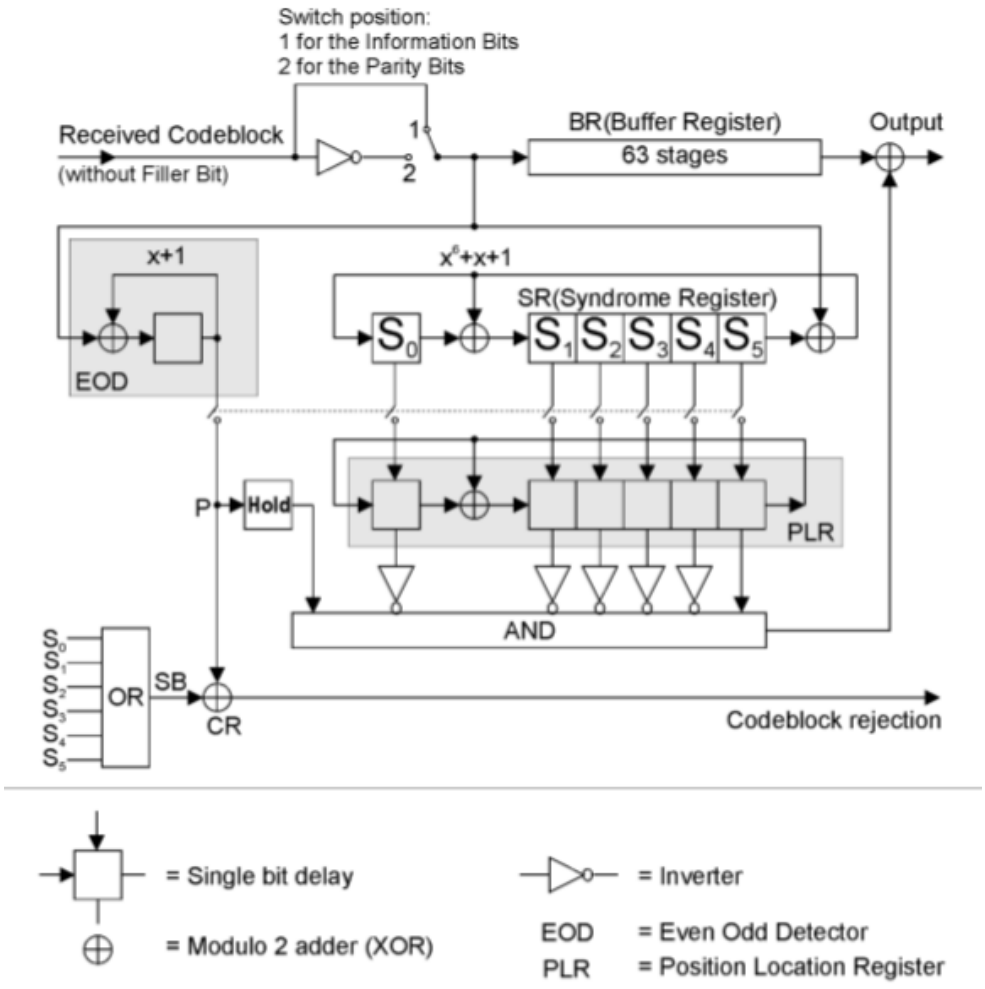

Figure 4.7 – Implementation of Error Correction Mode Decoder [18]

The decoder operates as follows: the Even Odd Detector (EOD) and the Syndrome Register (SR) are all set to zero before receiving a new BCH Codeblock. When the Coding layer switch to the Decoding state, the next 63 bits are input to the Buffer Register (BR), SR and EOD.

After the 63 bits, the BR is full, SR contains the syndrome and EOD is 1 if the parity is odd and 0 if it is even.

The seven switches close momentarily to transfer the contents of the SR to the Position Location Register (PLR) and to transfer the output of the EOD to the XOR gate and to the Hold device. This is done to save the result and to allow the

decoder to calculate the syndrome of another incoming BCH Codeblock while it tries to correct the actual one.

As the next BCH Codeblock is shifted into the buffer BR, the current one is shifted out and, if there is an error, it is corrected at the output XOR gate. This correction occurs when the contents of the PLR register are '*000001*' and the Hold output is '*1*'.

The decisions on whether the Codeword is correct, is rejected or must be corrected are summarized in the following table

Table 4.2 – Codeblock Decision for BCH Decoder (SEC)

DECISION	***EOD***	***SB***
'No Errors'	0	0
'Codeword Rejection'	0	1
'Codeword Rejection'	1	0
'Correction of single error'	1	1

The Decoding process, as already specified in Table 4.2, stops when an uncorrectable Codeword is found. The Tail Sequence in the CLTU has the same length of a BCH Codeword and it has a particular pattern (*C5 C5 C5 C5 C5 C5 C5 79* in hexadecimal) that makes it different from a valid or correctable Codeblock.

The performances of the code are report in the next subsection and another type of Coding Scheme is presented in Chapter 5.

4.4 SIMULATION AND PERFORMANCE RESULTS

In order to calculate the performance of the BCH SEC mode decoder, a MATLAB simulation is executed.

A BCH(*63,56*) encoder is implemented in MATLAB environment in the ways explained previously. In the simulation, a certain number of random bits are encoded, modulated and sent over an AWGN Channel.

At the receiver side, the bits are demodulated and decoded with the BCH SEC Mode Decoder described in the previous Paragraph. Bit Error Rate (BER) and Codeword Error Rate (CER) are calculated as function of $\frac{E_b}{N_0}$.

The BER is calculated by comparing the transmitted sequence of bits to the received bits and counting the number of errors. It is defined as:

$$BER = \frac{N_{Err-bits}}{N_{Tx-bits}}$$

It is the ratio between the number of erroneous received bits and the number of total transmitted bits. In the same way, the CER is the number of erroneous received codewords over the number of total transmitted codewords.

For the simulation, $\frac{E_b}{N_0}$ is incremented by a value of 0.5 dB every time a number of *500* erred codewords are received. Obviously, the more $\frac{E_b}{N_0}$ increases, the more time the simulation requires to obtain a reliable value of BER and CER since is more difficult to reach a so high number of errors and so the simulation is stopped before BER and CER become very small. In the next Figures are illustrated the results comparing the BCH with a system where no coding scheme is used. To obtain the performance of an uncoded system, the same MATLAB simulation, without exploiting the encoder and the decoder, is used

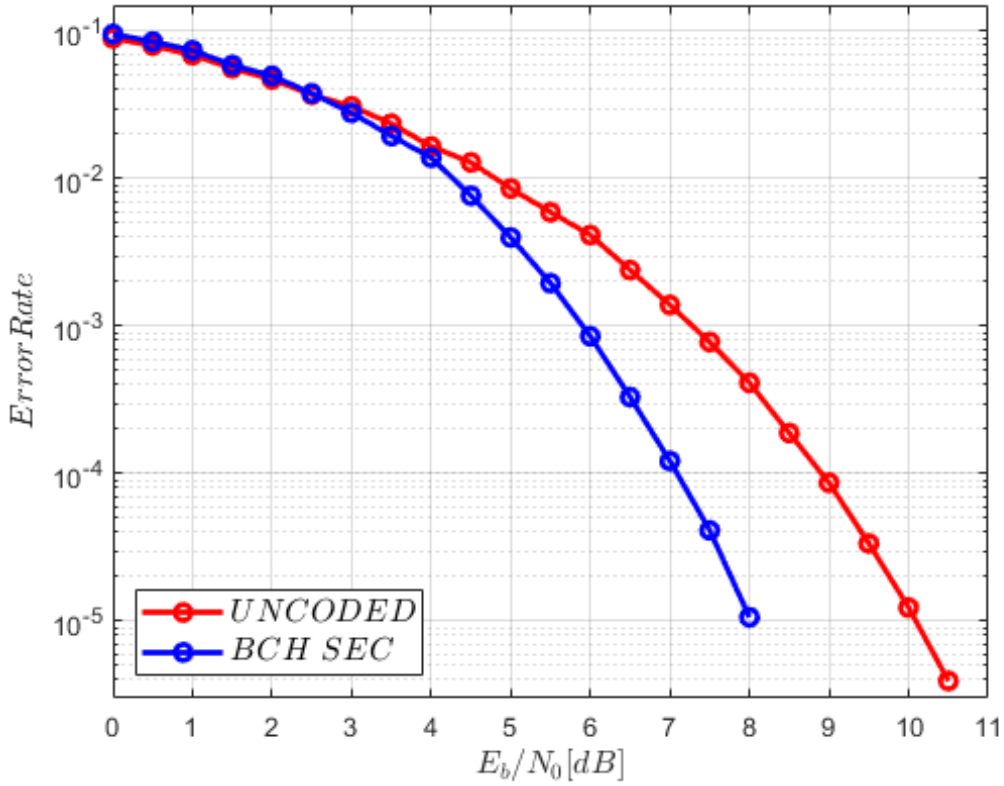

Figure 4.8 – BER versus E_b/N_0 with BCH and without Coding

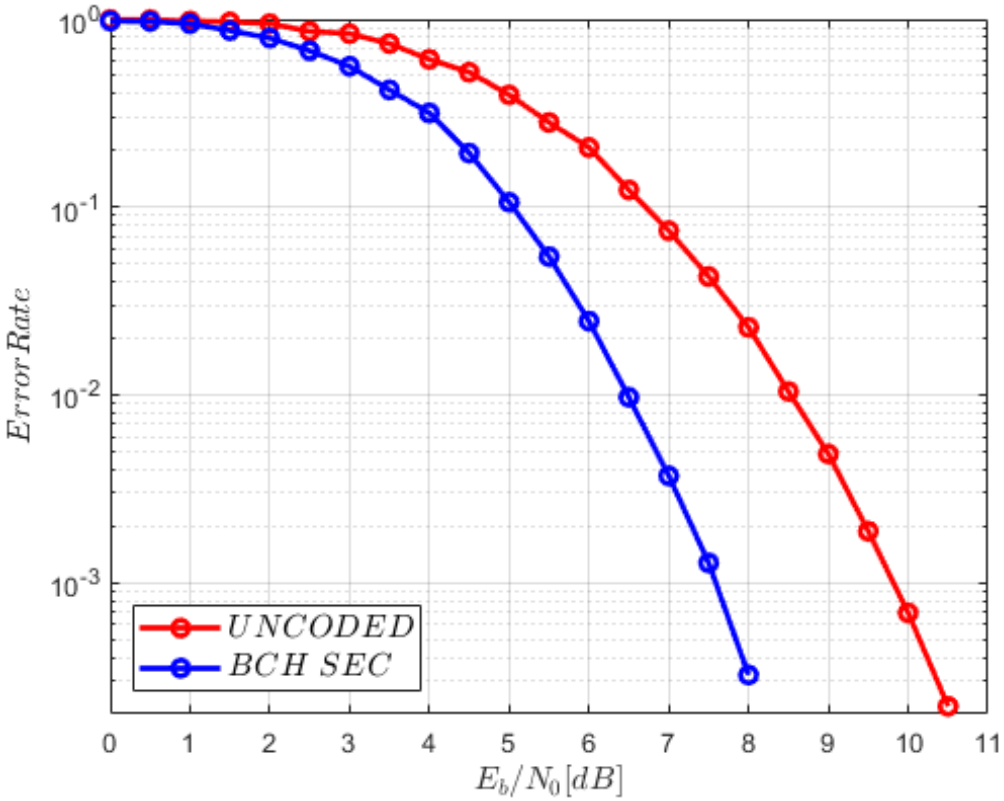

Figure 4.9 – CER versus E_b/N_0 with BCH and without Coding

As it is possible to see from the plots, the BCH decoder, that is able to correct one single error, gains about 2 dB with respect to an uncoded system. For example, to have a Bit Error Rate of 10^{-4} an $\frac{E_b}{N_0}$ of 7 dB is needed. The same $\frac{E_b}{N_0}$ results in a higher value of CER, since the decoder in not able to correct multiple error. The capabilities of detecting odd number of errors is not considered here.

The curves will be compared with the real-time execution of the system in which all the other blocks, described in the previous Section, are implemented in the receiver. To anticipate the results, the simulation is quite similar to the real-time prototype (see Section 6).

CHAPTER 5

5 LDPC Code

In this Chapter a different type of coding scheme is presented. It provides higher coding gain with respect to the BCH code, presented in the previous Chapter. However, its implementation is more expensive so, pros and cons will be analyzed here.

5.1 OVERVIEW

Low-Density Parity-Check (LDPC) Codes are one of the most powerful of channel coding scheme [19]. They are block codes with very sparse parity check matrix (small number of ones), that is a matrix that describes the linear relations that the components of a codeword must satisfy. These codes can be:

- Regular LDPC: have same number of edges leaving each node (same number of 1's in each column and same number of 1's in each row of the parity check matrix). They have good performance but a little worse than turbo codes;
- Irregular LDPC: the degree of each node is allowed to vary and they are competitive with turbo codes.

The parameters of the binary linear block code (C) are:

- k → number of information bits;
- n → number of code bits;
- $R_c = \frac{k}{n}$ → coding rate;
- d_{min} → minimum distance.

The block code can be described by different method, such as, Codebook, Parity check matrix (*H*) or generator matrix (*G*) and by graphical representation through a bipartite graph (Tanner Graph).

5.2 ENCODER

Encoding of LDPC, in general, has a non-negligible complexity. Two LDPC codes are specified by CCSDS Standard with codeword lengths ($n = 128, k = 64$) and ($n = 512, k = 256$).

Different types of encoder are present in literature which can provide high performance simplifying enormously the computation of the codeword. However, this project regards the implementation of the receiver structure and the encoding process is done by a Workstation. For this reason, this work does not investigate the different encoders but only the classic method ($u * G$ encoder) is described since it is implemented to be used by the transmitter.

For any linear block code *C(n, k)*, encoding can be performed by collecting the information bits into a *k*-bit information vector $\boldsymbol{u} = (v_1, \dots, v_k)$, and obtaining the *n*-bit codeword $\boldsymbol{c} = (c_1, \dots, c_n)$ as $\boldsymbol{c} = \boldsymbol{uG}$ where $\boldsymbol{G}$.

The generator matrix $\boldsymbol{G}$ can be obtained by the equation $\boldsymbol{GH^T} = \mathbf{0}_{kxr}$ where $\mathbf{0}_{kxr}$ is the all-zero matrix and $\boldsymbol{H}$ is the parity check matrix that is given by the Standard. For the short block length code, it is equal to

$$H_{64\times128} = \begin{bmatrix} I_M \oplus \Phi^7 & \Phi^2 & \Phi^{14} & \Phi^6 & 0_M & \Phi^0 & \Phi^{13} & I_M \\ \Phi^6 & I_M \oplus \Phi^{15} & \Phi^0 & \Phi^1 & I_M & 0_M & \Phi^0 & \Phi^7 \\ \Phi^4 & \Phi^1 & I_M \oplus \Phi^{15} & \Phi^{14} & \Phi^{11} & I_M & 0_M & \Phi^3 \\ \Phi^0 & \Phi^1 & \Phi^9 & I_M \oplus \Phi^{13} & \Phi^{14} & \Phi^1 & I_M & 0_M \end{bmatrix}$$

Figure 5.1 – Parity Check Matrix of LDPC(128,64)

where I_M and 0_M are the $M \times M$ identity and zero matrices, respectively, and Φ is the first right circular shift of I_M. That is, Φ has a non-zero entry in row i and column j if $j = i + 1 \bmod M$. It should be noted that Φ^2 is the second right circular shift of I_M, etc., and $\Phi^2 = I_M$. The operator $\oplus$ indicates modulo-2 addition [1].

The type of encoding just described is called systematic since the first *k*-bit of the codeword are equal to the information vector.

The short code is the one chosen for the implementation since, LDPC(*512,256*) would add too much complexity to the Rx.

The CLTU, as for BCH coding, has the same structure (Start Sequence, LDPC block, End Sequence) and nothing changes in the reception procedure. The length of the Start and End Sequences changes with respect to the BCH case. For LDPC(*128,64*), the CLTU is illustrated in Figure 5.2

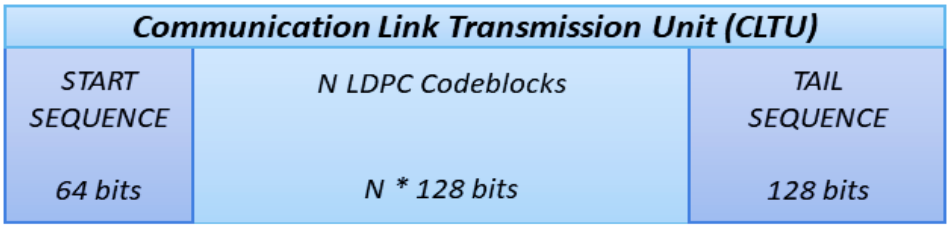

Figure 5.2 – CLTU Structure for LDPC(*128,64*) code

The Start Sequence is composed by *64* bits and has the pattern *0347 76C7 2728 95B0* (in hexadecimal). Its autocorrelation is reported in Figure 5.3

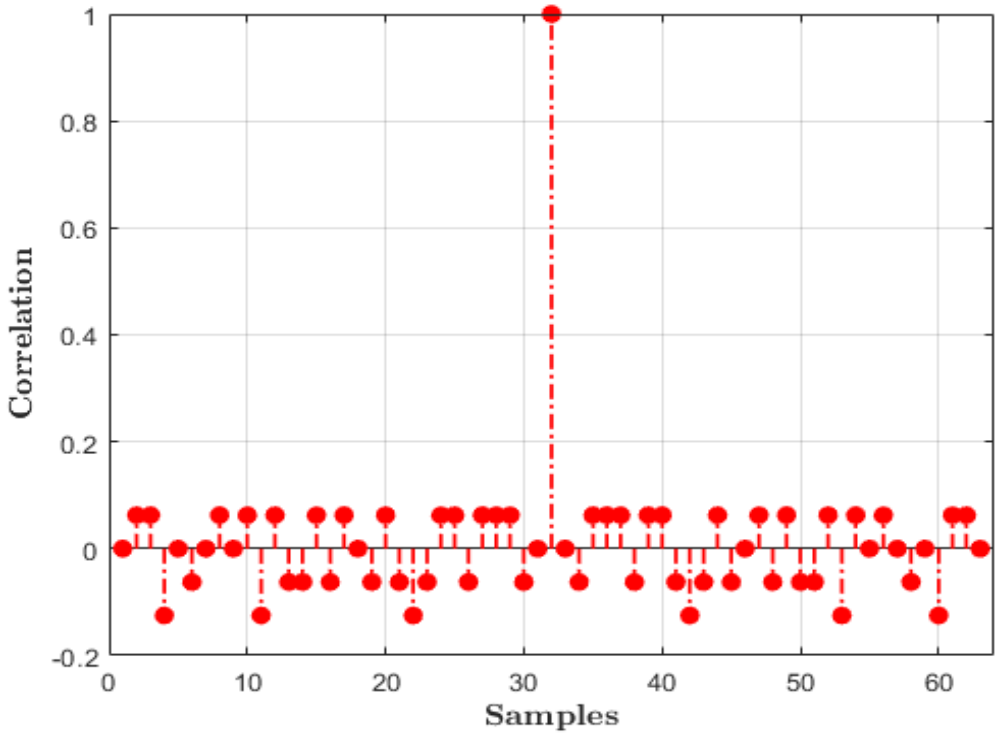

Figure 5.3 – LDPC Start Sequence Circular Autocorrelation

Instead, the cross correlation between Start and Idle Sequences is reported below

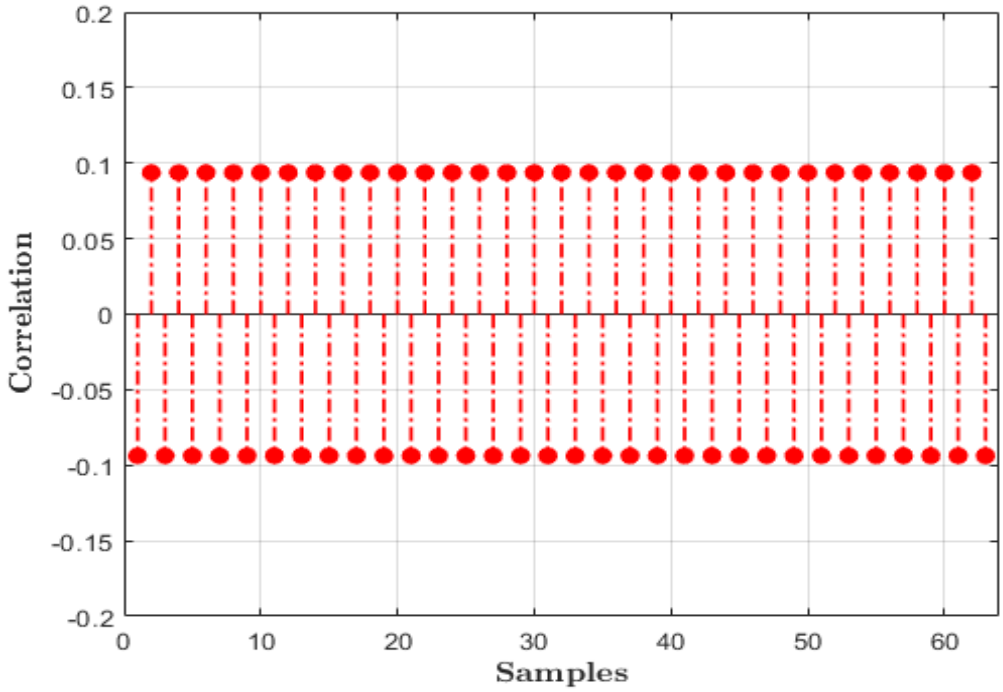

Figure 5.4 – Cross Correlation between LDPC Start Sequence and Idle Sequence

What it is easy to see is that the cross correlation gives lower values with respect to the BCH case where the mean value is *0.25*, while the autocorrelation presents some side lobes (but there are not the two peaks of Figure 4.3). This was expected since the length of the Start Sequence if four times longer.

The properties of the sequence are preserved looking at the cross correlation of it with its noisy version. Simulating an $\frac{E_b}{N_0} = 4$ the results is the following one

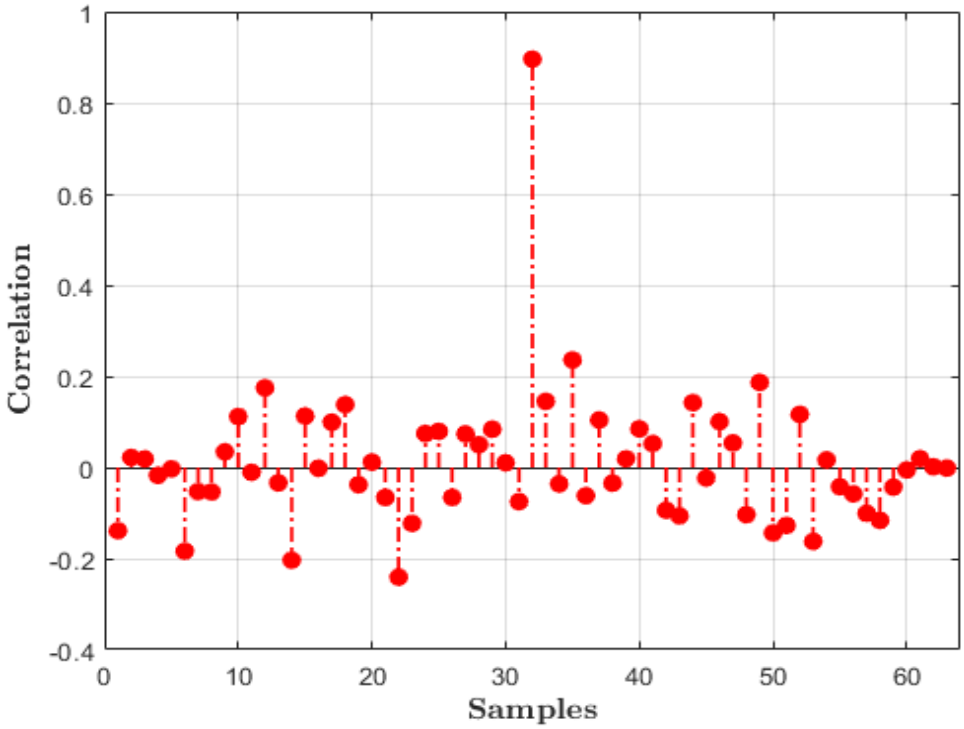

Figure 5.5 – LDPC Start Sequence Autocorrelation $\left(\frac{E_b}{N_0} = 4\right)$

The peak is still highly distinguishable from the side lobes so, in order to test the system implementing an LDPC decoder, the start sequence can be easily recognizable performing, at the Rx side, a cross correlation with the incoming signal and comparing the result with a threshold that can be between *0.4* up to *0.8*.

As for the BCH CLTU, after the start sequence there are *N* LDPC Codewords. Two observation must be done for the encoding process of these blocks. The

first one is that, if the frame that should be transferred does not fit exactly within an integral number of LDPC codewords, then fill bits must be appended to the last LDPC Codeword. The pattern of the fill consists in alternating ones and zeros (starting with a one).

The second observation is that LDPC coding, by itself, cannot guarantee sufficient bit transitions to keep receiver symbol synchronizers in lock. For this reason, a randomizer should be added in the Rx chain (see 5.3).

Lastly, an End Sequence is present after the last Codeword. In reality, this is an optional decision since it can be neglected relying to the ability of the decoder to exit from the decoding state when a non-correctable codeword is found. This reduces the complexity at the receiver stage and increases the throughput since the CLTUs can be transferred one back the other without sending the End Sequence in between. In this project, for completeness, the end sequence is introduced in the CLTU and the decoder looks for a non-correctable codeword until signaling the Coding Layer to switch to the Search state. The sequence has the pattern *C5C5 C5C5 C5C5 C579* (in hexadecimal).

5.3 RANDOMIZER

In order to maintain bit (or symbol) synchronization with the received communications signal, every data capture system at the receiving end requires that the incoming signal must have a minimum bit transition density. Since, as said in the previous Paragraph, LDPC encoding does not guarantee sufficient bit transition, a Randomizer must be used.

The Randomizer is a block used to increase randomness of the Tx binary sequence. A random binary sequence is a sequence of equi-probable $\left(p(0) = p(1) = \frac{1}{2}\right)$ and statistical independent bits.

The properties of a random binary sequence of length *N* are:

- No 0/1 bias → $\frac{N}{2}$ bits are 0 and $\frac{N}{2}$ are 1;
- Transition density → for any bit there is the probability of $\frac{1}{2}$ of having a transition (from 0 to 1 and viceversa);
- Autocorrelation → the sequence is 'orthogonal' to all its shifted versions. This means that when comparing it with cyclically shifted version, $\frac{1}{2}$ bits are equal and $\frac{1}{2}$ bits differ;
- Run distribution → probability of finding long run (sequence of same bit) is very small.

A Randomizer is a (digital) device able to output a pseudo-random binary sequence, i.e., a sequence with nearly optimal randomness properties. How the Randomizer works is illustrated in the following Figure

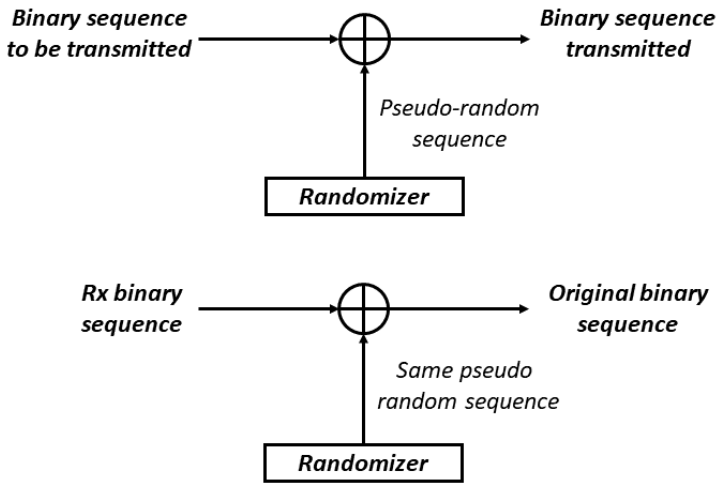

Figure 5.6 – How Randomizer works – Tx side (Up) – Rx side (Down)

The Randomizer is based on a Linear Feedback Shift Register (LFSR), that is a shift register whose input bit is a linear function of its previous state. The LFSR, given a starting seed (initial value of the registers), produces a periodic binary sequence with period $N = 2^M - 1$ where N is the number of shift registers used.

The period corresponds to all possible binary configurations (states), but all zero state must be avoided.

The properties of a random binary sequence are maintained, but the correlation is not exactly *0* as it should be. However, the difference is very small and becomes negligible when period *N* increases. Moreover, the autocorrelation properties are very good only when the entire period is used. When only a segment is used, out of phase autocorrelation may show quite high peaks.

The LFSR proposed by the CCSDS has *8* shift registers, so the max period is $N = 2^8 - 1 = 255$

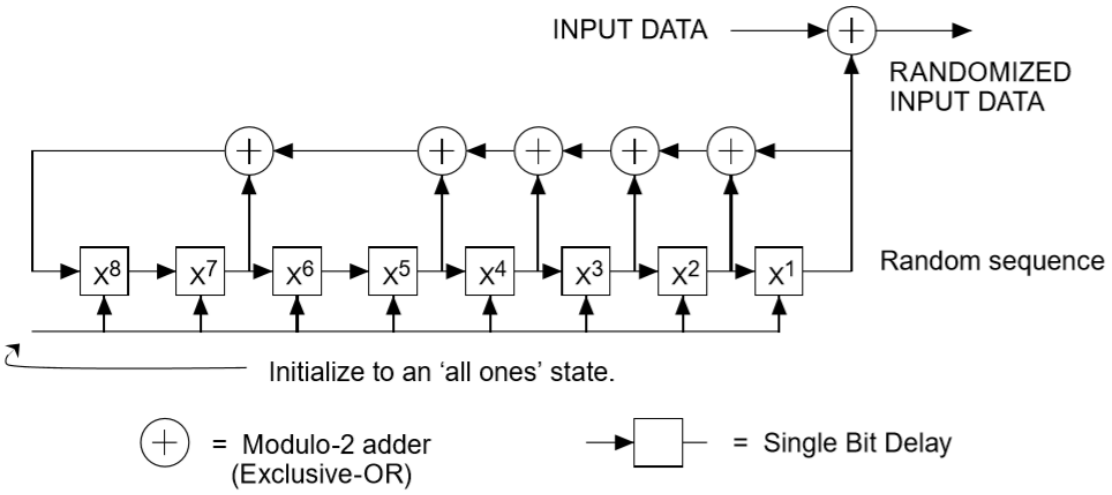

Figure 5.7 – LFSR proposed by CCSDS [1]

The generator polynomial is:

$$h(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$$

The Standard specifies how to generate the random sequence and how it must be applied to the transfer frames. Every time a new CLTU starts, the LFSR is reset. For this reason, in order to do not add complexity to the receiver, the sequence is generated once and put in a lookup table. This simplifies a lot the operation since the Rx does not have to make a lot of shifting and ex-or operations at every received CLTU.

5.4 DECODER

LDPC Decoder exploits 'soft symbols' to decode a Codeword. Soft decision decoding is a class of algorithms that takes a stream of bits or a block of bits and decodes them by considering a range of possible values that they may take. It considers the reliability of each received pulse to form better estimates of input data. This is opposite to the hard decision used by the BCH decoder where the bits are compared with a threshold and they are, definitely, considered as '*1*' or '*0*'.

Summarizing, the Phase Recovery block (Chapter 3.6) does not make a decision on the received bits, but it transfers them to the Coding Layer as they are and it does not trunk the information about the channel carrying by the bits. This, theoretically, provides a performance improvement of about *2-3 dB*.

Nothing is specified in the Standard about the type of decoder to be used. In literature, there are different kind of LDPC Decoders, as specified in [20], and some of them are presented here.

In particular, three decoders are briefly presented here, with particular attention to one of them that is the one that is actually exported in C language and executed on the development board.

The decoders are listed here:

- The iterative Sum-Product Algorithm (SPA);
- The iterative Min-Sum (MS) Algorithm;
- The iterative Normalized Min-Sum (NMS) Algorithm.

The SPA works on the Tanner graph, by implementing an iterative exchange and update of reliability values concerning each received bit.

As already anticipated, the matrix H can be represented by a bipartite graph, where n nodes correspond to codeword bits (message nodes, on the left) and r nodes corresponds to redundancy bits (check nodes, on the right).

Variable nodes and check nodes are connected each other (a *1* in H matrix corresponds to an edge) and they exchange some probabilistic information. $q_{i\to j}(x)$ is the information sent by the i-th variable node to the j-th check node and it is the probability that the bit takes the value x, with $x \in \{0,1\}$. In the same way, $r_{j\to i}(x)$ is the information sent by the j-th check node to the i-th variable node

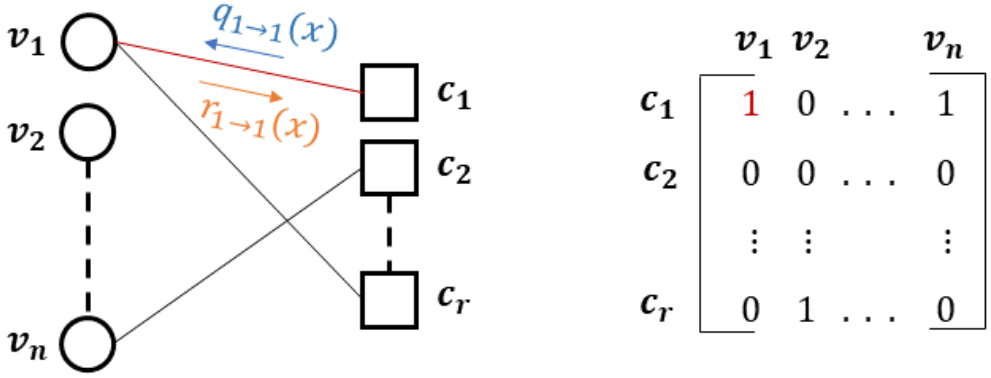

Figure 5.8 – Tanner Graph (left) – *H* matrix (right)

An LDPC code can be decoded by the belief propagation decoding algorithm, that directly matches the code bipartite graph. After variable nodes are initialized with the channel messages, the decoding messages are iteratively computed by all the variable nodes and check nodes and exchanged through the edges between the neighboring nodes.

The implementation of the algorithm is done in the Log-Likelihood Ratio (LLR), that is, for each received bit the log of the ratio between the probability that the bit is *1* and the probability that the bit is *0* is calculated. The LLR is reported on each edge and a Belief Propagation algorithm is applied. The iterative decoding

updates the value of the LLR at each iteration in order to calculate the syndrome. Once the syndrome is zero the decoder stops.

The LLR of a random variable U is defined as:

$$L(U) = ln\left(\frac{P(U=0)}{P(U=1)}\right)$$

where $P(U=x)$ is the probability that U takes the value x. In the LLR-SPA, the messages sent between the nodes are:

$$\Gamma_{i\to j}(\mathrm{x}_i) = ln\left(\frac{\mathrm{q}_{i\to j}(0)}{\mathrm{q}_{i\to j}(1)}\right), \qquad \Lambda_{j\to i}(\mathrm{x}_i) = ln\left(\frac{\mathrm{r}_{j\to i}(0)}{\mathrm{r}_{j\to i}(1)}\right)$$

For the iterative decoding, these values are initialized to $\Gamma_{i\to j}(\mathrm{x}_i) = L(\mathrm{x}_i)$ and $\Lambda_{j\to i}(\mathrm{x}_i) = 0$, where for a BPSK, the value $L(\mathrm{x}_i)$ is equal to:

$$L(\mathrm{x}_i) = ln\left(\frac{P(\mathrm{x}_i = 0 \mid \mathrm{y}_i)}{P(\mathrm{x}_i = 1 \mid \mathrm{y}_i)}\right) = \frac{2\,\mathrm{y}_i}{\sigma^2}$$

where $P(\mathrm{x}_i = x \mid \mathrm{y}_i)$ is the probability that the i-th transmitted bit takes the value x, conditioned on the received symbol y_i and on the variance of the noise σ^2. This is done for each pair of nodes. The value sent from the check nodes to the variable nodes is updated as:

$$\Lambda_{j\to i}(\mathrm{x}_i) = 2 \cdot atanh\left\{\prod tanh\left[\frac{1}{2}\Gamma_{i\to j}(\mathrm{x}_i)\right]\right\}$$

Then, the message sent from the variable nodes to the check nodes is updated as:

$$\Gamma_{i\to j}(\mathrm{x}_i) = L(\mathrm{x}_i) + \sum \Lambda_{j\to i}(\mathrm{x}_i)$$

At the same time is calculated the value of $\Gamma_i(\mathrm{x}_i)$ and an estimation of the transmitted bit is given by:

$$\hat{x}_i = \begin{cases} 0 & if \quad \Gamma_i(\mathrm{x}_i) \geq 0 \\ 1 & if \quad \Gamma_i(\mathrm{x}_i) < 0 \end{cases}$$

If the estimated codeword satisfies all the parity check matrix constraints, the decoding stops and the decoded codeword is given as output. Otherwise, the process listed above is repeated until a maximum number of iterations. If, after the last iteration, the estimated codeword still does not satisfies the constraint, the codeword is considered non-correctable and the coding layer skips to the search state.

The algorithm just explained is quite expensive in terms of performance since it requires some hard operations, such as $atanh(\cdot), exp(\cdot), mod(\cdot)$, updating of a matrix, etc. However, it is suitable for an eventual hardware implementation. Since the load of the Rx described in this work is not so high, the SPA can be implemented in software without particular performance bottleneck.

Other types of decoders, making use of some mathematical approximations, are present. The MS algorithm consists in applying the Max-Log approximation in the computation of the messages between variable and check nodes. The NMS algorithm is based on the same principle, but it introduces a normalizing factor on the LLR values in order to improve the Min-Sum algorithm.

They have similar performance with the one of SPA, but they introduce a small loss in the result.

5.5 Simulation and Performance results

In order to compare the performance of the different decoders a MATLAB simulation is performed.

The same kind of simulation in Chapter 4.4 is executed. Random bits are encoded with the classical method, described in Section 5.2, modulated and sent over an AWGN Channel. At the receiver side, the bits are demodulated and decoded

with the 3 different decoders just presented. BER and CER are calculated as function of $\frac{E_b}{N_0}$.

For the simulation, $\frac{E_b}{N_0}$ is incremented by a value of 0.5 dB every time a number of *20* erred codewords are received. The Randomizer is not implemented in this test and a maximum number of *20* decoding iteration is considered. Obviously, a smaller number of points are extracted since the decoding process requires more time with respect to the BCH one.

In the next Figures, the results are illustrated

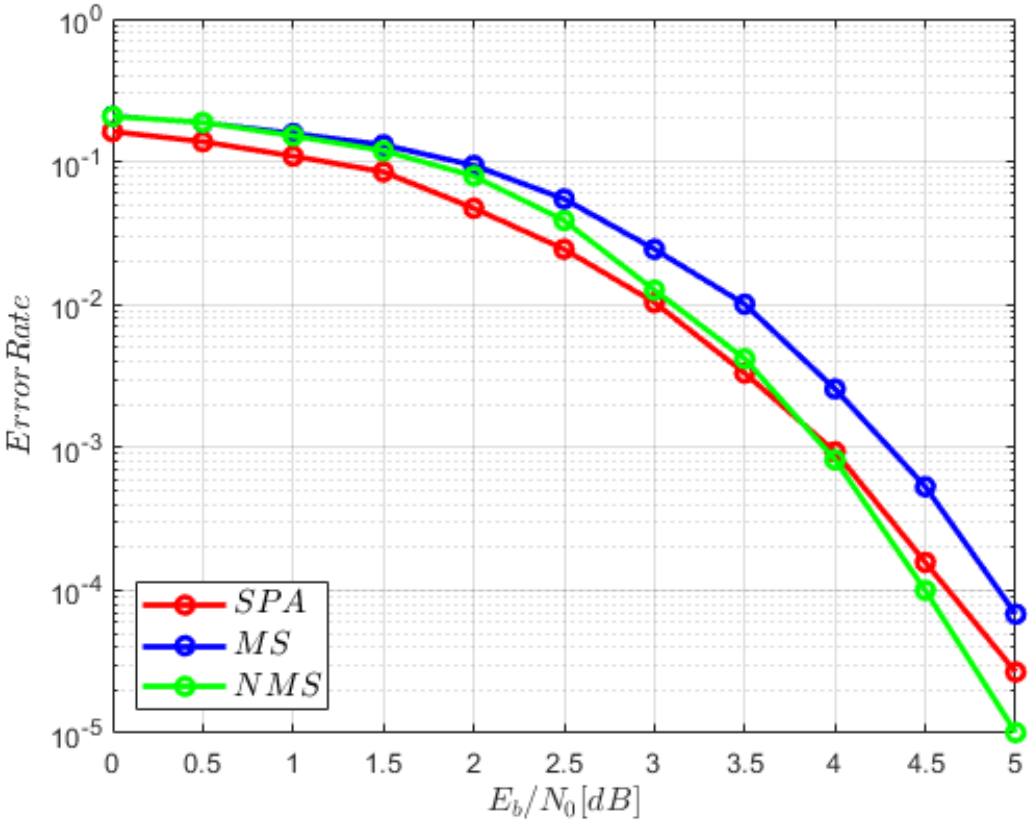

Figure 5.9 – BER versus E_b/N_0 with different LDPC decoder

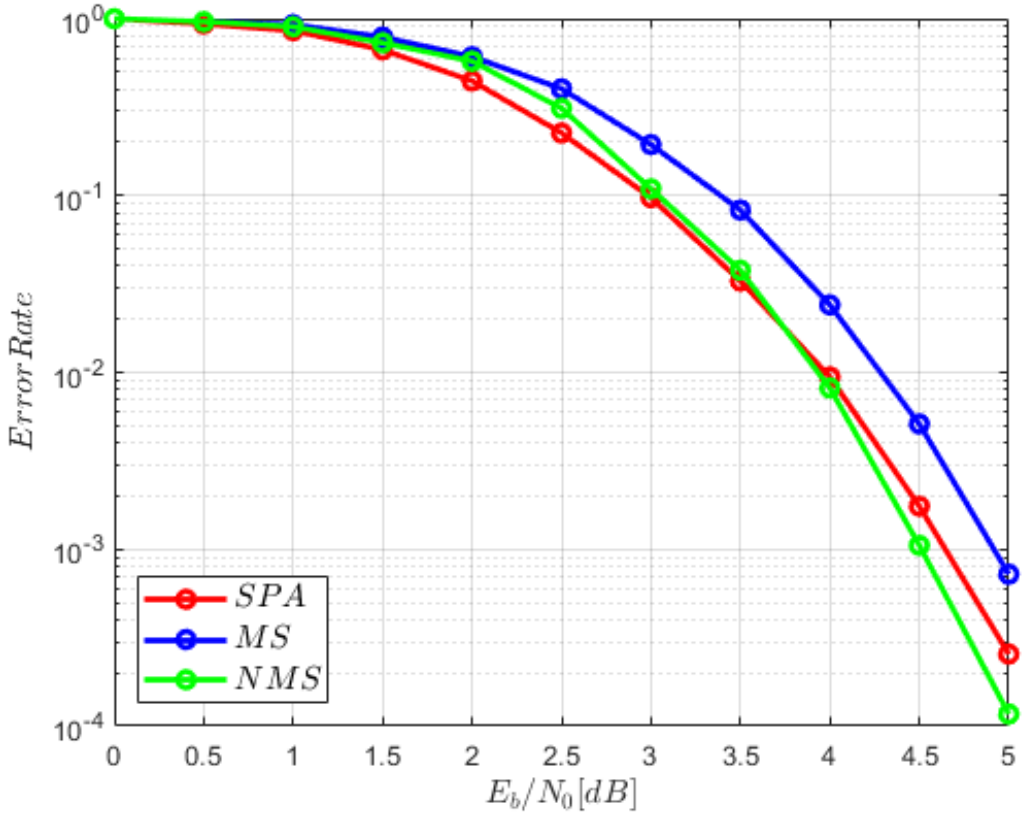

Figure 5.10 – CER versus E_b/N_0 with different LDPC decoder

The SPA decoder has the best performance in terms of BER and CER at very low value of SNR. From $\frac{E_b}{N_0} = 4\ dB$, NMS algorithm gives better results. This will be analyzed in detail as future work, when all the decoders will be implemented on the Zedboard. The results can be justified from the fact that the simulation is stopped before obtaining a confident value of error rate.

NMS algorithm (a normalization factor of *0.8* is used) seems to perform slightly better than MS, but Sum-Product Algorithm is the best one. However, MS and NMS are a good trade-off between complexity of the implementation and performance of the result.

Now, the performances of the SPA decoder (considered since it is the one implemented in practice), are compared with the performances of the BCH decoder, already presented. The same graphs of BER and CER are extrapolated, increasing the number of points for the SPA decoder

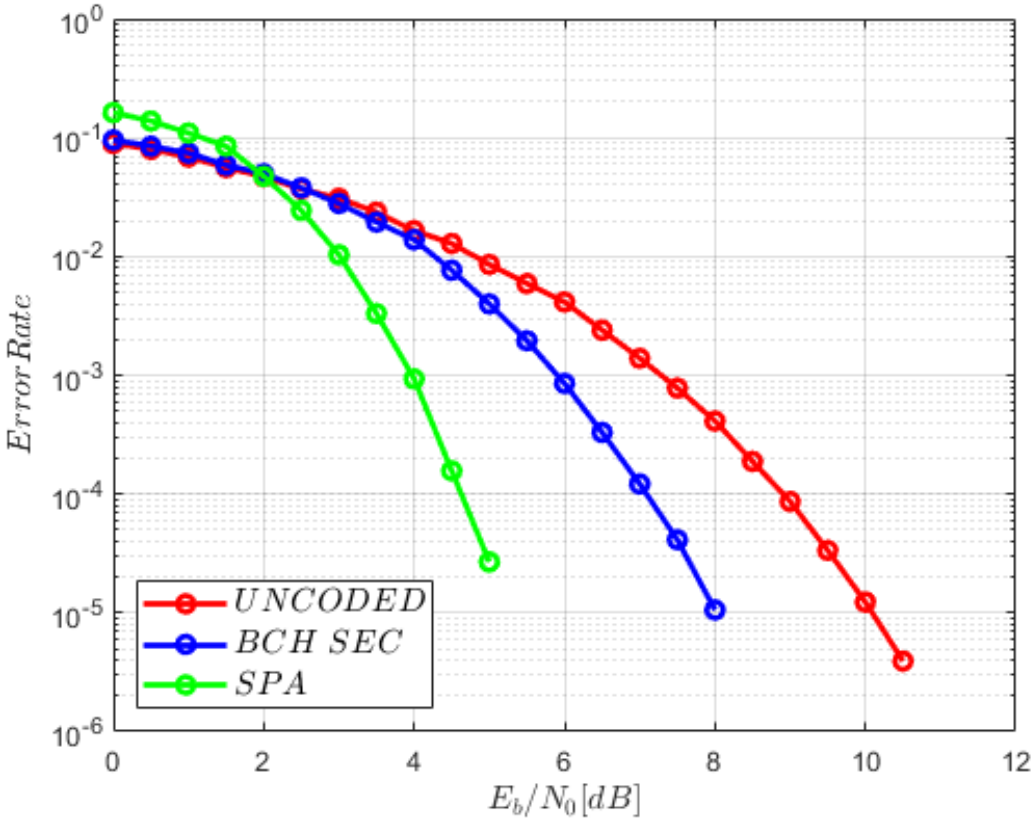

Figure 5.11 – BER versus E_b/N_0 – Comparison between Coding Schemes

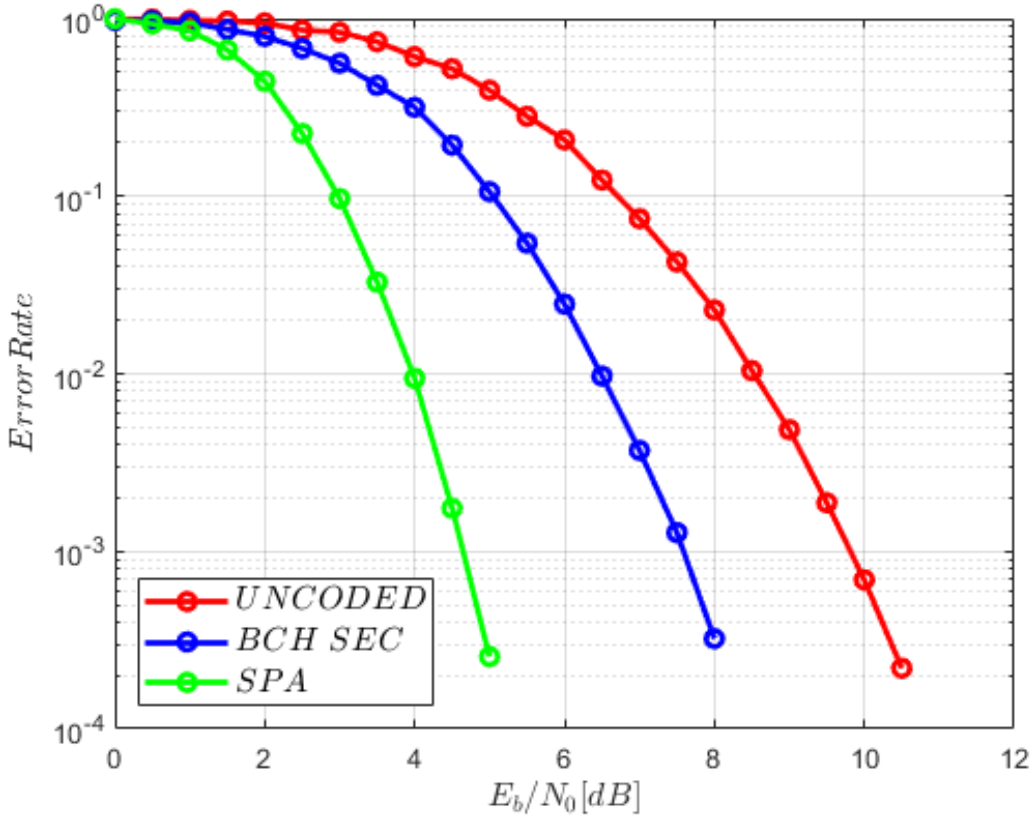

Figure 5.12 – CER versus E_b/N_0 – Comparison between Coding Schemes

With these plots, it is possible to see better the difference between LDPC and BCH coding schemes. It was already anticipated that, with LDPC code is possible to achieve a gain of about *2-3 dB* with respect to the BCH code. This is visible in the graph where LDPC has the best performance.

For example, considering an acceptable value of BER (10^{-4}), the result is that LDPC is able to achieve it at around *4.5 dB* of $\frac{E_b}{N_0}$, instead, BCH needed a value of *7 dB*. These results will be compared with the practical system implementation.

Using a complex coding scheme, such as the one described in this Chapter, with respect to use the BCH code, can make an important difference when working with very low values of SNR. Other types of simpler decoders will be investigated as future task, in order to make the implementation of the Rx simpler.

CHAPTER 6

6 Prototype Testing

This Section describes the set-up used to test the receiver architecture designed and simulated in the previous Chapters. All the receiver blocks are assembled together in order to write a software able to process a real-time signal. Improvement to the system and performance analysis are made to obtain a prototype that can be upgraded with future work.

6.1 TESTING OVERVIEW

The goal of this Chapter is to implement the functional baseband receiver designed in the previous Chapters, able to receive a signal and processing it to extract the information that it carries (Telecommand data). The test set-up used was already described in Section 2.6 and shown in Figure 2.8.
Recalling the Scheme, the Ethernet link is exploited to send the signal generated by the Workstation and it also used by the Zedboard to send back some useful variables that can be observed through graphical representation in the GNU Radio environment.

An example of the interface is reported in Figure 6.1. It can be observed that some parts of the screen are dedicated to the visualization of the transmitted signal and an Options tab is present to control its generation. The third part is dedicated to shown the signal processed by the receiver chain

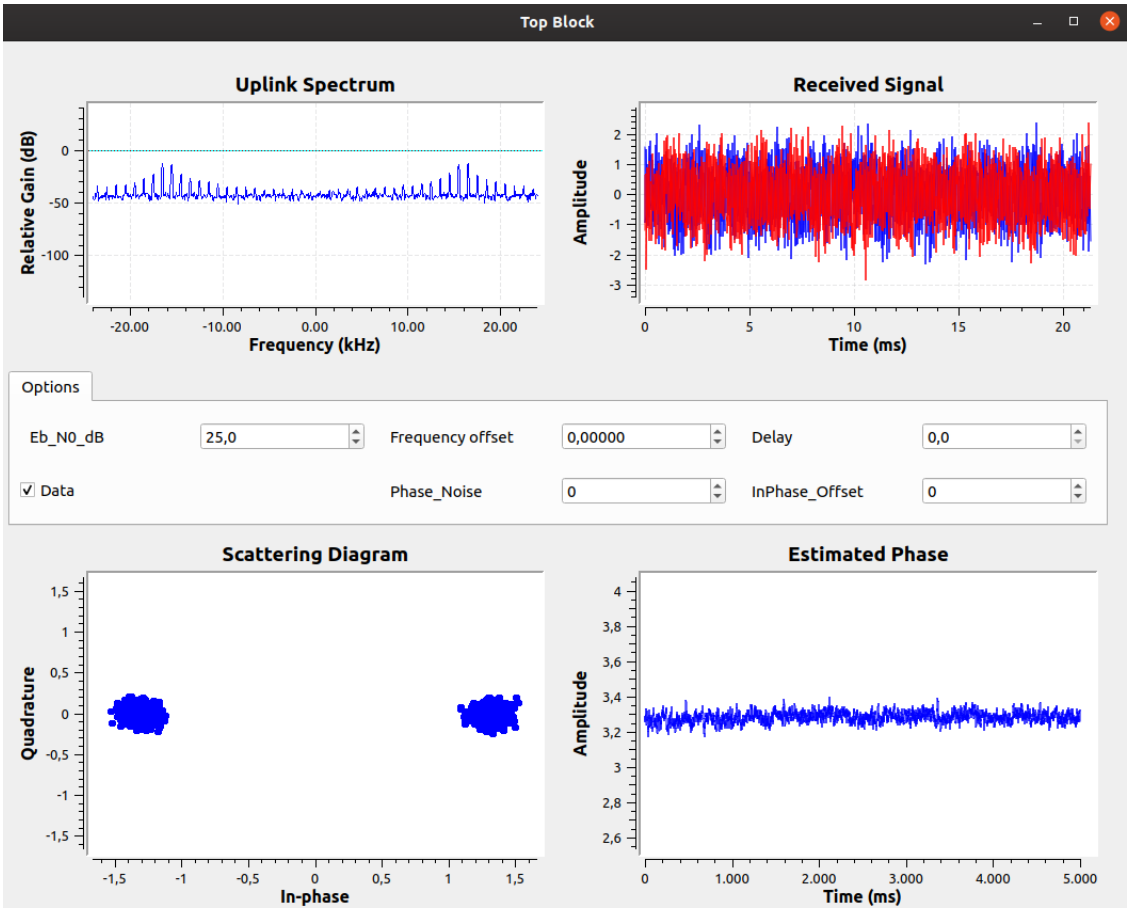

Figure 6.1 – GNU Radio Interface

More in detail, once the program is running is possible to see: the spectrum of the Tx signal, the shape of the signal in time and a 'Option' panel that can be used to modify the parameters. For example, is possible to add a delay in time to the signal, to add a frequency offset, to change the value of $\frac{E_b}{N_0}$ (modeled by the channel model block) and to add phase noise.

Other plots are used to shows the processing on the signal by the receiver that sends back some data. For example, the filtered signal and also the scattering diagram after the PLL is shown in Figure 6.2.

For what concern the receiver part, the Xilinx chip is programmed with the C code written exploiting all the information, about the blocks, derived by the MATLAB simulations. Using the UART interfaces is possible to print, also here,

some control variables on the terminal to check the correct execution of the software.

With the set-up just described, each block is tested and the obtained results are compared with the ones extracted in the MATLAB simulation.

6.2 CPU UTILIZATION

The results in terms of clock cycles needed to each block of the Physical Layer to perform its operation, is already given in Chapter 3. The Physical Layer and the Coding Layer are implemented in the software as two separated Finite State Machine (FSM).

The CPU utilization of the Coding Layer depends on the coding scheme used in the receiver. Considering the BCH Decoder, the following table summarizing the performance

Table 6.1 – Performance of Coding Layer FSM (with BCH Decoder)

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Frame Sync.	*532*	*1.6*	*1000*	*0.16*
Decoder	817	2.45	1000	0.25

Here, it is supposed that the Frame Synchronizer performs the correlation every time a new symbol enters the block (*1000* bits at second). Also for the BCH Decoder is supposed that it must decode *1000* bits in one second and it must decode and correct one bit on each received frame. These are usual operations that the decoder does every time a new start sequence is detected. So, the two blocks never work simultaneously and the CPU utilization must be summed to the one of the Physical Layer. As it can be notice, the Rx is not very complex at this stage, so the utilization of a more complex decoding scheme is reasonable.

To this purpose, the performance of the Coding Layer FSM, considering the LDCP encoder, is reported below

Table 6.2 – Performance of the Coding Layer FSM (with LDPC Decoder)

Type	Clock Cycles	Elapsed Time [μs]	Repetitions	CPU [%]
Frame Sync.	*1300*	*3.9*	*1000*	*0.39*
Decoder	194412	583.82	8	0.47

The same approach as for the BCH Coding Layer is followed. Obviously, the Frame Synchronizer requires more resources since the correlation is performed on *128* bits instead of *16*. For the Decoder are considered the clock cycles needed to decode one frame (*128* bits) with only one iteration. In one second, the decoder decodes around *8* frames. The performances are slightly worse than the BCH decoder, but not so high to exclude the utilization of the LDPC coding scheme on the Receiver. However, the difference can be seen at lower SNR where the Decoder must perform more than one iteration to decode the frames. Considering a maximum number of iterations of *20* and supposing that the decoder makes them for all receiver frames, the CPU utilization increases to 9.34 %.

Summarizing, the CPU utilization, distinguishing the case in which BCH and LDPC decoders are used, is shown in the following pie charts (the worst case for LDPC decoder is taken into account, i.e. it must perform the highest number of iterations)

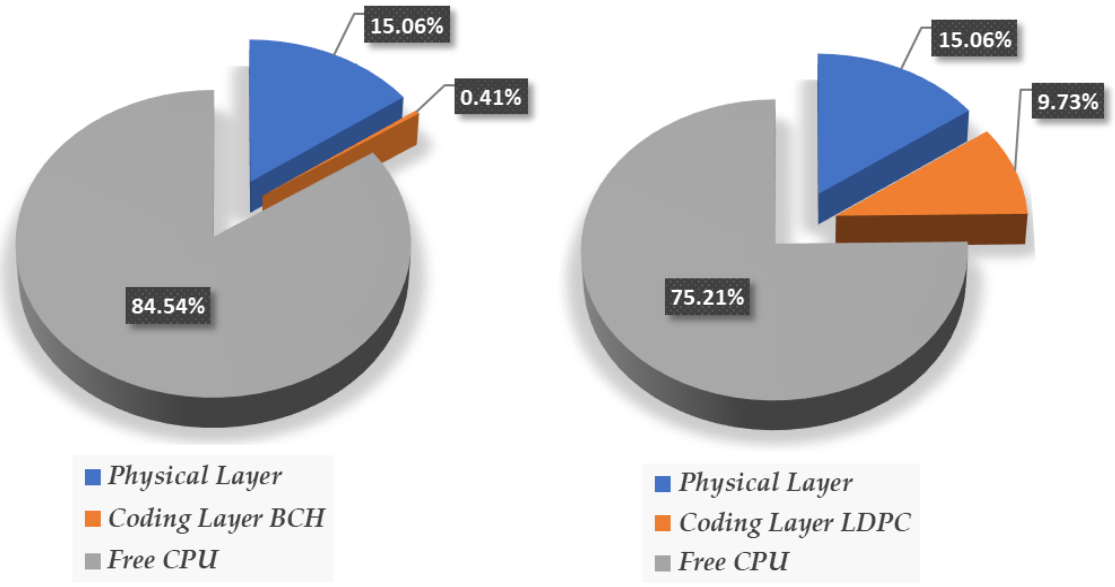

Figure 6.2 – CPU utilization by the two FSM

The utilization is calculated considering the number of time that each operation is performed in one second of execution. The CPU is always used at its maximum frequency (667 MHz), the L1 and L2 Caches are enabled by default and the software is running on a single core.

For what concern the Physical Layer, the percentage is the sum of all the contributions described in Chapter 3 and it is equal to 15.05 %.

More in detail, the chart in Figure 6.3 represents how the utilization is subdivided between each block

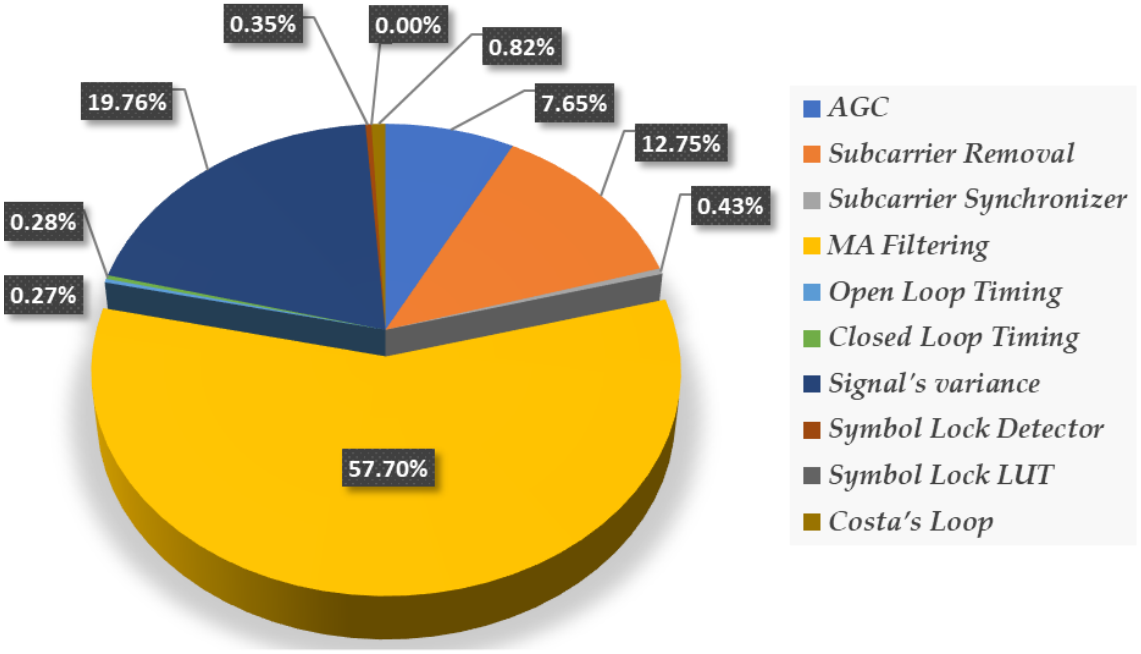

Figure 6.3 – CPU utilization by Physical Layer's blocks

Despite MA filtering and signal's variance are moved in hardware, they are the main contribution in the processing chain.

Instead, the two Coding Layers exploit the CPU as shown in Figure 6.4

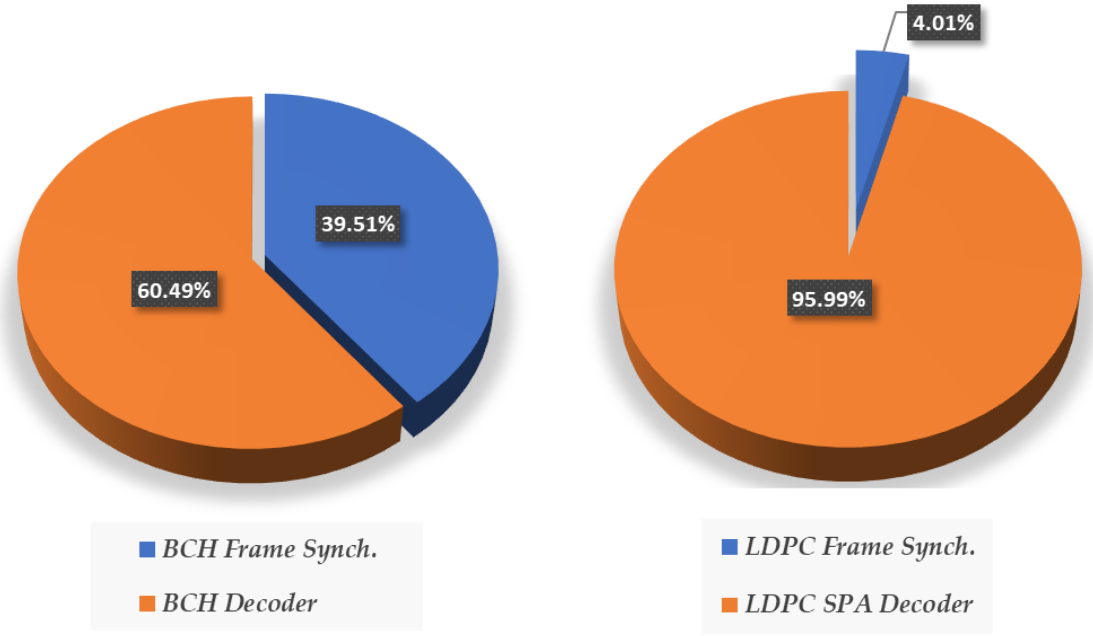

Figure 6.4 – CPU utilization by Coding Layer's blocks

6.3 BCH SEC DECODER

This Paragraph verifies the performance, in terms of BER and CER, of the BCH Decoder. First of all, the system is tested making a CLTU without coding the information bits. The comparison between the codeword error rate simulated in MATLAB and the one obtained the implemented software is shown below

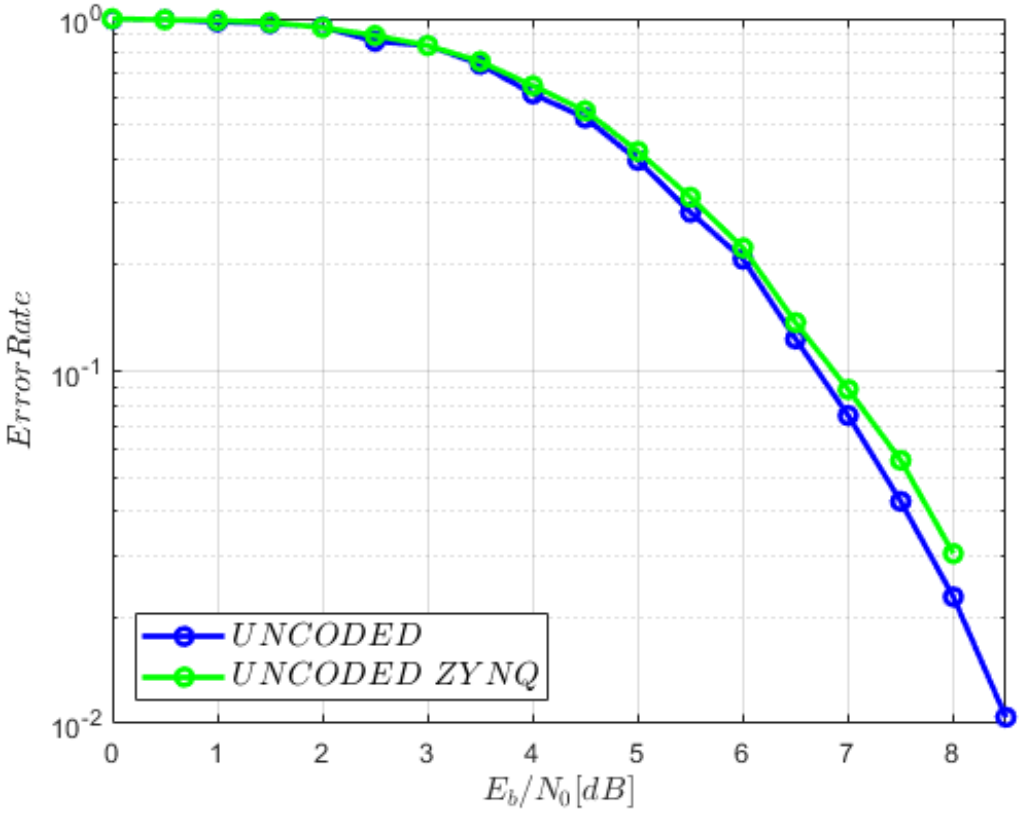

Figure 6.5 – CER of uncoded system

It can be seen how the two curves are, as expected, almost similar. So, the physical layer of the system is working properly, providing a hard decision of the processed symbols.

To test the BCH Decoder, a series of CLTUs, containing real NOOP Commands (the spacecraft does not use this TC, they are only used for test), are sent to the Receiver. As already seen in simulation, the Decoder works with very low BER at $\frac{E_b}{N_0}$ above $8\ dB$. Supposing this limit value, where in simulation the BER was around 10^{-5}, at least one error per Codeword is observed and correct as is

possible to see in the Xilinx Vitis Terminal (Figure 6.6). In the left part of the Image, the scattering diagram at the output of the PLL is shown

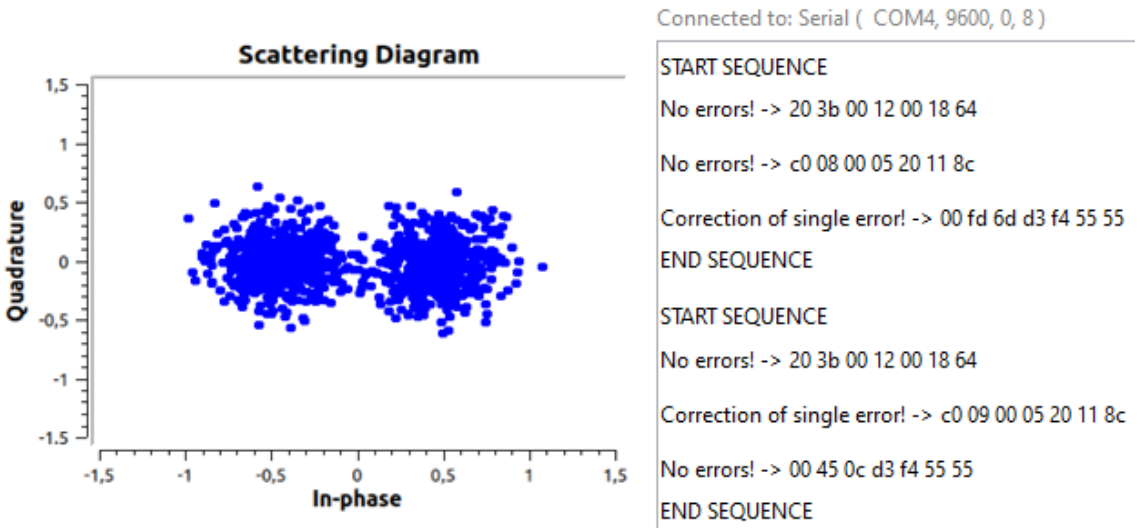

Figure 6.6 – BCH Decoder $\left(\frac{E_b}{N_0} = 8\ dB\right)$

Reducing $\frac{E_b}{N_0}$ to $7\ dB$, some Codeword present an even number of errors (they are rejected) or they present odd numbers of errors, but only one is corrected, so the Codeword is erred.

6.4 LDPC SPA Decoder

To test the LDPC Decoder (SPA), a C program is written that is able to create a CLTU by:

- Placing the Start Sequence;
- Encoding frames by means codifying characters, passed as input parameters to the code, in ASCII;
- Closing the last frame adding fill bits as a sequence of ones and zeros;
- Placing the Start Sequence.

The generated sequence is sent through the Ethernet interface exploiting the software GNU Radio that translates the bits into the signal that will be process by the Receiver.

As expected, the performance of LDPC Coding are better that the BCH one. At the same values of $\frac{E_b}{N_0}$ the LDPC decodes the signal with only one iteration and with no errors.

At lower SNR, the LDPC start to increase the number of iterations and at $\frac{E_b}{N_0} = 5\ dB$, its mean is around 3. From simulation results that this value gives a BER quite similar to the one obtained from BCH at $\frac{E_b}{N_0} = 8\ dB$. So, the gain of *2-3 dB* is also confirmed here. In the test, the alphabet is sent more times in separated CLTUs. After the character *'Z'* a sequence of *'U'* is present (in ASCII, it corresponds to the fill pattern *'01010101'*). The number next to each frame represents the number of iterations performed by the SPA to decode it

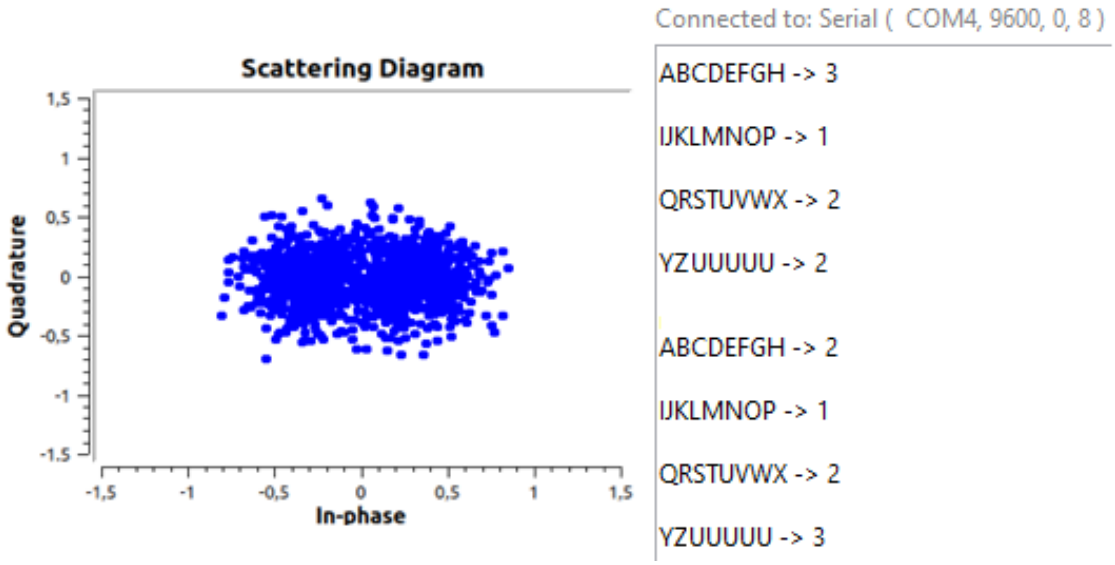

Figure 6.7 – LDPC Decoder $\left(\frac{E_b}{N_0} = 5\ dB\right)$

Reducing $\frac{E_b}{N_0}$ to *4 dB*, the constellation points are almost a unique one. The mean number of iterations increases and some uncorrected Codeword appears. In

Figure 6.8, one frame in the first CLTU is not corrected by the Decoder. This means that the decoder performed the maximum number of iterations, but the parity check conditions are not verified so, the decoder rejects the Codeword and the Coding Layer goes in Search state

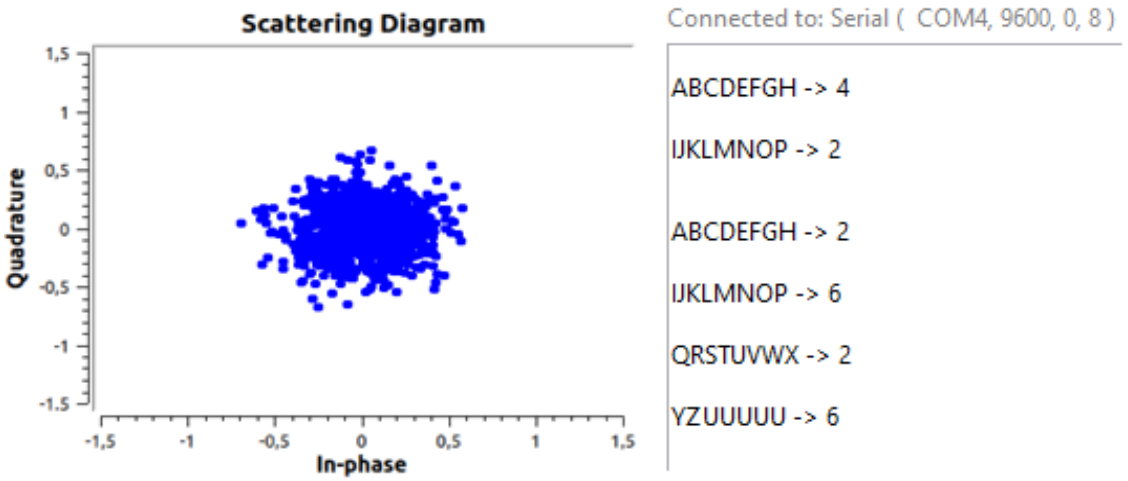

Figure 6.8 – LDPC Decoder $\left(\frac{E_b}{N_0} = 4\ dB\right)$

To better see the performance of the SPA decoder implemented on the Zedboard, a CER curve is extracted and compared with the one derived from the MATLAB simulation. In the figure is also reported the CER of the uncoded system is highlighted

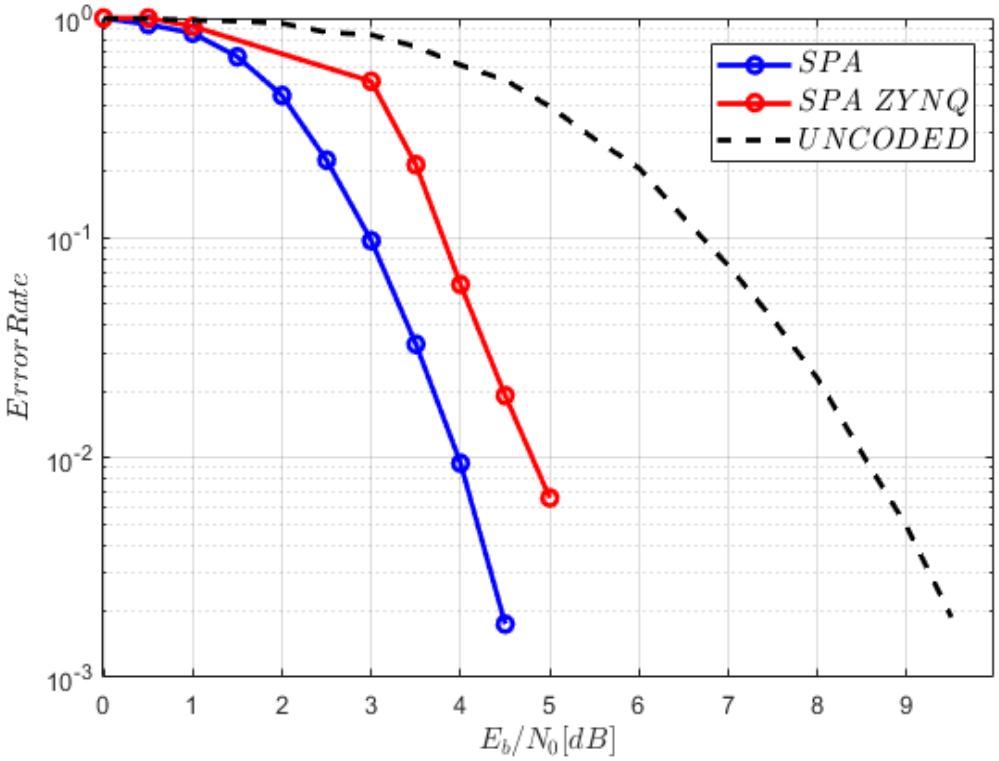

Figure 6.9 – SPA decoder CER performance on Zynq-7000

As it can be seen, the decoder's performance is quite similar to the theoretical one. Considering the same value of $\frac{E_b}{N_0}$ a penalty smaller than 1 dB is present, that is still a good reason to use LDPC code with respect to the BCH one. The calculation of the CER becomes very difficult for a value of $\frac{E_b}{N_0}$ greater than $5\ dB$, since it will be really small and, probably, closer to the theoretical behavior of the decoder at high SNR.

A last interesting observation is done on the maximum possible data rate that the system can achieve with LDPC code.

Starting from a data rate of 1 $kbit/s$ up to 4 $kbit/s$ the codeword error rate curves are reported in Figure 6.10

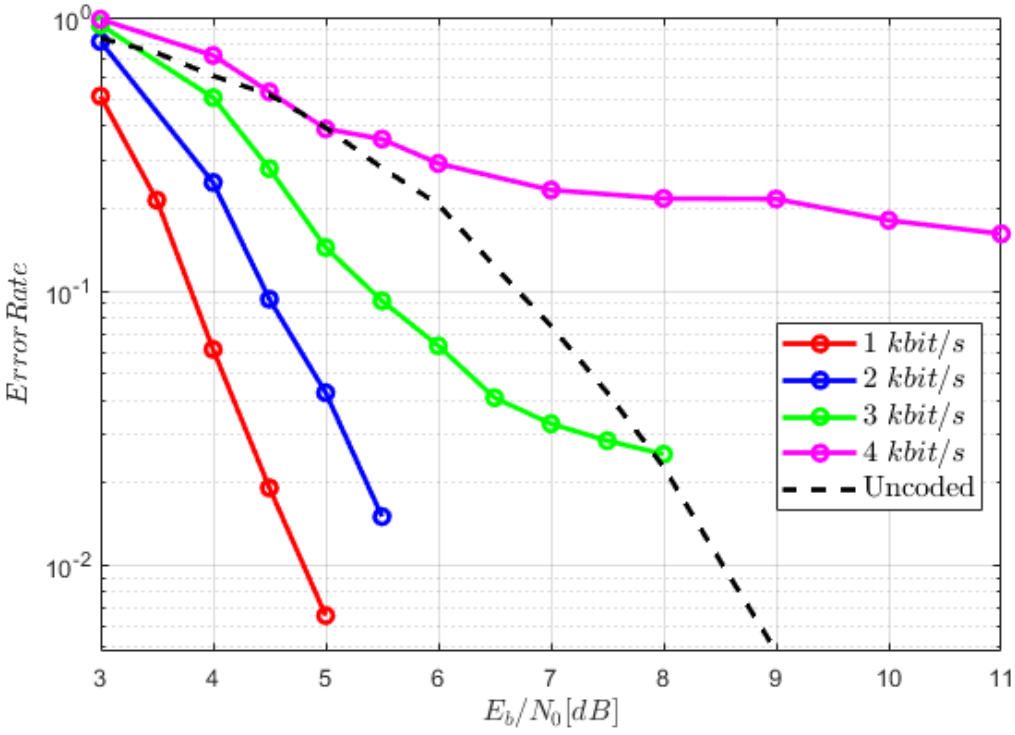

Figure 6.10 – SPA decoder CER performance at different data rates

It is evident how for a data rate greater or equal to 3 $kbit/s$ the system is unusable since the CER goes near (but also exceeds) the performance of an uncoded system. The reasons why this happens are different and most of them must be attributed to the physical layer. As seen in the design, all the blocks are be tuned to work at 1 $kbit/s$. In order to obtain a valid result, the study performed for each block should be done for the different data rates and the software must adapt them in a suitable way.

A really final observation regards the fact that increasing the data rate translates in a decreasing in the number of samples per symbol. First of all, this means that the data rate cannot physically increase up to a certain value and secondly, in the choice of the optimum sample a higher error is made since the system is implemented without making interpolation between samples. Obviously, the performance seen in Figure 6.10 can be increased taking into account these remarks.

6.5 CUSTOM IP CORE

This last paragraph provides an overview about the development flow of accelerators IP cores to implement in hardware, within the Zynq programmable logic, part of the architecture previously discussed.

As already anticipated in Chapter 2.6, the Programmable Logic of the Zedboard is based on the Xilinx 7-Series FPGA fabric and offers the designer the ability to implement their own custom logic which can work alongside the software running on the processor cores [7].

For designing and testing custom IP blocks, the Xilinx Vivado tool suite is used. A block of custom logic, in the PL, can be created, adding control and status monitoring capabilities by using memory mapped registers which the processors can access via the AXI4 interconnect.

The AXI4 signals are completely flexible from the point of view of the VHDL design. During the creation of a Xilinx IP block, the Vivado tools can be used to map each AXI signal onto the signal name that the designer used when creating the IP.

The first block implemented in hardware is the MA Filter. This thesis does not treat how the implementation in VHDL is done, but some observations are reported here.
First of all, the logic followed in the filter implementation is not the same as for the software one (Figure 3.22). If the same logic is used, the output of the filter can be obtained in one shot (only 1 clock cycle) since it is possible to use N multipliers and 1 adder with N inputs simultaneously (N is the number of filter taps). However, the resource utilization associated with this extremely parallel architecture would impact significantly the chip utilization.

For this reason, the logic followed is to use the same number of multipliers, but more adders with less inputs. In this way, more clock cycles are required, but less resources are wasted. As already seen in the performance results of the filter implementation, the hardware implementation is much faster than the software one.

In the previous Paragraph, it was given a reference of the importance to have an estimation of the SNR and it was said that the calculation of the signal's variance is essential. The calculation of the variance of a complex signal is quite expensive in terms of operations that must be executed. For this reason, also this block is implemented in hardware, resulting in a significant reduction of clock cycles needed to perform the operation. The same logic used for the software implementation is used here.

Lastly, an overview on how to add the two blocks in a simple architecture is given.

In the following picture is shown a basic block design in Vivado to which, the two peripherals are inserted

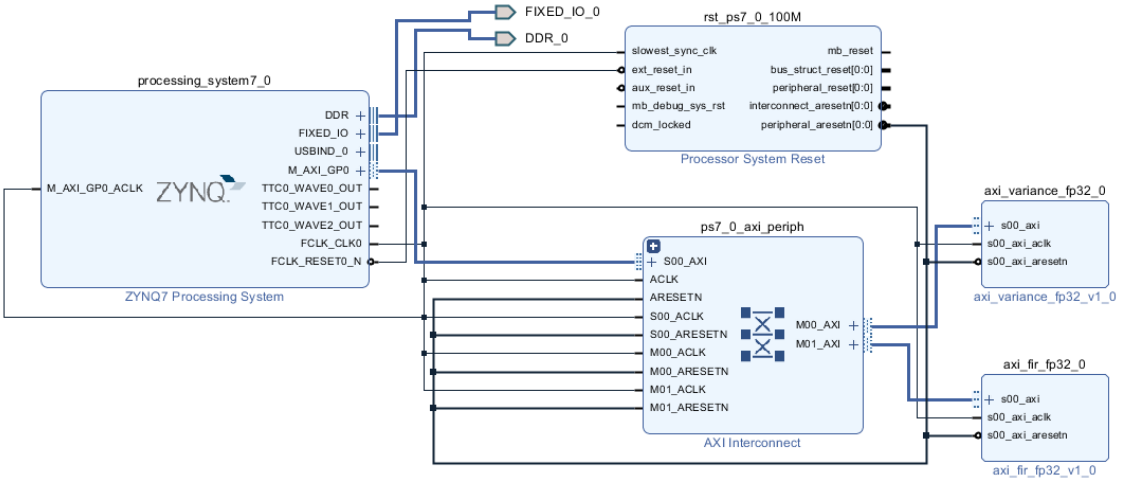

Figure 6.11 – Block Design with Vivado tool

In the block design, the Processing system is essential and it must be connected in some way to the custom peripherals. After, these blocks are added, running the *Connect Automation* command, the AXI Interconnect and the PS Reset are automatically added and connected to the PS and to the peripherals. These blocks are needed to properly interface the Zynq PS with the AXI4 peripherals.

After this important step, the design must be Validated, Synthetized, Implemented and, finally, the Bitstream can be generated [21].

At the end of the design process, it is possible to export the hardware platform, which can be used in Xilinx Vitis tool that is the environment used to write the application to be run on the Zedboard and to provide software drivers for the hardware configuration just presented.

CHAPTER 7

7 Final Remarks

This last Chapter summarizes the objectives of the thesis focusing on achieved results. Then, a list of topics that could be farther analyzed and studies in deep are presented as future work.

7.1 CONCLUSIONS

In this thesis work the implementation of a Software Defined Radio for a Satellite Radio System has been presented. The main goal to achieve was to obtain a Receiver prototype able to receive, process and decode a signal that is as close as possible to a real one.

At this purpose, a Receiver architecture has been analyzed and the design was adapted to the specifications that are chosen for the prototyping.
Simulation analysis, for each block of the receiver chain, were made performing simulations in MATLAB environment, creating a first implementation that was not able to work with a real-time signal, but able to process a signal having some specific characteristics.

Finally, a prototype was implemented on a development board with good results, which are very near to the ones extracted from the design phase. The prototype was tested and validated providing a solid basis for future developments.

Here, it is reported a summary of the objectives proposed in Chapter 1.2 focusing on the obtained results:

- ***Literature study of a Satellite Radio System focusing on the Receiver structure***: the study on a typical coherent satellite receiver gave information about its architecture. Following the CCSDS Standard, a series of specification about the transmitted signal in a TC system are fixed and they were a guideline in the modeling of the initial proposed Rx architecture;
- ***Study of a Base Station Transmitter to generate an Uplink signal, able to send realistic data***: it was chosen to use the software GNU Radio in order to model the transmitted signal basing on the fixed specification. This choice allowed to make a flexible transmitter structure that is able to save the signal as a stream of bits in a file, for the first part of simulation in MATLAB, but also to send the signal in real-time exploiting different interfaces for next implementation on the development board;
- ***Feasibility analysis of the proposed architecture through MATLAB simulations***: MATLAB was used as simulation environment. The receiver chain is implemented, optimizing each block and tuning their parameters for adapting them to the transmitted signal. A study on their performance has been done, choosing the best solution in the design process;
- ***Implementation on a Development Board able to work in real-time***: once the architecture has been frozen, all the designed blocks were implemented in C language on a development board;
- ***Performance measurements of each block finding which of them can be implemented in hardware***: two blocks that required too much resources, if implemented in software, were exported in hardware;

- ***Test of the prototype simulating a real environment condition***: the prototype was tested and validated through performance analysis. The receiver is also tested in harsh conditions, changing the SNR and adding some impairments to the transmitted signal.

7.2 FUTURE WORK

The test campaign of the Receiver prototype is still on-going and it is needed to assess in a more defined way the system performance and the system specifications.

When the specifications of the system will be defined in detail, the structure of the Telecommand System can be frozen for a future realization of the RF part of the receiver. This will allow to study the behavior of the baseband part of the receiver with the real impairments added by non-idealities of the RF front-end.

After that, following the same steps for the design and implementation of the TC system, the Telemetry system will be taken into account. Transmitter structure, MATLAB simulations, performance measurements and implementation on a development board will be performed.

The two projects will be merged together in order to have a complete satellite transponder prototype.

Bibliography

[1] Gara Quintana-Diaz, Roger Birkeland, "*Software-Defined Radios in Satellite Communications*", Norwegian University of Technology and Science, May 2018

[2] CCSDS Blue Book, "*TC synchronization and Channel Coding*", CCSDS 231.0-B-3, September 2017

[3] *"Communication Systems / Coherent Receivers"*, https://en.wikibooks.org/wiki/Communication_Systems/Coherent_Receivers

[4] *"GNU Radio"*, https://wiki.gnuradio.org/index.php/Main_Page

[5] *"AWGN Channel"*, https://it.mathworks.com/help/comm/ug/awgn-channel.html

[6] Zynq-7000 SoC, "*Technical Reference Manual*", UG585 (v1.12.2), July 2018

[7] Rich Griffin, Silica EMEA, "*Designing a Custom AXI-lite Slave Peripheral*", Version 1.0, July 2014

[8] Deep Space Network, "*Pseudo-Noise and Regenerative Ranging*", 810-005 214, Rev. B, 2019

[9] Lydi Smaini, "*RF Analog Impairments Modeling for Communication Systems Simulation: Application to OFDM-Based Transceivers*", September 1012

[10] "*Phase Look Loop (PLL)*", https://gssc.esa.int/navipedia/index.php/Phase_Lock_Loop_(PLL), 2011

[11] "*Bilinear Transform*", https://en.wikipedia.org/wiki/Bilinear_transform

[12] "*Building a Numerically Controlled Oscillator*" https://zipcpu.com/dsp/2017/12/09/nco.html, 2017

[13] Timo I. Laakso and Bing Zeng, "*Limit Cycles in Floating-Point Implementations of Recursive Filters – A Reviews*", IEEE International Symposium, June 1992

[14] Erkin Cubukcu, "*Root Raised Cosine (RRC) Filters and Pulse Shaping in Communication Systems*", Friday 2012 https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120008631.pdf

[15] "*Eye Pattern*", https://en.wikipedia.org/wiki/Eye_pattern

[16] Huang Lou and Pingfen Lin, "*A New Lock Detector for Gardner's Timing Recovery Method*", IEEE Transactions on Consumer Electronics, May 2008

[17] G. Karam, V. Paxal and M. Moeneclaey, "*Lock Detectors for Timing Recovery*", International Conference on Communications, June 1996

[18] CCSDS Green Book, "*TC Synchronization and Channel Coding – Summary of Concept and Rationale*", CCSDS 230.1-G-2, November 2012

[19] TU Wien, "*LDPC Codes*", https://www.nt.tuwien.ac.at/wp-content/uploads/2016/10/DC2-16_Ch7_LDPC.pdf

[20] Vikram Chandrasetty, Syed Mahfuzul Aziz, "*Resource Efficient LDPC Decoders*", Academic Press, December 2017

[21] "*Creating a Custom IP core using the IP Integrator*", Digilent, https://reference.digilentinc.com/learn/programmable-logic/tutorials/zedboard-creating-custom-ip-cores/start