Universidade de Brasília - UnB

Faculdade UnB Gama - FGa

Engenharia eletrônica

DVBS2X standard FPGA implementation of LLR, Physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications.

Autor: Pedro Henrique Andrade Trindade

Orientador: Prof. Leonardo Aguayo

Brasília, DF

2019

Pedro Henrique Andrade Trindade

DVBS2X standard FPGA implementation of LLR, Physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications.

Monografia submetida ao curso de graduação em (Engenharia eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGa

Supervisor: Prof. Leonardo Aguayo

Co-supervisor: Prof. Cristiano Jacques Miosso

Brasília, DF

2019

Pedro Henrique Andrade Trindade DVBS2X standard FPGA implementation of LLR, Physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications./ Pedro Henrique Andrade Trindade. – Brasília, DF, 2019- 98 p. : il. (algumas color.) ; 30 cm.

Supervisor: Prof. Leonardo Aguayo

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB

Faculdade UnB Gama - FGa , 2019.

1. . 2. . I. Prof. Leonardo Aguayo. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. DVBS2X standard FPGA implementation of LLR, Physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications.

CDU 02:141:005.6

Pedro Henrique Andrade Trindade

DVBS2X standard FPGA implementation of LLR, Physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications.

Monografia submetida ao curso de graduação em (Engenharia eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia eletrônica).

Trabalho aprovado. Brasília, DF, :

Prof. Leonardo Aguayo
Orientador

Convidado 1

Convidado 2

Brasília, DF
2019

Este trabalho é dedicado à todos aqueles que procuram nas perguntas, acima do que nas próprias respostas, uma tradução satisfatória do universo e da condição humana.

Acknowledgements

Gostaria de agradecer à Universidade de Brasília pela oportunidade de me desenvolver tanto academicamente, como profissionalmente e pessoalmente, proporcionando uma experiência única de aprendizado e crescimento pessoal.

Também gostaria de agradecer ao meu orientador Prof. Dr. Aguayo, que essencialmente, com o passar dos anos e disciplinas decorridas, trilhou o caminho e os fundamentos para que o desenvolvimento deste trabalho fosse possível.

Gostaria de agradecer meu avô Gabriel Riograndino Trindade, que me forneceu o dom da curiosidade e incitou uma animosidade distinta voltada à obtenção de conhecimento.

Meus primos e minha tia Angélica Trindade por serem a base do meu conceito de família, serem meu suporte tanto emocionalmente quanto mentalmente, mantendo-me firme e em pé no cumprimento destes cinco anos importantes de estudo.

Ao restante dos familiares, gostaria de encarar este trabalho como uma maneira de demonstrar a capacidade de resultados e indubtável necessidade de investimento em pesquisa.

À minha namorada, Larissa Aguiar, gostaria de agradecer profudamente por apoiar minhas escolhas e ser uma parte compreensiva importante em todo este extenso e provavelmente infinito processo de desenvolvimento pessoal.

Gostaria de agradecer a todos os colegas e pessoas que passaram pelo meu caminho e me trouxeram importantes experiências para constituir o adulto que me tornei.

Por final gostaria de agradecer à todos os docentes e técnicos da UNB que após longos anos de apoio incondicional, se tornam imprescindivelmente um pilar na construção de toda a minha formação acadêmica.

“Apesar de tudo,
eu levantarei de novo,
pegarei meu lápis do qual esqueci em meio à desaprovação,
e continuarei à desenhar.”
(Vincent Van Gogh)

Abstract

Within the paradigm of digital communication, there exist the context of software-defined radios, in which many advantages can be found over classical non-reconfigurable radios, for example, the rise in flexibility, that as consequence causes ease in maintenance and project error repair. A structure that adds to this scheme applying itself in a plethora of contexts and showing unique application characteristics is the Field Programmable Gate Array (FPGA) chips. In this text, it was projected three blocks from the DVBS2X standard for an SDR with the main application objective being satellite communication in Ku band. The blocks are: Symbol timing synchronization, that deals with the matching between the transmitted quadrature waveform sampling time and the receiver sampling time, based on a close-loop non-data aided Gardner Timing Error Detection algorithm. Log Likelihood Ratio block, that is a maximum à posteriori symbol estimator based on the received symbols corrupted by the AWGN channel. Symbol Physical Layer Descrambling, that makes the opposite operation from the Physical Layer descrambling defined for the transmitter in the DVBS2X protocol, multiplying the received symbol by a complex number defined with a golden sequence, implemented with two linear feedback shift registers in FPGA. The results were favorable with hardware architectures chosen and projected, each block was design with a Top-down design methodology and validated with vector files coded in high-level languages such as PYTHON and MATLAB. This text proposed and implemented three hardware solutions with synthesizable hardware complexity, based on the DVBS2X standard.

Key-words: FPGA. SDR. Symbol Timing Recovery. DVBS2, Log Likelihood Ratio, Physical Layer Scrambling, Satellite Communication, Software Defined Radio, VHDL.

Abstract

Dentre o paradigma da comunicação digital, existe o contexto de rádio definido por software, que pode trazer muitas vantages sobre os modelos de radio classicos não reconfiguráveis, por exemplo, o aumento na flexibilidade, que como consequencia traz uma maior facilidade em manutenção e conteção de erros de projeto. Uma estrutura que adiciona à morfologia se aplicando em um conjunto volumoso de contextos e mostrando características unicas de aplicação são os chips *field programmable gate array* (FPGA). Neste texto foram projetados três blocos do protocolo de comunicação DVBS2X para um SDR em que a aplicação principal é a comunicação por satélite em banda Ku. Os blocos são: *Symbol timing synchronization*, que lida com a correspondência o tempo de amostragem entre o sinal em quadratura transmitido e o recebido, baseando-se em um algoritmo *closed loop non data-aided Gardner Timing Error Detection.* O bloco *Log Likelihood Ratio*, que é um detector *maximum à posteriori* que estima o símbolo transmitido baseando-se nos símbolos recebidos corrompidos pelo canal AWGN. *Symbol Physical Layer De-scrambling*, que faz a operação oposto do bloco textitPhysical Layer Scrambling definido para o transmissor pelo padrão DVBS2Xm multiplicando o símbolo recebido por um número complexo definido por uma *golden sequence*, implemenetando dois *linear feedback shift registers* na FPGA. Os resultados foram favoráveis com arquiteturas de hardware escolhidas e projetadas, cada bloco foi projetado com uma metodologia *Top-down* e validado com arquivos vetores codificados em linguagem de alto nível como PYTHON e MATLAB. Este texto propôs e implementou três soluções em *hardware* viáveis com complexidade de implementação razoáveis, baseando-se sempre no protocol padrão e comparando com outros exemplos da literatura acadêmica.

Palavras-chaves: FPGA. SDR. Symbol Timing Recovery. DVBS2, Log Likelihood Ratio, Physical Layer Scrambling, Comunicação por satélite, Radio Definido por Softwae, VHDL.

List of Figures

List of symbols

$\Pi_T(t)$	Rect function of size T.
$sinc(t)$	Cardinal sine.
$\overset{\mathcal{F}}{\leftrightarrow}$	Fourier transform.
σ	Standard Deviation.
ω	Angular Frequency.
β	Roll-off factor.
$\triangleq$	Defined as.
$H(f)$	Filter Frequency Response.
τ	Generic Time Delay.
μ	Fractional Time delay.
ζ	Dampening Factor.
η	NCO counter.
ϕ	MPSK phase.
γ	APSK ring ratio.
$\bigcup$	Set union.
ρ	distance of a point constellation in APSK.

Contents

1 Introduction

1.1 Contextualization

Software-defined radio is a system paradigm for wireless communication, it's the nomenclature for a class of radios that can be reprogrammed and reconfigured in software. There are many pros and cons to this kind of approach, for example, an increase in flexibility, that as consequence causes ease in maintenance and error repair. This happens because unlike non-reconfigurable radios, SDRs enable a broader range of applications as the parameters can be modified with the demands of the system evolution. Most conventional radio have a fixed hardware-based communication standard, in which the variables are not accessible to be modified.

A structure that adds to this category applying itself in a plethora of contexts and showing unique application characteristics is the Field Programmable Gate Array (FPGA) chips. This architecture allows being modified by external control, often made statically, via USB cable and specific programming software, as in VIVADO and XILINX, or dynamically, usually helped by a micro-controller, changing in real-time.

The FPGA space in the development of software-defined radios is determined, among many reasons, as explained the article (HOSKING, 2012), by the appeal of meeting project requirements, because the FPGA processing is as intense as digital signal processors(DSPs) and is parallelizable as low-level non-reconfigurable hardware architectures such as ASICS, while also showing a reasonable flexibility:

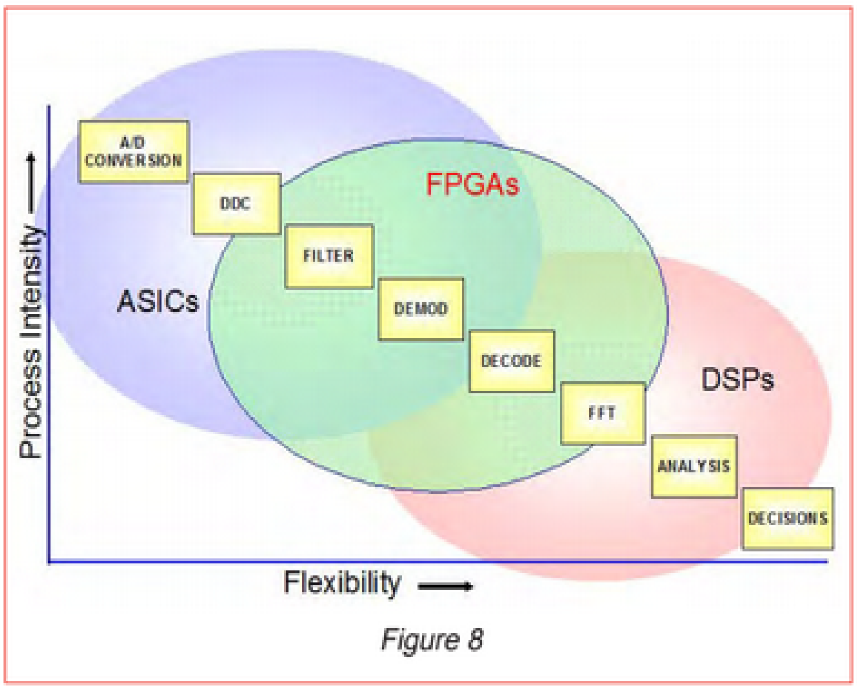


Figure 1 – FPGA space of use in SDRs. (HOSKING, 2012)

Figure 1 represents that frequently tasks as an analog-to-digital conversion is given to ASICS, because its simple operation architecture results in real-time processing, while to DSPs normally are given tasks of analysis and decision.

The part that encloses the FPGA tasks reaches these two different fronts, and the entire radio domain. A simplified wireless transceiver model can be seen in 87:

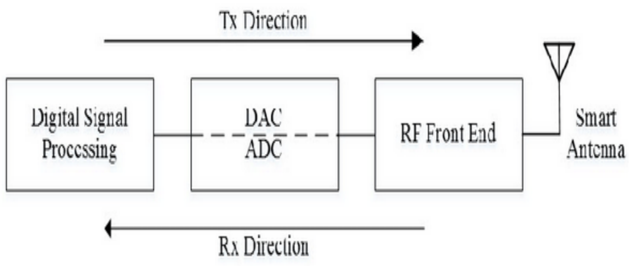


Figure 2 – Simplified model of wireless transceiver. (SADEK et al., 2015)

In this case, in figure 2 ASICs are used to interface with an antenna, a digital to analog conversion (D/A) and an analog to digital conversion (A/D), leaving space for the FPGA to act as an DSP, while expressing functional advantages, as the parallelism.

In the expansion of the satellite wireless communication market there is a trend that aggregates to the set of SDR application scenarios, which is the ascension in the number of Smallsats launched to orbit in these last years. The projection implies an increase for the next years, as cited in the article "Status and Trends of Smallsats and Their Launch Vehicles" (WEKERLY et al., 2017). There is a sustainable demand for the development of communication systems for satellites, given that the maintenance process, control , and data acquisition implies a necessity of microelectronics implementation and establish a communication model consistent with the system requirements.

The article (WEKERLY et al., 2017) also supplies a statistical projection over the trends in minisatellites growth over the next few years, shown in figure 3:

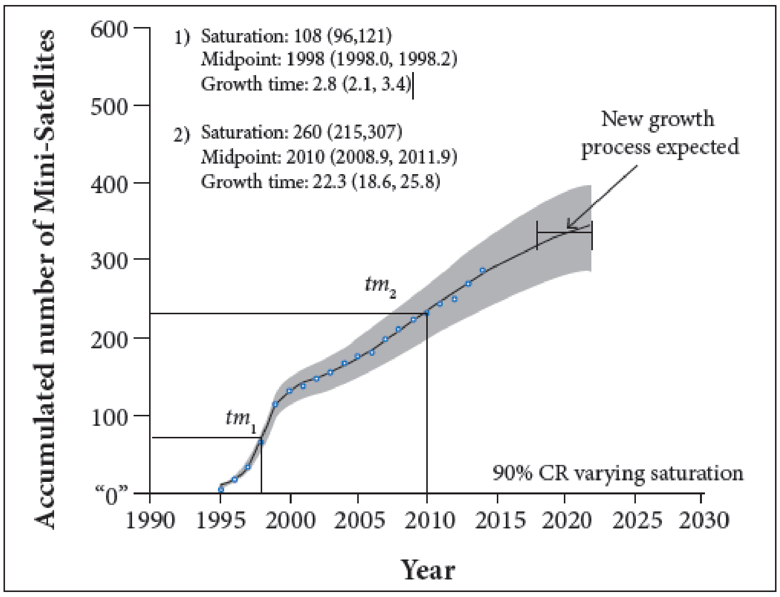


Figure 3 – Projection model of the number of minisats (101-500 kg) in orbit in space. (WEKERLY et al., 2017)

Adding this information to a possible application associated with FPGA use, favorable arguments can be found to adopt this paradigm in the project implementation. Accordingly to (BOWYER et al., 2015) the main arguments to use FPGA in SDR satellite RF communication are:

1. Flexibility to support multiple satellites;
2. Scalability, extensibility and modular design to support an evolution over time;
3. Ability to permit new characteristics and services (new protocols) without a hardware upgrade;

The FPGA, or field-programmable gate array, is, under the definition of (XILINX, 2018) a hardware architecture category that proposes a paradigm where the nature and structure of its working can be configured by the user or designer after its fabrication. These systems are based in a block matrix of configurable logic block (CLN) under configurable interconnections. Each CLB has a LUT, or lookup table, which is a portion of RAM indexed in the input or set of inputs, in a way to be possible to create any configuration of truth table necessary. Showed in figure 4:

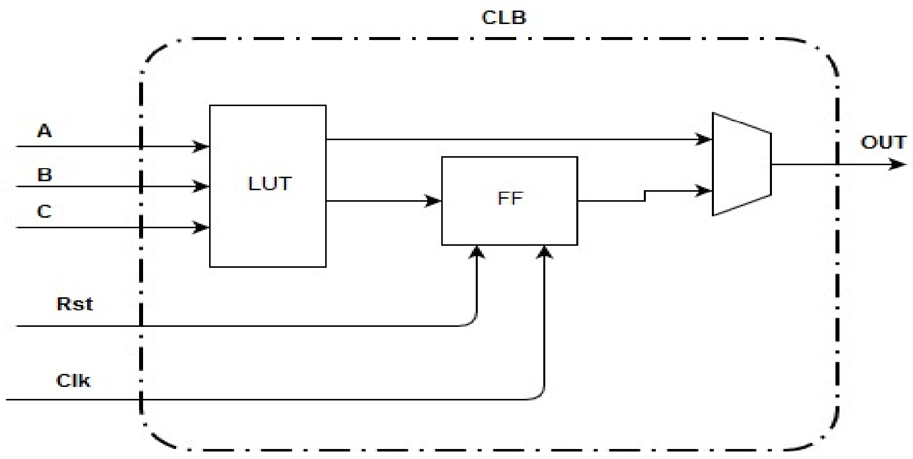


Figure 4 – Block diagram of a CLB.

It's called ASIC, or application-specific integrated circuits, hardware that already comes pre-programmed from the factory.

Created by the co-founder of the company Xilinx INC, Ross Freeman, the FPGA is broadly used to digital data processing, drawing attention over its parallelization capacity. Being able to the define its interconnections, having a more flexible set of possible practical applications.

1.2 Objectives

1.2.1 General Objective

Design three block of the DVBS2X protocol in FPGA: Symbol timing recovery, log likelihood ratio and physical layer de-scrambling block.

1.2.2 Specific Objectives

- Define and implement a solution in FPGA to those three blocks, while validating within its functional characteristics and curves.
- Simulate a noisy and time delayed input while verifying the effects in the general system.
- validate the output of each block within the defined methodology, which is explained in its specific section.

- Effectively correct both timing delay and AWGN noise in DVBS2 constellations.

As a mean to adequate to a standard protocol, it's first necessary to define the system building blocks. The next section explains the components that make up the system from a general point of view to the specific implementation in FPGA.

1.3 Digital communication principles

Digital communication as it is conceived today, with the abrupt technological ascension and the absolute presence both directly and indirectly in every corner of society and economic sectors, was defined after a continuous evolution process passing from analog-based systems, where many concepts and definitions are derived.

In the project of a modern communication system, digital and viable, both conceptually and in practice, it's necessary to start over with precise definitions. An example of a communication system is given by (SKALAR, 2017), that has the building blocks of what's needed to define a communication link:

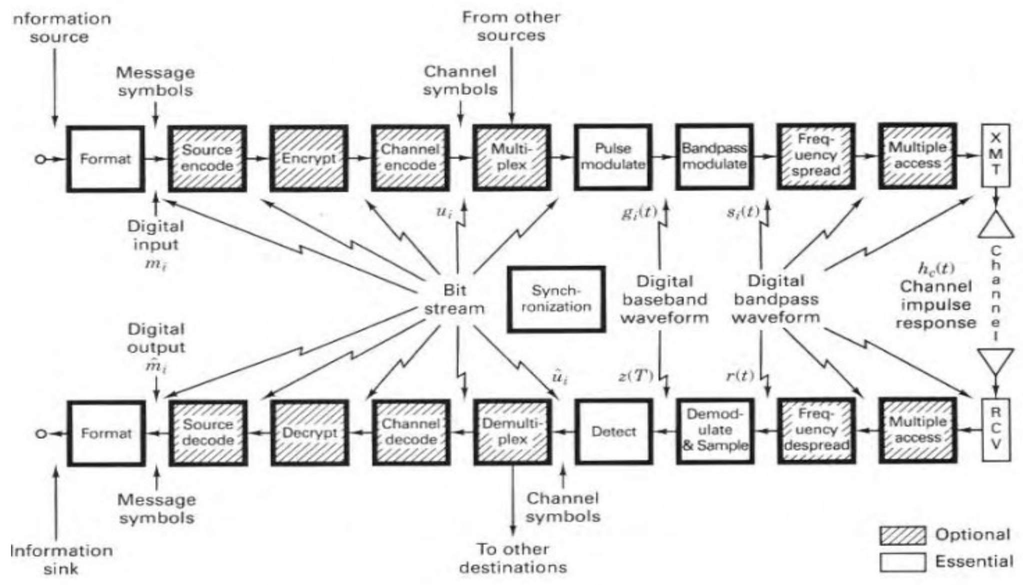


Figure 5 – Digital communication block diagram. (SKALAR, 2017, p.104)

The blocks of figure 5 can be simplified, showing systematically in a more general way the point-to-point link, showed in figure 6:

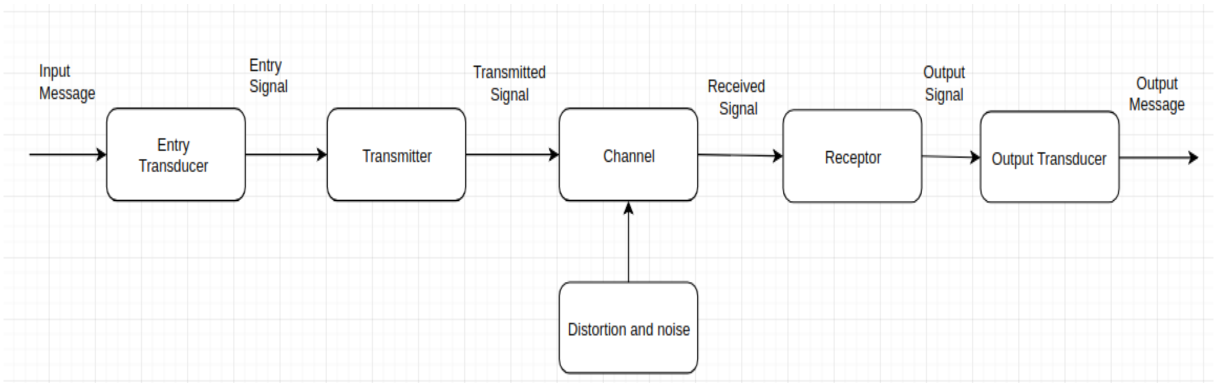


Figure 6 – Simplified communication system diagram. adapted from (LATHI, 1995)

Figure 6 model shows the fundamental structure of a communication system, being a digital communication system we translate the input and output transducers as digital information.

In the transmission block, it's necessary to define a modulated waveform, a step that determines the overall working of each block next. Because every block is intrinsically connected to the model as a whole there is a relatively large set of possible modulations to choose, many analogs, such as DSB + C, DSB-SC, SSB, VSB, FM, PM, and digital such as PAM, PWM, ASK, PSK, FSK e QAM, among others.

The channel block is where it is applied characteristic noises existent in a physical electronic component, also distortion effects, such as dampening and delay. It's possible then to think about line coding and pulse formatting, to avoid the error propagation to the next levels.

As a matter of project choice, based on the commercial availability, performance and to standardize the implementation, it was chosen to frame the Digital Video Broadcasting -Satellite- Second Generation (DVBS2) as a reference, this is a standard protocol used in many applications, such as digital television, HDTV and professional data content distribution.

A transmission block diagram of the DVBS2 standard can be found on (ETSI, 2014a):

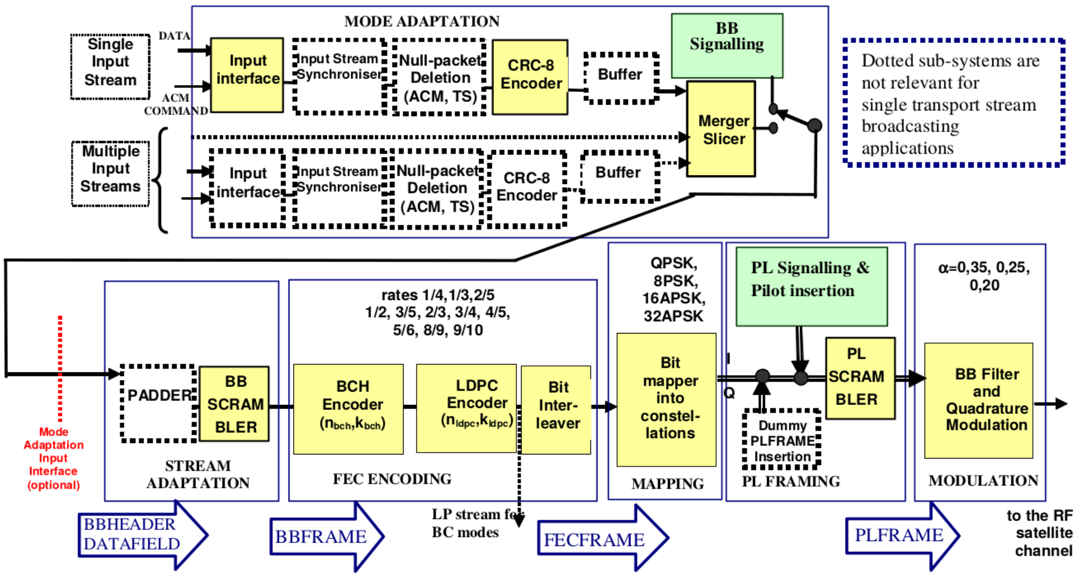

MODE ADAPTATION
Single Input Stream
DATA
ACM COMMAND
Input interface
Input Stream Synchroniser
Null-packet Deletion (ACM, TS)
CRC-8 Encoder
Buffer
BB Signalling
Merger Slicer
Multiple Input Streams
Dotted sub-systems are not relevant for single transport stream broadcasting applications
Mode Adaptation Input Interface (optional)
PADDER
BB SCRAM BLER
STREAM ADAPTATION
rates 1/4,1/3,2/5 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10
BCH Encoder (n_{bch},k_{bch})
LDPC Encoder (n_{ldpc},k_{ldpc})
Bit Inter-leaver
FEC ENCODING
QPSK, 8PSK, 16APSK, 32APSK
Bit mapper into constel-lations
MAPPING
PL Signalling & Pilot insertion
PL SCRAM BLER
Dummy PLFRAME Insertion
PL FRAMING
α=0,35, 0,25, 0,20
BB Filter and Quadrature Modulation
MODULATION
BBHEADER DATAFIELD
BBFRAME
LP stream for BC modes
FECFRAME
PLFRAME
to the RF satellite channel

Figure 7 – Functional block diagram of the DVBS2 system. (ETSI, 2014a)

Two features that exhibits an advantage in this protocol is that it has a powerful coding scheme represented by its forward error correction (FEC) encoding block, that concatenates a BCH code with an LDPC code, one hard decision and the other soft decision, the other advantage is that it has an adaptive coding and modulation (ACM), which allows a better bandwidth use, changing the system parameters dynamically in link. There are four different modulations, three roll-off factors and many code rates, as it's list in table 1, based in (ETSI, 2014a, p.14):

System Configuration	
QPSK	1/4,1/3, 2/5
	1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10
8PSK	3/5, 2/3, 3/4, 5/6, 8/9, 9/10
16APSK	2/3, 3/4, 4/5, 5/6, 8/9, 9/10
32APSK	3/4, 4/5, 5/6, 8/9, 9/10
FECFRAME (normal/ short)	64 800, 16 200 (Bits)
Roll-off	0.35, 0.25, 0.2

Table 1 – DVBS2 possible combinations of Modulation and code rates (MODCOD), frame sizes and Roll-off values

The Digital Video Broadcasting -Satellite- Second Generation Extended (DVBS2X) is a protocol that is an extension of the DVBS2, adding some features such as a very low signal-to-noise ratio mode (VL-SNR) giving it the ability to operate with very-low carrier-to-noise and carrier-to-interference ratios, with SNR to -10 dB, also new modulations and code rates.

The DVBS2X manual does not show an explicit diagram for the receptor, as a matter of fact, it leaves implied, but one can find in the overall articles that exemplify common implementations, as the article (LIMA et al., 2014), which was the reference that determined the block diagram of the receptor system:

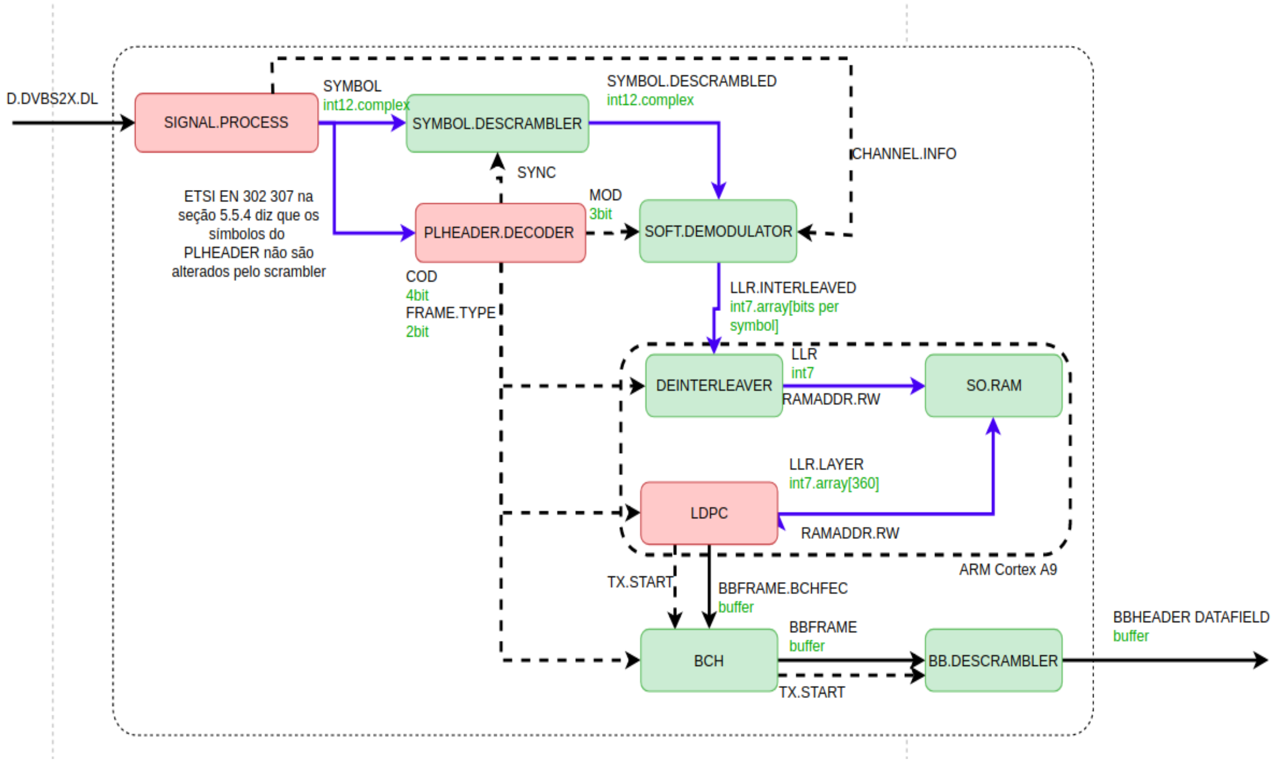


Figure 8 – Functional block diagram of the receptor.

The main objective is to implement the blocks: Symbol timing recovery (STR), symbol de-scrambling and soft demodulation block.

Inside the first signal processing block is the STR, the coarse and fine frequency correction will not be in the implementation scope of this text.

The symbol de-scrambling block multiplies the received symbols by a set of pre-determined complex numbers defined by a golden sequence given by the protocol manual.

The soft demodulation block calculates the LLR (Log-likelihood ratio) for a given received symbol.

Those blocks will be explained more thoroughly in its specific sections, but firstly, it's necessary to explain some of the basic principles that will be utilized over all this text, beginning with noise analysis, pulse shaping and ending with the modulation/demodulation techniques that will model all design choices within this text scope.

.

1.4 Additive White Gaussian Noise

Under the electronics scope, associated with the continuous advance of the technological ascension, there is the existence of inherent noise, of many natures, acting inside innumerable systems, many of which are sensitive to these perturbations.

Every system, with a plethora of different goals, being telecommunication, signal processing, electrical systems, among others, all of them are affected by the same condition related to the non-idealities found in the context of application, because there is, both intrinsically and inevitable a non-deterministic floating factor that affects the system, adding a degree of uncertainty inside the project scope of the circuit.

In noise analysis applied to circuits, there is overall a necessity to define in a clear and exact way what is being understood as noise, accordingly to (VASILESCU, 2006) the term noise is used excessively and many times in an erroneous way, because many definitions were proposed over the years, as a counterpoint, the author proposes some traditional definitions that have been more relevant, it's going to be used the following definition taken from the standard IEEE dictionary of electrical and electronics terms:

"An undesired perturbation superimposed over a wanted signal that tends to obscure its information contents."

There is an extensive quantity of different noises, classifications and models, under the domain of this text the model in focus will be the additive white Gaussian noise (AWGN). This noise is modeled in a way to approximate in a sufficiently precise manner the perturbation behavior that manifests inherent to the circuitry used. Due to the electron motion and collisions inside the crystal lattice of the building material, as a consequence of temperature, Those movements generate an instantaneous charge imbalance inside the material because there isn't a static equilibrium and the electron quantity doesn't maintain itself homogeneous under all the material length, generating a random voltage, leading to a necessity of employing certain statistical manipulation in the problem.

Mathematically the behaviour of the additive white Gaussian noise is inferred based in a sequence of empirical observations, that are:

1. The noise is additive, that is, if a wanted signal $f(t)$ exists and a noisy signal is defined as $f^{'}(t)$, then:
$$f^{'}(t) = f(t) + n(t), \tag{1.1}$$
where $n(t)$ represents the noise. This means that when a noisy signal is linearly amplified, the noise is also amplified, or
$$Af^{'}(t) = Af(t) + An(t). \tag{1.2}$$

2. Even with the noise behaviour being completely random, it's average is zero, which can be denoted as
$$E[n(t)] = 0. \tag{1.3}$$

3. The voltage amplitude recurrence have a predictable behaviour, having a Gaussian probability density function (PDF), which is given by the equation
$$\mathcal{Q}(v) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{v^2}{2\sigma^2}}, \tag{1.4}$$
where σ is the standard deviation, determined by the average noise power σ^2 that is reached with the designed system parameters and the environmental work conditions. This function have the plot:

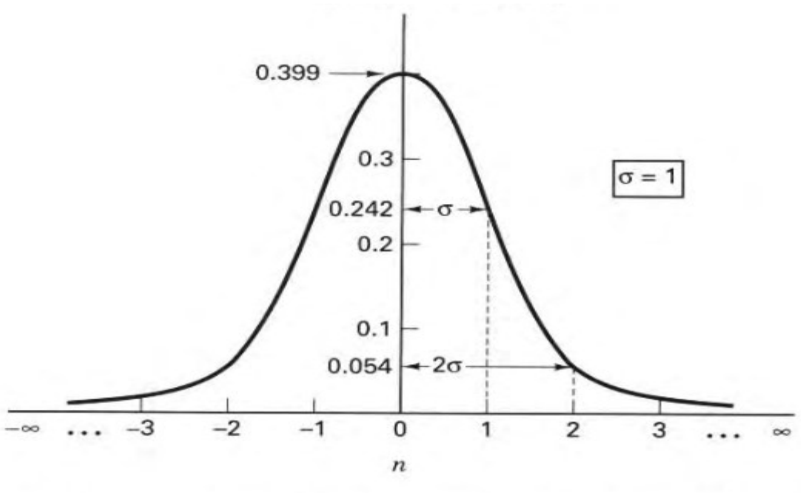


Figure 9 – Normalized Gaussian curve($\sigma = 1$). (SKALAR, 2017, p.31)

One can notice that the larger amplitudes occurs with less frequency, while the little variations occur with higher incidence.

4. The autocorrelation function can be defined as
$$R(\tau) = \int_{-\infty}^{\infty} f(t)f(t+\tau)d\tau, \tag{1.5}$$
this function has an abundance of notable aspects, like its Fourier transform represents the power spectral density function (PSD) of the inputted function, that is
$$\begin{aligned} f(t) &\leftrightarrow F(w), \\ R(t) &\leftrightarrow |F(w)|^2 . \end{aligned} \tag{1.6}$$

In the case of AWGN noise, the autocorrelation function is proportional to an impulse function (also know as Dirac's delta distribution), denoted by $\delta(t)$, which means that the noise is uncorrelated with itself and given the (1.6) relationship it's possible to predict that the average power density of the system is given by a constant in the domain of all frequencies.

This is an efficient model for telecommunications, but there is an intrinsic problem to this analysis when it's used in a high frequencies approach, because then we would have the noise signal power tending to infinity, what in a real physical world would be impossible. This effect is denominated as "ultraviolet catastrophe" and was eventually solved by quantum mechanics. But to all effects to these relatively small bandwidth problems, this model approximates reality very well.

5. The constant $\mathcal{N}_0$ is proportional to the temperature, more specifically $\mathcal{N}_0 = kT$ where k is the Boltzmann constant ($\sim 1.38 \times 10^{-23}\ J/K$) and T represents the equivalent noise temperature in Kelvin. The power spectral density function represents a quantity of power by hertz, integrating in the bandwidth frequency domain, it's possible to see that the average noise power is given by

$$\sigma^2 = \mathcal{N}_0 B = kTB. \tag{1.7}$$

With this result it's possible also to find the variance of the eq. (1.4).

This modeling is important for digital communication systems because one can determine the behavior and distinct error patterns in digital modulations and line codification. Over this model, it's possible to infer an error correction strategy for measuring the system transmission quality, quantifying as a BER plot or bit error rate.

This bit error plot usually describes a curve of error probability in function of the bit energy, or the energy necessary to send a bit in dB, divided by N_0 ($\frac{E_b}{N_0}$).

The Gaussian noise is the base noise to analyze the signal integrity of a given system, allowing many strategies for robust transception.

1.5 Rooted Raised Cosine Filter

Given a problem with a bandwidth limit, as usually is required by the solution, it's frequently necessary to use a more advanced tool-set for pulse shaping.

The problem in using a square pulse is that this type of pulse has an abrupt discontinuity in its border, implying high-frequency components in Fourier domain representation. This determines a theoretical infinite bandwidth in the frequency domain. Formally, it's shown that

$$\Pi_{T_s}(t) \overset{\mathcal{F}}{\longleftrightarrow} T_s sinc(fT_s). \tag{1.8}$$

But fortunately there is a duality in the Fourier transform, this duality shows that one can contour the pulse bandwidth problem using a *sinc* (cardinal sine) shape time function, with this shape the expected bandwidth in frequency is limited, as the Fourier transform gives a *rect* function.

Under this perspective there are some challenges to overcome, the first one is that the *sinc* function is an infinite domain signal, the second is that it's a non-causal signal, there is, there are parts in this signal before the time defined as zero, lastly the *sinc* function converges very slowly to zero in amplitude, implying more chance of inter signal interference (ISI).

To solve the slow convergence problem there is a specific function that resembles the *sinc* but falls to zero rapidly, this function is called *raised cosine* filter, or RCF, and it's defined as:

$$h(t) = \begin{cases} \frac{\pi}{4T_s} sinc\left(\frac{1}{2\beta}\right), & t = \pm\frac{T_s}{2\beta}. \\ \frac{1}{T_s} sinc(\frac{t}{T_s}) \frac{\cos\left(\frac{\pi\beta t}{T_s}\right)}{1-\left(\frac{2\beta t}{T_s}\right)^2}, & \text{otherwise.} \end{cases} \tag{1.9}$$

Under this definition there are some important parameters described, the first one is the symbol period T_s, in which this curve is parametric to, in manner to not occur inter-symbolic interference in transmission, there is, respecting the Nyquist criteria for null ISI, that is

$$h(nT_s) = \begin{cases} A, & n = 0, \\ 0, & n \neq 0. \end{cases} \tag{1.10}$$

It's guaranteed by this theorem that if a pulse is zero in an integer multiple of the symbol period, then there is a zero ISI condition.

There is another important parameter, also called roll-off factor or excess bandwidth, that defines the system bandwidth: The β value. This value determines how far in bandwidth the system is from an ideal rectangular pulse.

The plot of the RCF Fourier transform is:

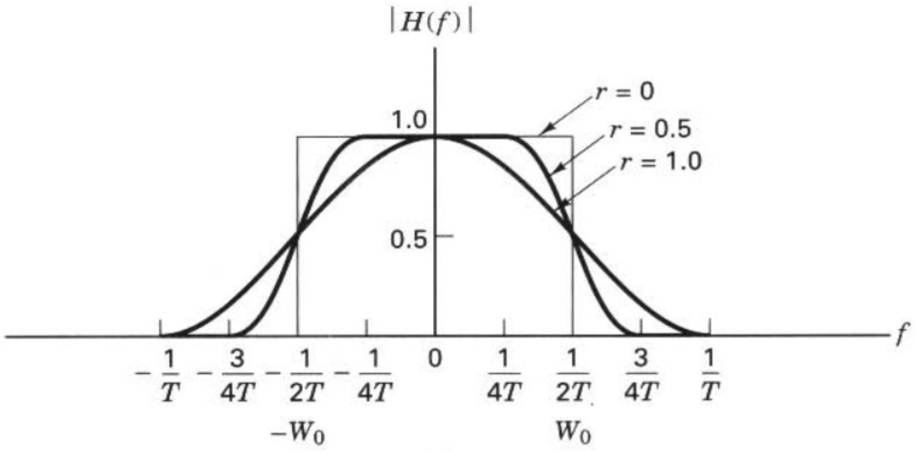


Figure 10 – Raised cosine pulse Fourier theoretical transform for different roll-off factors. (SKALAR, 2017, p.140)

One can notice in figure 10 that there are some frequency components remnants that go to zero in $\frac{1+\beta}{2T_s}$. When the roll-off factor is smaller, the closest the bandwidth is from an ideal form, or $\beta = 0$.

Lastly, to solve the non-causality problem, it's possible to delay the time signal in the form $h(t-d)$ and truncate the signal in values of time before 0. After a certain previously specified time it's performed a windowing of the signal, adding a phase. If the truncation is made in a way that the before-zero time components are close to zero, there is no change either in shape or absolute amplitude of the function in the frequency domain.

When the digital communication process as a whole is understood the RC pulse shows its importance. The pulse shaping is a vital component to guarantee information transmission integrity.

In the DVBS2X standard (ETSI, 2014a, p.33) a variation of the RCF function is applied, called *squared root raised cosine* filter, or SRRC, which is defined with the property

$$H_{rc}(f) = H_{srrc}(f).H_{srrc}(f), \tag{1.11}$$

which in the DVBS2 standard it's frequency form is defined as

$$H_{srrc}(f) = \begin{cases} 1, & \text{for } |f| < f_N(1-\beta) \\ \frac{1}{2} + \frac{1}{2} sin\left(\frac{\pi}{2f_N}\left[\frac{f_N - |f|}{\beta}\right]\right), & \text{for } f_N(1-\beta) < |f| < f_N(1+\beta). \\ 0, & \text{otherwise.} \end{cases} \tag{1.12}$$

In the DVBS2 protocol there is 3 different possible β factors: 0.35, 0.25 and 0.2. These pulses were simulated in PYTHON language, giving the plot in figure 11:

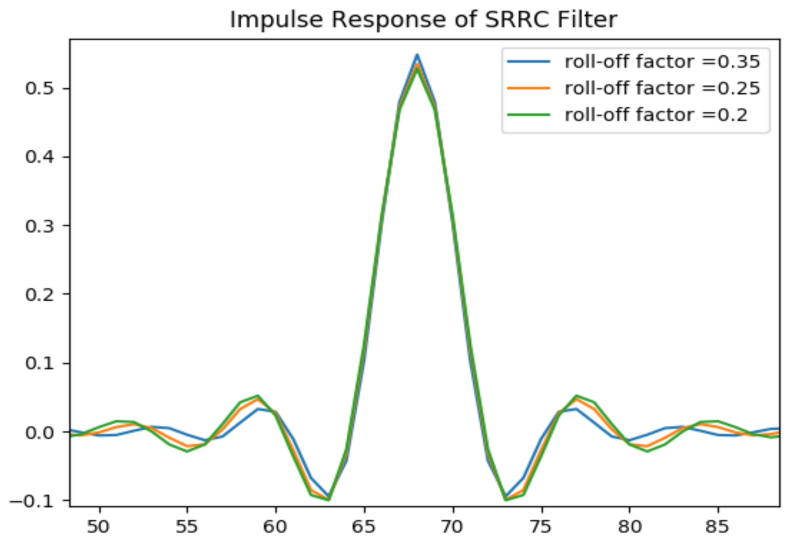


Figure 11 – Impulse response for the three roll-off factors defined in DVBS2 protocol.

1.6 Matched Filter and Modulation.

As explained in (SKALAR, 2017, p.173), the analytic expression for PSK (also ASPK) is

$$s_i(t) = \sqrt{\frac{2E_i}{T_s}} \cos\left(\omega_0 t - \phi_i\right), \tag{1.13}$$

where E is the symbol energy, T_s is the symbol period, w_0 is the carrier frequency and ϕ_i, $i = 0, 1, 2..\ M$ is the phase defined for each symbol representation.

Based on the equation 1.13 and with the trigonometrical identity

$$A.\cos(a - b) = A\cos(a)\cos(b) + A\sin(a)\sin(b), \tag{1.14}$$

then $s(t)$ can also be represented as

$$s_i(t) = \sqrt{\frac{2E_i}{T}} \left[\cos(\phi_i)\cos(w_0 t) + \sin(\phi_i)\sin(w_0 t)\right]. \tag{1.15}$$

If we define $\sqrt{E_i}\cos(\phi_i) = a_{i1}$, $\sqrt{E_i}\sin(\phi) = a_{i2}$, $\sqrt{\frac{2}{T_s}}\cos(w_0 t) = \psi_1(t)$ and $\sqrt{\frac{2}{T_s}}\sin(w_0 t) = \psi_2(t)$, then signal $s_i(t)$ can be represented as

$$s_i(t) = a_{i1}\psi_1(t) + a_{i2}\psi_2(t), \tag{1.16}$$

one can notice that $\psi_1(t)$ and $\psi_2(t)$ signals are orthogonal to each other in a symbol period, which means it's inner product is

$$\int_0^T \psi_1(t)\psi_2(t)dt = 0, \tag{1.17}$$

and the inner product with itself for there two signals is

$$\int_0^T \psi_j(t)\psi_j(t)dt = 1, \tag{1.18}$$

while the inner product with $s_i(t)$ is

$$\int_0^T s_i(t)\psi_j(t)dt = a_{ji}. \tag{1.19}$$

As explained in section 1.5, it's necessary for transmission to use a SRRC filter before symbol transmission, which means the symbol in DVBS2 has the shape defined by the equation

$$s_i(t) = h(t).\sqrt{\frac{2E_i}{T}}\cos(\omega_0 t - \phi_i). \tag{1.20}$$

A cartesian representation of the constants a_{ji} can be made, for example, for the QPSK bit maping in the DVBS2 protocol (ETSI, 2014a, p.26):

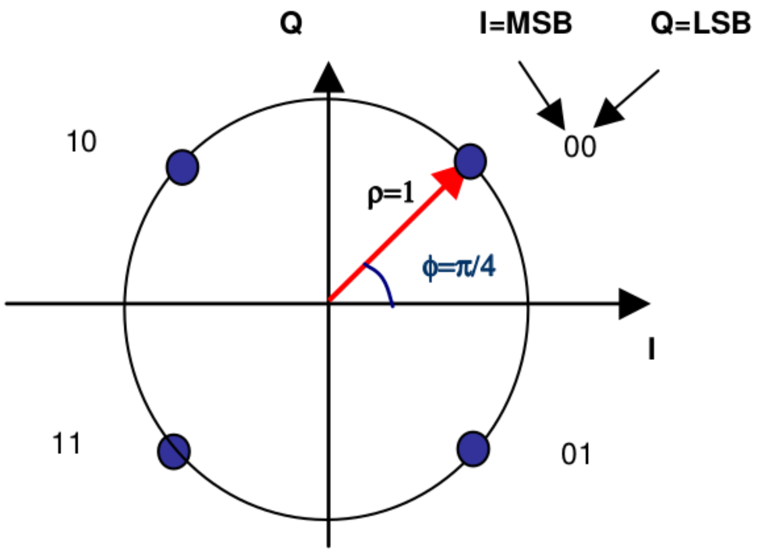


Figure 12 – DVBS2 Bit mapping in QPSK constellation. (ETSI, 2014a, p.26)

Where $\rho = \sqrt{(a_{1i}^2 + a_{2i}^2)}$ and $\phi = \arctan(\frac{a_{1i}}{a_{2i}})$, for the 8PSK constellation:

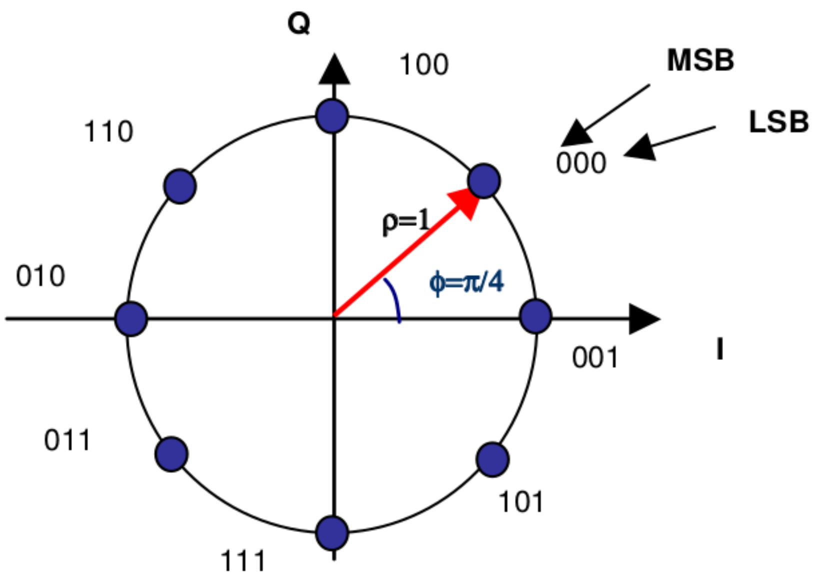


Figure 13 – DVBS2 Bit mapping in 8PSK constellation. (ETSI, 2014a, p.26)

In the 16APSK modulation the constellation is composed of two concentric rings of uniformly spaced 4 and 12 PSK, with an inner ring of radius R_1 and an outer ring of radius R_2, the ratio between the outer radius over the inner radius is defined as $\gamma = R_1/R_2$, this γ value depending on the code rate value, given at the header of the frame by a modulation and codification (MODCOD) parameter, the possible values is given in table 1, the possible γ values is given by the DVBS2 protocol and it is:

Code Rate	γ
2/3	**3.15**
3/4	**2.85**
4/5	**2.75**
5/6	**2.70**
8/9	**2.60**
9/10	**2.57**

Table 2 – Table with all possible γ values for 16APSK defined by the DVBS2 protocol.

As table 2 shows, when the code rate gets smaller, or, in other words, more correction, the γ value also gets smaller, to guarantee a optimum constellation ratio for better power efficiency. The overall constellation for the 16ASPK has the form:

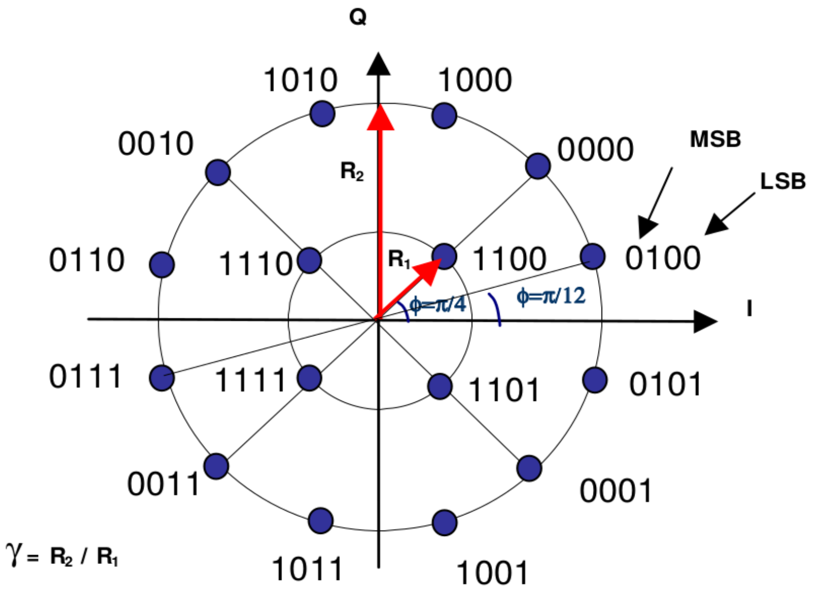


Figure 14 – DVBS2 Bit mapping in 16APSK constellation. (ETSI, 2014a, p.26)

The DVBS2X protocol gives some more bit mapping guidelines, with more constellations and code rates, this text will be focused the DVBS2 constellations also carried by the DVBS2X protocol and the constellations advised for an SNR less than 5 dB. Given this symbol representation, one option of symbol demodulation/detection is given by (SKALAR, 2017, p.108):

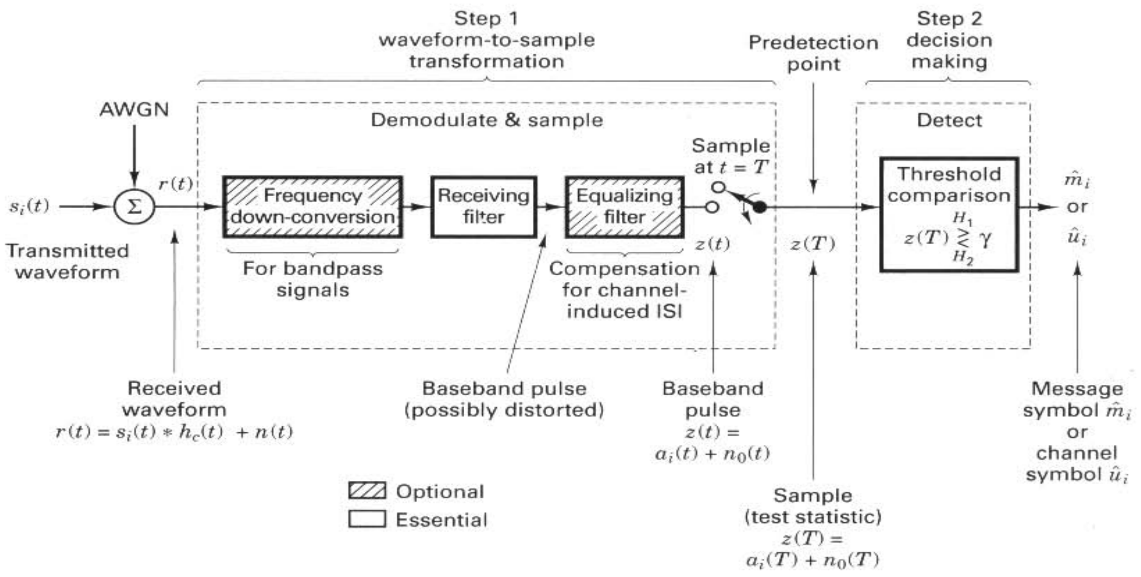

Step 1 waveform-to-sample transformation
Predetection point
Step 2 decision making
AWGN
Demodulate & sample
Detect
$s_i(t)$
Transmitted waveform
$r(t)$
Frequency down-conversion
Receiving filter
Equalizing filter
Sample at $t = T$
Threshold comparison
$z(T) \overset{H_1}{\underset{H_2}{\gtrless}} \gamma$
$\hat{m}_i$ or $\hat{u}_i$
$z(t)$
$z(T)$
For bandpass signals
Compensation for channel-induced ISI
Received waveform $r(t) = s_i(t) * h_c(t) + n(t)$
Baseband pulse (possibly distorted)
Baseband pulse $z(t) = a_i(t) + n_0(t)$
Message symbol $\hat{m}_i$ or channel symbol $\hat{u}_i$
▨ Optional
☐ Essential
Sample (test statistic) $z(T) = a_i(T) + n_0(T)$

Figure 15 – Two step demodulation system for digital communication systems. (SKALAR, 2017, p.108)

In figure 15, the frequency down-conversion can be seem as a QAM demodulation, being a multiplication with the base functions and a low-pass filter, the received filter it's what is called a matched filter, which is the filter that optimizes the signal-to-noise ratio (SNR) and its impulse response can be defined with the equation for the DVBS2X SRRC pulse as

$$h(t) = h_{srrc}(T - t) \tag{1.21}$$

and

$$z(t) = a_{ij} h_{srrc}(t) * h(t) = a_{ij} \int_0^t h_{srrc}(\tau) h_{srrc}(T - t - \tau) d\tau, \tag{1.22}$$

in equation (1.22), it's possible to analyze that the point of maximum correlation is at $t = T$, this is the ideal sampling point for the matched filter.

A code in PYTHON was made to generate all bit mapping constellations from QPSK to 16APSK and perform the matched filtering with the SRRC pulse in manner to afterward be possible to then quantize in point fixed representation and input in VHDL code, as explained in the general methodology section 2. The following figure shows an example of matched filtering for the QPSK in the DVBS2X protocol:

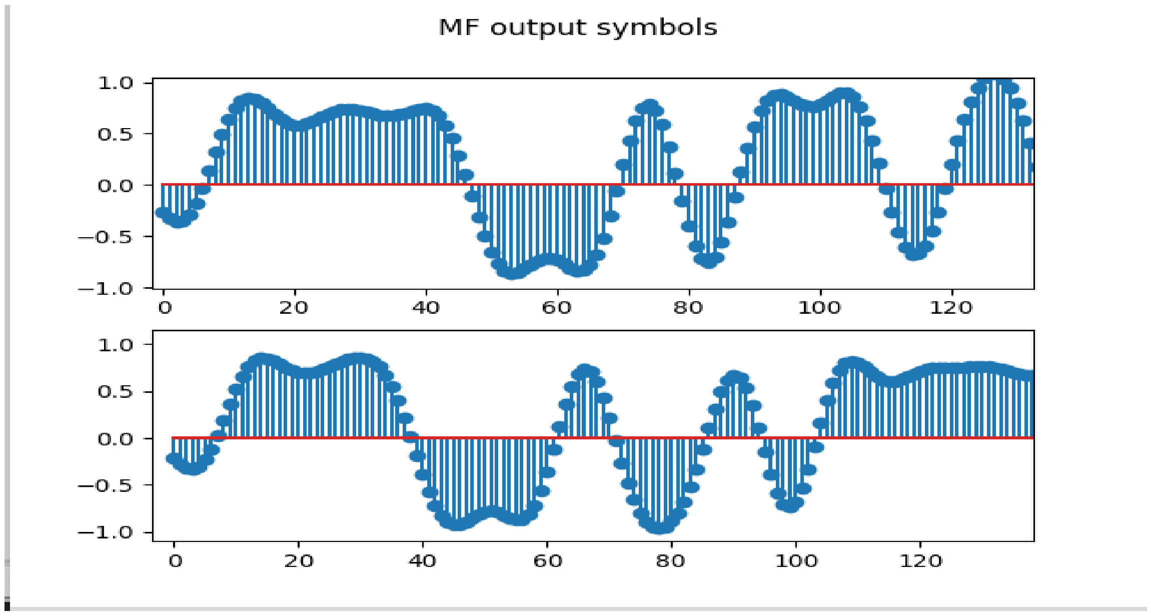
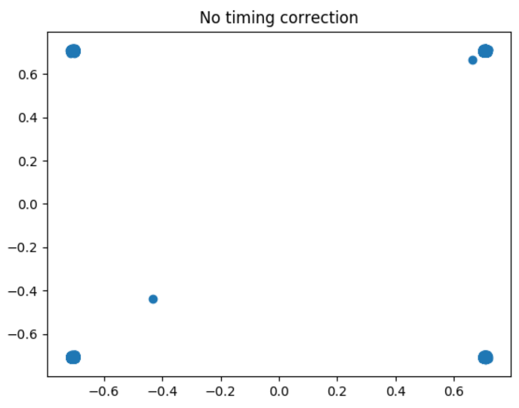


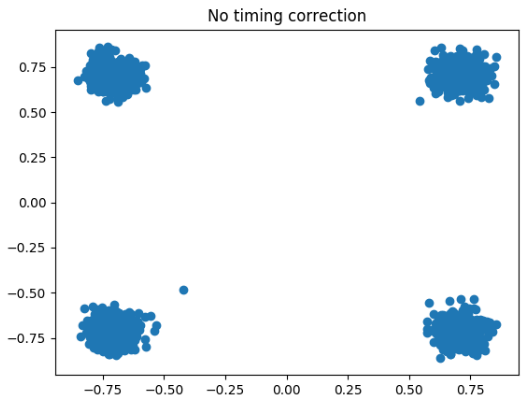
Figure 16 – I and Q representation of the matched filter output for a QPSK with SRRC pulse.

The last block in the figure 15 is the LLR, a maximum *a posteriori* detector, that will be explained in its section.

The AWGN noise explained in section 1.4, affects this demodulation system as an two dimensional Gaussian curve over the constellation points, with its variance dependent on the E_b/N_0 value, as shown in figure 17:



(a) QPSK with AWGN for $E_b/N_0 = 80dB$.



(b) QPSK with AWGN for $E_b/N_0 = 20dB$.

Figure 17 – QPSK for different AWGN signal power.

In the figure 17, it's possible to see that when the E_b/N_0 value is larger the received constellation is more close to the ideal one transmitted, when the value is small, the constellation gets more scattered around the QPSK points. Lastly is necessary to understand some fundamental FPGA resources for implementation analysis.

2 Methodology

2.1 Project Design Validation

The design flow used is a top-down functional simulation approach (SHORT, 2009, p.16), which means every algorithm implemented in this project was first evaluated in a high-level language environment. In some blocks the chosen language was PYTHON and in others MATLAB, the top-bottom methodology can be exemplified by the diagram 18:

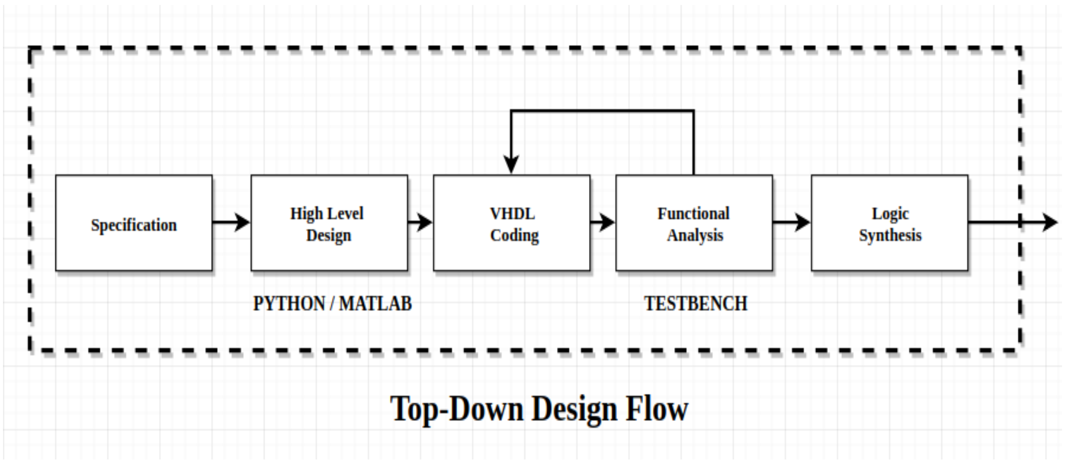


Figure 18 – Top-down design flow.

As shown in figure 18, the blocks are specified based on its theoretical functional blocks and equations, then these algorithms are simulated, after the validation of the desired algorithm then the next step is the VHDL coding, where every defined block is coded under a Doxygen coding standard, to simplify the documentation process. To maintain the synchronization and facilitate block integration every block has it's outputs/inputs based in the Xilinx AXI IP peripheral standard (XILINX., 2011) with some modifications, the diagram 19 shows the standard way the VHDL blocks in this project were coded:

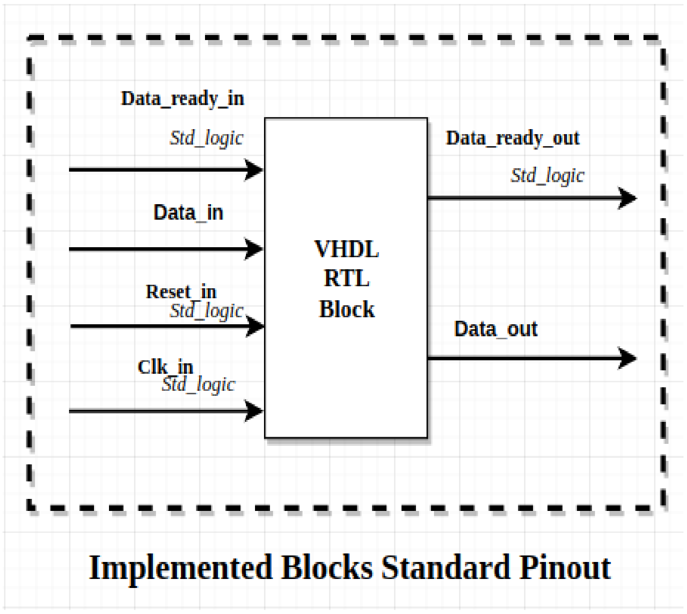


Figure 19 – Block standard pinout.

Every time there is a stable data input the block anterior to the block in question has to generate a pulse signal in data_ready_in for exactly one clock period, in other words, if a pulse of data_ready_in last for more than one period, for example, two periods, the block will interpret as if there were two different inputted data.

When there is a stable output to be read, the block will generate a pulse of data_ready_out for one clock period. The reset_in input serves to asynchronously reset every buffer and output of the block, leaving it in the default initial state.

This process can be represented by the wave-form:

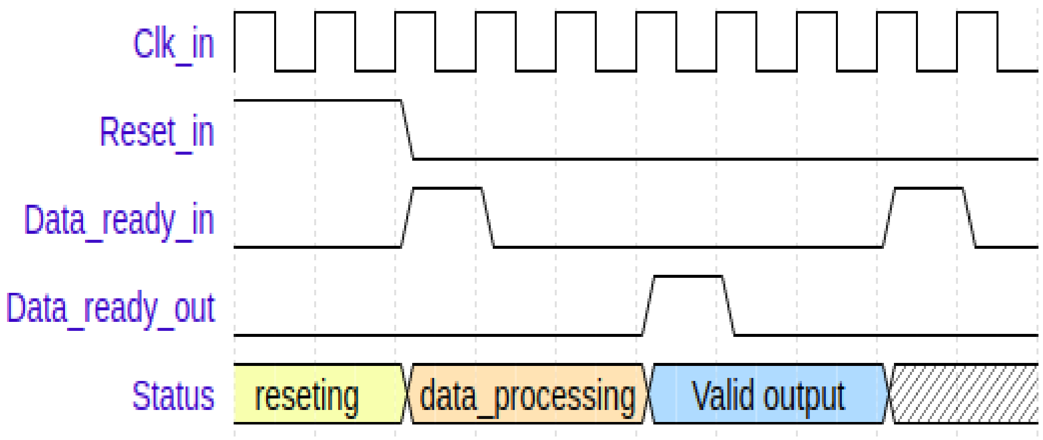


Figure 20 – Pinout typical use waveform.

The main difference from the standard showed in the figure 20 and the AXI standard is the lack of an acknowledge pin, which for the blocks in this project will not be necessary, as none of them needs to store the information in large buffers.

The general workflow for a functional simulation can be found in (SHORT, 2009):

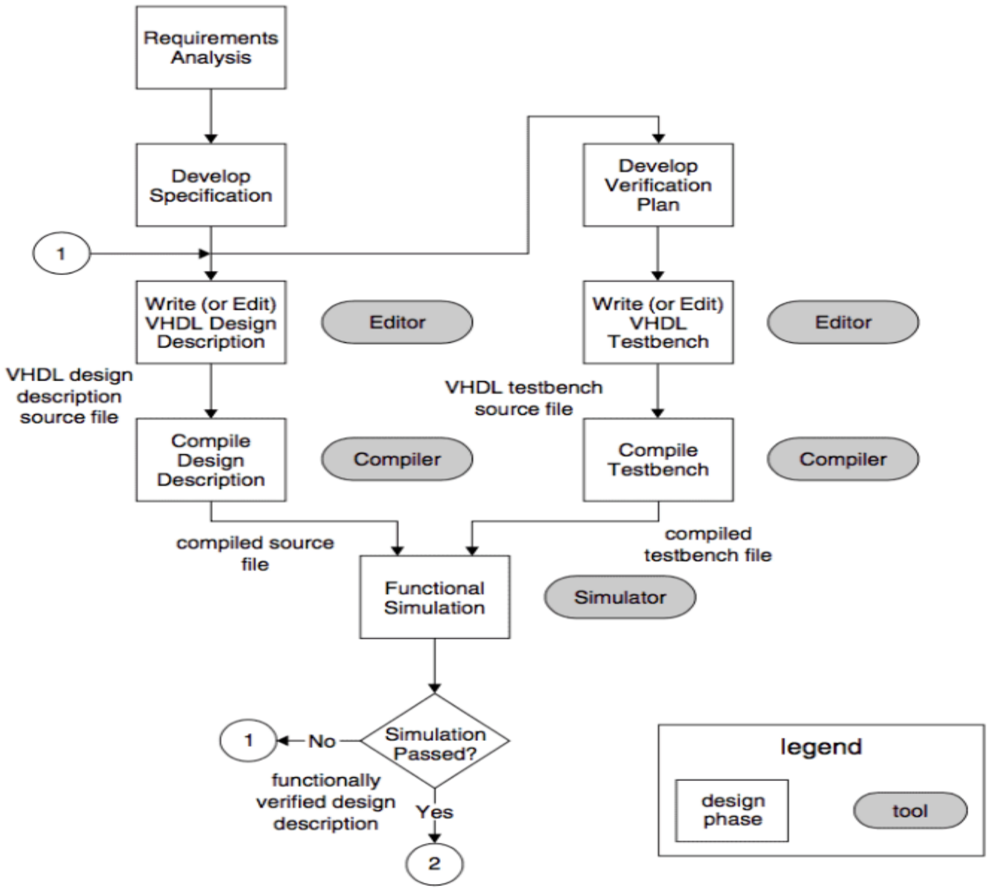

Requirements Analysis
Develop Specification
Develop Verification Plan
1
Write (or Edit) VHDL Design Description
Editor
Write (or Edit) VHDL Testbench
Editor
VHDL design description source file
VHDL testbench source file
Compile Design Description
Compiler
Compile Testbench
Compiler
compiled source file
compiled testbench file
Functional Simulation
Simulator
1
No
Simulation Passed?
functionally verified design description
Yes
2
legend
design phase
tool

Figure 21 – Functional simulation flow. (SHORT, 2009, p.17)

The objective of the simulation is to validate every functionality and guarantee that there are no faults in the design.

The blocks simulated in PYTHON and MATLAB generates a test vector .txt file input and output, in manner that the VHDL testbench is able to read the std_logic_vector entries and apply to the device under test (DUT), then the generated output is compared with the high-level simulation output by the same testbench, counting how many errors were encountered. This process is exemplified by the figure 22:

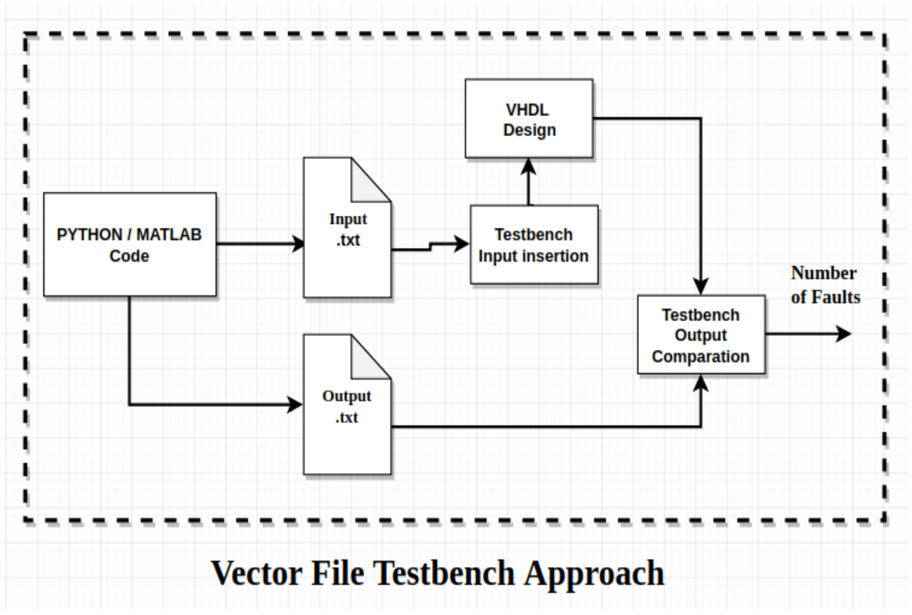

VHDL Design
PYTHON / MATLAB Code
Input .txt
Testbench Input insertion
Number of Faults
Testbench Output Comparation
Output .txt
Vector File Testbench Approach

Figure 22 – Test vector simulation process.

This methodology was implemented in all three blocks: LLR, Timing Synchronization and Symbol De-scrambling. This method guarantees more quality and less uncertainty over the system functionality.

The system fixed point was based on the AD9365 chip, which is an ADC with 12 bit, frequently implemented in SDR FPGA Modules like the ADALM Pluto, which is a ZYNQ based SDR, a System on Chip (SoC) device which integrates an ARM-based processor with the programmability of FPGA. The least significant bit was not implemented, to maximize SNR.

Figure 23 shows the fixed point representation implemented:

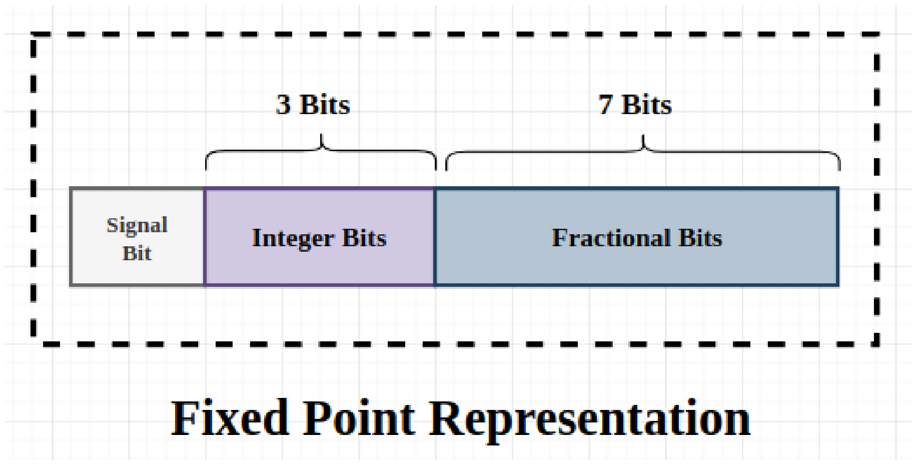


Figure 23 – 11 signed fixed point binary representation.

In the next sections each block will be explained more thoroughly, explaining it's characteristics and equations, starting with the log likelihood ratio block.

3 Log Likelihood Ratio Block

3.1 Forward Error Correction Block.

inside the DVB-S2X Forward Error Correction (FEC) block exist two different types of error correction codes, the LDPC code, and the BCH code, as it is defined by the protocol manual (ETSI, 2014a). These two codes have different characteristics within its correction method, the first being a soft decision block while the other being a hard decision one, which means that the LDPC works in an iterative way using a set of soft bits, that represents the likelihood of a certain bit being "1" or "0", differently of the BCH code, which works with a fixed operation method for the syndrome decoding, outputting a hard corrected version of the inputted message.

For the LDPC algorithm, it's necessary to input the first set of initial values for the soft bits, within the Log-Likelihood Ratio representation (LLR). This first LLR is extracted using an *à posteriori* analysis over the transmission channel, being modeled by implying the presence of an additive white gaussian noise (AWGN).

The LLR is defined as the logarithm of the ratio between the likelihood of a given bit b_j been sent as "1" and the likelihood of it been sent as "0", given a noise-corrupted received symbol. The fundamental objective it's to estimate the degradation of the sent symbol after being received in the channel. The idea is to estimated with this calculation which symbol was sent in the transmitter.

The LLR can be determined as

$$LLR(b_j) \triangleq \ln\left(\frac{\mathcal{L}(b_j = 0 \mid \mathbf{r})}{\mathcal{L}(b_j = 1 \mid \mathbf{r})}\right). \tag{3.1}$$

As a consequence of Bayes probability theorem,

$$\ln\left(\frac{\mathcal{L}(b_j = 0 \mid \mathbf{r})}{\mathcal{L}(b_j = 1 \mid \mathbf{r})}\right) = \ln\left(\frac{\mathcal{L}(\mathbf{r} \mid b_j = 0)\mathcal{L}(b_j = 0)/\mathcal{L}(\mathbf{r})}{\mathcal{L}(\mathbf{r} \mid b_j = 1)\mathcal{L}(b_j = 1)/\mathcal{L}(\mathbf{r})}\right). \tag{3.2}$$

The likelihood of a certain sent bit being "1" or "0" it's assumed as being equal in value, because there are blocks inside the DVBS2X standard such as the BB de-scrambler that makes an scrambling of the bits to ensure a more homogeneous distribution of bits in a frame.

Them the equation can be simplified in this specific case to

$$LLR(b_j) \triangleq \ln\left(\frac{\mathcal{L}(b_j = 0 \mid \mathbf{r})}{\mathcal{L}(b_j = 1 \mid \mathbf{r})}\right) = \ln\left(\frac{\mathcal{L}(\mathbf{r} \mid b_j = 0)}{\mathcal{L}(\mathbf{r} \mid b_j = 1)}\right). \tag{3.3}$$

In equation (3.3) one can interpret that the LLR depends on the proportion between the likelihood of a received symbol being in its coordinates position given its bit b_j being sent as "0" over the likelihood of the position with it being sent originally as "1".

For an AWGN channel the probability density function for receiving $\mathbf{r}$ given it was sent the symbol $\mathbf{S_i}$ in the transmitter can be described as

$$V_i = \mathcal{L}(\mathbf{r}|s = s_i) = \frac{1}{2\pi\sigma^2} e^{-|\mathbf{r}-\mathbf{S_i}|^2/2\sigma^2}. \tag{3.4}$$

Applying equation (3.4) for every bit instead of symbol in equation (3.3), the result for the LLR with quadrature waveform in a constellation can be represented as

$$LLR(b_j) = \ln\left(\frac{\sum_{i\in\{A_{b_j}=0\}} V_i}{\sum_{i\in\{A_{b_j}=1\}} V_i}\right), \tag{3.5}$$

where $A_{b_j} = x$ represents the set of symbols that have it's j-th bit equals to $x = \{0, 1\}$.

The article (OLIVATTO, 2016, p.69) provides an algorithm example for the implementation of the LLR calculation which is both computationally efficient and diminishes hardware complexity, the article supposes that the symbol deviation it's a consequence of the AWGN noise and also uses the logarithm Jacobi identity, reaching the following equation;

$$LLR(b_j) = \max\left(\bigcup D_i\right) - \max\left(\bigcup D_k\right), \tag{3.6}$$

where:

$$D_j = -\frac{|\mathbf{r} - \mathbf{S}_j|^2}{2\sigma^2}, \tag{3.7}$$

That is, D_j it's the euclidean distance squared between the reference constellation symbol coordinate and the received symbol, all that normalized by minus two times the noise power (or the Gaussian distribution variance). In equation (3.6) the indexes "i" and "k" refers to the symbols within the constellation that have in its codification the bit value in the position "j" equals to "0" in the "i" case, and "1", in the "k" index. For example, for the 8PSK modulation and "j" being the least significant bit, we have

$$LLR(b_0) = \max(D_0, D_2, D_4, D_6) - \max(D_1, D_3, D_5, D_7). \tag{3.8}$$

Equation (3.6) allows a computational implementation that is relatively simplified as opposed the use of exponentials that appear by the noise nature.

3.2 LLR block design

3.2.1 Design Considerations

As a mean to simplify the VHDL design, based on the identity

$$max\left(\bigcup(-KD_i)\right) = -Kmin\left(\bigcup D_i\right), \tag{3.9}$$

the equation (3.6) was normalized by a factor of $-\rho_{max}/2\sigma^2$, where ρ_{max} is the maximum radius of distance to a symbol to be considered, or in other words, the whole equation was normalized by a factor of D_{max}. The final equation to be designed is

$$LLR(b_j) = \frac{1}{\rho_{max}}\left[\min\left(\bigcup \rho_k\right) - \min\left(\bigcup \rho_i\right)\right]. \tag{3.10}$$

In this way the equation (3.6), that have an infinite domain and image, gets quantized to a maximum limit. The LLR of the equation (3.10) goes from -1 to 1 in the fixed point binary representation of choice, and any distance ρ bigger than ρ_{max} gets quantized to ρ_{max}.

3.2.2 General Block design

Based on the reference it was possible to conceive a functional block diagram for the LLR, before implementing in actual VHDL code:

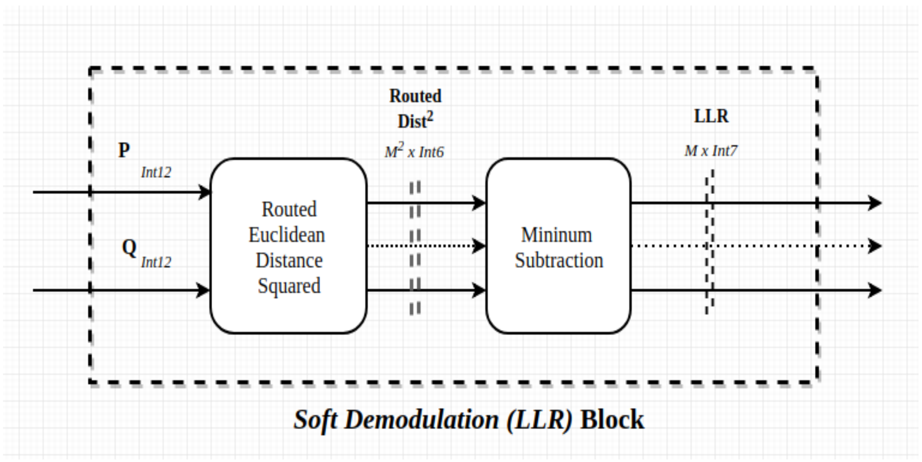

Routed
Dist²
M² x Int6
LLR
M x Int7
P
Int12
Q
Int12
Routed Euclidean Distance Squared
Mininum Subtraction
Soft Demodulation (LLR) Block

Figure 24 – Proposed LLR functional diagram block.

The fundamental working of the diagram 24 is to apply the equation (3.6) in a fixed-point representation, the system was thought of as a concatenation of two blocks: the routed euclidean squared block, that calculates the distance squared between the received symbol and all the symbols from the reference constellation them routes the signals to the next block accordingly and the minimum subtraction block, that outputs the LLR value for every bit position.

The soft Demodulator was built with a pipeline with the following stages:

- **Stage 1:** Calculates the euclidean distance between the received symbol and each of the symbols in the used constellation. if the symbol distance is bigger than a maximum distance, the outputted distance becomes the maximum distance. The RTL block was generated in VHDL:

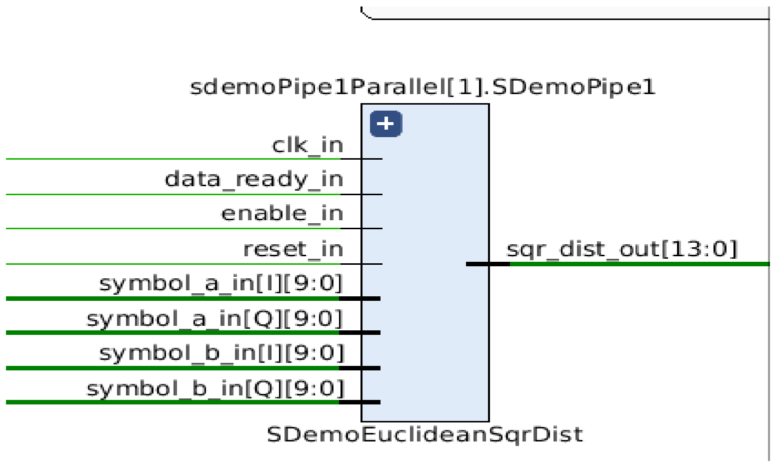


Figure 25 – RTL diagram of one of the first stage pipeline parallel components.

- **Stage 2:**

  Step 1: For each bit, determines the maximum distance between the received symbol and the symbols that have the bit equals to '1' in it's index and the ones that have the bit equal to '0'. the distances used are the values formatted in stage 3.

  Step 2: The LLR is determined through the subtraction between the maximum symbol distances that have the bit '1' in it's index and the maximum distances of the symbols that have the bit '0'.

  The RTL block was generated in VHDL:

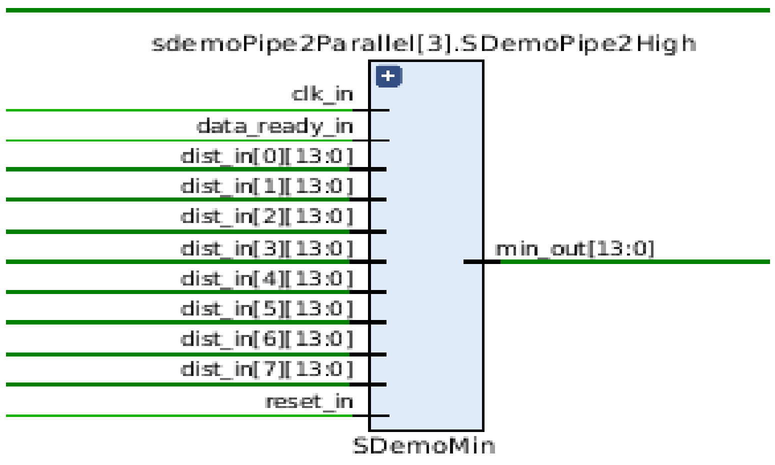


Figure 26 – RTL diagram of one of the second stage pipeline parallel components.

- **Stage 3:**

Step 1: Normalizes the distances calculated in the pipeline's stage 1, with a maximum value as a parameter, that depends of the estimated SNR of the signal. This process of normalization consists in multiplying the values for a scale factor less than 1.

Step 2: Formats the results calculated in the stage 2, in a 7 bit fixed point signed representation. Firstly, the normalized values are saturated to used more efficiently the representation proposed, with 5 bit fractional representation. Then, it's selected just the first integer bit and the first 5 more significant bits in the fractional part, the other bits are discarded.

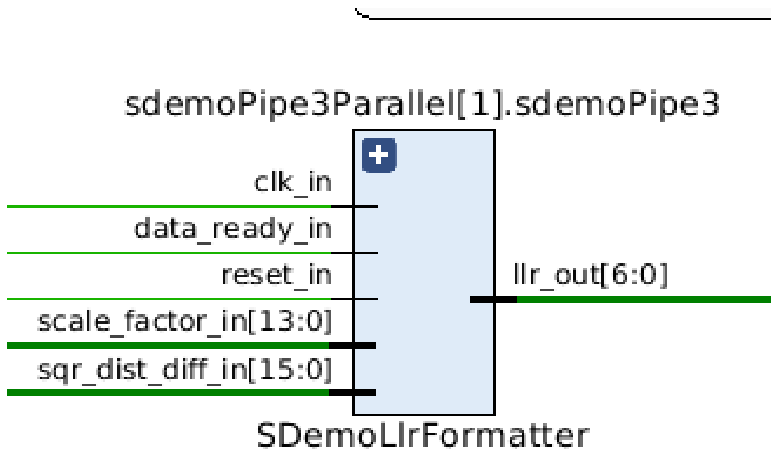


Figure 27 – RTL diagram of one of the third stage pipeline parallel components.

Also in the specific case of the DVBS2X protocol (ETSI, 2014b, p.25) for the very low SNR mode (VL-SNR), which uses a $\pi/2BPSK$, with the spreading factor $= 2$ then the FECFRAME bits are repeated twice before mapping in the constellation, with the following rules

$$\begin{aligned} I_{2i-1} &= Q_{2i-1} = (1/\sqrt{2})(1 - y_{2i-1}), \\ I_{2i} &= -Q_{2i} = -(1/\sqrt{2})(1 - y_{2i}). \end{aligned} \tag{3.11}$$

Where y represents the FECFRAME bits. Because we're sending two times the information, the LLR equation (3.3) can be writen as

$$LLR(b_j) \triangleq \ln\left(\frac{\mathcal{L}(b_j = 0|\mathbf{r}_1 \cap \mathbf{r}_2)}{\mathcal{L}(b_j = 1|\mathbf{r}_1 \cap \mathbf{r}_2)}\right). \tag{3.12}$$

$\mathbf{r}_1$ and $\mathbf{r}_2$ are the received symbols, they're Gaussian distributed, if it's assumed that they are uncorrelated, them it's possible to make an approximation.

It's possible to to use this relationship in this specific case

$$\mathcal{L}(\mathbf{r_1} \cap \mathbf{r_2}|b_j = x) = \mathcal{L}(\mathbf{r_1}|b_j = x)\mathcal{L}(\mathbf{r_2}|b_j = x), \tag{3.13}$$

and with the same reasoning used in the derivation (3.6), one can get to

$$LLR(b_j) = \ln\left(\frac{\mathcal{L}(\mathbf{r_1} \mid b_j = 0)}{\mathcal{L}(\mathbf{r_1} \mid b_j = 1)}\frac{\mathcal{L}(\mathbf{r_2} \mid b_j = 0)}{\mathcal{L}(\mathbf{r_2} \mid b_j = 1)}\right) = LLR_{\mathbf{r_1}}(b_j) + LLR_{\mathbf{r_2}}(b_j). \tag{3.14}$$

The equation (3.14) shows that the resulting LLR from the reoccurrence of the same symbol can be represented as the sum of the LLR calculated from each received symbol sequentially.

The overall result gives that the spreading factor for LLR can be represented as the sum of the previously two LLRs, this was then implemented in VHDL , with a section in the RTL for the accumulation of those values:

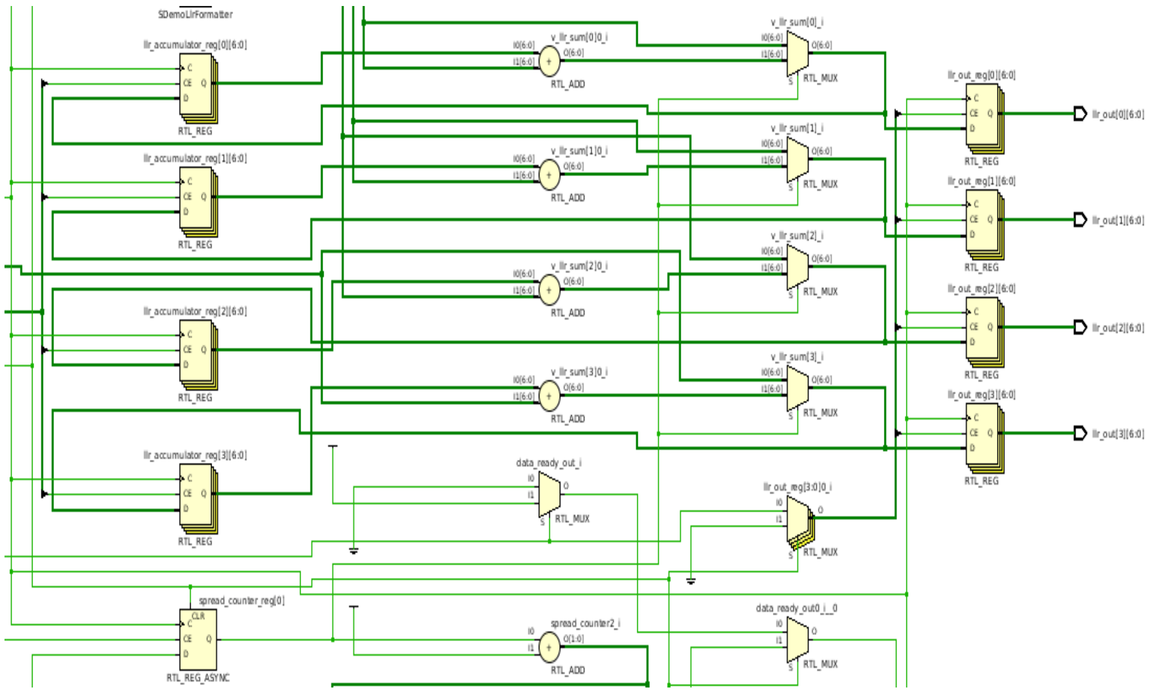


Figure 28 – RTL section for LLR accumulation in spreading factor = 2, in the LLR RTL design VIVADO interface.

One can notice in figure 28 that there's a buffer to store the last LLR value, then sum with the next received, the mux present in the RTL is controlled by the spreading factor value, if "1" then the output is simply the output without accumulation.

3.2.3 Routed euclidean distance squared block.

The functional block diagram proposed to the implementation can be seen in figure 29:

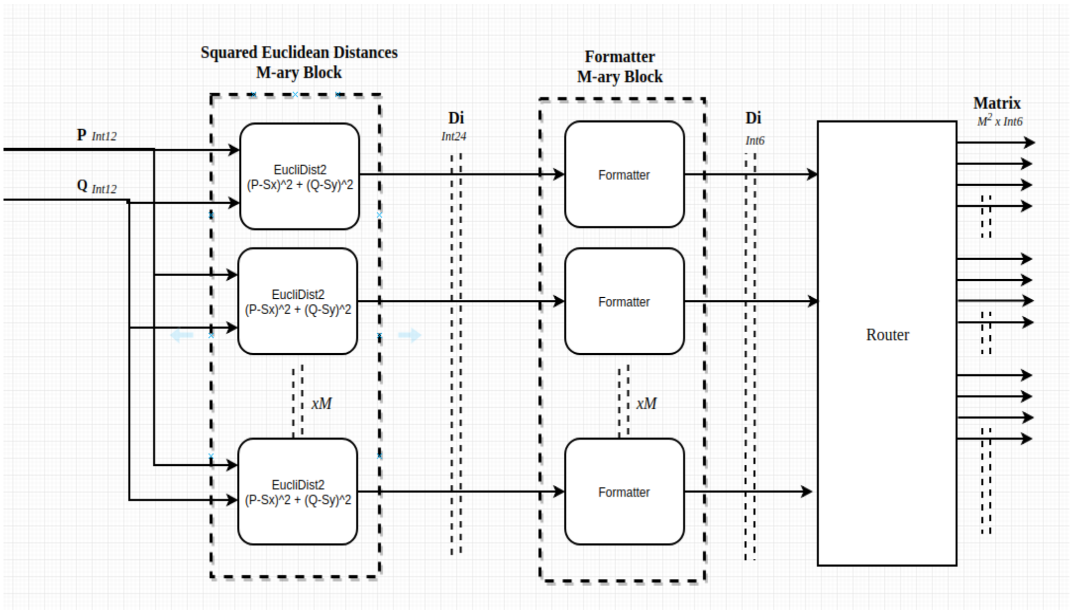


Figure 29 – Proposed routed euclidean distance squared functional diagram block.

Figure 29 shows three main blocks, the squared euclidean distances M-ary block, that receives as an input a symbol and as a parameter all possible constellation symbol coordinates, one pair for each Euclidist2 block, the Formatter M-ary block, that receives the output operation from the euclidean distance block and as a parameter the maximum distance inverse, at the end of the block there is a router, that routes the output accordingly to the rule in (3.10).

The section of the RTL design block for the first pipeline that encloses the Squared Euclidean distance block is shown in figure 30:

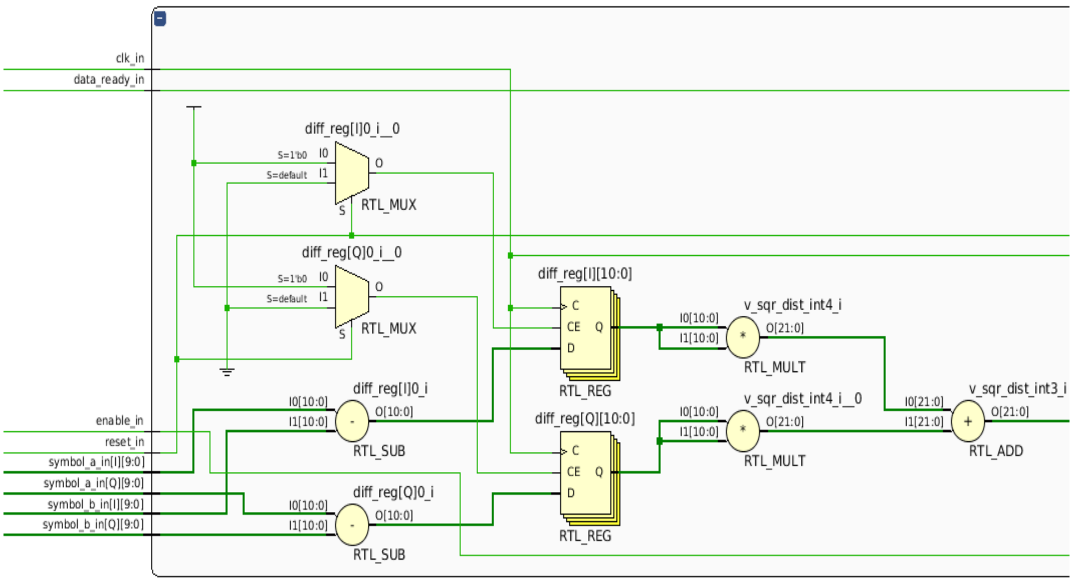


Figure 30 – RTL block for the squared euclidean distance.

One can notice in the figure 30 that in a first clock cycle it calculates the diference of the received symbol and the reference symbol, then in another cycle calculates the square of those values and sum.

In the RTL block of the first pipeline there is a part rounds the last bit before the least significant ones ones that will be discarded, which was a design decision made latter. The output has the same number of fractional bits as the original input, this part also sees if the result is bigger than the maximum defined distance, then if that's the true, the output value is the maximum distance. This block can be seem in figure 31:

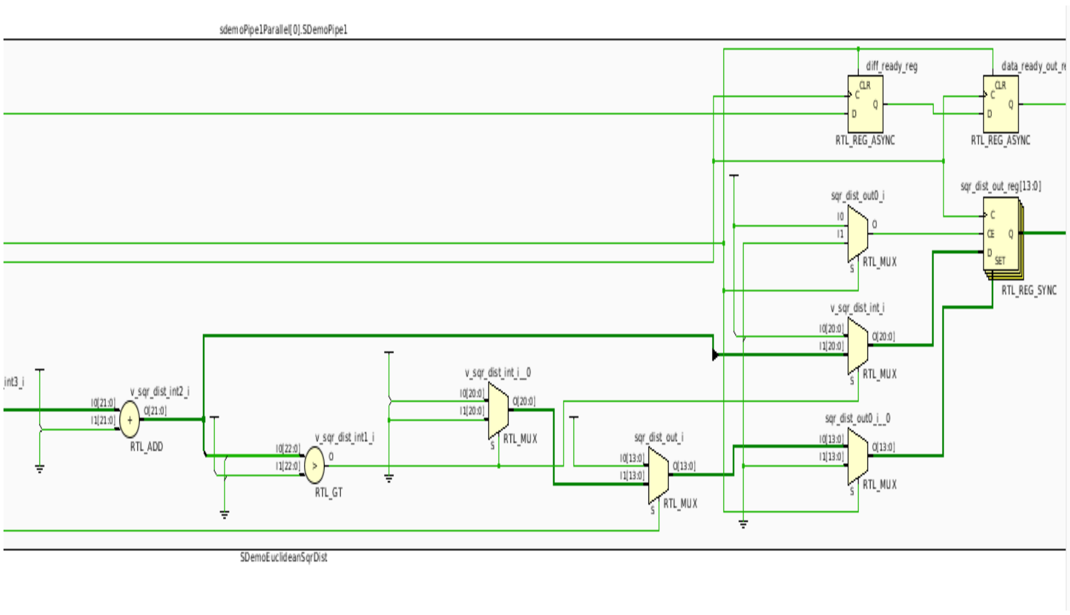
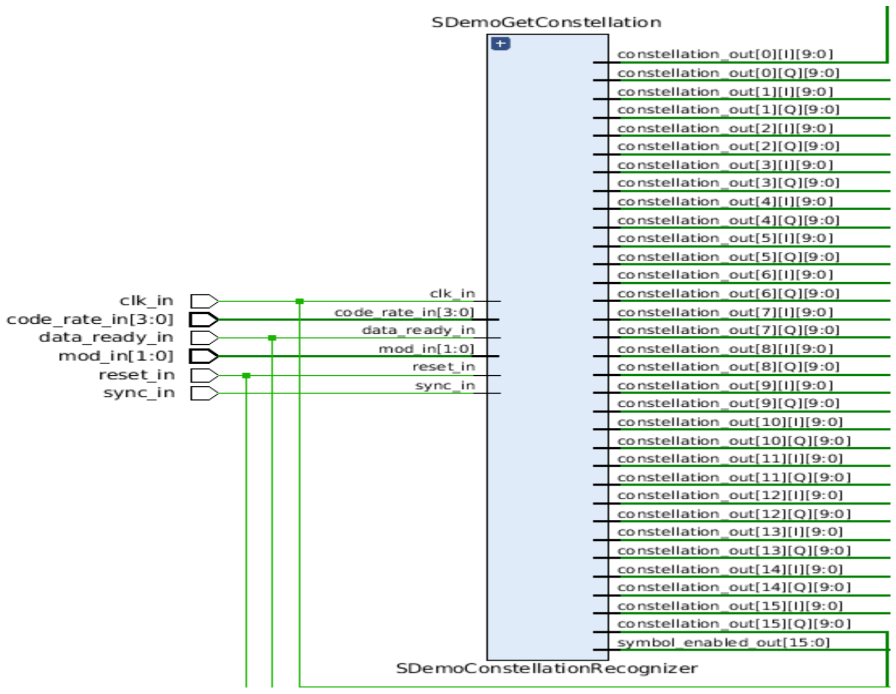


Figure 31 – RTL block that rounds and discards least significant bits in the first pipeline.

3.2.4 Constellation recognizer

The constellation recognizer block is an external block that receives the Code rate and Modulation, then outputs the symbol coordinates for all points in the inputted constellation, feeding it to the squared euclidean distances M-ary block.

The RTL design for this block made in VHDL can be seen in figure 32:



SDemoGetConstellation
clk_in
code_rate_in[3:0]
data_ready_in
mod_in[1:0]
reset_in
sync_in
constellation_out[0][I][9:0]
constellation_out[0][Q][9:0]
constellation_out[1][I][9:0]
constellation_out[1][Q][9:0]
constellation_out[2][I][9:0]
constellation_out[2][Q][9:0]
constellation_out[3][I][9:0]
constellation_out[3][Q][9:0]
constellation_out[4][I][9:0]
constellation_out[4][Q][9:0]
constellation_out[5][I][9:0]
constellation_out[5][Q][9:0]
constellation_out[6][I][9:0]
constellation_out[6][Q][9:0]
constellation_out[7][I][9:0]
constellation_out[7][Q][9:0]
constellation_out[8][I][9:0]
constellation_out[8][Q][9:0]
constellation_out[9][I][9:0]
constellation_out[9][Q][9:0]
constellation_out[10][I][9:0]
constellation_out[10][Q][9:0]
constellation_out[11][I][9:0]
constellation_out[11][Q][9:0]
constellation_out[12][I][9:0]
constellation_out[12][Q][9:0]
constellation_out[13][I][9:0]
constellation_out[13][Q][9:0]
constellation_out[14][I][9:0]
constellation_out[14][Q][9:0]
constellation_out[15][I][9:0]
constellation_out[15][Q][9:0]
symbol_enabled_out[15:0]
SDemoConstellationRecognizer

Figure 32 – Proposed routed euclidean distance squared functional diagram block.

Where Sync_in is a signal that acknowledges the block the start of each frame.

3.2.4.1 Formatter block

The formmatter block receives an 24 bit fixed point representation input of the distance, makes the normalization described in equation (3.10) by multiplying with the inverse maximum distance and converts the 48 bit resulting binary word in a equivalent 7 bit representation. This process is exemplified in figure 33:

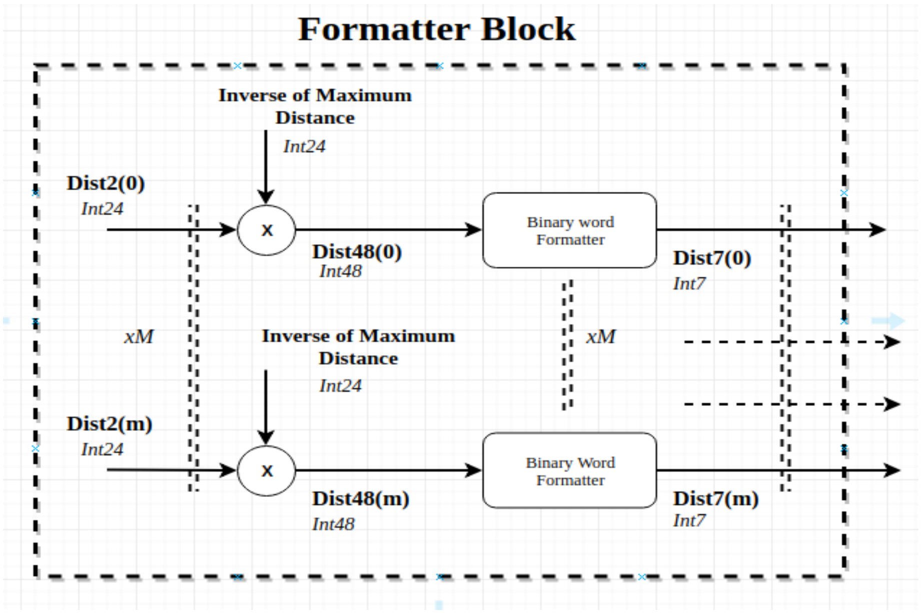


Figure 33 – Proposed Formatter functional diagram block.

In the figure 33 we can see that the number of outputs is the same as inputs, that equals the number of constellation points for a given modulation. The RTL design for this block can be seen in the third step of the pipeline defined, as shown in figure 34:

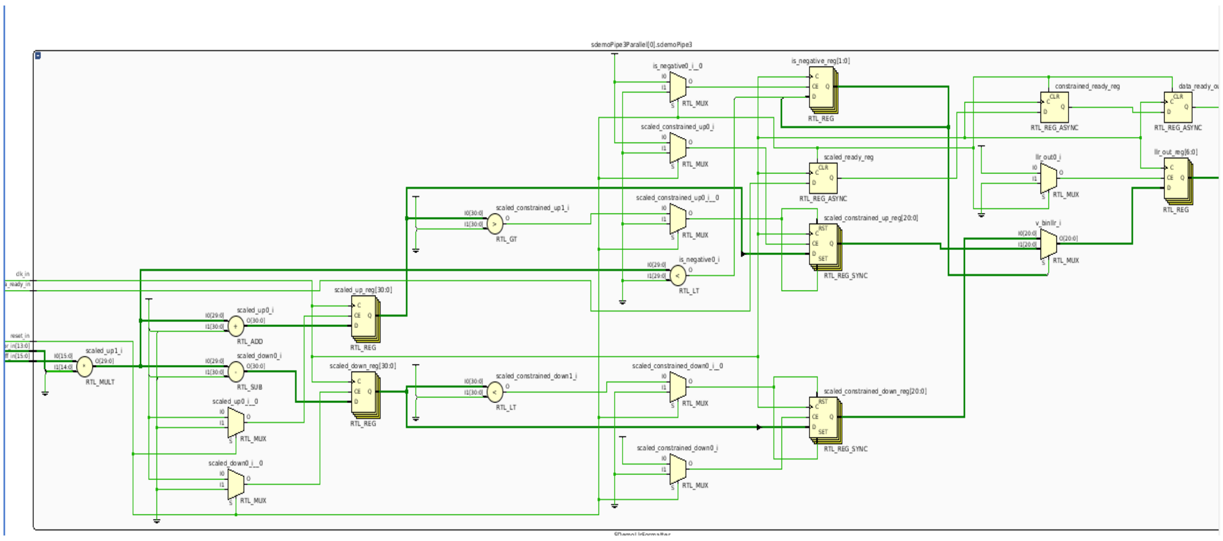


Figure 34 – RTL block design one component of the formatter block.

The system multiplies by the scale factor in the entry, then the system checks if its result is positive or negative so it can round and saturate accordingly with the upper and lower bounds of maximum and minimum LLR in the representation choice, the LLR is an infinite representation function, that's why in a finite fixed point application it's necessary to define a range of values.

3.2.5 Minimum subtraction block

As exemplified in (3.8), for a given bit, each min block receives half the number of distance inputs, where this distribution is defined by it's constellation mapping. The min distances are then subtracted and the LLR is outputted, as showed in figure 35:

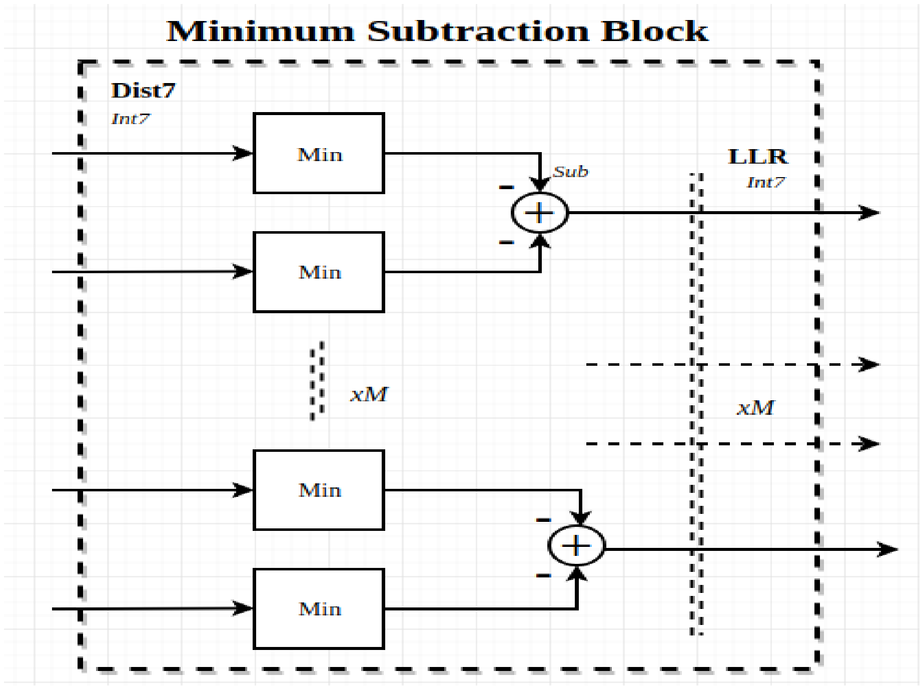


Figure 35 – Proposed minimum subtraction functional diagram block.

There is a total of $Log_2(M)$ LLR outputs, one for each bit position in the constellation mapping. Where the LLR output is a 7 bit fixed point signed representation.

The minimum block is implemented comparing the inputs two to two, in a "tournament" structure, exemplified by the diagram 36:

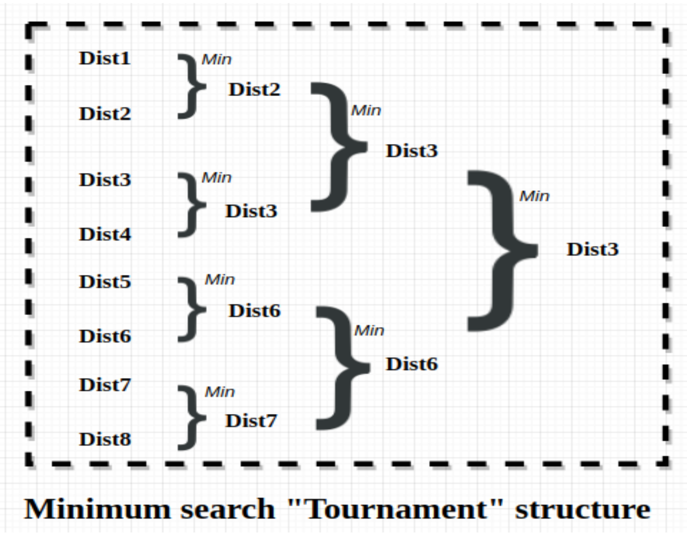


Figure 36 – Operation to find global minimum between a set of distance values.

The RTL block given by the VHDL code for the minimum block was implemented as in figure 37:

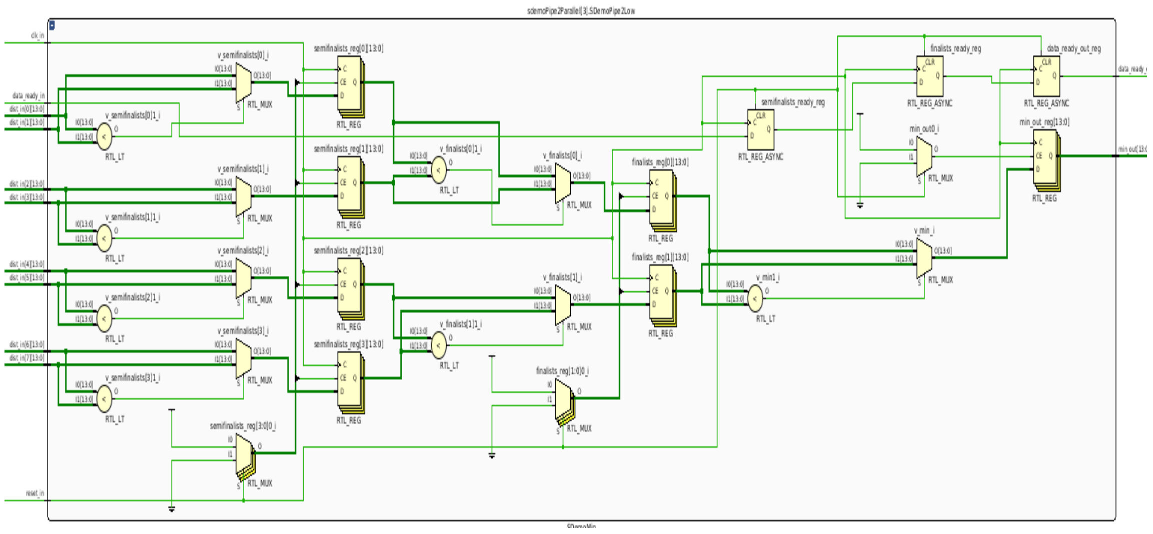


Figure 37 – RTL block for the second pipeline in LLR block.

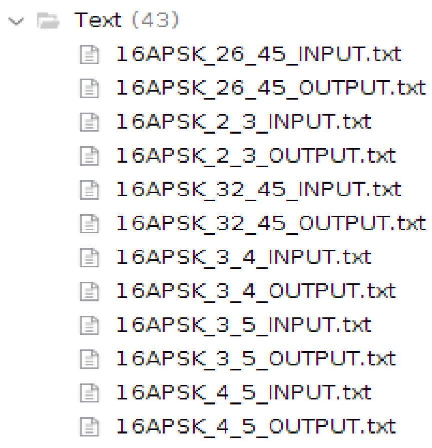
It's possible to see in figure 37 the structure explained in figure 36, where for each clock operation, the values are compared two to two and passes to the next set of buffers, for a 16APSK modulation this block takes 3 clock cycles to find the minimum value. Also the block is reusable for the other modulations with smaller orders, inputting the max distance in the non-used by the others modulations inputs.

3.3 Block Validation Methodology

All algorithms for the LLR block were firstly validated in MATLAB, then eventually the design was implemented following the directives from the previous sections, them for the LLR block testbench the modulations QPSK, 8PSK, 16APSK and $\pi/2BPSK$ with spreading factor 2 were tested. Because 16APSK constellations change when the code rate changes, each code was counted as different constellations, the following cases were tested: 16APSK: 9/10, 8/9, 8/15, 7/15, 5/6, 4/5, 3/5, 3/4, 32/45, 2/3 and 26/45. This are the modulations/code rate recommended in the DVBS2 and DVBS2X for a E_s/N_0 less than 5 dB.

For every one of the the cases cited previously the MATLAB algorithm generated a text vector with the INPUT in fixed point binary signed representation and the expected OUTPUT vector file previously decoded by the high level language, in a 7 bit signed fixed point representation.

In total 43 text vector files were generated, updated to the VIVADO project interface, as shown in figure 38:



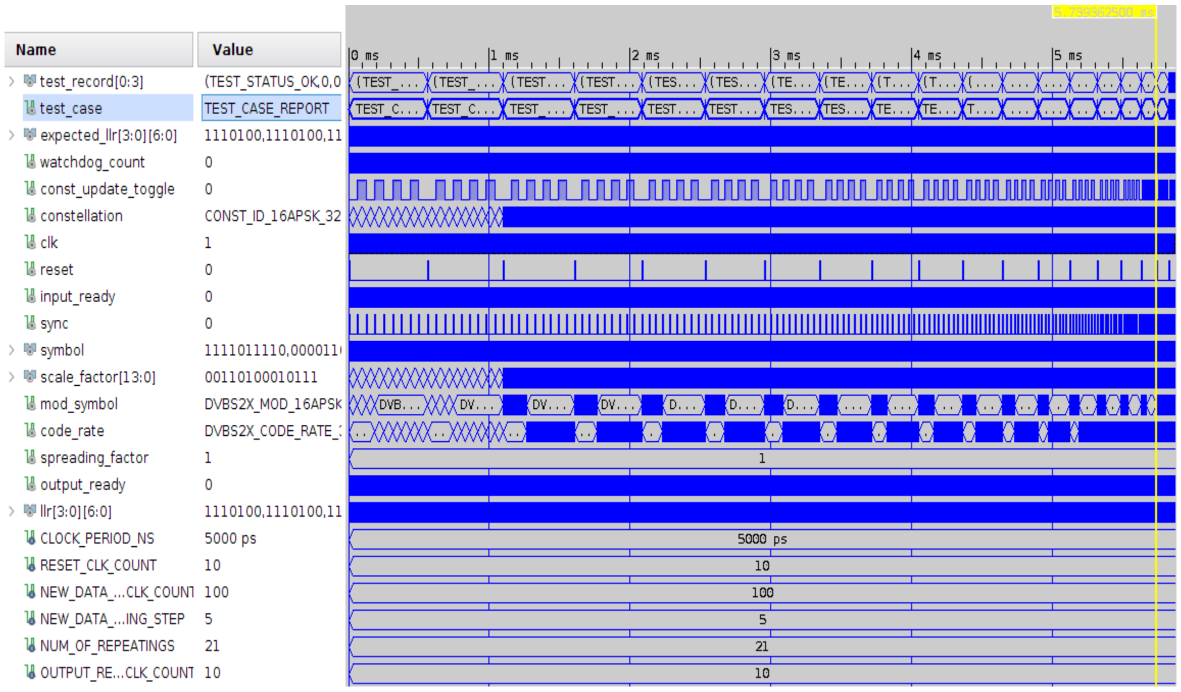
Text (43)
16APSK_26_45_INPUT.txt
16APSK_26_45_OUTPUT.txt
16APSK_2_3_INPUT.txt
16APSK_2_3_OUTPUT.txt
16APSK_32_45_INPUT.txt
16APSK_32_45_OUTPUT.txt
16APSK_3_4_INPUT.txt
16APSK_3_4_OUTPUT.txt
16APSK_3_5_INPUT.txt
16APSK_3_5_OUTPUT.txt
16APSK_4_5_INPUT.txt
16APSK_4_5_OUTPUT.txt

Figure 38 – Vector text files in VIVADO project.

The DUT is aproved if it passed every testcase without any faults. Where there is an error counter observing when the output response don't match with the vector file one.

3.3.1 Block testbench results

After every block as designed and coded in VHDL, the testbench simulation was made, and gave the results in figure 39:

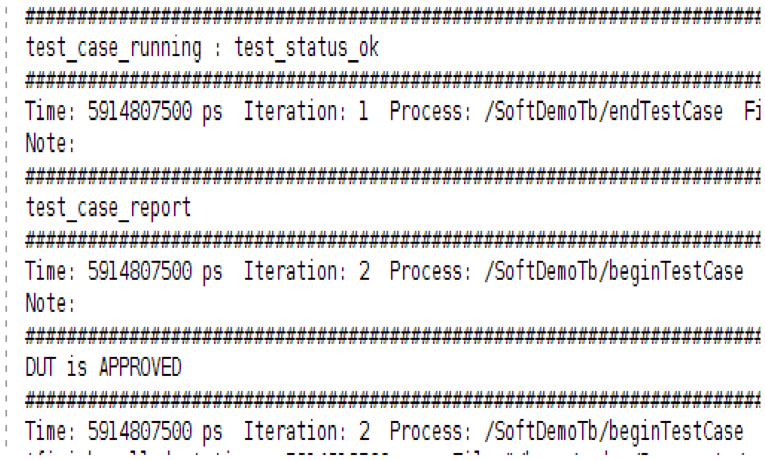


Name	Value
test_record[0:3]	(TEST_STATUS_OK,0,0
test_case	TEST_CASE_REPORT
expected_llr[3:0][6:0]	1110100,1110100,11
watchdog_count	0
const_update_toggle	0
constellation	CONST_ID_16APSK_32
clk	1
reset	0
input_ready	0
sync	0
symbol	1111011110,000011
scale_factor[13:0]	00110100010111
mod_symbol	DVBS2X_MOD_16APSK
code_rate	DVBS2X_CODE_RATE_
spreading_factor	1
output_ready	0
llr[3:0][6:0]	1110100,1110100,11
CLOCK_PERIOD_NS	5000 ps
RESET_CLK_COUNT	10
NEW_DATA_...CLK_COUNT	100
NEW_DATA_...ING_STEP	5
NUM_OF_REPEATINGS	21
OUTPUT_RE...CLK_COUNT	10

Figure 39 – VIVADO LLR block testbench simulation.

The figure 39 shows every test case being tested, highlighted in blue at the high left corner, being tested with different scale factors, MODCOD values and changing constellations dynamically.

The final result output in the TCL terminal is shown in figure 40:

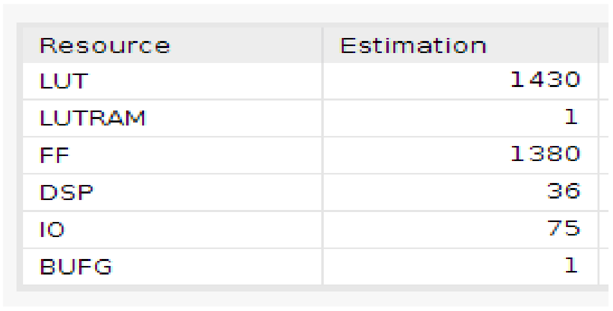


```
############################################################################
test_case_running : test_status_ok
############################################################################
Time: 5914807500 ps  Iteration: 1  Process: /SoftDemoTb/endTestCase  Fi
Note:
############################################################################
test_case_report
############################################################################
Time: 5914807500 ps  Iteration: 2  Process: /SoftDemoTb/beginTestCase
Note:
############################################################################
DUT is APPROVED
############################################################################
Time: 5914807500 ps  Iteration: 2  Process: /SoftDemoTb/beginTestCase
```

Figure 40 – VIVADO terminal DUT approval report.

Which means every test case passed without any fault, validating the system design.

Figure 41 shows the resources used for the synthesis design estimated by the VIVADO:



Resource	Estimation
LUT	1430
LUTRAM	1
FF	1380
DSP	36
IO	75
BUFG	1

Figure 41 – Post-synthesis resource utilization estimation.

Being the most complex in hardware implementation from the other two blocks, the LLR block resources necessity rises with the constellation order, but every higher order implementation can couple easily and dynamically change to smaller orders constellation, for example a LLR implemented to support 16APSK modulation also supports 8PSK, QPSK, BPSK, etc, but not 32APSK.

4 Timing Recovery Block

4.1 *Timing Recovery* Block.

Accordingly to (RICE, 2009, p.434): "Conceptually, symbol timing synchronization is the process of estimating a clock signal that is aligned in both phase and frequency with the clock used to generate the data at the transmitter. Because it is not efficient to allocate spectrum to transmit a separate clock signal from the transmitter to the receiver for the purposes of timing synchronization, the data clock must be extracted from the noisy received waveforms that carry the data. For matched filter detectors, the clock signal is used to identify the instants when the matched filter output should be sampled.". This sample text implies the main objective of correcting the sampling time signal, which is to closely match the receptor with the transmitter generated one, in terms of eye diagram, that means sampling in the maximum average eye opening moment, any timing error can be thought as any variation of this behavior.

The basic problem formulation implies the existence of a transmitted signal, modified by both an Additive White Gaussian Noise (AWGN) and a delay, the received signal can be thought mathematically as (RICE, 2009, p.436)

$$r(t) = G_a \sum_k a(k)p(t - kT_s - \tau) + \omega(t), \tag{4.1}$$

where $p(t)$ is the unit energy pulse, T_s is the sampling time, τ is the unknown timing delay, G_a is the gain of the transmitter to receiver medium, ω is the AWGN noise and a(k) is the QAM constellation coordinate representation or the k-th PAM symbol. The right sampling time T_i or interpolated period can be represented in two parts, one that represents a multiple of the the actual sampling time of the A/D interface input (M_k) and other that represents a fractional space between two samples (μ_k), this last one implies the necessity to interpolate in the image 43, to generate an estimation of the symbol wave between two adjacent samples.

The block diagram 42 represents this system:

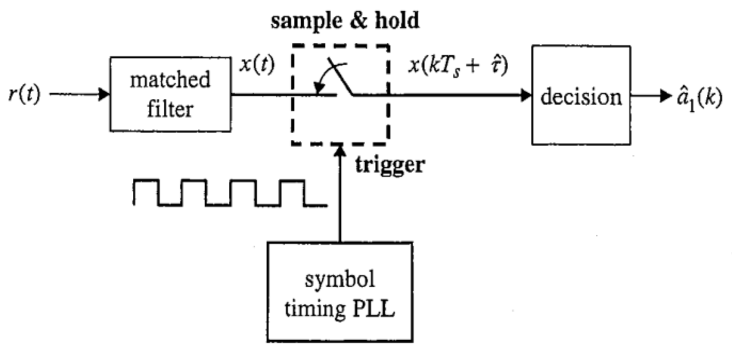


Figure 42 – Block Diagram of basic PAM detector. (RICE, 2009)

Within this scheme there are many different possible implementation architectures, one in especial can be spotted frequently in the literature, which is the Closed-Loop STR (Symbol Timing Recovery) using the NDA(Non Data-Aided) Gardner TED (Timing Error Detection). It is cited in the main reference textbook (RICE, 2009), in (SAVVOPOULOS; ANTONAKOPOULOS, 2009) and in (GARDNER, 1986). Inspired by those reference implementations, was possible to define the diagram 43:

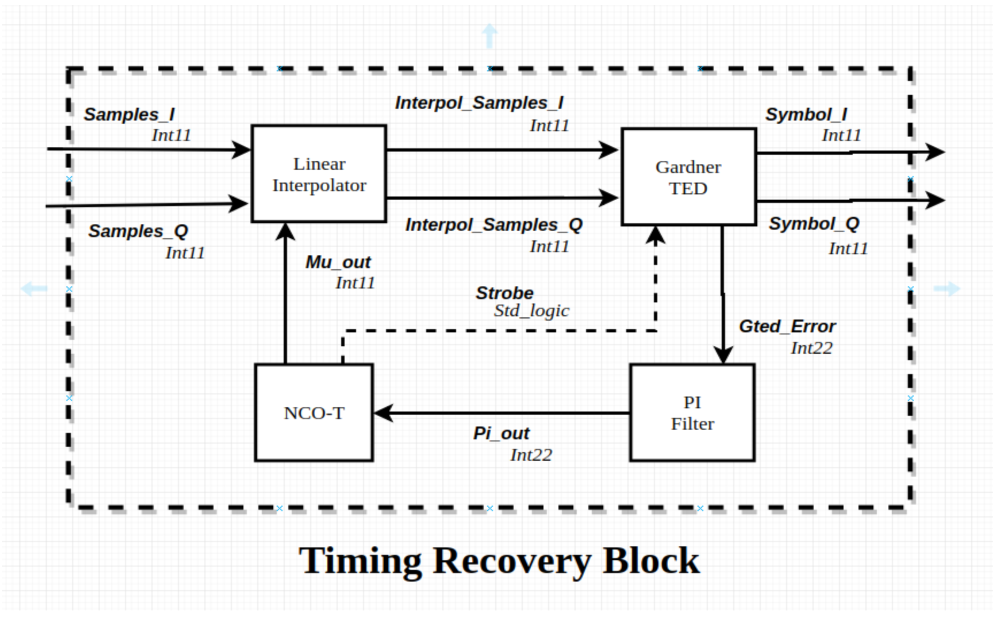


Figure 43 – Proposed timing recovery implementation diagram block.

There are four fundamental operational blocks in this system: Linear Interpolator, Gardner TED (Timing Error Detector), PI (Proportional-integrator) filter and NCO-T (Numerically Control Oscillator), that are further explained in the next subsections.

4.1.1 Linear Interpolator

An interpolator is basically an spline approximation of a continuous wave $x(t)$ with a polynomial of defined order, in the form

$$x(t) \approx c_p t^p + c_{p-1} t^{p-1} + c_{p-2} t^{p-2} + \ \dots + c_1 t + c_0, \tag{4.2}$$

as cited in (GARDNER, 1993), one interpolating architecture that can be reasonable implemented, feasible both by it's performance and hardware complexity, is a Farrow linear interpolating structure, which computes the operation

$$x(kT_i) = x((m(k) + \mu(k))T_s) = \mu(k)x((m(k) + 1)T_s) + (1 - \mu(k))x(m(k)T_s), \tag{4.3}$$

(4.3) is a linear approximation between two discrete adjacent samples, where $\mu(k)$ is received as an external signal, as shown in figure 44:

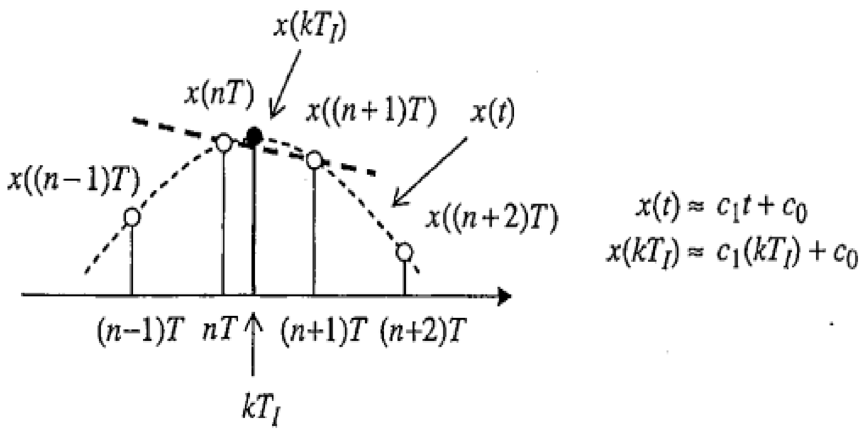


Figure 44 – Example of linear interpolation between two samples. (RICE, 2009, p. 466).

Equation (4.3) can be thought as an FIR(finite impulse response) filter with one tap, the article (GARDNER, 1993) explains that using a farrow structure filter, we can decrease the hardware and operational complexity. With this reference it's possible to define the block diagram 45:

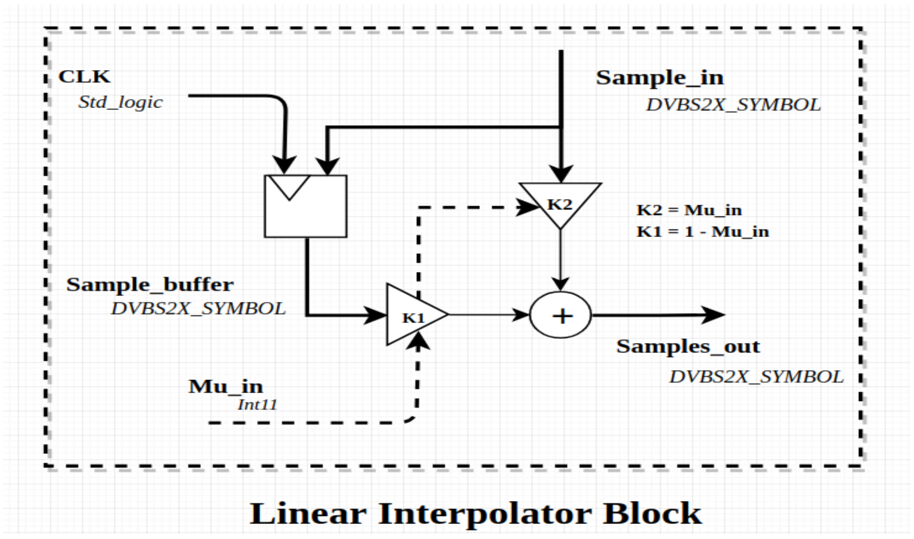


Figure 45 – Linear farrow structure block diagram.

The logic and structure of 45 was later implemented in VHDL language, as we can see in figure 46 the RTL design generated by the software VIVADO:

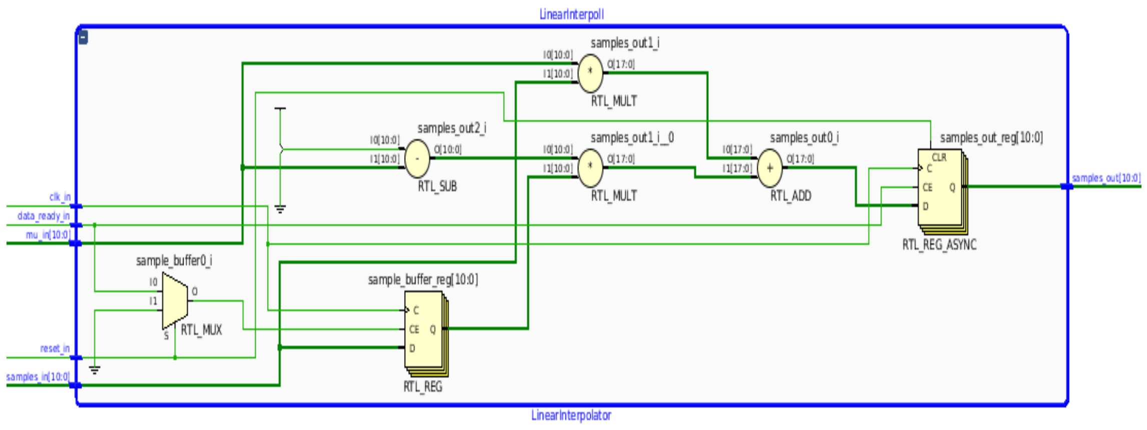


Figure 46 – VHDL generated RTL block.

As one can observe in 46, the overall structure is the same as 45, plus the reset_in, clk_in and data_ready_in signals, that are defined within most blocks in this project to maintain timing. Also can be noted that the fixed point representation of the "one" is given as a constant to the block and appear in equation (4.3) and maps to the subtraction showed in the 46 RTL block.

4.1.2 Gardner Timing Error Detection

In terms of timing error detection (TED), there are four algorithms that are most frequently cited in the literature: Zero crossing timing error detection (ZCTED), Gardner TED (GTED), Mueller and Miller TED and early late TED (ELTED). The GTED is an example of a non data-aided timing error detector, which means that it only uses the shape of the received wave to defined it's output, while the data-aided ones uses the constellation information to calculate the output.

Based on what's implemented in the article (SCIAGURA et al., 2007), it shows, comparing different STR algorithms in the same context in FPGA, that GTED is a both feasible and reasonable efficient TED algorithm within the SDR FPGA paradigm.

The equation that defines the GTED output for a quadrature signal (implying a imaginary and real inputs) is shown in (SAVVOPOULOS; ANTONAKOPOULOS, 2009) and is given by

$$e_k = y_i\,(kT - T/2)\,\{y_i[(k-1)T] - y_i(kT)\} + y_q\,(kT - T/2)\,\{y_q[(k-1)T] - y_q(kT)\}, \quad (4.4)$$

and in this equation we can understand that GTED measures the error as being the difference of two adjacent symbols samples multiplied by the sample in the middle. In fact, GTED only needs previous samples to calculate its error. In the case of quadrature wave forms being received, we just sum the contribution of both parts, both I and Q waveforms.

Also based in (SCIAGURA et al., 2007) was possible to define a logic block to the GTED implementation:

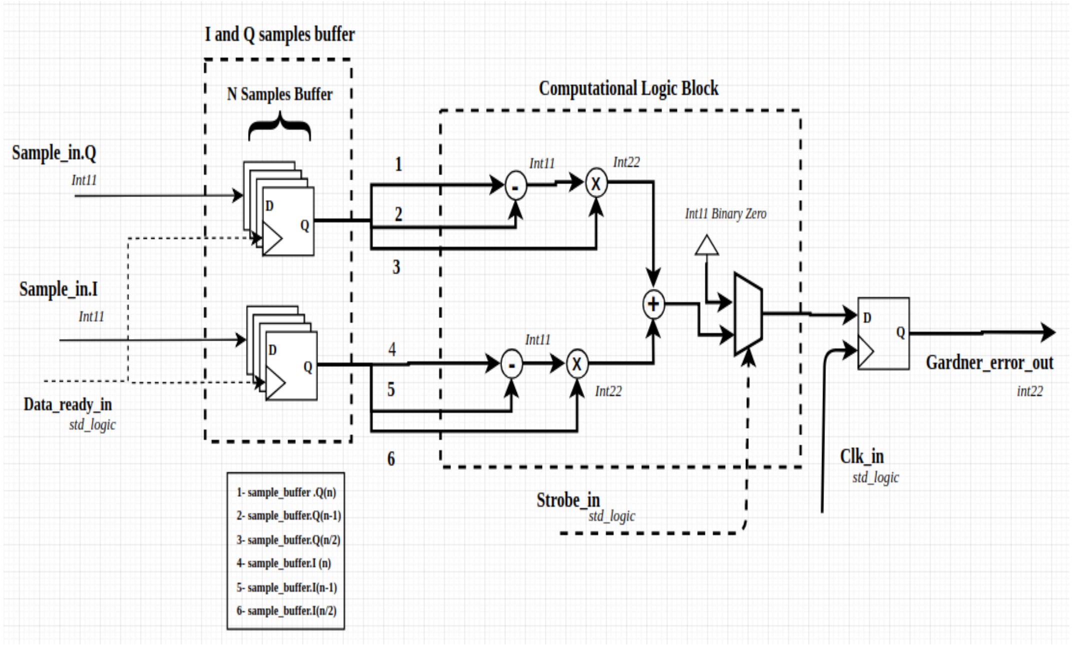


Figure 47 – Gardner Algorithm TED block diagram.

Based on block diagram 47 was possible to implement the following RTL diagram,separated by sample_buffer block and combinational logic block, using VHDL, while the number of samples per symbol was implemented as an generic to the block, the figure 48 is showing the case for a four samples per symbol sampling ratio:

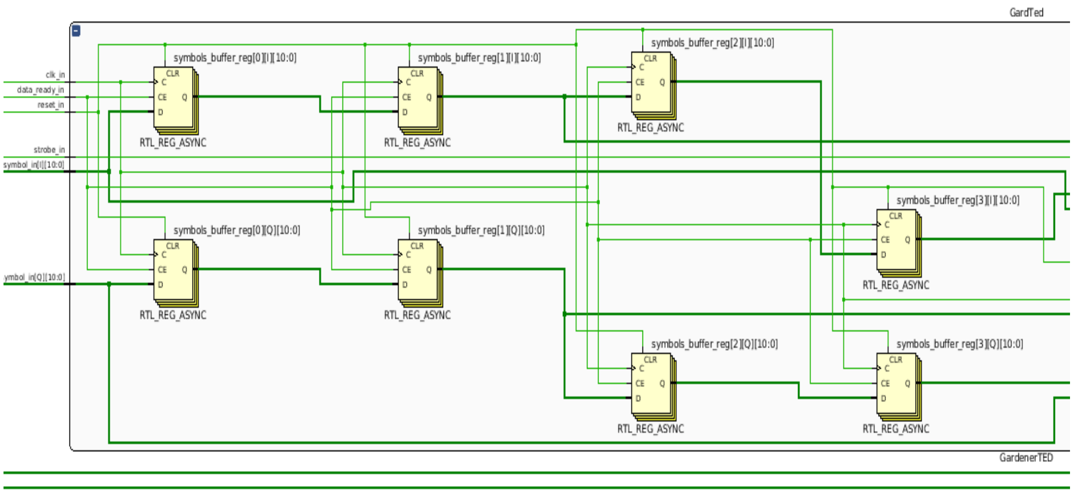

GardTed
clk_in
data_ready_in
reset_in
strobe_in
symbol_in[I][10:0]
ymbol_in[Q][10:0]
symbols_buffer_reg[0][I][10:0]
symbols_buffer_reg[1][I][10:0]
symbols_buffer_reg[2][I][10:0]
symbols_buffer_reg[3][I][10:0]
symbols_buffer_reg[0][Q][10:0]
symbols_buffer_reg[1][Q][10:0]
symbols_buffer_reg[2][Q][10:0]
symbols_buffer_reg[3][Q][10:0]
RTL_REG_ASYNC
GardenerTED

Figure 48 – Sample buffer VHDL RTL diagram.

The combinational block that applies the equation (4.4) is invariant to the number of samples per symbol and it's design was implemented as in figure 49:

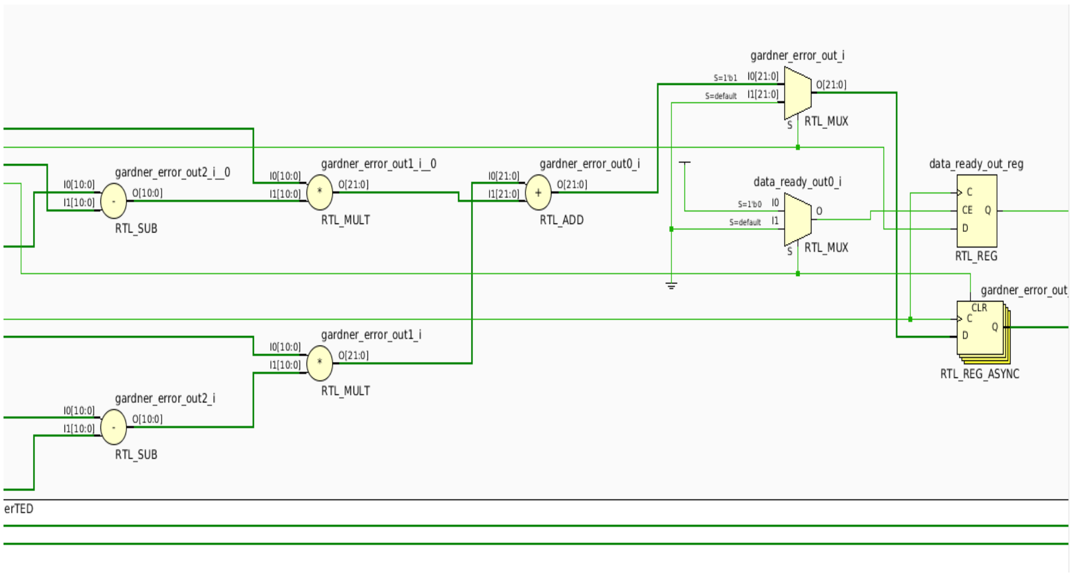


Figure 49 – Combinational logic VHDL RTL diagram.

4.1.3 PI Filter

Based in the implementation of (SAVVOPOULOS; ANTONAKOPOULOS, 2009), where a first order proportional-integrator is chosen, this filter modelling was obtained with more detail in (RICE, 2009, p.718), based in this text the block diagram in figure 50 was proposed:

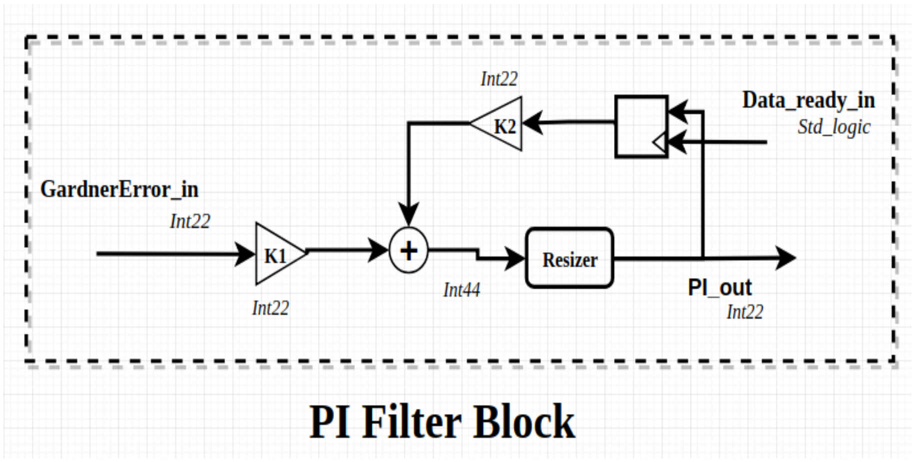


Figure 50 – PI filter functional block diagram.

In this system the proportional part is represented by the K_1 buffer, the integrator part is represented by the closed loop with a flip-flop digital buffer.

The proportional part is where the system multiplies the input by the K_1. The integrator part is where the previous value of PI output is stored and multiplied by the K_2 constant.

In figure 50 the resizer block acts as a manner to keep the overall output to a predetermined fixed point size, that happens because the sum and multiplication with the filter constants K_1 and K_2 alters the representation size.

The value for K_1 and K_2 can be found in (RICE, 2009, p.738), with the equations

$$\begin{aligned} K_0K_pK_1 &= \frac{4\zeta\theta_n}{1+2\zeta\theta_n+\theta_n^2} \\ K_0K_pK_2 &= \frac{4\theta^2}{1+2\zeta\theta_n+\theta_n^2}, \end{aligned} \tag{4.5}$$

Where

$$\theta_n = \frac{B_nT_s}{N\left(\zeta+\frac{1}{\zeta}\right)}. \tag{4.6}$$

B_nT_s is a system parameter that represents the noise bandwidth times the sampling period, N is the number of samples per symbol, ζ is the loop damping factor and K_0 and K_p are respectively the NCO gain (which in our case is -1) and the GTED gain, which can be calculated by the equation

$$K_p = \frac{4K^2E_{avg}}{T_s}\frac{1}{4\pi\left(1-\frac{\beta^2}{4}\right)}\sin\left(\frac{\pi\beta}{2}\right) \tag{4.7}$$

The values K_p , K_1 and K_2 are previously calculated in PYTHON then converted to fixed point signed representation to be inputted in the system code as constants. The resulting VHDL RTL block implemented based on this modelling is represented by the figure 51 taken from the VIVADO interface:

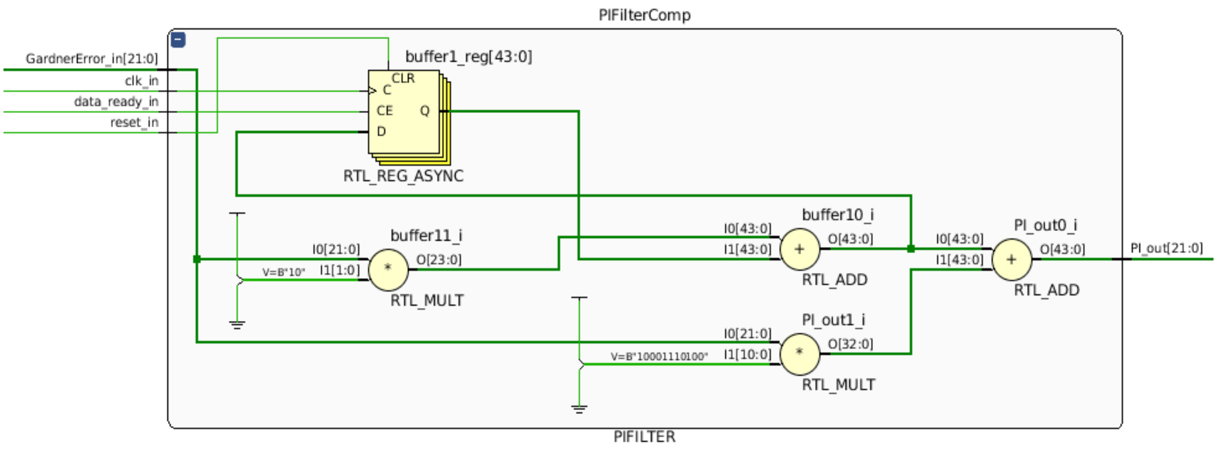


Figure 51 – PI VHDL RTL block.

4.1.4 NCO-T

As explained both in (RICE, 2009, p.475) and in (SAVVOPOULOS; ANTONAKOPOULOS, 2009), the objective of the NCO is to underflow (or overflow, in a crescent counter) in manner to align exactly with the base-point indexes m_k, in average after N samples, where N is the number of samples per symbol, updating the fractional index μ_k after every iteration. Implementing a decreasing counter, the counter decrements of 1/N in average and follows the recursion

$$\eta(n+1) = (\eta(n) - W(n)) \ \ mod\ 1. \tag{4.8}$$

where $W(n) = 1/N + \nu(n)$ and $\nu(n)$ is the PI filter output, closing the loop and correcting the right sampling moment. In (RICE, 2009, p.477) can be found a illustration that shows the relationship of similar triangles between $\eta(m_k)$ and $1 - \eta(m_k + 1)$, as shown in figure 52:

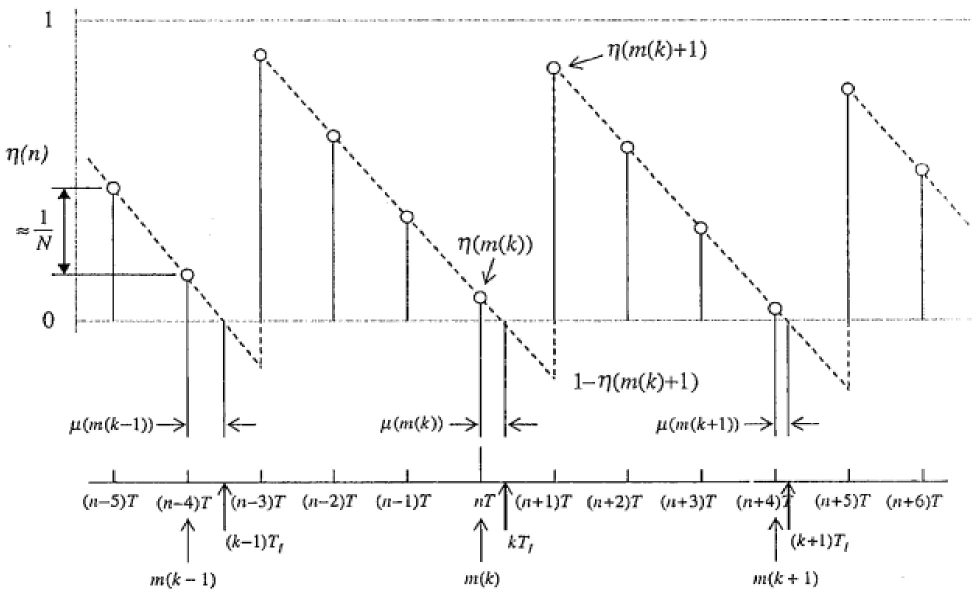


Figure 52 – Theoretical counter operation in NCO. (RICE, 2009)

Based on the figure 52 observation the relationship can be denoted as:

$$\frac{\mu(m_k)}{\eta(m_k)} = \frac{1-\mu(m_k)}{1-\eta(m_k+1)} \tag{4.9}$$

Solving for μ the following equation can be obtained:

$$\mu(m_k) = \frac{\eta(m_k)}{1-\eta(m_k+1)+\eta(m_k)} = \frac{\eta(m_k)}{W(m_k)} \tag{4.10}$$

Because (4.10) is a division and creates a more complex implementation in the digital level, then it's often seen the μ being approximated. Both in (FIALA; LINHART, 2015) and in (CARDENAS; AREVALO, 2015) FPGA implementations of symbol synchronization the designer made a decision to approximate μ as a constant multiplication because $W(m_k) = 1/N + \nu(m_k) \approx 1/N$ then μ can be approached as

$$\mu(m_k) = \eta(m_k)N. \tag{4.11}$$

As an iterative algorithm, after a step error, the μ needs a number of samples after it converges and locks in the stable number that represents the fractional delay, as can be seen in figure 53:

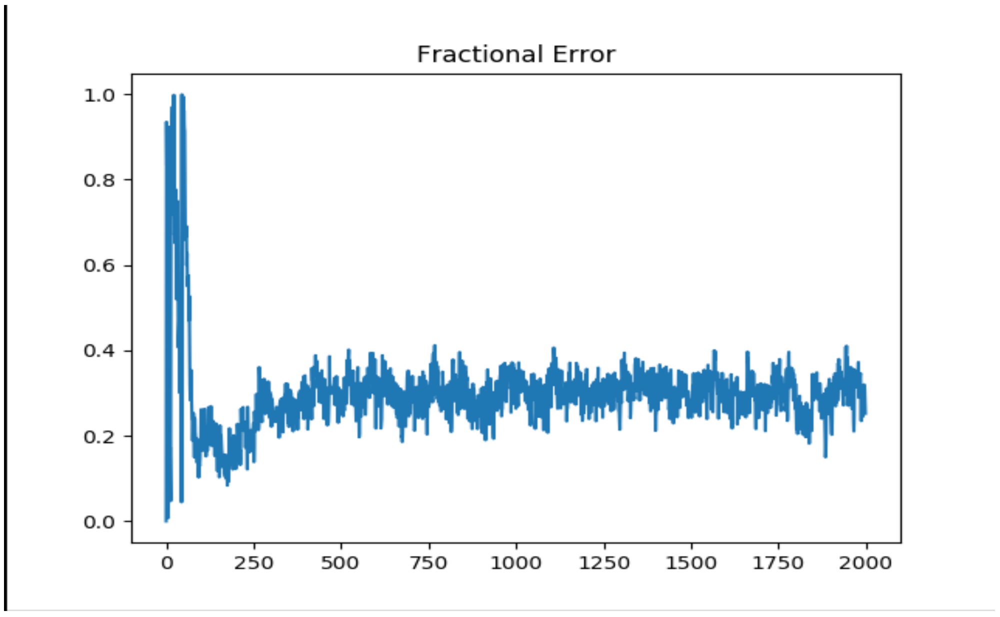
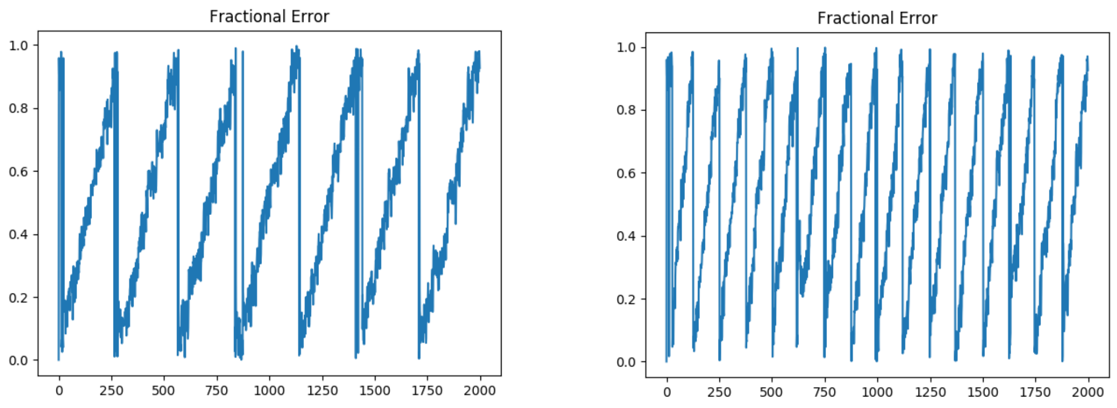


Figure 53 – Example of μ converging as the algorithm iterates.

When faced with a frequency step, the μ behaves as an "saw-tooth" wave, where the slope depends on the re-sampling factor, which is the proportion between the sampling frequency in the receptor over the transmitter, figure 54 show this behaviour:

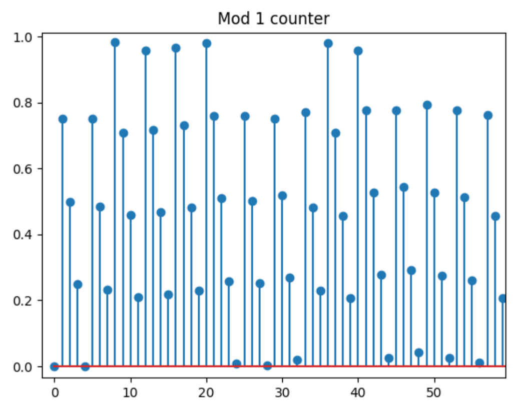


(a) μ slope with re-sampling factor = 1.001.

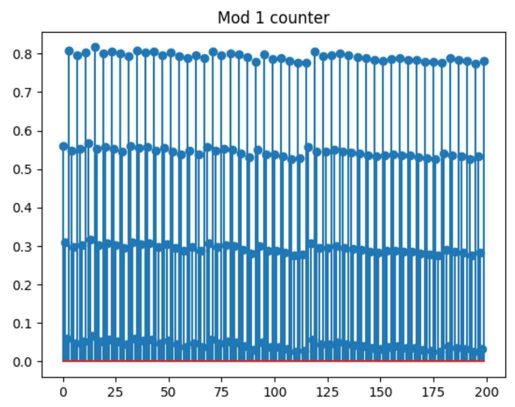
(b) μ slope with re-sampling factor = 1.002.

Figure 54 – Plot of μ behaviour with different re-sampling factors.

As one can notice, the whole system is basically a PLL and as a PLL is a closed loop, which means that the system control have a time to stabilize and lock, the same happens with the NCO counter, before the system locks, the counter moves in a pretty unstable manner then after the lock it is possible to see an average pattern close to what is ideal, as shown in the figures 55a and 55b:



(a) Counter before μ lock.



(b) Counter after μ lock.

Figure 55 – Mod 1 counter algorithm plots.

4.2 PYTHON and VIVADO simulation results

4.2.1 Block validation methodology

The validation methodology for the timing recovery block was based on the article (NAVARRO et al., 2002), where the author simulates and validates a timing error recovery loop with Gardner TED equal to the implemented in this text, firstly the S-curve for the Gardner TED is obtained, the S-curve is a plot of the average or expected value outputted from the timing error detection block versus the timing delay normalized by the timing symbol period, or

$$S(\tau) = E[V_d(k)|\tau], \tag{4.12}$$

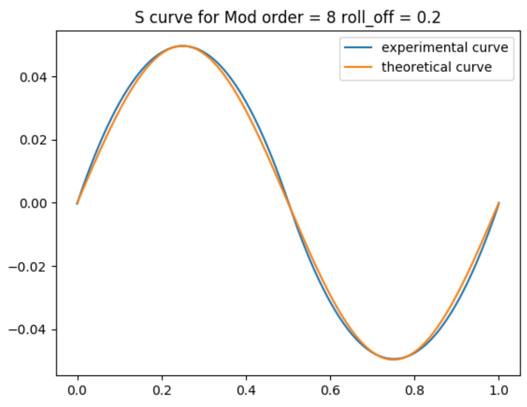
where $V_d(k)$ is the output from the GTED and $\tau = \frac{t_{delay}}{T_{symbol}}$. As stated in (RICE, 2009, p.459), the S-curve for the GTED can be represented analytically by the equation

$$s(\tau) = \frac{4K^2 E_{avg}}{T_s} C(\alpha) \sin\left(2\pi \frac{\tau}{T_s}\right) \tag{4.13}$$

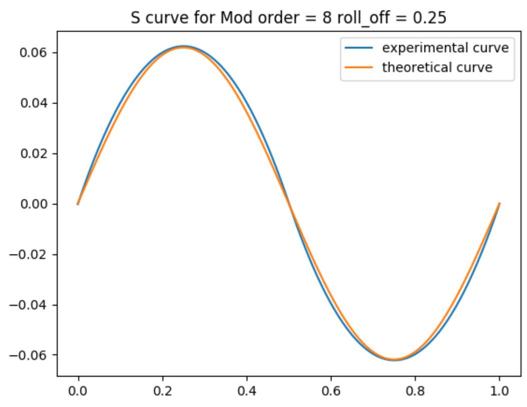
where K is the channel gain, which for our simulations will be considered 1, E_{avg} is the average symbol energy, which for the bit mapping in the DVB-S2 protocol is in most modulation order cases design to be 1 (ETSI, 2014a, p.25), T_s is the symbol period, as the system is synchronous based in the inputted samples, the sample period(T_{sample}) was set as 1 for the simulation which means $T_s = N$, with N being the number of samples per symbol, $C(\alpha)$ is a factor that depends on the roll-off factor, that can be determined by the equation

$$C(\alpha) = \frac{1}{4\pi\left(1 - \frac{\alpha^2}{4}\right)} \sin\left(\frac{\pi\alpha}{2}\right) \tag{4.14}$$

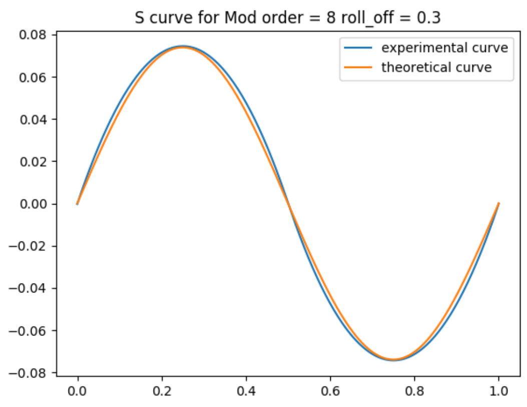
In the PYTHON simulated algorithm the GTED S-curve was plotted for the roll-off values of 0.2, 0.25 and 0.3 as it's determined by the DVB-S2 standard, the experiment was run for 2000 symbols each, giving 3 curves, showed in figure 56:



(a) S-curve for $\alpha = 0.2$.



(b) S-curve for $\alpha = 0.25$.



(c) S-curve for $\alpha = 0.3$.

Figure 56 – S-curve behaviour for different roll off values.

In figures 56a, 56b and 56c, can be noticed that the experimental curves fitted closely to the analytical ones.

As explained in the article (NAVARRO et al., 2002), there were two main input errors tested, the phase step error and the frequency step, basically the phase step is a delay in the transmitted samples when the wave form reaches the receptor, the frequency step; on the other hand, represents a difference in the sampling frequency between the transmitter and the receiver.

The phase step algorithm was made based on (RORABAUGH, 2004, p.356) where it's explained the process of interpolation necessary to shift the wave-form by a non-multiple of the sampling time as it happens in real applications. The interpolation of choice was a piece-wise linear interpolation, as it's shown in (RORABAUGH, 2004, p.368), the author asserts it provides sufficient accuracy for a signal sampled in a sufficiently high rate. For each constellation, there is, QPSK, 8PSK and 16APSK, the symbol synchronization block were tested both in PYTHON simulation and the VHDL design, with the phase step of $2.3T_s$, a shift of 2 samples and a third, in a system with 4 samples per symbol, the frequency step was tested with a re-sampling frequency of 1.001 the original sampling frequency.

The symbols were generated originally with AWGN based in a $E_b/N_0 = 20dB$, the system was simulated with $\zeta = 1/\sqrt{2}$ and a $B_nT_s = 5 \times 10^{-3}$.

After the VIVADO tesbench, the outputted results were then converted to a .txt file and applied to a PYTHON code, to plot the constellation results.

4.2.2 8PSK

4.2.2.1 8PSK PYTHON simulation results

Firstly the 8PSK constellation was generated based in the DVB protocol, made in a PYTHON algorithm, the result can be seen in figure 57:

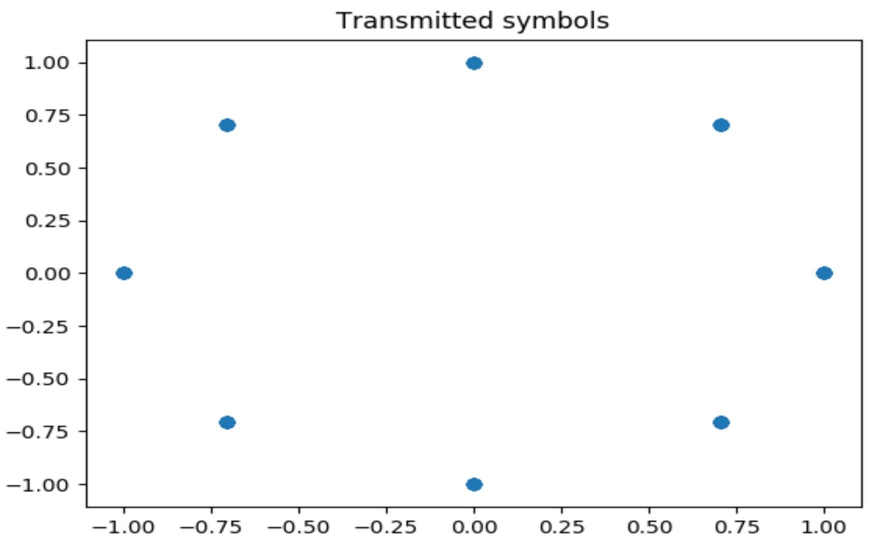


Figure 57 – Ideal 8PSK constellation.

Originally the simulation was set without the timing error correction block. The results are shown in figure 58:

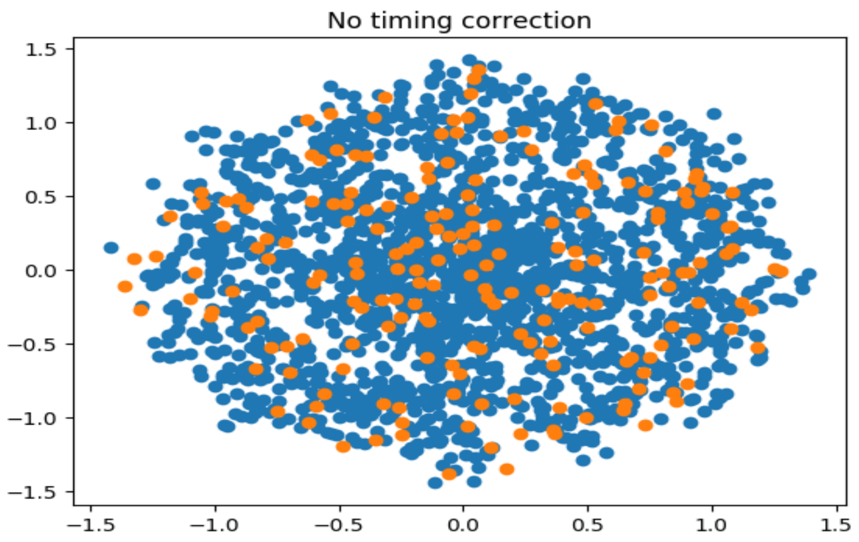


Figure 58 – Decoded symbols with no timing correction.

The first 10% symbols were highlighted, as a mean of seeing the symbols before the lock in the symbol synchronization block, as showed in figure 59:

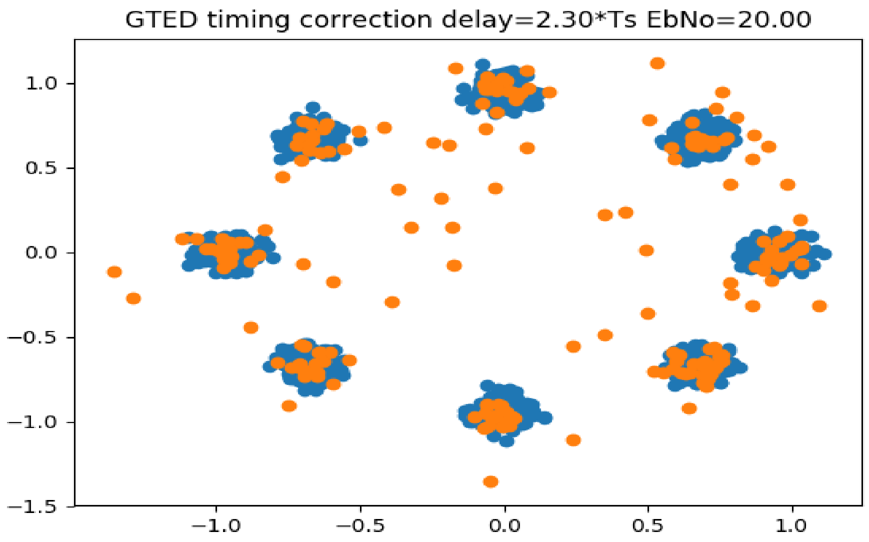


Figure 59 – Resulted symbols with Gardner algorithm.

It's possible to notice an overall improvement in the received constellation, the symbols before the lock can be seen as the constellation more distant points, then the system corrects the phase step and receives the correct symbols, with an error margin caused by the AWGN noise. The system μ convergence behaviour can be seen with the plot 60:

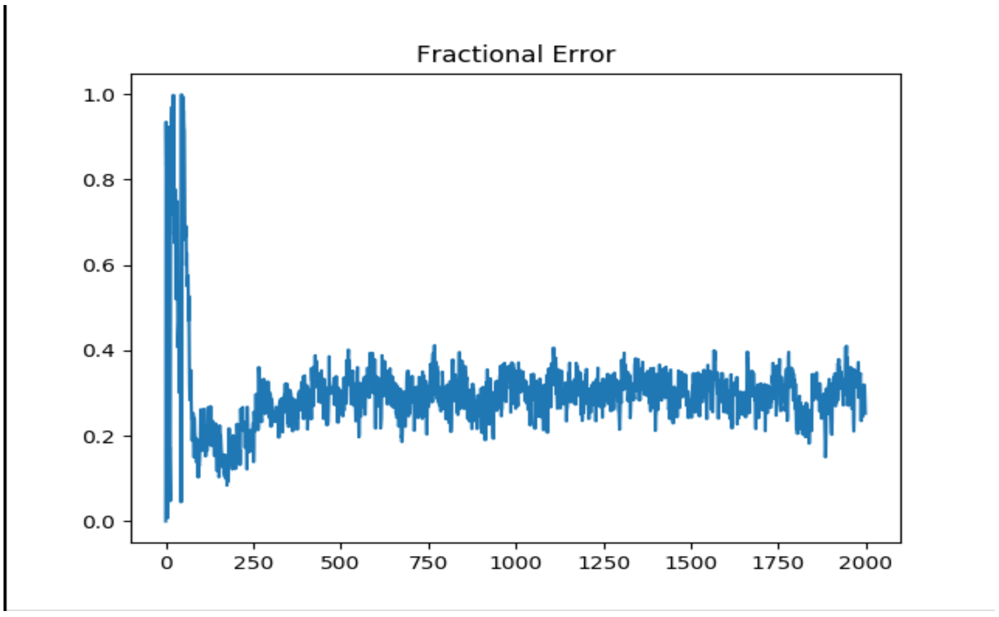


Figure 60 – Convergence of the μ value.

The μ value converges to 0.3, which is exactly the input fractional phase step. The GTED algorithm has a noticeable self-noise, which means that μ never converges to the exact fractional delay value, only oscillates around it.

When the frequency step was applied, it gave the constellation shown in figure 61:

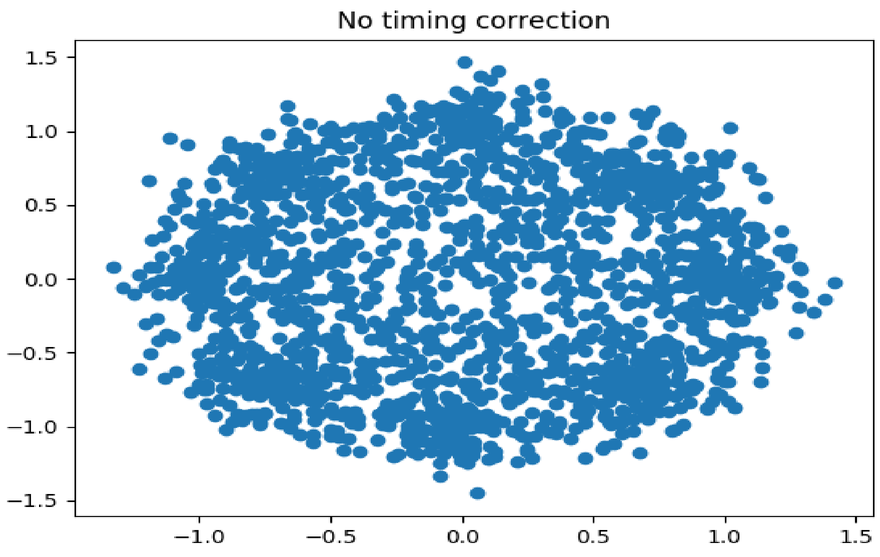


Figure 61 – 8psk constellation after frequency offset.

It is possible to observe in figure 61 that the spreading makes the overall constellation almost unrecognizable, while one is still able to notice some heavier point density in the right sending points.

When the algorithm were applied in PYTHON, it gave the constellation result in figure 62:

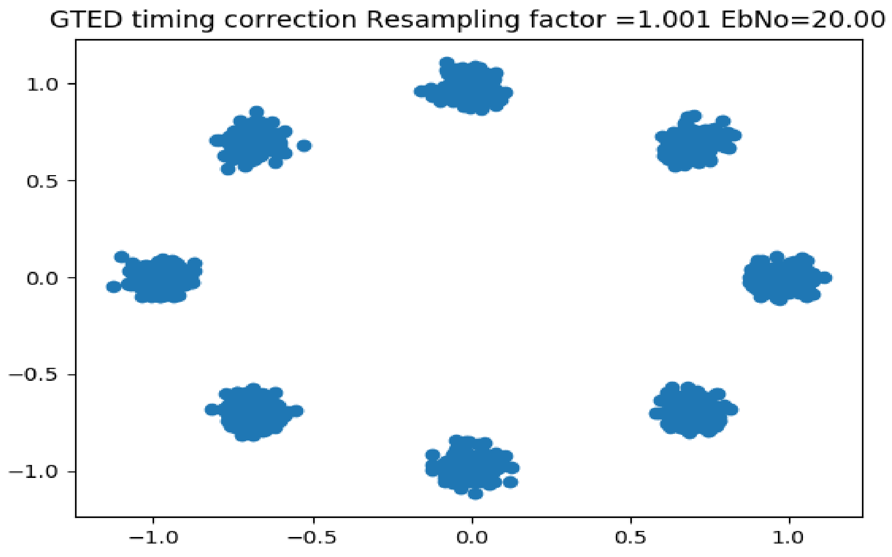


Figure 62 – 8PSK constellation after frequency step.

It's possible to see in figure 62 a much more clean constellation, with the defining points approximately in the right coordinates. A great improvement compared to the constellation received in figure 61.

When analysed the μ value convergence, it gave the figure 63;

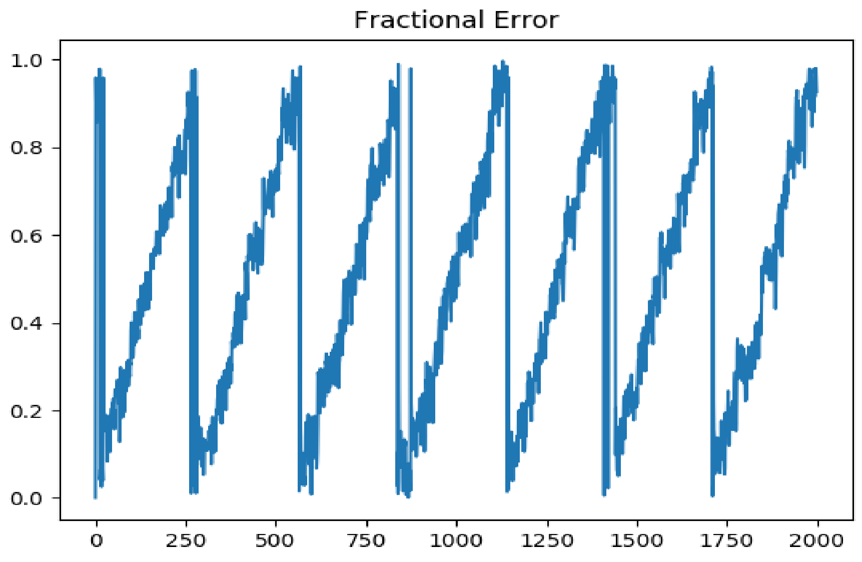


Figure 63 – μ value plot in the frequency offset.

As explained in chapter 4.1.4, this line shape that can be seen in figure 63 is natural to occur in a situation of sampling frequency difference and happened in the algorithm as expected.

Now after validating the algorithm in PYTHON, it was possible to take the inputs convert it to fixed point representation and apply to the testbench in the VIVADO software, as explained in section 2.

4.2.2.2 8PSK VIVADO simulation results

The first case, where a step delay were applied, the VHDL system gave the results in figure 64:

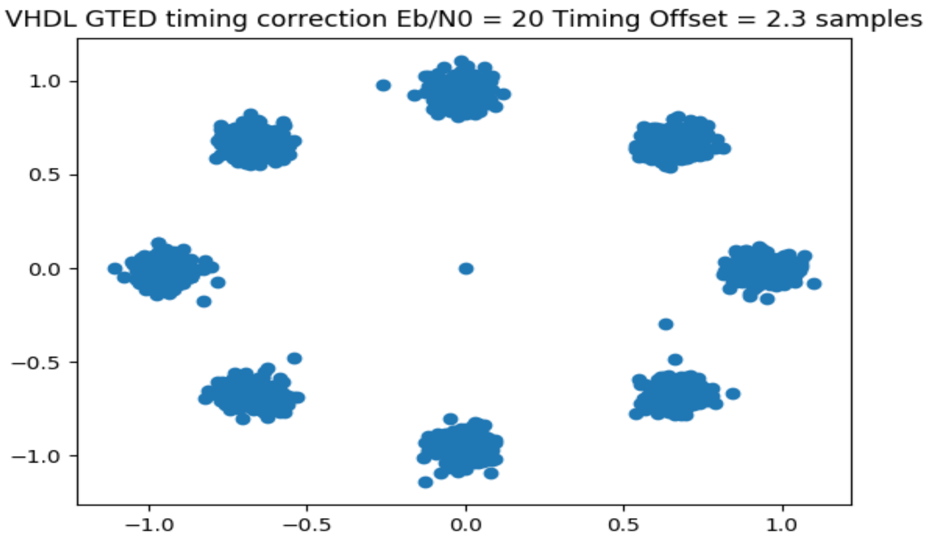


Figure 64 – Results out of VHDL simulation.

The constellation in VHDL of figure 64 is relatively more clean than the one from the PYTHON output 59, this is due to a more quickly μ convergence, probably caused by the fixed point representation due to quantization.

The plot for the μ was given by the VIVADO simulation interface, shown in figure 65:

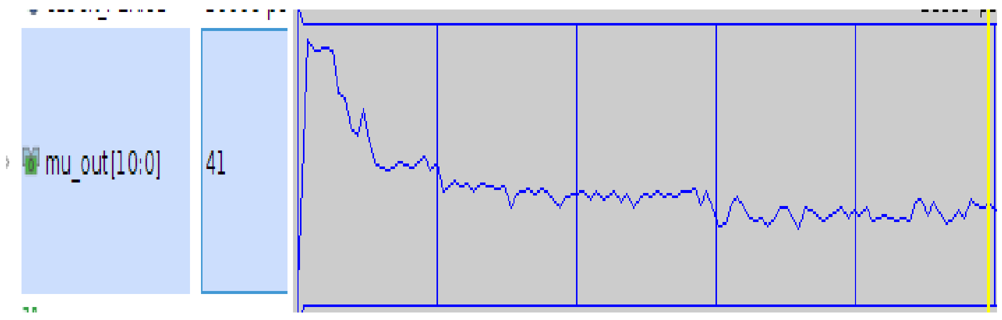


Figure 65 – μ convergence out of VHDL simulation.

If the value of convergence 40 is translated from it's fixed point binary representation to decimal it gives $\mu_{dec} = \mu_{binary}/2^{frac\ bits} = 40/2^7 = 0.312$, which is approximately the value of the fractional error, showing the system is approaching to the expected value. The frequency step was then applied to the system in VHDL, giving the constellation results in figure 66:

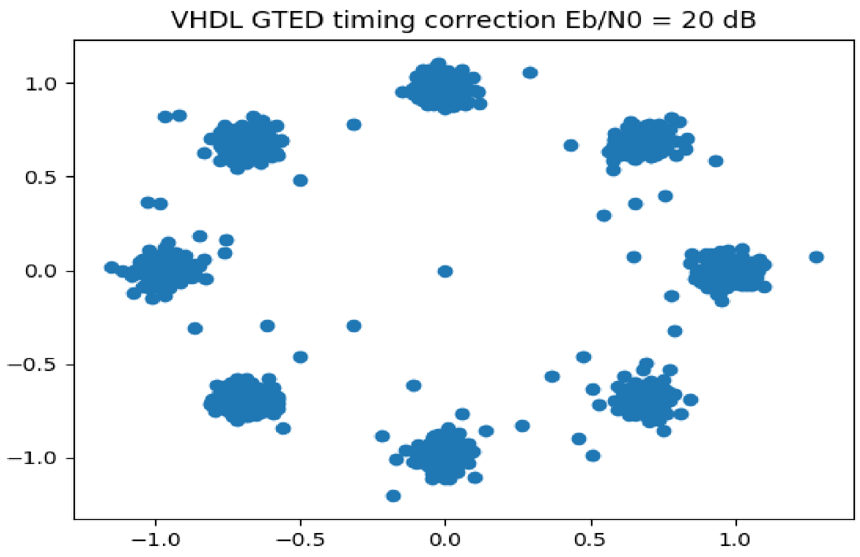


Figure 66 – Results out of VHDL simulation for the frequency step.

One can see that different from the case of the timing delay step, where the quantization helped stabilized the constellation, in the case of a frequency step the system was not as efficient as the algorithm in PYTHON, but ot managed to give a much more clear output constellation that the inputted one 58.

The μ value plot was then taken from the VIVADO simulation interface, it gave the results in figure 67:

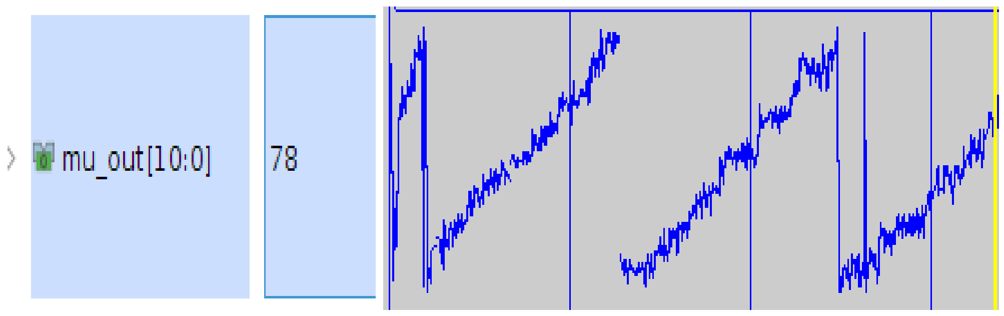


Figure 67 – μ plot out of VHDL simulation.

The μ VHDL plot had a line shape as expected from the theory, implying it is trying to interpolate to a fixed fractional space but the optimal interpolation point keeps changing with time.

Next this whole process was implemented for a QPSK modulation and then for a 16APSK modulation, as will be shown in the next sub-sections.

4.2.3 QPSK

4.2.3.1 QPSK PYTHON simulation results

the QPSK constellation points were generated in a algorithm in PYTHON, giving the constellation points in figure 68:

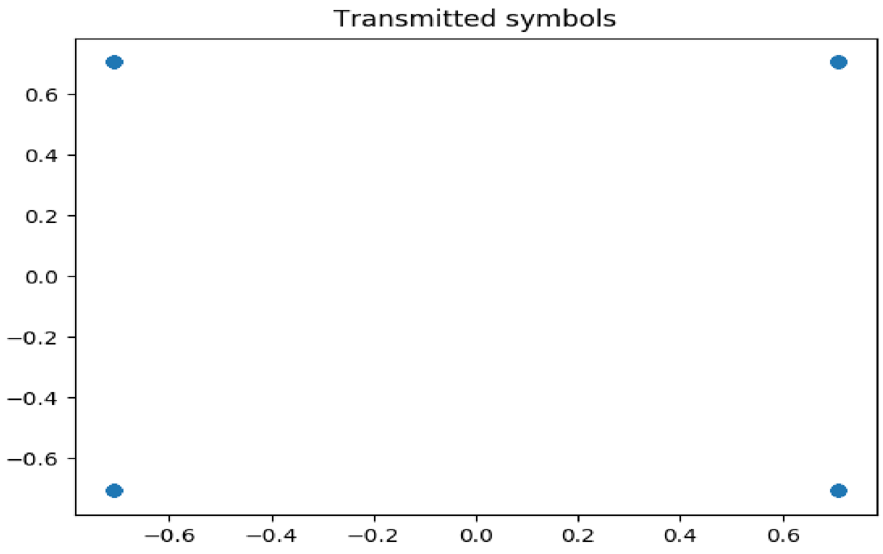


Figure 68 – Ideal transmitted symbols..

The QPSK constellation showed in 68 is formed of four point with 90^o angle from each other and a distance from the origin of 1.

To the system was them applied the AWGN channel and the timing delay step, with no TED the constellation can be seen in figure 69:

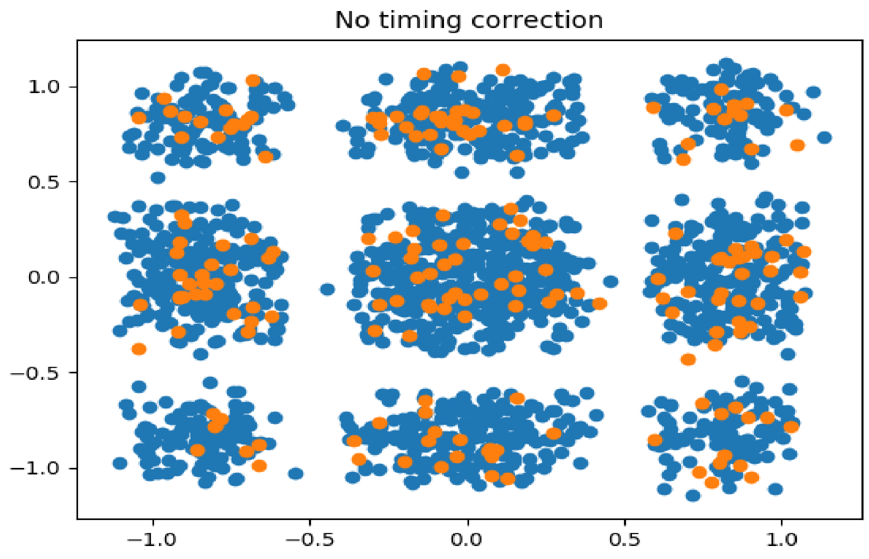


Figure 69 – Decoded symbols with no timing correction.

One can notice that in figure 69, constellation formed some "patches" , many in places where there were no points in the original constellation, implying de-modulation error.

Then the GTED algorithm was applied, giving the result in figure 70:

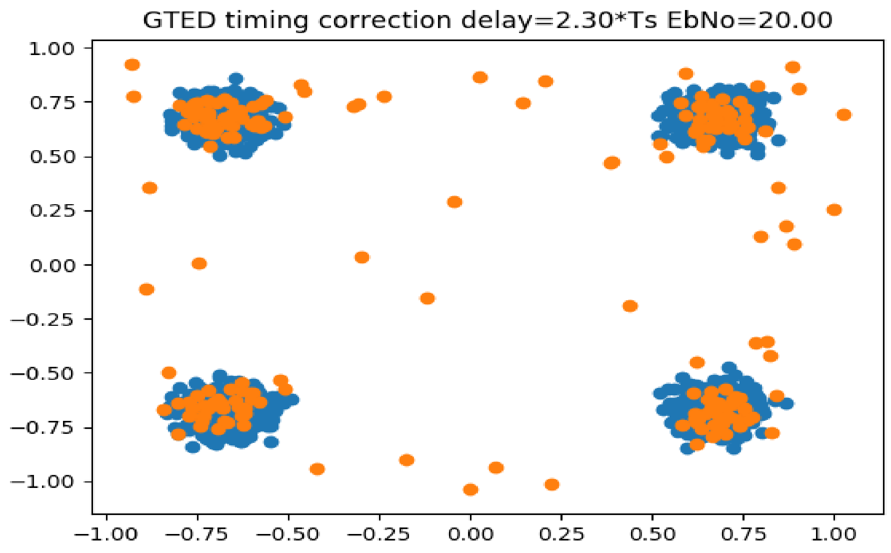


Figure 70 – Resulted symbols with Gardner algorithm.

Like the case in the 8PSK modulation in figure 59, the system took 250 samples to stablize and lock, them all other symbols were close to the transmitted points.

Them the frequency step was applied to the PYTHON code without timing correction, giving the figure 71:

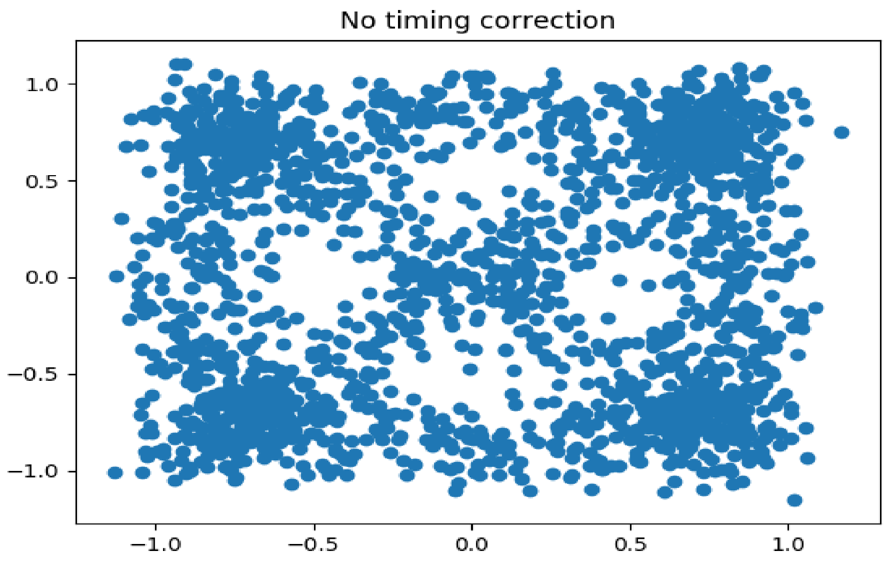


Figure 71 – QPSK constellation after frequency step no TED.

Closely resembling the 8PSK case, it's possible to see in figure 71 a heavier density around the points that actually are part of the original constellation, but many other points that transit around, making this specific shape.

When the GTED algorithm was applied in PYTHON it showed a major improvement from the no timing correction one, were every point received is closely matching some point from the original constellation. The results are shown in figure 72:

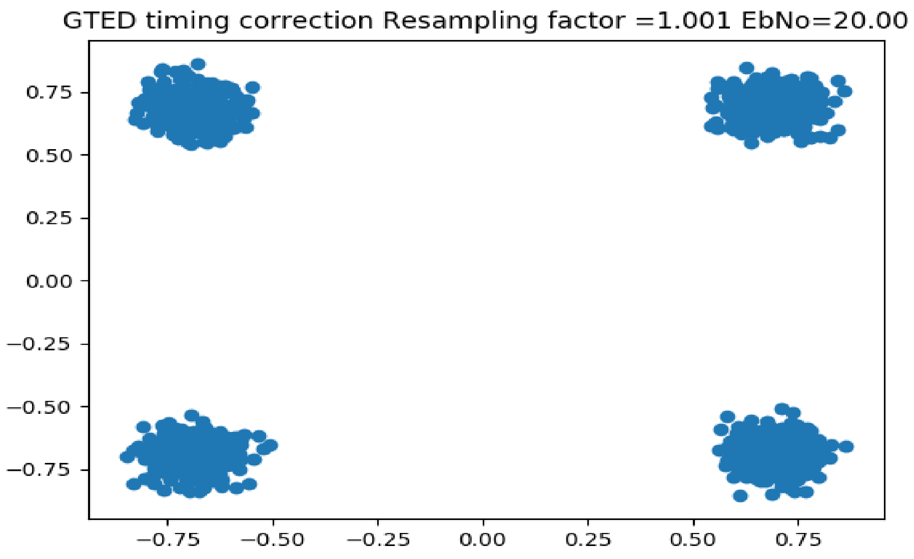


Figure 72 – QPSK constellation after frequency step with GTED.

4.2.3.2 QPSK VIVADO simulation results

The inputs were then simulated in the VHDL system, the outputted constellation can be seen in figure 73:

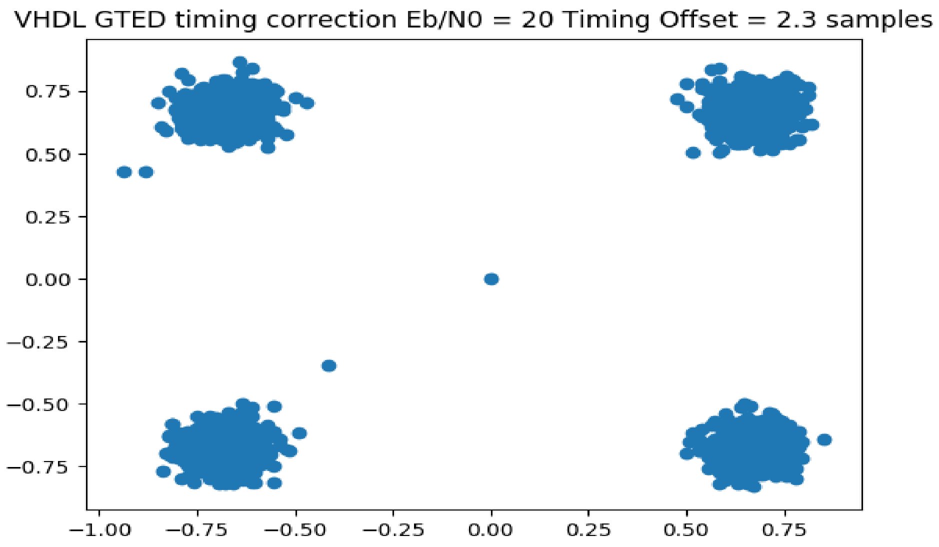


Figure 73 – Results out of VHDL simulation.

Like in the 8PSK case, the system was able to converge and lock.

The plot of the μ value given by the VIVADO testbench interface is shown in figure 74:

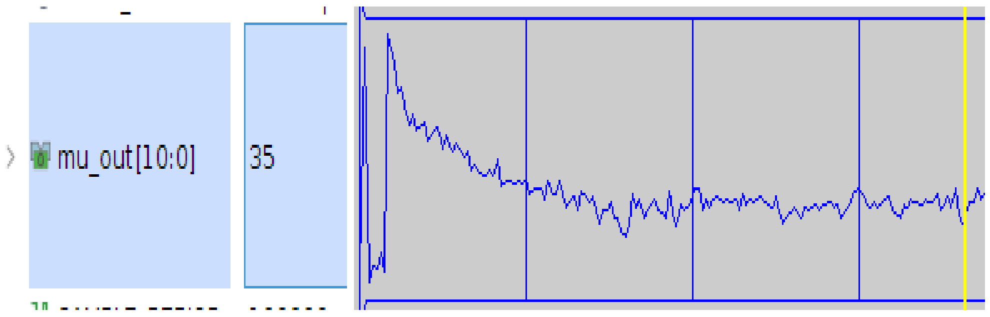


Figure 74 – μ convergence out of VHDL simulation.

The value 35 in the fixed point representation used is $\mu_{dec} = 35/2^7 = 0.273...$ which is approximately close to our desired value of 0.3, and the system has it's self noise, which contributes for this ever lasting oscillation around the desired value.

When the frequency step was inputted in the VHDL system, the output constellation is shown in figure 75:

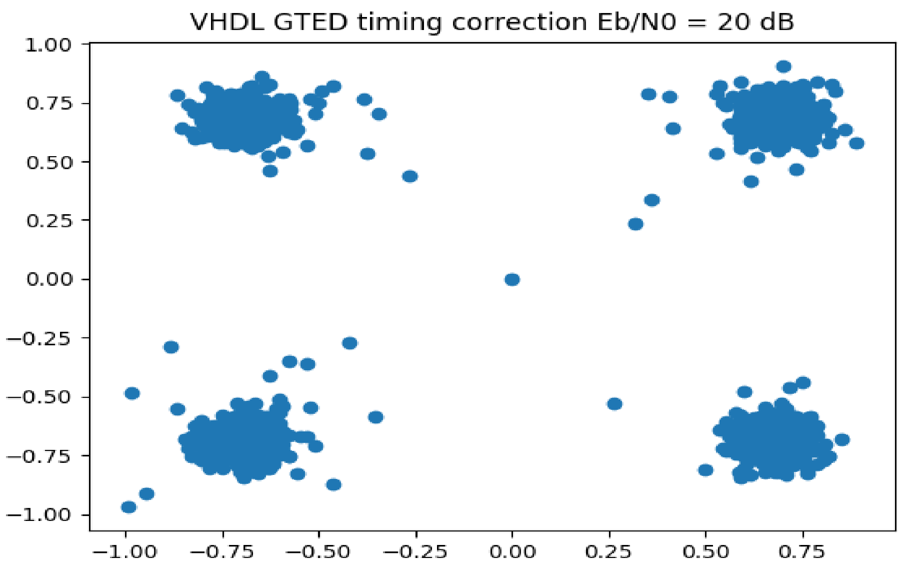


Figure 75 – Results out of VHDL simulation for the frequency step.

Much like the 8PSK case, the VHDL system was not able to give a constellation as clear as the PYTHON one, but was able to give a improve compared to the received constellation.

Figure 76 shows μ plot from the VIVADO simulation interface:

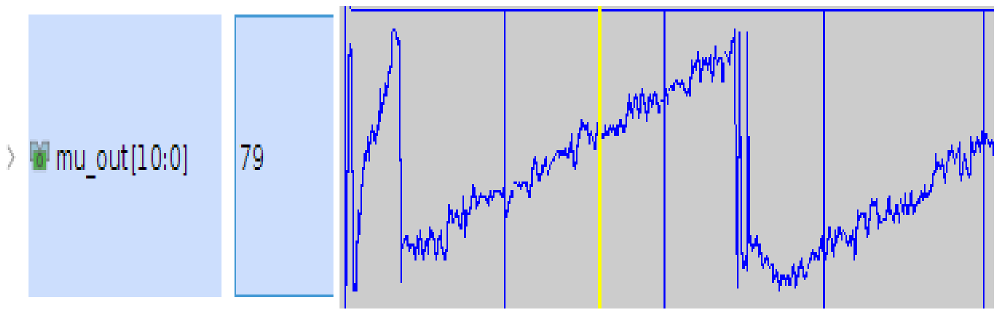


Figure 76 – μ plot out of VHDL simulation.

Where the same line pattern can be found, implying the system is trying to lock, but the symbol optimal interpolation moment changes in time.

4.2.4 16APSK

4.2.4.1 16APSK PYTHON simulation results

The figure 77 shows the ideal symbols generated by the PYTHON code:

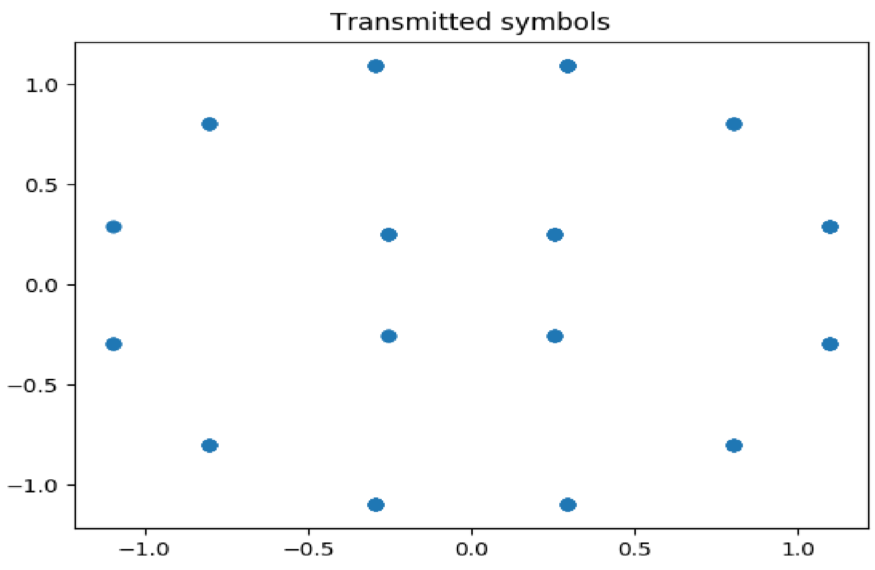


Figure 77 – Ideal transmitted symbols..

As can be seen in figure 77, the geometry for a 16APSK constellation is more complex than for the other ones showed until here, and the margin for errors is smaller.

Them the symbols were passed through the timing step delay and AWGN channel, giving the results in figure 78:

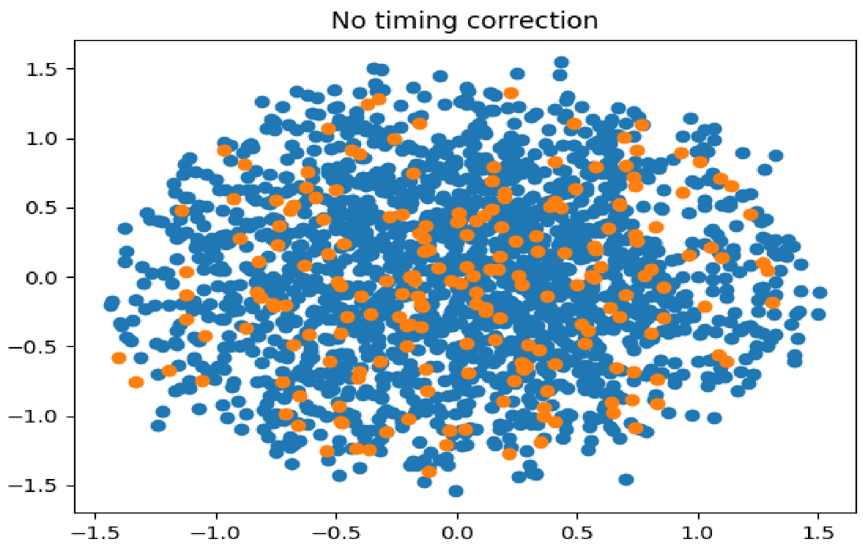


Figure 78 – Decoded symbols with no timing correction.

In figure 78 is possible to see that the original constellation is almost unrecognizable.

After being processed through the timing correction algorithm, the resulted constellation is seen in figure 79:

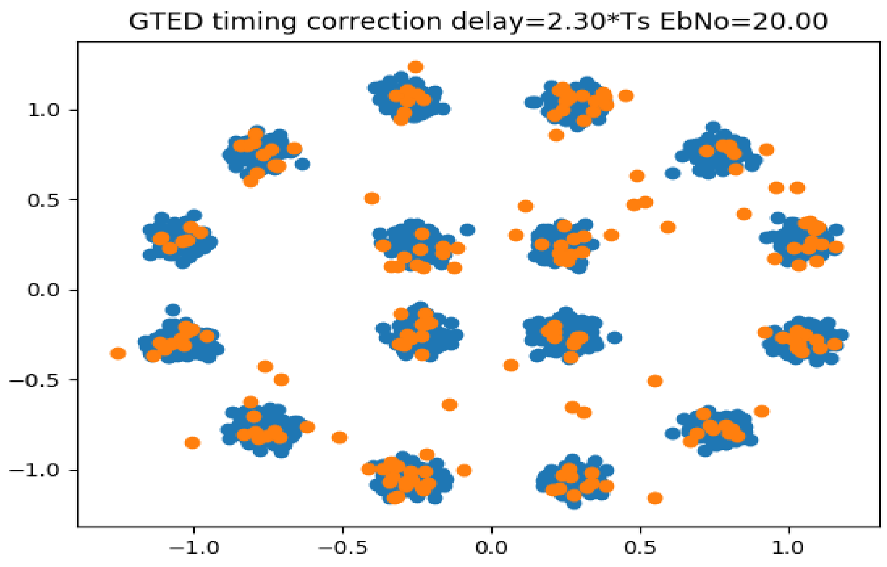


Figure 79 – Resulted symbols with Gardner algorithm.

is possible to see the constellation much more well-defined, implying less errors in the reception.

The frequency step was them applied to the 16APSK constellation, giving the constellation in figure 80:

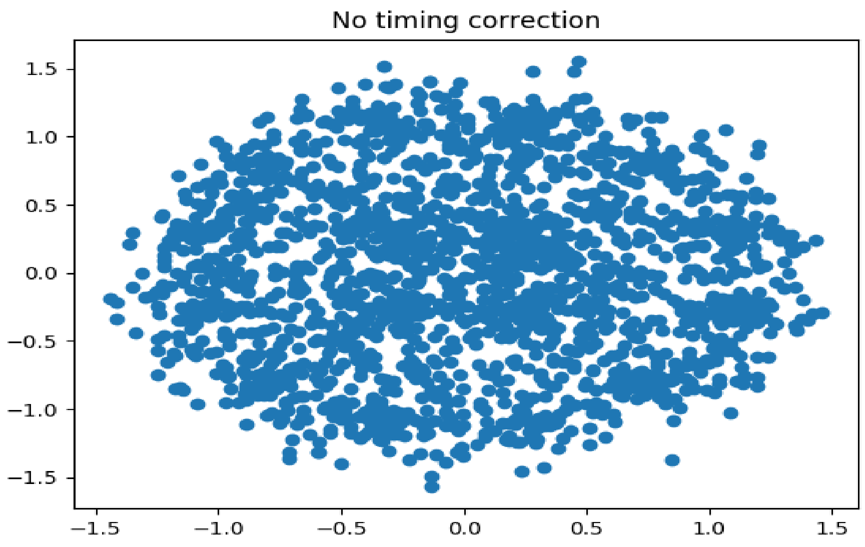


Figure 80 – 16APSK constellation after frequency step no TED.

In the constellation, the difference in sampling time adds errors, being impracticable for reliable information transmission.

Then the timing error correction algorithm was applied, giving the results in figure 81:

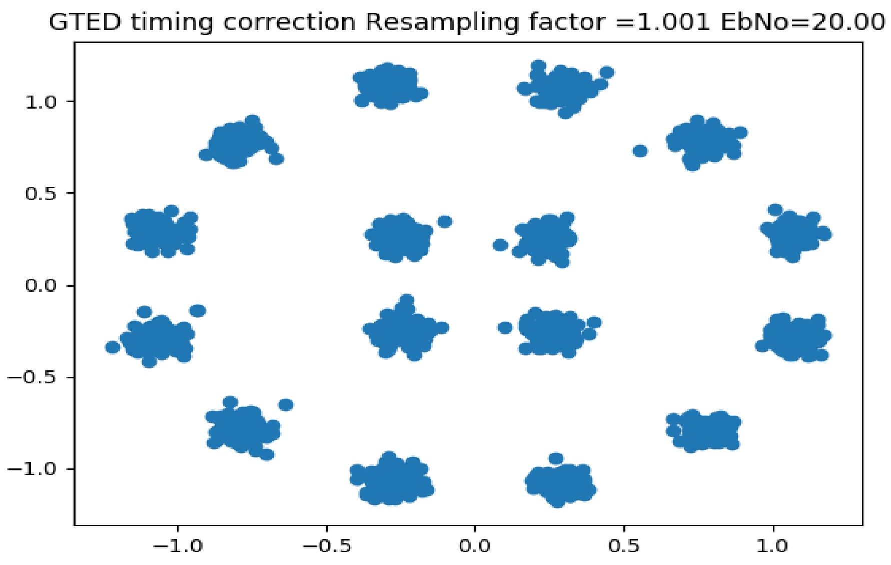


Figure 81 – 16APSK constellation after frequency step with GTED.

The constellation had a major improvement with GTED error correction algorithm, even when symbols are closer to each other like in the 16APSK case.

4.2.4.2 16 APSK VIVADO simulation results

The timing delay step input was applied to the VHDL coded system, the resulted constellation is shown in figure 82:

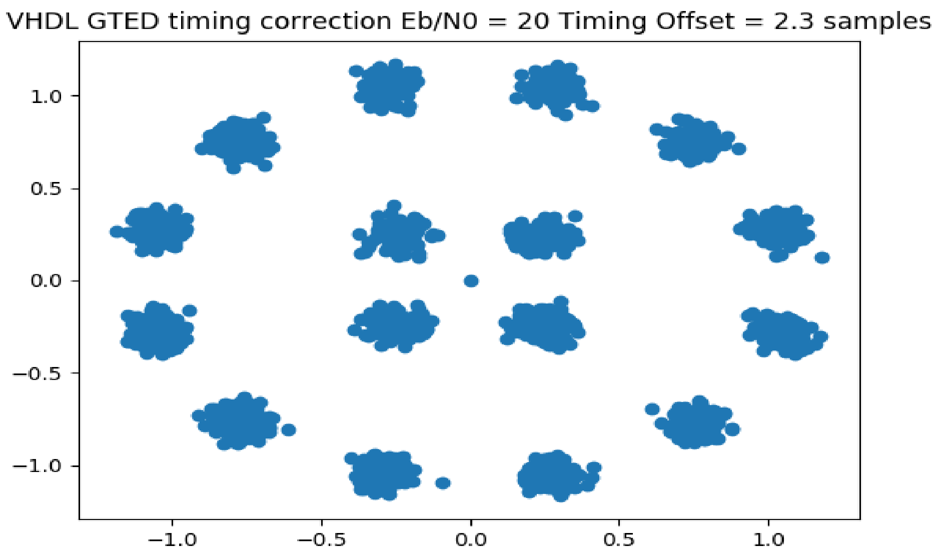


Figure 82 – Results out of VHDL simulation.

The figure 82 shows a very noticeable improvement from the no timing correction case, in which the constellation was almost unrecognizable.

The figure 83 shows the μ value plot from the VIVADO simulation interface:

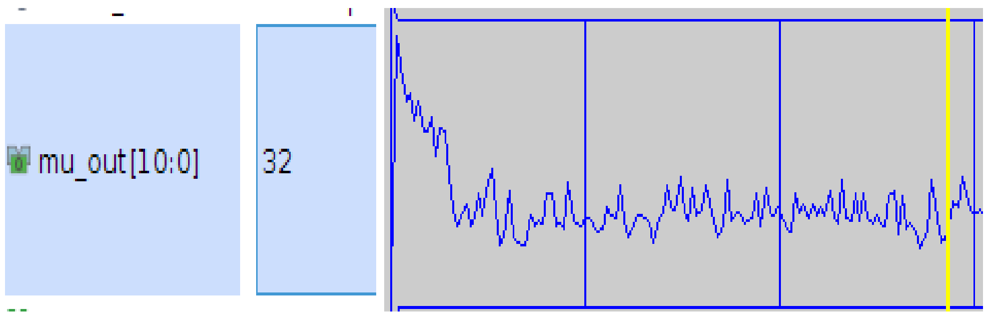


Figure 83 – μ convergence out of VHDL simulation.

32 in 11 bits fixed point signed representation is $\mu_{dec} = 32/2^7 = 0.25$, which is close to the value expected, as one can see from the plot, the system actual oscillates around an average value.

The frequency step input was then applied to the system, giving the output in figure 84:

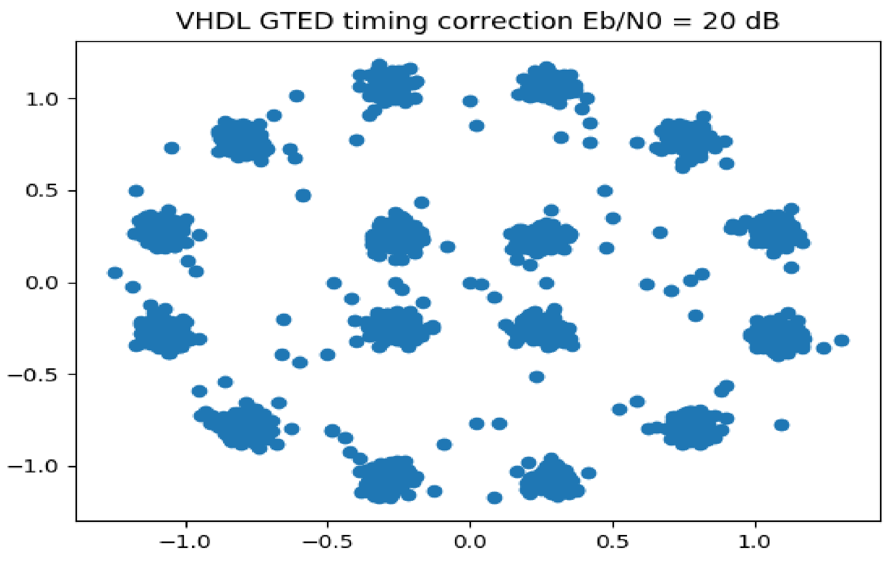


Figure 84 – Results out of VHDL simulation for the frequency step.

Like in the others modulations tested, there was improvement compared to the original inputted constellation, but different from the PYTHON simulation.

Figure 85 shows the μ value plot for the frequency step:

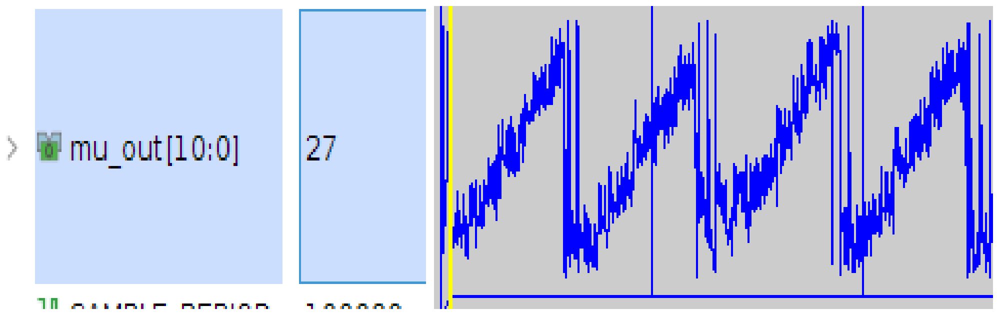
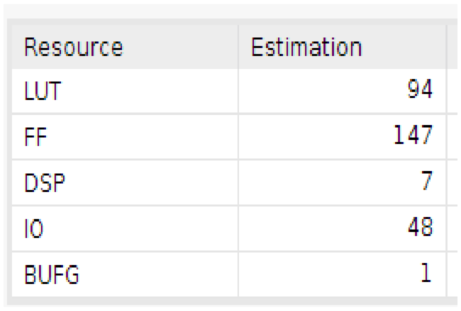


Figure 85 – μ plot out of VHDL simulation.

It's possible to see the same expected line shape plot, implying that the system implemented is really non-data aided, because the response depends on the overall shape of the transmitted waveform, not it's constellation contents.

Figure 86 shows the resources used by the Timing Recovery block estimated by the VIVADO interface:



Resource	Estimation
LUT	94
FF	147
DSP	7
IO	48
BUFG	1

Figure 86 – Post-Synthesis resource estimation for the Timing Recovery Block.

5 Symbol De-Scrambler Block

5.1 Symbol De-Scrambler Block

In the Symbol Descrambler is inputted the received symbols coordinates and the block outputs the symbols multiplied by a unitary complex factor with a phase multiple of $\pi/2$.

In (ETSI, 2014a) is specified that the transmission symbols that do not belong to the PLHEADER, or physical layer symbol header, are multiplied by a complex number defined by a pseudo-random sequence. The block Symbol Descrambler of the receptor is responsible for accomplish the inverse operation, multiplying the received symbols by a factor inverse to the factor in the transmission.

For each symbol it is calculated the next complex factor in the sequence, the n^{th} Gold code sequence z_n is defined as

$$z_n(i) = [x((i+n)\ modulo\ (2^{18}-1) + y(i)]\ modulo\ 2. \tag{5.1}$$

where $x(i)$ and $y(i)$ are sequences that follow the recursion:

$$\begin{aligned} &x(i+18) = x(i+7) + x(i)\ modulo\ 2, \\ &y(i+18) = y(i+10) + y(i+7) + y(i+5) + y(i)\ modulo\ 2. \end{aligned} \tag{5.2}$$

The initial conditions for $i = 0, 1..17$ for $y(i)$ and $x(i)$ is:

$$\begin{aligned} &x(0) = 1, x(1) = x(2) = ...x(17) = 0, \\ &y(0) = y(1) = y(2) = ...y(17) = 1. \end{aligned} \tag{5.3}$$

The Gold code sequence is then converted to a integer $R_n(i)$ between 0 and 3, with the following equation:

$$R_n(i) = 2z_n((i+131072)\ modulo\ (2^{18}-1)) + z_n(i). \tag{5.4}$$

This set of equations and relationships can be represented in the form of logic circuit as a combination of two linear feedback shift register (LFSR), as shown in figure 87:

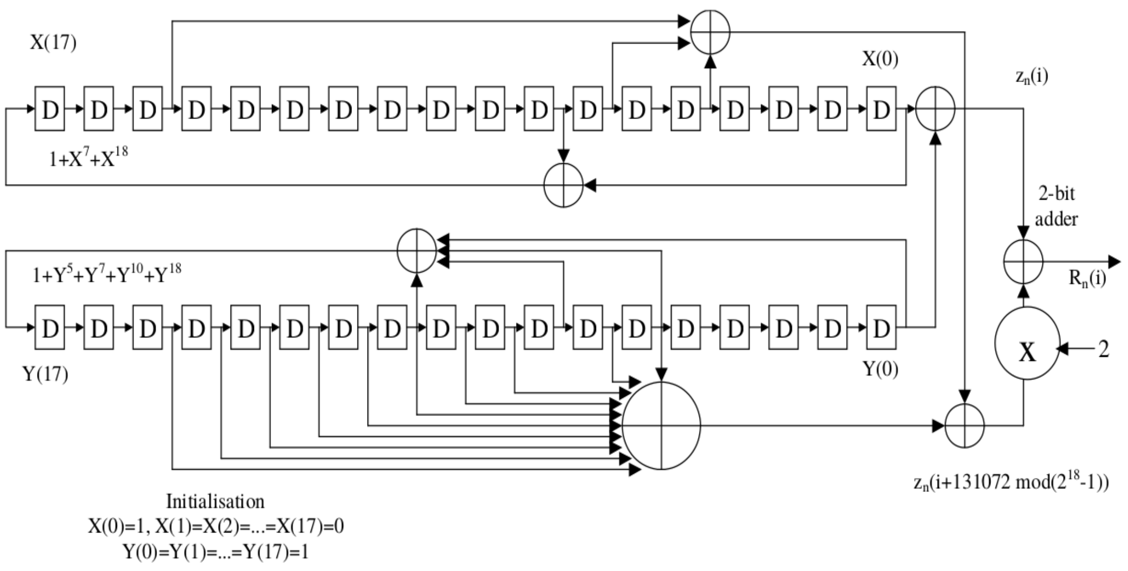


Figure 87 – Symbol Scrambling block from the DVBS2X standard (ETSI, 2014a, p.33)

where the n defines the iteration in which the sequence computation begins and the R_n is then used to define a complex factor, of the type

$$C_I(i) + jC_Q(i) = e^{jR_n(i)\pi/2}, \tag{5.5}$$

as the phase is defined as a multiple of $\pi/2$ then the factor acts as a predictable table of sign changes in the symbol coordinates, as shown in table 3 :

Rn	$C_I(i) + C_Q(i)$	$I_{Scrambled}$	$Q_{Scrambled}$
0	1	I	Q
1	j	-Q	I
2	-1	-I	-Q
3	-j	Q	-I

Table 3 – Table of signal changes based on inputted symbol coordinates I and Q.

So as a mean to reverse the process of scrambling, in the receptor is necessary to implement the table 4:

Rn	$C_I(i) + C_Q(i)$	$I_{De-scrambled}$	$Q_{De-scrambled}$
0	1	I	Q
1	-j	Q	-I
2	-1	-I	-Q
3	j	-Q	I

Table 4 – Table of the descrabling signal changes based on inputted symbol coordinates I and Q.

5.2 RTL VHDL blocks

Firstly the LFSR that represents the $x(i)$ sequence were coded, the generated RTL block design is exhibited in figure 88:

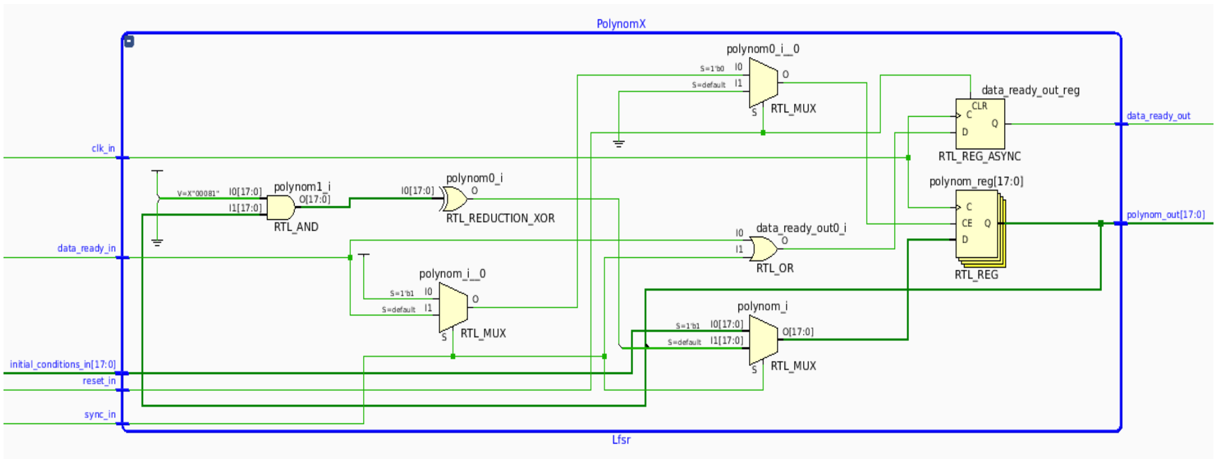


Figure 88 – RTL block of the LFSR for the x(i) equation.

Then the code that generates the $y(i)$ LFSR were made, figure 89 shows the RTL diagram:

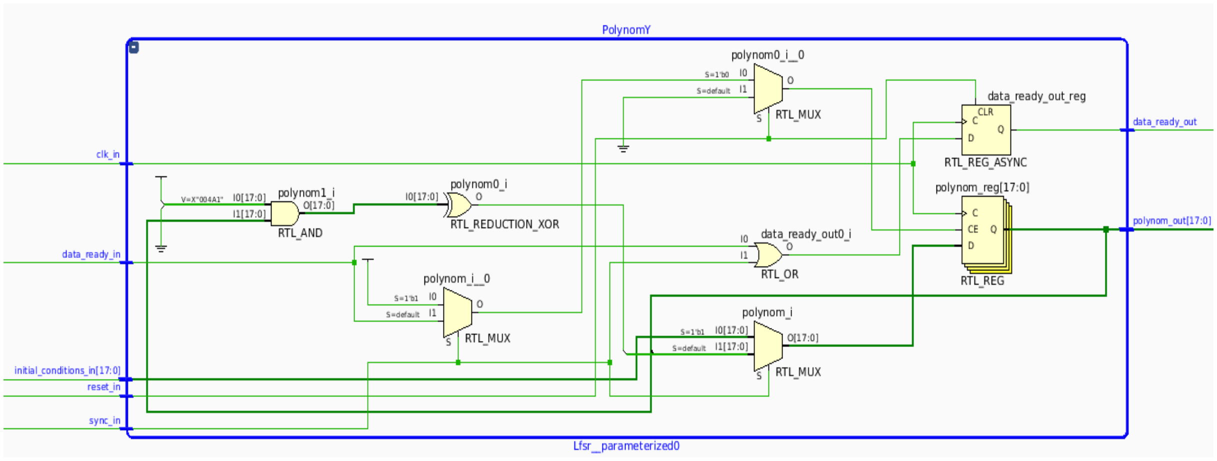


Figure 89 – RTL block of the LFSR for the y(i) equation.

Both those blocks were combined with the connection combinational logic defined by the protocol block design showed in the figure 87, generating the RTL in figure 90:

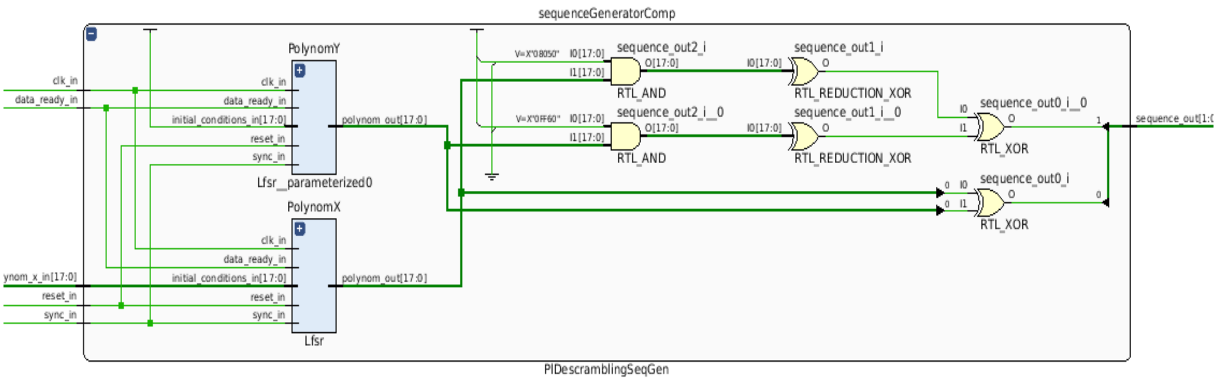


Figure 90 – Sequence generator RTL block.

And lastly, the code that apply the table 4, or in other words, the codes that inverts the symbols order and makes the two's complement when changing the symbol signal is necessary. were coded.

The RTL in figure 91 shows the whole system Top-level block:

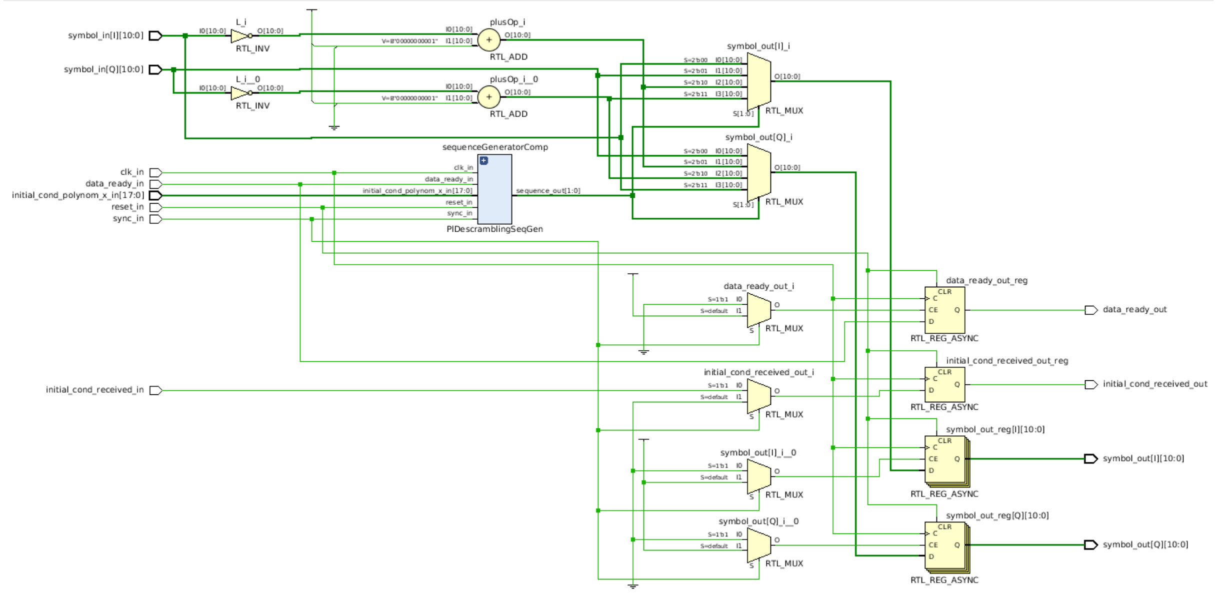
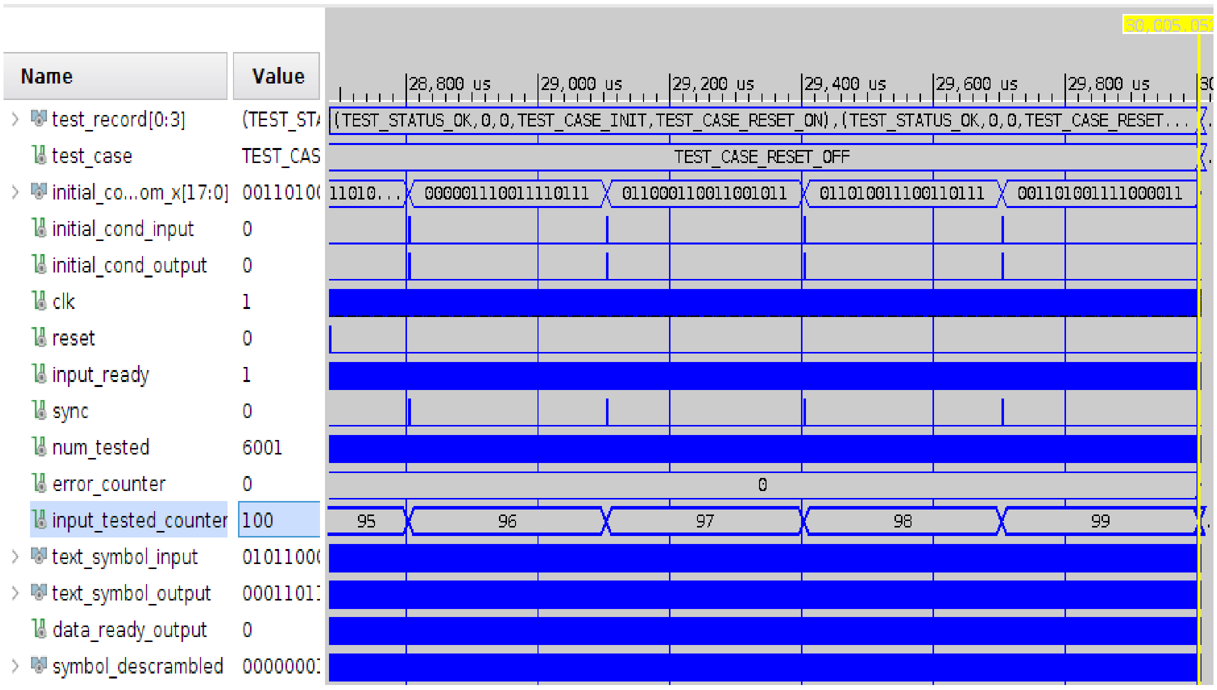


Figure 91 – Symbol de-scrambling RTL block.

5.3 VIVADO simulation results

The system was simulated with 100 different n values and for each n value there were 6000 different inputted symbols tested, in other words, it as tested 6×10^5 different inputted symbols. The results in the VIVADO GUI are shown in figure 92:

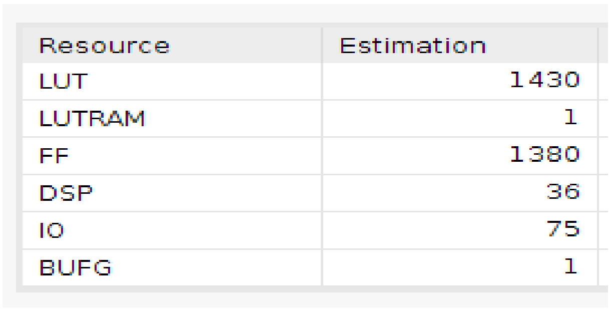


Name	Value
test_record[0:3]	(TEST_ST/
test_case	TEST_CAS
initial_co...om_x[17:0]	00110100
initial_cond_input	0
initial_cond_output	0
clk	1
reset	0
input_ready	1
sync	0
num_tested	6001
error_counter	0
input_tested_counter	100
text_symbol_input	0101100(
text_symbol_output	0001101:
data_ready_output	0
symbol_descrambled	0000000:

Figure 92 – VIVADO symbol de-scrambling testbench.

In figure 92 the signal error counter can be seen as 0, which implies there were no errors in the design simulation, that it was able to equally match the system simulated in high-level MATLAB language.

Figure 93 exhibits the resources table estimated post-synthesis for the symbol de-scrambling block generated by the VIVADO interface:



Resource	Estimation
LUT	1430
LUTRAM	1
FF	1380
DSP	36
IO	75
BUFG	1

Figure 93 – Post-synthesis resource utilization estimation.

6 Conclusion

In this text was implemented for the downlink three blocks from the DVBS2X protocol in VHDL language. As the standard only specify clearly the transmitter, inspired by the academic literature, the architecture implemented was validated as an viable alternative for FPGA SDR implementation, every and each algorithm was validated based on a previous high-level language (PYTHON or MATLAB) language and compared in a text vector simulation to guarantee that the results were matching. The overall curves of convergence and constellation effects for the Timing Recovery block matched with the proposed reference (NAVARRO et al., 2002), both for the Frequency step test case and the timing delay offset test case. The LLR block and the Symbol Physical layer De-scrambler also showed no faults for every test case simulated.

The importance of those blocks is made clear when the effects of the AWGN channel and differences in timing occur, being vital for a reliable information transmission in a satellite link. The constellation forms without correction were almost unrecognizable with the original send, implying errors that would not be corrected otherwise.

The Gardner Timing Error correction algorithm showed to be viable in terms of hardware complexity for an implementation in low level description hardware language, like VHDL. Also the considerations made in this text for the LLR generated a viable option for hardware implementation, with no transcendental functions and no non-constant divisions.

For a future work, the blocks can be integrated with each other and all the other blocks in the DVB-S2X standard, also to be completed the SDR, it's necessary to implement an uplink, or in other words a link from the receptor to the transmitter. As a matter of standard, a viable choice for the uplink protocol could be the Return Channel Via Satellite Second Generation Protocol (DVB-RCS2), which is an on-demand multimedia satellite communication system.

Bibliography

BOWYER, M. et al. *Software Defined Radio (SDR) Architecture to Support Multi-Satellite Communications*. Hampshire, United Kingdom: [s.n.], 2015. Cited in page 25.

CARDENAS, D.; AREVALO, G. All digital timing recovery and FPGA implementation. 2015. Cited in page 69.

ETSI. *Digital Video Broadcasting (DVB): Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*. [S.l.: s.n.], 2014. Cited 12 times in the pages 15, 17, 28, 29, 35, 37, 38, 39, 47, 71, 89, and 90.

ETSI. *Digital Video Broadcasting (DVB): Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVBS2 Extensions (DVB-S2X)*. [S.l.: s.n.], 2014. Cited in page 51.

FIALA, P.; LINHART, R. Symbol synchronization for SDR using a polyphase filterbank based on an FPGA. 2015. Cited in page 69.

GARDNER, F. M. A BPSK/QPSK timing-error for detector for sampled receivers. *IEEE Transactions on communications*, 1986. Cited in page 62.

GARDNER, F. M. Interpolation in digital modems- part ii: Implementation and performance. *IEEE Transactions on communications*, 1993. Cited in page 63.

HOSKING, R. H. *Putting FPGAs to Work in Software Radio Systems*. New Jersey: [s.n.], 2012. Cited 3 times in the pages 15, 23, and 24.

LATHI, B. P. *Modern Digital and Analog Communication Systems*. 2a. ed. [S.l.]: Oxford University Press, 1995. Cited 2 times in the pages 15 and 28.

LIMA, E. R. de et al. *A Detailed DVB-S2 Receiver Implementation: FPGA Prototyping and Preliminary ASIC Resource Estimation*. [S.l.: s.n.], 2014. Cited in page 30.

NAVARRO, R. D. et al. Performance analysis of the Gardner Timing detector over $\pi/4$-dqpsk modulation. 2002. Cited 3 times in the pages 71, 72, and 95.

OLIVATTO, V. B. *Analysis of LDPC decoders for DVB-S2 using LLR approximations*. Campinas, SP: Universidade Estadual de Campinas, 2016. Cited in page 48.

RICE, M. *Digital Communication: A discrete time approach*. [S.l.: s.n.], 2009. Cited 9 times in the pages 16, 61, 62, 63, 66, 67, 68, 69, and 71.

RORABAUGH, C. B. *Simulating Wireless Communication Systems*. USA: [s.n.], 2004. Cited in page 72.

SADEK, A. et al. *On the Use of Dynamic Partial Reconfiguration for Multi-band/Multi-standard Software Defined Radio*. Cairo University, Giza, Egypt: IEEE, 2015. Cited 2 times in the pages 15 and 24.

SAVVOPOULOS, P.; ANTONAKOPOULOS, T. Comparative performance analysis of symbol timing recovery for dvb-s2 receivers. 2009. Cited 4 times in the pages 62, 65, 66, and 68.

SCIAGURA, E. et al. An efficient and optimized FPGA feedback M-PSK symbol timing recovery architecture based on the gardner timing error detector. 2007. Cited in page 65.

SHORT, K. L. *VHDL For Engineers*. 1st edition. ed. [S.l.]: Pearson, 2009. ISBN 97801314244784. Cited 3 times in the pages 15, 43, and 45.

SKALAR, B. *Digital Communications: Fundamentals and Applications*. 2a. ed. Nova Jersey, Estados Unidos: Prentice Hall, 2017. Cited 6 times in the pages 15, 27, 32, 35, 36, and 39.

VASILESCU, G. *Electronic Noise and Interfering Signals*. 1a. ed. [S.l.]: Springer Science and Business Media, 2006. Cited in page 31.

WEKERLY, T. et al. *Status and Trends of Smallsats and Their Launch Vehicles — An Up-to-date Review*. São José dos Campos: [s.n.], 2017. Cited 2 times in the pages 15 and 25.

XILINX. *AXI Reference Guide*. [S.l.: s.n.], 2011. Cited in page 43.

XILINX. *Field Programmable Gate Array*. [s.n.], 2018. Disponível em: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>. Cited in page 25.