Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

Implementação em FPGA do módulo corretor fino de frequência e desenvolvimento do simulador de enlace em aritmética de ponto fixo aderentes ao protocolo DVB-S2X

Autor: Ítalo Barbosa Santos
Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF
2021

Ítalo Barbosa Santos

Implementação em FPGA do módulo corretor fino de frequência e desenvolvimento do simulador de enlace em aritmética de ponto fixo aderentes ao protocolo DVB-S2X

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

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2021

Ítalo Barbosa Santos

Implementação em FPGA do módulo corretor fino de frequência e desenvolvimento do simulador de enlace em aritmética de ponto fixo aderentes ao protocolo DVB-S2X/ Ítalo Barbosa Santos. – Brasília, DF, 2021-

83 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2021.

1. Hardware Reconfigurável. 2. Sincronismo de Frequência e Simulador de Enlace. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação em FPGA do módulo corretor fino de frequência e desenvolvimento do simulador de enlace em aritmética de ponto fixo aderentes ao protocolo DVB-S2X

CDU 02:141:005.6

Ítalo Barbosa Santos

Implementação em FPGA do módulo corretor fino de frequência e desenvolvimento do simulador de enlace em aritmética de ponto fixo aderentes ao protocolo DVB-S2X

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, 11 de Novembro de 2021:

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Gilmar Silva Beserra
Convidado 1

Prof. Dr. Daniel Costa Araujo
Convidado 2

Brasília, DF
2021

Agradecimentos

Gostaria de agradecer a minha família por sempre me proporcionar a melhor educação possível para que eu chegasse onde estou hoje em dia; minha namorada Tamires Saboia que durante a pandemia foi meu maior apoio emocional; meus amigos do grupo SAMU por ter me ajudado a manter o foco durante o curso; meu amigo Marcos Adriano por ter sido principal responsável pela oportunidade dada em participar do projeto que originou o tema apresentado neste documento e meus professores por me conceder o conhecimento necessário para o desenvolvimento deste trabalho, em especial meu orientador Daniel Mauricio Muñoz Arboleda.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Neste trabalho é apresentado o desenvolvimento de um simulador da etapa de recepção de sinais (Fixlink) de um canal de comunicação satelital aderente ao protocolo DVB-S2X, usando aritmética de ponto fixo e linguagem Python, o qual servirá como modelo de referência para soluções em *hardware*. Utilizou-se como referência um simulador já existente em ponto flutuante (Satlink), *scripts* do MATLAB e módulos já desenvolvidos em *hardware*. Foram desenvolvidos modelos em alto nível para do filtro RRC, do sincronizador de quadros (FrameSync) e do corretor fino de frequência (FFC), sendo este último implementado em FPGA no kit de desenvolvimento da Xilinx Zedboard. O modelo do filtro RRC foi avaliado usando quadros modulados em QPSK, 8PSK e 16APSK com resolução de 12 bits, sendo 9 bits para parte fracionaria. Com essa resolução obteve-se um valor de MSE na casa de 10^{-8} para a parte real dos símbolos e um erro de 10^{-6} para a parte imaginária em relação a uma implementação em ponto flutuante, e para implementação em VHDL obteve-se um erro MSE na casa de 10^{-6}. O modelo do FrameSync foi avaliado comparando o desempenho com o simulador Satlink, variando o valor Eb/N0 de -2 a 10 dB. Ambos os simuladores apresentaram o mesmo desempenho a partir do valor de Eb/N0 = 2dB. Foi feito um teste de estresse no *FrameSync* para induzir falhas ao adicionar quadros falsos dentro de um grupo de 10 quadros verdadeiros, a falha ocorre somente quando dois picos falsos da correlação apresentam o distanciamento igual a um quadro verdadeiro. O FFC já possuía um módulo de estimação de frequência implementado em alto nível e em VHDL, porém não atendiam os requisitos da norma, sendo necessário alterar a resolução para 14 bits com 10 bits de parte fracionaria. Após a alteração o estimador passou a atender os requisitos do projeto obtendo um erro RMS de $3.09x10^{-5}$, e na simulação comportamental em VHDL ficou abaixo do valor de $5.20e^{-5}$ exigido. A implementação deste módulo tem uma latência de 20,22 ms enquanto o sub módulo corretor teve uma latência de 75 ns e *throughput* de um ciclo de relógio (100 MOPS). A implementação do *Timming Recovery* em *software* foi comparada com a implementação em *hardware* e foi descoberto a diferença arquitetural. O corretor grosseiro de frequência (CFC) não apresentava o módulo de correção dos símbolos por isso teve que ser implementado no Fixlink com a mesma abordagem usada no FFC. Durante a integração foi descoberta a necessidade da sobre amostragem na entrada do CFC por isso o filtro RRC foi deslocado para após o CFC. Finalmente, na integração do corretor de fase (PC) foi encontrada a ausência do algoritmo de *Unwrapping* que limita a rotação de fase entre π e $-\pi$. Após implementar esse algoritmo no Fixlink, os testes de integração até o PC foram satisfatórios sendo possível observar a correção nas constelações de saída do Fixlink de acordo com o esperado nos cenários de Eb/N0 de 80 dB e 2 dB.

Palavras-chaves: DVB-S2X. DVB-S2. FPGA. Sincronização de Quadros. Filtro RRC.

Correção Fina de Frequência. Simulador Python. Correção de Frequência. Sincronização de Símbolos.

Abstract

This work presents the development of a Python fixed point simulator (Fixlink) that will serve as a reference model for applications developed in hardware, for the implementation of a radio defined by software adhering to the DVB-S2X protocol. For this, was used as a reference an existing floating point simulator (Satlink), MATLAB scripts and modules already developed in hardware to create some of the modules present in the signal processing chain. With this, the development of high-level models for three modules was done for: RRC filter, frame synchronizer (FrameSync) and the fine frequency corrector (FFC), the last one being implemented in FPGA in Xilinx Zedboard's development kit. As for the high-level models, the RRC filter was made from the implementation in MATLAB where was used modulated frames in QPSK, 8PSK and 16APSK with a resolution of 12 bits being 9 bits for the fractional part. With this resolution was obtained an MSE value in the place of e^{-8} for the real part of the symbols and an error of e^{-6} for the imaginary part in relation to a floating point implementation and for implementation in VHDL we obtained an MSE error in the place of e^{-6}. To validate the FrameSync implementation, its testing was done by comparing the performance with the Satlink simulator by varying the Eb/N0 value from -2 to 10 dB. In this test, the simulators showed the same performance from the value of Eb/N0 = 2dB. A stress test was also performed on FrameSync to induce its failure by adding false frames within a group of 10 true frames, the failure only occurs when two false correlation peaks have the spacing equal to a true frame. The FFC already had its frequency estimation module implemented both in high level and in VHDL, but it did not meet the project requirements, being necessary to change the resolution to 14 bits with 10 bits for the fractional part. After the change, the estimator started to meet the project requirements, obtaining an RMS error of $3.0897e^{-5}$ in the behavioral simulation in VHDL, which was below the value of $5.20e^{-5}$ required. Finally, the implementation of the FFC Estimator module had a latency of 20.22 ms while the FFC Corretor sub module had a latency of 75ns with throughput of one clock cycle. The implementation of Timming Recovery in software was compared to the implementation in hardware and it was found that the implemented architectures are different. The coarse frequency corrector (CFC) did not have the symbol correction module so it had to be implemented in Fixlink with the same approach used in FFC. During the integration, the need for oversampling at the CFC input was discovered, so the RRC filter was moved to after the CFC. Finally, in the integration of the phase corrector (PC) it was found the absence of the algorithm of Unwrapping that limits the phase rotation between π and $-\pi$. After implementing this algorithm in Fixlink, the integration tests up to the PC were satisfactory and it was possible to observe correction in the Fixlink output constellations as expected in the 80 dB and 2 dB Eb/N0 scenarios.

Key-words: DVB-S2x. DVB-S2. FPGA. Frame synchronization. RRC filter. Fine Frequency Corrector. Python simulator. Frequency Correction. Symbol Synchronization.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

GSM	*Global System for Mobile communications*
RF	Rádio Frequência
FPGA	*Field-programmable gate array*
ASIC	*Application Specific Integrated Circuits*
DVB-S2x	*Digital Video Broadcasting - Satellite Second Generation Extension*
DVB-RCS2	*Digital Video Broadcasting - Second Generation Return Channel Satellite*
RRC	*Root Raised Cossine*
CFC	*Course Frequency Corrector*
FFC	*Fine Frequency Corrector*
PC	*Phase corrector*
AGC	*Automatic Gain Control*
CPS	Cadeia de Processamento de Sinais
MSE	*Mean Squared Error*
VHDL	*VHSIC Hardware Description Language*
PS	*Processing System*
PL	*Programmable Logic*
DVB-S2	*Digital Video Broadcasting - Satellite - Second Generation*
MODCODs	*Modulation and code rates*
DVB-S	*Digital Video Broadcasting - Satellite*
SOF	*Start of Frame*
PLSC	*Physical Layer Signaling Code*
LDPC	*Low density parity check code*
IDE	*Integrated Development Environment*

ROM	*Read Only Memory*
ILA	*Integrated Logic Analyzer*
QPSK	*Quadrature Phase Shift Keying*
8PSK	*8 Phase Shift Keying*
16APSK	16 *Amplitude and Phase Shift Keying*
LUT	*Look Up Table*
FF	*Flip Flop*
DSP	*Digital Signal Processing*
BRAM	*Block Random Access Memory*
RTL	*Register Transfer Level*
AWGN	*Additive White Gaussian Noise*
RMS	*Root Mean Square*
FIFO	*First in First Out*

Lista de símbolos

dB	Decibel
Eb/N0	Razão da energia do símbolo pela energia do ruído
SnR	Relação Sinal-Ruído
N0	Energia do Ruído
ω	Frequência Angular
σ^2	Variância
σ	Desvio Padrão
ζ	Fator de amortecimento
μ	Atraso de Tempo Fracionário
θ	Fase

Sumário

1 Introdução

1.1 Contextualização

O acesso a informação é visto hoje em dia como algo necessário para se viver em sociedade, porém nem todos os locais do mundo são de fácil acesso. Muitas vezes existem cidades que estão dentro de regiões rurais onde há falta de torres de comunicação e com isso o acesso a meios de comunicação como, por exemplo, *Global System for Mobile Communications* (GSM) se torna inviável. Esses fatores somados com à grande extensão territorial de alguns países, como o Brasil, torna a construção de torres de telefonia por todo país inviável economicamente.

Por isso é necessário buscar outros meios de transmissão. Uma das soluções viáveis para esse problema é a utilização de um sistema de comunicação por satélite (COMINETTI; MORELLO, 2000), porém para isso é necessário levar em consideração diversos fatores como, a taxa de transmissão de dados, uso de protocolos de comunicação, circuitos de rádio frequência (RF), etc. Sistemas de comunicação via satélite em geral possuem grau de complexidade alto quanto à implementação do seu receptor principalmente devido a problemas de sincronia, por isso buscam-se alternativas mais robustas para a implementação de tais sistemas.

O presente trabalho faz parte de um projeto de pesquisa que visa o desenvolvimento de um sistema de comunicação satelital baseado em *Field Programmable Gate Arrays* (FPGAs). A adoção dos FPGAs, além de permitir prototipar uma solução dedicada baseada em circuito integrado de aplicação específica (ASIC, do inglês Application Specific Integrated Circuits), também pode servir como solução final, devido à sua alta taxa de transmissão de dados e capacidade de reconfiguração (Monmasson; Cirstea, 2007). No projeto adotou-se a utilização dos protocolos de transmissão de TV digital: *Digital Video Broadcasting - Satellite Second Generation Extension* (DVB-S2X) no *forward link* e *Digital Video Broadcasting - Second Generation Return Channel Satellite* (DVB-RCS2) no *return link* por possuírem métodos de sincronização de complexidade razoável, além de serem protocolos flexíveis quanto a sua implementação (CASINI; GAUDENZI; GINESI, 2004). Dessa forma, o desenvolvimento das arquiteturas foram feitas a partir das normas estabelecidas por esses protocolos.

Uma restrição do projeto em tela é que os terminais de comunicação são móveis, fato que impõe desafios adicionais dado que a onda eletromagnética transmitida é degradada com o efeito doppler, e isso causa erros de sincronia os quais precisam ser corrigidos para ter sucesso na recuperação do sinal.

A arquitetura da cadeia de processamento de sinais (CPS) está presente na figura 1. A CPS é a responsável por corrigir os erros de sincronia na transmissão, ela é composta pelos seguintes módulos: filtro raiz de cosseno levantado (RRC, do inglês *Root Raised Cossine*), Sincronizador de símbolos (*Timming recovery*), Sincronizador de quadros (*Frame Syncronization*), Corretor de frequência grosseiro (CFC, do inglês *Course Frequency corrector*), corretor fino de frequência (FFC, do inglês *Fine Frequency corrector*), corretor de fase (PC, do inglês *Phase corrector*), controle automático de ganho (AGC, do inglês *Automatic Gain Control*) e o estimador de SNR (SNR *Estimation*).

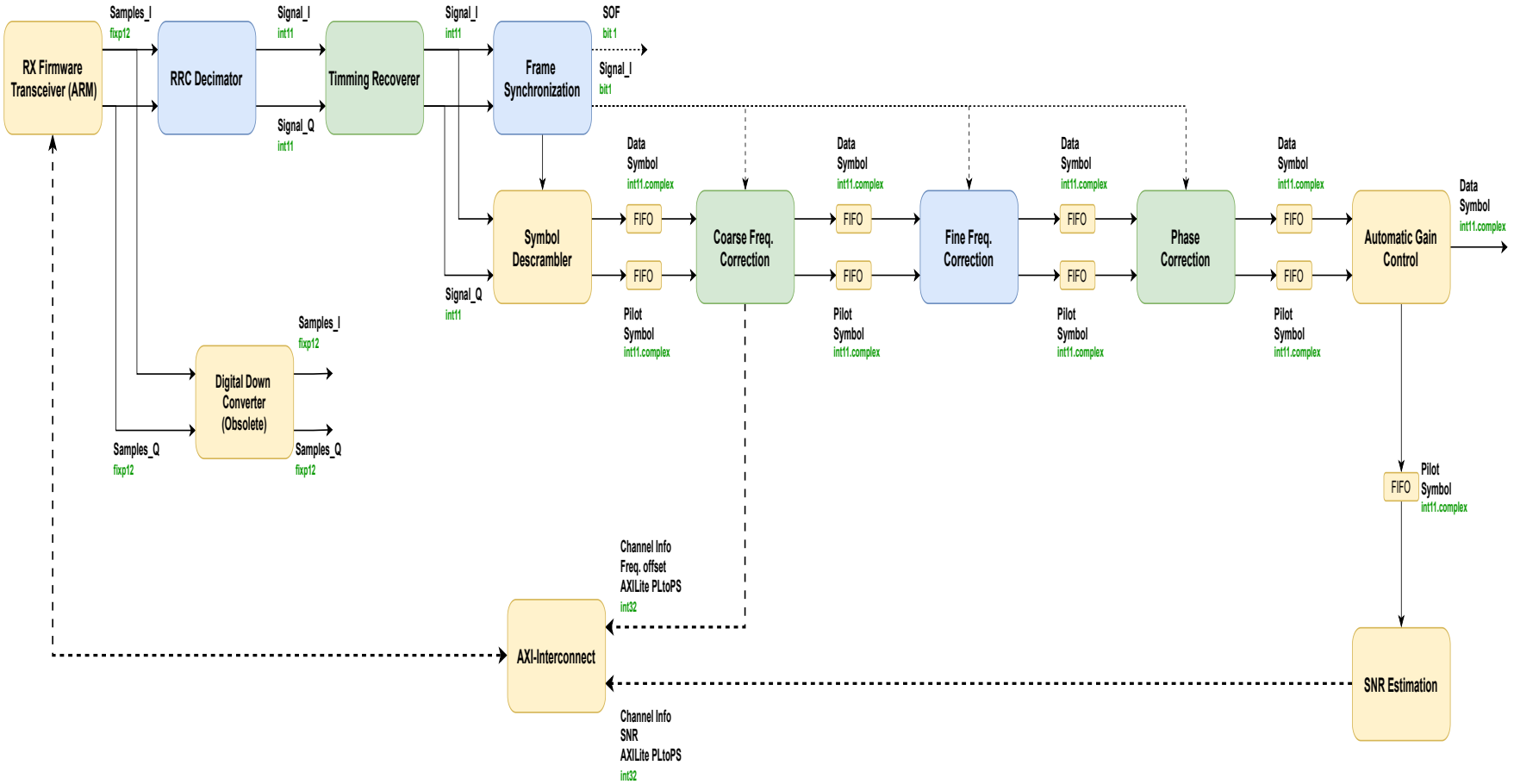

Figura 1 – Arquitetura da cadeia de Processamento de Sinais DVB-S2X. Em verde estão destacados os módulos que compõem o simulador Fixlink desenvolvidos por outras pessoas. Em azul estão os módulos cujos modelos em alto nível foram desenvolvidos pelo o autor deste documento.Sendo o bloco Fine Freq. Correction (FFC) o alvo da implementação em *hardware* deste trabalho.

1.2 Justificativa

Os módulos que integram a CPS possuem um grau de complexidade matemático alto para serem implementados diretamente em FPGAs. Por isso viu-se a necessidade do desenvolvimento de um ambiente de simulação de enlace em *software*, a fim de verificar a o desempenho e confiabilidade dos algoritmos propostas assim como da sua respectiva implementação em *hardware*. O projeto de pesquisa previu o desenvolvimento de um simulador de enlace (cujo nome é Satlink) para avaliar o desempenho do enlace associado a um simulador sistêmico da rede satelital. Entretanto, notou-se que o Satlink não era capaz de servir como modelo referência para a CPS do receptor RF, pois a complexidade matemática dos módulos implicou na dificuldade em realizar uma comparação fiel com as arquiteturas implementadas em FPGAs, dado que a implementação do simulador foi realizada em alto nível de abstração, além de estar baseado em aritmética de ponto flutuante, enquanto o desenvolvimento em hardware deveria ser feito em aritmética de ponto fixo.

Além disso, é importante salientar que, como o simulador Satlink não foi projetado para servir de modelo de referência para a implementação em *hardware*, ele não é capaz de realizar a geração vetores binários para testes que permita a comparação ponto a ponto entre o modelo de referência e a implementação em hardware. Outro fator que justifica o desenvolvimento de um simulador que sirva como referência para a implementação em hardware é a falta de flexibilidade durante a simulação. Em particular no Satlink não é possível realizar uma simulação somente da CPS e nem de um sub conjunto de módulos selecionados pelo usuário dentro da mesma cadeia.

1.3 Estado da Arte

O algoritmo usado no desenvolvimento do FFC, detalhado na sção 2.3.4, foi implementado em FPGA por (SILVA et al., 2015). O algoritmo foi implementado por meio da máquina de estados ilustrada na figura 2, essa máquina de estados foi utilizada como referência na implementação em *hardware* do FFC deste documento e a descrição da cada estado é feita no capitulo 3.

A implementação foi feita no kit de desenvolvimento Stratix IV GX da Altera e com a seguinte utilização de recursos: 3559 LUTs, 3851 registradores e 13 DPS (SILVA et al., 2015). Entretanto, no artigo publicado só foi descrito a implementação do estimador do FFC, não foi citada a técnica usada para rotacionar os símbolos. Por isso essa utilização de recursos não é para todo o FFC.

No teste feito em (SILVA et al., 2015) teve com desvio de frequência normalizada de 0,004 com Eb/N0 variando de -2 a 7 dB. Nesse teste o autor quis verificar se sua

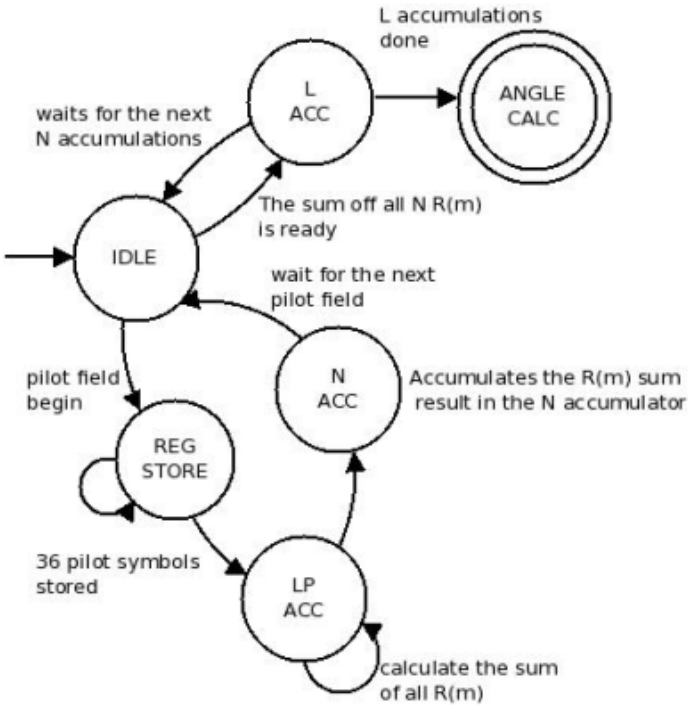

Figura 2 – Máquina de Estados para o estimador do FFC. Retirado de (SILVA et al., 2015)

.

implementação era capaz deixar a frequência residual do FFC a baixo de $6x10^{-5}$ que é o exigido pela norma (ETSI, 2014b). Na figura 3 está o resultado obtido por ele e nota-se a eficiência de sua implementação tendo os valores residuais de frequência em todos cenários abaixo do valor máximo exigido pela norma.

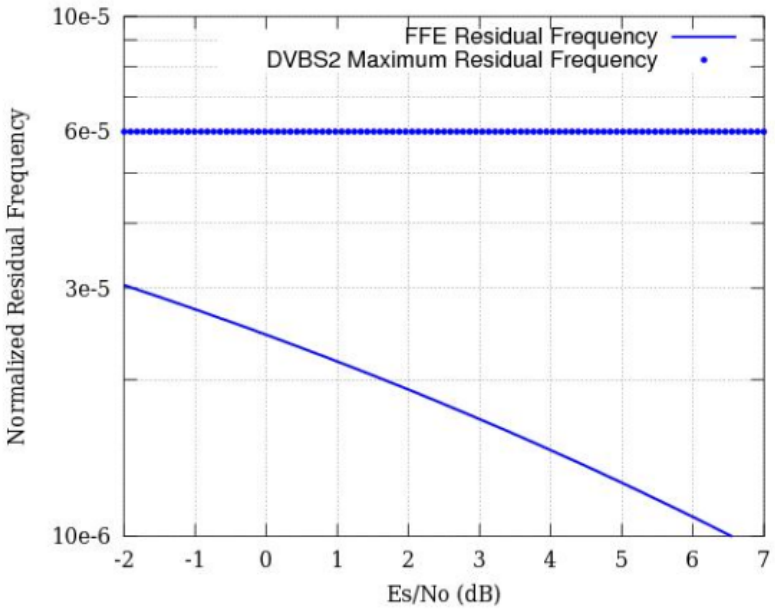

Figura 3 – Resultados obtidos para o estimador de frequência do FFC. Retirado de (SILVA et al., 2015)

.

1.4 Objetivos

1.4.1 Objetivo Geral

Desenvolver um simulador da CPS para o receptor DVB-S2x em ponto fixo (daqui em diante chamado de Fixlink) a fim servir como modelo de referência para o sistema de enlace satelital implementado em FPGA.

1.4.2 Objetivos especificos

1.4.2.1 Simulador Fixlink

- Desenvolvimento do simulador em Python com base nos modelos já existentes para a CPS.
- Integração de todos os módulos presentes na CPS com possibilidade da visualização dos resultados da simulação ponto a ponto ou dos módulos individualmente.
- Implementação da função de geração de vetores binários para testes em *hardware*.
- Possibilitar a comparação bit a bit das soluções em *hardware* ou, quando não for possível a comparação, estimar o erro quadrático médio (MSE, do inglês *Mean Squared Error*) entre o simulador e a solução em hardware.

1.4.2.2 Implementação em Hardware do FFC

- Adaptar o estimador de desvio de frequência em *VHSIC Hardware Description Language* (VHDL) para atender a norma do protocolo DVB-S2X.
- Desenvolvimento e implementação do corretor FFC em VHDL, utilizando vetores de teste do simulador em ponto fixo.
- Implementar a solução em FPGAs e validar o módulo com auxilio do simulador em ponto fixo.

1.5 Organização do Documento

O trabalho está organizado em cinco capítulos sendo eles os seguintes. O primeiro capitulo é a introdução com uma breve contextualização do problema e os principais objetivos do projeto, o segundo capitulo é mostrado a base teórica do problema e uma maior contextualização dos módulos da CPS, o terceiro capítulo apresenta a metodologia de trabalho assim como as ferramentas e *softwares* utilizados. No capítulo quatro são apresentados os resultados obtidos e a descrição dos testes realizados e por fim o último capítulo descreve as conclusões de todo o trabalho desenvolvido.

2 Fundamentação teórica

2.1 Distorções do Canal

Antes de iniciar a discussão sobre as técnicas usadas como referência para o desenvolvimento do Fixlink é necessário uma breve contextualização sobre as distorções que afetam o sinal durante o enlace.

A depender do canal, o sinal pode sofrer varias distorções até a sua chegada no seu destino final. Sendo as principais distorções a atenuação do sinal, que cresce de acordo com o tamanho do canal e as distorções de fase causadas por fenômenos físicos como o efeito dos múltiplos caminhos e o efeito *Doppler* (LATHI; DING, 2010). Sendo o efeito *Doppler* a principal fonte de ruído de fase e frequência a ser corrigido pelo Fixlink.

A distorção causada pelo efeito de múltiplos caminhos ocorre quando o sinal chega no receptor por dois ou mais caminhos, isso causa um efeito de atraso no sinal recebido tendo no receptor múltiplas reflexões do sinal (LATHI; DING, 2010). A principal causa desse efeito, em canais sem fio, é a presença de obstáculos como, por exemplo, prédios e morros que causam o efeito de difração da onda eletromagnética transmitida (LATHI; DING, 2010).

Quanto ao efeito *Doppler*, ele ocorre quando o receptor está em movimento, como é o caso do projeto que envolve o Fixlink. Ao considerar um sinal transmitido em banda base $m(t)cos(\omega_c t)$, o desvio causado pelo efeito *Doppler* aparece como um desvio de fase ω_d. Sendo assim o sinal no receptor passa a ser $m(t)cos(\omega_c + \omega_d)t$ com $\omega_d = \frac{v_d}{c}\omega_c$, onde v_d é a velocidade de deslocamento do receptor, que no caso do projeto é a velocidade do terminal móvel, e c é a velocidade da luz (LATHI; DING, 2010).

Adicionalmente, devido a aleatoriedade do ruído presente nos sistemas comunicações é necessário ter um modelo de canal que seja capaz de simular um cenário real de ruído. Por isso comumente utiliza-se um canal AWGN (*Additive White Gaussian Noise*) como modelo(LATHI; DING, 2010). Nesse canal um processo aleatório Gausssiano de média zero é utilizado para caracterizar o ruído aleatório causado, principalmente, pelo ruído térmico dos componentes eletrônicos (SKLAR; HARRIS, 2020).

Tal modelagem usa a função densidade de probabilidade do processo Gaussiano, presente na equação 2.1, para modelar o ruído a cada amostra n de forma aditiva ao sinal de entrada, onde σ^2 é a variância do ruído da amostra n (SKLAR; HARRIS, 2020). Além disso considera-se que o ruído térmico ocorre de forma igual em todo espectro de frequência do sinal caracterizando-o como um ruído branco (SKLAR; HARRIS, 2020).

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2 \right] \tag{2.1}$$

Graças a isso tem-se que o ruído branco Gaussiano possui valor de densidade espectral de energia constante , com o valor de $N_0/2$, em toda sua banda como pode ser visto na figura 4 (SKLAR; HARRIS, 2020). Por mais que em um cenário real o ruído não seja igual em todo seu espectro de frequência, a modelagem utilizada no canal AWGN não deixa de ser uma boa forma de caracterizar o ruído presente em um cenário real (SKLAR; HARRIS, 2020). Por isso no Fixlink, em quase todos os testes, o sinal transmitido passa por um canal AWGN.

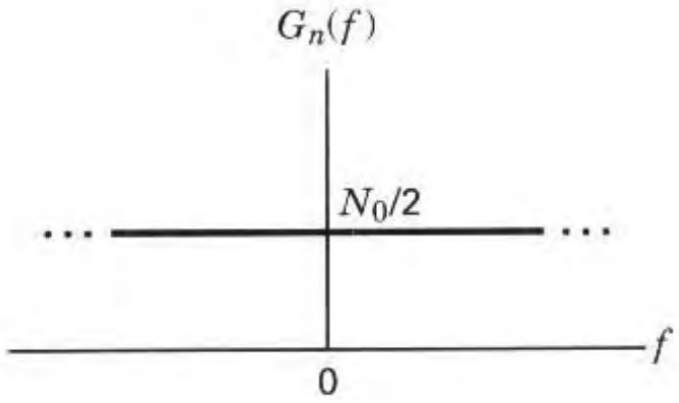

Figura 4 – Função densidade espectral de potênciao do ruído branco Gaussiano. Retirado de (SKLAR; HARRIS, 2020)

.

2.2 Protocolo DVB-S2X

O protocolo DVB-S2X é uma extensão do protocolo *Digital Video Broadcasting - Satellite - Second Generation* (DVB-S2), sendo a principal diferença a adição de etapas de filtragem mais precisas e a adoção de esquemas de modulação e taxas de códigos (MODCODs, do inglês *Modulation and code rates*) mais refinados (ETSI, 2014a).

O sistema (DVB-S2) propõe três funcionalidades superiores em relação a sua versão anterior, o protocolo *Digital Video Broadcasting - Satellite* (DVB-S), sendo elas: melhor performance de transmissão, maior flexibilidade, por ser capaz de operar sob diferentes características dos transponders nos satélites existentes no mercado, e por fim possui receptor de complexidade razoável (Morello; Mignone, 2006).

O sinal recebido por esse sistema é do tipo FECFRAME ilustrado, na figura 5, o qual é estruturado em *slots*. O primeiro *slot* forma o cabeçalho (PLHEADER) do quadro e é formado por 90 símbolos. Sendo os 26 primeiros símbolos o inicio do quadro (SOF, do

inglês *Start of Frame*) e os 64 símbolos restantes o *Physical Layer Signaling Code*(PLSC). O PLSC é responsável por armazenar informações de configuração de cada quadro como, por exemplo, o tipo de modulação usada nos dados e a taxa de código. O cabeçalho é independente do tipo de modulação realizada na codificação de *Low density parity check code* (LDPC), por isso ele é sempre modulado em $\pi/2$-BPSK (SUN; JIANG; LEE, 2004). Os dados presentes no quadro são guardados em *slots* com 90 símbolos e caso seja adicionado os blocos de pilotos, eles serão adicionados a cada 16 *slots* de dados em *slots* com 36 símbolos de pilotos (Morello; Mignone, 2006).

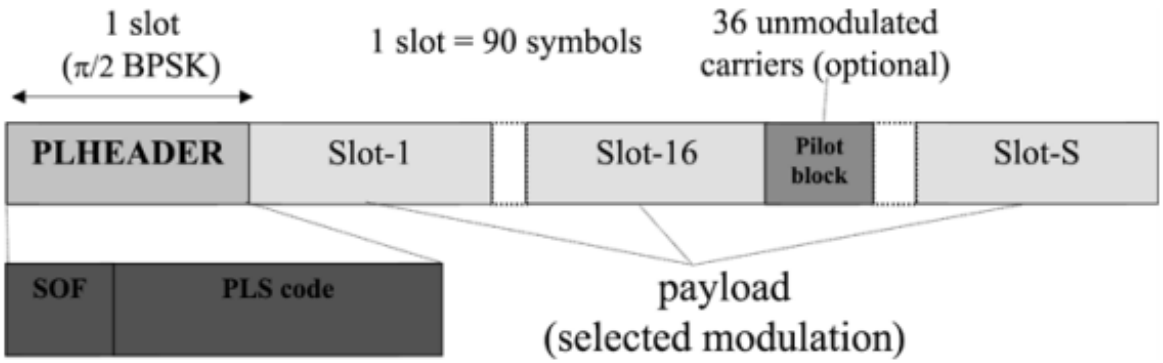

Figura 5 – Estrutura do FECFRAME. Retirado de (Morello; Mignone, 2006)

.

2.3 Sincronização dos símbolos

A sincronização dos símbolos é um do maiores desafios do sistema DVB-S2, pelo fato de que o sistema trabalha sobre baixos valores de SnR e há uma grande variação entre os quadros de acordo com o esquema de modulação utilizado. Isso implica na dificuldade em manter a sincronização entre eles, na figura 6 está presente um exemplo de diagrama de blocos usado para realizar a sincronização dos símbolos.

A correção apresentada na figura 6 é feita da seguinte maneira: os símbolos passam primeiro pela sincronização de tempo(*Timming Recovery*) onde é feita a sincronização da amostragem do sinal (blocos *INTERPOLATOR* e *SYMBOL CLK RECOVERY: GARDNER*); em seguida o sinal passa por um filtro onde também é feita a sub amostragem do sinal(bloco *MATCHED FILTER + DWNSAMPL*), nessa etapa é feito ajuste da formatação do pulso recebido.

Depois é iniciada a sincronização dos quadros, que é o processo de localização dos SOFs nos quadros recebidos (bloco *FRAME SYNCH*) e também é feita a separação dos blocos de dados e de pilotos (blocos de *DEMUX*); em seguida é feita a correção grosseira de frequência(blocos *INTEGRATOR + LOOK UP TABLE*, *MODFIED L&R FREQUENCY ESTIMATOR*, *COARSE FED DELAY&MUL*, 2^{nd} *ORDER LOOP FILTER* e *NCO*),

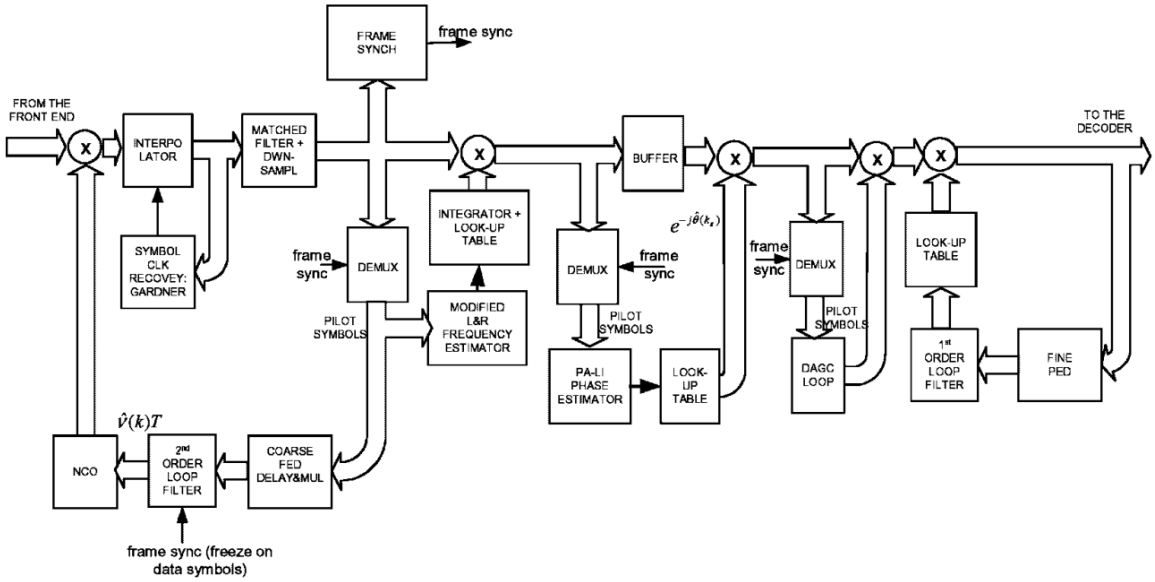

Figura 6 – Diagrama de blocos do sincronizador de símbolos DVB-S2. Retirado de (CASINI; GAUDENZI; GINESI, 2004)

.

correção de fase (blocos *PA-LI PHASE ESTIMATOR* e *LOOK-UP TABLE*) e por fim a correção fina de frequência(blocos *LOOK-UP TABLE*, 1^{st} *ORDER LOOP FILTER* e *FINE PED*) (CASINI; GAUDENZI; GINESI, 2004).

O diagrama citado anteriormente é um bom modelo para a implementação da cadeia CPS, porém existem outros bons modelos como, por exemplo, a implementação feita por (Albertazzi et al., 2005) e por (LIMA et al., 2014), sendo a última uma implementação em FPGA.

Na figura 7 está o diagrama de blocos proposto por (Albertazzi et al., 2005). A diferença mais notável é o uso do filtro (bloco *Matched filter*) antes do *Timming Recovery*(bloco *Timing recovery*) e a separação do módulo que retira a sobre amostragem do filtro (bloco *Symbol sampling*). Sendo assim nota-se que o posicionamento do filtro para antes da etapa do *Timming Recovery* não é prejudicial para o sistema desde que a sub amostragem continue sendo feita após ele.

Quanto a implementação de (LIMA et al., 2014), presente na figura 8, segue uma abordagem diferente no que se diz respeito ao posicionamento dos corretores de frequência. O FFC (blocos *FINE FREQ ESTIMATOR*, *PHASE ACC* e *CORDIC*) passou a estar antes da aplicação do PC (blocos *COARSE PHASE ESTIMATOR*, *INTERPOLATOR*, *UNWRAPPER* e *CORDIC*), além disso foi adicionado, no fim cadeia de sincronização, o AGC(bloco *DIGITAL AGC*) e o estimador de SnR (blocos *SNR ESTIMATOR* e *NOISE POWER CALC*). Essa implementação foi a principal referência utilizada no desenvolvimento do Fixlink e do projeto, além de que foi a base inicial para selecionar os algoritmos que seriam usados em cada módulo da CPS descritos no próximo tópico.

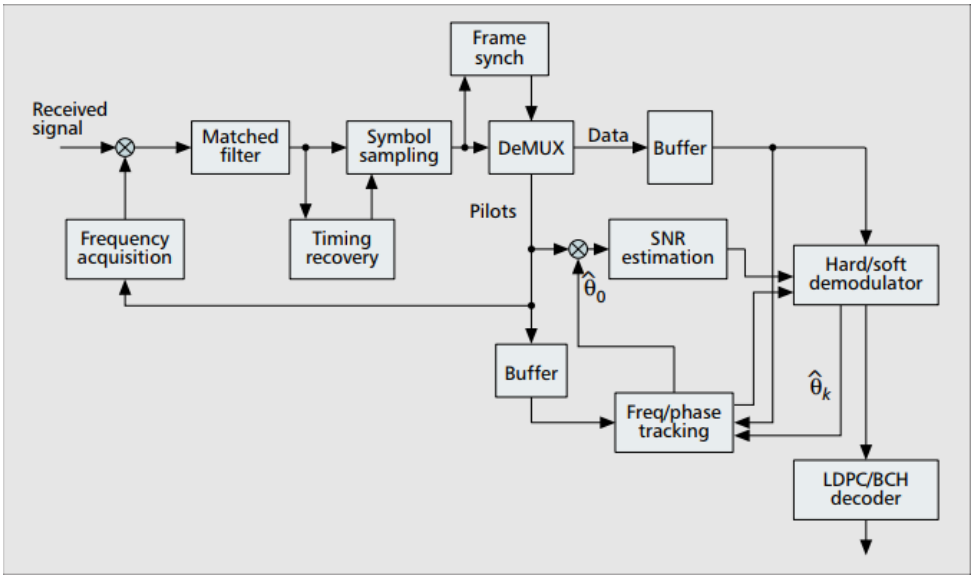

Figura 7 – Diagrama de blocos do sincronizador de símbolos DVB-S2 proposto por (Albertazzi et al., 2005). Retirado de (Albertazzi et al., 2005)

.

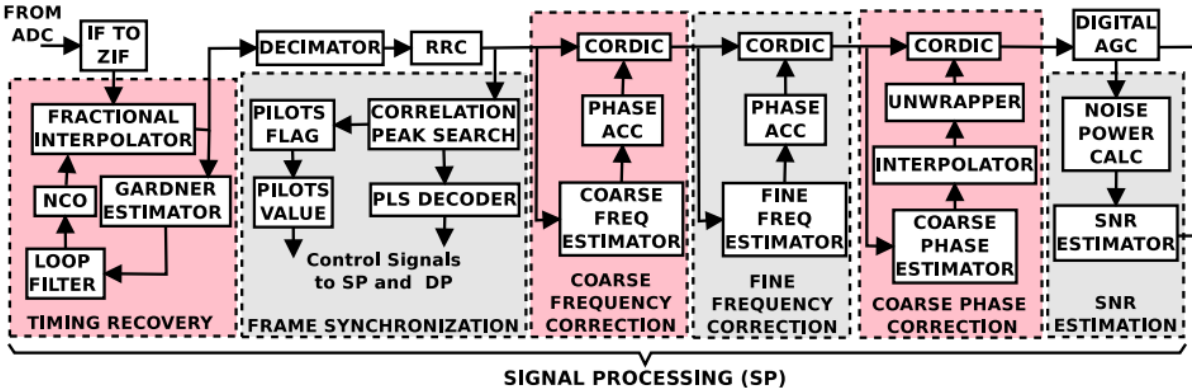

Figura 8 – Diagrama de blocos do sincronizador de símbolos DVB-S2 proposto por (LIMA et al., 2014). Retirado de (LIMA et al., 2014)
com modificações.

2.3.1 Filtro RRC

Dentro do cenário da comunicação digital é de extrema importância de se ter uma boa formatação de pulso, esse procedimento é feito por meio de filtros com finalidade de facilitar o processo de recuperação de sinal. Para isso é necessário que ambos os filtros, tanto do transmissor quanto do receptor, estejam bem sincronizados e ambos devem atendam bem o critério de Nyquist para evitar a sobreposição dos símbolos transmitidos(Mukumoto; Wada, 2014). Comumente utiliza-se o filtro RRC para a formatação do pulso transmitido, sua resposta em frequência descrita nas equações 2.2, 2.3 e 2.4, onde r é o fator de *roll-off* e f_N é a frequência de Nyquist.

$$H(f) = 1, \qquad para \qquad |f| < f_N(1-r) \tag{2.2}$$

$$H(f) = \sqrt{\frac{1}{2} + \frac{1}{2}\sin\left(\frac{\pi(f_N - |f|)}{2rf_N}\right)}, \qquad para \qquad f_N(1-r) \leq |f| \leq f_N(1+r) \tag{2.3}$$

$$H(f) = 0, \qquad para \qquad |f| < f_N(1+r) \tag{2.4}$$

2.3.2 *Timming Recovery*

A sincronização realizada pelo *Timming Recovery* é feita com o objetivo de sincronizar o *clock* da amostragem usado na transmissão com o da recepção (RICE, 2009). Para realizar a sincronização é necessário realizar o calculo do erro de tempo de amostragem. Em (GARDNER, 1986) é descrito um algoritmo capaz de detectar o erro de tempo, esse algoritmo está presente na equação 2.5.

$$\mu_t(k) = y_I(k-1/2)[y_I(k) - y_I(k-1)] + y_Q(k-1/2)[y_Q(k) - y_Q(k-1)] \tag{2.5}$$

O algoritmo consiste basicamente em calcular a diferença entre a amostra atual $y(k)$ com a anterior $y(k-1)$ e multiplica pelo ponto médio das duas amostras $y(k-1/2)$, esse cálculo é feito tanto para os símbolos em fase (y_I) quanto para os em quadratura (y_Q) que são somado no fim do algoritmo (GARDNER, 1986).

2.3.3 Sincronizador de Quadros (FrameSync)

A sincronização de quadros (em inglês, *Frame Syncronization*) é o procedimento de detecção do inicio do quadro, para isso utiliza-se das propriedades de autocorrelação do PLFRAME. Pelo fato de que o cabeçalho é formado por uma sequencia de 26 símbolos binários codificados em $\pi/2$, tal sequencia forma SOF (Albertazzi et al., 2005).

É importante ressaltar que não é viável aplicar a autocorrelação somente no SOF, pois seu tamanho é muito pequeno o que implica um desempenho insatisfatório para baixos valores de SNRs(Morello; Mignone, 2006). Uma das formas de contornar isso é a utilização também dos símbolos do PLSC, que também são codificados em $\pi/2$. Os cálculos das correlações diferenciais são feitos com base nas equações 2.6 e 2.7, onde $z(k)$ é o k-ésimo símbolo recebido e o tap_{SOF} é uma sequencia complexa presente na equação 2.8 e o tap_{PLSC} é o vetor complexo na equação 2.9 (ETSI, 2014b). É importante ressalta que o tap_{PLSC} deve ter seus valores invertidos caso haja a presença de blocos pilotos no quadro (ETSI, 2014b).

$$C_{SOF} = \sum_{k=0}^{N_{SOF}-1} z(k)z^*(k+1) \times tap_{SOF}(k) \tag{2.6}$$

$$C_{PLSC} = \sum_{k=0}^{N_{PLSC2}/2-1} z(2k)z^*(2k+1) \times tap_{PLSC}(2k) \tag{2.7}$$

$$tap_{SOF} = [-j,-j,-j,-j,j,j,j,j,-j,j,j,j,-j,j,j,-j,-j,j,-j,-j,j,-j,j,j,-j] \tag{2.8}$$

$$\begin{aligned} tap_{PLSC} = [&-j,j,j,-j,-j,-j,j,-j,-j,j,j,j,j,j,\\ &-j,-j,-j,-j,j,j,-j,j,j,-j,j,-j,j,j,-j,-j] \end{aligned} \tag{2.9}$$

Após a realização das correlações diferenciais é feito um janelamento para realizar a detecção dos picos de correlação do SOF e o PLSC. A implementação desse janelamento é o maior desafio da sincronização dos quadros, essa técnica funciona da seguinte maneira: os símbolos são acumulados dentro de um intervalo de tempo e é feito uma busca por picos de correlação do SOF e do PLSC. Após encontrar esses picos verifica-se se o intervalo tempo entre os picos encontrados são coerentes com o intervalo dos símbolos de um quadro completo (QING et al., 2008).

A figura 9 ilustra um exemplo de como é realizado tal procedimento, na imagem, L é o tamanho do janelamento usado e são encontrados, nesse exemplo, 3 candidatos. Por isso é feita uma medição da distância entre esses candidatos a SOF. Logo se a distância entre eles for de um quadro, considera-se que os picos são verdeiros e verifica-se sua posição de ocorrência que é equivalente a posição do SOF daquele quadro. Com isso é feito localização dos quadros e depois são separados os blocos de dados e pilotos para serem usados nos corretores de frequência.

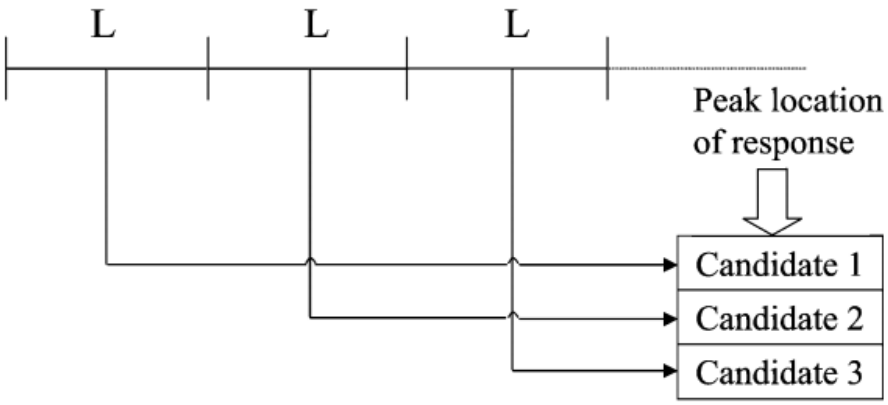

Figura 9 – Exemplo da detecção dos picos de correlação pelo janelamento. Retirado de (Morello; Mignone, 2006)

.

2.3.4 Corretor Grosso de Frequência (CFC)

A primeira etapa de correção de frequência é a "grosseira", o primeiro algoritmo usado nessa etapa é o detector de erro de frequência (FED, do inglês *Frequency Error Detector*), presente na equação 2.10(MENGALI, 2013).

$$e(k) = Im\{z^p[k]c^*[k-2]z^{*p}[k-2]c[k]\} \tag{2.10}$$

O algoritmo do FED consiste em utilizar a parte imaginaria do resultado da multiplicação dos 4 seguintes termos. O k-éismo símbolo piloto recebido ($z^p[k]$), o símbolo piloto de referência conjugado ($c^*[k]$), o símbolo conjugado de 2 amostras atrasadas do k-ésimo símbolo piloto recebido ($z^{*p}[k-2]$) e o símbolo de piloto de referência ($c[k]$). Ou seja, nota-se que esse algoritmo precisa armazenar três amostras dos símbolos de entrada.

Após o cálculo do erro, o resultado deve passar por um filtro PI para retirar parte do ruído de frequência residual. Para isso o filtro utiliza duas constantes que são calculadas pelas equações 2.11 e 2.12 (RICE, 2009). Sendo B_n a largura banda normalizada do sinal que de acordo com a norma (ETSI, 2014b) deve ser de 10^{-4} e ζ é o fator de amortecimento do filtro.

$$k_1 = \frac{4\zeta B_n}{\zeta + \frac{1}{4\zeta}} \tag{2.11}$$

$$k_2 = \frac{4B_n^2}{(\zeta + \frac{1}{4\zeta})^2} \tag{2.12}$$

Por fim é necessário passar a saída do filtro PI por um filtro de média móvel e sua implementação segue o algoritmo recursivo apresentado na equação 2.13. Onde e é o erro de frequência após a passagem do filtro PI. Um ponto importante é que o CFC deve

entregar um resíduo de frequência normalizada para o FFC na ordem de 10^{-4} (ETSI, 2014b).

$$E_{average} = E_{average} + [e(N-1) - e(0)].\frac{1}{N} \tag{2.13}$$

2.3.5 Corretor Fino de Frequência (FFC)

A segunda etapa de correção de frequência é o ajuste fino do FFC e seu maior desafio está na estimação do desvio de frequência normalizada residual deixada pelo CFC. Dentro da literatura é comum encontrar a aplicação de diversos métodos baseados em algoritmos de *Data-Aided* (Gong et al., 2012), os quais utilizam o calculo de varias correlações e consequentemente eleva o grau de complexidade da implementação do módulo e também aumenta a utilização de recursos.

Usualmente utiliza-se um grande conjunto de símbolos de pilotos a fim de realizar um treinamento do algoritmo para estimar o possível desvio de frequência do sinal recebido. Existem três algoritmos que descrevem como esse treinamento deve ser realizado, são eles: algoritmo L&R, algoritmo de Fitz e o algoritmo M&M (Gong et al., 2012). Foi com base no algoritmo de L&R que foi derivado um método de estimação do desvio de frequência onde é utilizado a autocorrelação dos pilotos baseado em um algoritmo *feedforward* (CASINI; GAUDENZI; GINESI, 2004).

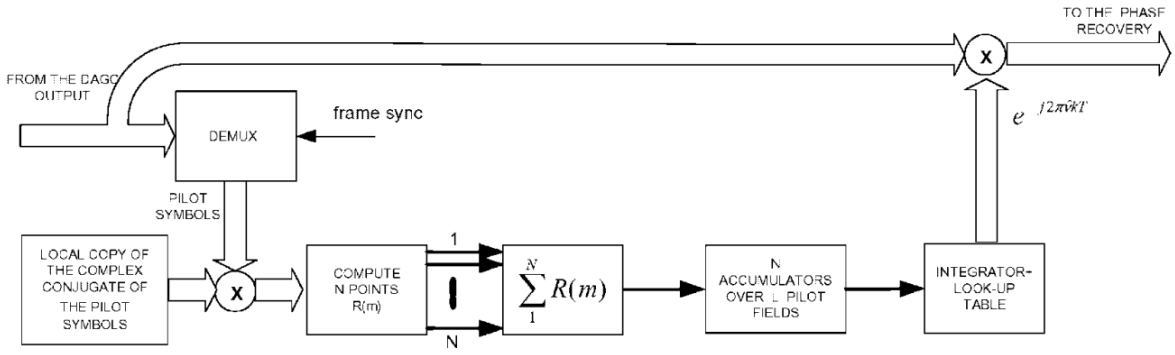

Figura 10 – Diagrama de blocos do estimador de frequência. Retirado de (CASINI; GAUDENZI; GINESI, 2004)

.

O diagrama de blocos que descreve esse algoritmo está presenta na figura 10, ele funciona da seguinte maneira: na primeira etapa é aplicada a equação 2.15 onde é feito o cálculo de $s(k)$, que é a multiplicação entre os símbolos de pilotos recebidos $z^p(k)$ com o conjugado dos símbolos pilotos de referência $c^*(k)$. Na próxima etapa é feito o cálculo presente na equação 2.14, nessa equação realiza-se o cálculo dos pontos de autocorrelação calculados sobre os L_p símbolos de pilotos, onde L_p é o número de pilotos em um bloco, que por padrão são 36 símbolos (ETSI, 2014b).

Por fim é aplicada a equação 2.16 que realiza a acumulação dos valores de autocorrelação sobre sucessivos blocos de pilotos recebidos e calcula-se o argumento dessa acumulação. Tendo assim o valor estimado para o desvio de frequência dos símbolos, um ponto importante sobre esse algoritmo é que para ele convergir são necessários somente 1000 blocos de pilotos (CASINI; GAUDENZI; GINESI, 2004).

$$R_l(m) = \frac{1}{L_p - m} \sum_{k=m}^{L_p-1} s(k)s^*(k-m) \tag{2.14}$$

$$s(k) = z^p(k)c^*(k) \tag{2.15}$$

$$\hat{\nu}T = \frac{1}{\pi(N+1)} arg\left\{ \sum_{l=1}^{L} \sum_{m=1}^{N} R_l(m) \right\} \tag{2.16}$$

2.3.6 Corretor de Fase (PC)

Depois corrigir a frequência resta realizar a correção de fase. O algoritmo usado foi o de Máxima Verossimilhança, mostrado na equação 2.17, sendo recomendado pela norma (ETSI, 2014b). O algoritmo consiste em realizar o acúmulo, de todos os símbolos dentro de um bloco de pilotos de tamanho L, da multiplicação dos símbolos de pilotos de entrada $z^p(k)$ pelo conjugado dos símbolos pilotos de referência $c^*(k)$.

$$\hat{\theta}^{(p)} = arg\left\{ \sum_{k=0}^{L-1} c^*(k)z^p(k) \right\} \tag{2.17}$$

Além desse algoritmo, em (LIMA et al., 2014) o PC foi implementado com o uso de mais dois algoritmos. O primeiro é a interpolação linear que tem o objetivo de estimar o desvio de fase a cada nova entrada de blocos de pilotos. O outro é o algoritmo de *Unwrapping* que serve para fazer o desdobramento do ruído de fase retirando as ambiguidades que estão fora do intervalo de $-\pi$ e π.

3 Aspectos Metodológicos e Ferramentas

3.1 Proposta de Trabalho

O simulador em alto nível (Fixlink) foi implementado na linguagem de programação Python. Para isso utilizou-se tanto o ambiente Linux Ubuntu 16.0 com a IDE de desenvolvimento Pycharm e o Windows 10 com a IDE Spyder com o Python na versão 3.7. Tendo como principais bibliotecas usadas, tanto no desenvolvimento quanto na analise dos resultados, a *numpy*; *scipy* e *matplotlib*. Os códigos foram projetados a partir dos modelos já implementados em *hardware*, ao adaptar a lógica do código VHDL para o alto nível Python; em códigos do MATLAB, os quais foram refeitos em Python, e em códigos do próprio simulador Satlink adaptado para ponto fixo. Durante o desenvolvimento foram feitas comparações entre as saídas do Fixlink com o Satlink e com a implementação em VHDL. A fim de validar a adaptação dos módulos para ponto fixo, foi utilizado como métrica os valores do erro MSE (Mean Squared Error) obtidos a cada etapa de calculo e as constelações de saída dos símbolos.

3.2 Filtro RRC

Para o desenvolvimento do modelo de referência do filtro RRC, foi utilizado o modelo implementado no MATLAB o qual realiza a aplicação do filtro RRC como um filtro FIR. Inicialmente a implementação realiza o cálculo dos coeficientes do filtro RRC com base nas equações 3.1, 3.2, 3.3. Sendo que as duas ultimas equações são responsáveis por tratar a singularidade da equação 3.1 nos pontos $t = 0$ e $t = \pm\frac{1}{4r}$ (JOOST, 2010). O valor de r representa o fator de *roll-off* que de acordo com a norma (ETSI, 2014b) deve ser de 0,2. Tendo os coeficientes do filtro ainda é necessário normalizar a energia do filtro, esse procedimento foi feito ao dividir os coeficientes do filtro por eles mesmos ao quadrado.

$$h(tT_c) = \frac{\sin(\pi t(1-r)) + 4rt\cos(\pi t(1+r))}{\pi t(1-(4rt)^2)} \tag{3.1}$$

$$h(0) = (1-r) + \frac{4r}{\pi} \tag{3.2}$$

$$h\left(\pm\frac{1}{4r}T_c\right) = \frac{r}{\sqrt{2}}\left(\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4r}\right) + \left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4}\right)\right) \tag{3.3}$$

Entretanto, antes da aplicação do filtro sobre o sinal de entrada, os coeficientes devem ser quantizados. A quantização é feita a partir do número de bits da palavra na entrada do filtro da transmissão. Neste caso, o filtro RRC da transmissão está no transceptor, cuja entrada possui um tamanho de 10 bits. Por isso os coeficientes do filtro na recepção também devem ser quantizados em 10 bits.

A aplicação do filtro sobre o sinal é realizada com a função do MATLAB *upfirdn* nessa função o sinal de entrada passa, primeiramente, pelo processo sobre amostragem com a quantidade de amostras por símbolos definida pelo usuário; aplica o filtro FIR e realiza a sub amostragem. A mesma função com mesmo nome e funcionalidade existe dentro da biblioteca *scipy* do Python e foi utilizada para adaptar o código MATLAB para Python. Por fim, o filtro RRC possui o tamanho de sua saída variado de acordo com a necessidade do usuário. Para este trabalho a saída do RRC foi de 12 bits (1 bit de sinal, 2 bits inteiros e 9 bits fracionários), além da inclusão de uma *flag* responsável por criar vetores de teste da entrada e saída do filtro em uma arquivo txt com a resolução escolhida pelo usuário.

3.3 Sincronizador de Quadros (FrameSync)

O código foi desenvolvido com base tanto na implementação em VHDL quanto no modelo em MATLAB existentes. É importante ressaltar que a arquitetura antes do desenvolvimento do Fixlink realizava o calculo das correlações diferenciais para o SOF, PLSC e do primeiro bloco de pilotos. Contudo, esse método utilizava muitos recursos do kit FPGA escolhido para a aplicação, por isso após utilizar o modelo do Satlink para verificar o desempenho do FrameSync sem a correlação dos pilotos foi feita a alteração para o calculo somente das correlações do SOF e do PLSC.

O modelo implementado no Fixlink segue o diagrama de blocos presentes na figura 11, nesse diagrama é realizado a correlação diferencial do SOF e do PLSC, que são somados e se os valores dos picos passarem do limiar de decisão, é feita uma verificação no janelamento avaliando se o intervalo de dois picos são correspondentes ao tamanho esperado de um quadro. Ou seja, o módulo depende de que no mínimo dois quadros passem por ele para que seja possível localizar o SOF dos quadros e os picos que não estão no espaçamento de um quadro são considerados falsas detecções.

Para realizar as correlações diferenciais foi implementado em VHDL uma FIFO de 64 posições tanto para o calculo da correlação do SOF quanto para a do PLSC. A FIFO foi recriada em Python, porém seu tamanho dependia de qual correlação seria executada, ou seja, caso a correlação fosse para o SOF a FIFO teria 26 posições e caso fosse para o PLSC teria 64 posições. Devido a essa diferença de tamanhos entre o SOF e o PLSC os picos da correlação acontecem em momentos distintos o que impossibilita a soma de

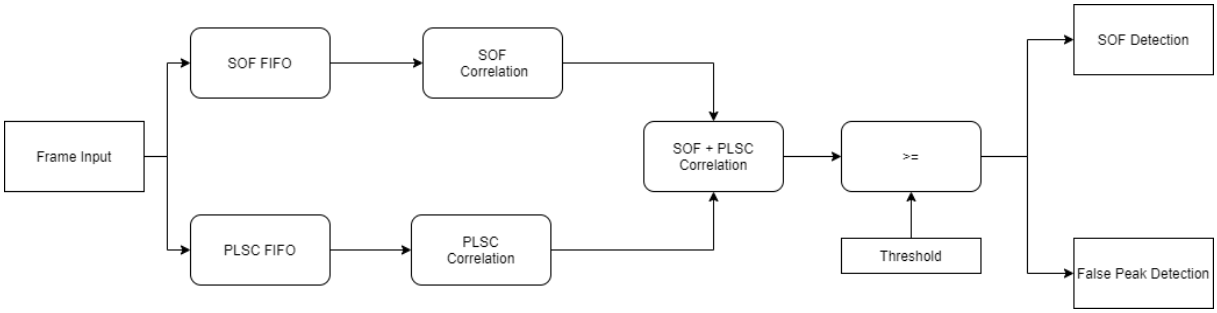

Figura 11 – Diagrama de blocos do modelo em Python ponto fixo para o *frame sync*.

ambos de forma a direta. Por isso o código VHDL apresenta um módulo de atraso para ajustar o instante em que resultado da correlação do SOF deve sair para sincronizar a soma com os picos do PLSC. No Fixlink o atraso é feito com o deslocamento do vetor que armazena a correlação do SOF em 64 posições ao adicionar zeros no inicio do vetor.

Quanto ao janelamento implementado em VHDL ele armazena as posições em que ocorreram os picos acima do limiar em um vetor e calcula a diferença entre todos os picos e retorna as posições em que a diferença foi igual ao tamanho de um quadro. O limite de armazenamento em VHDL é de 8 picos, a implementação do Fixlink seguiu a mesma abordagem utilizando métodos presentes na biblioteca *numpy* para fazer a manipulação e armazenamento dos vetores, porém não há a limitação de 8 picos. A ausência desse limite no FixLink se dá por causa da complexidade da implementação dessa limitação e que a ocorrência de mais 8 picos só é observada em valores muito baixos de Eb/N0 (inferiores a -3 db), que são valores em que o FrameSync tem uma probabilidade de detecção dos quadros muito baixo.

3.4 Corretor Fino de Frequência (FFC)

O FFC é composto por dois sub módulos, o FFC Estimator que calcula o desvio de frequência dos símbolos e o FFC Corrector que executa a correção com base no desvio estimado. O diagrama de blocos na figura 12 mostra como é feita a conexão entre os sub módulos. Basicamente o FFC Estimator utiliza os pilotos para estimar o desvio de frequência e sua saída é direcionada para dois FFC Correctors, o primeiro executa a correção dos blocos de dados e o outro executa a correção nos blocos de pilotos.

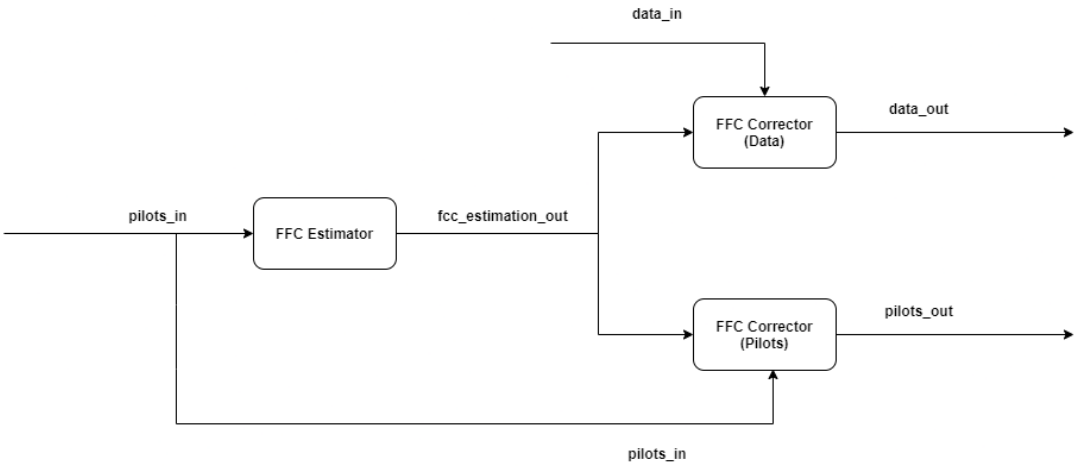

Figura 12 – Diagrama de blocos geral do FFC

.

3.4.1 Sub módulo FFC Estimator

Durante a execução do projeto o FFC já estava em desenvolvimento, por outra pessoa, e o sub módulo FFC Estimator já havia sido parcialmente finalizado tanto no Fixlink quanto em VHDL. Entretanto foram feitas algumas alterações, pelo autor deste documento, para atender os requisitos da norma e para realizar a implementação em *hardware*. A implementação do estimador foi feita com base no algoritmo *feedforward* (CASINI; GAUDENZI; GINESI, 2004).

A implementação das equações desse algoritmo em VHDL foi feita por meio de uma máquina de estados baseada na implementação de (SILVA et al., 2015), presente na figura 2, que faz o controle da entrada dos blocos de pilotos e os cálculos das equações 2.14, 2.15 e 2.16. Na figura 13 está a máquina de estados implementada no FFC Estimator, o primeiro estado IDLE é o ponto inicial e de *reset*, sendo que a máquina permanece nesse estado até a chegada de símbolos pilotos, indicados pelo sinal *pilots_ready_in*, proveniente do módulo CFC.

O próximo estado REG_STORE o qual armazena os símbolos de pilotos em uma FIFO de 36 posições e um contador verifica quantos símbolos de pilotos entraram no

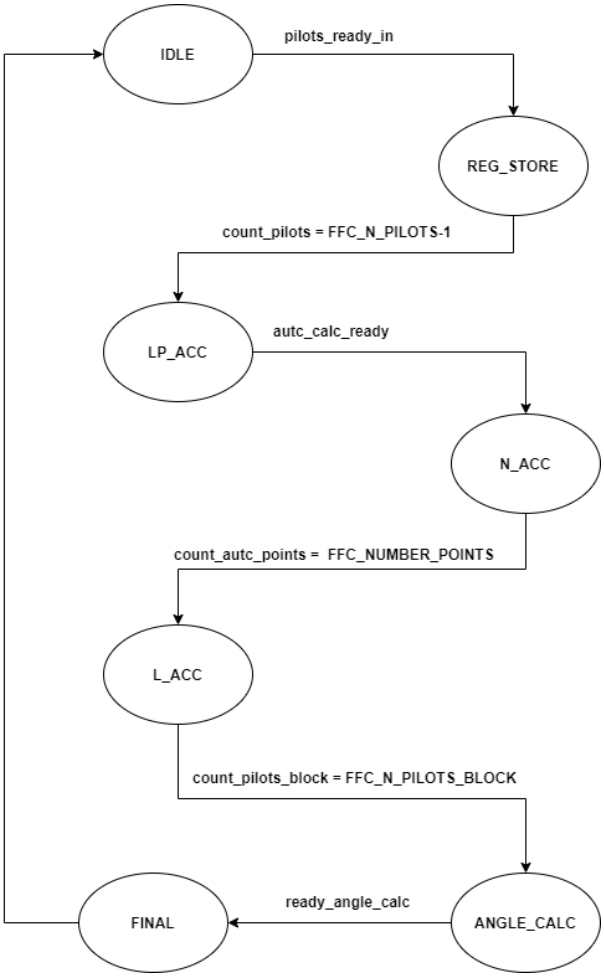

Figura 13 – Máquina de estados principal do FFC Estimator

.

módulo que ao chegar no valor de 36 (valor de *FFC_N_PILOTS*-1) a máquina passa para o próximo estado. No LP_ACC é feita a autocorrelação dos pilotos presente na equação 2.14 e para isso foi utilizado uma segunda máquina de estados, presente na figura 14.

Nessa máquina de estados o estado IDLE é o ponto inicial e de *reset* que tem sua transição quando a FIFO dos blocos de pilotos tem todas suas posições ocupadas, indicado pelo sinal *block_pilots_ready_in*. O estado ACCUMULATOR executa o somatório presente na equação 2.14, sua transição é feita quando o acúmulo é realizado para todos os 36 símbolos. Para isso é utilizado um contador que é incrementado a cada novo acúmulo feito e por fim no estado AVERAGE é feita a multiplicação dos acúmulos pela constante $\frac{1}{L_p - m}$.

Voltando para máquina de estados da figura 13, nos próximos 3 estados é feito o cálculo da equação 2.16. Nos estados N_ACC e L_ACC é feito os acúmulos presentes na equação 2.16. A transição desses estados é feita quando o contador responsável por contar a quantidade de blocos pilotos, indicado pelo sinal *count_pilots_block*, chega ao valor

igual a 1000 (valor de *FFC_N_PILOTS_BLOCK*). No último estado ANGLE_CALC é calculado argumento do ângulo desse acúmulo que é multiplicado pela constante $\frac{1}{\pi(N+1)}$.

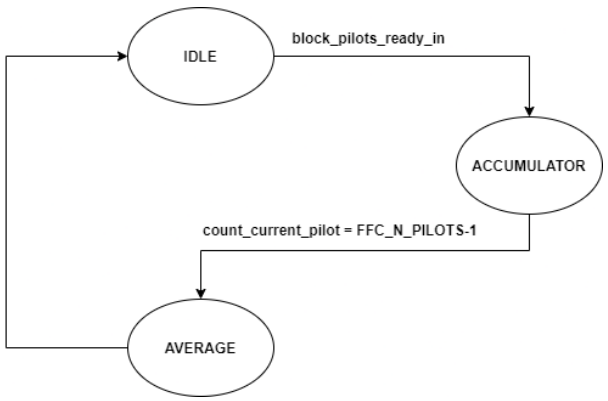

Figura 14 – Máquina de estados que executa a auto correlação dos pilotos.

Um ponto importante sobre esse cálculo final do argumento é que no Fixlink o cálculo é feito através da função *angle* da biblioteca *numpy*, nessa função é extraído o ângulo do argumento complexo de entrada. Enquanto a implementação em VHDL foi utilizado o IP CORDIC da Xilinx para calcular o arco tangente e extrair o ângulo do argumento. Sendo assim, não é possível uma comparação bit a bit nessa etapa final do módulo.

No Fixlink, o algoritmo do FFC havia sido organizado em arquivos separados cada um representando um estado da máquina implementada em VHDL. Contudo, a forma de execução da implementação em *software* não condizia com um cenário real de operação. Da forma em que estava o teste, só era possível testar o módulo se existisse exatamente 1000 blocos de pilotos na entrada. Isso impossibilitava um cenário com a entrada continua de novos blocos de pilotos ou até mesmo verificar o valor de frequência estimado antes da passagem dos 1000 blocos de pilotos.

Nesse sentido, foi feita uma reorganização dos códigos, criando uma classe principal e na construção do objeto (pertencente a essa classe) foi criada uma FIFO com 1000 posições preenchida com valores nulos. Essa FIFO funcionava como uma fila, ou seja, nela é armazenado o resultado do acúmulo presente na equação 2.16 na sua última posição, depois é feito o descarte do valor no inicio da fila e por fim realiza-se a soma de todos os valores armazenados na FIFO. Com essa adaptação foi possível atualizar o valor estimado de frequência da forma desejada sem a limitação de só ser possível acionar o módulo ao ter exatamente 1000 blocos de pilotos.

Mesmo com o sub módulo FFE Estimator já implementado ele ainda não atendia os requisitos da norma (ETSI, 2014a), a qual exige valores de erro RMS máximos entre o desvio de frequência estimado e o desvio real para determinados valores de Eb/N0, conforme observado na tabela 1. No modelo do Fixlink foi realizado a alteração da resolução de entrada do módulo para melhorar seu desempenho e atender melhor a norma. A

resolução dos símbolos passou de 12 bits (1 bit de sinal,3 bits inteiros, 8 bits fracionários) para 14 bits (1 bit de sinal,3 bits inteiros, 10 bits fracionários). A mesma adaptação foi feita no código VHDL.

Tabela 1 – Valores do erro RMS para estimador de frequência fina exigidos na norma (ETSI, 2014a)

Eb/N0 (dB)	**Máximo Erro RMS Exigido**
-2	6e-5
1	5.20e-5
6.5	2.72e-5
10	1.82e-5

A figura 15 ilustra as conexões do sub bloco do FFC Estimator após as adaptações e a descrição das entradas e saídas do sub módulo está presente na tabela 2. O sub módulo recebe como entrada dois sinais para os símbolos, a parte real(em fase) e parte imaginaria (em quadratura), dos quadros com 14 bits em ponto fixo além da presença de sinais de sincronização e controle. Na saída o sub módulo apresenta o valor estimado para o desvio de frequência com 47 bits em ponto fixo sendo 43 bits para a parte fracionada e sinais de controla para indicar a saída de um valor valido.

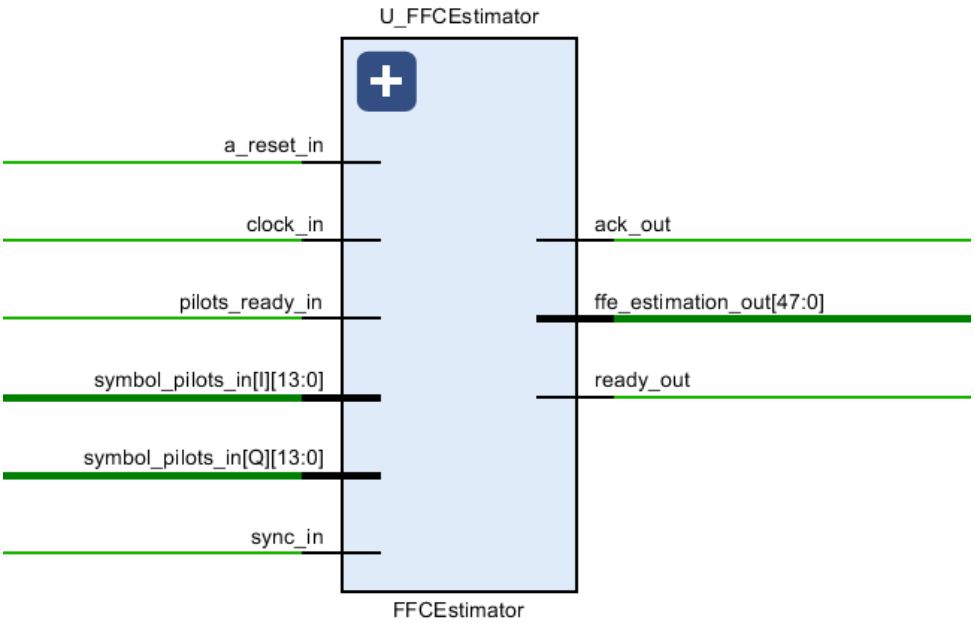

Figura 15 – Conexão do sub-bloco FFC Estimator

.

Tabela 2 – Descrição das entradas e saídas do sub módulo FFC Estimator

Nome	**Tipo**	**Descrição**	**Nº de Bits**
a_reset_in	Entrada	Reset assíncrono	1
clock_in	Entrada	Sinal de clock	1
pilots_ready_in	Entrada	Indica a leitura de uma nova entrada	1
symbol_pilots_in[I]	Entrada	Parte real dos símbolos pilotos	14
symbol_pilots_in[Q]	Entrada	Parte imaginaria dos símbolos pilotos	14
sync_in	Entrada	Indica a entrada de um novo quadro	1
ack_out	Saída	Indica que o módulo pode receber novas entradas	1
ffe_estimation_out	Saída	Valor estimado para o desvio de frequência	48
ready_out	Saída	Indica que a saída foi gerada	1

3.4.2 Sub módulo FFC Corrector

A implementação do sub módulo FFC Corrector, utilizou como base a implementação no Satlink que faz a multiplicação dos símbolos de entrada pelo exponencial complexa $e^{-j\theta}$, sendo θ a rotação de fase de cada símbolo, calculada por meio da equação 3.4. Onde o desvio de frequência estimado f_e é multiplicado pelo instante de amostragem k e depois é feito a soma com o acúmulo de fase do bloco anterior θ_{acc}. Com isso, foi observado que os símbolos sofrem um desvio de frequência proporcional à seu instante de amostragem, ou seja, o primeiro símbolo não sofre desvio ($k = 0$), o segundo sofre uma vez o desvio ($k = 1$), o terceiro sofre duas vezes o desvio ($k = 2$) e assim por diante.

$$\theta = 2\pi k f_e + \theta_{acc} \tag{3.4}$$

Logo para a implementação é necessário saber a posição de chegada dos símbolos. Para o simulador em ponto fixo esse problema pode ser solucionado ao utilizar funções da biblioteca *numpy*. Entretanto, para a implementação *hardware* é necessário a implementação de um contador o qual deve incrementar a cada chegada de um novo símbolo, porém é necessário impor uma limitação para o contador, ou seja, determinar um ponto de *reset*. Inicialmente, a limitação foi de 1440 símbolos para que a correção ocorra a cada bloco de dados novo, porém após alguns testes notou-se que o melhor ponto de *reset* do

contador seria quando o desvio de fase chegasse próximo a 2π. Como o ponto de *reset* em 2π teve um desempenho mais satisfatório optou-se por usá-lo na implementação em VHDL.

O diagrama de blocos do FFC Corrector implementado em VHDL está presente na figura 16. O sub módulo recebe como entradas tanto dados como pilotos, pois ambos precisam ser corrigidos para serem usados no próximo módulo da CPS, o PC. A primeira etapa, chamada *Phase Rotation*, faz o calculo do desvio de fase de cada símbolo, usando o valor de desvio de frequência estimado pelo FFC Estimator. O contador com *reset* em 2π foi implementado nessa etapa. O próximo sub bloco, chamado *Phase to MEMO*, converte o valor da rotação de fase dos símbolos para um endereço de memoria correspondente ao valor de seno e cosseno na tabela do bloco Sintable. Na etapa final é feita a multiplicação dos símbolos de entrada com os valores de seno e cosseno tabelados e assim é feita a rotação final dos símbolos.

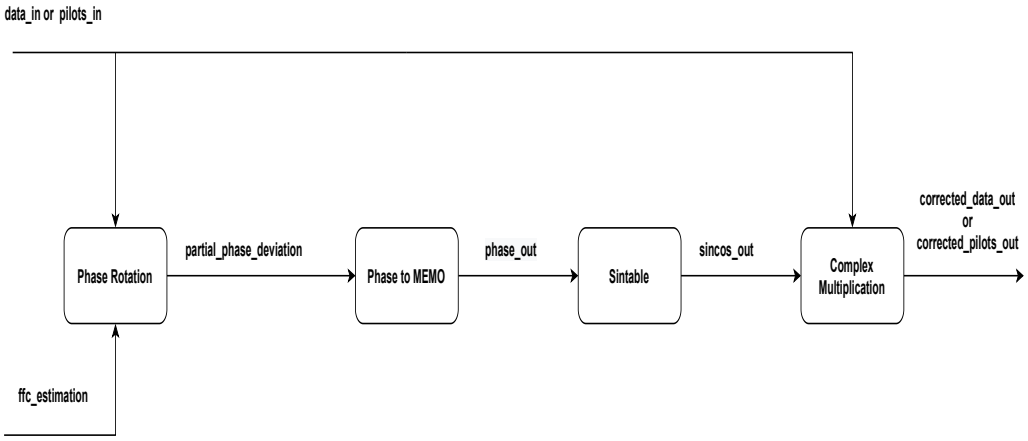

Figura 16 – Diagrama de blocos do FFC Corrector

.

Na figura 17 é possível observar as conexões do sub bloco do FFC Corrector e a tabela 3 descreve todas as entradas e saídas desse sub módulo.

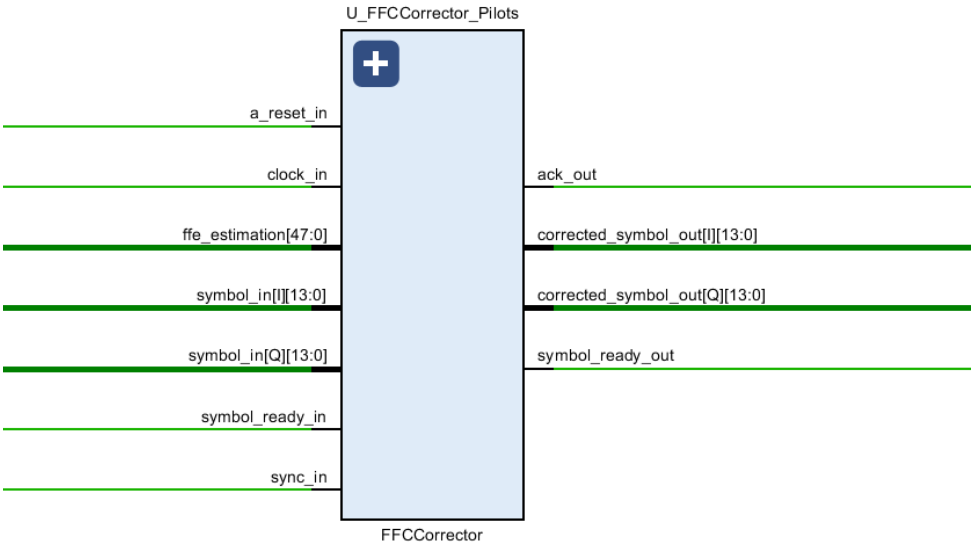

Figura 17 – Conexão do sub-bloco FFC Corretor

.

Tabela 3 – Descrição das entradas e saídas do sub módulo FFC Corretor

Nome	**Tipo**	**Descrição**	**Nº de Bits**
a_reset_in	Entrada	Reset assíncrono	1
clock_in	Entrada	Sinal de clock	1
ffe_estimation_out	Entrada	Valor estimado para o desvio de frequência	48
symbol_in[I]	Entrada	Parte real dos símbolos	14
symbol_in[Q]	Entrada	Parte imaginaria dos símbolos	14
symbol_ready_in	Entrada	Indica a leitura de um novo símbolo	1
sync_in	Entrada	Indica a entrada de um novo quadro	1
ack_out	Saída	Indica que o módulo pode receber novas entradas	1
corrected_symbol_out[I]	Saída	Parte real dos símbolos corrigidos	14
corrected_symbol_out[Q]	Saída	Parte imaginaria dos símbolos corrigidos	14
ready_out	Saída	Indica que a saída foi gerada	1

3.4.3 Implementação em *Hardware*

O teste físico do módulo FFC foi feito no kit de desenvolvimento Zedboard Zynq-7000 AP, figura 18, que conta com um processador ARM-Cortex A9 dual core; dois osciladores um com 667 MHz para o PS (*Processing System*) e o outro com 100MHz para uso da PL (*Programmable Logic*); 8 chaves DIP/*slide* ; 7 botões *push-up*; botões de *reset* e 9 LEDs para o usuário (DIGILENT, 2014).

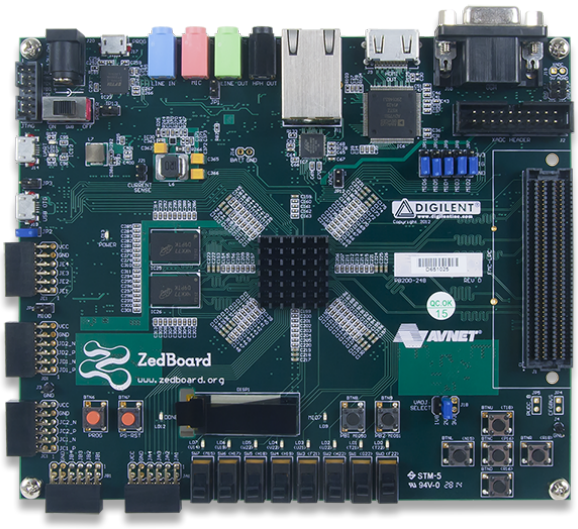

Figura 18 – Kit de desenvolvimento ZedBoard Zynq-7000
. Retirado de (DIGILENT, 2014)

Para o teste em *hardware* foram utilizados o IP ILA (*Integrated Logic Analyzer*) da Xilinx e dois pares de memorias ROMs (*Read Only Memory*) para armazenar os vetores de teste. Além disso foi desenvolvido um circuito para controlar a leitura das memorias ROMs. O funcionamento desse controlador será detalhado no próximo capítulo. O resultado do módulo FFC foi extraído pelo ILA e para verificar se o módulo funcionou como esperado um *script* em Python foi usado para ler a saída gravada em um arquivo do tipo csv e gerar a constelação de saída.

3.5 Integração da cadeia CPS

3.5.1 Analise dos códigos já implementados

O primeiro passo para a integração do Fixlink foi analisar as implementações feitas por outros integrantes do projeto que desenvolveram os seguintes módulos: *Timming Recovery* (TRINDADE, 2019), CFC (PINTO, 2020) e PC (PINTO, 2020).

A primeira análise foi feita no *Timming Recovery*, cujo código foi feito adaptando a implementação do Satlink para aritmética de ponto fixo.

O algoritmo usado pelo Satlink foi o algoritmo Gardner (GARDNER, 1986) (vide Seção 2.3.2), usando uma rotação de fase para simular o NCO (*Numerically Control Oscillator*), que na CPS está presente no transceptor. O NCO faz parte do modelo implementado para o *Timming Recovery*, pois ele é o responsável em realimentar o sistema ao capturar a saída do algoritmo Gardner é retornar o erro de tempo de amostragem μ usado como peso nos símbolos de entrada, compensando assim o erro a cada nova iteração do sistema.

O teste do código havia sido feito com o filtro RRC do Satlink em ponto flutuante. Dessa forma, a única adaptação feita no código foi alterar o filtro para o RRC do Fixlink. Depois dessa mudança restou comparar a implementação em *software* com a implementação em *hardware*.

Quanto ao CFC a implementação desse módulo depende da integração com o transceptor para realizar a correção completa dos símbolos, por isso só foi implementado o estimador. O código já implementado para o estimador é compostos pelos 3 sub módulos descritos abaixo.

- Detector de Erro de Frequência (FED, do inglês *Frequency Error Detector*): O detector foi implementado de acordo com a equação 2.10 onde foi usado uma FIFO com 3 posições para armazenar as amostras necessárias para os cálculos.
- Filtro PI (Proporcional Integrador): Os coeficiente $k1$ e $k2$ do filtro foram calculados com base nas equações 2.11 e 2.12, respectivamente, mas o valor da banda não foi o especificado na norma e sim o valor de 2.10^{-2}. Os valores encontrados foram de $6,665.10^{-4}$ e $8,887.10^{-6}$ para $k1$ e $k2$, respectivamente.
- Filtro de média móvel: O filtro seguiu a implementação recursiva apresentada em 2.13.

Devido à falta de um sub módulo que executa a correção dos símbolos após estimar o desvio de frequência foi necessário criar um corretor para esse módulo no Fixlink. A correção implementada segue a mesma implementação feita no FFC Corrector, por isso

só foi necessário reorganizar os códigos já existentes para o estimador do CFC e adaptar a resolução das operações feitas para corrigir os símbolos. Contudo, no FFC foi utilizado um modelo em ponto fixo para o Sintable, mas esse módulo possui um tempo de execução muito lento em *software*. Portanto, optou-se por retira-lo no CFC e usar funções da *numpy* para estimar os valores de seno e cosseno.

Com a inclusão do corretor no CFC restou analisar o código do PC. Durante a análise do código não foi encontrado nenhum ponto que necessita-se de alteração. O algoritmo do PC usado é de máxima verossimilhança e de interpolação linear recomendado pela norma (ETSI, 2014a).

3.5.2 Integração do códigos

Para facilitar a integração de todos os códigos, foi feita uma organização em classes com seus cálculos organizados em métodos. Com essa adaptação em todos módulos foi possível acessar e alterar facilmente a resolução de entrada e saída dos módulos. A resolução em bits que apresentou o melhor desempenho para cada um dos módulos da CPS está presente na tabela 4.

Tabela 4 – Parâmetros de simulação do Fixlink configuráveis pelo usuário.

Módulo	**Resolução (Entrada)**	**Parte Fracionaria (Entrada)**	**Resolução (Saída)**	**Parte Fracionaria (Saída)**
Timming Recovery	11	7	11	7
Filtro RRC	12	9	12	9
FrameSync	11	7	11	7
CFC	14	10	14	10
FFC	14	10	14	10
PC	14	10	14	10

A execução da simulação principal é feita quadro por quadro, sendo necessário a passagem de mais de um quadro para que seja possível executar todas as etapas de correção. Também foram adicionadas *flags* de geração de vetores binários da entrada e saída de todos os módulos. Porém é importante ressaltar que como a simulação roda quadro por quadro os vetores de teste serão criados para cada quadro simulado, o que pode ocupar um espeço em disco considerável da máquina em que está sendo realizado os testes.

A criação dos quadros usados durante a simulação é feita com métodos disponibilizados pelo simulador Satlink, assim como o canal AWGN e o canal que aplica o desvio de frequência que também pertence ao simulador Satlink. A descrição, nome, e tipos de dados dos parâmetros para a simulação geral que podem ser definidos pelo usuário, são apresentados na tabela 5.

Tabela 5 – Parâmetros de simulação do Fixlink configuráveis pelo usuário.

Nome da variável	**Tipo**	**Descrição**
number_frames	Inteiro	Número de quadros usados na simulação.
seed	Inteiro	Semente usada na geração do ruído do canal AWGN e para os dados aleatórios dos quadros.
modcod_name	String	Modulação escolhida para os dados
roll_off_factor	Float	Fator de *roll-off* do filtro RRC
num_samples_symbol	Inteiro	Número de amostras por símbolos
filter_span	Inteiro	Span do filtro RRC
fo_real	Float	Desvio de frequência normalizada aplicado nos símbolos
ebno	Float	Valor de Eb/N0 do canal AWGN

Durante a integração do Fixlink alguns problemas foram encontrados nos módulos implementados pelos outros integrantes do projeto. O primeiro problema foi observado no CFC, quando o filtro RRC foi posicionado de acordo com o digrama apresentado na figura 1, antes do FrameSync, observou-se que a correção dos símbolos no CFC não estava sendo realizada.

A partir disso foi feita uma investigação no sub módulo que realiza a correção no CFC, porém nada de atípico foi encontrado visto que o CFC havia funcionado de acordo com o esperado durante seu teste individual. Por isso foi feita uma investigação no simulador Satlink a fim de verificar como havia sido feita a integração do CFC.

Nesta investigação foi descoberta a ausência do *Timming Recovery* e do FrameSync na integração do Satlink e também foi notado que o filtro RRC estava posicionado após o CFC, sendo assim foi necessário averiguar o porque desse posicionamento. Ao alterar os valores de Eb/N0, fator de roll-off do filtro RRC e seu span não foi obtido nenhum resultado que justificasse esse posicionamento. Entretanto, ao alterar o fator de sobre amostragem notou-se que essa era causa do mal funcionamento do CFC, a entrada dele precisava estar sobre amostrada para que fosse feita a correção de frequência. Com isso o filtro RRC que também é responsável por retirar a sobre amostragem teve que ser deslocado para após o CFC, porém essa alteração gerou impactos negativos no FrameSync que precisa estar posicionado antes do CFC para poder separar o quadro em blocos de dados e blocos de pilotos.

Além de que o FrameSync não consegue operar com os símbolos sobre amostrados, por isso foi necessário adicionar uma *flag* de *bypass* no Framesync que usa como entrada o valor da sobre amostragem, sendo assim, caso a sobre amostragem dos símbolos seja

maior que 1 o FrameSync é desativado realizando somente a separação dos blocos de dados e blocos de pilotos sem identificar o SOF do quadro. É importante ressaltar que essa abordagem só é possível em *software*.

Adicionalmente durante os testes do *Timming Recovery*, que serão apresentados no próximo capitulo, notou-se uma diferença entre as arquiteturas em *software* e em *hardware*. E devido a essa diferença de arquiteturas e a seu posicionamento no inicio da CPS o *Timming Recovery* gera um impacto negativo na comparação das saídas entre o Fixlink e o VHDL dos módulos posicionados após ele, na CPS. Por isso também foi adicionada uma *flag* de *bypass* a qual pode ser acionada ou não pelo usuário.

Por fim o último problema encontrado foi no PC. Durante a integração com esse módulo notou-se um comportamento atípico em sua correção. Basicamente, a correção estava sendo feita corretamente durante uma sequência de quadros, porém a saída de alguns dos quadros voltava a apresentar o ruído de fase. Ao analisar o módulo PC implementado pelo Satlink notou-se a ausência do algoritmo de desdobramento de fase, que limita o ângulo da rotação de fase entre π e $-\pi$ retirando ambiguidades de fases, no Fixlink.

Esse algoritmo foi feito da seguinte maneira: primeiro é feita a diferença entre o ângulo de rotação atual com o anterior; depois é feito o ajuste dessa diferença para o intervalo de $-\pi$ e π, o ajuste é feito ao calcular o resto da divisão entre a diferença dos ângulos de rotação e π.

Por fim é feita a soma da diferença ajustada com o ângulo de rotação anterior. Após implementar esse algoritmo no Fixlink a integração com PC foi feita com sucesso. Uma observação importante sobre o simulador é seu tempo de execução, tanto o módulo FFC e PC utilizam o sub módulo Sintable em ponto fixo, o qual apresenta um tempo de execução muito alto em *software*. No intuito de resolver esse problema, o módulo foi desativado na integração porém é possível ativa-lo através de uma flag, mas o tempo de execução aumenta consideravelmente sendo inviável utilizar o Fixlink em computadores convencionais.

Por fim, devido aos problemas citados anteriormente não foi possível finalizar a integração total da CPS, faltando integrar o AGC (*Automatic Gain Controller*) e o estimador SNR (*Signal to Noise Ratio*). Entretanto, a integração até o PC foi realizada com sucesso e o resultado dessa integração será apresentado no próximo capitulo.

4 Resultados

4.1 Filtro RRC

O modelo em MATLAB implementou o filtro RRC como um filtro FIR. No simulador Fixlink seguiu-se a mesma abordagem e com isso obteve-se a resposta ao filtro, com os coeficientes não quantizados, presente na figura 19. Foram utilizados os seguintes parâmetros no filtro RRC para gerar os resultados dos testes: fator de *roll-off* = 0.2, número de amostras por símbolos = 8 e *span* do filtro = 8 (ordem do filtro 65 TAPS).

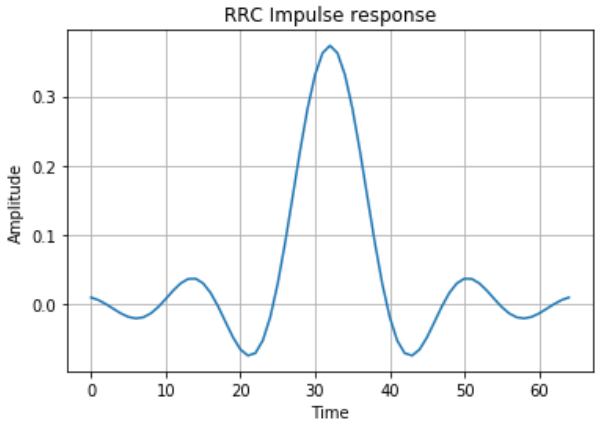

Figura 19 – Respostas ao impulso do filtro RRC

.

Para simular o processo de filtragem do quadro utilizaram-se três modulações, sendo elas: QPSK com taxa de código 1/4 com Eb/N0 de 1dB; 8PSK com taxa de código 3/5 com Eb/N0 de 1db e 16APSK com taxa de código 2/3 com Eb/N0 de 1dB. Todos os quadros utilizados na simulação são do tipo *short-frame*, tendo uma resolução dos símbolos de entrada de 12 bits, sendo 9 bits para a parte fracionara. A saída do filtro na recepção possui a mesma resolução. As figuras 20, 21 e 22 apresentam a constelação transmitida e recebida após passar pelo filtro RRC. Nas figuras, os pontos em vermelho são as entradas com ruído do canal AWGN (*Additive White Gaussian Noise*), sem desvio de frequência. O efeito do filtro RRC é observado nos pontos em preto, que são os símbolos após a passagem pelo filtro. O funcionamento do filtro é observado com o agrupamento dos símbolos em preto dentro de suas respectivas posições na constelação, seguindo o padrão de sua modulação.

O desempenho da notação em ponto fixo do Fixlink foi verificado ao calcular o erro MSE com a saída em ponto flutuante para os mesmos cenários. A tabela 6 apresenta esses valores, com o erro na ordem 10^{-8} para os símbolos I (em fase) e 10^{-6} para os símbolos Q

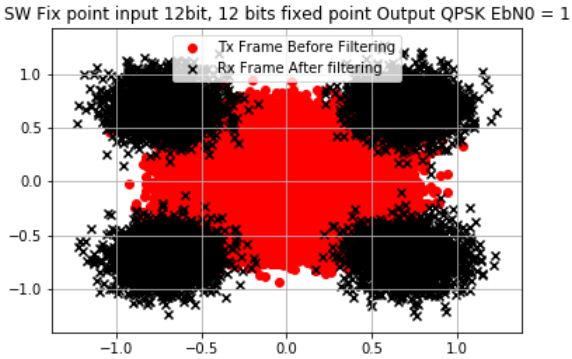

Figura 20 – Constelação transmitida e recebida, modulação QPSK

.

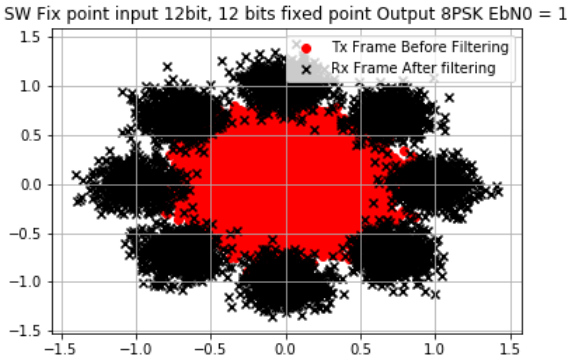

Figura 21 – Constelação transmitida e recebida, modulação 8PSK

.

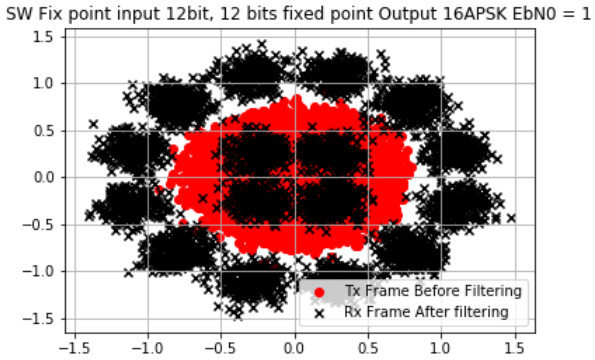

Figura 22 – Constelação transmitida e recebida, modulação 16APSK

.

(em quadratura) é possível concluir que a notação em ponto fixo 12 bits é eficiente para os cenários testados.

O principal objetivo do Fixlink é servir de referência para a implementação VHDL, por isso foi feita a comparação da saída do Fixlink com a simulação comportamental em VHDL. Nesse teste foi obtida a tabela 7, a qual apresenta os valores de 11 amostras

Tabela 6 – Valores de MSE para representação em float Python e a representação 12 bits ponto fixo. Eb/N0 = 1 dB.

Modulação	**Erro MSE**
QPSK	-2.369e-08+1.894e-06i
8PSK	3.286e-08+1.899e-06i
16APSK	2.86e-08+1.908e-06i

do Fixlink e da implementação em VHDL, assim como o valor MSE para 500 amostras. Para esse módulo não foi possível uma comparação bit a bit, mas o valor do MSE está baixo demonstrando a eficiência do Fixlink em servir de modelo de referência para o filtro RRC. O erro que impede a comparação bit a bit está nos bits finais dos símbolos, como é ilustrado na figura 23. As variáveis *Out_intP_rx* e *rrc_i_out* representam os símbolos de saída em fase e *Out_intQ_rx* e *rrc_q_out* os símbolos de saída em quadratura, a diferença está um valor a mais ou a menos o que indica a diferença nos últimos bits.

Tabela 7 – Comparação entre os símbolos de saída em VHDL e em python ponto fixo. Eb/N0 = 1 dB.

Simulação VHDL	**Simulação Python**
0.000000+0.000000j	0.000000+0.000000j
0.001953+0.001953j	0.001953+0.001953j
-0.001953-0.007812j	-0.001953-0.007812j
0.001953+0.017578j	0.001953+0.017578j
-0.019531-0.074219j	-0.021484-0.074219j
-0.042969-0.058594j	-0.044922-0.060547j
-0.072266+0.007812j	-0.074219+0.005859j
-0.074219-0.060547j	-0.076172-0.062500j
-0.714844+0.683594j	-0.714844+0.681641j
-0.714844-0.761719j	-0.714844-0.761719j
0.677734-0.789062j	0.675781-0.789062j
Erro MSE:	**4.718e-06+4.397e-06j**

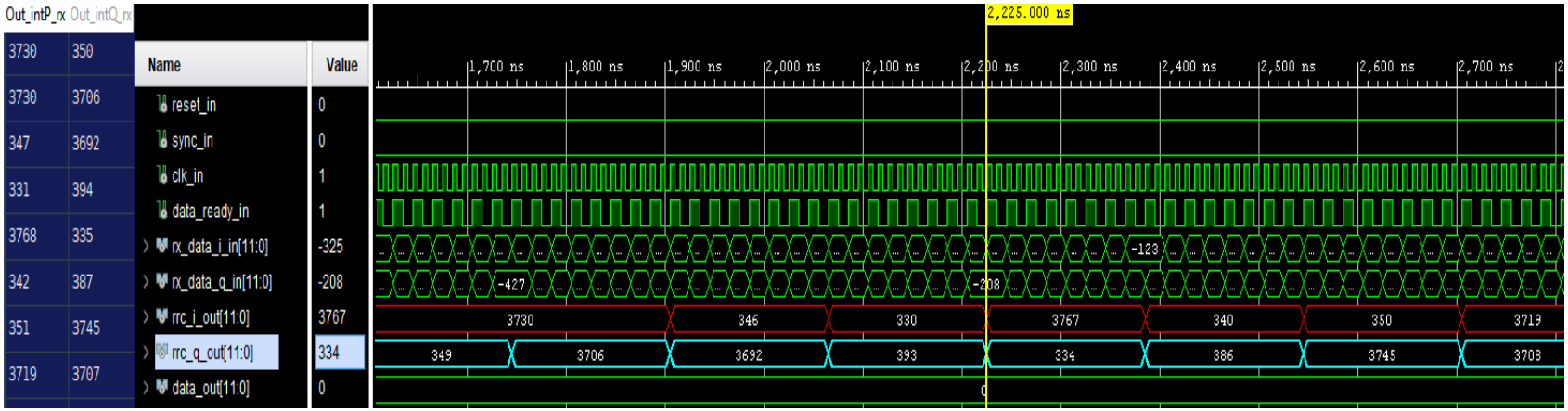

Figura 23 – Comparação entre os símbolos de saída entre a simulação em *software* e em *hardware*

.

4.2 *Timming Recovery*

Como o Timming Recovery foi implementado por (TRINDADE, 2019), os testes realizados serviram para verificar se as arquiteturas implementadas no Fixlink e na implementação VHDL são iguais. O teste foi feito ao analisar o valor do MSE para μ nos valores de Eb/N0 de -2 db a 9 db com passo de 1 db com quadros modulados em QPSK. O resultado desse teste está presente na figura 24. Graças a esse teste foi possível analisar que o erro entre as duas implementações está alto para valores inferiores a 2 db de Eb/N0 e a partir desse ponto o MSE passa a ser inferior a 0,1. Contudo, esses valores abaixo de 0,1 não são suficientes para considerar que as arquiteturas são semelhantes, por isso foi feito o calculo dos valores de MSE entre as saídas.

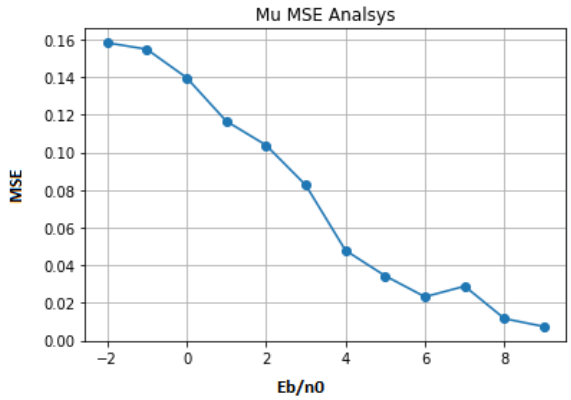

Figura 24 – Valores de MSE do Mu a cada valor Eb/N0

.

Na figura 25 estão presentes os valores de MSE para os símbolos de saída simulados no mesmo cenário do teste anterior. O gráfico da esquerda é a comparação entre os símbolos I (em fase) e o da direita dos símbolos Q (em quadratura). Com essa análise confirma-se que as arquiteturas são diferentes devido ao valor alto de MSE obtido na maioria dos valores de Eb/N0 além do comportamento atípico no ponto $Eb/N0 = 7db$ dos gráficos. É importante ressaltar que nesse teste, para cada valor de Eb/N0, foram usadas 8000 amostras para calcular o erro MSE dos sinais de saída.

Por fim como as arquiteturas são diferentes, o último teste realizado foi feito para verificar se a implementação em VHDL de fato realiza a correção dos símbolos. Para isso comparou-se as constelações de saída do Fixlink com o VHDL. Na figura 26 está presente o desenho de ambas constelações. As constelações foram criadas com valor de $Eb/N0 = 7db$ e em ambas é possível ver o desenho da modulação simulada (QPSK). Por fim na figura 27 é ilustrada a constelação da entrada que foi a mesma para ambas implementações, com isso conclui-se que efetivamente a implementação em VHDL do *Timming Recovery* executa a correção, porém existe algum ponto na implementação em VHDL (ainda não identificado) que é diferente da implementação do Fixlink.

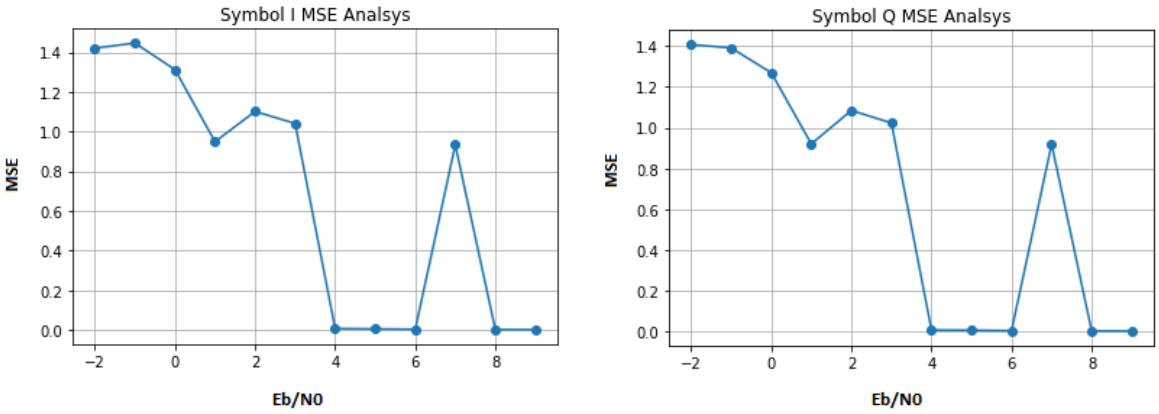

Figura 25 – Valores de MSE dos símbolos de saída Fixlink Vs. VHDL

.

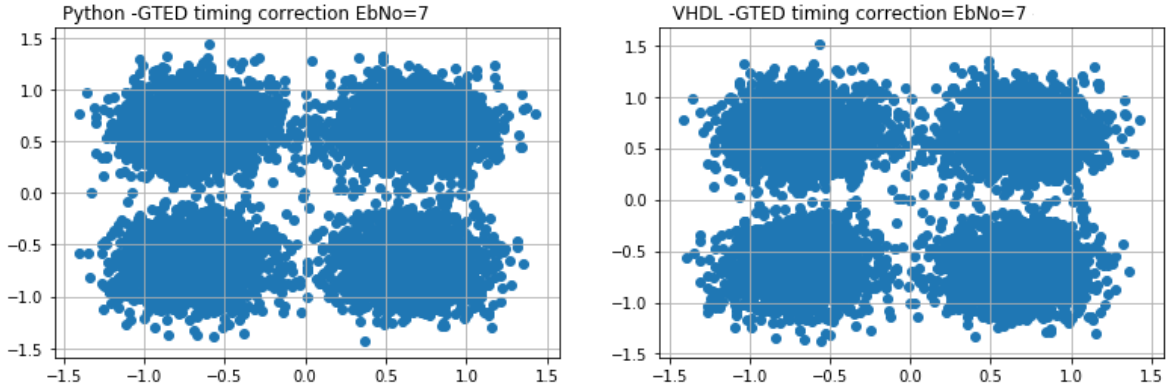

Figura 26 – Contelações de saída do Fixlink (Esquerda) Vs. VHDL (Direita)

.

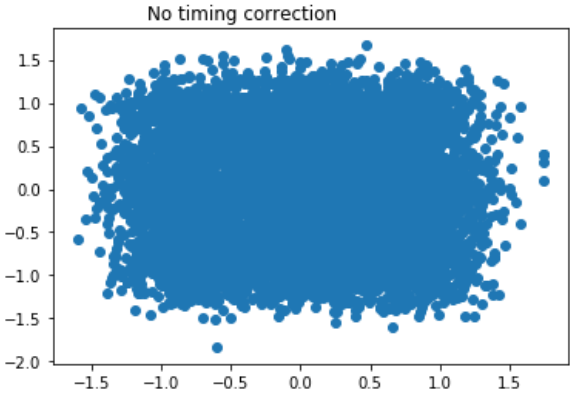

Figura 27 – Constelação do quadro transmitido na simulação do *Timming Recovery*

.

4.3 Sincronizador de quadros (FrameSync)

Os testes iniciais do FrameSync implementado no Fixlink foram feitos com a finalidade de comparar seu desempenho com o Satlink. No primeiro teste foi comparado o desempenho dos simuladores para os cenários de Eb/N0 variando de -2 db a 10 db ao passo de 1 db por simulação. Durante o teste foram passados 2 quadros agrupados com dois blocos de dados, de tamanhos sortidos, concatenados no inicio e no fim dos 2 quadros agrupados. A modulação foi escolhida de forma aleatória assim como o tamanho dos quadros, com isso a posição do SOF dos quadros em todos os testes era aleatória.

Esse formato de teste foi realizado 50 vezes por valor de Eb/N0 simulado, ou seja, para cada valor de Eb/N0 foram passados 100 quadros a serem detectados. Desse teste foi obtido o gráfico presente na figura 28. Observa-se na curva em azul o desempenho do simulador Fixlink que se iguala ao Satlink(curva em laranja) no ponto de Eb/N0 de 2 db tendo cerca de 90% dos quadros detectados e abaixo desse valor o modelo do Fixlink possui o desempenho inferior ao Satlink. Observa-se que para valores abaixo de 1db, o Fixlink passa a ter uma detecção de quadros inferior a 80%, motivo pelo qual a operação do FrameSync só é aceitável em valores superiores a 1db.

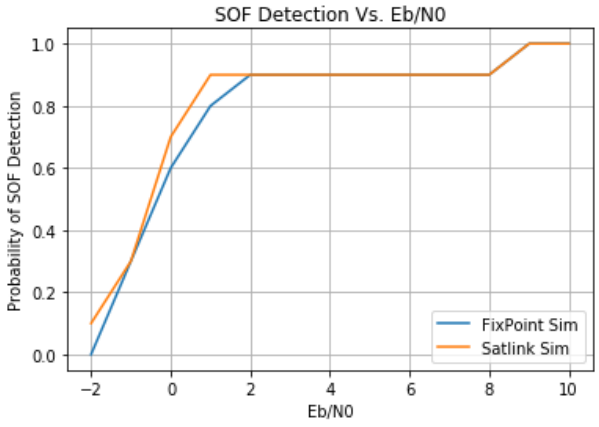

Figura 28 – Probabilidade da detecção de um frame em função do valor de Eb/N0

.

A fim de validar a arquitetura implementada no Fixlink foi feita a comparação com o VHDL da saída dos correlatores do PLSC e do SOF apresentada, respectivamente, nas tabelas 8 e 9. Nas tabelas são apresentados os valores em inteiro de 8 saídas. Conclui-se que é possível a comparação bit a bit nesse ponto do FrameSync, pelo fato de que o valor do MSE para 8000 amostras em ponto fixo 11 bits (1 bit de sinal, 3 bits parte inteira e 7 bits fracionários) teve o valor 0.

Para finalizar os testes no FrameSync foi feito um teste de estresse ao substituir 90 símbolos de dados pelos símbolos do cabeçalho de um quadro, essa substituição serviu para simular a inserção de SOFs falsos no meio de um quadro verdadeiro. O objetivo desse teste foi verificar em qual condição o FrameSync iria falhar. Foram realizados 4

testes alterando a posição dos SOFs falsos dentro de um grupo de 10 quadros verdadeiros do tipo *short* (8370 símbolos) modulados em QPSK com SNR variando de -5 db a 9 db ao passo de 1db. A descrição de cada teste está abaixo:

- **Teste 1 :** Um SOF falso posicionado a 100 símbolos após o SOF do primeiro quadro;
- **Teste 2 :** Dois SOFs falsos, o primeiro posicionado na mesma distância do teste 1 e o segundo posicionado 500 símbolos após o SOF do segundo quadro;
- **Teste 3 :** Dois SOFs falsos, o primeiro posicionado 500 símbolos após o SOF do primeiro quadro e o outro a uma distância equivalente de um quadro do tipo *short* em relação ao primeiro SOF falso;
- **Teste 4 :** Três SOFs falsos, os dois primeiros com o mesmo posicionamento do teste 3 e o terceiro a uma distância de um quadro do tipo *short* do segundo SOF falso;

Tabela 8 – Comparação entre os as correlações de saída em VHDL e em python ponto fixo para o PLSC

Correlação PLSC VHDL	**Correlação PLSC FixLink**
20537122	20537122.0
2958397	2958397.0
962002	962002.0
506873	506873.0
54025	54025.0
749114	749114.0
309805	309805.0
40225	40225.0
Erro MSE:	**0**

Tabela 9 – Comparação entre os as correlações de saída em VHDL e em python ponto fixo para o SOF

Correlação SOF VHDL	**Correlação SOF FixLink**
3013865	3013865.0
135349	135349.0
228709	228709.0
612509	612509.0
161668	161668.0
127517	127517.0
306709	306709.0
109898	109898.0
Erro MSE:	**0**

Os resultados dos testes foram organizados na tabela 10 com a quantidade de quadros falsos e verdadeiros detectados para cada SNR simulada. Nos testes 1 e 2 na maioria dos cenários de SNR, os SOFs falsos não foram encontrados devido a falta de picos com a distância equivalente a um quadro em relação a eles, mostrando o bom funcionamento da técnica de janelamento utilizada.

Contudo, em todos os testes foram encontrados quadros falsos em um cenário extremo com SNR de -5 db que pela falta da limitação de 8 picos por janelamento apresenta uma quantidade exagerada de picos dentro do janelamento.

Nos testes 3 e 4 o posicionamento dos SOFs foi proposital para forçar a detecção falsa dos quadros, a detecção é presenciada a partir do valor de SnR de -3 db. Em todos os testes a detecção dos quadros verdadeiros foi a mesma, mas em casos com SNR inferiores a 0 db foram perdidos alguns dos quadros. A causa desse efeito é que devido ao ruído os valores dos picos dos SOFs ficam com sua magnitude menor do que o valor do limiar de decisão causando a perda de alguns quadros.

Tabela 10 – Quantidade de quadros falsos e quadros verdadeiros detectados em cada teste.

SnR	**Teste 1**		**Teste 2**		**Teste 3**		**Teste 4**	
(db)	**Falsos**	**Verdadeiros**	**Falsos**	**Verdadeiros**	**Falsos**	**Verdadeiros**	**Falsos**	**Verdadeiros**
-5	18	6	18	6	18	6	18	6
-4	0	7	0	7	0	7	0	7
-3	0	9	0	9	2	9	3	9
-2	0	9	0	9	2	9	3	9
-1	0	9	0	9	2	9	3	9
0	0	10	0	10	2	10	3	10
1	0	10	0	10	2	10	3	10
2	0	10	0	10	2	10	3	10
3	0	10	0	10	2	10	3	10
4	0	10	0	10	2	10	3	10
5	0	10	0	10	2	10	3	10
6	0	10	0	10	2	10	3	10
7	0	10	0	10	2	10	3	10
8	0	10	0	10	2	10	3	10
9	0	10	0	10	2	10	3	10

4.4 Corretor Grosso de Frequência (CFC)

A alteração feita no CFC, em relação ao seu modelo já implementado, foi a adição do sub módulo que executa a correção dos símbolos. Para os testes do corretor usou-se o modelo implementado pelo Satlink como meio de comparação. No primeiro teste foi feita a visualização das constelações na saída do CFC de 400 quadros modulados em QPSK com Eb/N0 de 80 db. A comparação entre as constelações dos últimos quadros testados é observada na figura 29, nota-se uma dispersão maior dos símbolos na simulação do Fixlink.

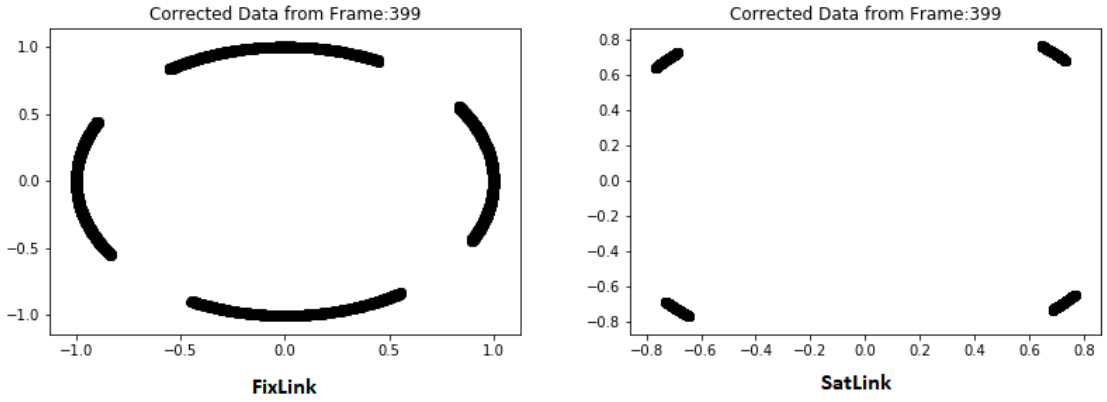

Figura 29 – Comparação das constelações de saída do Fixlink (Esquerda) com o Satlink (Direita)

.

Apesar dessa dispersão já ser esperada, devido a notação em ponto fixo, ela estava muito maior que o desejado, por isso foi necessário verificar qual dos cálculos estavam com a resolução insuficiente. A primeira análise foi feita nos valores da rotação de fase dos símbolos, alterou-se a resolução dessa etapa para verificar se algum efeito de melhoria era observado na constelação de saída. Ao verificar o erro MSE entre os simuladores para a rotação de fase dos símbolos, figura 30, identificou-se que o problema de resolução não estava nesse ponto, pois os valores de MSE estão na escala de 10^{-8}. Contudo, é importante fazer uma observação quanto ao crescimento do erro de acordo com a quantidade de blocos de pilotos, que em uma situação extrema com uma quantidade acima de 1000 quadros testados o erro pode aumentar consideravelmente, o que pode gerar uma possível falha na correção dos símbolos.

O próximo ponto investigado foi a multiplicação complexa da exponencial com os símbolos de entrada. Nesse ponto final foram alterados os valores de entrada de 12 bits (1 bit de sinal, 3 bits inteiros e 8 fracionários) para 14 bits(1 bit de sinal, 3 bits inteiros e 10 fracionários) e obteve-se um resultado mais satisfatório observado na figura 31. Nota-se, nas constelações da figura, um melhor agrupamento dos símbolos da constelação ao comparar a implementação de 12 bits com a de 14 bits, alcançando um desempenho mais

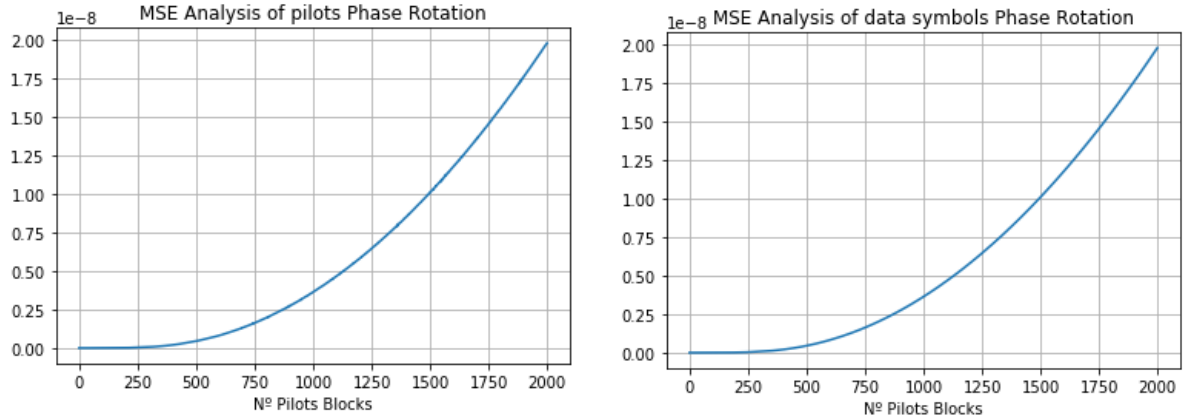

Figura 30 – Valores de MSE para os valores da rotação de fase dos símbolos de pilotos (Esquerda) e símbolos de dados (Direita)

.

satisfatório para o corretor do CFC.

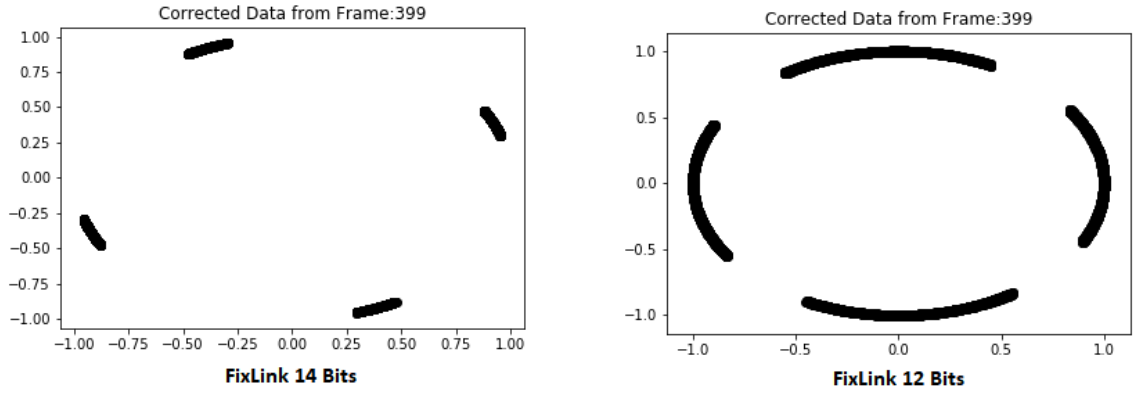

Figura 31 – Comparação das constelações de saída do Fixlink 14 bits (Esquerda) com o Fixlink 12 bits (Direita)

.

4.5 Corretor Fino de Frequência (FFC)

4.5.1 Simulações

O primeiro teste realizado no FFC foi analisar o comportamento do módulo ao alterar a resolução do quadro de entrada, a simulação foi feita para os desvios de frequência normalizada de 0,004; 0,003; 0,002; 0,001 e 0,0005 com valores de EbN0 variando de -2db a 10db com passo de 1db. Na figura 32 estão presentes os resultados da simulação em ponto fixo para um grupo de 1000 blocos de pilotos após alterar a resolução do quadro de entrada de 12 bits para 14 bits (1 bit de sinal, 2 bits inteiros e 11 bits fracionários). Na figura 33 está o desempenho anterior com 12 bits, nota-se o desempenho inferior a resolução de 14 bits nos desvios de 0,004 e 0,003 que não atendem a norma antes do valor de Eb/N0 de 1db. É importante ressaltar que as curvas apresentam um comportamento atípico causado por conta da semente usada na geração do ruído aleatório do canal AWGN que é aplicado nos símbolos pilotos de teste. O canal usado faz parte do simulador Satlink, mas a causa desse comportamento atípico não foi explicada com clareza pela equipe que desenvolveu o simulador Satlink.

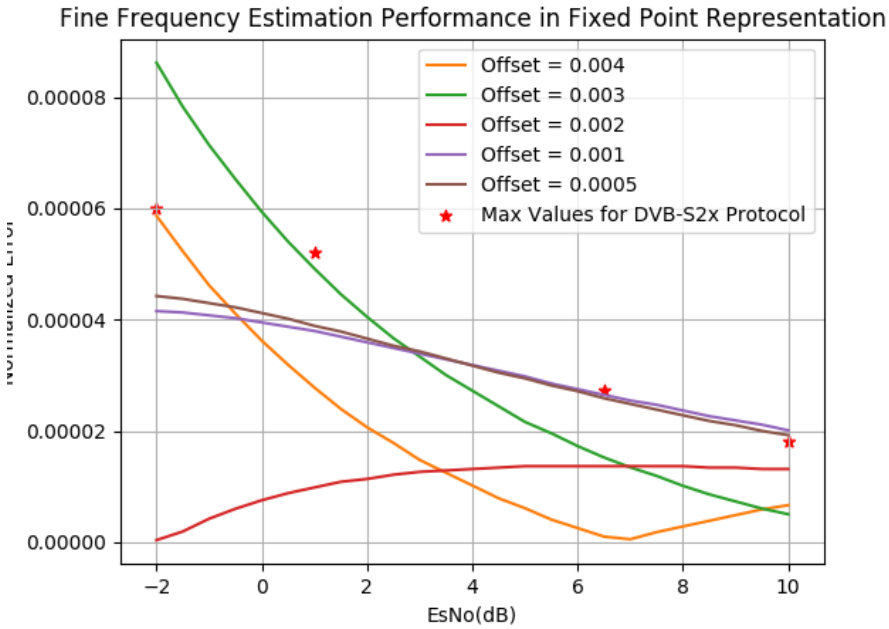

Figura 32 – Simulação do modelo em alto nível do FFE implementado em python

.

Ao observar o gráfico notou-se que os valores de RMS (para a maioria dos desvios simulados e cenários de Eb/N0) estão abaixo dos pontos destacados em vermelho, que são os valores exigidos pela norma. Mesmo com o comportamento atípico do gráfico os resultados se mostraram suficientes para realizar a mesma alteração em VHDL. Após alterar a resolução em VHDL, a simulação comportamental foi realizada com 1000 blocos de pilotos com $Eb/N0 = 6.6dB$. Durante a simulação a variável *ffe_estimation_out*,

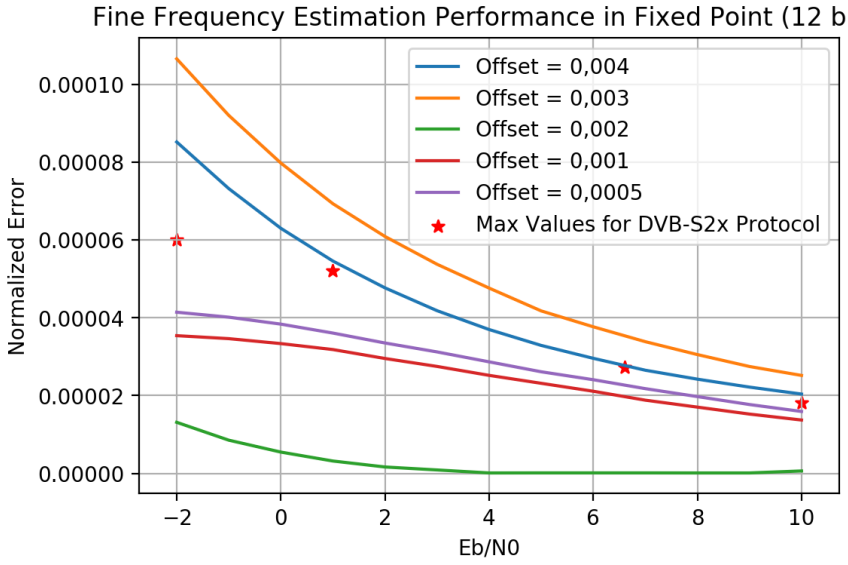

Figura 33 – Simulação do modelo em alto nível do FFE implementado em python com resolução de 12 bits

.

presente na figura 34, apresenta a transição de um valor nulo para um valor valido após o sinal de *ready_out* apresentar o valor lógico 1. Isso ocorre no instante de 20,22ms (destacado em amarelo na mesma figura) esse tempo é a latência do sub módulo FFC Estimator, que realiza a estimação do desvio de frequência do FFC.

Para verificar o desempenho entre as implementações, os valores do erro RMS para os desvios de frequência estimados entre o simulador Satlink, o simulador ponto fixo e a simulação comportamental foram organizados na tabela 11. Nela estão presentes os valores para somente valores de Eb/N0 presente na norma (ETSI, 2014a) com o desvio de frequência de 0,004, que é o pior cenário de operação do FFC. Na tabela 11 os valores de RMS das implementações devem ficar abaixo dos valores da norma, com isso somente a implementação do Fixlink atende a norma para o valor de -2db, porém nos demais valores as três implementações atendem a norma. É importante ressaltar que a implementações do Fixlink e em VHDL apresentam valores diferentes, pois o calculo final do argumento é feito de forma diferente. Em particular, o Fixlink utiliza uma aproximação do ângulo por pelo função *angle* da *numpy*, enquanto em VHDL é usado o IP *Cordic* da Xilinx.

O próximo teste foi para o FFC Corrector, que executa a rotação dos símbolos com base no valor estimado. Inicialmente foram testados dois pontos de *reset* para o acúmulo de fase dos símbolos de entrada. O primeiro ponto seria a cada novo bloco de dados e o segundo quando o acúmulo chegar a 2π. Na figura 35 é possível observar na constelação da esquerda que o *reset* em 2π é mais eficiente que a cada novo bloco da dados (figura da direita), por isso o mesmo ponto de *reset* foi adotado em VHDL. Por fim para validar o funcionamento do código do Fixlink foi feita a comparação entre as constelações de saída do Fixlink com o Satlink que pode ser vista na figura 36. A comparação foi feita

Tabela 11 – Comparação dos valores do erro RMS para estimador de frequência fina. Desvio de frequência real de 0.004

Eb/N0 (dB)	**Máximo Erro RMS Exigido**	**Erro RMS Satlink**	**Erro RMS Fixlink**	**Erro RMS Sim.Comportamental**
-2	6e-5	6.18e-05	5.86e-05	6.19e-5
1	5.20e-5	4.17e-05	2.77e-05	3.0897e-5
6.5	2.72e-5	2.12e-05	1.004e-06	4.0474e-6
10	1.82e-5	1.41e-05	6.65e-06	3.7101e-6

com $Eb/N0 = 6.6db$. As constelações são similares, porém não são iguais, mesmo assim a comparação foi suficiente para validar a correção feita pelo Fixlink.

Para a simulação comportamental foi utilizado um valor de Eb/N0 de 6.6db e como foi citado anteriormente o calculo final do argumento do Fixlink é diferente do VHDL. Por isso para verificar o funcionamento do FFC Corrector foi forçado no arquivo de *testbench* o valor de 0,00399999999990541 para o desvio de frequência estimado. A validação foi feita no próprio arquivo de *testbench*. Na figura 37 é possível ver as formas de onda da simulação comportamental. A variável *corrected_symbol_out* (forma de onda em azul) é a saída do FFCCorretor e a variável *reg_previous_symbol_in1* (forma de onda em roxo) é a saída do Fixlink lida por meio de um arquivo .txt. Nesse cenário, a validação é feita através da *flag* que apresenta o valor "PASSED"caso ambas variáveis de saída apresentem o mesmo valor. Quanto a latência desse sub módulo, a marcação em amarelo na figura 37 mostra o tempo 75ns para a primeira saída valida, sendo a latência do circuito, e a partir desse ponto uma nova saída é gerada a cada ciclo de *clock*, alcançando portanto uma taxa de processamento de 100 MOPS a uma frequência de *clock* de 100 MHz.

4.5.2 Implementação em Hardware

Para a implementação em *hardware* o teste foi feito primeiramente estimando uma de frequência valida antes de executar a correção dos dados, ou seja, o corretor só é acionado após um valor valido do FFC Estimator. O motivo dessa abordagem foi que por serem necessários 1000 blocos de pilotos para o estimador convergir e como cada quadro possui 5 blocos de pilotos seria necessário 200 quadros (no mínimo) para testar o módulo, exigindo muito espaço nas memorias ROMs. Sendo que todos os dados corrigidos antes da passagem dos 1000 blocos de pilotos seriam inúteis para a finalidade de validação do módulo FFC, porém é importante ressaltar que dentro de um cenário de integração do FFC com PC é necessário a passagem desses quadros descartados no teste do FFC.

Durante o teste foram usados dois pares de memorias ROMs o primeiro par com os símbolos I e Q dos 1000 blocos de pilotos que passam pelo FFC Estimator e a outra os que compõem um quadro, modulado em QPSK, a ser corrigido no FFC Corrector. O primeiro

momento do teste da integração dos módulos FFC Estimator e FFC Corrector foi realizar a simulação comportamental da integração assim como do circuito que faz o controle da leitura das memorias ROMs. Na figura 38 estão as formas de onda dessa simulação. É possível observar o funcionamento da leitura das memorias ROMs nas variáveis *addra* e *addrb* que são, respectivamente, o endereço de memoria com os blocos de pilotos e da memória de dados a serem corrigidos.

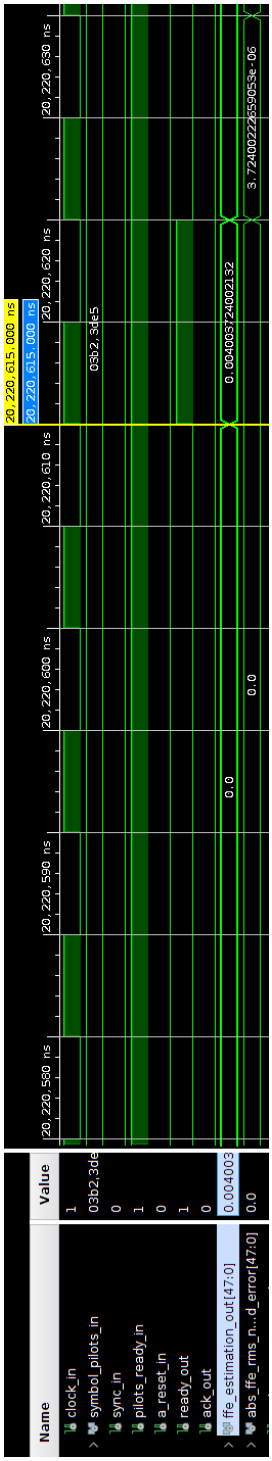

Figura 34 – Simulação comportamental do estimador de frequência fina para Eb/N0 = 6.6dB

.

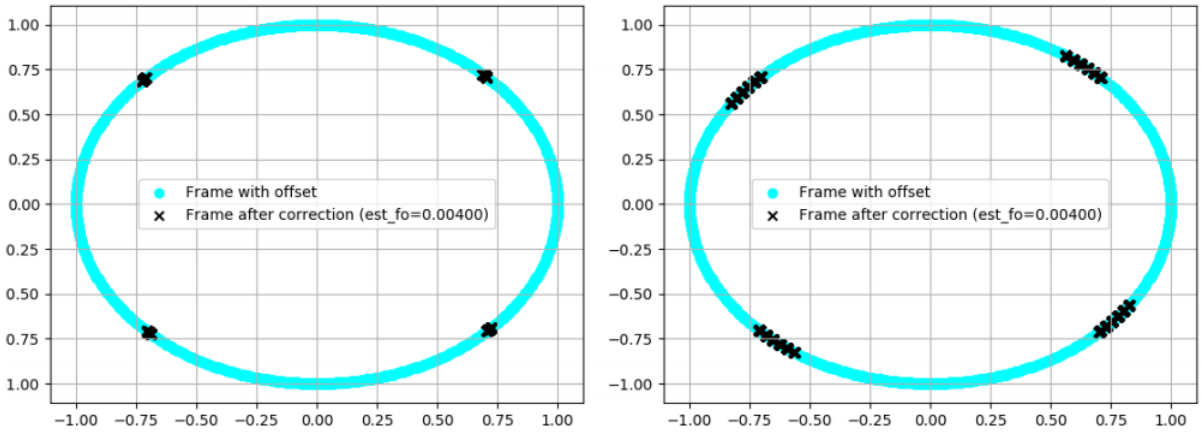

Figura 35 – Corretor com reset em 2π (esquerda), corretor com *reset* a cada bloco de dados (direita)

.

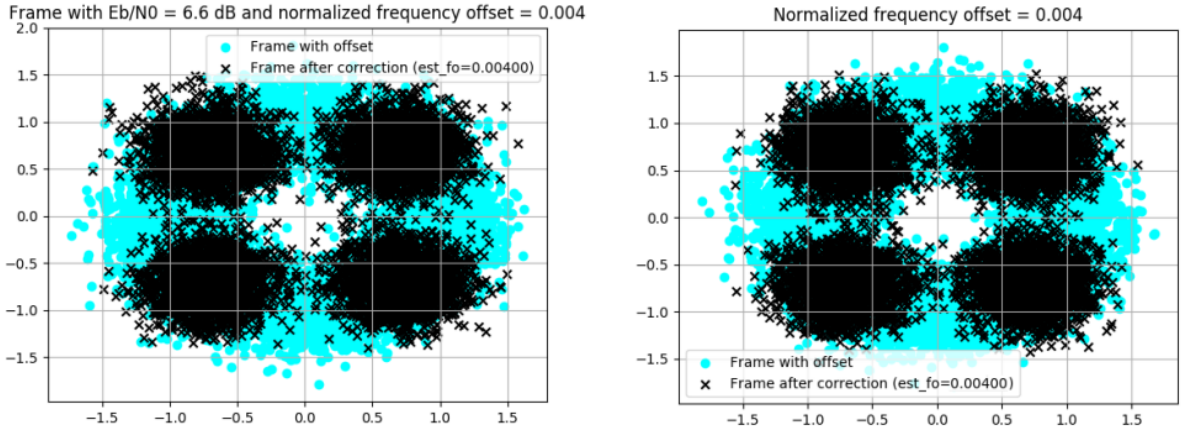

Figura 36 – Corretor modelo Fixlink (esquerda), corretor modelo Satlink (direita)

.

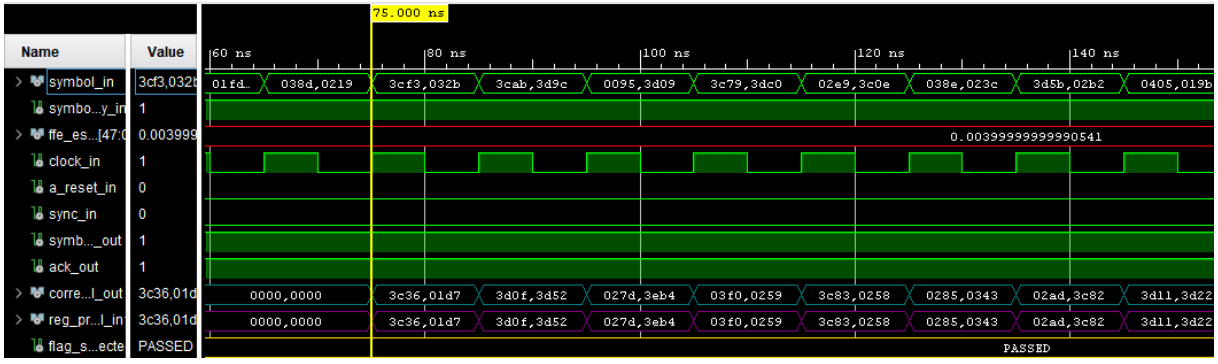

Figura 37 – Simulação comportamental do corretor de frequência fina

.

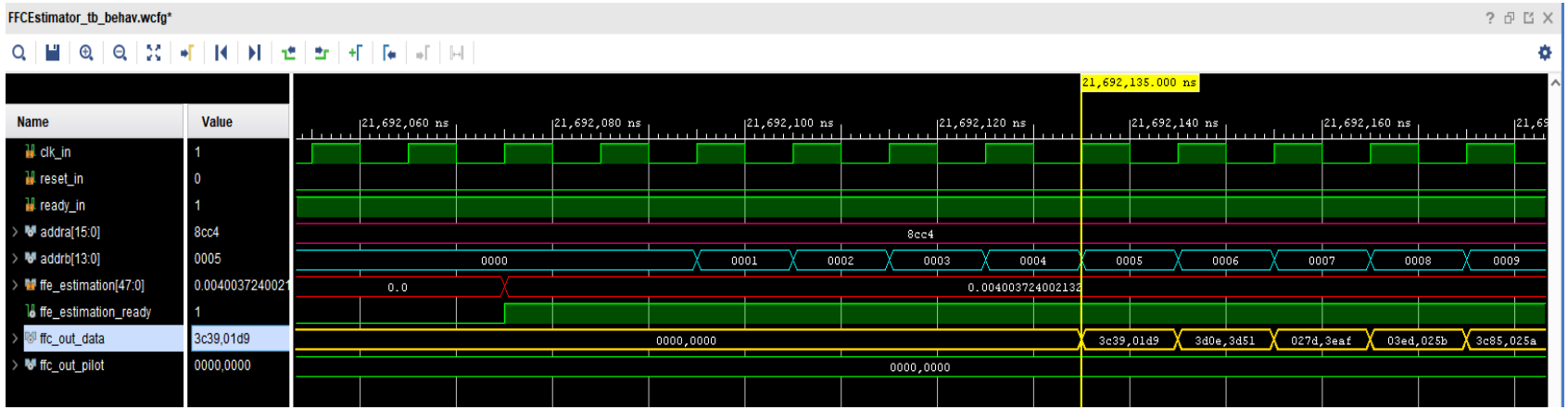

Figura 38 – Simulação comportamental do controlador das memorias ROM e da integração dos sub-módulos FFCCorretor e FFCEstimator

.

Após verificar o funcionamento do controle das memórias durante a simulação comportamental fez-se a síntese e a implementação do circuito. A tabela 12 apresenta a utilização dos recursos do kit de desenvolvimento Zedboard com toda a unidade teste que inclui o ILA, as memórias ROMs e o controlador das memórias. A tabela 13 apresenta a utilização somente das arquiteturas implementadas para o corretor fino de frequência. A tabela 13 apresenta a comparação dos recursos utilizados pela implementação do estimador de desvio de frequência deste documento que apresentou uma menor utilização tanto de LUTs, FFs e DSPs em relação a implementação feita por (SILVA et al., 2015).

Tabela 12 – Utilização de recursos pós implementação da arquitetura de teste físico do módulo corretor fino de frequência.

Recurso	**Estimativa**	**Disponível**	**Utilização** (%)
LUT	10436	53200	19.62
LUTRAM	181	17400	1.04
FF	5685	106400	5.34
DSP	24	220	10.91
BRAM	105	140	75.00

Tabela 13 – Utilização de recursos pós implementação somente do *top level* do módulo corretor fino de frequência.

Arquitetura	**LUTs**	**FFs**	**DSPs**	**BRAMs**
Dvbs2xFFC	8597(16.16%)	3106 (2.92%)	24 (10.91%)	0 (0%)
FFCEstimator	3174	2353	8	0
FFCCorretorPilots	2728	356	8	0
FFCCorretorData	2695	351	8	0

Tabela 14 – Comparação da utilização de recursos da implementação do estimador de frequência do FFC implementado neste documento e o implementado por (SILVA et al., 2015)

Arquitetura	**LUTs**	**FFs**	**DSPs**	**BRAMs**
FFCEstimator	3174	2353	8	0
Estimador de (SILVA et al., 2015)	3559	3851	13	0

O *layout* do teste em hardware é apresentado na figura 39 em amarelo está destacado o ILA, as memórias ROMs e o controlador das memórias; em rosa o módulo que executa a correção dos pilotos; em verde o corretor dos dados e em vermelho o módulo do estimador de frequência. Por fim na figura 40 está presente o resultado do teste em hardware o qual foi utilizado como *trigger* do ILA o sinal de data_ready_out e como resultado obteve-se os dados corrigidos e o valor dos endereços das memórias ROMs. Para verificar a correção do quadro criou-se um *script* em Python para realizar a leitura do

arquivo csv de saída do ILA e com isso criou-se a constelação do quadro corrigido presente na figura 41. A constelação é diferente da gerada pelo Fixlink e do Satlink mostrada anteriormente por conta do estimador, mas a constelação é suficiente para validar o módulo pois é possível observar o desenho esperado para a modulação QPSK testada.

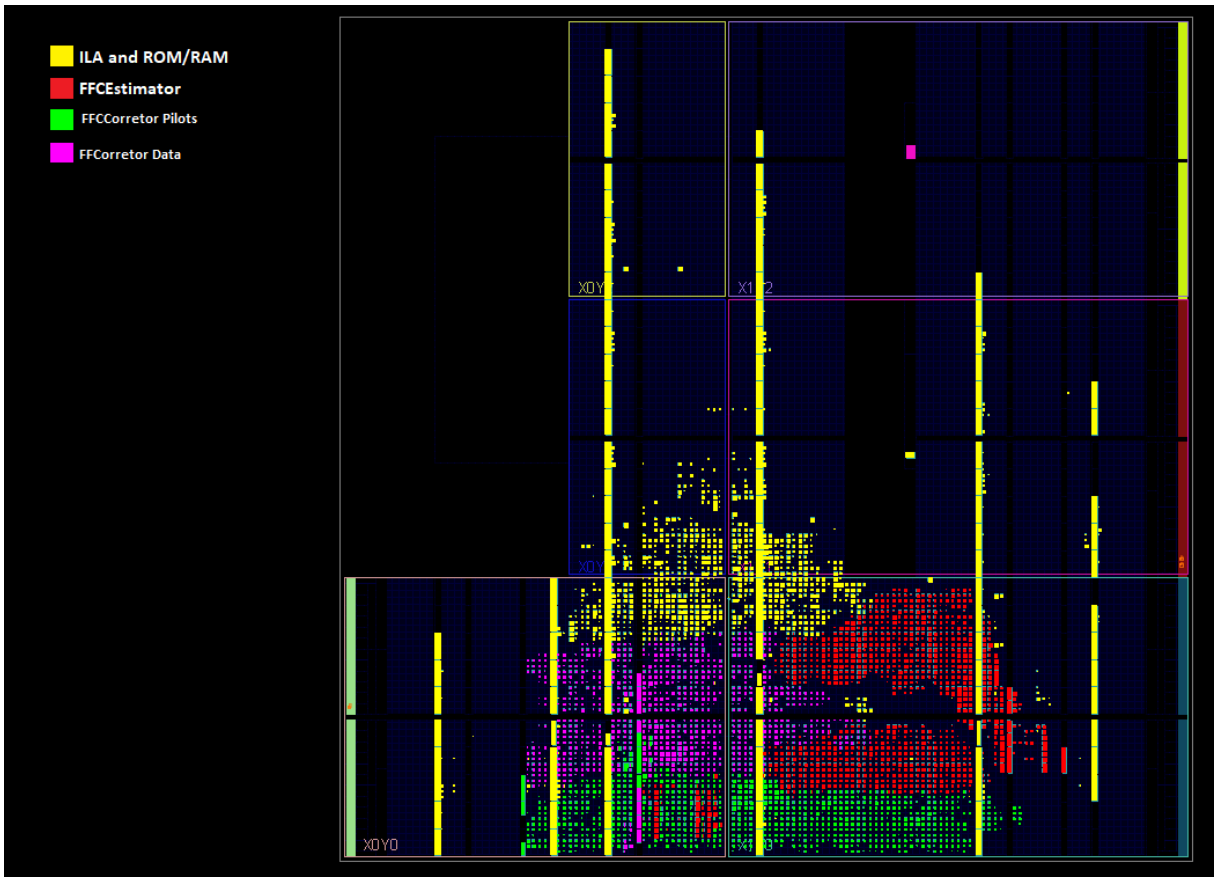

Figura 39 – *Layout* do teste em Hardware no kit de desenvolvimento Zedboard.

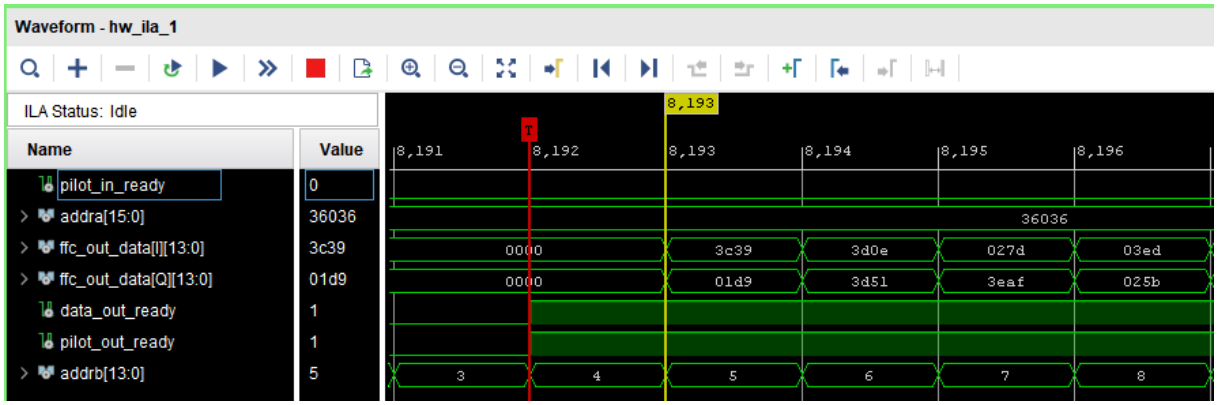

Figura 40 – Resultado do teste em Hardware.

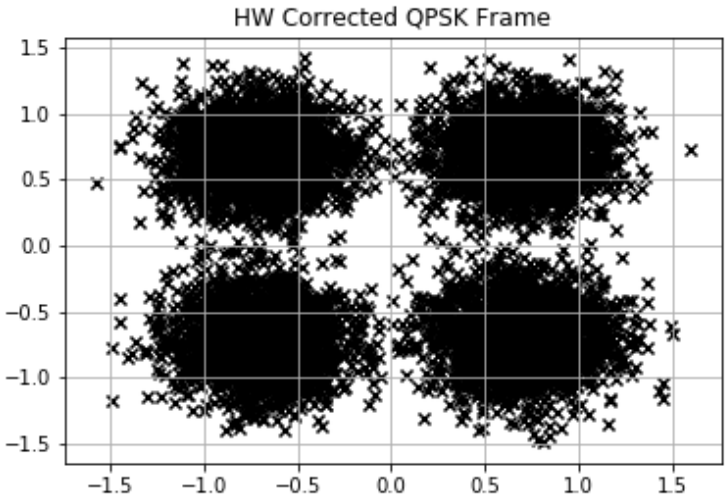

Figura 41 – Constelação corrigida em Hardware.

4.5.3 Integração dos módulos

O primeiro ponto importante sobre o teste de integração foi o teste do PC. O PC não foi testado individualmente pois inicialmente não foi necessário nenhuma alteração no código (desenvolvido por (PINTO, 2020)). Por isso seu desempenho só foi verificado durante a integração dos corretores de frequência e foi durante esse teste que verificou-se a ausência do algoritmo de desdobramento de fase que limita a rotação de fase entre π e $-\pi$.

O teste da integração foi feito sobre o cenário de $Eb/N0 = 80db$, com sobre amostragem de 2 símbolos por amostra, *span* do filtro RRC igual a 10 e desvio de frequência normalizada de 0,2. As análises dos resultados foram feitas com base na visualização das constelações de saída. Foram usados 400 quadros modulados em QPSK. Essa quantidade foi utilizada para garantir a convergência dos módulos CFC, FFC e PC.

Outro ponto importante do teste é que o sub módulo *sintable* presente no CFC, FFC e PC foi desativado durante o teste de integração, pois ao utiliza-lo o tempo de execução do teste estava sendo cerca de 4 horas por quadro. Sendo assim, a passagem de 400 demoraria cerca de 1600 horas (equivalente a mais ou menos dois meses). Ao alterar o sub módulo *sintable* para o calculo de seno e cosseno por meio de funções da biblioteca *numpy*, toda a integração passou a demora menos de 2 horas.

O primeiro ponto observado foi a constelação após o CFC e antes do RRC, que foi deslocado devido a necessidade dos símbolos sobre amostrados na entrada do CFC. Na figura 42 é destacado o efeito da correção feita pelo CFC, a constelação de saída apresenta vários pontos devido ao fator de sobre amostragem que só é retirado após o RRC.

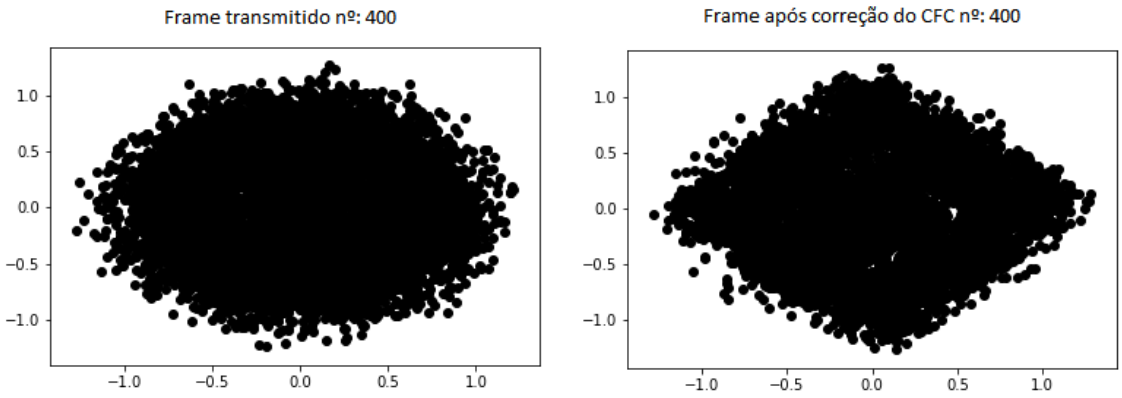

Figura 42 – Comparação entre a constelação do quadro transmitido (Esquerda) com a constelação após passar pelo CFC (Direita). Eb/N0 = 80 db

.

Por fim foi visualizado as constelações de saída de cada etapa de correção de frequência como ilustra a figura 44. A constelação presente na esquerda é a saída após retirar a sobre amostragem no filtro RRC, é possível observar o desenho da constelação

esperada para a modulação QPSK. Contudo a constelação ainda está rotacionada, essa rotação de fase será posteriormente corrigida pelo PC. Na figura 43 está presente o desvio de frequência estimado pelo CFC a cada bloco de pilotos, é importante ressaltar que apesar do desvio aplicado no canal foi de 0,2 o valor estimado convergiu para 0,1. O motivo dessa convergência foi o fator de sobre amostragem dividiu o valor de frequência aplicado entre os símbolos, nesse caso o valor real de desvio de frequência foi divido por dois.

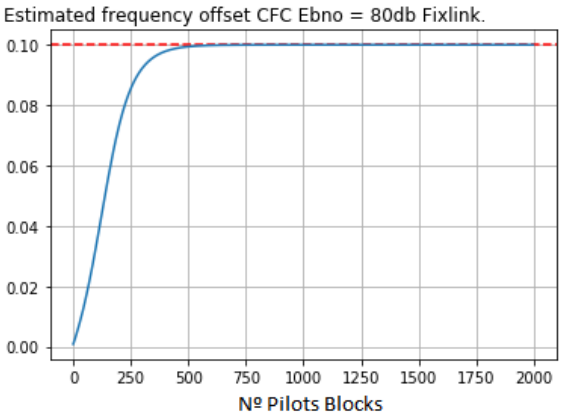

Figura 43 – Desvio de frequência estimado pelo CFC a cada novo bloco de piloto

.

A etapa seguinte da correção foi a aplicação do FFC, apresentada na constelação central da figura 44. Se comparado com a constelação de saída do CFC, não é possível observar nenhuma mudança. Entretanto, na figura 45 se apresentam os desvios de frequência normalizada estimada pelo FFC a cada bloco de piloto, com isso é notado o funcionamento esperado FFC ao diminuir consideravelmente o desvio de frequência residual do CFC. Observa-se ainda que a partir da passagem de 1500 blocos de pilotos o desvio estimado fica muito próximo de 0.

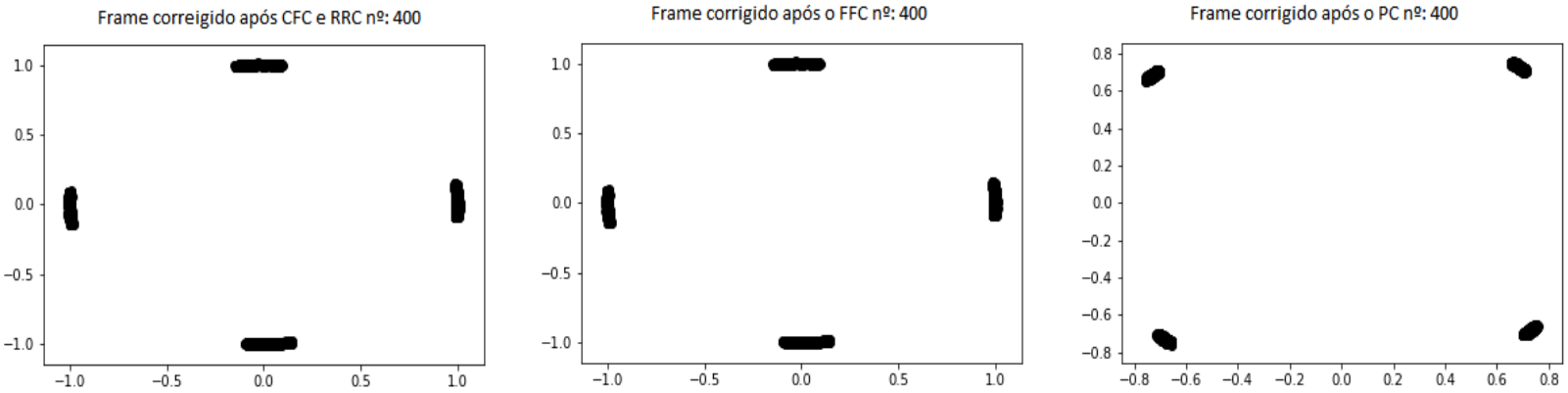

Figura 44 – Constelações de saída da integração dos corretores de frequência, após a correção do CFC (Esquerda) após correção do FFC (Centro) e após correção do PC (Direita). EbN0 = 80 db

.

Por fim na constelação da direita na figura 44 é apresentada a última correção de frequência realizada. Nesta etapa o PC realiza a última rotação na constelação, colocando-a na posição esperada. Essa constelação é observada em todas as saídas a partir do quadro de número 350. Como os testes foram feitos em uma cenário ideal de Eb/N0 de 80db os mesmos testes foram executados com Eb/N0 de 2db. As mesmas constelações dos pontos comentados anteriormente podem ser vistas na figura 46, assim como os mesmos efeitos são vistos. Entretanto, por causa do cenário ruidoso e a ausência do módulo AGC os símbolos das constelações estão mais espalhados o que dificulta a visualização do desenho esperado para a modulação QPSK na última etapa de correção.

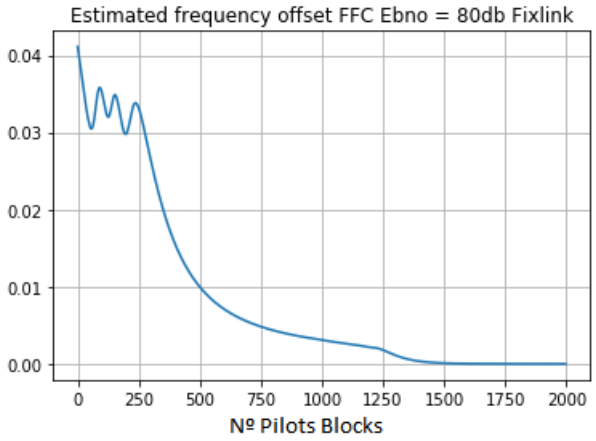

Figura 45 – Desvio de frequência estimado pelo FFC a cada novo bloco de piloto

.

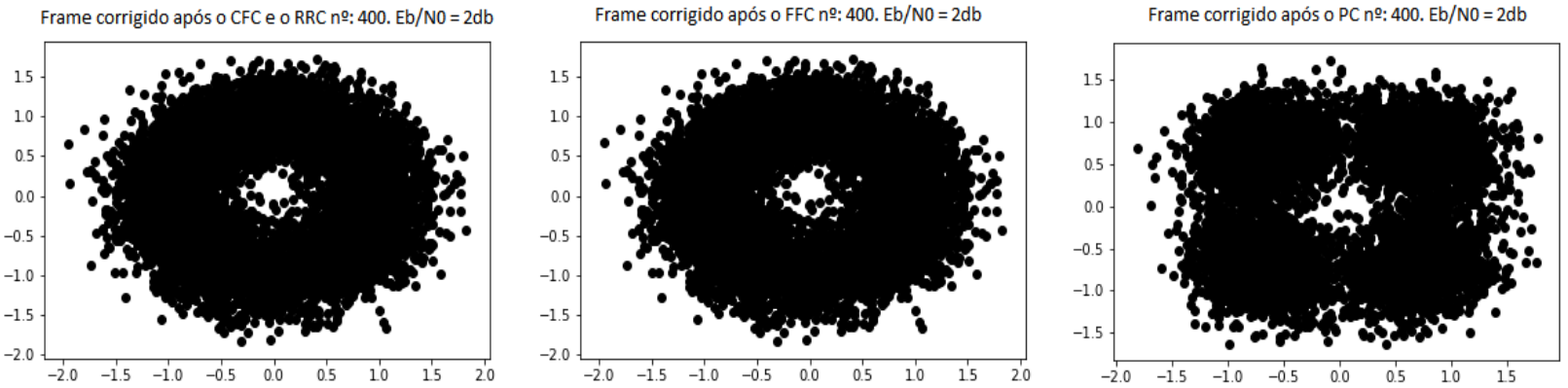

Figura 46 – Constelações de saída da integração dos corretores de frequência, após a correção do CFC(Esquerda) após correção do FFC (Centro) e após correção do PC (Direita). EbN0 = 2 db

.

5 Conclusão e Trabalhos Futuros

A integração do Fixlink até o ponto do PC teve resultados satisfatórios, sendo possível observar a correção completa da constelação de entrada. Contudo, devido a problemas na integração do CFC a CPS não pôde ser integrada por completo no Fixlink restando a integração do AGC e do estimador de SNR.

Adicionalmente, durante o processo de integração do CFC foi descoberto a necessidade da sobre amostragem na entrada desse módulo. Por isso o filtro RRC teve que ser deslocado para depois do CFC e o FrameSync passou a ter uma *flag* de *bypass*, pois o FrameSync não opera com os símbolos sobre amostrados. Com isso é necessário realizar uma investigação maior no módulo CFC para averiguar se o algoritmo usado no estimador de fato só opera com sobre amostragem ou se o problema está na adaptação da correção dos símbolos. Visto que em cenário de integração dos módulos em VHDL a correção dos símbolos é feita pelo transceptor.

A investigação no CFC é necessária principalmente pelo fato que as implementações usadas como referência para a arquitetura da CPS presente em (CASINI; GAUDENZI; GINESI, 2004) e (LIMA et al., 2014) posicionam o RRC e fazem a sub amostragem antes do CFC. Por isso acredita-se que algo não foi levado em consideração durante o desenvolvimento do CFC em VHDL ou até mesmo que a correção usado pelo Satlink, usada como base no Fixlink, não seja a correta para esse módulo.

Além do problema encontrado no CFC, o PC teve que ser modificado com a adição do algoritmo de desdobramento de fase. A importância desse algoritmo só foi notada durante a integração em que foram testados diversos quadros sequencialmente, por isso a ausência do algoritmo na implementação individual do PC. O algoritmo teve que ser implementado no Fixlink, porém esse algoritmo não está presente em VHDL sendo necessário implementá-lo antes de realizar a integração em *hardware*.

Quanto ao *Timming Recovery* os modelos em Python e em VHDL são diferentes e, consequentemente, os resultados são diferentes, motivo pelo qual este módulo também possui uma *flag* de *bypass*. Posteriormente será necessário extrair da implementação em VHDL (TRINDADE, 2019) um modelo alto nível e assim atualizar o modelo em Python, pois só assim será possível o uso desse módulo como referência durante a integração da CPS em VHDL.

O FrameSync apesar da criação da *flag* de *bypass* apresentou excelentes resultados em seu janelamento sem apresentar nenhum comportamento fora do esperado durante os teste de estresse. A identificação falsa dos quadros sempre irá ocorrer quando dois picos de ruído estiverem com a distância igual ao tamanho de um quadro e passarem o

limiar de decisão. Com isso em mente esse cenário só ocorre se for forçado ou se a SNR do canal possuir um valor igual ou inferior a -5 db. Sendo que a operação eficiente do FrameSync só é observada nos pontos iguais ou superiores a 2 db, por isso dificilmente ocorrerá detecções falsas de quadros. Contudo, o ruído gera impactos negativos nos valores dos picos de correlação diminuindo os valores dos picos onde são identificados o SOF e aumentando o valor dos demais, por isso apesar do bom funcionamento no teste de estresse ainda há perdas de quadros pelo FrameSync em valores de Eb/N0 inferiores a 9 db, porém até o ponto de Eb/N0 de 2 db as perdas não são grandes o suficiente para prejudicar por completo o módulo.

Por fim, o Fixlink no estado atual de desenvolvimento serve perfeitamente como um modelo de referência para os módulos individualmente e para toda a integração dos corretores de frequência. Porém ainda é necessário executar as adaptações citadas anteriormente para que o simulador seja utilizado por completo na integração de toda CPS em *hardware*.

Referências

Albertazzi, G. et al. On the adaptive dvb-s2 physical layer: design and performance. *IEEE Wireless Communications*, v. 12, n. 6, p. 62–68, 2005. Citado 4 vezes nas páginas 10, 28, 29 e 30.

CASINI, E.; GAUDENZI, R. D.; GINESI, A. Dvb-s2 modem algorithms design and performance over typical satellite channels. *International Journal of Satellite Communications and Networking*, v. 22, n. 3, p. 281–318, 2004. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.791>. Citado 7 vezes nas páginas 10, 18, 28, 33, 34, 38 e 77.

COMINETTI, M.; MORELLO, A. Digital video broadcasting over satellite (dvb-s): a system for broadcasting and contribution applications. *International Journal of Satellite Communications*, v. 18, n. 6, p. 393–410, 2000. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/1099-1247%28200011/12%2918%3A6%3C393%3A%3AAID-SAT664%3E3.0.CO%3B2-K>. Citado na página 18.

DIGILENT. 2014. Disponível em: <https://digilent.com/reference/programmable-logic/zedboard/start>. Citado na página 45.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. *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: DVB-S2 extensions (DVB-S2X). Europe, 2014. Citado 6 vezes nas páginas 12, 26, 40, 41, 47 e 63.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. *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 1: DVB-S2. Europe, 2014. Citado 6 vezes nas páginas 22, 31, 32, 33, 34 e 35.

GARDNER, F. A bpsk/qpsk timing-error detector for sampled receivers. *IEEE Transactions on Communications*, v. 34, n. 5, p. 423–429, 1986. Citado 2 vezes nas páginas 30 e 46.

Gong, F. et al. Initial-estimation-based adaptive carrier recovery scheme for dvb-s2 system. *IEEE Transactions on Broadcasting*, v. 58, n. 4, p. 654–659, 2012. Citado na página 33.

JOOST, M. Theory of root-raised cosine filter. 2010. Disponível em: <https://www.michael-joost.de/rrcfilter.pdf>. Citado na página 35.

LATHI, B.; DING, Z. *Modern Digital and Analog Communication Systems*. Oxford University Press, 2010. (Oxford series in electrical and computer engineering). ISBN 9780198073802. Disponível em: <https://books.google.com.br/books?id=nmYYNAEACAAJ>. Citado na página 25.

LIMA, E. R. de et al. A detailed dvb-s2 receiver implementation: Fpga prototyping and preliminary asic resource estimation. In: *2014 IEEE Latin-America Conference on Communications (LATINCOM)*. [S.l.: s.n.], 2014. p. 1–6. Citado 5 vezes nas páginas 10, 28, 29, 34 e 77.

MENGALI, U. *Synchronization Techniques for Digital Receivers*. Springer US, 2013. (Applications of Communications Theory). ISBN 9781489918079. Disponível em: <https://books.google.com.br/books?id=w0gGCAAAQBAJ>. Citado na página 32.

Monmasson, E.; Cirstea, M. N. Fpga design methodology for industrial control systems—a review. *IEEE Transactions on Industrial Electronics*, v. 54, n. 4, p. 1824–1842, 2007. Citado na página 18.

Morello, A.; Mignone, V. Dvb-s2: The second generation standard for satellite broad-band services. *Proceedings of the IEEE*, v. 94, n. 1, p. 210–227, 2006. Citado 5 vezes nas páginas 10, 26, 27, 31 e 32.

Mukumoto, K.; Wada, T. Realization of root raised cosine roll-off filters using a recursive fir filter structure. *IEEE Transactions on Communications*, v. 62, n. 7, p. 2456–2464, 2014. Citado na página 30.

PINTO, A. S. R. *Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase aderentes ao protocolo DVB-S2X*. 92 f. Monografia (Trabalho de Conclusão de Curso - Bacharelado em Engenharia Eletrônica) — Universidade de Brasília, Brasília, 2020. Citado 2 vezes nas páginas 46 e 72.

QING, L. et al. Optimal frame synchronization for dvb-s2. In: *2008 IEEE International Symposium on Circuits and Systems*. [S.l.: s.n.], 2008. p. 956–959. Citado na página 31.

RICE, M. *Digital Communications: A Discrete-time Approach*. Pearson/Prentice Hall, 2009. ISBN 9780130304971. Disponível em: <https://books.google.com.br/books?id=EB3r7JtXlWwC>. Citado 2 vezes nas páginas 30 e 32.

SILVA, G. da et al. A novel fine frequency estimation serial architecture applied in satellite communications. In: . [S.l.: s.n.], 2015. Citado 7 vezes nas páginas 10, 12, 21, 22, 23, 38 e 69.

SKLAR, B.; HARRIS, F. *Digital Communications: Fundamentals and Applications*. Pearson Higher Education & Professional Group, 2020. (Communications Engineering & E). ISBN 9780134588568. Disponível em: <https://books.google.com.br/books?id=SvI4tgEACAAJ>. Citado 3 vezes nas páginas 10, 25 e 26.

SUN, F.-W.; JIANG, Y.; LEE, L.-N. Frame synchronization and pilot structure for second generation dvb via satellites. *International Journal of Satellite Communications and Networking*, v. 22, n. 3, p. 319–339, 2004. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.793>. Citado na página 27.

TRINDADE, P. H. A. *DVBS2X standard FPGA implementation of LLR, physical layer de-scrambling and symbol timing synchronization (STR) blocks for satellite applications*. 98 f. Monografia (Trabalho de Conclusão de Curso - Bacharelado em Engenharia Eletrônica) — Universidade de Brasília, Brasília, 2019. Citado 3 vezes nas páginas 46, 54 e 77.

Apêndices

APÊNDICE A – Esquemático RTL do top level do FFC

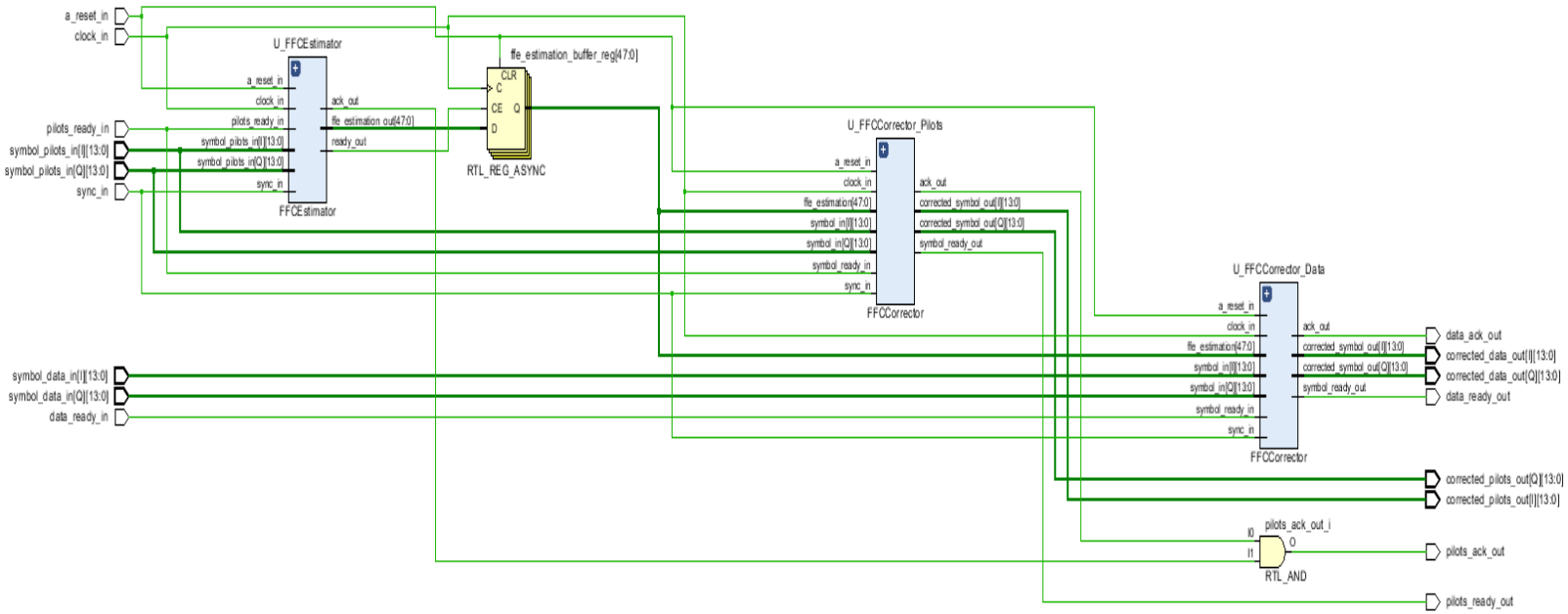

Figura 47 – Esquemático RTL do top level do FFC

.

APÊNDICE B – Esquemático RTL do teste em Hardware

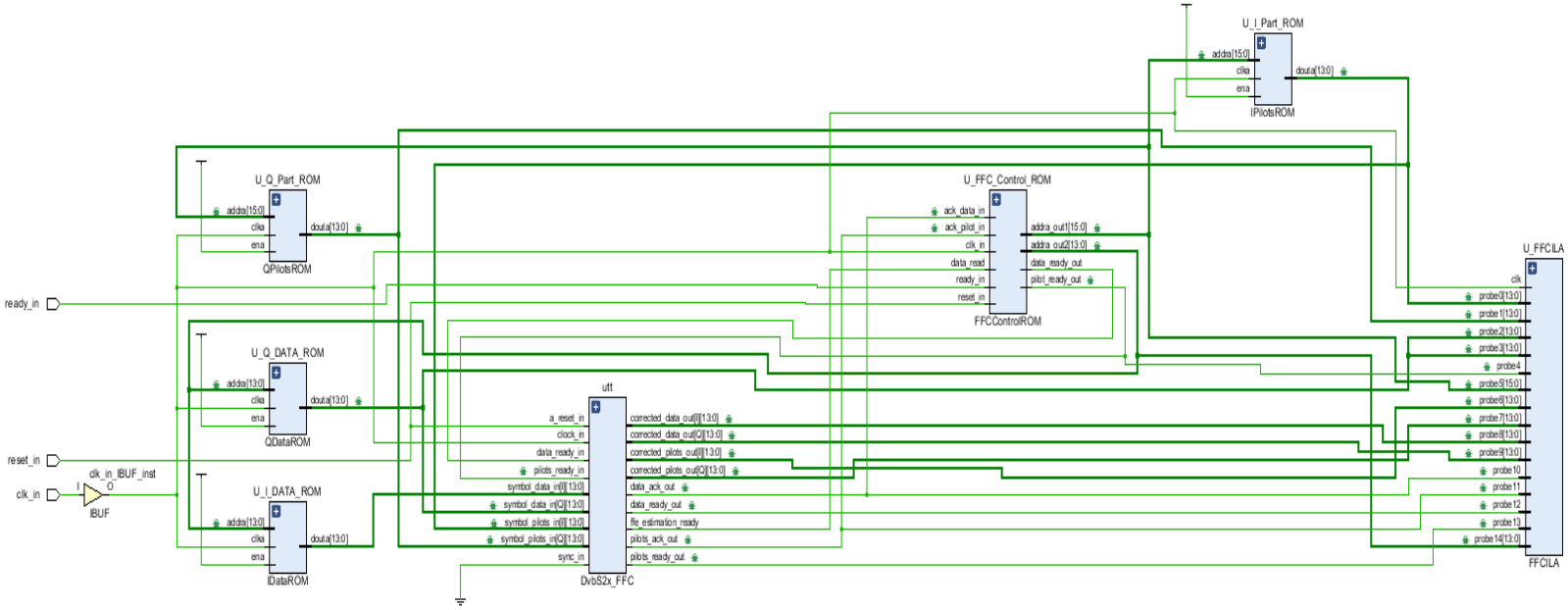

Figura 48 – Esquemático RTL do teste em Hardware

.