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

Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase aderentes ao protocolo DVB-S2X

Autor: Anderson Sales Rodrigues Pinto
Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF
2020

Anderson Sales Rodrigues Pinto

Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase 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

2020

Anderson Sales Rodrigues Pinto

Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase aderentes ao protocolo DVB-S2X/ Anderson Sales Rodrigues Pinto. – Brasília, DF, 2020-

91 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 , 2020.

1. Hardware Reconfigurável. 2. Sincronismo de Frequência e Fase. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase aderentes ao protocolo DVB-S2X

CDU 02:141:005.6

Anderson Sales Rodrigues Pinto

Implementação em FPGA dos módulos correção grosseira de frequência e correção de fase 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, 23 de novembro de 2020:

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Leonardo Aguayo
Convidado 1

Prof. Dr. Gilmar Silva Beserra
Convidado 2

Brasília, DF
2020

Dedico este trabalho de fim de curso à todos meus familiares, em especial a minha tia (in memorian) e minha avó(in memorian) que muito me apoiaram e se orgulharam de mim por ter entrado na UnB.

Agradecimentos

Gostaria de agradecer primeiramente a Deus por me ter dado vida e forças nos momentos em que mais precisei durante a minha jornada pela Universidade de Brasília. Graças a Ele e a muitas pessoas ao meu redor pude realizar minha graduação com sucesso.

Agradeço aos meus pais Adilson Rodrigues Pinto e Maria Lucineide Sales Pinto por sempre me apoiarem durante minha graduação, bem como minhas irmãs e meus outros familiares, que sempre me incentivaram a estudar cada vez mais e que com certeza estão e sempre estarão orgulhosos de mim ao fim da graduação.

Também agradeço aos meus grandes amigos e colegas que fiz durante minha graduação, seja com quem fiz dupla ou grupo, ou apenas colegas de outros cursos, que com certeza ajudaram e muito neste período, com muitos momentos de descontração que fizeram com que a graduação fosse muito menos estressante. Agradeço à minha amiga e companheira Luísa Caroline, por estar sempre comigo me apoiando em tantos momentos difíceis que passei pela graduação.

Agradeço à Universidade de Brasília e aos seus servidores, onde tive a oportunidade de usufruir do seu espaço de trabalho e aprendizado, bem como o corpo docente, que com certeza é um dos melhores que já vi, em especial meu orientador Prof. Dr. Daniel Muñoz, sempre muito atencioso e prestativo na orientação do meu TCC. Agradeço também a minha eterna professora de Prática de Eletrônica Digital 1 Profª. Drª. Mariana Bernades, com quem tive a honra de sempre ajudar como monitor na disciplina em praticamente todos os semestres.

Gostaria também de agradecer aos técnicos do laboratório NEI, que sempre que precisei estavam lá de prontidão para me ajudar e para ajudar meus amigos e colegas de curso, o que sempre me inspirou também em ajudar os outros na minha graduação.

Agradeço à Autotrac por ter me dado oportunidade de realizar meu PIBIC, o que me concedeu uma vasta experiência profissional e acadêmica, que com certeza levarei comigo pelo resto da vida. Agradeço ao engenheiro Zoe Magalhães por ter me ajudado bastante durante meu período na Autotrac.

Por fim, mais uma vez, agradeço a todos os amigos, colegas e professores da Universidade de Brasília, e espero que este trabalho, no futuro, seja de grande serventia para todos aqueles que, assim como eu, estejam nessa jornada da graduação no curso de Engenharia Eletrônica.

"Se eu vi mais longe, foi por estar sobre ombros de gigantes."
(Isaac Newton)

Resumo

O protocolo DVB-S2X é uma extensão do protocolo DVB-S2, que é a segunda geração do protocolo de comunicação DVB-S, muito utilizado em transmissão de televisão digital por satélite, mas que também pode ser aplicado a serviços interativos com clientes e até em aplicações mais profissionais, como rastreamento de frotas por exemplo. Sob as mesmas condições de transmissão este protocolo tem um desempenho típico 30% maior em relação ao seu protocolo anterior(Morello; Mignone, 2006). Neste protocolo, assim como em outros, há uma parte de recepção e processamento dos dados recebidos, de forma que eles sejam decodificados corretamente, com o mínimo de ruído possível. Nesta etapa de processamento, uma das etapas mais importantes é justamente o processo de sincronismo de frequência e fase da portadora, que busca corrigir os desvios de frequências acarretados principalmente pelas diferenças entre as frequências dos osciladores do transmissor e do receptor, e por conta do efeito Doppler, que também causa um desvio cumulativo de fase. Para solucionar estes problemas é preciso a implementação de um módulo corretor de frequência, cujo funcionamento, neste caso em específico, é análogo a um PLL, bem como a implementação de um módulo corretor de fase. Aplicado ao protocolo DVB-S2X, estas etapas de sincronismo de frequência e fase se dividem em uma mais grosseira e uma mais fina (ETSI, 2014a). Neste trabalho será apresentada a implementação em hardware reconfigurável dos módulos de correção grosseira de frequência e de correção de fase. As arquiteturas de hardware fazem uso de aritmética de ponto fixo e foram validadas inicialmente a nível de simulação comportamental e posteriormente através de uma implementação física em um dispositivo FPGA usando diversos valores de desvio de fase e de frequência. No caso particular do módulo corretor de frequência foi feito um teste em malha aberta. Os circuitos implementados foram caracterizados obtendo, para o módulo de Correção Grosseira de Frequência: um desvio mínimo e máximo suportado de 0.02 e 0.2, respectivamente; erro RMSE abaixo de 0.05; frequência de operação de 100MHz; bom consumo de recursos do módulo; consumo de energia baixo (0,091W); valores de latência e taxa de transmissão de 90ns e 25MSPS, respectivamente; nenhum problema de temporização; módulo ainda em malha aberta, porém foi implementado e validado na FPGA utilizando ILA e memórias ROMs. Para o módulo de Correção de Fase: O RMSE maior encontrado foi para a modulação 16-APSK, de aproximadamente $3,545.10^{-4}$; a frequência máxima de operação foi também de 100MHz; o consumo de recursos ficou um pouco elevado; latência e taxa de transmissão de 1780ns e 25 MSPS, respectivamente; consumo de energia baixo (0,091W); nenhum problema com temporização e módulo completo implementado e validado em FPGA por meio de memórias ROMs e LEDs.

Palavras-chaves: DVB-S2x. DVB-S2. Arquitetura Reconfigurável. Sincronismo de frequência. Sincronismo de Fase. FPGA. VHDL.

Abstract

The protocol DVB-S2X is an extension of the DVB-S2 protocol, which is the second generation of the DVB-S2 protocol, very used in digital TV satellite transmission, but that can be also applied in clients iteractive services and even in professional applications, like fleet tracking for example. Under same transmission conditions, this protocol has a typical performance 30% bigger than its previous protocol (Morello; Mignone, 2006). In this protocol, just like others, there is a receiver and signal processing part, so that they can be decoded rightly, with minimal possible noise. In this processing stage, one of the most important stages is precisely the carrier frequency and phase synchronization, that seeks to correct the frequency deviations caused mainly by frequencies differences between the transmitter and receiver oscillators, and by the Doppler effect, that also cause a cumulative phase deviation. To solve these problems it's necessary an implementation of a frequency correction module, whose operation, in this specific case, is analogous to a PLL, as well as the implementation of a phase correction module . Applied to this protocol, these frequency and phase synchronization steps are divided in a coarse one and in a fine one (ETSI, 2014a). In this document will be presented the hardware implementation of Coarse Frequency Correction and Phase Correction modules, Their hardware architecture uses fixed point arithmetic and they were validated initially at behavioral simulation, and after that they were validated using a phisical implementation in a FPGA, using many different values of phase and frequency offsets. In particular, the Coarse Frequency Correction module it was made an open loop test. The implemented circuits were tested, obtaned the following results for the Coarse Frequency Correction module: a minimum and maximum frequency offset of 0.02 and 0.2, respectively; RMSE below 0.05; frequency operation of 100MHz; good resources consumption; low power consumption (0.091W); latency and throughput values of 90ns and 25 MSPS, respectively; no timing problems and even in open loop, it was implemented and validated in FPGA using ILA and ROM memories. In the case of Phase Correction: the greather RMSE found was for the 16-APSK modulation, that is nearly $3.545.10^{-4}$; maximum frequency operation is 100MHz; resources consumption was pretty high; lantecy and throughput values of 1780ns and 25 MSPS, respectively; low power consumption (0.091W); no timing problems and the module was fully implemented and validated in FPGA using ROMs and LEDs.

Key-words: DVB-S2x. DVB-S2. Reconfigurable Architecture. Frequency Synchronization. Phase Synchronization. FPGA. VHDL.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

8-APSK	*Eight Amplitude and Phase-Shift Keying*
8-PSK	*Eight Phase Shift Keying*
16-QAM	*Sixteen Quadrature Amplitude Modulation*
32-APSK	*Thirty Two Amplitude and Phase-Shift Keying*
ADC	*Analog-Digital Converter*
AGC	*Automatic Gain Control*
BCH	*Bose–Chaudhuri–Hocquenghem*
BPSK	*Binary Phase Shift Keying*
CFC	*Coarse Frequency Correction*
CORDIC	*COordinate Rotation DIgital Computer*
DDC	*Digital Down Converter*
DDS	*Direct Digital Synthesis*
DM	*Delay-Multiply*
DVB-S	*Digital Video Broadcasting - Satellite*
DVB-S2	*Digital Video Broadcasting - Satellite - Second Generation*
DVB-S2X	*Digital Video Broadcasting - Satellite - Second Generation Extensions*
FED	*Frequency Error Detector*
FFC	*Fine Frequency Correction*
ILA	*Integrated Logic Analyzer*
IP	*Intellectual Property*
LDPC	*Low-Density-Parity-Check*
L&R	*Luise & Reggiannini*
ML	*Maximum Likelihood*

M&M	*Morelli & Mengali*
MSPS	*Mega Símbolo Por Segundo*
SoC	*System on Chip*
PC	*Phase Correction*
PI	Proporcional Integrador
PLL	*Phase-Locked Loop*
QPSK	*Quadrature Phase-Shift Keying*
RMS	*Root-Mean-Square*
RMSE	*Root-Mean-Square Error*
RRC	*Root-Rised-Cosine*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

Lista de símbolos

dB	Decibel
Es/No	Razão da energia do símbolo pela energia do ruído
SnR	Relação Sinal-Ruído
θ	Ângulo do desvio de fase
σ	Direção da rotação do vetor
ζ	Fator de amortecimento

Sumário

1 Introdução

1.1 Contextualização

O protocolo DVB-S2X é uma extensão do protocolo DVB-S2, que é um protocolo utilizado principalmente transmissão satelital de televisão digital, possuindo um rendimento de até 30% em relação ao seu antecessor, o protocolo DVB-S (Morello; Mignone, 2006). O protocolo DVB-S2X permite uma transmissão adaptativa, seja alterando a taxa de código, o tipo de modulação ou o tipo de frame (quadro). Também possui um sistema bastante robusto de correção de erros, utilizando LDPC concatenado com código de BCH, que permite operar entre 0,7dB a 1dB do limite de Shannon (Shannon, 1949). Com o protocolo DVB-S2X, pode-se trabalhar com sinais de *SNR* muito baixa, da ordem de -10dB (ETSI, 2014b).

Devido a sua capacidade de operação com um *SNR* tão baixo, o protocolo DVB-S2X também pode ser utilizado em outros tipos de aplicação, como no mercado de aviação civil, no mercado marítimo, em terminais VSAT de alta frequência, ou em até pequenos terminais portáteis para jornalistas e outros profissionais. Outra aplicação bastante pertinente para a qual o DVB-S2X pode ser utilizado é no rastreamento de frotas por satélite. Neste tipo de aplicação o protocolo DVB-S2X poderia ser utilizado em toda ou em parte do sistema de comunicação, seja na parte de transmissão ou na parte de recepção.

No entanto, em todo sistema de telecomunicação, há a necessidade de se processar os sinais recebidos pelo transmissor, a fim de se extrair ao máximo os ruídos injetados ao sinal transmitido. Devido a ruídos existentes no meio em que os dados são transmitidos, características importantes do sinal, como a frequência e a fase da portadora, podem ser comprometidas durante o seu trajeto, assim dificultando ou até mesmo impossibilitando a decodificação correta deste sinal. No protocolo DVB-S2X o desvio de frequência inicial chega em $\pm$ 5MHz, o que representa 20% da taxa de símbolo a 25Mbaud (CASINI; GAUDENZI; GINESI, 2004). O desvio de frequência inserido no sinal transmitido também acarreta em um desvio cumulativo de fase, sendo também necessário o sincronismo de fase do sinal recebido. Logo, é preciso que se faça um sincronismo de frequência e de fase da portadora, de forma que a decodificação do sinal recebido seja feita de forma correta.

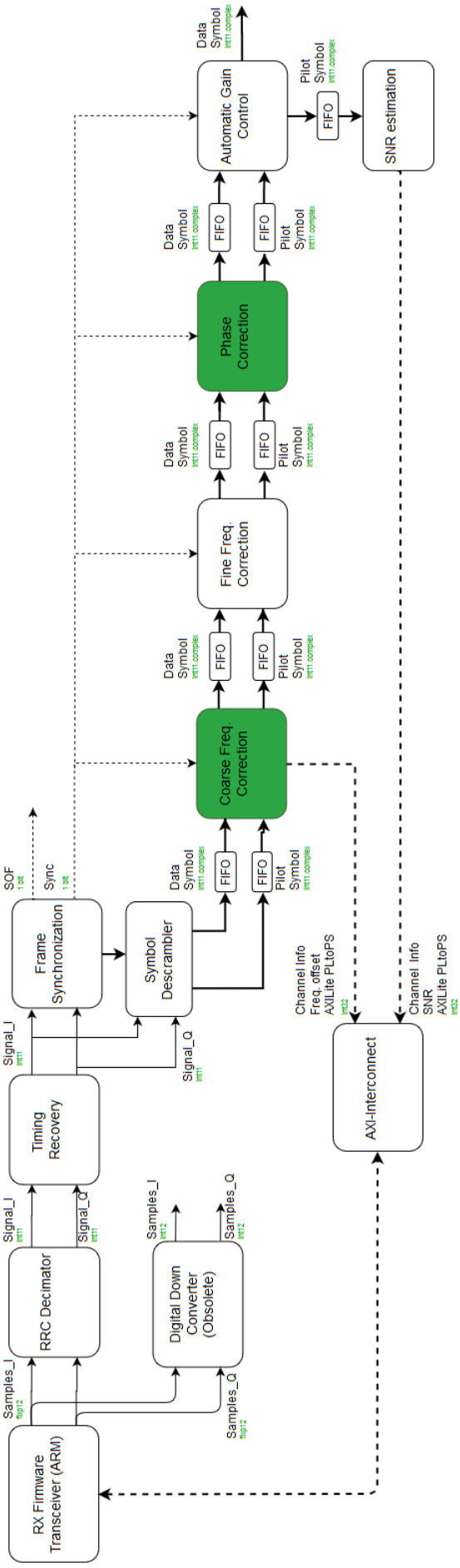


Figura 1 – Arquitetura da cadeia de sincronização de símbolos. Em verde, os módulos Corretor Grosseiro de Frequência e Corretor de Fase.

Na figura anterior (1) é possível observar que a cadeia de sincronização de símbolos inicia-se com o *Transceiver* (Transceptor), que realiza a conversão do sinal para banda base, onde logo após tem-se o filtro RRC (*Root-Rised-Cosine*), utilizado para filtrar o sinal recebido já em banda base. Após a filtragem tem-se o bloco de sincronismo de tempo (*Timing Recovery*) para realizar a correção de tempo do sinal de entrada. A seguir apresenta-se o sincronizador de quadros (*Frame Synchronization*), módulo responsável por realizar a sincronização de quadro do sinal, informando o tamanho do quadro, modulação e taxa de código. A seguir tem-se o *Symbol Descrambler*, que recebe os sinal de entrada e separa-o em símbolos de dados e de pilotos. Logo após esta etapa tem-se a primeira correção a ser feita, que é a Correção Grosseira de Frequêcia (*Coarse Frequency Correction*), a primeira etapa de correção de frequência da portadora. A seguir tem-se o módulo de Correção Fina de Frequência (*Fine Frequency Correction*), usado para corrigir os desvios de frequência remanescentes do módulo anterior. Para concluir o sincronismo de frequência, tem-se o módulo de Correção de Fase (*Phase Correction*), que corrige os desvios de fase acumulados em detrimento dos desvios de frequência acometidos ao sinal de entrada. Por fim, tem-se os módulos de AGC (*Automatic Gain Control* e o *SNR Estimation*, sendo o primeiro o responsável por realizar a correção de ganho do sinal e o segundo por estimar o ganho do mesmo.

Neste trabalho de conclusão de curso será apresentado um modelo de correção grosseira de frequência e um modelo de corretor de fase, destacados em verde na figura 1. O primeiro é uma das etapas de sincronismo de frequência existentes do protocolo DVB-S2X, e o segundo sendo a etapa de correção de fase do mesmo protocolo, que é também o protocolo base utilizado para a implementação dessas correções de frequência e fase.

1.2 Descrição do Problema

Quando se trabalha com sistemas de rastreamento satelital e em outros tipos de sistemas de telecomunicações, deve-se garantir que o sinal enviado chegue ao receptor com a menor taxa de ruído possível, assim como o receptor deve, na medida do possível, processar esse sinal de forma que os ruídos inseridos nele sejam retirados ao máximo, para que a decodificação do sinal seja feita de forma mais otimizada.

A etapa de tratar o sinal recebido com o objetivo de minimizar o ruído que foi inserido nele se chama de Sincronização de Símbolos, onde ocorrerão várias etapas de tratamento do sinal, como por exemplo a conversão de frequência intermediária para frequência de banda base, sincronização do tempo, correção de frequência, correção de fase, dentre outras. Uma das etapas mais importantes do processamento do sinal é justamente a sincronização da frequência da portadora, que na grande maioria dos sistemas de telecomunicações sempre vem com um desvio de frequência em relação ao transmissor.

Há dois principais motivos para que isso aconteça. O primeiro deles é justamente que, em grande parte dos sistemas de comunicação, os osciladores das portadora do transmissor e receptor não são os mesmos, por mais que os circuitos construídos tenham tido as mesmas especificações dificilmente terão a mesma frequência de oscilação. O segundo motivo é o efeito Doppler, que causa um desvio de frequência do sinal a medida que o receptor se afasta ou se aproxima do transmissor.

Para que este problema seja corrigido é necessário que em todo receptor tenha algum módulo de correção de frequência da portadora. Este módulo tem por objetivo rastrear os desvios de frequência da portadora e corrigi-la, de forma que a decodificação do sinal não seja comprometida. Muitas vezes esta etapa de correção da frequência da portadora se divide em dois módulos: o módulo de correção mais grosseira e outro de correção mais fina.

No entanto, os desvios de frequência da portadora também fazem com que ocorra um desvio cumulativo de fase, sendo também necessária a correção destes desvios utilizando um módulo de correção de fase. Esta etapa de correção de fase é feita apenas por um módulo, posteriormente aos módulos de correção de frequência.

No presente documento apresenta-se a implementação dos módulos de correção grosseira de frequência e correção de fase , ambos em linguagem de descrição de hardware, seguindo as normas do protocolo de comunicação satelital DVB-S2X.

1.3 Justificativa

Hardwares reconfiguráveis são conhecidos por permitirem a paralelização nas implementações de algoritmos, permitindo alcançar maiores taxas de processamento em comparação as implementações em software, assim como também permite a atualização deste mesmo hardware futuramente. Logo, por estes motivos, será utilizada uma FPGA para a implementação do CFC e do PC.

Os algoritmos a se utilizar para a implementação dos módulos de correção grosseira de frequência e de correção de fase devem ser algoritmos que não devem consumir muitos recursos da FPGA, evitando-se a utilização de operações que demandem muitos recursos e alto custo computacional. Assim, foi escolhido o algoritmo de laço de realimentação para a implementação em hardware do CFC. Para a implementação do PC, foi escolhido os algoritmos de Máxima Verossimilhança (ML) e de Interpolação Linear. Ambos os algoritmos evitam o uso de operações de divisão.

O algoritmo selecionado para o CFC necessita de um filtro de segunda ordem para poder suavizar suas estimativas de erros de frequência e fazer com que o sistema convirja e entre em regime permanente. Sendo assim, foi escolhido um filtro PI, dado que sua

implementação em hardware não utiliza muitos recursos.

Para suavizar ainda mais as estimativas de erro, é necessário que as saídas do filtro PI sejam novamente filtradas, porém não com um filtro tão robusto, mas sim com um filtro de média móvel.

De forma a corrigir os símbolos, é necessário que seja utilizado algum bloco que transforme essas estimativas de erros de frequência em valores de senos e cossenos. O bloco que se encarregará das correções de frequência será o bloco de Transceiver (o primeiro bloco da figura 1), que possui um DDS (*Direct Digital Synthesis*) interno.

Já com relação ao módulo PC, os algoritmos escolhidos necessitam do cálculo de ângulo. Sendo assim, uma ótima escolha para calcular estes ângulos será a utilização de um *CORDIC* (*COordinate Rotation DIgital Computer*), um algoritmo iterativo baseado em somas e substrações muito utilizado em circuitos digitais.

A correção de símbolos do PC se dará por uma multiplicação complexa, que rotaciona os símbolos. Logo tem-se a necessidade de implementar todo um bloco Corretor de Fase, capaz de calcular os valores de seno e cosseno do desvio de fase estimado e realizar a correção do símbolo de entrada. Como pode ser um símbolo de dado ou de piloto, este módulo deve ser generalizado, de forma que seja possível utilizá-lo tanto para corrigir os símbolos de dados quanto de pilotos.

1.4 Objetivos

O objetivo deste trabalho é implementar um corretor grosseiro de frequência (CFC) e um corretor de fase (PC) utilizando linguagem de descrição de hardware, bem como validá-los por meio de modelos de referências em linguagens de alto nível, como Python e Matlab, assim como realizar seus testes de implementação física por meio de analisadores lógicos.

1.5 Objetivos Específicos

1.5.1 *Coarse Frequency Correction*

- Desenvolvimento e implementação em VHDL do bloco de Detector de Erros de Frequência (FED), de forma que os desvios de frequência sejam estimados para serem posteriormente corrigidos.
- Desenvolvimento e implementação em VHDL do bloco do Filtro PI, de forma que os desvios de frequência detectados pelo FED sejam corrigidos.

- Desenvolvimento e implementação em VHDL do bloco do filtro de Média Móvel, utilizado para suavizar as saídas do Filtro PI e fazendo com que o desvio de frequência residual seja menor.
- Integração do CFC com o bloco DDS, localizado internamente no Transceiver, por meio do barramento AXI-DMA, de forma que as correções sejam aplicadas aos sinais em fase e quadratura.
- Testes de bancada recebendo com vetores de teste com diversos desvios de frequência.

1.5.2 *Phase Correction*

- Desenvolvimento e implementação em VHDL do bloco de Estimação de Fase, subdivididos em três sub-blocos, cada um realizando uma etapa da estimação do desvio de fase para os símbolos de entrada.
- Desenvolvimento e implementação em VHDL do bloco Corretor de Fase parametrizável, de forma que seja possível reutilizá-lo para corrigir tanto os símbolos de dados quanto os símbolos de pilotos.
- Testes de bancada recebendo diferentes modulações.

1.6 Organização do documento

O restante deste documento esta organizado da seguinte maneira: O capítulo 2 versa sobre a fundamentação teórica, onde serão apresentados conceitos do protocolo DVB-S2X, de correção grosseira de frequência e de correção de fase, assim como o estado da arte, que apresenta trabalhos onde o corretor grosseiro de frequência e o corretor de fase já foram implementados. O capítulo 3 trata sobre toda a metodologia envolvida na implementação dos módulos CFC e PC, como os blocos que serão implementados, a forma de validação, as definições de arquitetura, proposta de desenvolvimento, o kit de desenvolvimento e a proposta de caracterização dos módulos. Já no capitulo 4 serão apresentados os resultados obtidos pela simulação e validação dos módulos aqui implementados. Por fim, tem-se as conclusões do trabalho, onde serão discutidos os resultados e trabalhos para o futuro.

2 Fundamentação Teórica

No presente capítulo serão abordados conceitos fundamentais para a implementação do corretor grosseiro de frequência e para a implementação do corretor de fase, sendo eles o protocolo DVB-S2X, o bloco sincronização de símbolo, *SoC*, hardware reconfigurável e os proriamente ditos CFC e PC. Além disso serão apresentados alguns trabalhos sobre implementações semelhantes destes módulos.

2.1 Protocolo DVB-S2X

O protocolo DVB-S2X é uma evolução do protocolo DVB-S2, que por sua vez é um protocolo de comunicação via satélite para transmissão de televisão digital. Neste mais novo protocolo ocorreram várias melhorias, dentre elas aplicações VSAT (*Very Small Aperture Terminal*), DSNG (*Digital Satellite News Gathering*) e DTH (*Direct To Home*). O protocolo DVB-S2X também permite que as transmissões sejam adaptadas em termos de taxa de código, tipo de quadro, tipo de modulação, entre outras características, a fim de economizar energia na etapa de recepção.

Por se tratar não de um novo protocolo, mas sim de uma extensão, o protocolo DVB-S2X também compartilha de várias características do protocolo DVB-S2, como o uso de códigos LDPC(*Low-Density Parity Check*) concatenado com códigos de BCH (*Bose Chaudhuri Hocquenghem*), permitindo operações QEF (*Quasi-Error-Free*) entre 0,7dB e 1dB do limite de Shannon, a depender do modo de transmissão.

Possui também uma maior gama de taxa de código (de $\frac{1}{4}$ a $\frac{9}{10}$ e 4 constelações, variando a eficiência de espectro entre 2bit/s/Hz até 5bit/s/Hz (ETSI, 2014a). O uso de modulações como QPSK, 8-PSK, 16-APSK e 32-APSK o que aumenta sua capacidade de transmissão em até 30% em relação a versão DVB-S (Morello; Mignone, 2006).

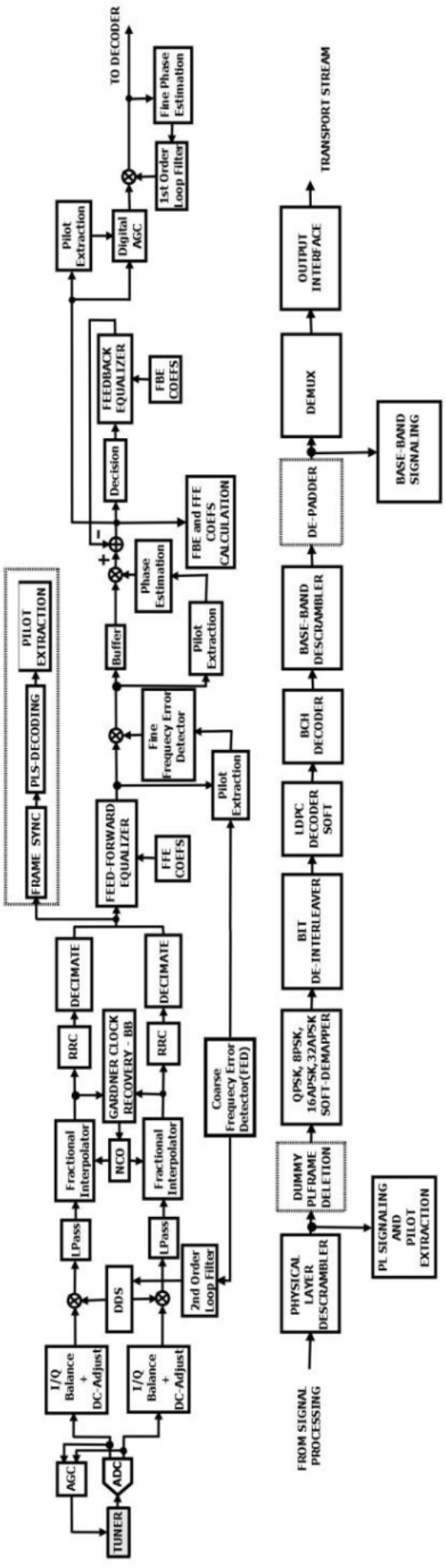


Figura 2 – Arquitetura básica do receptor DVB-S2X. Retirado de (LIMA et al., 2013)

Na arquitetura de recepção do protocolo DVB-S2X temos componentes básicos para a decodificação do sinal como um conversor ADC, utilizado para amostrar o sinal recebido e mais para frente todo um conjunto de blocos de filtros, corretores de frequência e fase e estimação da relação sinal-ruído. Depois deste módulo o sinal processado irá direto para o módulo de decodificação, onde ele será finalmente decodificado por meio de desmapeadores de constelações, decodificadores LDPC, decodificadores BCH, dentre outros módulos, a fim de finalmente o sinal recebido ser recebido pela interface de saída.

O tipo de sinal recebido por este receptor é baseado em um FECFRAME, que é um conjunto de símbolos que possui S slots cada um com 90 símbolos.

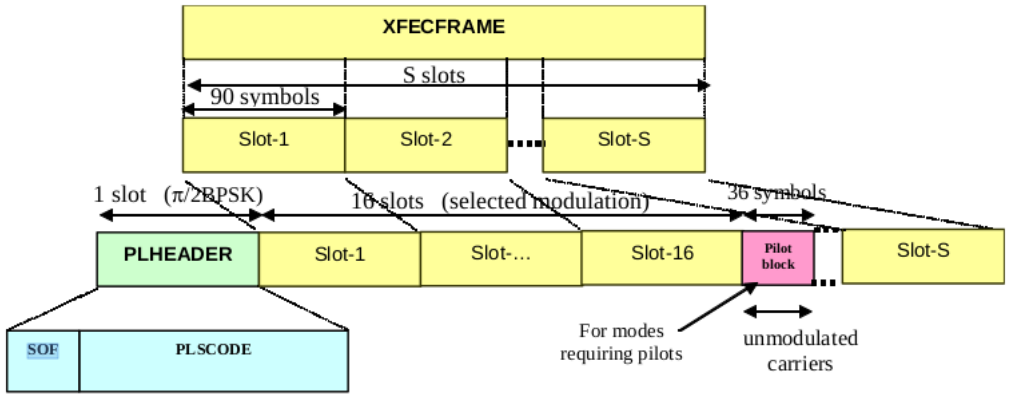


Figura 3 – Estrutura do frame (quadro) do protocolo DVB-S2X. Retirado de (ETSI, 2014a).

Cada um destes slots possuem um cabeçalho (*PLHEADER*) e em sequência possuem vários slots de dados (dependentes do tipo de modulação), sendo que a cada 16 blocos de dados temos logo em seguida um bloco de 36 símbolos-pilotos, que serão utilizados para facilitar a sincronia de fase, tempo e frequência no receptor.

No cabeçalho do quadro também temos um bloco chamado de início de quadro (*Start of Frame - SOF*), seguido de um bloco de sinalização de código da camada física (*Physical Layer Signalling code - PLSCODE*). SOF é um bloco que possui um padrão de 26 símbolos, ao passo que o bloco do PLSCODE possui 64 símbolos, totalizando os 90 símbolos que cada slot é composto. Para que este quadro seja devidamente recebido e processado há a necessidade de todo um módulo dedicado a esta tarefa, que é o módulo de Sincronização de Símbolos, onde ocorrerão as mais diversas etapas de processamento deste quadro, retirando as informações uteis e eliminando ao máximo possível os ruídos adquiridos pela transmissão deste sinal.

2.2 Sincronização de Símbolos

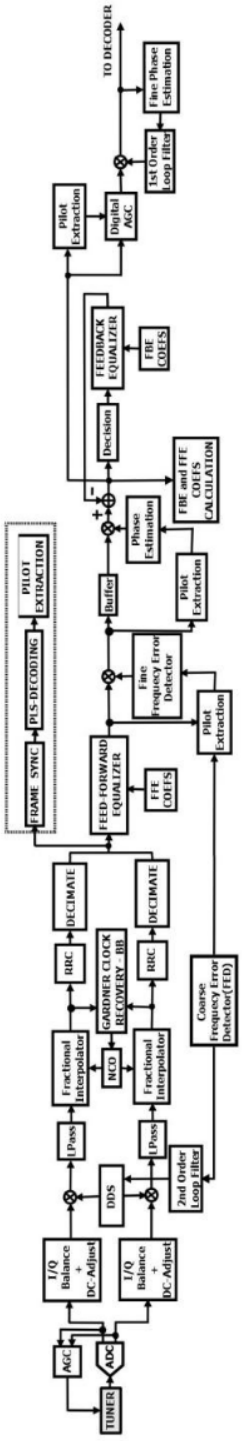


Figura 4 – Módulo de Sincronização de Símbolos. Retirado de (LIMA et al., 2013)

No módulo de Sincronização de Símbolos, o sinal recebido passará por um processamento de sinal, retirando o ruído produzido pelos desvios de tempo de amostragem, de frequência e de fase, de forma que a informação seja melhor decodificada pelo módulo de decodificador de sinal.

A primeira etapa deste módulo é a amostragem do sinal recebido, que se dá por um ADC, que se encarregará de transformar um sinal recebido em formato analógico para

um formato digital, assim sendo possível processá-lo digitalmente.

Após passar pela etapa de amostragem, o sinal vai ser convertido para banda-base por meio de um DDS e um filtro passa-baixas.

Depois de convertido para banda-base o sinal entrará em um bloco de sincronização de tempo, onde serão detectadas discrepâncias de tempo no sinal por meio do algoritmo de Gardner (Gardner, 1993) e corrigirá o erro de temporização da portadora por meio de uma interpolação fracionária. Após esta etapa o sinal passará por um filtro cosseno levantado (RRC).

Na próxima etapa acontece a sincronização dos quadros, utilizando a autocorrelação de cabeçalho e um algoritmo de detecção de picos (*Peak Search Algorithm*) (VENTZAS; PETRELLIS, 2011). Após esta etapa já se tem as informações de modulação, SOF, taxa de código, tipo de quadro e símbolos-pilotos. Estes parâmetros por sua vez serão enviados aos próximos blocos para a realização dos sincronismos de frequência (uma grosseira e uma fina), fase e correção de amplitude.

Uma das etapas mais importantes no processamento do sinal é a etapa de correção de frequência, a qual se divide em correção grosseira de frequência (*Coarse Frequency Correction*) e correção fina de frequência (*Fine Frequency Correction*). A norma do protocolo DVB-S2X estabelece que o máximo erro de frequência que o CFC deve entregar ao FFC é de 100 kHz ou 10^{-4} em frequência normalizada (CASINI; GAUDENZI; GINESI, 2004).

Assim como a frequência, é necessário também corrigir os ruídos de fase, que foram acumulados graças aos desvios de frequências que ocorreram nele. No protocolo DVB-S2X esta etapa de correção de fase também é realizada por dois módulos corretores de fase, sendo um a Correção Grosseira de Fase e o outro a Correção Fina de Fase. Para este trabalho tem-se apenas o módulo de Correção Grosseira de fase, que será chamado somente de Correção de Fase.

2.3 Correção Grosseira de Frequência

Devido a vários fatores externos e até internos ao sistema de transmissão e recepção de um sinal, este fica sujeito a ruídos de frequências que comprometem a portadora, causando o que chamamos de desvio de frequência, fazendo com que o sinal transmitido chegue ao receptor com certas discrepâncias em sua constelação.

Esta é a primeira etapa de correção da frequência da portadora. Nesta etapa, o sinal recebido precisa ter sua frequência da portadora corrigida de forma "grosseira". Para que esta correção seja feita, a norma recomenda o uso do algoritmo de Detector de Erro de Frequência (*Frequency Error Detector*), que se dá pela fórmula abaixo (CASINI;

GAUDENZI; GINESI, 2004):

$$e(k) = Im\{z^p(k)c^{*p}(k-2)z^{*p}(k-2)c^p(k)\}, \tag{2.1}$$

onde c é a constante de símbolo piloto ($\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j$), z é o símbolo-piloto que entrará para ser corrigido, * significa o complexo conjugado, Im significa a parte imaginária da operação, e k é o índice da amostra, tal que $k \in N$ (em função do quadro).

Após estimado o erro de frequência e depois de filtrá-lo com um filtro PI, aplica-se um algoritmo de média móvel utilizando 36 amostras da saída do filtro PI, a fim de deixar os resultados melhores para a correção da frequência. O algoritmo se baseia na equação abaixo

$$E_{average} = \frac{1}{N}\sum_{n=0}^{N-1} e(n), \tag{2.2}$$

sendo $e(n)$ o erro calculado em um instante n.

Um filtro de média móvel mais pertinente para implementação em FPGA é a média móvel recursiva, que pode ser implementado de forma que a média seria um incremento da amostra mais nova e um decremento da amostra mais antiga, como mostra a equação abaixo

$$E_{average} = E_{average} + [e(M-1) - e(0)].\frac{1}{M}, \tag{2.3}$$

sendo $e(0)$ a amostra mais antiga, $e(M-1)$ a amostra mais nova e M é o número total de amostras da janela tal que $M \in N$. A fim de deixar o cálculo menos custoso para o hardware a divisão foi transformada para uma multiplicação por uma constante.

Após esta etapa essa média da estimativa de erro irá alimentar um DDS (*Direct Digital Synthesis*), localizado geralmente em um DDC (*Digital Down Converter*), que por sua vez corrigirá os sinais na entrada. É importante ressaltar que o funcionamento do módulo CFC como um todo é análogo ao de um PLL (*Phase-Locked Loop*).

Um exemplo visual do que ocorre com o sinal está mostrado na figura 5, onde temos um sinal 16-QAM com um desvio de frequência de 350 kHz, que, com as devidas correções de frequência da portadora, foi reconstituído com poucos desvios de frequências residuais, que podem ser corrigidos depois com um módulo de Correção de Fase.

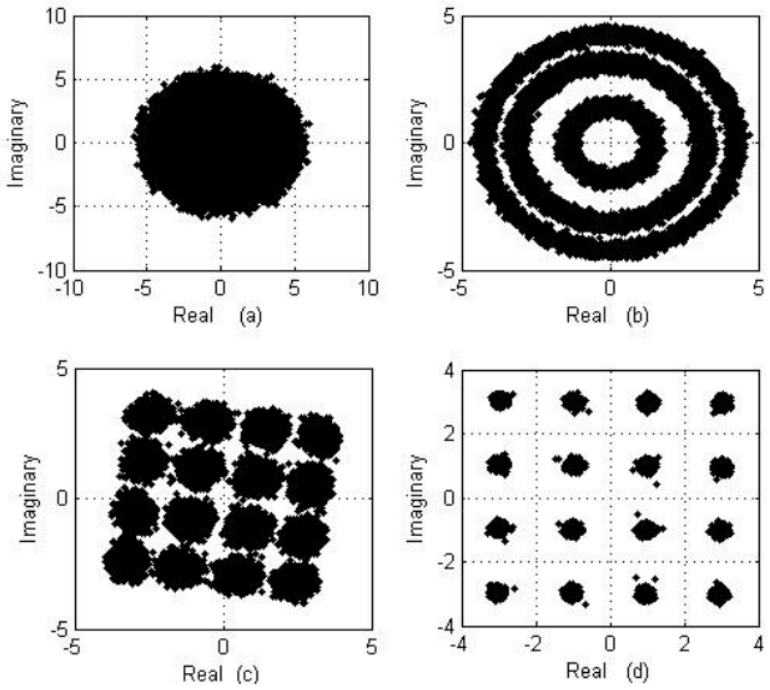


Figura 5 – Correção da frequência da portadora de um sinal modulado em 16-QAM.(HAMZEHYAN; DIANAT; SHIRAZI, 2012)

2.4 Correção de Fase

Mesmo após as devidas correções de frequência, o sinal recebido ainda possui ruídos de fase, que foram causados devido aos desvios de frequências que foram inseridos nele, ocasionando um acúmulo de fase, fazendo com que o sinal recebido ainda tenha algumas discrepâncias em sua constelação.

Para corrigir estes ruídos de fase na portadora tem-se os módulos de correção de fase, sendo um de correção mais grosseira, que funciona muito bem em modulações de ordem menor como a Q-PSK e a 8-PSK, e um de correção mais fina, que consegue corrigir melhor os erros de fase para modulações maiores, como a 16-APSK e a 32-APSK. Neste trabalho apenas será trabalhado o módulo de correção mais grosseira, sendo chamado de módulo de Correção de Fase.

A norma recomenda o uso dos algoritmos de Máxima Verossimilhança (ML) e de Interpolação Linear para realizar a estimação e correção destes desvios de fase, sendo o primeiro algoritmo usado para a estimação e correção de fase dos símbolos de pilotos e o segundo algoritmo para a estimação e correção de fase dos símbolos de dados. A equação para o algoritmo de ML é (CASINI; GAUDENZI; GINESI, 2004):

$$\hat{\theta}^{(p)} = arg\left\{ \sum_{k=0}^{L-1} [(c^{(p)}(k)^*) z^{(p)}(k)] \right\}, \tag{2.4}$$

sendo $\hat{\theta}^{(p)}$ o desvio de fase estimado para o bloco de pilotos, $c^{(p)}(k)^*$ a constante de símbolo

piloto, porém complexa conjugada ($\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j$), $z^{(p)}(k)$ é o símbolo piloto de entrada e L é o comprimento do bloco de pilotos, que é 36.

Para calcular a função arg, uma forma mais pertinente para FPGAs é utilizar um algoritmo CORDIC, que nada mais é do que um algoritmo iterativo capaz de realizar o cálculo de funções trigonométricas, como o arco-tangente, por meio de micro-rotações de vetores de entrada (X e Y). As equações que regem o funcionamento do CORDIC são (Muñoz et al., 2010):

$$X_{i+1} = X_i - m.\sigma_i.2^{-i}.Y_i \tag{2.5}$$

$$Y_{i+1} = Y_i + m.\sigma_i.2^{-i}.X_i \tag{2.6}$$

$$Z_{i+1} = Z_i - \sigma_i.\theta_i, \tag{2.7}$$

sendo i o índice da iteração, θ_i é o valor de ângulo já pré-calculado para as micro-rotações, σ_i indica a direção da rotação e m indica o tipo de coordenada ($m = 1$ para circular e m = -1 para hiperbólica) (Muñoz et al., 2010).

Abaixo tem-se a equação do algoritmo de Interpolação Linear (CASINI; GAUDENZI; GINESI, 2004):

$$\hat{\theta}(k_s) = \hat{\theta}(\bar{l}.L_s) + \left[\hat{\theta}((\bar{l}+1).L_s) - \hat{\theta}(\bar{l}.L_s)\right], \tag{2.8}$$

onde $\hat{\theta}(k_s)$ é o desvio de fase do símbolo de dado na posição k_s, $\hat{\theta}(\bar{l}.L_s)$ é o desvio de fase estimado anteriormente, $\hat{\theta}((\bar{l}+1).L_s)$ é o desvio de fase estimado atual, $\left(\frac{1}{L_s}\right)$ é o comprimento do bloco de dados invertido (geralmente $\frac{1}{1440}$) e k_s é o índice do símbolo de dado, geralmente variando de 1 até 1440.

Estimados os valores para os desvios de fase dos símbolos de dados e dos símbolos de pilotos, necessita-se que estes desvios sejam corrigidos. Uma forma de se realizar esta correção se dá por uma multiplicação complexa destes símbolos pelo seu valor de seno e cosseno complexo conjugado, sendo o ângulo o desvio de fase estimado para estes símbolos.

2.5 Hardware Reconfigurável e SoC

A figura 6 representa a arquitetura interna de um FPGA (*Field Programmable Gate Array*), que é um chip que possibilita a programação a nível de portas lógicas.

Nessa arquitetura em específico temos vários blocos lógicos programáveis (CLB), Blocos de Memória RAM (BRAM), Blocos DSP (*Digital Signal Processor*), que realizam operações de multiplicação ou soma, e temos outros blocos de propósitos gerais, como o XADC (Conversor Analógico-Digital).

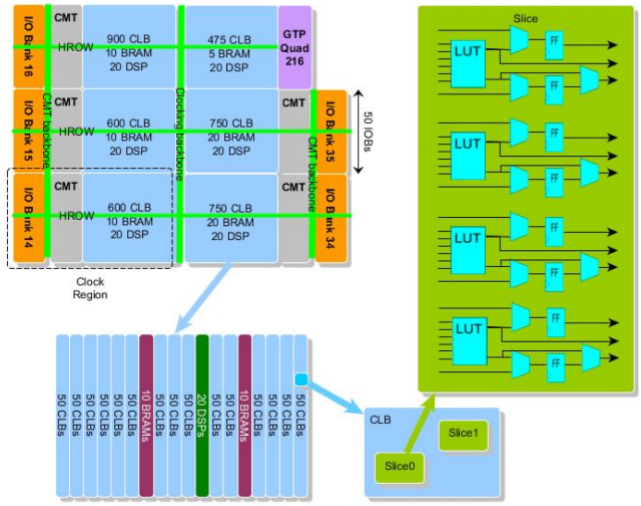


Figura 6 – Estrutura interna de uma FPGA Artix 7.(EB, 2015)

Toda programação dos blocos é armazenada em latches do tipo SRAM internos. A princípio a programação é volátil, porém há como configurar para que quando o dispositivo seja ligado novamente a programação anterior continue atuando.

Cada CLB desta FPGA contém 2 Slices, que são basicamente um agrupamento de LUT (*Look-Up Tables*), flip-flops e multiplexadores que, em quantidade, são respectivamente 4, 8 e 12 destes componentes.

Cada Slice pode ser de dois tipos: SLICEL e SLICEM. O SLICEM tem a possibilidade de ser configura como um bloco de memória distribuída com 256 bits ou até como um registrador de deslocamento de 128 bits. Já o SLICEL apenas pode ser configurado para operações combinacionais.

FPGAs são programados usando linguagens de descrição de hardware. Os mais comuns e adotados pela industria são o VHDL e o Verilog, os quais permitem descrever circuitos combinacionais e sequenciais de alta complexidade.

Nos dia de hoje um dos requisitos em sistemas embarcados de alto desempenho computacional é o aproveitamento da robustez e alto poder computacional que oferece o hardware e a flexibilidade do software no sentido de facilidade de atualização de código. Nesse sentido, os SoCs (*System on Chips*), são uma plataforma adequada para o co-projeto hardware software, no qual é possível executar tarefas em alto nível e acelerar, através de instruções customizadas, cálculos complexos usando co-processadores desenvolvidos diretamente em hardware. Alguns SoCs comerciais fazem uso de processadores ARM, os quais tem o suporte de um sistema operacional, e de barramentos dedicados para a conexão com o hardware reconfigurável.

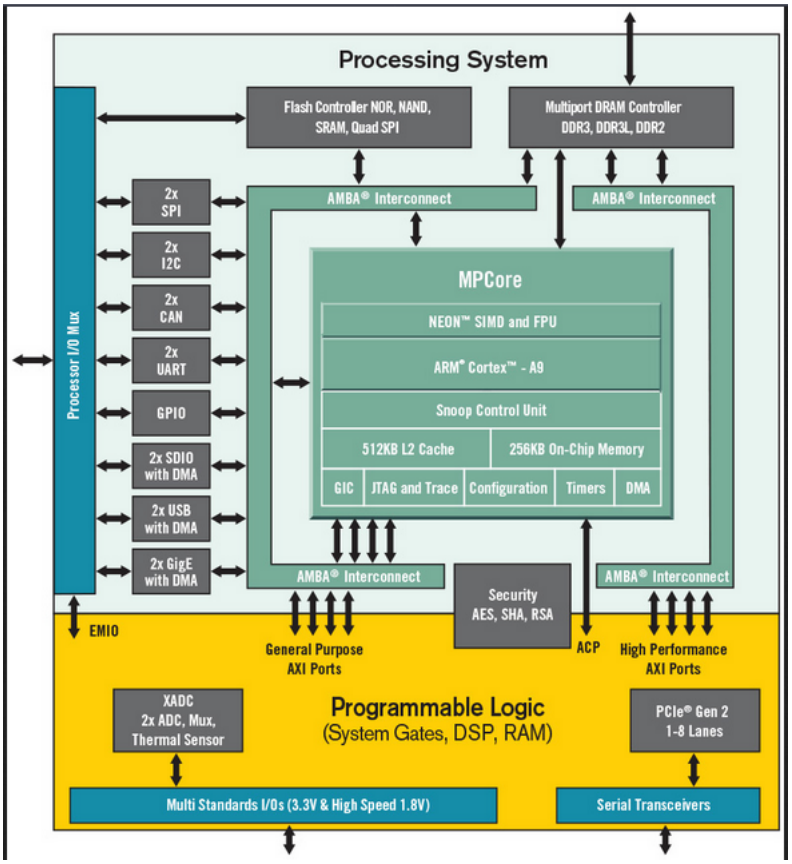


Figura 7 – SoC FPGA Zynq-7000.(XILINX,)

A figura 7 mostra a arquitetura geral de um SoC FPGA Zynq-7000. Nela é possível ver o bloco hardware programável que contém DSP, RAM, portas lógicas e barramentos AXIs, todos estes elementos básicos de todo hardware FPGA.

Na parte superior da figura temos a arquitetura SoC em si que está conectada a parte do bloco lógico programável, onde podemos ver que ela possui um processador *Single Core* ARM Cortex^{TM}-A9, uma memória L2 cache de 512KBs e uma memória On-Chip de 256KBs.

O bloco do processador se conecta por meio de um barramento AMBA aos blocos gerais da ZYNQ, como os blocos de GPIO, UART, I2C, dentre outros.

2.6 Estado da Arte

Os módulos de correção grosseira de frequência e correção de fase já foram implementados em FPGA e em software anteriormente pela comunidade científica. As implementações muitas vezes diferem de algoritmos, técnicas, aplicações e linguagens que foram utilizados.

Uma primeira implementação do módulo CFC foi feita em (de Lima et al., 2014). Nessa implementação os autores utilizaram um algoritmo derivado do Algoritmo de Kay (Kay, 1989). A figura abaixo mostra a arquitetura geral proposta nesse trabalho. Note que o módulo CFC está logo depois do bloco de Sincronização de Quadro (*Frame Synchronization*) e antes do bloco de Correção Fina de Frequência (*Fine Frequency Correction*):

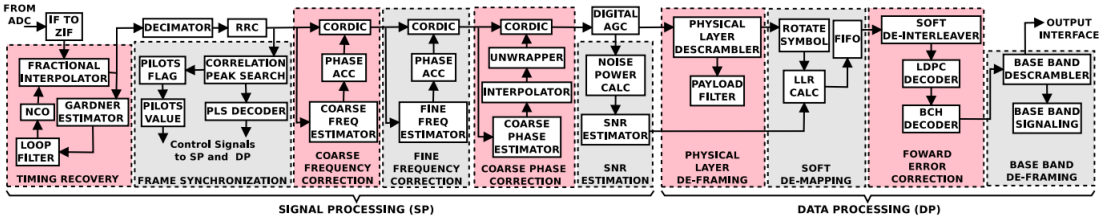


Figura 8 – Arquitetura do receptor DVB-S2 proposta em (de Lima et al., 2014).

A saída do bloco do algoritmo de Kay alimenta um acumulador de fase, que por sua vez irá para um CORDIC (*COordinate Rotation DIgital Computer*), realizando a correção. A saída do módulo CFC irá alimentar o bloco de FFC.

Também no mesmo trabalho há a implementação de um corretor de fase. Este módulo vem logo após o corretor fino de frequência, e faz uso de correlações e pilotos para estimar o erro de fase. Com o erro de fase estimado, o módulo também utiliza dos algoritmos de Interpolação Linear para a estimação do erro de fase entre blocos de pilotos consecutivos e de *Unwrapper*, este último para eliminar as ambiguidades de fase.

Os resultados obtidos pelos autores mostram que o receptor implementado em hardware foi capaz de sincronizar e demodular um sinal 8PSK com todos os ruídos de fase e AWGN esperados em um sistema receptor digital, exceto para uma taxa de código de 3/5. Na figura 9 apresenta-se um dos principais resultados obtidos pelos autores na fase de sincronismo de frequência e fase. As figuras à esquerda representam duas constelações em banda base, com SNR de 16dB (topo) e 21dB (abaixo), respectivamente. Já nas figuras à direita temos as mesmas constelações após passarem pelas etapas de sincronismo de frequência e fase, tanto a etapa de correção grosseira quanto a etapa de correção fina de frequência, bem como pela etapa de correção de fase.

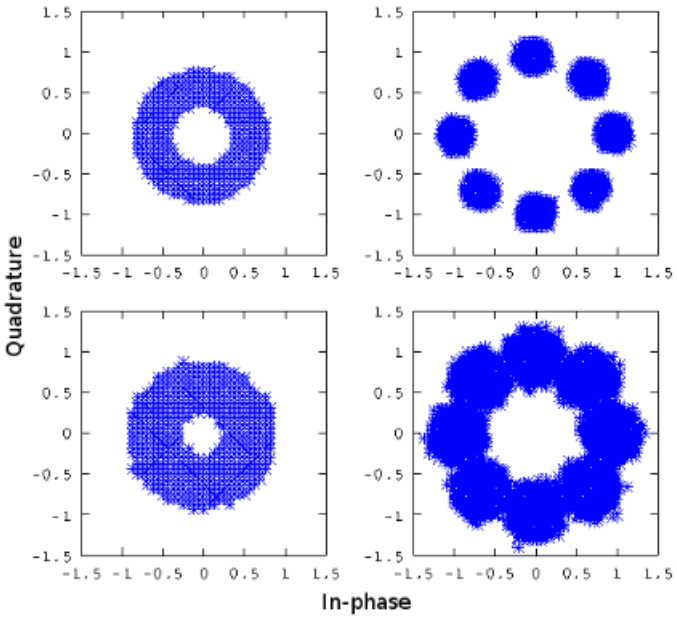


Figura 9 – Constelações capturadas do protótipo em FPGA, sendo 16dB (acima) e 21dB (abaixo) de SNR. (de Lima et al., 2014)

Ainda Lima et al. (LIMA et al., 2013) apresentam uma arquitetura para o receptor DVB-S2 na qual a correção grosseira de frequência difere da que foi proposta em (de Lima et al., 2014). O módulo corretor grosseiro de frequência tem um bloco de detector de erro de frequência, que está alimentando um filtro de segunda ordem, que por sua vez está alimentando um DDS. Os autores não esclarecem qual o algorítimo utilizado para o CFC. Entretanto, alegando que a plataforma FPGA possuía uma baixa taxa de erros de frequência, o módulo CFC não foi implementado, dando espaço apenas ao módulo FFC.

Neste mesmo trabalho há a presença de dois módulos corretores de fase, sendo um corretor grosseiro de fase e um corretor fino de fase. O autor não deixou claro como foi implementado o corretor fino de fase, porém o corretor grosseiro de fase foi implementado utilizando algoritmos de correlação e pilotos para estimar e corrigir os erros de fase, da mesma forma como foi em (de Lima et al., 2014).

Um dos resultados encontrados pelos autores com relação a sincronismo de frequência e fase está mostrado na figura 10. Na figura da esquerda apresenta-se o sinal recebido de uma modulação 8-PSK, contendo erros de tempo, frequência e de amostragem. Já na figura do meio, temos o mesmo sinal, porém com a amostragem já corrigida. Por fim, na figura da direita, o sinal com todas as suas correções realizadas, principalmente no que se trata de correções de frequência e fase da portadora.

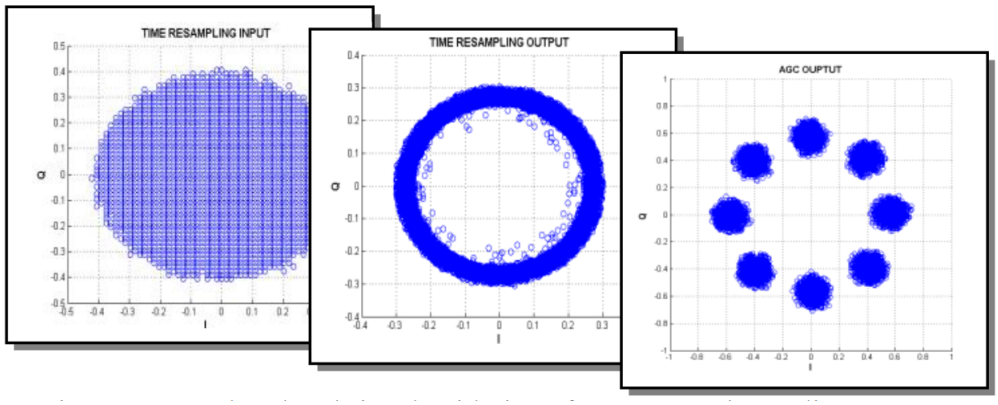


Figura 10 – Resultados obtidos para a correção da frequência da portadora (LIMA et al., 2013).

Em (CASINI; GAUDENZI; GINESI, 2004) foi implementado todo o sistema de transmissão e recepção do protocolo DVB-S2. Na figura 11 mostra-se a arquitetura em alto nível do transmissor e a figura 12 mostra que a arquitetura de recepção é similar ao que foi apresentado na sessão 2.1.

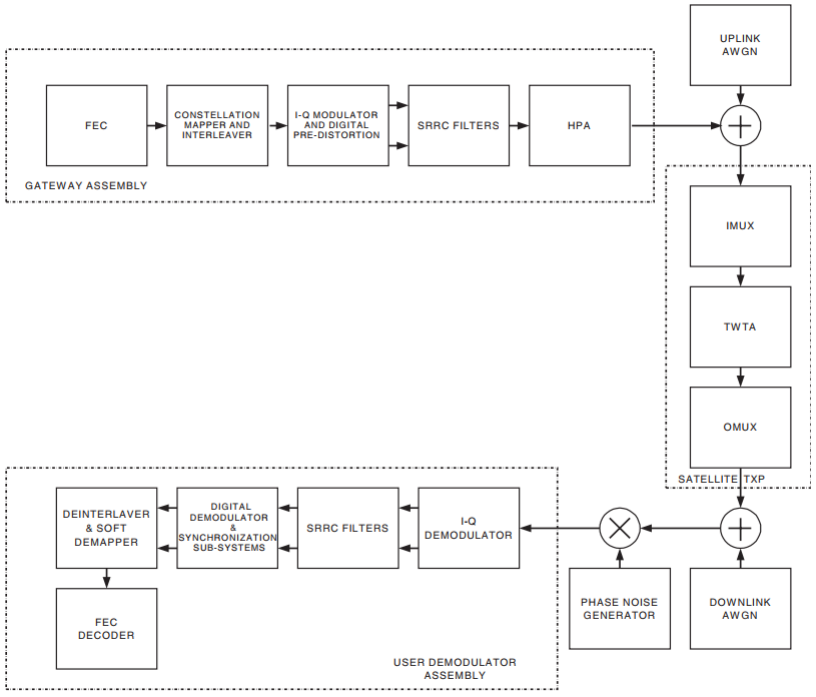


Figura 11 – Arquitetura em alto nível. Retirado de (CASINI; GAUDENZI; GINESI, 2004).

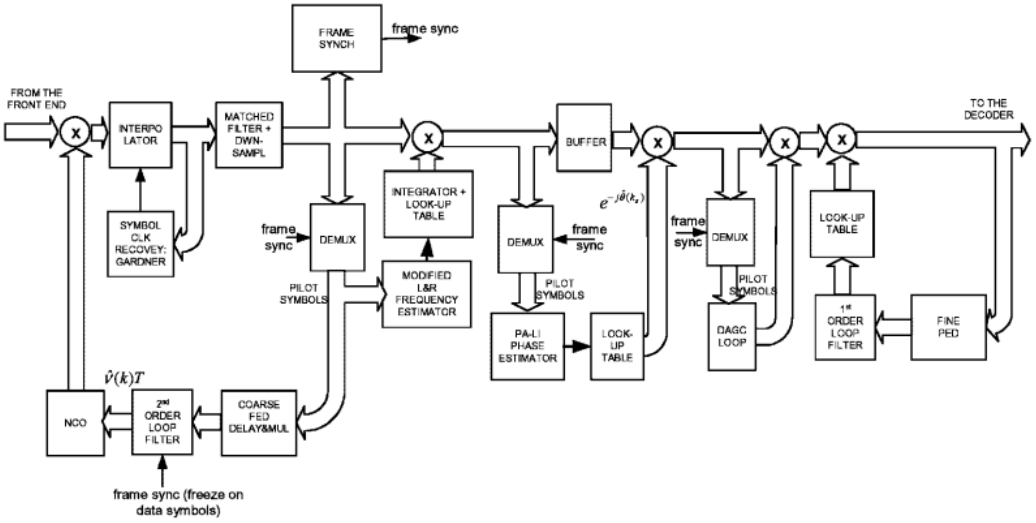


Figura 12 – Arquitetura do receptor DVB-S2X. Retirado de (CASINI; GAUDENZI; GINESI, 2004)

O módulo CFC foi implementado utilizando da equação 2.1. Segundo o mesmo trabalho, para o pior caso de Es/No, o módulo CFC conseguiu corrigir desvios de frequência do sinal de entrada de forma que o máximo desvio de frequência residual foi de 100 kHz.

Na figura 13 tem-se o rastreamento de frequência realizado pelo módulo implementado nesse trabalho, que utilizou um filtro com largura de banda de 10^{-4} com Es/No de -2dB.

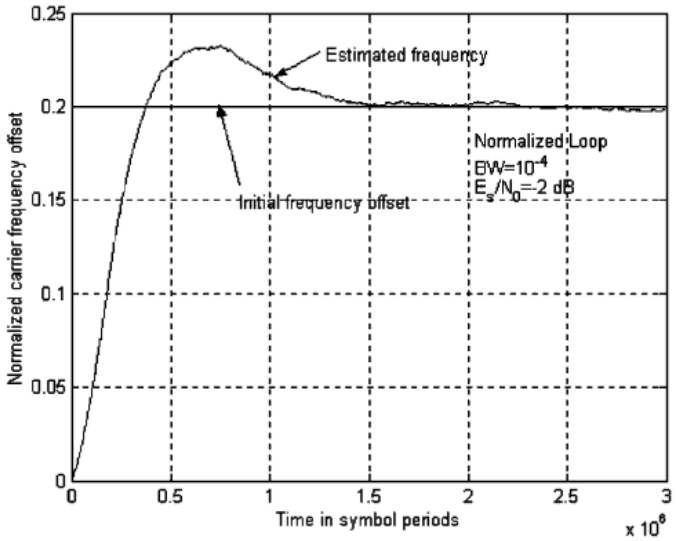


Figura 13 – Curva de rastreamento de frequência normalizada em função do tempo em símbolos de piloto. Retirado de (CASINI; GAUDENZI; GINESI, 2004)

Ainda neste trabalho, foram implementados dois módulos de correção de fase. O primeiro deles foi um corretor grosseiro de fase que utiliza do algoritmo de ML (2.4), e do algoritmo de Interpolação Linear (2.8). Também foi utilizado um algoritmo de *Unwrapper*, para deixar os valores estimados de fase dentro de um intervalo entre $-\pi$ a π. Abaixo, na figura 14, é ilustrado o comportamento destes algoritmos de estimação de fase no que se diz respeito ao erro RMS residual de fase, em graus, por Es/N0.

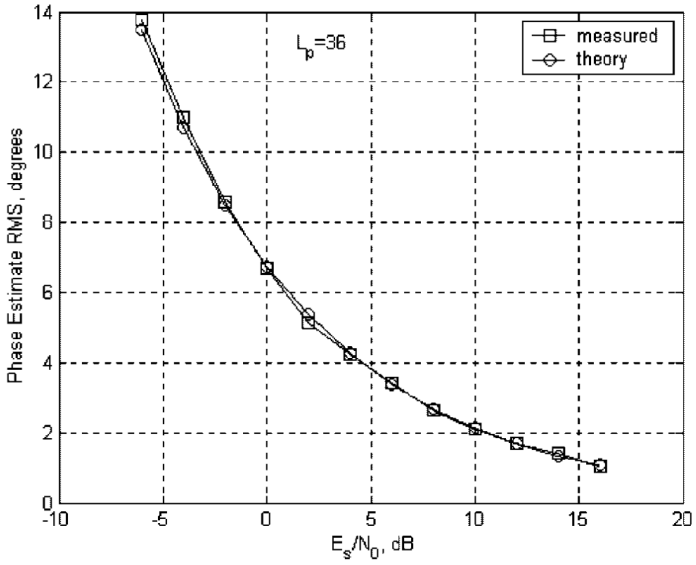


Figura 14 – Curva do erro RMS residual, em graus, em função do valor de Es/N0. Retirado de (CASINI; GAUDENZI; GINESI, 2004).

Em (Qi Luo; Xiaojun Cheng; Zucheng Zhou, 2007) também foi implementado o protocolo de comunicação do receptor DVB-S2 em FPGA, mostrado na figura 15.

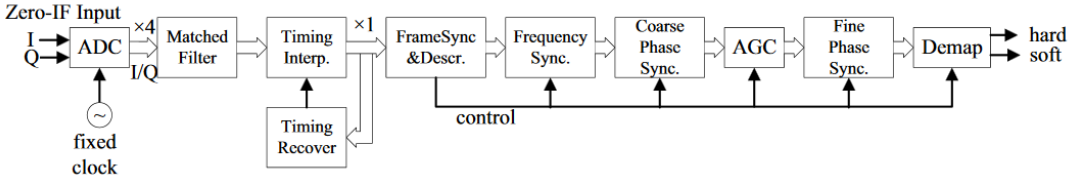


Figura 15 – Arquitetura do receptor DVB-S2 implementada em FPGA (Qi Luo; Xiaojun Cheng; Zucheng Zhou, 2007).

Nessa arquitetura o módulo de correção de frequência foi implementado seguindo o algoritmo L&R modificado. Os resultados obtidos por este módulo estão mostrados na figura 16, para um número de símbolos de 2500, requerendo mais tempo para estabilizar (entre 0.06 e 0.15s).

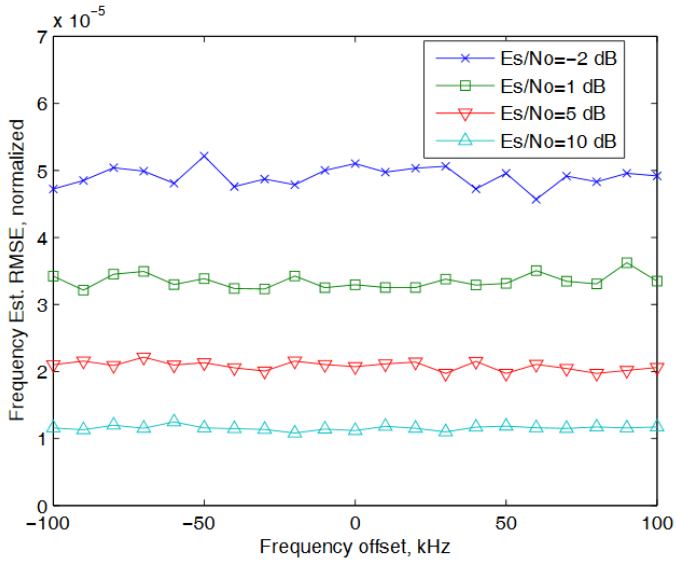


Figura 16 – Resultados obtidos para o módulo de correção de frequência (Qi Luo; Xiaojun Cheng; Zucheng Zhou, 2007).

Os autores deste trabalho também implementaram dois módulos de correção de fase, sendo um para a correção grosseira e um outro para uma correção mais fina. O módulo de correção grosseira de fase utilizou os algoritmos de ML, *Unwrapper* e de Interpolação Linear para a estimação e correção do erro de fase.

Na figura 17 tem-se a curva RMSE da estimação de fase, em graus, em função dos valores de Es/N0, sendo que desta vez foram traçadas curvas para diferentes valores de desvios de frequências:

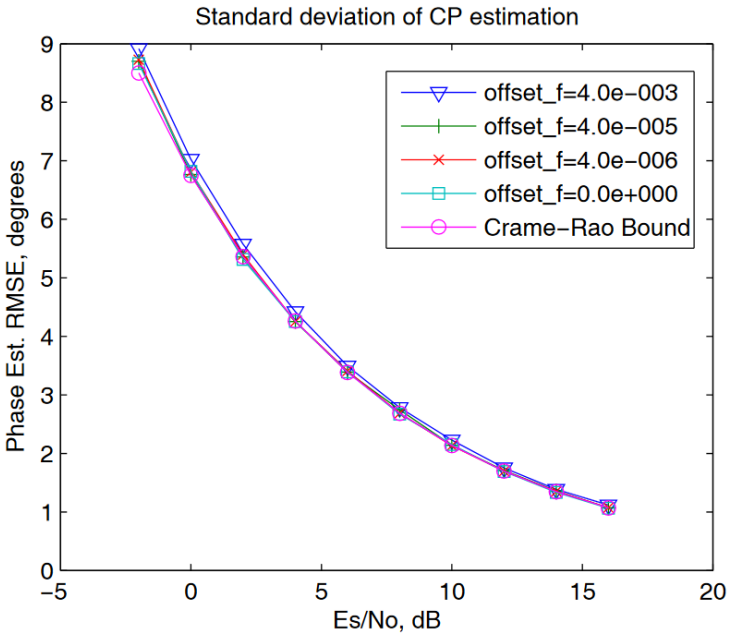


Figura 17 – Curva RMSE do estimador de fase (Qi Luo; Xiaojun Cheng; Zucheng Zhou, 2007).

Em (Jong Gyu Oh; Joon Tae Kim, 2010) foi proposto um módulo de correção grosseira de frequência alternativo para os sistemas de transmissão baseados em DVB-S2. A arquitetura proposta pelos autores pode ser vista na figura 18.

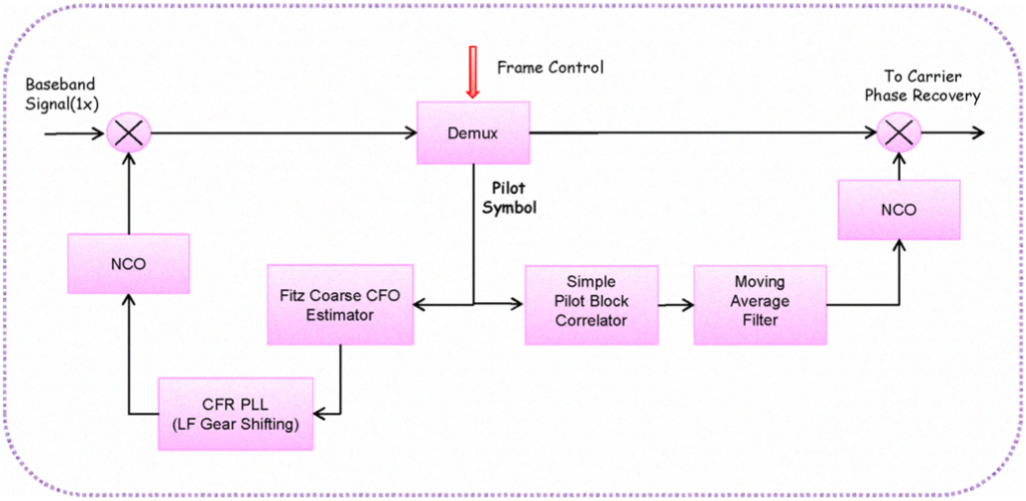


Figura 18 – Arquitetura do módulo de correção de frequência proposta em (Jong Gyu Oh; Joon Tae Kim, 2010)

Os autores implementaram tanto o módulo CFC quanto o módulo FFC. Em particular, o módulo CFC foi implementado utilizado o algoritmo de Fitz (Fitz, 1991), pois segundo os autores este algoritmo possui apenas mais uma operação de arco-tangente e menos multiplicações, além de ter um desempenho melhor do que os outros algoritmos estudados por eles. Os resultados obtidos estão na figura 19.

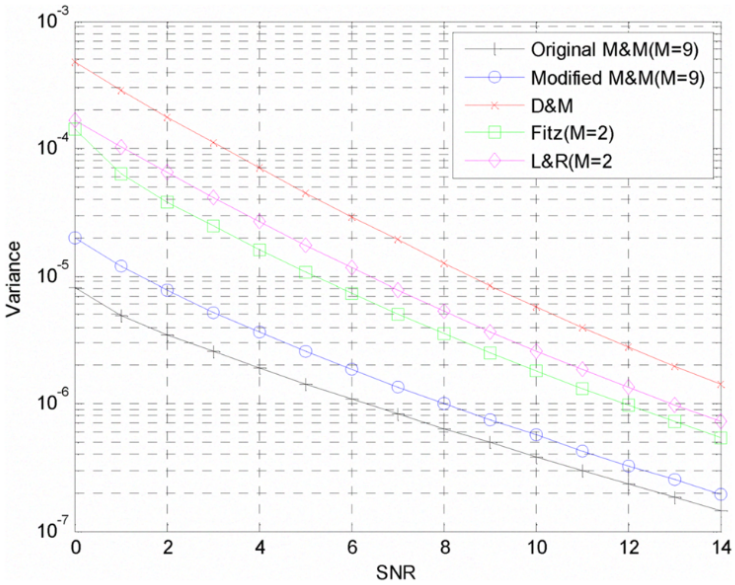


Figura 19 – Comparação entre os algoritmos estudados em (Jong Gyu Oh; Joon Tae Kim, 2010).

Neste mesmo trabalho os autores também implementaram um módulo para a correção do erro de fase. Este módulo utilizou novamente dos mesmos algoritmos citados anteriormente no trabalho de Casini et.al (CASINI; GAUDENZI; GINESI, 2004) para realizar a estimação e correção dos erros de fase. A seguir, na figura 20, apresenta-se a estrutura proposta do módulo de Correção de Fase dos autores:

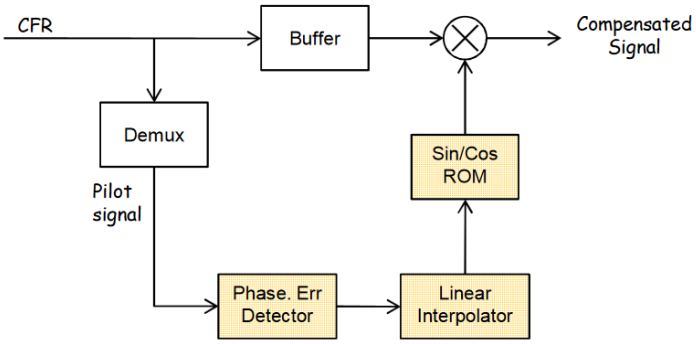


Figura 20 – Estrutura do módulo de Correção de Fase (Jong Gyu Oh; Joon Tae Kim, 2010).

Park et al. (Park et al., 2007) descreve a implementação de um módulo corretor grosseiro de frequência também em FPGA (vide figura 21).

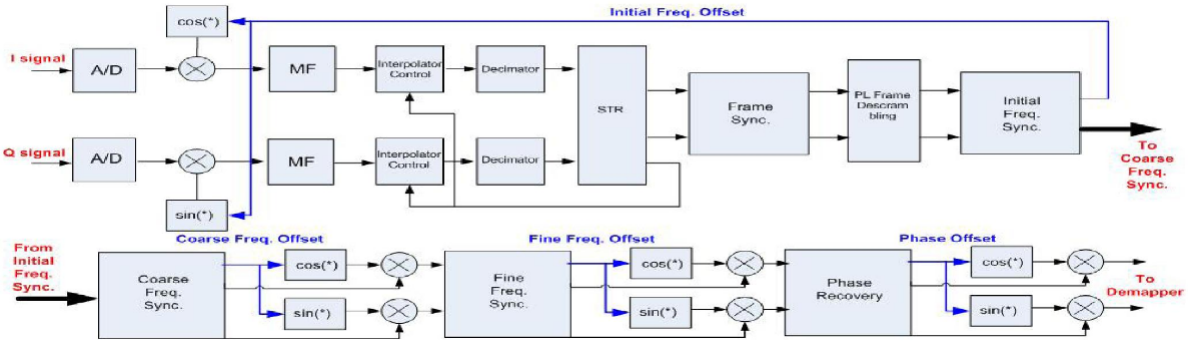


Figura 21 – Arquitetura do receptor DVB-S2.(Park et al., 2007)

Na arquitetura proposta pelos autores, o módulo CFC vem precedido de um corretor inicial de frequência, que faz uso do algoritmo M&M(MENGALI, 2013). Após ter uma primeira correção de frequência, o sinal recebido passa pelo corretor grosseiro de frequência, implementado utilizando o algoritmo L&R (Luise; Reggiannini, 1995).

Os resultados obtidos podem ser observados na figura 22. Nela apresenta-se a comparação dos três algoritmos: o algoritmo de Fitz(Fitz, 1991), o algoritmo de L&R e o algoritmo M&M. Pode-se observar que o algoritmo M&M foi o que apresentou o melhor desempenho, obtendo um desvio de frequência residual menor do que os outros dois algoritmos.

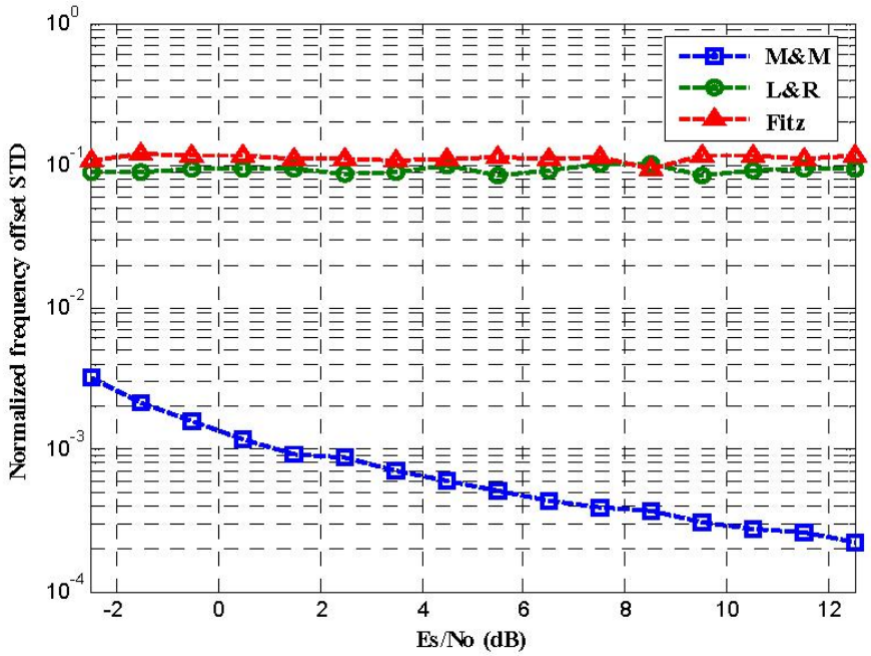


Figura 22 – Resultados obtidos pelos autores (Park et al., 2007).

Com relação a correção de fase, o trabalho cita que o algoritmo utilizado para a correção dos erros de fase é apenas utilizar a diferença de fase dos símbolos de pilotos.

No trabalho reportado em (Park et al., 2008), foi proposta uma forma mais eficiente de se corrigir desvios de frequência. Segundo os autores, esta alternativa corrige um desvio de frequência de 6,25%, obtendo um desvio residual de menos de 0,03%. Para isso utilizaram, dentre outros dois algoritmos, o algoritmo L&R. Neste trabalho não houve uma implementação de um módulo para a correção de fase. Na figura 23 mostra-se a arquitetura principal proposta pelos autores.

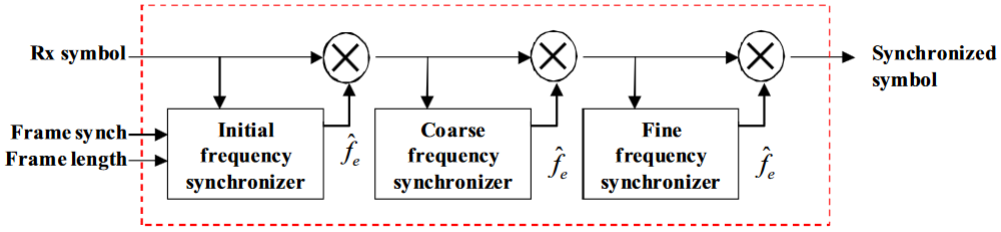


Figura 23 – Arquitetura de sincronismo de frequência proposta em (Park et al., 2008)

Abaixo identifica-se o desempenho dos algoritmos avaliados pelos autores, para diferentes desvios de frequência e Es/No. Pode-se notar que, dentre todos eles, o algoritmo L&R apresentou melhor desempenho.

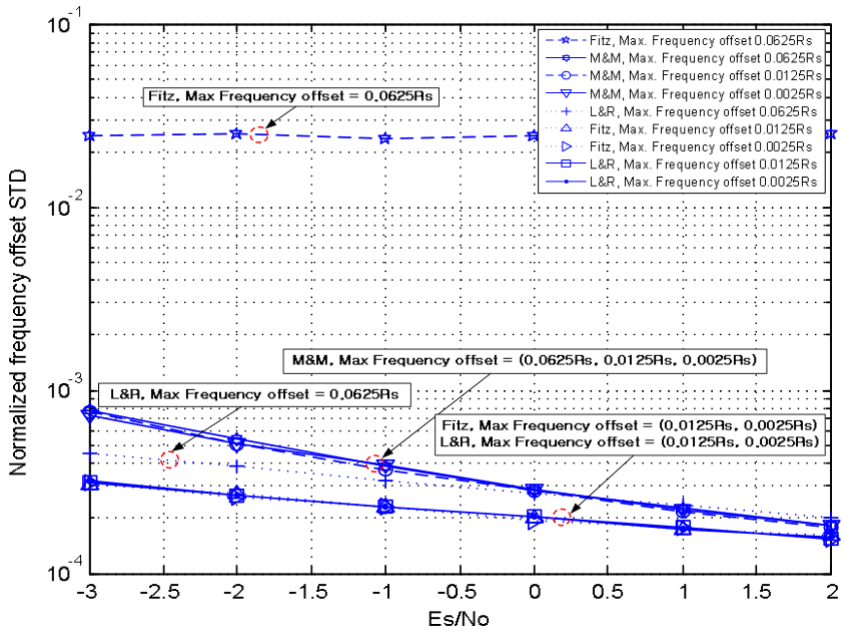


Figura 24 – Desempenho dos algoritmos selecionados para a correção grosseira de frequência.

Em (Savvopoulos; Papandreou; Antonakopoulos, 2008), o receptor DVB-S2 foi implementado em um rádio definido por software (*Software Defined Radio - SDR*), onde pode-se ver sua arquitetura na figura 25.

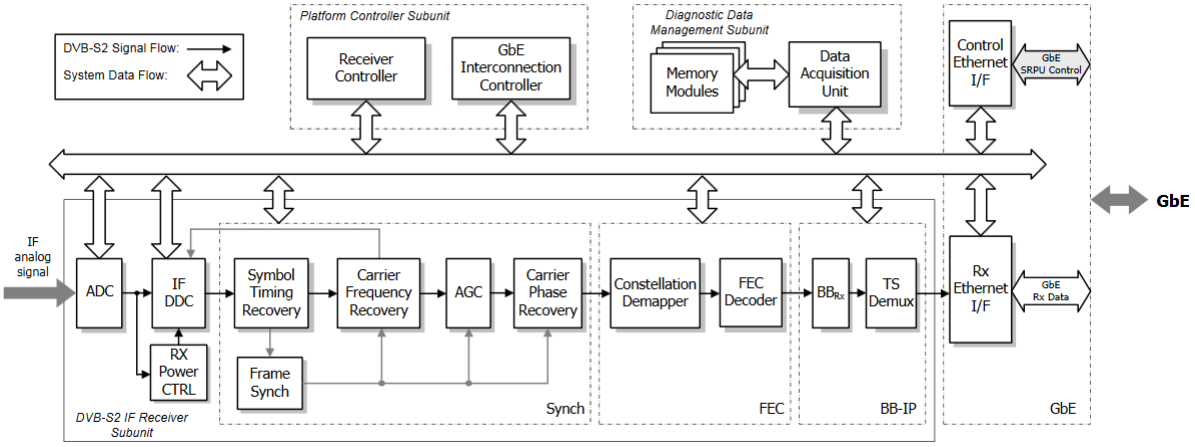


Figura 25 – Arquitetura geral do receptor DVB-S2.(Savvopoulos; Papandreou; Antonakopoulos, 2008)

Os autores também implementaram uma unidade de gerenciamento dos dados a serem enviados para a arquitetura definida por software, de forma que a operação, a aquisição de dados e o dimensionamento dos dados enviados fossem controlados por esta sub-unidade. Na figura 26 pode-se ver tal arquitetura.

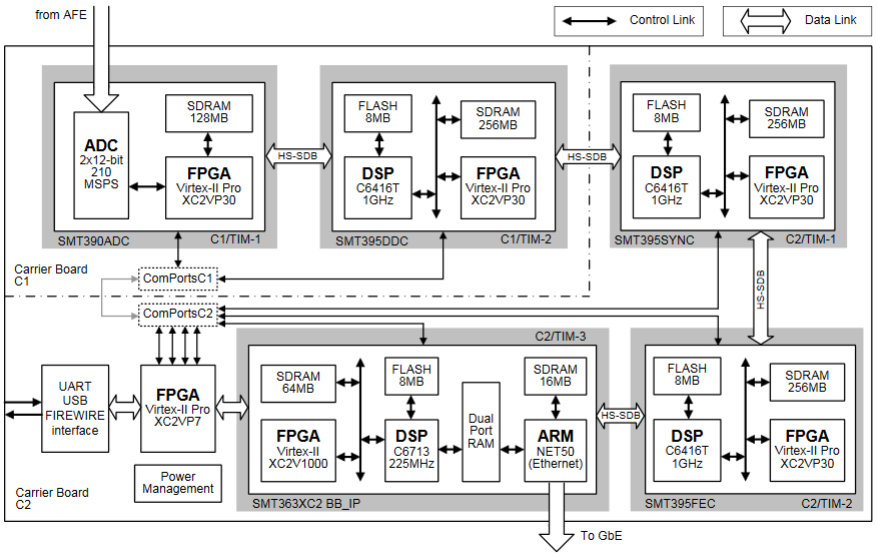


Figura 26 – Arquitetura de hardware da sub-unidade implementada.(Savvopoulos; Papandreou; Antonakopoulos, 2008)

Nota-se que a arquitetura geral do receptor neste trabalho se assemelha muito às arquiteturas já mostradas anteriormente. Em particular, a correção de frequência também está dividida em duas etapas, sendo uma delas a correção grosseira. O módulo CFC utilizado pelos autores foi implementado em um laço de realimentação de segunda ordem, baseado no algoritmo DM (MENGALI, 2013) para detectar os erros de frequência. A etapa de correção é realizada por um NCO (*Numerically Controlled Oscillator* - Oscilador Numericamente Controlado), implementado no DDC. O desvio de frequência resultante deste módulo de correção grosseira de frequência ficou em algumas centenas de kHz.

No mesmo trabalho foram implementados dois módulos para a correção de fase, novamente um módulo para a correção fina de fase e um outro para a Correção Grosseira de Fase. No que se diz respeito do último, os algoritmos utilizados foram novamente o de ML e de Interpolação Linear.

Em (Savvopoulos; Papandreou; Antonakopoulos, 2009), os autores também implementaram um módulo de correção grosseiro de frequência aplicado ao protocolo DVB-S2, porém se baseando em uma implementação multi-tarefas com DSPs. Na figura 27 apresenta-se a arquitetura básica do receptor.

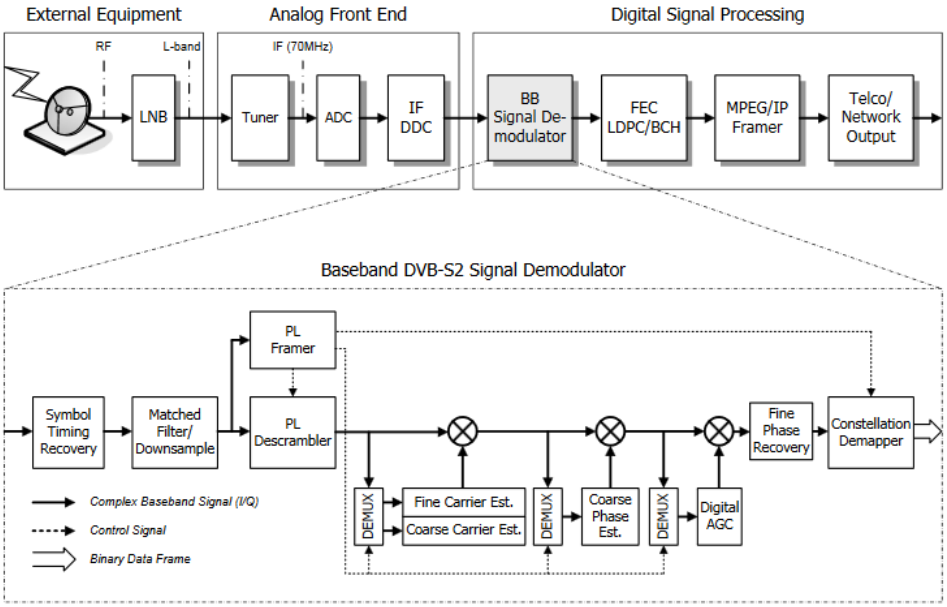


Figura 27 – Arquitetura implementada do receptor DVB-S2 (Savvopoulos; Papandreou; Antonakopoulos, 2009).

A etapa de correção do desvio de frequência da portadora foi divida em duas etapas: grosseira e fina. No que se diz respeito a etapa grosseira, os autores utilizaram do algoritmo DM (MENGALI, 2013). A arquitetura deste módulo pode ser vista na figura 28:

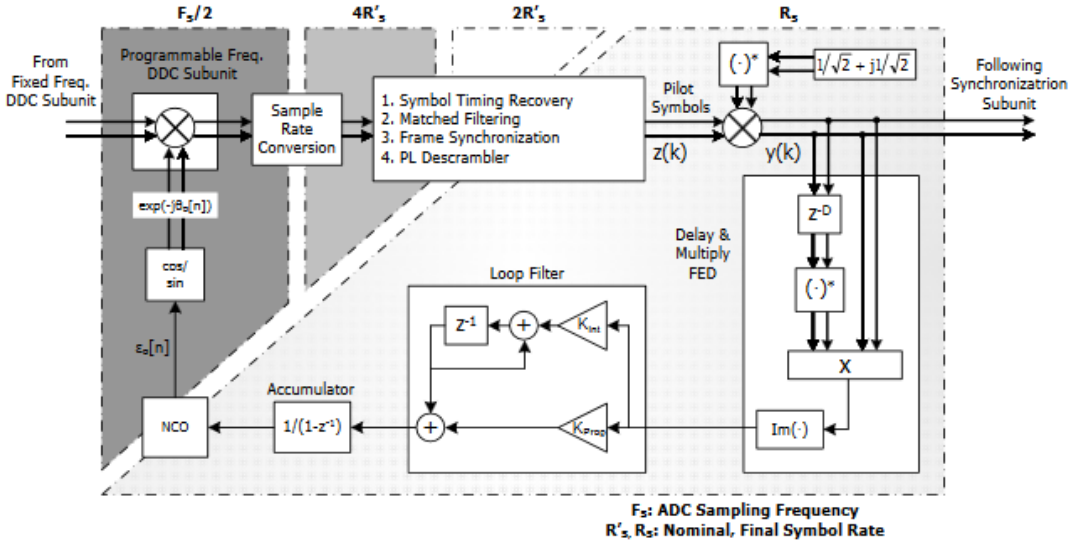


Figura 28 – Arquitetura do CFC implementada. (Savvopoulos; Papandreou; Antonakopoulos, 2009)

Nota-se que a arquitetura proposta por estes autores condiz bastante com o que foi descrito da arquitetura do CFC implementada por (Savvopoulos; Papandreou; Antonakopoulos, 2008).

No que se diz respeito a etapa de correção da fase da portadora, novamente nota-se a presença de dois módulos para a correção de fase, sendo um módulo para uma correção mais grosseira da fase e um outro módulo para uma correção fina da fase. Com relação ao módulo de correção grosseira de fase, os autores utilizaram novamente os mesmos algoritmos do estudo anterior, o de ML e de Interpolação linear. Para a correção da fase é utilizado LUTs de seno e cosseno. Na figura 29 apresenta-se a arquitetura do módulo de correção grosseira de fase.

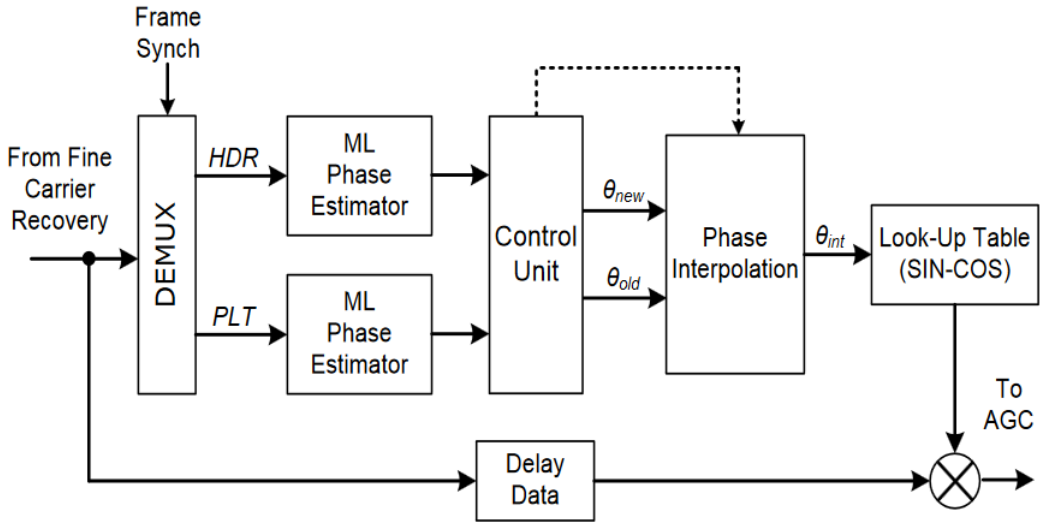


Figura 29 – Arquitetura do PC implementada (Savvopoulos; Papandreou; Antonakopoulos, 2009).

Na tabela abaixo resume os detalhes de como foi implementado cada módulo de correção grosseira de frequência e de correção de fase em diferentes trabalhos, seja a implementação em hardware, software ou SoC.

Tabela 1 – Detalhes da implementação de blocos corretores de frequência e corretores de fase.

Paper	Algoritmo Utilizado	Tipo de implementação	Técnica de Correção	Consumo de recursos CFC e PC (HW)
(de Lima et al., 2014)	Algoritmo de Kay (CFC) Correlação e pilotos (PC)	FPGA	Cordic (ambos)	LUT:5895 LUT:2663 Registradores: 3709 Registradores: 2179 Bits de Memória: 64 Bits de Memória: 1728 DSPs: 96 DSPs: 27
(LIMA et al., 2013)	não menciona (CFC) Correlação e pilotos (PC)	FPGA (ambos)	Filtro 2^a ordem e DDS (CFC) não menciona (PC)	LUT:41892 LUT:2358 Registradores: 32315 Registradores: 1892 Bits de Memória: 1488 Bits de Memória: 0 DSPs: 507 DSPs: 8
(CASINI; GAUDENZI; GINESI, 2004)	Algoritmo de realimentação (CFC) ML e Interpolação Linear (PC)	FPGA	Filtro 2^a ordem e DDS (CFC) LUT (PC)	Não menciona
(Qi Luo; Xiaojun Cheng; Zucheng Zhou, 2007)	L&R modificado (CFC) ML e Interpolação Linear (PC)	FPGA	CORDIC e NCO (CFC) Rotação de símbolo (PC)	Não menciona
(Jong Gyu Oh; Joon Tae Kim, 2010)	Algoritmo de Fitz (CFC) ML e Interpolação Linear (PC)	FPGA	PLL e NCO (CFC) ROM de Seno e Cosseno (PC)	Não menciona
(Park et al., 2007)	L&R e M&M (CFC) Não se aplica (PC)	FPGA	Seno e Cosseno (CFC) Não se aplica (PC)	Não menciona
(Park et al., 2008)	L&R (CFC) Não se aplica (PC)	FPGA	Arco tangente (CFC) Não se aplica (PC)	LUT:62954
(Savvopoulos; Papandreou; Antonakopoulos, 2008)	L&R (CFC) ML e Interpolação Linear (PC)	SoC	Loop Filter e NCO	não menciona
(Savvopoulos; Papandreou; Antonakopoulos, 2009)	L&R (CFC) ML e Interpolação Linear (PC)	SoC	Loop Filter e NCO	não menciona

3 Metodologia

Para a implementação correta dos módulos CFC e PC, foram estudados vários trabalhos de diferentes autores para selecionar os algoritmos mais pertinentes para a implementação destes módulos, avaliando o consumo de recursos que estes algoritmos poderiam precisar e também a sua complexidade. As duas metodologias utilizadas foram a *top-down* e *bottom-up*.

Fazendo uso da metodologia *top-down*, cada módulo tem seu modelo de referência em ponto fixo implementado utilizando linguagem de alto nível, sendo neste caso o Python, e assim podendo reproduzir o funcionamento deles, e traçar estratégias de implementação em hardware como precisão numérica, tempo de execução, taxa de transmissão, operadores lógicos e aritméticos, dentre outros. Já nesta etapa também se verifica se o algoritmo atinge o desempenho necessário segundo a norma.

A metodologia *bottom-up* é adotada após a etapa da implementação do modelo em alto nível, iniciando a implementação em hardware dos módulos CFC e PC. Cada bloco implementado que faz parte destes módulos é testado separadamente em simulações comportamentais, comparando seu valor de saída com o valor de saída equivalente no modelo em alto nível. Caso os valores estejam saindo iguais, o bloco implementado está validado.

3.1 Proposta de desenvolvimento

Como visto nos trabalhos citados na sessão anterior, o módulo CFC geralmente vem acompanhado de um módulo FFC, que precede o módulo PC, sendo que o FFC deve suportar um desvio residual máximo de frequência vindo do CFC de 100 kHz (CASINI; GAUDENZI; GINESI, 2004).

A frequência mínima de trabalho que será utilizada pela FPGA será de 100 MHz, bem como o máximo desvio de frequência que será adotado para o sinal de entrada será de 0.2 (frequência normalizada).

O módulo PC será capaz de corrigir os desvios de fase de forma satisfatória para as modulações Q-PSK e 8-PSK, sendo que para a modulação 16-APSK, ainda seria necessária uma outra etapa de correção de fase, desta vez sendo uma mais fina, que aconteceria logo após o sinal passar pelo AGC.

O protocolo DVB-S2X tem suporte para uma gama de modulações (BPSK, QPSK, 8PSK, 16-QAM, dentre outras), porém para as validações em específico do CFC será utilizada a modulação QPSK.

3.2 Módulo *Coarse Frequency Correction*

Após a leitura dos trabalhos, foram selecionados dois algoritmos para se fazer a estimativa de frequência: o algoritmo de Kay e o algoritmo de realimentação, utilizado por (CASINI; GAUDENZI; GINESI, 2004).

Para averiguar o bom funcionamento do módulo, foram implementados e simulados os dois algoritmos em linguagem Python. Após os testes foi verificado que o algoritmo de realimentação apresentou um melhor desempenho a uma taxa de Es/No de -2dB, bem como a norma especifica. Diante destes resultados o algoritmo selecionado para a implementação em VHDL foi o algoritmo de realimentação mencionado pela norma em (ETSI, 2014a).

A forma como o módulo CFC será implementado é baseada na norma em (ETSI, 2014a) e no trabalho (CASINI; GAUDENZI; GINESI, 2004). A arquitetura geral do módulo CFC que será implementada em VHDL é apresentada na figura 30.

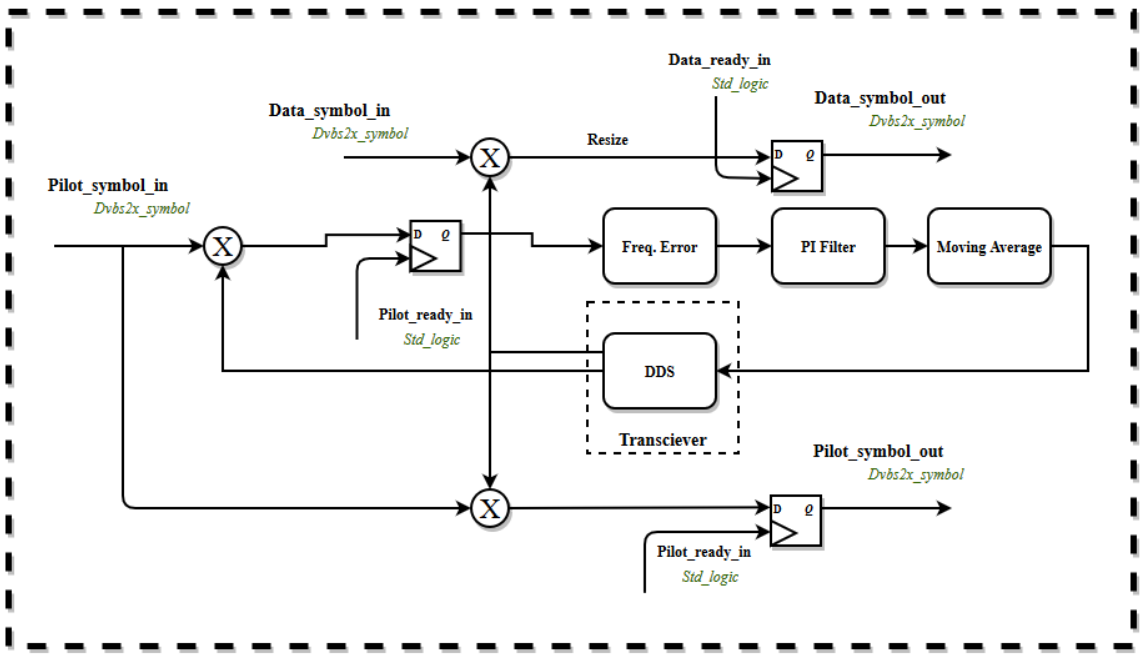


Figura 30 – Diagrama de blocos proposto para o módulo CFC.

Para que o sincronismo dos blocos dentro do módulo não fosse comprometido devido a operações que poderiam acontecer antes ou depois do esperado, cada um deles possuem seus respectivos sinais de controle, sendo principalmente: *data_ready_in*, que é o sinal que indica que o bloco está recebendo um dado, e o *data_ready_out*, que é o sinal que indica que o bloco tem uma saída válida.

Como forma de trabalhar com números de partes fracionárias, o módulo CFC trabalhará com números em representação de ponto fixo, sendo que a entrada do módulo possui 11 bits (1 bit de sinal, 3 bits de inteiros e 7 bits de fracionários).

Para economizar recursos e explorar ao máximo a robustez do hardware, as operações realizadas dentro deste módulo sofrerão truncamentos sempre que necessário, dessa forma sendo necessário menos DSPs, que é um recurso que pode ser utilizado mais em módulos que contém muitas operações, como um LDPC por exemplo. Realizar uma implementação em hardware sem observar o consumo de recursos muitas vezes pode impedir que o bloco seja usado em uma arquitetura de um receptor DVB-S2X caso FPGA utilizado não tenha recursos suficientes.

3.2.1 *Frequency Error Detector*

Nesta arquitetura apresenta-se como um dos blocos básicos o Detector de Erro de Frequência (*Frequency Error Detector - FED*) que irá estimar o erro de frequência baseado na equação 2.1. Para que a implementação do FED fosse feita em linguagem VHDL de forma mais otimizada e mais simples, esta equação foi trabalhada de forma a reduzi-la em termos mais simples. Considerando,

$$z^{p}(k) = A + jB \tag{3.1}$$

$$z^{*p}(k-2) = C \pm jD \tag{3.2}$$

e sabendo que, segundo (ETSI, 2014a),

$$c^{p}(k) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j, \tag{3.3}$$

e como apenas a parte imaginária da equação será utilizada para implementar o algoritmo, foi implementada a seguinte equação

$$e(k) = BC \pm AD, \tag{3.4}$$

sendo BC e AD as resultantes imaginárias da multiplicação das amostras de entrada. Após esta simplificação, foi pensada uma arquitetura mais pertinente para a implementação em hardware para o FED, considerando fazer uso do paralelismo oferecido pelo hardware. Esta arquitetura ficou como na figura 31 a seguir

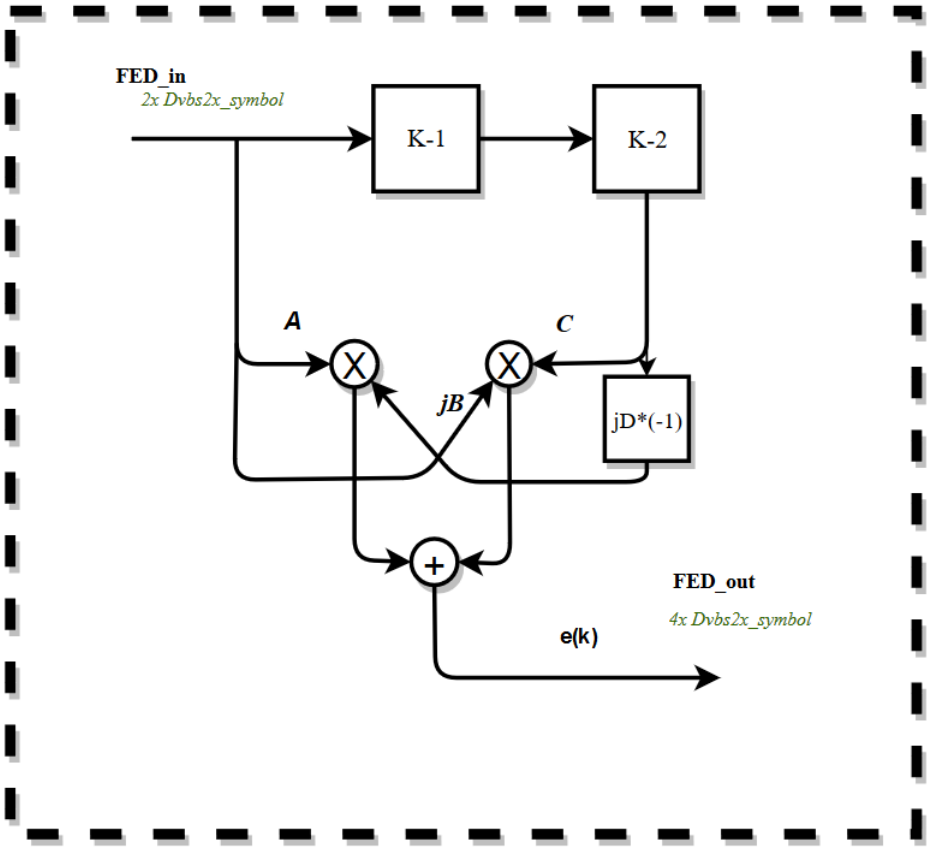


Figura 31 – Arquitetura planejada para o FED.

Onde há duas operações de multiplicação acontecendo em paralelo para que, após sua conclusão, os números calculados serão somados e finalmente irão para fora do bloco, sendo a entrada do Filtro PI.

3.2.2 Filtro Proporcional Integrador

O próximo bloco a ser implementado foi o filtro Proporcional Integrador (PI). Este filtro possui como principais constantes k_1 e k_2, que foram calculadas utilizando das seguintes fórmulas (RICE, 2009).

$$k_p k_0 k_1 = \frac{4\zeta B_n}{\zeta + \frac{1}{4\zeta}} \tag{3.5}$$

$$k_p k_0 k_2 = \frac{4B_n^2}{(\zeta + \frac{1}{4\zeta})^2} \tag{3.6}$$

Onde B_n é a largura de banda normalizada para o filtro, que segundo (ETSI, 2014a) deve ser de 10^{-4}, mas para a implementação foi considerado apenas 2.10^{-2}, devido a rapidez na convergência do algoritmo e ao desvio residual de frequência ainda estar dentro da faixa recomendada de 10^{-4} em frequência normalizada. Já k_p é a constante de ganho de erro do sistema que, neste caso, foi considerada como 4π.

Com estes valores as constantes de ganho k_1 e k_2 calculadas foram:

$$k_1 = 6,665.10^{-04}$$

$$k_2 = 8,887.10^{-06} \ .$$

Por serem muito pequenas, as constantes calculadas foram convertidas para uma representação em ponto fixo de bits fracionários maiores, para que se obtivesse uma maior precisão nos cálculos do filtro. A constantes k_1 e k_2 foram representadas com 22 bits no total e com 20 bits fracionários.

Na figura 32 apresenta-se a arquitetura planejada para o filtro PI:

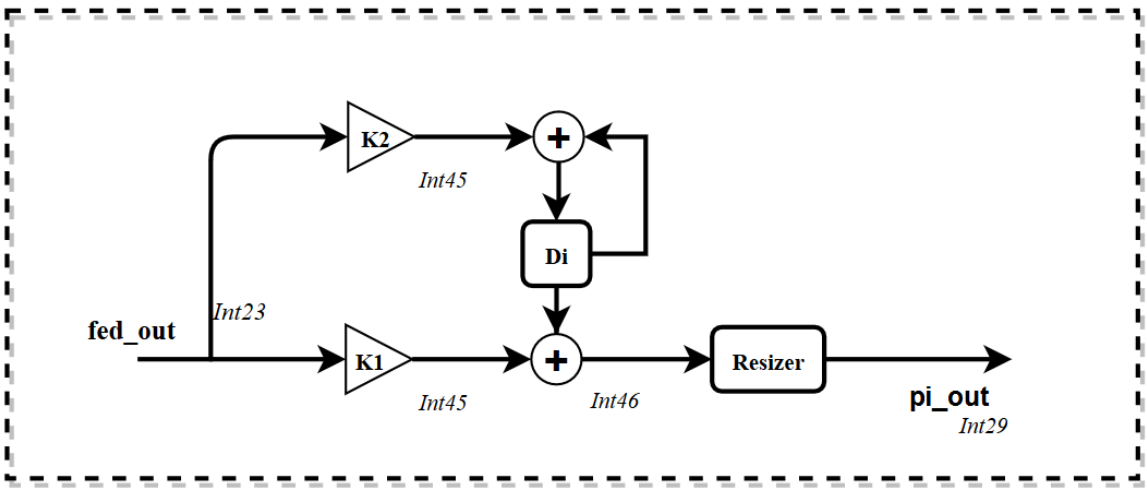


Figura 32 – Arquitetura planejada para o filtro PI.

3.2.3 Filtro de Média Móvel

De forma a deixar a saída do filtro PI ainda mais estável e para evitar alguns sobressaltos, foi implementada também uma média móvel para suavizar a saída do filtro PI, de forma que, no momento que a frequência da portadora for corrigida, essa correção não seja brusca, pois isto pode acarretar em um maior desvio de frequência residual. A arquitetura foi elaborada baseando-se na equação 2.3 e ficou como na figura 33:

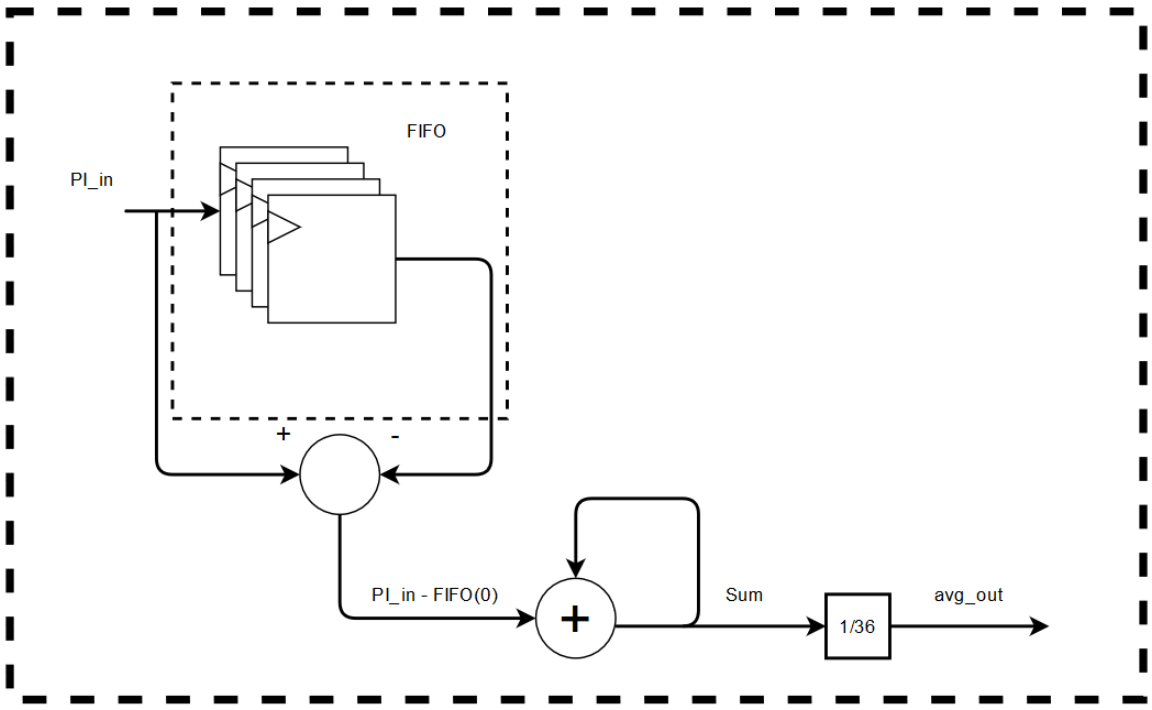


Figura 33 – Arquitetura planejada para a média móvel.

Nesta arquitetura, pode-se ver que será utilizado uma FIFO (*Fist-In-First-Out*) para que as estimativas do filtro PI sejam devidamente armazenadas para que se obtenha a média móvel. Como uma operação de divisão em hardware consome muitos recursos, a divisão que será realizada neste bloco não será uma divisão comum, mas sim uma multiplicação por uma constante $\frac{1}{36}$, transformada em um número binário de ponto fixo de 11 bits, sendo 8 bits fracionários, que é aproximadamente 0,027777.

3.2.4 *Direct Digital Synthesis*

Depois da média móvel tem-se o DDS, que é uma tabela de senos e cossenos, onde a frequência se altera com o passo do endereço de memória. Este bloco irá receber do bloco da média móvel um número positivo ou negativo que irá indicar se a frequência deve ser corrigida ou para mais, ou para menos. Logo, o bloco do DDS sempre irá começar a oscilar em uma frequência central (que será 100 kHz), que será alterada a partir das saídas recebidas do bloco de média móvel. A variação mínima de frequência adotada para o DDS será de 3,1 kHz. Na figura 34 tem-se um diagrama de blocos de como ele funciona.

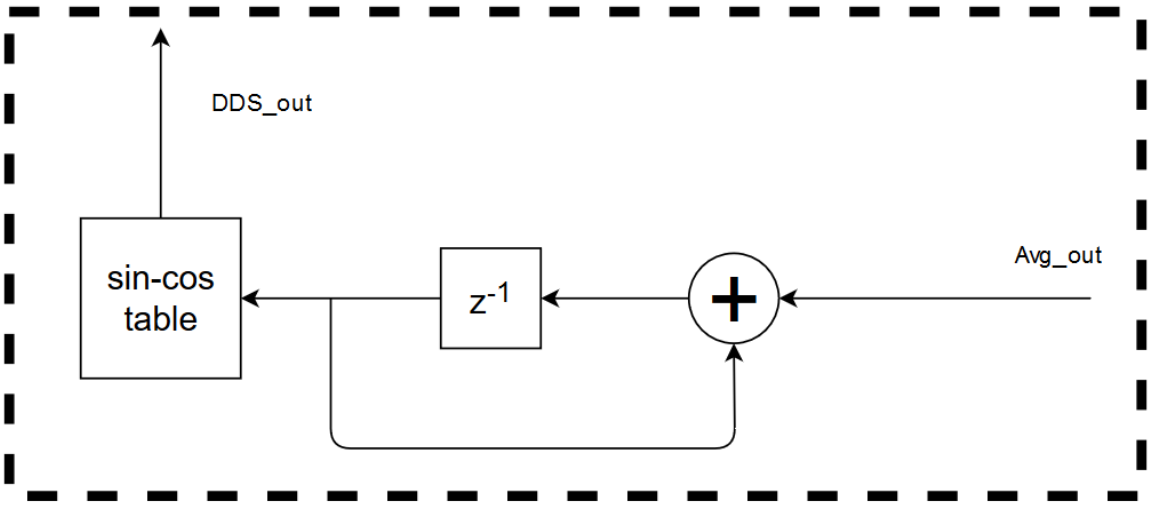


Figura 34 – Arquitetura planejada para o DDS.

Este bloco está contido no começo da cadeia de Sincronização de Símbolos, no módulo *Transceiver*, e receberá os valores de saída do Filtro de Média Móvel por meio do barramento AXI (*Advanced eXtensible Interface*), responsável por realizar a comunicação entre hardware e software em um SoC FPGA.

3.3 Módulo *Phase Correction*

Da mesma forma como foi planejada a implementação do módulo CFC, o módulo PC também teve um modelo em ponto fixo implementado e testado. O algoritmo utilizado foi o algoritmo que continha a Máxima Verossimilhança (ML) e a Interpolação Linear, que foram citados majoritariamente nos trabalhos reportados pela literatura científica.

Da mesma forma como o CFC, o módulo PC também conta com sinais de sincronização, como *data_ready_in* e *data_ready_out* (para os dados) e o *pilot_ready_in* e *pilot_ready_out* (para os pilotos), utilizados para o correto sincronismo entre os submódulos e blocos contidos no PC.

O módulo PC também trabalha com números representados em ponto fixo, porém desta vez com 13 bits (1 bit de sinal, 3 bits de inteiros e 9 bits de fracionários).

Como sempre, deve-se atentar sempre ao consumo de recursos que ele pode precisar, tendo em vista que será preciso armazenar em LUTs valores de muitas constantes e de senos, o que será melhor explicado nos itens a seguir.

O módulo PC será implementado baseando-se também na norma (ETSI, 2014a) e no trabalho de Casini (CASINI; GAUDENZI; GINESI, 2004), e baseado nisso planejou-se uma arquitetura mostrada na figura 35.

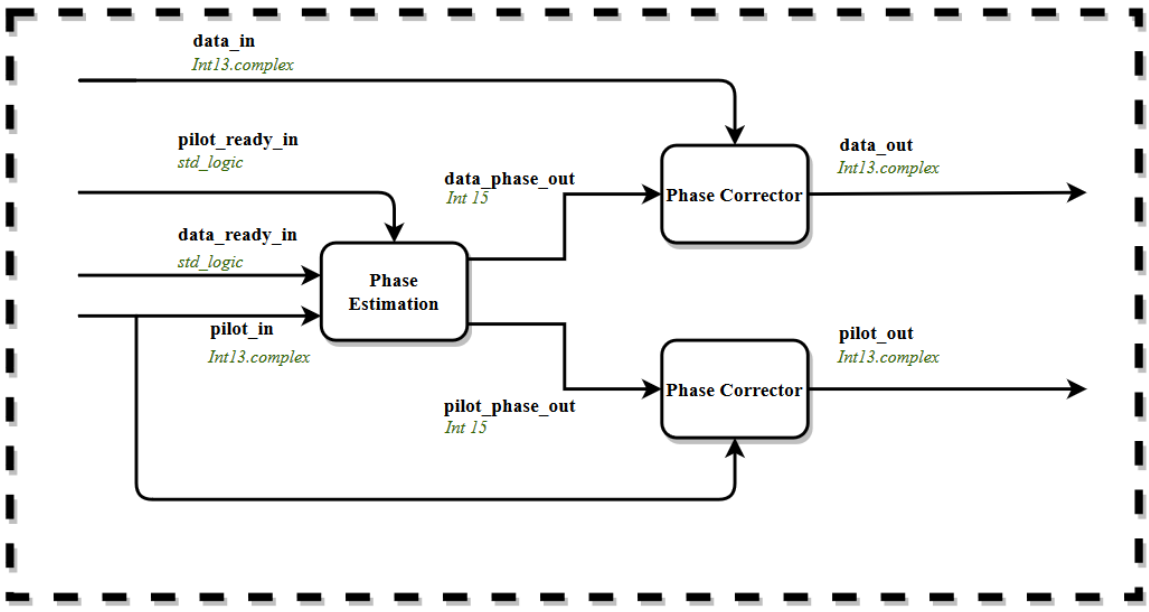


Figura 35 – Arquitetura proposta para o módulo PC.

Optou-se pela implementação de três grandes blocos: o Estimador de Fase (*Phase Estimation*) e dois blocos chamados Corretor de Fase (*Phase Corrector*), sendo que estes dois últimos serão praticamente idênticos, mudando apenas alguns parâmetros de um para o outro, indicando se o símbolo a ser corrigido é um dado ou piloto.

3.3.1 Estimador de Fase

O bloco Estimador de Fase nada mais é do que o conjunto de todos os sub-blocos necessários para a estimação do erro de fase do PC. Este bloco contém todos os sub-blocos necessários para a implementação dos algoritmos selecionados para a construção do PC. Na figura 36 é possível observar todos os sub-blocos utilizados para a estimação de fase.

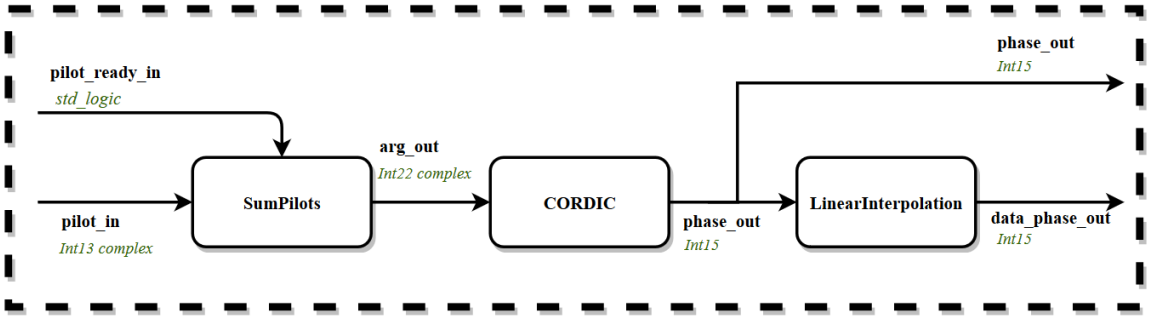


Figura 36 – Arquitetura proposta para o Estimador de Fase.

O primeiro sub-bloco (*SumPilots*) realiza a primeira parte da equação 2.4, fazendo apenas o somatório de todos o bloco de pilotos de entrada e multiplicando no fim pelo valor constante do símbolo de piloto, porém em seu formato complexo conjugado. A equação que este sub-bloco implementa está mostrada a seguir:

$$ML_{partial} = c^{(p)*} \sum_{k=0}^{N-1} z^{(p)}(k). \tag{3.7}$$

Ainda neste mesmo bloco, este resultado é reduzido para um valor entre [-1 1], de forma que o sub-bloco posterior, o *CORDIC*, que será implementado na forma de um IP já fornecido pela ferramenta Vivado, possa realizar o cálculo final do algoritmo de ML, estimando o ângulo do desvio de fase. Na figura 37 apresenta-se a arquitetura em diagrama de blocos que implementa o primeiro sub-bloco.

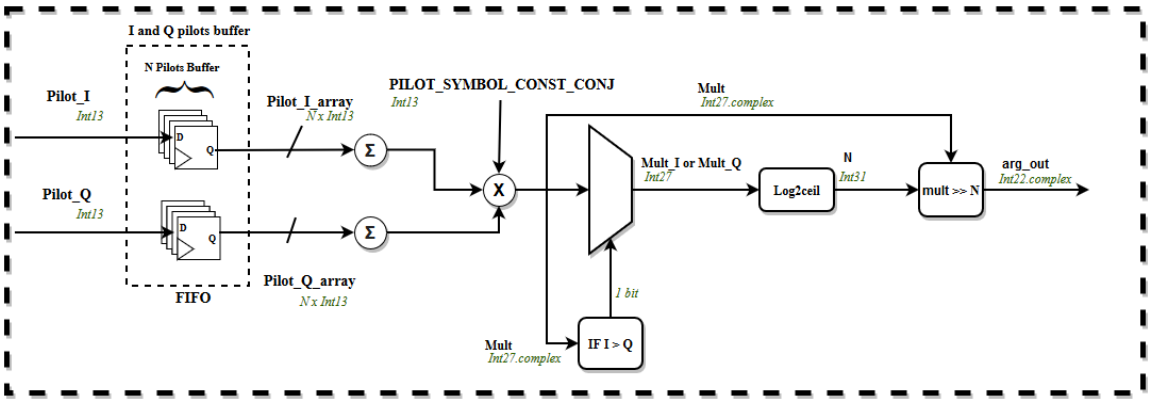


Figura 37 – Arquitetura proposta para o *SumPilots*.

O desvio de fase calculado pelo *CORDIC* será utilizado tanto para corrigir os símbolos pilotos de entrada quanto para estimar os valores de desvios de fase contidos nos símbolos de dados de entrada, servindo de argumento para o algoritmo de Interpolação Linear, representado pela equação 2.8 e implementado pelo último sub-bloco do bloco estimador de fase, chamado de *LinearInterpolation*. A seguir, na figura 38, apresenta-se a arquitetura proposta para o último sub-bloco.

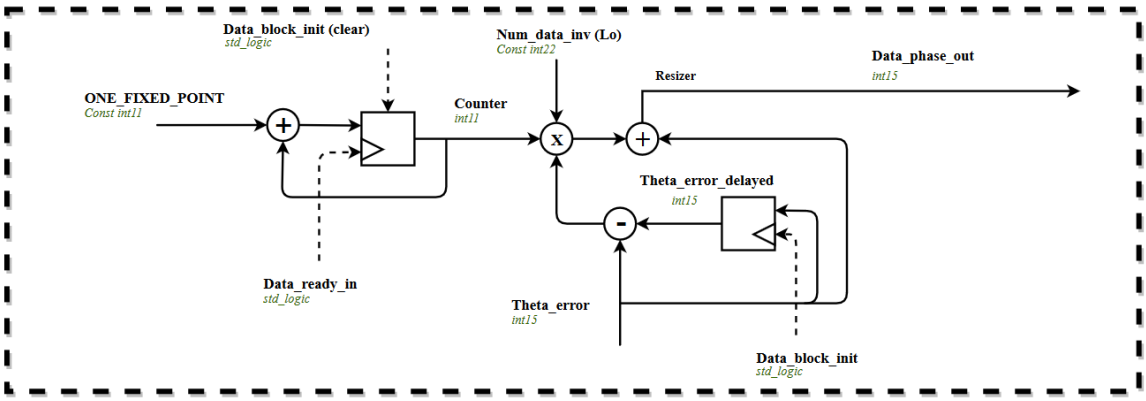


Figura 38 – Arquitetura proposta para o *LinearInterpolation*.

Este último sub-bloco será responsável por realizar a estimação de fase para a porção de símbolos de dados presente entre blocos de pilotos subsequentes, atentando-se sempre a quantidade de símbolos de dados (L_s) presente entre eles. A depender da modulação do sinal, o último bloco de dados de um *frame* possui menos símbolos de dados do que o habitual, que é 1440 símbolos. A quantidade de *slots* que este *frame* transporta também pode ser menor a depender da modulação. Na tabela 2 são ilustradas estas situações, para as modulações $\pi/2$-BPSK Q-PSK, 8-PSK e 16-APSK.

Tabela 2 – Quantidade de slots e símbolos de dados por modulação.

Modulação	Slots	Ls (último slot)
$\pi/2$-BPSK	12	360
Q-PSK	6	900
8-PSK	4	1080
16-APSK	3	1170

3.3.2 Corretor de Fase

Encerrada a estimação de fase, é chegada a hora de corrigir os símbolos de dados e pilotos de entrada, etapa realizada agora pelo bloco Corretor de Fase (*Phase Corrector*), que rotaciona os símbolos de entrada na direção oposta ao seu desvio. Na figura 39 apresenta-se a arquitetura proposta.

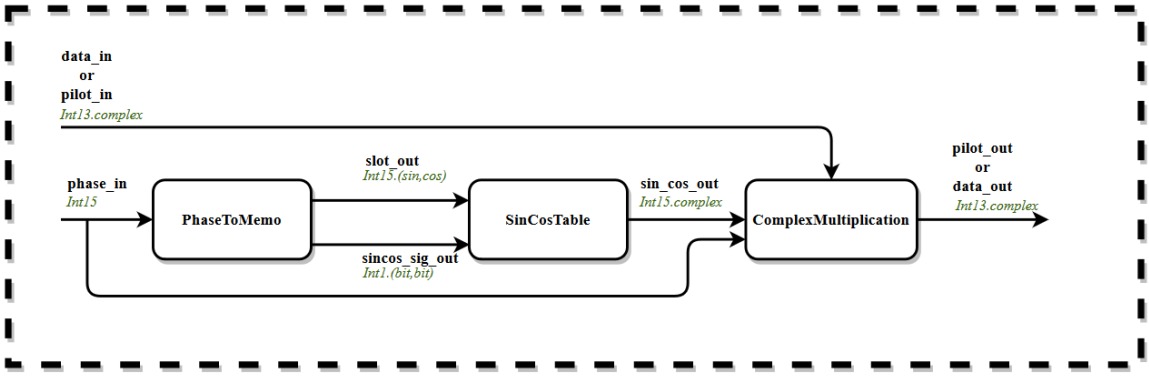


Figura 39 – Arquitetura proposta para o Corretor de Fase.

Já no início apresenta-se o sub-bloco *PhaseToMemo*, utilizado para converter o valor de desvio de fase calculado pelo bloco Estimador de Fase, podendo ser dos dados ou dos pilotos, para duas posições de memórias distintas, por meio de algumas operações de soma e subtração por valores inteiros referentes aos dois primeiros quadrantes do círculo trigonométrico (de 0 a π). A figura 40 mostra a arquitetura proposta.

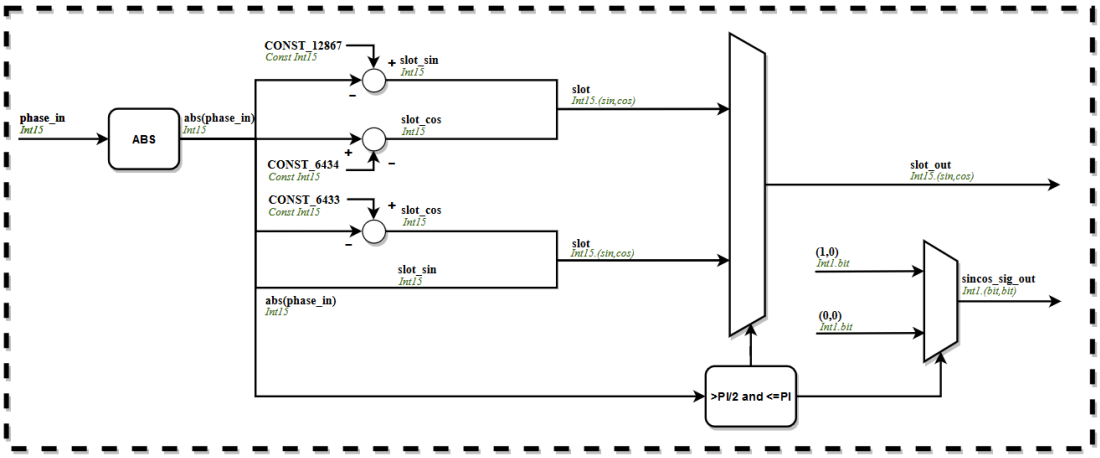


Figura 40 – Arquitetura proposta para o *PhaseToMemo*.

Este sub-bloco também informa o sinal dos valores de seno e cosseno de acordo com o valor da fase de entrada. Se for um valor maior do que $\frac{\pi}{2}$, ele indicará que o valor de cosseno a ser calculado pelo bloco posterior, o *SinCosTable*, será negativo. Como este módulo recebe apenas ângulos referentes aos dois primeiros quadrantes do círculo trigonométrico, apenas o sinal do cosseno muda.

A seguir tem-se o sub-bloco *SinCosTable*, que recebe os valores de posição de memória calculados pelo sub-bloco anterior e localiza em uma *LUT* os valores de seno e cosseno referentes a eles. Caso necessário, este sub-bloco também realiza a inversão do sinal do cosseno também de acordo com o que o sub-bloco anterior tenha enviado. Na figura 41 pode-se observar a arquitetura proposta.

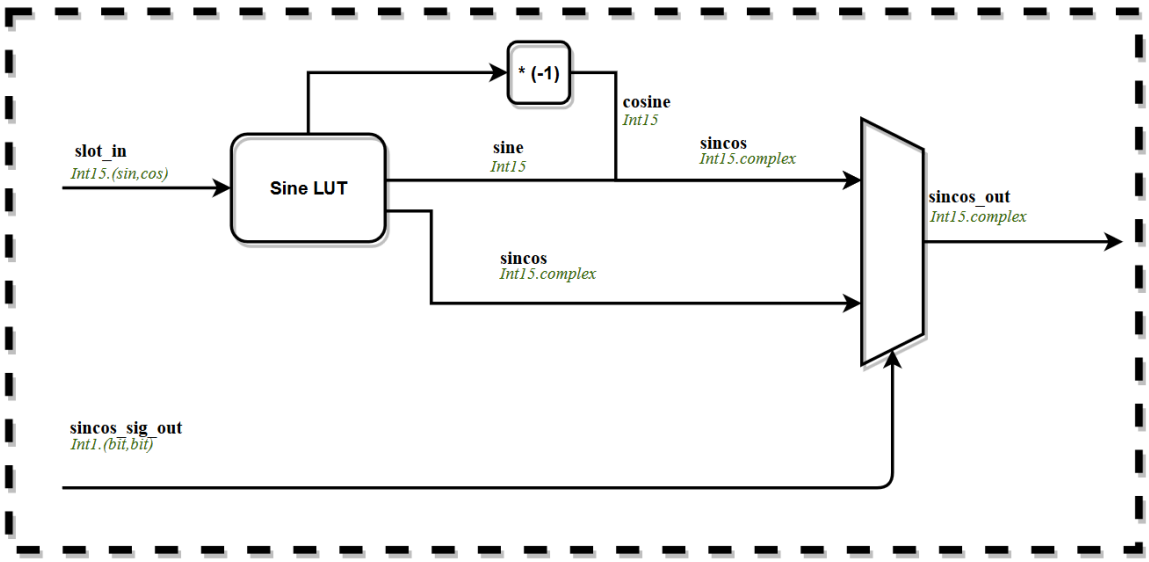


Figura 41 – Arquitetura proposta para o *SinCosTable*.

Os valores da LUT são valores de seno de apenas 1/4 do círculo trigonométrico, todos de 15 bits, sendo 12 bits de parte fracionária. Dessa forma, economiza-se bastante recursos necessários para a implementação de todo o módulo PC.

Por último, tem-se o sub-bloco encarregado de corrigir de fato os dados ou pilotos de entrada do módulo PC. Este módulo realiza a multiplicação complexa entre os símbolos de entrada (dado ou piloto) com seus respectivos valores de seno e cosseno, porém complexo conjugado, de forma a rotacionar os símbolos, realizando assim a corrição. Na figura 42 apresenta-se a arquitetura proposta.

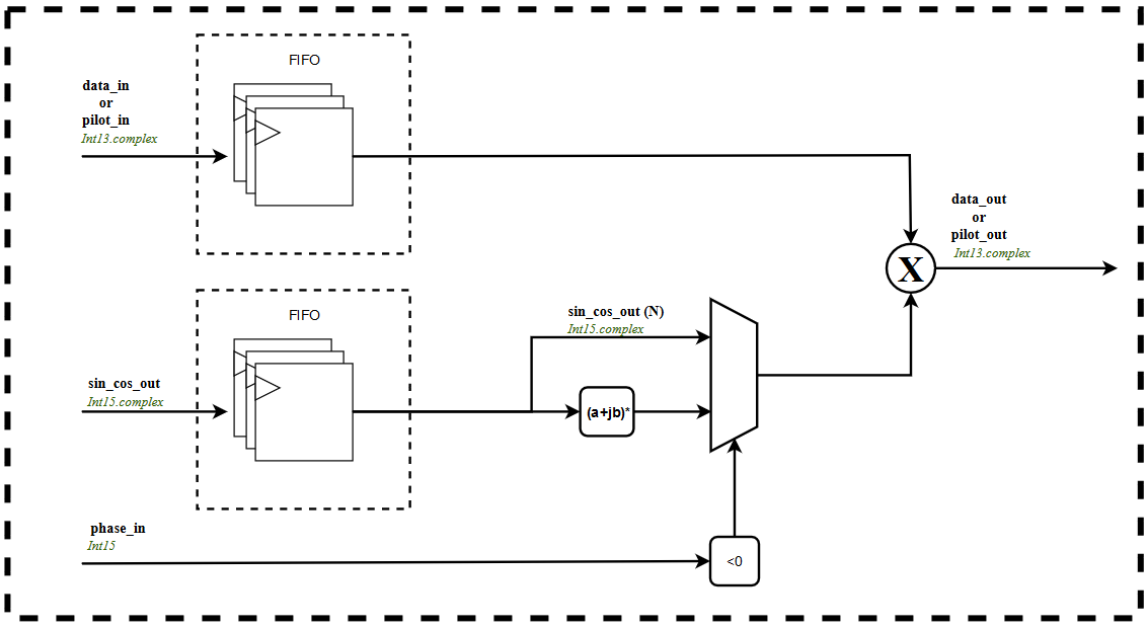


Figura 42 – Arquitetura proposta para o *ComplexMultiplication*.

Nota-se que há duas FIFOs na entrada do sub-bloco. Uma delas é responsável por armazenar os valores de dados ou piloto enquanto a estimação de fase está sendo feita, e a outra armazena os valores de seno e cossenos a ser utilizado para corrigir estes símbolos, invertendo o sinal do seno caso o valor do desvio de fase calculado seja maior do que zero, pois todos os valores de seno saem positivos do módulo anterior. O tamanho das FIFOs variam de acordo com o tipo de símbolos a serem corrigidos.

3.4 Kit de Desenvolvimento ZCU014

O kit de desenvolvimento que será utilizado é o kit Zybo Zynq-7000 ARM/FPGA SoC Trainer Board, mostrado na figura 43. O kit conta com um SoC FPGA Zynq 7010 da xilinx, e as características principais são: processador CORTEX-A9 Dual Core 667MHz; suporte a programação J-TAG, Quad-SPI flash e microSD; 4 switches; 6 pushbuttons; suporte aos protocolos UART, I2C, SPI e CAN; conector Ethernet RJ-45; conexão VGA; dentre outras.

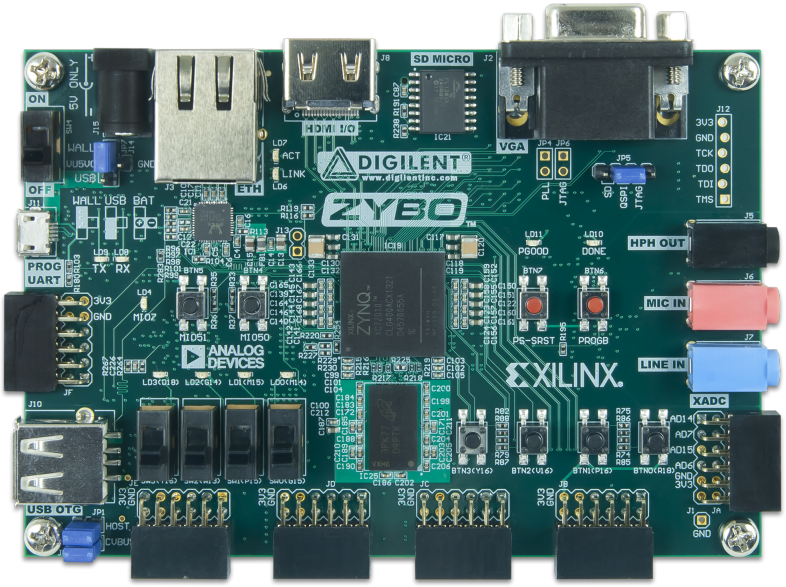


Figura 43 – Kit de desenvolvimento Zybo Zynq-7000 ARM/FPGA SoC Trainer Board (DIGILENT,)

Este kit será utilizado para a implementação dos dois módulos que serão aqui trabalhados, o CFC e o PC. Em particular, o kit de desenvolvimento será usado para a validação em hardware dos módulos desenvolvidos em VHDL.

3.5 Proposta de Verificação

3.5.1 Correção Grosseira de Frequência

Como forma de validar a implementação em hardware do CFC, serão feitas simulações comportamentais injetando vetores de teste para diferentes valores de Es/No e desvios de frequências para o CFC. A saída do módulo implementado em hardware será comparada com a saída obtida pelo simulador (modelo de referência) adaptado para operar com aritmética de ponto fixo. Como critério de validação será adotado o RMSE (*Root Mean Square Error*).

Serão feitos também testes de implementação física antes da integração com o *Transceiver* utilizando o ILA (*Integrated Logic Analyzer*), que é um analisador lógico oferecido pela ferramenta Vivado, e outros testes após a integração com o Trasciever, que contém o DDS, bem como com outros módulos, como o filtro RRC e o Time Recovery. Os testes serão feitos mediante integração com o simulador, testando os módulos sob as mesmas condições de contorno, a fim de validar o correto funcionamento do módulo CFC.

3.5.2 Correção de Fase

Para a validação do módulo PC também serão feitas simulações comportamentais injetando vetores de testes, comparando a saída do modelo em hardware com a do modelo em alto nível implementado em aritmética de ponto fixo. O critério de validação também será o RMSE.

Também serão feitos testes de implementação física do módulo PC, utilizando o ILA (*Integrated Logic Analyzer*) ou simplesmente implementando um bloco de comparação de valores de saída esperados do módulo PC implementado, baseando-se nos valores de saída que o módulo enviar nas simulações comportamentais.

3.6 Proposta de caracterização do módulo

3.6.1 Correção Grosseira de Frequência

Após a implementação de cada bloco do módulo CFC, é necessária a caracterização dos mesmos, de forma que os recursos consumidos por ele sejam aferidos. Os recursos serão analisados em termos de ocupação de LUTs, LUT-RAM, Flip-flops, DSPs e BRAMs.

Uma outra caracterização importante para uma implementação em hardware é a frequência máxima de operação do circuito, que a depender da implementação realizada, pode não ser muito maior do que a frequência padrão do clock da FPGA (no caso da Zybo Zynq-7000 será usado um clock de 100MHz). Foi justamente por isso que cada bloco do módulo CFC possui seus sinais internos de controle de operações, para garantir que as operações realizadas dentro de cada bloco fossem realizadas da forma mais estável possível. Por fim, a última caracterização que será realizada é o consumo de potência do circuito em hardware, que também pode ser estimado pelo software Vivado.

3.6.2 Correção de Fase

Assim como o CFC, o módulo PC também será caracterizado por seus recursos consumidos em termos de LUTs, LUT-RAM, Flip-flops, DSPs e BRAMs.

Também no módulo PC será caracterizada a frequência máxima de operação, que assim como no módulo CFC, pode não ser muito alta, ainda mais pelo fato de que este módulo possui muitos blocos e sub-blocos, consequentemente mais operações, dificultando o aumento da frequência de operação. O consumo de potência do módulo completo também será estimado pelo software Vivado.

4 Resultados

4.1 Resultados de Síntese

4.1.1 Correção Grosseira de Frequência

Ao fim da implementação do módulo, foi utilizada a ferramenta do Vivado de geração de esquemático para poder visualizar se a arquitetura planejada refletia na arquitetura codificada. Na figura 44 pode-se visualizar a arquitetura implementada ainda sem a integração com o *Transceiver*, em malha aberta.

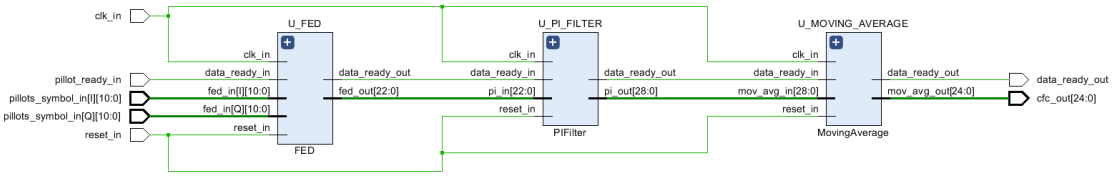


Figura 44 – Diagrama esquemático implementado em VHDL para o módulo de Correção Grosseira de Frequência.

Para fins de sincronismo, cada operação intermediária dos blocos acima eram acionada por sinais intermediários, que garantiam que a próxima operação fosse realizada com os dados estáveis. Sem estes sinais internos, o tempo de hold (*hold time*) e o tempo de setup (*setup-time*) de algum bloco poderiam ficar comprometidos, de forma que os resultados das operações poderiam se adiantar, se atrasar, ou até mesmo não saírem para as operações subsequentes. Como observado, a arquitetura implementada, com exceção do *Transceiver*, está condizente com que se apresenta na etapa de planejamento do projeto.

Verificada a coerência entre o que foi planejado e o que foi implementado, realizou-se a síntese do projeto e foi extraído o consumo de recursos do módulo CFC. A seguir, na tabela 3 apresenta-se os resultados obtidos:

Tabela 3 – Consumo de recursos do módulo CFC implementado em hardware.

Recursos	Estimados	Disponíveis	Utilização (%)
LUT	357	17600	2,03
LUTRAM	58	6000	0,97
FF	367	35200	1,04
DSP	3	80	3,75
IO	51	100	51,00
BUFG	1	32	3,13

O número de *Look-Up-Tables* consumidos se dá por conta das operações de somas e multiplicações que acontecem dentro do módulo. Os flip-flops consumidos pelo módulo são por conta das FIFOs e registradores que foram utilizados na implementação. Os blocos de DSPs utilizados foram poucos, algo satisfatório para uma implementação em hardware. Como o módulo está com suas entradas e saídas desconectadas, o consumo de IOs foi além do esperado.

A seguir, na tabela 4 apresenta-se a interface de conexão do módulo CFC, na tabela 4, listando todas as entradas e saídas do módulo completo.

Tabela 4 – Interface de conexão do módulo CFC.

Nº de elementos	Nº de bits por elemento	Nome	Descrição
1	1	clk_in	Sinal de clock.
1	1	reset_in	Reset Assíncrono.
2	11	pilot_symbol_in	Entrada de símbolo pilotos.
1	1	pilot_ready_in	indica que um símbolo de piloto está disponível para leitura.
1	1	data_ready_out	Indica que uma estimação de frequência está disponível para leitura
1	25	cfc_out	Estimativa de erro de frequência.

Na tabela 5 apresenta-se o consumo de recursos de módulos CFC implementados por Lima et al. em diferentes trabalhos. Junto a eles apresenta-se o consumo de recursos consumidos pelo módulo implementado neste trabalho. Nota-se que o módulo CFC aqui implementado foi mais econômico no consumo de recursos.

Tabela 5 – Consumo do módulo CFC em diferentes trabalhos.

(de Lima et al., 2014)	(LIMA et al., 2013)	Módulo Implementado
LUT:5895 Registradores: 3709 Bits de Memória: 64 DSPs: 96	LUT:41892 Registradores: 32315 Bits de Memória: 1488 DSPs: 507	LUT:357 LUTRAM: 58 Flip-Flops: 367 DSPs: 3

4.1.2 Correção de Fase

Encerrada a implementação do módulo PC, na ferramenta Vivado utilizou-se a ferramenta de visualização de esquemático para a geração do esquemático da figura 45, verificando se o que foi gerado refletia a arquitetura planejada. Observa-se que o esquemático gerado pela ferramenta reflete a arquitetura planejada para o módulo completo.

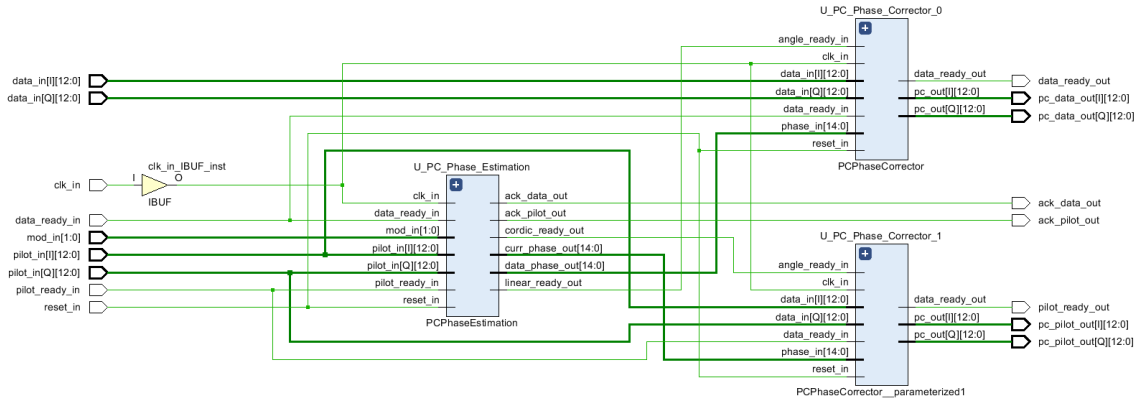


Figura 45 – Diagrama esquemático implementado em VHDL para o módulo de Correção de Fase.

Como dito na Metodologia e assim como no módulo CFC, o módulo PC também apresenta sinais de sincronismo entre os blocos e sub-blocos, de forma que o sincronismo seja garantido entre eles, evitando que o tempo de setup e tempo de hold fiquem comprometidos, de forma que os resultados que saem de algum bloco ou sub-bloco possa sair errado.

Ainda na ferramenta de geração de esquemáticos, foi analisado se o esquemático para o bloco de estimação de fase também refletia a arquitetura planejada. A seguir, na figura 46, o seu diagrama gerado:

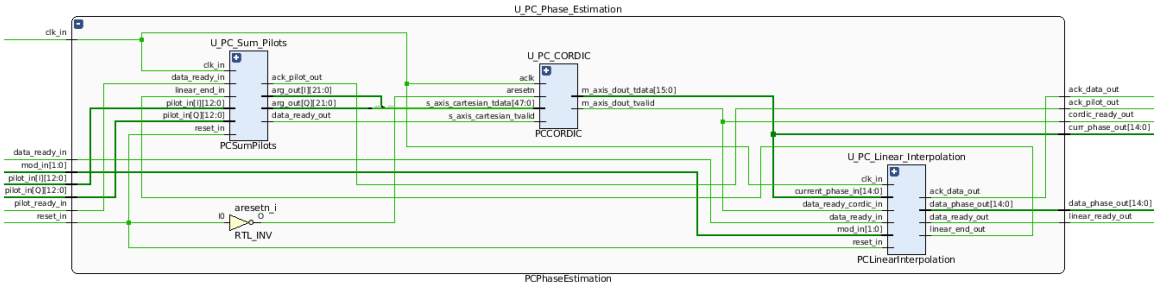


Figura 46 – Diagrama esquemático implementado em VHDL para o bloco Estimador de Fase.

Novamente os mesmos passos foram feitos para verificar se o bloco corretor de fase foi implementado de acordo com a arquitetura planejada. Na figura 47 apresenta-se o seu diagrama:

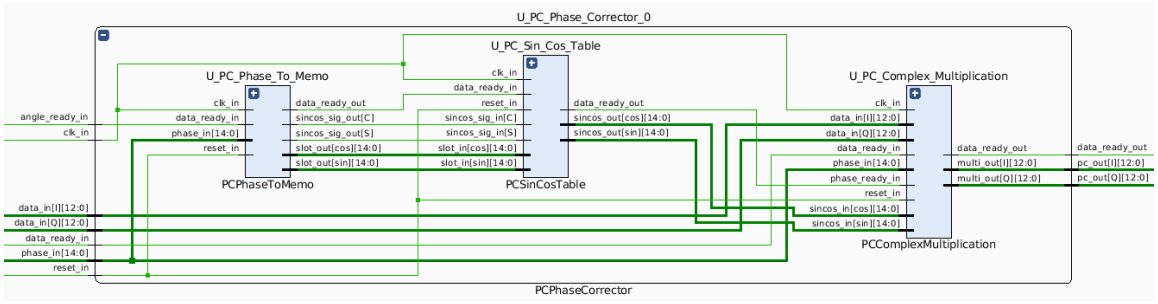


Figura 47 – Diagrama esquemático implementado em VHDL para o bloco Corretor de Fase.

Verificado os esquemáticos, realizou-se a etapa de síntese do módulo PC e extraiu-se o consumo de recursos, que estão apresentados a seguir, na tabela 6:

Tabela 6 – Consumo de recursos do módulo PC implementado em hardware.

Recursos	Estimados	Disponíveis	Utilização %
LUT	7832	17600	44,50
LUTRAM	3	6000	0,05
FF	2641	35200	10,27
DSP	14	80	17,50
IO	114	100	104,00
BUFG	1	32	3,13

O alto número consumido de LUTs se deu por conta principalmente da tabela de senos e cossenos, onde estão contidos os valores de senos do primeiro quadrante do círculo trigonométrico. os Flip-Flops consumidos neste módulo são por conta dos registradores e FIFOs utilizados na implementação. As DSPs que foram utilizadas neste módulo encarregaram-se de realizar as multiplicações, sendo todas multiplicações complexas. Assim como no CFC, o alto número de IOs consumidas se dá por conta do módulo estar com suas entradas e saídas desconectadas.

Na tabela a seguir (7), apresenta-se a interface de conexão do módulo PC, listando todas as entradas e saídas do módulo completo:

Tabela 7 – Interface de conexão do módulo PC.

Nº de elementos	Nº de bits por elemento	Nome	Descrição
1	1	clk_in	Sinal de clock.
1	1	reset_in	Reset Assíncrono.
2	13	data_in	Entrada de símbolo de dados.
2	13	pilot_in	Entrada de símbolo pilotos.
1	1	data_ready_in	indica que um símbolo de dado está disponível para leitura.
1	1	pilot_ready_in	indica que um símbolo de piloto está disponível para leitura.
1	2	mod_in	Informa ao módulo qual a modulação está sendo feita.
1	1	data_ready_out	Indica que um símbolo de dado foi corrigido e está disponível para leitura.
1	1	ack_data_out	Indica que o módulo está disponível para receber um símbolo de dados.
1	1	pilot_ready_out	Indica que um símbolo piloto foi corrigido e está disponível para leitura.
1	1	ack_pilot_out	Indica que o módulo está disponível para receber um símbolo de piloto.
2	13	pc_pilot_out	Símbolo piloto corrigido.
2	13	pc_data_out	Símbolo de dado corrigido.

Na tabela 8 apresenta-se o consumo de recursos do módulo PC implementado em diferentes trabalhos de Lima et al., juntamente com o consumo de recursos do módulo PC aqui implementado.

Tabela 8 – Consumo do módulo PC em diferentes trabalhos.

(de Lima et al., 2014)	(LIMA et al., 2013)	Módulo Implementado
LUT:2663 Registradores: 2179 Bits de Memória: 1728 DSPs: 27	LUT:2358 Registradores: 1892 Bits de Memória: 0 DSPs: 8	LUT:7832 LUTRAM: 3 Flip-Flops: 2641 DSPs: 14

O consumo de recursos do PC implementado neste trabalho superou, em grande parte, o consumo de recursos dos trabalhos citados. Isso se dá pelo fato de que os autores destes trabalhos utilizaram outras formas para a correção dos símbolos, como o CORDIC. Em compensação, a velocidade do cálculo do seno e cosseno por meio de LUTs é muito maior.

4.2 Simulação comportamental

4.2.1 Correção Grosseira de Frequência

Para averiguar o funcionamento deste módulo, foi implementada uma simulação comportamental para comparar as saídas do modelo de referência em Python com as saídas do módulo implementado em hardware. Os resultados podem ser vistos na figura 48, onde as saídas do bloco implementado em hardware e a do modelo de simulação em Python estão representadas como uma forma de onda contínua, se assemelhando muito a uma forma de onda dente-de-serra, devido ao comportamento do módulo ainda em malha aberta, apenas acumulando estimativas de erros. A primeira forma de onda é do módulo implementado em hardware, já a segunda do modelo em ponto fixo feito em Python.

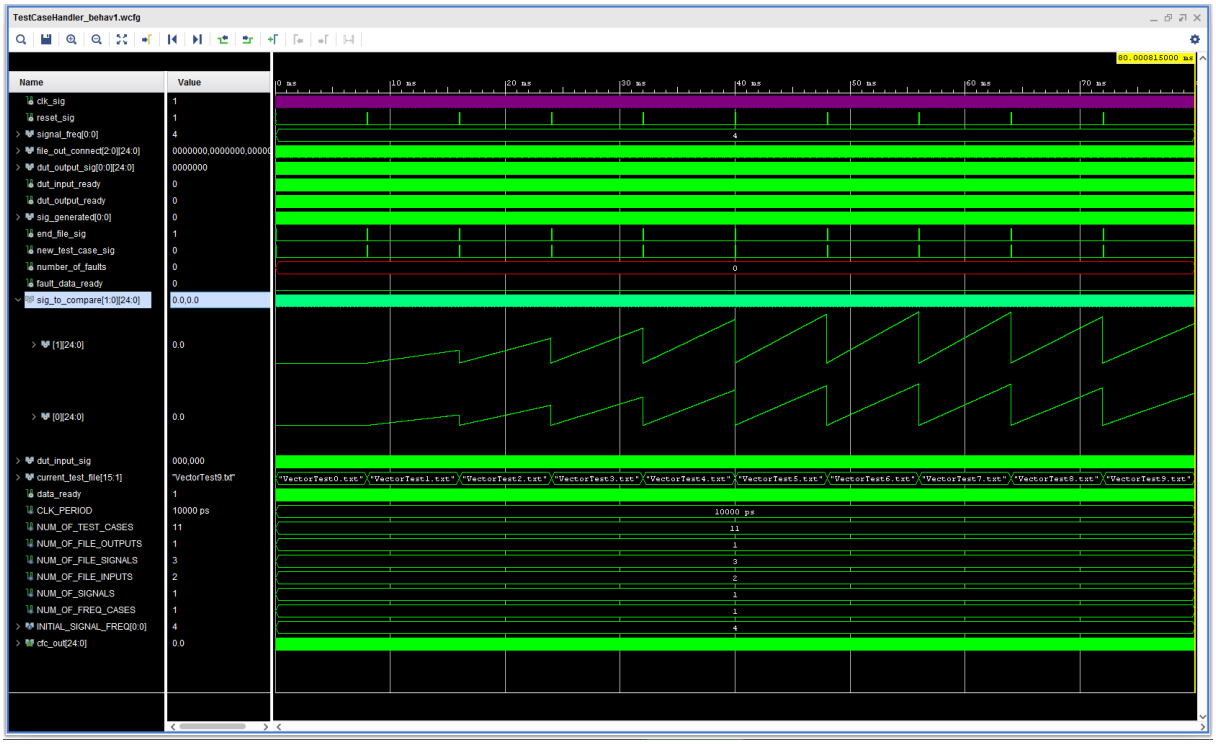


Figura 48 – Simulação comportamental do módulo CFC para diferentes desvios de frequências.

Foram testados 10 casos contendo 10 desvios de frequências diferentes, variando de 0,02 a 0,2 em frequência normalizada, com um sinal modulado em QPSK, com um Es/No de -2dB. Como observado, a arquitetura implementada conseguiu acumular estes erros da mesma forma como a implementação em software, acompanhado a curva de acúmulos de estimativas de erros de frequência.

Uma outra simulação foi realizada, desta vez foram testados diferentes valores de Es/No, variando de -2 a 10dB, em passos de 2dB. Abaixo pode-se visualizar a saída do módulo em relação a saída do simulador em Python. Novamente, a primeira forma de onda é do módulo em hardware. A segunda, do modelo de referência em Python.

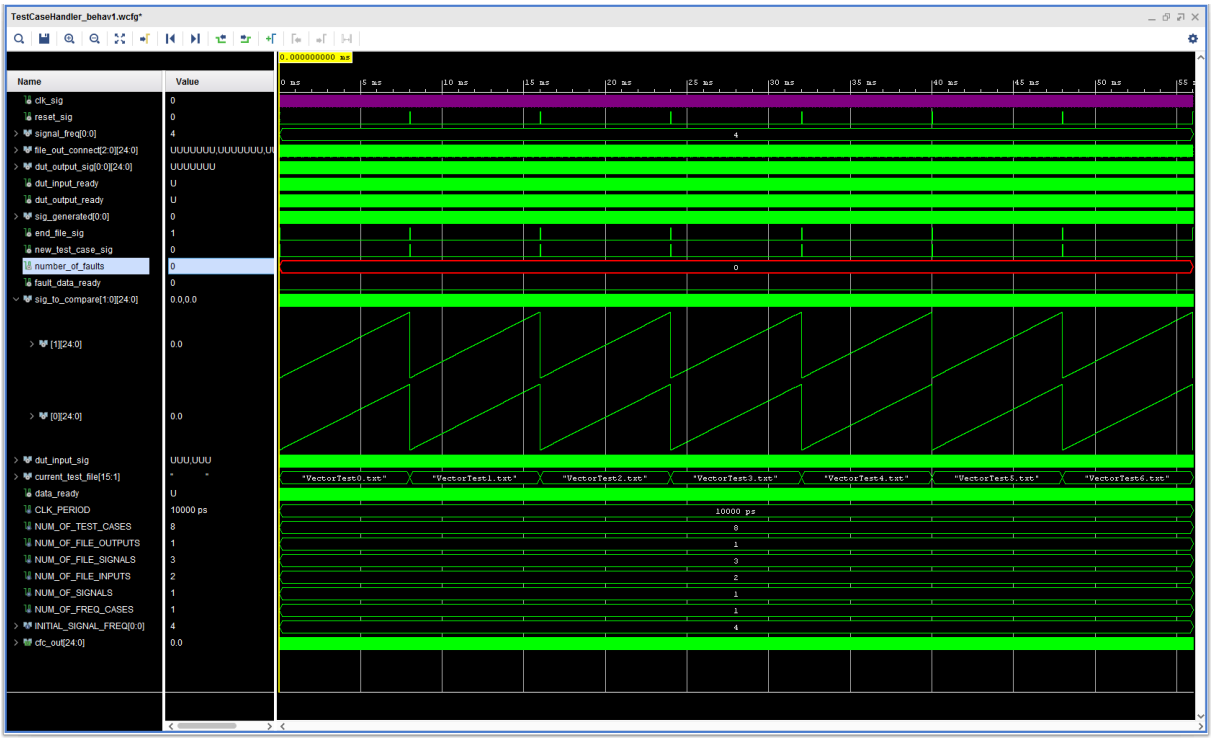


Figura 49 – Simulação comportamental do módulo CFC para diferentes valores de Es/No.

Nesta simulação, novamente o módulo em hardware conseguiu acompanhar o modelo em Python, também sem erros.

Devido a latência do módulo como um todo, cada estimativa de erro saía a cada 4 ciclos de *clock*, gerando um atraso de 1 ciclo de *clock* de uma estimativa do software para a estimativa em hardware. Isso foi corrigido facilmente atrasando as saídas do modelo de referência na simulação comportamental.

A partir das simulações realizadas, foi possível estimar a latência e a taxa de processamento (*throughput*) do módulo. Sua latência foi estimada em 9 ciclos de relógio, o que para um período de clock de 10ns significa uma latência de 90ns. Já com relação a taxa de transferência, cada estimativa de erro tinha um atraso de 40ns, ou 4 ciclos de relógio, o que significa que o módulo tem uma taxa de processamento de $\frac{1}{40ns} = 25$ MSPS

4.2.2 Correção de Fase

De forma a averiguar o bom funcionamento do módulo PC, implementou-se uma simulação comportamental para comparar as saídas do modelo de referência em Python (ponto fixo) com as saídas do módulo implementado em hardware. O resultado da simulação pode ser visto na figura 50, onde foram simulados dois *frames* de uma modulação Q-PSK, com um desvio de frequência de 0.1. As quatro primeiras formas de onda em amarelo são os símbolos pilotos corrigidos, enquanto as outras quatro formas de ondas restantes são os símbolos de dados corrigidos. As posições ímpares dos vetores que contém os dados e pilotos corrigidos contém os valores de saída do modelo de referência em Python, enquanto as posições pares contém os valores de saída do modelo em hardware.

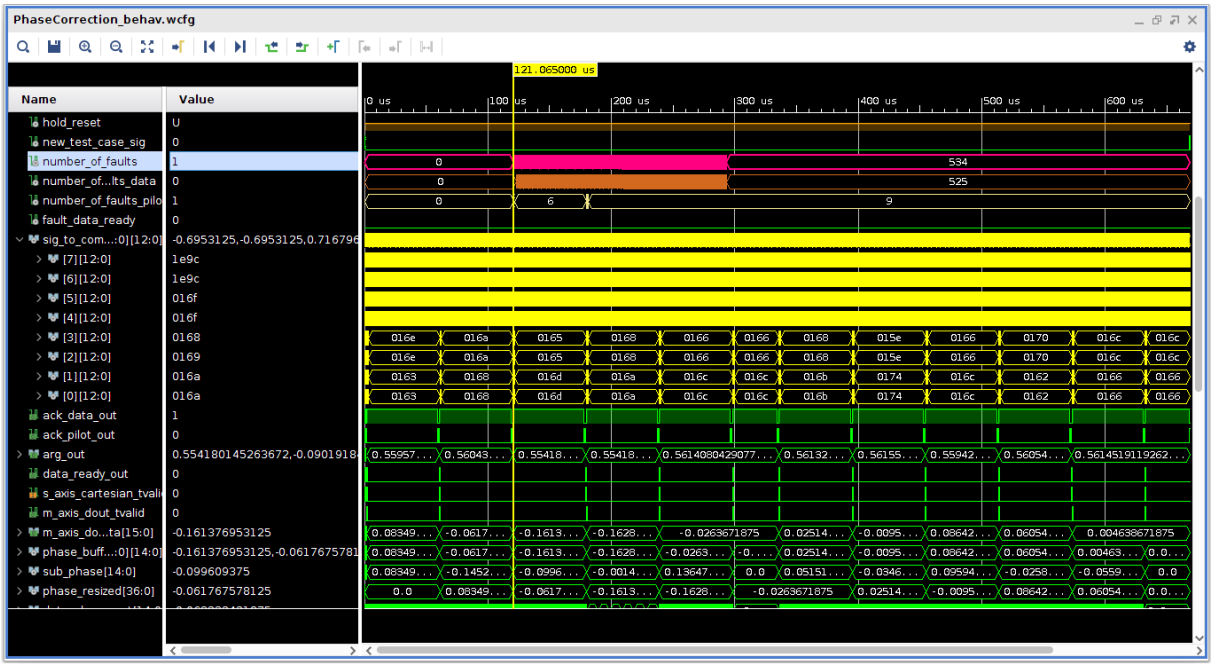


Figura 50 – Simulação comportamental do módulo PC para dois frames Q-PSK.

De forma a testar se a mudança de modulação está funcionando para a interpolação linear, foram realizadas outras simulações comportamentais mudando a modulação. Na figura a seguir (51) apresenta-se uma outra simulação do módulo PC, dessa vez para uma modulação 8-PSK:

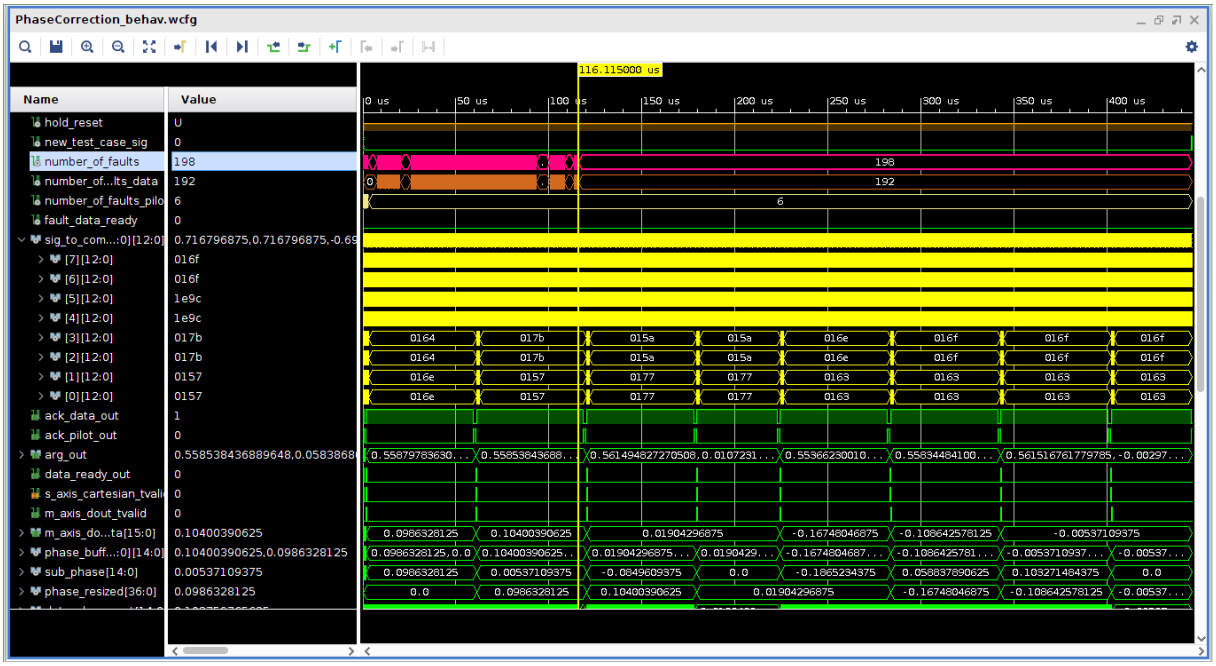


Figura 51 – Simulação comportamental do módulo PC para dois frames 8-PSK.

Por fim, na figura 52 apresenta-se a simulação comportamental para a modulação 16-APSK:

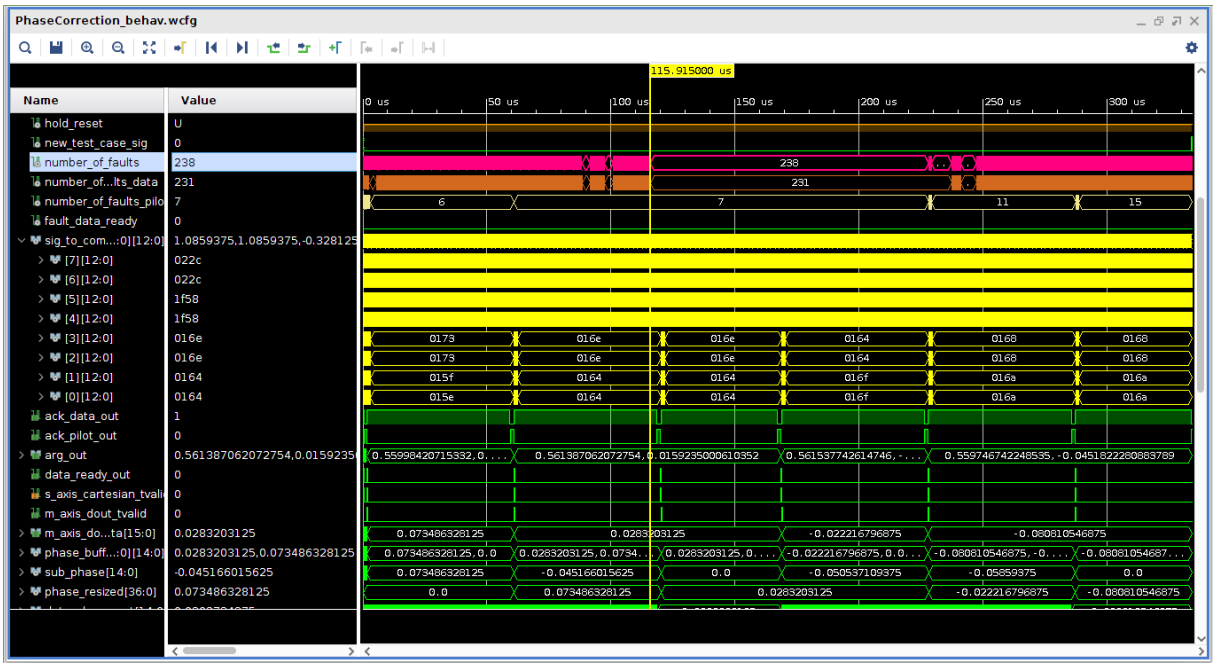


Figura 52 – Simulação comportamental do módulo PC para dois frames 16-APSK.

Nestas simulações nota-se que o módulo implementado em hardware e o modelo de referência se diferiram em alguns valores de saída. Isso se deu porque o modelo de referência calcula o ângulo utilizando uma série de Taylor do arco-tangente, enquanto o modelo em hardware utiliza um CORDIC fornecido pela ferramenta Vivado.

A partir das simulações realizadas, foi possível estimar a latência e taxa de processamento do módulo PC. A latência do módulo foi de 178 ciclos de relógio, grande parte sendo por causa da etapa de estimação de fase. As saídas do módulo tinham um atraso de 4 ciclos de relógio, o que significa uma taxa de processamento de $\frac{1}{40ns} = 25\text{MSPS}$.

4.3 Implementação em Hardware

4.3.1 Correção Grosseira de Frequência

De forma a validar a implementação física do módulo CFC, foi criada um circuito para realizar o teste em hardware, com sua arquitetura mostrada na figura 53 a seguir:

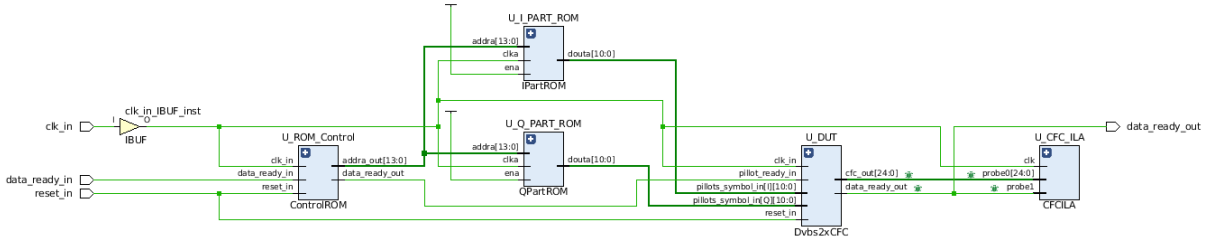


Figura 53 – Arquitetura do teste em hardware do módulo CFC.

Nesta figura pode-se observar que há quatro blocos novos: o primeiro deles é o ILA (CFCILA), que é um analisador lógico utilizado para capturar as saídas geradas pelo módulo CFC e exportá-las para poder compará-las com as saídas esperadas na sua simulação comportamental. Tem-se também dois blocos de memórias ROMs (IPartROM e QPartROM), cada uma armazenando 10000 valores de vetores de testes, simbolizando os símbolos pilotos de entrada. Por fim, para controlar estas memórias ROMs apresenta-se o bloco controlador de memórias ControlROM, que é responsável por incrementar os endereços de memória das duas memórias aqui implementadas.

A partir desta arquitetura foi possível extrair informações a respeito do tempo de *setup* e de *hold* do módulo CFC. A seguir, na tabela 9 estão estas informações. É importante lembrar que, além das restrições de IO's convencionais, foram adicionadas restrições de tempo para o módulo.

Tabela 9 – Análise de *Timing* do módulo CFC

Setup (ns)	Hold (ns)
0,893	0,047

Como observado, não houve nenhum valor negativo, indicando que o circuito implementado não apresenta problemas de temporização.

Extraídas as estimativas de tempo, foi extraído também o *layout* de ocupação do circuito de teste em hardware de módulo CFC na placa. A seguir, na figura 54, é possível observar como ficou a ocupação do circuito do teste em hardware do CFC. Nota-se que não foi ocupado muito espaço na placa.

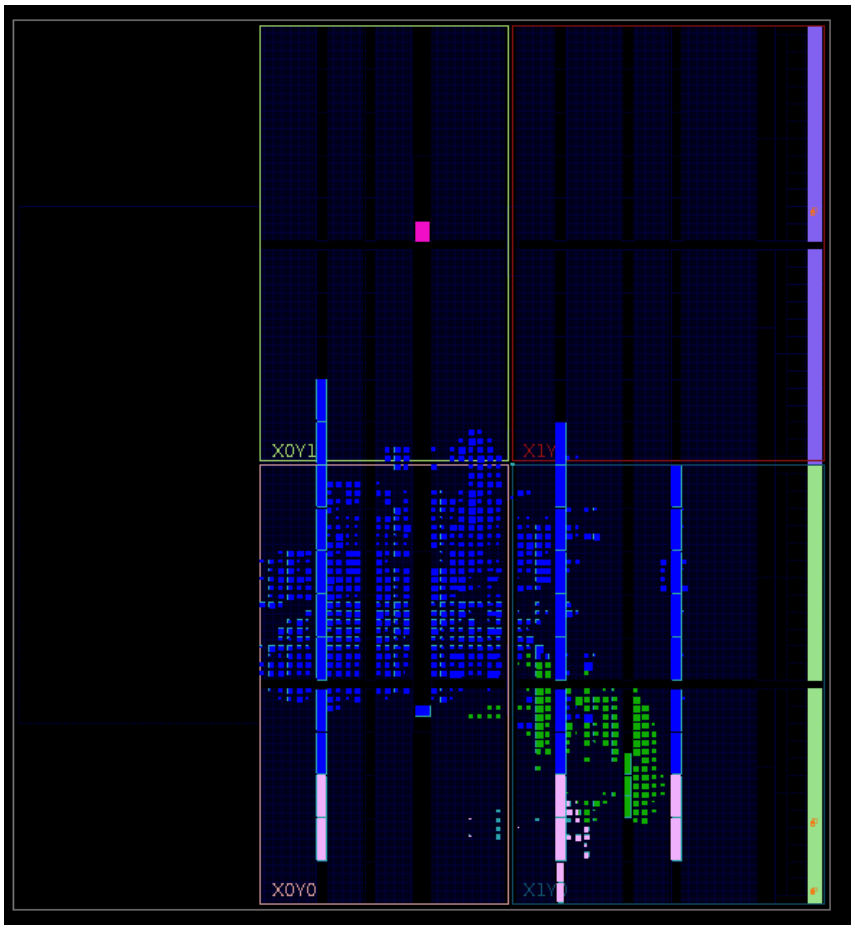


Figura 54 – Ocupação na placa do circuito de teste do módulo CFC. Em azul, o ILA. Em verde, o módulo CFC. Em bege, as memórias ROMs e o bloco ControlROM.

A seguir apresenta-se a figura 55 com as estimativas de consumo de potência do circuito implementado para o teste em hardware do módulo CFC. Nota-se que a maior parte do consumo de potência é realizado pela parte estática do circuito.

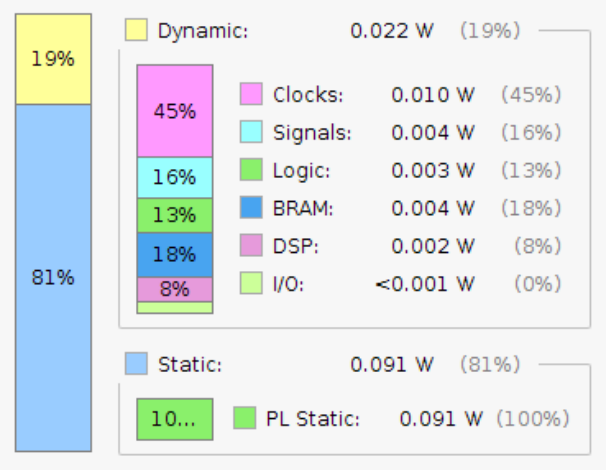


Figura 55 – Consumo de potência do circuito de testes do módulo CFC.

Por fim, o teste em hardware foi realizado. Os vetores de testes utilizados foram de uma simulação com desvio de frequência de 0.2 e Es/No de -2dB. Na figura 56 apresenta-se o resultado deste teste em hardware.

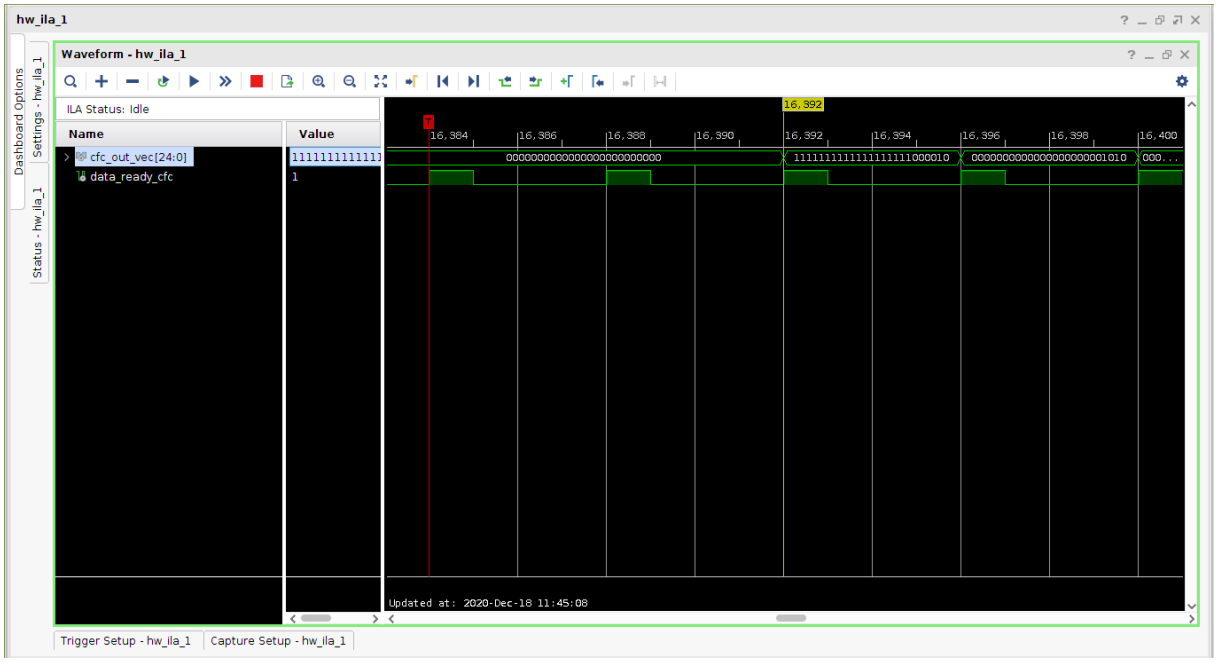


Figura 56 – Teste em hardware do módulo CFC utilizando o ILA.

Realizado o teste em hardware, foram exportadas suas saídas para uma planilha e foram extraídos os 10000 valores de saídas esperados e comparou-se com os valores esperados na simulação comportamental. Todos os valores estavam idênticos, validando assim o teste em hardware.

4.3.2 Correção de Fase

Para realizar o teste em hardware do módulo PC, também foi implementado um circuito responsável por realizar este teste, que pode ser observado na figura 57 a seguir:

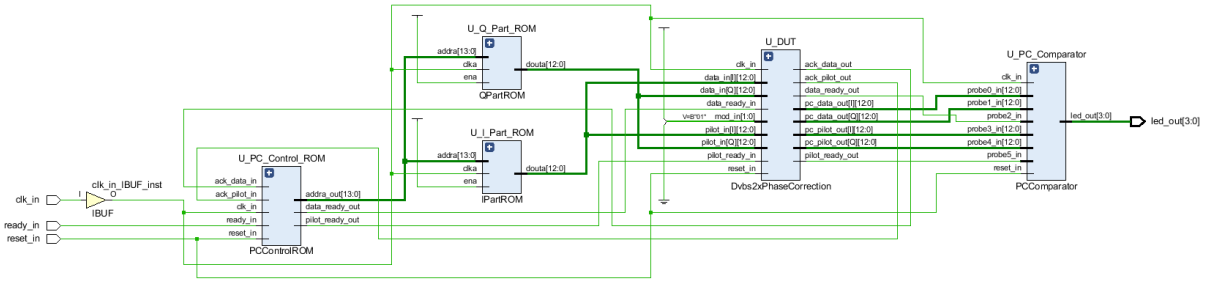


Figura 57 – Arquitetura do teste em hardware do módulo PC.

Observa-se que esse circuito é muito similar ao circuito do teste em hardware do módulo CFC, porém desta vez não foi utilizado o ILA para capturar as saídas do módulo, mas sim um outro bloco chamado de *PCComparator*, que foi implementado de forma que, ao receber as saídas de dados ou pilotos corrigidas, ele irá comparar com seus valores esperados na simulação comportamental (valores do modelo em hardware), que estão armazenados em memórias ROMs, e contabilizar quantos valores saíram conforme o esperado. Se todos os valores de dados e pilotos forem corrigidos conforme o esperado, ele irá acender dois LEDs na placa. As memórias ROMs neste circuito estão com 8316 símbolos cada uma, dentre eles 8100 símbolos de dados e 216 símbolos de pilotos. A modulação escolhida para o teste em hardware foi a modulação Q-PSK.

Após a implementação do circuito, foram extraídas informações de *Timing* do circuito, mostrada na tabela a seguir (10. Novamente, foram feitas restrições tanto de IO's quanto de tempo.

Tabela 10 – Análise de *Timing* do módulo PC

Setup (ns)	Hold (ns)
0,068	0,074

Novamente, como não há nenhum valor negativo tanto do tempo de *setup* quanto do tempo de *hold*, o circuito não apresenta problemas de temporização.

Após estimado os valores de *Timing*, foi extraído o *layout* do circuito de teste em hardware do módulo PC. Na figura 58 pode-se observar o layout. Observa-se que foi ocupado um grande espaço na placa, principalmente por conta do bloco comparador (*PCComparator)*:

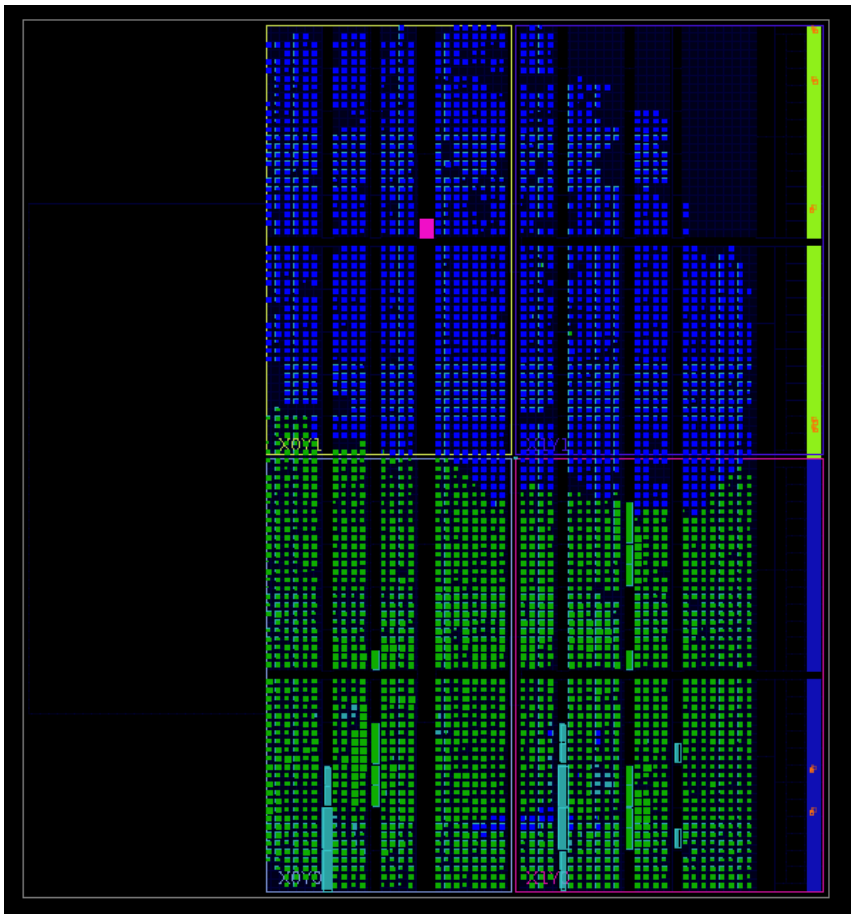


Figura 58 – Ocupação na placa do circuito de teste do módulo PC. Em azul, o *PCComparator*. Em verde, o módulo PC. Em turquesa, as memórias ROMs e o bloco ControlROM.

A seguir foi extraído o consumo de potência do circuito, que pode ser observado na figura 59. Observa-se que, como este módulo apresenta muito mais sinais e lógicas sequenciais e combinacionais, o consumo de potência dinâmico ficou maior do que o consumo estático.

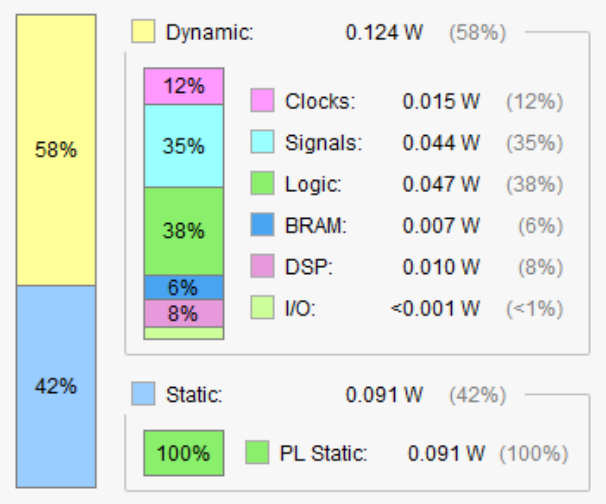


Figura 59 – Consumo de potência do circuito de testes do módulo PC.

Por fim, foi realizado o teste em hardware do módulo PC. Primeiro o módulo foi reiniciado por meio de um botão. Após a reinicialização, pressionou-se um botão de *ready* para iniciar o módulo. Como mostra na figura 60, os dois LEDs acenderam, sendo que o mais à direita indica que todos os símbolos de dados foram corrigidos corretamente, e o LED mais à direita indica que todos os símbolos pilotos foram corrigidos corretamente mostrando que o módulo PC comportou-se como esperando, validando assim o teste em hardware.

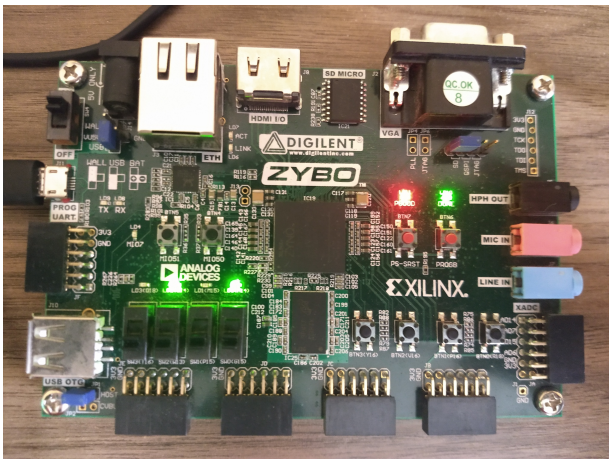


Figura 60 – Teste em hardware do módulo PC.

4.4 Curvas características

4.4.1 Correção Grosseira de Frequência

Ao se realizar as simulações comportamentais, foram gerados 10 arquivos contendo as saídas do módulo CFC para a primeira simulação, e 7 arquivos para a segunda simulação. Estes arquivos foram utilizados para comparar com os arquivos gerados da simulação em software para calcular o RMSE, com o intuito de caracterizar o módulo CFC. Como não houve nenhum erro entre o modelo de referência em ponto fixo e o modelo em hardware, as saídas foram comparadas com o modelo de referência em ponto flutuante.

A partir destes arquivos coletados, foram feitas curvas para caracterizar o módulo CFC. A primeira curva feita foi a curva do RMSE por desvio de frequência. Pode-se ver na figura 61 como ela se comportou.

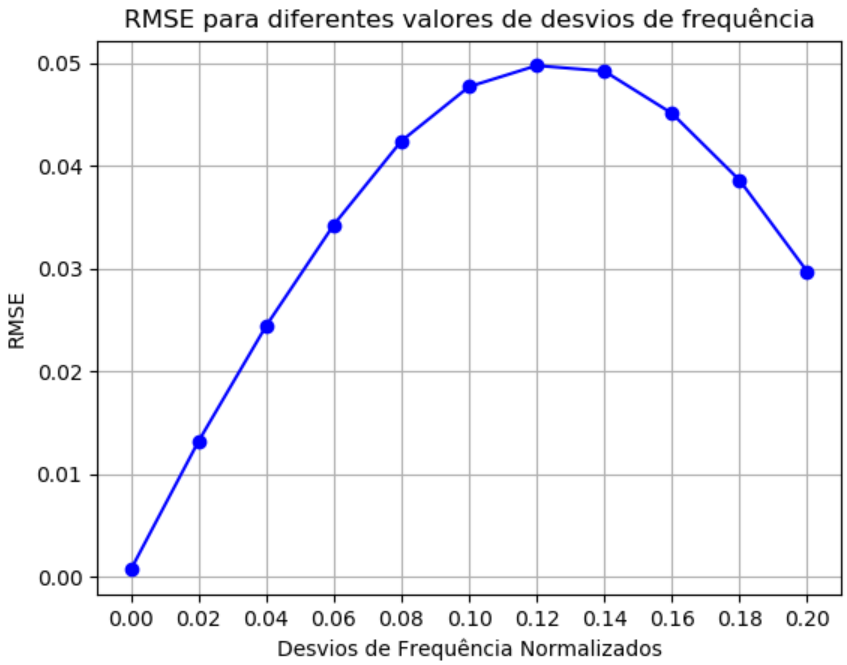


Figura 61 – Curva RMSE x Desvios de Frequência Normalizado.

Nota-se que o RMSE diminui a partir do desvio de frequência de 0,12. O que acontece é que, para desvios de frequências maiores do que este, as entradas ruidosas do módulo começam a ter mais componentes negativas, fazendo com que a média móvel seja, de certa forma, decrementada, fazendo com que as estimativas de erro cresçam de forma mais vagarosa em relação a estimativas feitas com desvios de frequências um pouco menores.

Já na figura 62 pode-se ver, para um desvio de frequência fixo em 0.2, o RMSE para diferentes valores de Es/No, variando de -2 a 10dB.

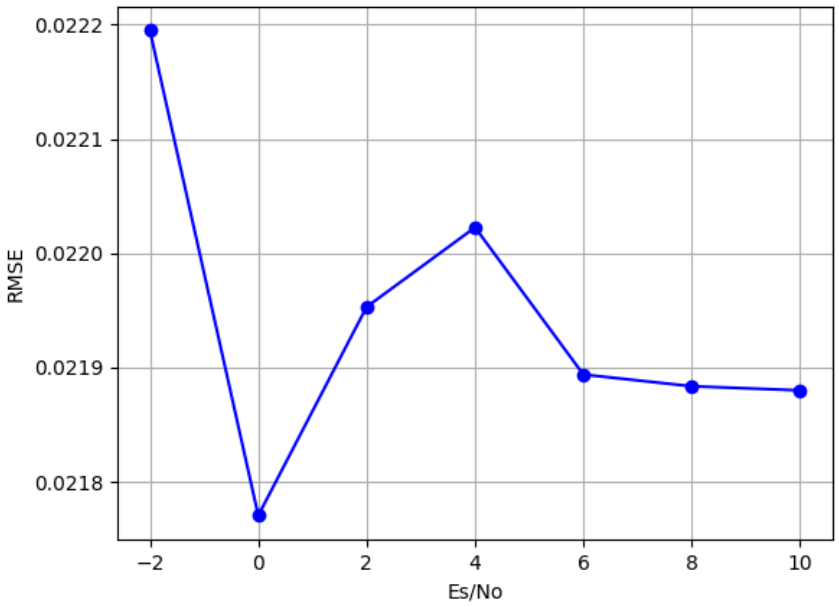


Figura 62 – Curva RMSE x Es/No.

Como o módulo de correção grosseira de frequência está apenas acumulando os erros estimados, o RMSE não teve um decaimento tão contínuo. A partir de 4dB de Es/No o decaimento realmente acontece, mostrando que para valores maiores de Es/No o RMSE de fato fica menor. Pode-se observar também que os valores estão muito próximos, indicando que uma variação de Es/No não influencia tanto nessa curva de RMSE.

Já na terceira curva a seguir (figura 63) pode-se observar o RMSE para diferentes quantidades de blocos pilotos, para um desvio de frequência de 0.2 em frequência normalizada, com um Es/No de -2dB:

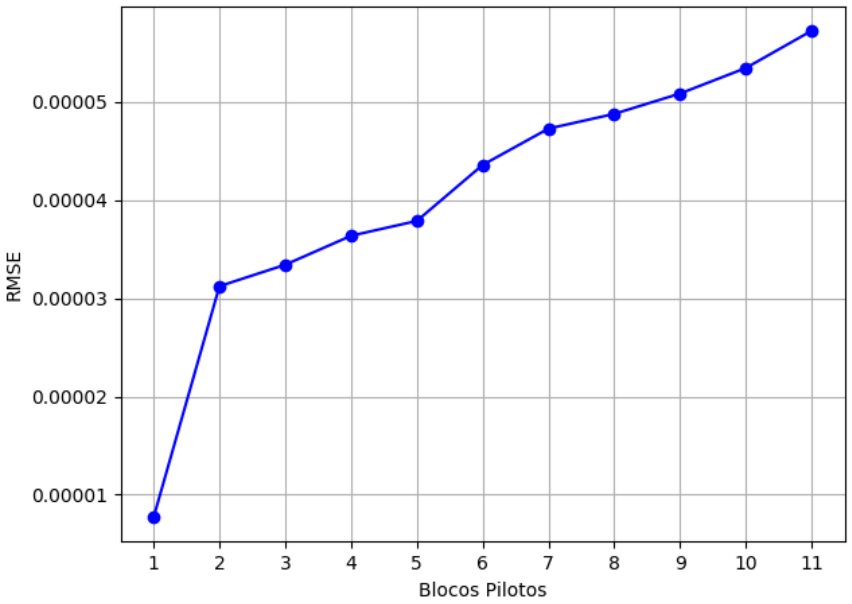


Figura 63 – Curva RMSE x blocos-pilotos.

Por se tratar ainda de um módulo sem a integração com o módulo *Transceiver*, consequentemente apenas acumulando as estimativas de erros em uma média móvel, sem ainda corrigir as entradas, o RMSE tende a crescer a medida que os blocos pilotos passam. Este valor tende a um limite, que pode ser visto na figura 61 ao se olhar no ponto do desvio de frequência normalizado correspondente.

4.4.2 Correção de Fase

Se tratando do módulo PC, a partir das saídas da simulação comportamental obteve-se as constelações corrigidas para o modelo em hardware e comparou-se com o modelo de referência em alto nível em ponto fixo, calculando o RMSE. A seguir, na figura 64, apresentam-se as constelações Q-PSK corrigidas dos dois *frames*, sendo as duas figuras da esquerda (figuras 64a e 64c) as constelações do modelo em hardware. Já as figuras da direita (figuras 64b e 64d) são as constelações referentes ao modelo de referência.

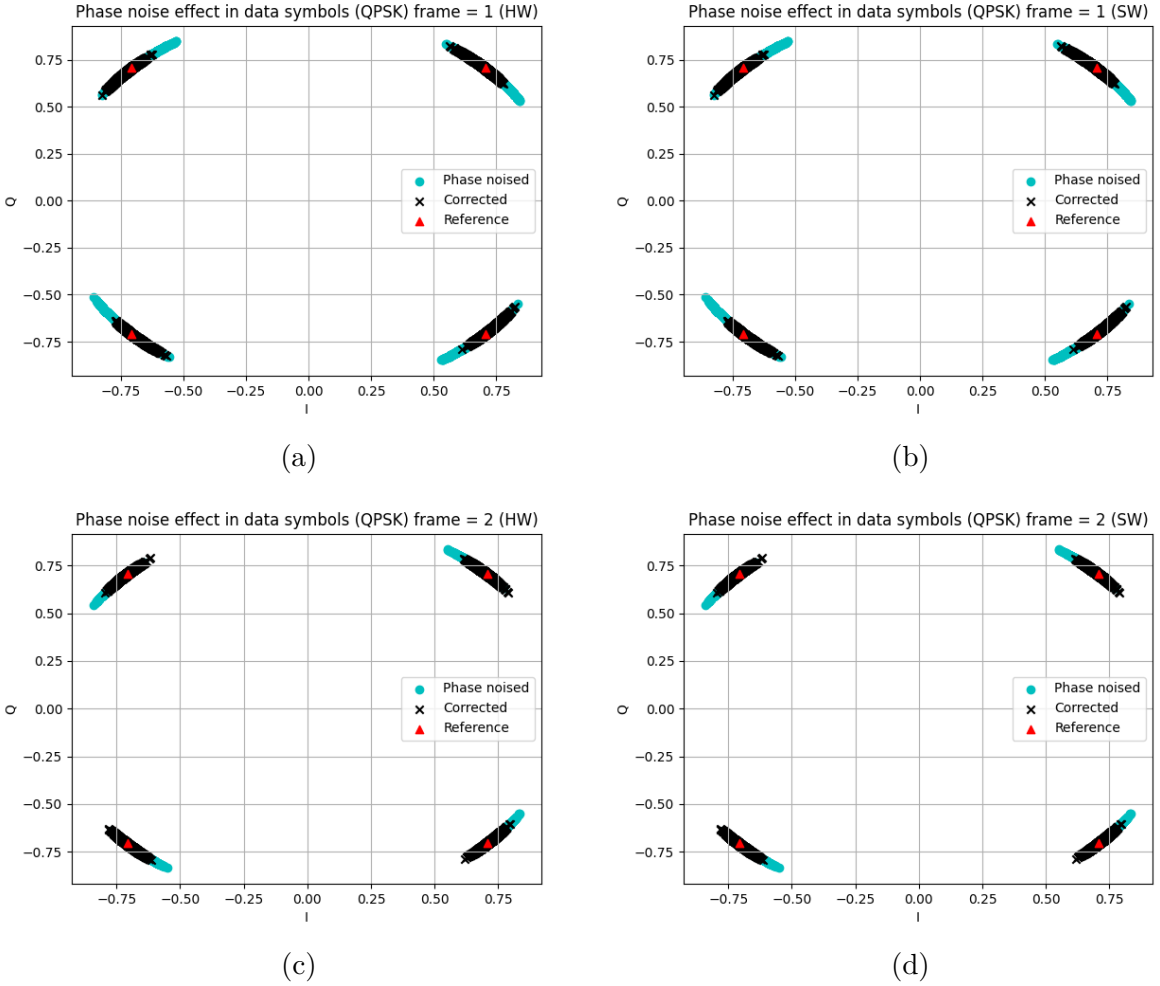
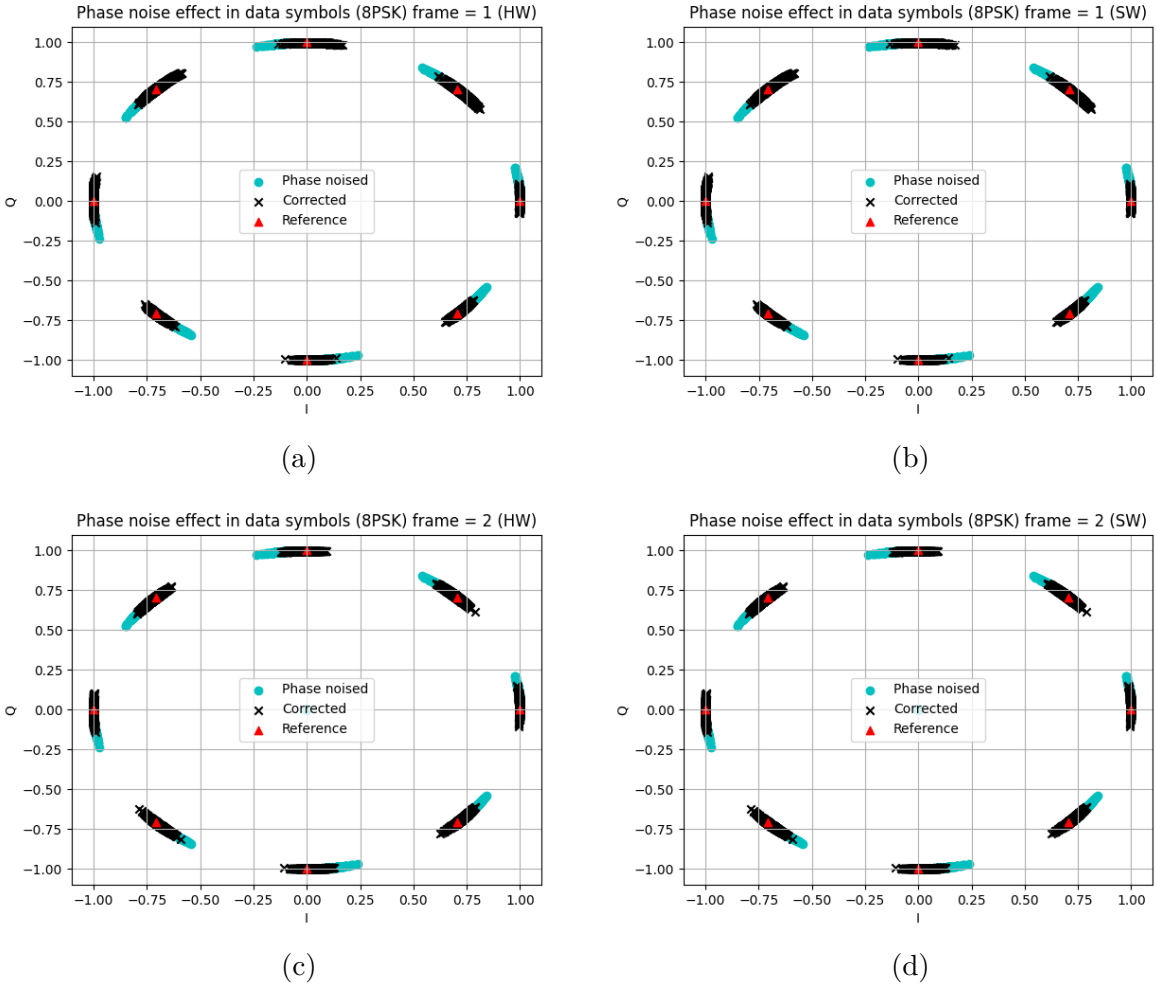


Figura 64 – Constelações Q-PSK corrigidas.

O RMSE calculado foi de $2,475.10^{-4}$, um erro bem pequeno que foi ocasionado pela pequena diferença do cálculo entre a função arco-tangente e o *CORDIC* de alguns ângulos. Mesmo com este erro, as constelações das versões hardware e software são praticamente idênticas.

A seguir, na figura 65, foram obtidas as constelações para a modulação 8-PSK . Novamente, nas figuras à esquerda ,65a e 65c, apresentam-se as constelações do modelo em hardware, enquanto nas figuras à direita, figuras 65b e 65d, apresentam-se as figuras do modelo de referência em software.



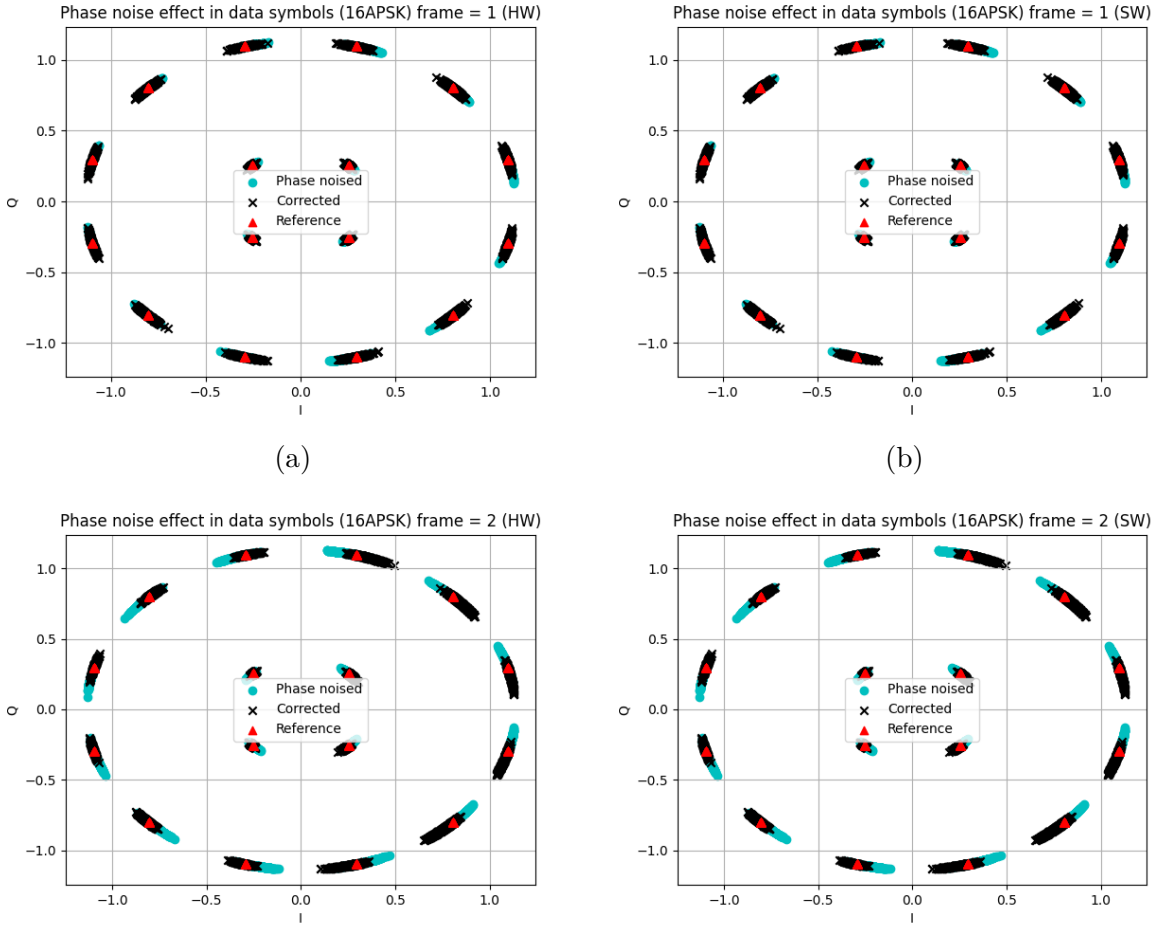
(a) (b)

(c) (d)

Figura 65 – Constelações 8-PSK corrigidas.

O erro RMS calculado para esta modulação foi de $2,837.10^{-4}$, novamente um erro muito pequeno ocasionado pela mesma razão discutida na modulação Q-PSK. E ainda assim, as modulações no modelo em software e hardware estão praticamente idênticas.

Da última modulação simulada também foram extraídas suas constelações, mostradas na figura 66. Nas figuras da esquerda,66a e 66c, assim como nas modulações Q-PSK e 8-PSK, situam-se as constelações do modelo em hardware. Já nas figuras da direita, 66b e 66d, situam-se as constelações do modelo de referência em software.



(a) (b)

(c) (d)

Figura 66 – Constelações 16-APSK corrigidas.

Nessa modulação, o erro RMS calculado foi de $3,545.10^{-4}$, Novamente um erro pequeno, um pouco maior do que nas modulações anteriores e ocasionado pelos mesmos motivos também.

Por fim, obteve-se uma curva de de erro de estimação de fase residual, em graus, para diferentes valores de Es/No, que pode se observar no gráfico da figura 67. Neste gráfico apresentam-se três curvas: a curva do modelo em ponto fixo, em vermelho; a curva do modelo em software, em azul; e a curva de referência que pode ser encontrada no trabalho referenciado pela norma, o trabalho realizado por Casini (CASINI; GAUDENZI; GINESI, 2004). Nota-se que o modelo em hardware, para os menores valores de Es/No, foi o que mais se aproximou desta curva de referência.

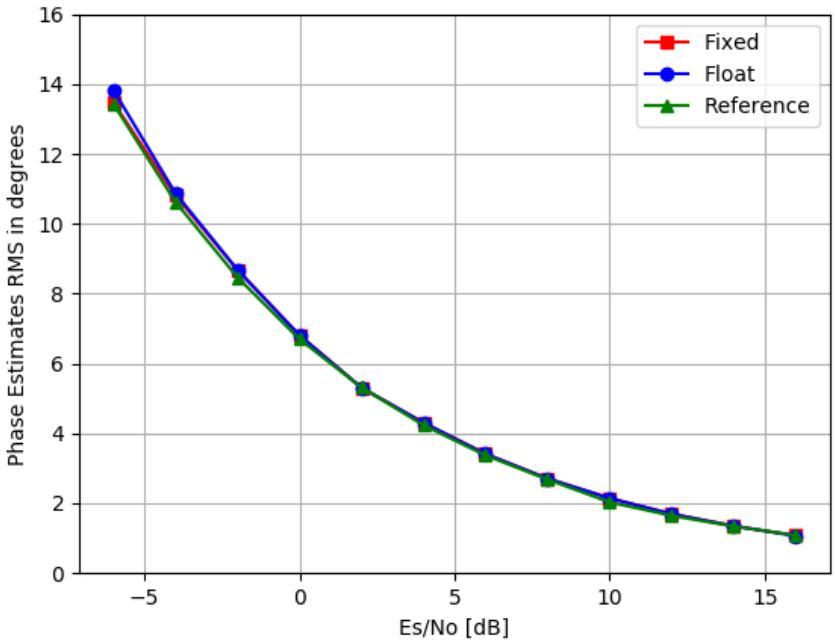


Figura 67 – Curva RMSE de estimação de fase x Es/No.

5 Conclusão

Devido ao árduo trabalho em realizar as operações aritméticas da forma mais otimizada possível, conseguiu-se que o módulo CFC não consumisse muitos recursos da FPGA, assim fazendo com que a arquitetura implementada em hardware, em comparação a trabalhos anteriormente citados neste documento, fosse muito econômica. De fato, ao utilizar uma arquitetura em hardware para realizar essas operações, foi obtida uma maior robustez com os dados trabalhados para a aplicação específica. Os truncamentos e transformações de divisões para multiplicação por constantes, contribuíram bastante para a robustez deste módulo e para a economia de recursos.

O módulo CFC em malha aberta, ainda sem a integração com o *Transceiver*, conseguiu acompanhar as curvas de acúmulos de erro do modelo implementado em software, assim comportando-se como esperado e conseguindo validar seu funcionamento. O módulo implementado em hardware foi capaz de detectar desvios de frequência entre 0,02 e 0,2 em frequências normalizadas da mesma forma como o modelo de referência conseguiu. O fato do RMSE não ter crescido proporcionalmente com o aumento do desvio de frequência na curva da figura 61 se dá ao fato de que, com o aumento do desvio de frequência, o módulo começa a receber mais entradas de valores negativos e, se tratando de um módulo que somente acumula estimativas de correção dentro de uma média móvel, faz com que as estimativas calculadas pelo FED e até mesmo pelo filtro PI sejam menores, consequentemente fazendo com que a média móvel, que é a saída do módulo implementado, não acumulasse valores muito altos. No caso da segunda simulação, mostrada na figura 62, os valores de RMSE são muito próximos, mas de fato tendo um comportamento de decaimento com o aumento do Es/No, o que se era esperado. Porém, em valores não muito altos de Es/No, o RMSE tende a aumentar um pouco, comportamento este explicado por algumas discrepâncias de truncamento entre o simulador em Python e o módulo implementado em hardware. Na figura 63 o RMSE tentou a aumentar por conta do módulo CFC ainda não estar corrigindo as entradas recebidas, mas sim acumulando as estimativas de erros calculadas e, consequentemente, aumentando o RMSE.

O consumo de potência do módulo CFC está dentro do esperado, a julgar que o módulo não possui muitos blocos e algoritmos mais complexos, tendo seu maior consumo de potência atrelado a área mais estática do chip. O espaço ocupado pelo módulo, a se esperar pelo consumo de recursos dele, foi pequeno, tendo a maior parte ocupado pelo bloco ILA, como se é esperado.

O teste em hardware do módulo CFC acabou por validar o seu correto funcionamento, ainda que não foi possível inserir mais de 10000 valores de vetores de testes para

cada memória ROM, por questões de limitações da FPGA escolhida. Mesmo assim, os valores esperados coincidiram com os valores enviados pelo módulo no teste em hardware. Isto mostra que o módulo funcionou tanto na simulação quanto em um teste mais prático, e como não apresentou problemas de *Timing*, não houveram valores diferentes dos esperados entre os valores extraídos.

Com Relação ao módulo PC, o mesmo também foi trabalhado de forma que fossem economizados o maior número de recursos possíveis sem precisar deixar o módulo muito lento. Ainda assim, o módulo consumiu mais recursos do que os trabalhos citados, principalmente pela escolha do método de correção dos símbolos, que foi necessário o uso de muitas LUTs para armazenar os valores de seno. Ainda assim, essa técnica foi muito bem otimizada, tendo em vista que apenas o primeiro quadrante do círculo trigonométrico foi amostrado para se obter os valores de seno.

As constelações extraídas das simulações comportamentais, quando comparadas com suas recíprocas do modelo de referência em Python, comportaram-se praticamente de forma igual, já que o erro RMS entre elas, para as três modulações testadas, foram bem pequenos. Quando se observam as constelações também é possível notar que elas foram corrigidas corretamente, com uma ressalva para a modulação 16-APSK, que precisaria de mais uma etapa de correção de fase, desta vez uma mais fina.

O espaço ocupado pelo módulo, por conta do seu consumo de recursos, foi aproximadamente metade da FPGA. Se contar com o bloco de comparação (*PCComparator*, os dois juntos ocuparam quase todo o espaço da FPGA. De fato a tabela de senos, em troca da rapidez no cálculo para a correção dos símbolos, fez com que fosse ocupado mais espaço na FPGA. Como este módulo também possui algoritmos um pouco mais complexos do que o algoritmo do módulo CFC, o consumo de potência do módulo na parte dinâmica foi superior ao da parte estática, o que se era esperado.

Como se pode observar no teste em hardware, todos os símbolos de dados e de pilotos foram corrigidos corretamente, mostrando que o módulo não só funciona na simulação comportamental, mas também já implementado em FPGA. Novamente, devido ao fato de não ter acontecido nenhum problema de *Timing*, tanto com relação ao tempo de *setup* e ao tempo de *hold*, não houve nenhum resultado diferente do que era esperado, validando finalmente o módulo PC implementado em hardware.

Pode-se observar também que na curva da figura 67 o modelo de referência em ponto fixo foi o que mais se aproximou da curva de referência do trabalho citado pela norma, mostrando que nem sempre por não ter a mesma precisão que a representação em ponto flutuante quer dizer que o modelo vá errar mais. O erro entre a estimação de fase do modelo em hardware e o modelo de referência em ponto fixo foi muito pequeno, não influenciando em praticamente nada no formato da curva.

Em trabalhos futuros, o módulo CFC aqui implementado, terá seu desenvolvimento completado ao ser integrado ao bloco DDS, que está contido no módulo *Transceiver* no receptor do protocolo DVB-S2x, na cadeia de sincronização de símbolos. Nesta etapa de integração, tem-se vários outros módulos cruciais na integração do receptor DVB-S2X, como a sincronização de quadro, sincronização de tempo, FFC, dentre outros. Serão feitos testes de bancada envolvendo estes módulos do receptor DVB-S2X junto com o CFC e o PC, com o auxílio de um simulador de enlace, injetando vetores de testes no sistema integrado, de forma que todos os módulos, incluindo o CFC e o PC, sejam testados sob as mesma condições de contorno. Com essa integração, espera-se que as próximas curvas características deste módulo serão também curvas de rastreio de frequência, e também de RMSE para diferentes cenários de Es/No e desvios de frequências.

Já com relação ao módulo PC, por não precisar de uma realimentação, está praticamente completo. No futuro pretende-se capturar as constelações corrigidas quando o mesmo estiver integrado ao FFC e ao CFC, sendo possível estimar se o conjunto de módulos de sincronização de frequência e fase são capazes de corrigir os desvios de frequência e fase de uma portadora. A seguir, estes três módulos serão integrados a todos os módulos do receptor DVB-S2X, testando-os sob as mesmas condições de contorno e extraindo as constelações corrigidas de cada modulação e traçando a curva de estimação de erro de fase para quando estiver tudo integrado.

Referências

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 15 vezes nas páginas 9, 18, 29, 30, 31, 32, 37, 38, 39, 42, 48, 49, 50, 56 e 85.

de Lima, E. R. 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 6 vezes nas páginas 9, 35, 36, 48, 65 e 68.

DIGILENT. Disponível em: <https://reference.digilentinc.com/reference/programmable-logic/zybo/start>. Citado 2 vezes nas páginas 10 e 62.

EB. *EB's Blog*. 2015. Disponível em: <https://eb.dy.fi/2015/11/xilinx-artix-arty/>. Citado 2 vezes nas páginas 9 e 33.

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 9 vezes nas páginas 7, 8, 9, 25, 27, 50, 51, 53 e 56.

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 na página 18.

Fitz, M. P. Planar filtered techniques for burst mode carrier synchronization. In: *IEEE Global Telecommunications Conference GLOBECOM '91: Countdown to the New Millennium. Conference Record*. [S.l.: s.n.], 1991. p. 365–369 vol.1. Citado 2 vezes nas páginas 41 e 42.

Gardner, F. M. Interpolation in digital modems. i. fundamentals. *IEEE Transactions on Communications*, v. 41, n. 3, p. 501–507, March 1993. Citado na página 29.

HAMZEHYAN, R.; DIANAT, R.; SHIRAZI, N. New variable step-size blind equalization based on modified constant modulus algorithm. *International Journal of Machine Learning and Computing*, p. 30–34, 01 2012. Citado 2 vezes nas páginas 9 e 31.

Jong Gyu Oh; Joon Tae Kim. An alternative carrier frequency synchronization scheme for DVB-S2 systems. In: *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2010. v. 1, p. 529–533. Citado 5 vezes nas páginas 9, 10, 41, 42 e 48.

Kay, S. A fast and accurate single frequency estimator. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 37, n. 12, p. 1987–1990, Dec 1989. Citado na página 35.

LIMA, E. D. et al. Design and FPGA prototyping of a DVB-S2 receiver: Towards a VLSI implementation in CMOS. In: . [S.l.: s.n.], 2013. Citado 8 vezes nas páginas 9, 26, 28, 36, 37, 48, 65 e 68.

Luise, M.; Reggiannini, R. Carrier frequency recovery in all-digital modems for burst-mode transmissions. *IEEE Transactions on Communications*, v. 43, n. 2/3/4, p. 1169–1178, Feb 1995. Citado na página 42.

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 3 vezes nas páginas 42, 45 e 46.

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, Jan 2006. Citado 4 vezes nas páginas 7, 8, 18 e 25.

Muñoz, D. M. et al. FPGA based floating-point library for cordic algorithms. In: *2010 VI Southern Programmable Logic Conference (SPL)*. [S.l.: s.n.], 2010. p. 55–60. Citado na página 32.

Park, J. W. et al. Efficient coarse frequency synchronizer using serial correlator for DVB-S2. In: *2008 IEEE International Symposium on Circuits and Systems*. [S.l.: s.n.], 2008. p. 1520–1523. Citado 3 vezes nas páginas 10, 43 e 48.

Park, J. W. et al. An Efficient Data-Aided Initial Frequency Synchronizer for DVB-S2. In: *2007 IEEE Workshop on Signal Processing Systems*. [S.l.: s.n.], 2007. p. 645–650. Citado 4 vezes nas páginas 10, 42, 43 e 48.

Qi Luo; Xiaojun Cheng; Zucheng Zhou. FPGA design and implementation of carrier synchronization for DVB-S2 demodulators. In: *2007 7th International Conference on ASIC*. [S.l.: s.n.], 2007. p. 846–849. Citado 4 vezes nas páginas 9, 39, 40 e 48.

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 na página 53.

Savvopoulos, P.; Papandreou, N.; Antonakopoulos, T. A software-radio test-bed for measuring the performance of DVB-S2 receiver circuits. In: *2008 10th International Workshop on Signal Processing for Space Communications*. [S.l.: s.n.], 2008. p. 1–7. Citado 5 vezes nas páginas 10, 44, 45, 47 e 48.

Savvopoulos, P.; Papandreou, N.; Antonakopoulos, T. Architecture and DSP implementation of a DVB-S2 baseband demodulator. In: *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*. [S.l.: s.n.], 2009. p. 441–448. Citado 5 vezes nas páginas 10, 45, 46, 47 e 48.

Shannon, C. E. Communication in the presence of noise. *Proceedings of the IRE*, v. 37, n. 1, p. 10–21, 1949. Citado na página 18.

VENTZAS, D.; PETRELLIS, N. Peak Searching Algorithms and Applications. *Proceedings of the IASTED International Conference on Signal and Image Processing and Applications, SIPA 2011*, 08 2011. Citado na página 29.

XILINX. Disponível em: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. Citado 2 vezes nas páginas 9 e 34.