Diego Orlando Barragán Guerrero

IMPLEMENTAÇÃO EM FPGA DE ALGORITMOS DE SINCRONISMO PARA OFDM

Campinas
2013

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Diego Orlando Barragán Guerrero

IMPLEMENTAÇÃO EM FPGA DE ALGORITMOS DE SINCRONISMO PARA OFDM

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Engenharia Elétrica, na área de concentração: Telecomunicações e Telemática.

Orientador: Prof. Dr. Luís Geraldo Pedroso Meloni

Este exemplar corresponde à versão final da tese defendida pelo aluno Diego Orlando Barragán Guerrero, e orientada pelo Prof. Dr. Luís Geraldo Pedroso Meloni

_____

Campinas
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

B271i

Barragán Guerrero, Diego Orlando, 1984-
Implementação em FPGA de algoritmos de sincronismo para OFDM / Diego Orlando Barragán Guerrero. – Campinas, SP : [s.n.], 2013.

Orientador: Luís Geraldo Pedroso Meloni.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. FPGA. 2. OFDM. 3. VHDL (Linguagem descritiva de hardware). 4. Processamento digital de sinais. 5. Sincronização. I. Meloni, Luís Geraldo Pedroso,1958-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: FPGA implementation of synchronization algorithms for OFDM
Palavras-chave em inglês:
FPGA
OFDM
VHDL (Hardware descriptive language)
Digital signal processing
Synchronization
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Luís Geraldo Pedroso Meloni [Orientador]
Omar Carvalho Branquinho
Gustavo Fraidenraich
Data de defesa: 28-08-2013
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Diego Orlando Barragán Guerrero

Data da Defesa: 28 de agosto de 2013

Título da Tese: "Implementação em FPGA de Algoritmos de Sincronismo para OFDM"

Prof. Dr. Luís Geraldo Pedroso Meloni (Presidente): _____

Prof. Dr. Omar Carvalho Branquinho: _____

Prof. Dr. Gustavo Fraidenraich: _____

Resumo

Os sistemas OFDM são intrinsecamente sensíveis a erros de sincronismo de tempo e frequência. O sincronismo é uma etapa fundamental para a correta recepção de pacotes. Esta dissertação descreve como se implementar vários algoritmos de sincronismo para OFDM em FPGA usando os símbolos do preâmbulo definidos no padrão IEEE 802.11a. Além disso, foi implementado o algoritmo CORDIC (necessário para a etapa de estimação e compensação de desvio de portadora) em modo rotacional e vetorial para um sistema coordenado circular, comparando o desempenho de várias arquiteturas com o intuito de otimizar a frequência de operação e relacionar o erro do resultado com o número de iterações realizadas. Conforme mostrado nos resultados, são obtidas estimativas com boas aproximações para desvios de 0, 100 e 200 kHz. Os resultados obtidos constituem um instrumento importante para a melhor escolha de implementação de algoritmos de sincronismo em FPGA. Verificou-se que os diferentes algoritmos não apenas possuem valores de variância distintos, mas também frequências de operação diferentes e consumo de recursos da FPGA. Ao longo do projeto foi considerado um modelo de canal *tapped-delay*.

Palavras-chave: OFDM, sincronismo de tempo, detecção do pacote, CFO, VHDL, FPGA.

Abstract

OFDM systems are intrinsically sensitive to errors of synchronization in time and frequency. Synchronization is a key step for correct packet reception. This thesis describes how to implement in FPGA several synchronization algorithms for OFDM using the symbols of the preamble defined in IEEE 802.11a. In addition, the CORDIC algorithm is implemented (step required for carrier frequency offset estimation and compensation) in rotational and vectoring mode for a circular coordinate system, comparing the performance of various architectures in order to optimize the operating frequency and relate the error of the result with the number of iterations performed. As shown in the results, estimates are obtained with good approximations for offsets of 0, 100 and 200 kHz. The obtained results are an important instrument for the best choice of synchronization algorithm for implementation in FPGA. It was found that the different algorithms have not only different values of variance, but also different operating frequency and consumption of the FPGA resources. Throughout the project a tapped-delay channel model was considered in the analysis.

Key-words: OFDM, time synchronization, packet detection, CFO, VHDL, FPGA.

Sumário

A minha mãe Mery Beatriz e ao meu pai José Vicente (in memorian).

Agradecimentos

Ao professor Dr. Luís Meloni, pela oportunidade de trabalhar com a sua orientação e pelos conhecimentos transmitidos durante o mestrado e no desenvolvimento do presente trabalho.

À minha família, pelo apoio incondicional durante esta jornada.

À minha esposa Vanesa, por me apoiar sempre e ser uma companheira indispensável.

Aos meus amigos e demais colegas da FEEC, em especial aos meus colegas do laboratório.

Aos membros da banca examinadora, pelas sugestões para a melhoria do presente trabalho.

À FEEC/UNICAMP a ótima estrutura que oferece aos estudantes e pesquisadores.

À CNPq e SENESCYT pelo apoio financeiro.

"You are right", Nietzsche replied. "Our writing equipment takes part in the forming of our thoughts."

Nicholas Carr, *The Shallows*

Lista de figuras

Lista de tabelas

Glossário

ADC	Analogous Digital Converter
AGC	Automatic Gain Control
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
CFO	Carrier Frequency Offset
CLB	Configurable Logic Blocks
CORDIC	COordinate Rotation DIgital Computer
CSA	Carry Select Adder
DCM	Digital Clock Manager
DDS	Direct Digital Synthesizer
ETSI	European Telecommunications Standards Institute
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
ICI	Inter Carrier Interference
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
IG	Guard Interval
IOB	Input/Output Block
ISI	Inter Symbol Interference
LC	Logic Cell
LTS	Long Training Sequence
LUT	Look up Table
MAC	Multiply-accumulate
MATLAB	MATrix LABoratory
MIMO	Multiple-input multiple-output
ML	Maximum Likelihood
OFDM	Orthogonal frequency-division multiplexing
PLL	Phase-locked loop
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RCA	Ripple Carry Adder
RTL	Register Transfer Level
SNR	Signal to Noise Ratio
SRAM	Static Random Access Memory
SRT	Shift Register LUT

STS	Short Training Sequence
VHDL	VHSIC Hardware Description Language
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network

Lista de símbolos

r_n	Amostra do símbolo recebido
r_n^*	complexo conjugado da amostra r_n
$R(d)$	Autocorrelação do sinal
$P(d)$	Potência do sinal
$M(d)$	Métrica de comaparação
T_a	Período de amostragem
W	Largura de banda
N_{FFT}	Número de pontos da FFT
N_{dados}	Subportadoras de dados
$N_{pilotos}$	Subportadoras de pilotos
Δ_F	Distância entre subportadoras
N	Número de amostras do símbolo
$N_{\max}$	Amostras do prefixo cíclico com amostras do símbolo anterior
$x(t)$	Símbolo transmitido na k-ésima subportadora
x_n	Símbolo digital transmitido na k-ésima subportadora
$\mathbb{Z}$	Número inteiro
$A(a,b)$	Racional em ponto fixo.
f_k	Frequência da k-ésima subportadora
X_k	Amplitude complexo do sinal na k-ésima subportadora
T_s	Duração temporal do símbolo
θ	Deslocamento temporal do pacote
t_i	Tempo aleatório de inicio do pacote
$\tau_{\max}$	Espalhamento temporal máximo do canal
$\Delta\theta$	Diferença temporal entre o valor estimado e o valor verdadeiro
r_{short}	Símbolos STS
r_{long}	Símbolos LTS
w_{short}	Função de janelamento
R_{dif}	Diferença de autocorrelações
$\Lambda\,(d)$	Correlação cruzada
ϕ	Desvio de fase
ε	Desvio de frequência normalizado
Δf	Desvio de frequência da portadora

Trabalhos publicados pelo autor

1. Diego Barragán, Karlo Lenzi e Luís Meloni. “Desempenho do algoritmo paralelo CORDIC em implementação em FPGA”. XXX Simpósio Brasileiro De Telecomunicações, Brasília, Brasil, 2012

2. Diego Barragán e Luís Meloni. “Comparison of parallel and pipelined CORDIC algorithm using RCA and CSA”. IWT International Workshop on Telecommunications, Santa Rita do Sapucai, Brasil, 2013.

3. Diego Barragán e Luís Meloni. “Implementação em FPGA de algoritmos de sincronismo temporal para OFDM”. XXXI Simpósio Brasileiro De Telecomunicações, Fortaleza, Brasil, 2013.

Capítulo 1

Introdução

A técnica de Multiplexação por Divisão de Frequência Ortogonal (*Orthogonal Frequency Division Multiplexing* - OFDM) é caracterizada por dividir a banda do canal em várias sub-bandas ortogonais entre si. Esta técnica tem se mostrado muito robusta, tanto em canais AWGN como em canais com múltiplos percursos sendo adotada em vários sistemas de comunicação, como, por exemplo: no padrão para Difusão de Áudio Digital (*Digital Audio Broadcasting* - DAB) em toda a Europa, nos padrões de Transmissão de Vídeo Digital Terrestre Europeu (DVB-T), e japonês (ISDB-T); em algumas redes locais sem fio (Hiperlan Tipo/2, IEEE 802.11a/g, IEEE 802.16, etc), em canais telefônicos tais como ADSL (*Assymetric Digital Subscriber Line*), sob o nome de DMT (*Discrete MultiTone*), e suas derivações.

No entanto, a técnica OFDM é muito sensível aos erros de sincronismo, o que não acontece com sistemas de portadora única. Assim, o receptor precisa efetuar diversas tarefas de sincronismo. A primeira tarefa é o sincronismo temporal, que consiste em determinar o tempo ótimo no qual a leitura dos símbolos deve ser realizada para minimizar os efeitos das interferências entre portadoras (ICI) e entre símbolos (ISI). A segunda tarefa é o sincronismo em frequência que consiste em alinhar ao máximo a frequência das portadoras do receptor e do transmissor para evitar a ICI. Isto deve ser feito com muita precisão, pois a recepção correta do sinal depende da ortogonalidade das subportadoras, o que pode ser severamente afetado se esse sincronismo não for preciso.

Em síntese, o sincronismo é uma etapa fundamental para a correta recepção de pacotes, sendo assim necessário avaliar o desempenho e factibilidade de implementação dos algoritmos de sincronismo para OFDM. Este trabalho apresenta arquiteturas para implementação em FPGA (*Field-programmable gate array*) de vários algoritmos de sincronismo que usam os símbolos do preâmbulo do pacote. Isto é, este projeto não apenas se propõe implementar os principais algoritmos de sincronismo existentes para sistemas OFDM, considerando diferentes condições de recepção, mas também avaliar e comparar os recursos computacionais de que cada um deles necessita.

1.1 Motivação

O objetivo deste trabalho é implementar em hardware digital vários algoritmos de sincronismo de tempo para um sistema OFDM e comparar seus desempenhos, tanto nos recursos necessários e na frequência de operação quanto na variância dos resultados. Da mesma forma, busca-se implementar um circuito de sincronismo de frequência baseado no algoritmo CORDIC

(*COordinate Rotation DIgital Computer*) otimizado. Ao longo do estudo, os parâmetros do padrão IEEE 802.11a são considerados no projeto.

Este trabalho oferece uma investigação completa sobre os aspectos de mapeamento dos algoritmos de sincronismo em blocos de circuitos projetados para uma FPGA. Embora já existam vários trabalhos na literatura que examinam os algoritmos de sincronismo para receptores OFDM [1][2][3], ainda não há muita documentação nos campos de implementação, arquitetura e otimização de recursos e frequência de operação em uma FPGA.

Espera-se que esta pesquisa possa ser usada como ponto de partida para futuras investigações em implementação de algoritmos de sincronismo.

1.2 Organização da dissertação

O segundo capítulo deste trabalho começa por examinar as principais características e tópicos de uma FPGA. Também inclui uma introdução aos sistemas OFDM e uma caracterização dos problemas de sincronismo, com uma seção que explica a geração das amostras do preâmbulo. Inclui também a descrição matemática dos algoritmos de sincronismo, em conjunto com uma explicação da representação em ponto fixo em FPGA.

O terceiro capítulo trata do ambiente de simulação, síntese e hardware utilizado, incluindo o modelo de canal e a quantização das sequências de entrada ao circuito.

O quarto capítulo apresenta uma descrição completa do processo de implementação dos algoritmos de sincronismo e os resultados derivados da simulação, tais como frequência de operação, número de *slices*, LUTs (*Look up Tables*), variância de detecção e diagrama temporal.

Ao final, este trabalho é concluído com uma análise dos resultados da experiência e recomendações para futuras pesquisas.

Capítulo 2

Conceitos básicos

Nesta seção, é apresentada uma introdução da arquitetura da FPGA, assim como a representação em ponto fixo dos sinais usados na simulação e nos conceitos da matemática de precisão finita. Da mesma forma, a seção aborda uma introdução a sistemas OFDM, problemas de sincronismo, parâmetros do padrão 802.11a e formação do preâmbulo. São apresentados os diferentes algoritmos para detecção do pacote, sincronismo de tempo e sincronismo de frequência, baseados na autocorrelação e na correlação cruzada. A seção termina com a descrição matemática e as configurações dos parâmetros do algoritmo CORDIC, necessárias para a estimação e compensação do desvio da frequência da portadora.

2.1 Introdução a FPGA

Antes do advento da lógica programável, circuitos lógicos eram construídos em placas de circuitos utilizando componentes discretos ou por meio da integração de portas lógicas em circuitos integrados para aplicações específicas.

FPGA é um circuito integrado que contém um grande número (na ordem de milhares) de células lógicas idênticas. Essas células lógicas podem ser vistas como componentes-padrões que podem ser configurados independentemente e interconectados a partir de uma matriz de trilhas condutoras e *switches* programáveis, conforme mostrado na Figura 2.1. Um arquivo binário com extensão *.bit* é gerado para a configuração da FPGA a partir de ferramentas de software, seguindo um determinado fluxo de projeto. Esse arquivo binário contém as informações necessárias para especificar a função de cada unidade lógica e para seletivamente fechar os *swiches* da matriz de interconexão. Em suma, o *array* de unidades lógicas e a matriz de interconexão, que podem ser programados pelo usuário, formam a estrutura básica da FPGA para especificação de circuitos integrados complexos.

Na família das FPGA da Xilinx, a menor unidade lógica configurável é denominada *slice* de lógica e é mostrada na Figura 2.2. O *slice* é bastante versátil e pode ser configurado para operar como *Look-Up Table* (LUT), RAMs distribuídas e registradores de deslocamento. Na operação como LUT, recursos adicionais como *flip flops* tipo D, multiplexadores, lógica de transporte (*carry*) dedicada e portas lógicas, podem ser utilizados em conjunto com as LUTs para implementar funções booleanas, multiplicadores e somadores com palavras binárias de comprimento variável. Na operação como SRL (*Shift Register LUT*), esses recursos adicionais podem ser utilizados na implementação de contadores, conversores serial-paralelo e paralelo-serial, entre outras funcionalidades. Os recursos adicionais mencionados podem ainda ser utilizados na intercone-

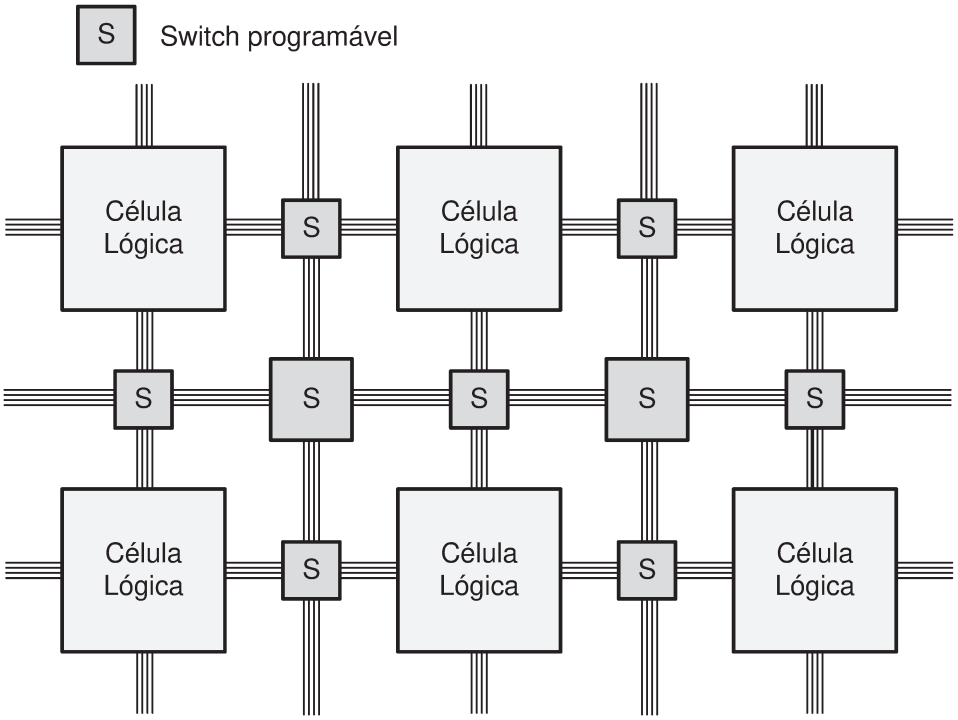


Figura 2.1: Estrutura conceitual de um dispositivo FPGA.

xão de unidades lógicas para a implementação, por exemplo, de multiplicadores, contadores, somadores e memórias com qualquer comprimento de palavra binária. O limite para esse comprimento é determinado pela quantidade de unidades lógicas disponíveis na FPGA.

Além dos recursos-padrões (*slices*), uma FPGA pode disponibilizar no arranjo bidimensional recursos bastante sofisticados, tais como multiplicadores dedicados, MACs programáveis (multiplicador e acumulador, também denominados de *XtremeDSP slice*), blocos de memória, DCM (*Digital Clock Manager* utilizados para multiplicar ou dividir a frequência de um sinal de *clock*), microcontroladores (Microblaze ou PowerPC) e transceptores multi-giga bit (que servem para implementar interfaces seriais para transferência de bits em alta velocidade). A utilização de tais recursos embarcados possibilita otimizar o consumo de área (*slices*) e também desenvolver projetos mais eficientes.

O termo Programável em Campo, por sua vez, significa que as funções da FPGA são definidas por um programa do usuário, em vez de serem definidas pelo fabricante do dispositivo. Em circuitos integrados típicos (ASIC - *Application Specific Integrated Circuit*), a implementação é realizada na fase do projeto. Nas FPGAs, dependendo do CI, o dispositivo pode ser programado permanentemente, semi-permanentemente como parte do processo de montagem da placa, ou carregado a partir de uma memória *flash* a cada vez que o dispositivo é ligado. No último caso, a tecnologia utilizada para implementação da FPGA é a de memória estática (SRAM). Por este motivo, toda vez que o dispositivo é desligado, perde-se a programação. Essa flexibilidade de programação, associada a potentes ferramentas de desenvolvimento e modelagem, possibilita ao usuário acesso a projetos de circuitos integrados complexos sem os altos custos de engenharia associados aos ASICs [4].

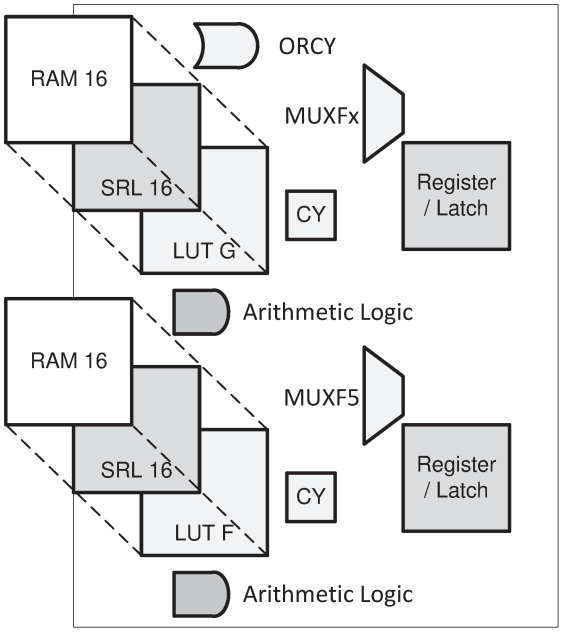


Figura 2.2: *Slice*: unidade lógica básica.

2.2 Visão geral da FPGA Spartan-3 da Xilinx

Este trabalho utiliza dispositivos FPGA Spartan-3 da Xilinx. Com base na relação entre o número de células de lógica e as contagens de blocos de entrada e saída, a família está dividida em várias subfamílias [5]. Apresenta-se a continuação os recursos computacionais da Spartan-3.

2.2.1 Célula lógica, slice e CLB

O elemento mais básico do dispositivo Spartan-3 é uma célula lógica (LC), a qual contém uma LUT de quatro entradas e um *flip-flop* de tipo D. Além disso, uma célula lógica contém um circuito de *carry* usado para implementar funções aritméticas, e um circuito de multiplexagem, o qual é usado para implementar multiplexadores de maior escala. A LUT também pode ser configurada como uma memória estática de acesso aleatório (SRAM) de 16-por-1 ou um registrador de deslocamento de 16 bits [5].

Para aumentar a flexibilidade e melhorar o desempenho, oito células lógicas são combinadas em conjunto com uma estrutura de encaminhamento interno especial. Na terminologia da Xilinx, duas células lógicas são agrupadas para formar um *slice*, e quatro *slices* são agrupados para formar um bloco lógico configurável (CLB).

2.2.2 Célula macro

O dispositivo Spartan-3 contém quatro tipos de blocos de macro: multiplicador combinacional, bloco RAM, *Digital Clock Manager* (DCM) e bloco de entrada/saída (IOB). O multiplicador combinacional aceita dois números de 18 bits como entradas e calcula o produto, podendo ser configurado para realizar uma multiplicação assíncrona ou síncrona. A memória RAM é um bloco síncrono SRAM de 18 kbits, que pode ser disposto em vários tipos de configurações como,

por exemplo, memórias RAM, ROM, FIFOs, LUTs, conversores de largura de dados, *buffers* circulares e registros de deslocamento, cada um suportando várias larguras e comprimentos de dados. O DCM é um bloco para manipular sinais de relógio permitindo multiplicar ou dividir a frequência de referência de entrada, recondicionar um relógio para garantir 50% do ciclo de trabalho, eliminar inclinação do relógio e deslocar a fase de um sinal de relógio, seja por uma fração fixa de um período de relógio ou por incrementos precisos. Um IOB fornece uma interface programável e bidirecional para controlar o fluxo de dados entre os pinos de entrada/saída do dispositivo e a lógica interna. Ele pode ser configurado para suportar uma ampla variedade de entradas/saídas de padrões de sinalização.

2.3 Representação em ponto fixo

No momento de projetar um sistema de comunicações sem fio, existem limitações de área, potência e desempenho que devem ser atendidas. Isto é especialmente verdadeiro para dispositivos portáteis, tais como laptops, para os quais capacidade e a duração da bateria são considerações importantes. Resolver em hardware digital operações em ponto-flutuante requer maior área e consumo de potência do que operações em ponto fixo. Portanto, existe a necessidade de mapear os algoritmos de sincronismo que empregam operações em ponto flutuante para operações em ponto fixo. Na aritmética de ponto fixo, a posição decimal na representação em bits do número é conhecida em cada etapa do cálculo, economizando recursos destinados para registro e decodificação quando se trabalha com uma representação de ponto flutuante. A intenção de usar notação em ponto fixo é minimizar os recursos de hardware e potência.

O VHDL, em sua representação básica, lida com sinais em ponto fixo com ou sem sinal. Na representação com sinal, é utilizada a notação complemento de 2. Nessa notação, o número negativo é obtido pelo oposto dos bits do número positivo e, em seguida, somando-se 1. A soma de um número positivo pelo seu complemento de 2 vai dar sempre zero se for descartado o *carry* que excede o comprimento em bits da palavra. Por exemplo, +3 + (-3)= 0011 + 1101 = 10000. Também os limites +8 e -8 têm a mesma representação, resultando que apenas um deles deve ser mantido na faixa de valores possíveis. Neste caso, o número 1000 é incluído como -8 porque o bit mais significativo é 1. Desta forma, o bit mais significativo pode ser utilizado para indicar o sinal do valor: 0 para positivo e 1 para negativo.

Uma palavra binária de M bits, quando é interpretada como um racional de ponto fixo em complemento de 2, pode assumir valores em um conjunto P de racionais dado por

$$P = \left\{ p/\,2^b \,|\, -2^{M-1} \leq p \leq 2^{M-1} - 1, p \in \mathbb{Z} \right\}. \tag{2.1}$$

Note-se que P contém 2^M elementos. Denotamos tal representação como $A(a, b)$, onde $a = M - b - 1$, sendo a os bits da parte inteira e b os bits da parte fracionária.

O valor de um número x de M bits na representação $A(a, b)$ é dado por

$$x = \left(\frac{1}{2^b}\right) \left[-2^{M-1}\, x_{M-1} + \sum_{0}^{M-2} 2^n\, x_n \right], \tag{2.2}$$

onde x_n representa o bit n de x. O comprimento da representação $A(a, b)$ é dado por

$$-2^{M-1-b} \leq x \leq +2^{M-1-b} - 1/\,2^b. \tag{2.3}$$

2.3.1 Conceitos de matemática de precisão finita

Precisão

A precisão é o número máximo de bits não nulos representáveis. Por exemplo, $A(13,2)$ possui uma representação de 16 bits. Para representações em ponto fixo, a precisão é igual ao comprimento da palavra binária.

Resolução

A resolução é a menor magnitude não nula representável. Por exemplo, $A(13,2)$ possui uma resolução de $1/2^2 = 0.25$

Faixa

Faixa é a diferença entre o número mais negativo representável e o número mais positivo representável:

$$X_F = X_{Max+} - X_{Max-}\,. \tag{2.4}$$

Por exemplo, o número $A(13,2)$ possui uma faixa de -8192 a +8191.75, ou seja 16383.75.

Acurácia

A acurácia é a magnitude da diferença máxima entre um valor real e a sua representação. Por exemplo, a acurácia para um número $A(13,2)$ é 1/8. Note-se que a acurácia e a resolução são relacionadas do seguinte modo:

$$A\left(x\right) = R\left(x\right)/2, \tag{2.5}$$

onde $A(x)$ é a acurácia de x e $R(x)$ é a resolução de x.

Faixa dinâmica

A faixa dinâmica é a razão do valor absoluto máximo representável para o valor absoluto mínimo positivo (i.e. não nulo) representável. Para um racional com signo em ponto fixo $A(a,b)$, a faixa dinâmica é representada por

$$2^a \,/\, 2^{-b} = 2^{a+b} = 2^{M-1}\,. \tag{2.6}$$

2.4 OFDM

A técnica OFDM é um esquema de modulação multiportadora. O transmissor de multiportadora é composto por um conjunto de moduladores, cada um com frequência diferente de portadora. O transmissor combina as saídas dos moduladores e gera o sinal a ser transmitido.

Suponha-se que N dados a ser transmitidos são X_i, com $i = 0, 1, ..., N-1$, onde X_i representa um número complexo em uma dada constelação, tal como QPSK ou QAM. A i-ésima frequência portadora para X_i é f_i. Assim, a saída do transmissor de multiportadora de valor complexo é dada por

$$x\left(t\right)=\sum_{i=0}^{N-1}X_i\,e^{j2\pi\,f_i\,t}. \tag{2.7}$$

Um transmissor digital irá gerar na saída dados discretos em instantes $t = kT_s$, onde T_s é o intervalo de amostragem e k inteiro. Assim, a saída do transmissor digital de multiportadoras é dada por

$$x\left(k\,T_s\right)=\sum_{i=0}^{N-1}X_i\,e^{j2\pi\,f_i\,k\,T_s}. \tag{2.8}$$

Além disso, se as frequências portadoras foram uniformemente espaçadas no domínio da frequência por um espaçamento de frequência de f_s (i.e. $f_i = i\,f_s$, $i = 0, 1, ..., N-1$), tem-se

$$x\left(k\,T_s\right)=\sum_{i=0}^{N-1}X_i\,e^{j2\pi\,i f_s\,k\,T_s}. \tag{2.9}$$

Com uma separação mínima de $f_s = 1/\left(N\,T_s\right)$ para manter a ortogonalidade entre os sinais nos diferentes moduladores, o sinal OFDM é dado por

$$x\left(k\,T_s\right)=\sum_{i=0}^{N-1}X_i\,e^{\frac{j2\pi ki}{N}}. \tag{2.10}$$

O espectro das subportadoras OFDM de $x(kT_s)$ é mostrado na Figura 2.3.

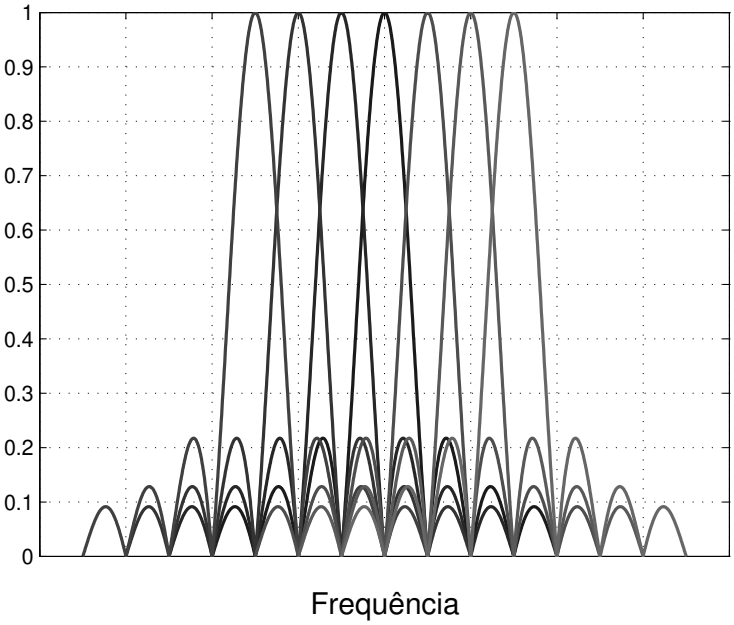


Figura 2.3: Multiplexação na frequência usando portadoras ortogonais.

Assim, um sinal OFDM $\vec{x}_q = \left[x_q\left(0\right), x_q\left(1\right), ..., x_q\left(N-1\right)\right]$ com N amostras, é sintetizado pela equação

$$x_q\left(k\,T_s\right) = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} X_{i,q}\, e^{j2\pi \frac{ik}{N}}, \tag{2.11}$$

onde $X_{i,q}$ são os símbolos transmitidos na i-ésima portadora do q-ésimo símbolo OFDM. Utiliza-se o fator $1/\sqrt{N}$ para que sejam mantidas as condições do Teorema de Parseval (energia do sinal no domínio do tempo igual à energia no domínio da frequência).

A equação (2.11) é a Transformada Inversa de Fourier Discreta (IDFT) de N pontos. Se N é uma potência de dois, então existem vários algoritmos e arquiteturas para implementar a operação da IDFT, tornando a tecnologia OFDM uma solução viável para os sistemas de comunicação modernos.

Com o intuido de reduzir a ICI e a ISI e que a equalização de um subcanal OFDM se reduza a um filtro de apenas um ganho, OFDM implementa um intervalo de guarda (IG) entre cada símbolo que é constituído por uma extensão cíclica do símbolo, denominada prefixo cíclico [6]. Para gerar o prefixo cíclico, um trecho final do símbolo de comprimento N_g amostras, é repetido em sua parte inicial, conforme mostrado na Figura 2.4.

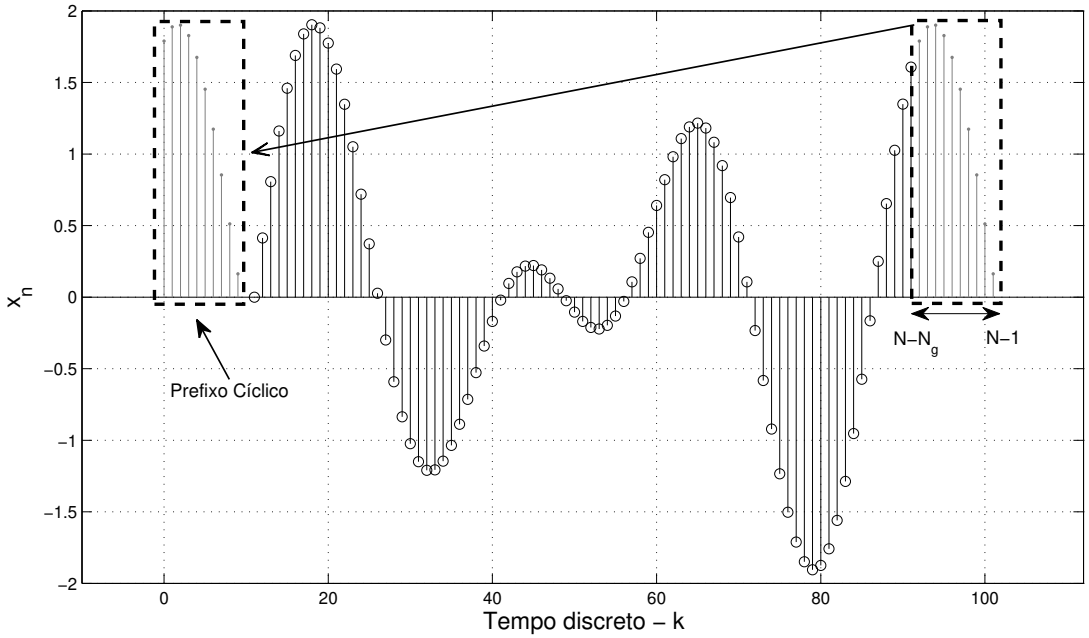


Figura 2.4: Prefixo cíclico.

2.5 Problemas de sincronismo em OFDM

Um dos requisitos em um sistema OFDM é a necessidade de etapas de sincronismo antes da demodulação das subportadoras. Um receptor OFDM primeiro detecta a presença do pacote, encontra os limites do símbolo e determina o instante ótimo de sincronismo para minimizar os efeitos da interferência intersimbólica (ISI - *Inter-Symbol Interference*) e da interferência entre subportadoras (ICI - *Intercarrier Interference*). Além disso, o receptor passa a estimar e

compensar o desvio de frequência da portadora provocado por descasamentos entre os osciladores no transmissor e no receptor, desvios Doppler ou ruído de fase introduzido por osciladores.

Existem dois efeitos destrutivos causados por um desvio na frequência da portadora em sistemas OFDM. Um deles é a redução da amplitude do sinal, já que as funções *sinc* que compõem o espectro estarão deslocadas, fazendo com que a amostragem não ocorra mais no valor máximo da função. Outro efeito é a introdução de ICI das subportadoras adjacentes, devido à perda da ortogonalidade. A fim de suprimir o ICI e, portanto, reduzir a degradação da relação sinal-ruído (SNR), o CFO (*Carrier Frequency Offset*) residual deve ser suficientemente pequeno [7].

A ISI se deve à utilização de amostras do símbolo anterior ou posterior na DFT para obtenção do sinal no domínio da frequência. A ICI é decorrente do desalinhamento entre a janela da DFT e o sinal do símbolo recebido.

2.5.1 Erro de temporização

Considerando-se que o pacote possui um tempo de início aleatório t_i em relação à referência $t = 0$, o deslocamento temporal do pacote em amostras é dado por

$$\theta = \frac{t_i}{T_a}, \tag{2.12}$$

onde T_a é o período de amostragem (50 ns). Desta forma, o objetivo do sincronismo de tempo é identificar θ. Sendo $\hat{\theta}$ a estimativa do ponto do sincronismo, então a diferença entre o valor estimado e o verdadeiro valor é dada por

$$\Delta\theta = \hat{\theta} - \theta. \tag{2.13}$$

Considerando-se um canal com multipercurso no qual cópias do sinal transmitido chegam em instantes de tempos diferentes, uma parte do sinal contido no prefixo cíclico irá conter amostras do símbolo anterior. O número de amostras do prefixo cíclico que contém sinais do símbolo anterior é dado por

$$N_{\max} = \frac{\tau_{\max}}{T_a}, \tag{2.14}$$

onde $\tau_{\max}$ é o espalhamento temporal máximo do canal. O comprimento do intervalo de guarda no padrão 802.11a é de 800 ns para cada símbolo de dados. Sendo o comprimento do atraso do canal multipercurso geralmente de 200 ns ou menos, o comprimento de 800 ns é suficiente para diminuir o ISI no sistema. Apesar da presença do intervalo de guarda, ainda é possível a ocorrência de ISI na recepção, em caso de que o ponto de sincronismo determine uma janela de DFT que contenha amostras do símbolo seguinte ou amostras afetadas pelo multipercurso do símbolo anterior [1].

Considerando-se apenas os erros de temporização, isto é, considerando-se uma sincronismo perfeito de portadora e canal AWGN, o sinal recebido é descrito por

$$r_q\left(k + \hat{\theta}\right) = s_q\left(k + \Delta\theta\right) + w_q\left(k + \hat{\theta}\right). \tag{2.15}$$

Considera-se, sem perda de generalidade, que $\Delta\theta$, θ e $\hat{\theta}$ são inteiros.

Removendo prefixo cíclico e após a FFT, temos

$$\hat{X}_{i,q} = \sum_{k=0}^{N-1} r_q\left(k+\hat{\theta}\right) e^{-j2\pi ki/N} = \sum_{k=0}^{N-1} \left[s_q\left(k+\Delta\theta\right) + w_q\left(k+\hat{\theta}\right)\right] e^{-j2\pi ki/N}. \tag{2.16}$$

Dado que o comprimento em amostras do intervalo de guarda é de 16, se $-16+N_{max} \leq \Delta\theta \leq 0$, o vetor $\vec{s}_q = [s_q(\Delta\theta) \ldots s_q(N-1+\Delta\theta)]$ contém todas as amostras do símbolo OFDM, porém rotacionadas circularmente de $\Delta\theta$ amostras. Isto é, para o caso em que $\Delta\theta \in \{-16+N_{\max}, 0\}$ os símbolos recebidos $\hat{X}_{i,q}$ apenas sofrem uma mudança de fase em relação aos símbolos enviados $X_{i,q}$

$$\hat{X}_{i,q} = X_{i,q}\, e^{j2\pi i\Delta\theta/N}. \tag{2.17}$$

A Figura 2.5 mostra as possíveis posições da janela de DFT para o caso da recepção de três símbolos OFDM.

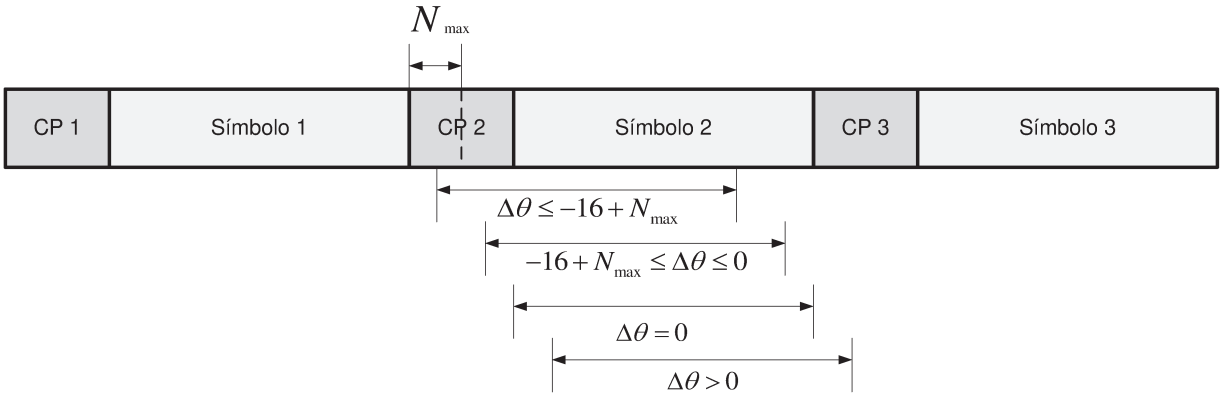


Figura 2.5: Posições possíveis para a janela DFT.

Considerando a Figura 2.5, a posição da janela de DFT pode ser dividida em quatro possibilidades:

1. $\Delta\theta \leq -16+N_{\max}$: Amostras do símbolo anterior serão utilizadas na DFT, devido ao multipercurso, resultando em interferência intersimbólica e interferência entre subportadoras.

2. $-16+N_{\max} \leq \Delta\theta \leq 0$: Serão utilizadas amostras do prefixo cíclico na DFT.

3. $\Delta\theta = 0$: Neste caso, a estimativa do ponto de sincronismo é perfeita e a posição da janela da DFT será a ideal.

4. $\Delta\theta > 0$: Neste caso, amostras do símbolo seguinte serão utilizadas, resultando em ISI e ICI.

2.5.2 Diferença de frequência entre portadoras

Considerando-se um perfeito sincronismo temporal ($\Delta\theta = 0$) e canal AWGN, o sinal recebido é representado como

$$r_q(k) = s_q(k)\, e^{j\left(\frac{2\pi}{N}\varepsilon k + \phi\right)} + w_q(k), \tag{2.18}$$

onde $\varepsilon = \frac{\Delta f}{f_s/N}$ é o desvio de frequência Δf normalizado pelo espaçamento entre portadoras f_s/N, sendo $f_s = 2W$ a taxa de amostragem, N o número de portadoras do sinal OFDM e $\phi = \frac{2\pi}{N}\varepsilon q\,(N+L)$ a fase acumulada, a cada q-ésimo símbolo OFDM, devido à diferença de frequência ε. No receptor, após retirado o prefixo cíclico, a FFT fica representada como

$$\hat{X}_{i,q} = \sum_{k=0}^{N-1} \left[s_q(k)\, e^{j\left(\frac{2\pi}{N}\varepsilon k + \phi\right)} + w_q(k) \right] e^{-j2\pi ki/N}. \tag{2.19}$$

Já que o sincronismo temporal foi considerado perfeito, a variável $s_q(k)$ em (2.19) pode ser substituída pela variável $x_q(k)$ de (2.11), obtendo-se

$$\hat{X}_{i,q} = e^{j\left[\pi\varepsilon\left(\frac{N-1}{N}\right)+\phi\right]} \frac{\operatorname{sen}(\pi\varepsilon)}{N \operatorname{sen}\left(\frac{\pi\varepsilon}{N}\right)} X_{i,q} + \xi_{i,q} + W_{i,q}, \tag{2.20}$$

onde o termo $\xi_{i,q}$ representa a interferência entre portadoras, mais especificamente detalhada por

$$\xi_{i,q} = \frac{e^{j\phi}}{N} \sum_{h=0, h\neq i}^{N-1} X_{h,q} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}k(h-i+\varepsilon)}, \tag{2.21}$$

e $W_{i,q}$ é um ruído branco no domínio da frequência

$$W_{i,q} = \sum_{k=0}^{N-1} \left[w_q(k) \right] e^{-j2\pi ki/N}. \tag{2.22}$$

Através das equações apresentadas até agora, pode-se resumir os efeitos causados pela diferença de frequência entre portadoras (CFO). Na equação (2.20), nota-se que ocorre uma atenuação do símbolo transmitido $X_{i,q}$, bem como uma adição de interferência de todas as portadoras com índice $h \neq i$, como apresentado em (2.21).

Qualquer pequeno desvio de frequência ε quebra a ortogonalidade entre as portadoras, adicionando interferência e atenuações nos símbolos recebidos. A Figura 2.6 ilustra essa interferência entre portadoras, para um offset de $\varepsilon = 0.3$, isto é, 30% do espaçamento entre subportadoras. Nota-se que as portadoras adjacentes ($h = i \pm 1$) são as que mais causam interferência. As outras portadoras também interferem, mas com menos intensidade [8].

Em [9] é analisada a degradação na taxa de bits causada pela diferença de frequência entre portadoras em um canal AWGN. Como esperado, a degradação é maior se comparada aos sistemas de portadora única. Esta degradação é descrita por

$$D(dB) \simeq \frac{10(\pi\varepsilon)^2 \mathrm{SNR}}{3\ln 10} = \frac{10(\pi\Delta f T_c N)^2 \mathrm{SNR}}{3\ln 10}. \tag{2.23}$$

Note-se que a degradação aumenta com o quadrado do número de subportadoras, se Δf e T_c são fixos. Para canais dispersivos, a degradação é limitada superiormente por

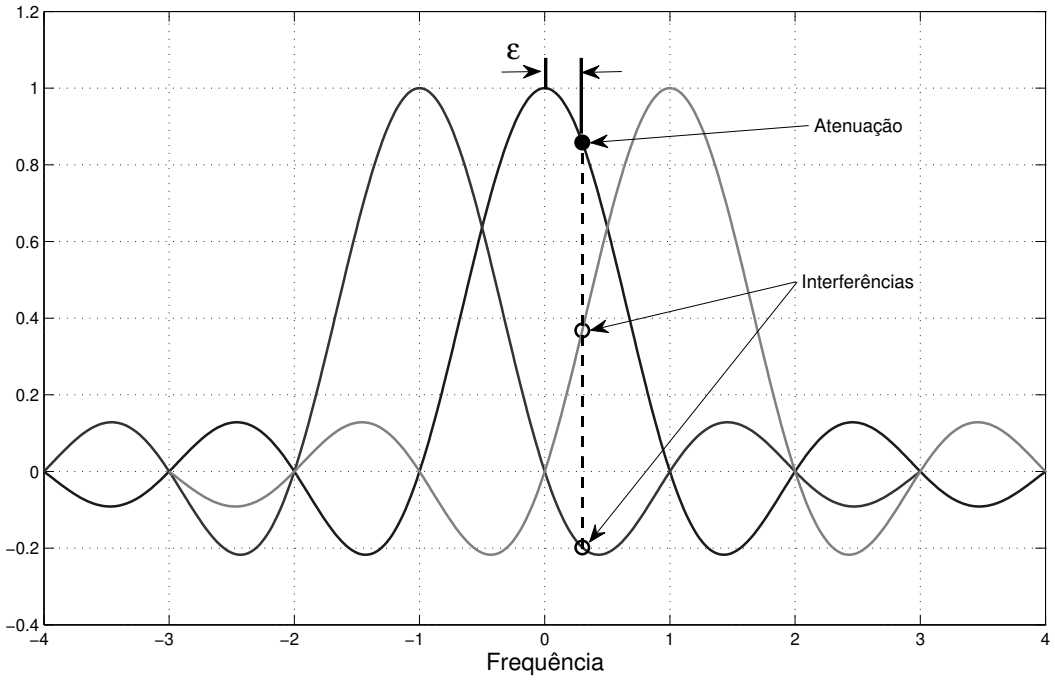


Figura 2.6: Interferência entre subportadoras.

$$D(dB) \leq 10\log_{10}\left(\frac{1 + 0.5947 \cdot \mathrm{SNR}\mathrm{sen}^2(\pi\varepsilon)}{\mathrm{sinc}^2(\pi\varepsilon)}\right). \tag{2.24}$$

O fator 0.5947 encontra-se a partir de o limite inferior da somatória de todas as interferências das subportadoras.

2.6 Padrão IEEE 802.11a

Podem-se dividir os métodos de sincronismo em métodos cegos e métodos *data-aided* [1]. Nesta dissertação, os métodos abordados são *data-aided*, baseados em operações de autocorrelação e correlação cruzada no domínio do tempo das amostras dos símbolos de um preâmbulo.

As redes sem fio do padrão IEEE 802.11 que usam OFDM na camada física, utilizam um formato de preâmbulo semelhante e, portanto, os mesmos métodos acima citados servem para padrões como IEEE 802.11a/g/n/p [10][11][12]. Assim, considerando-se que o padrão 802.11a possui uma extensa informação em propostas de implementação de métodos de sincronismo em hardware digital, foi o padrão escolhido para esta pesquisa.

O padrão IEEE 802.11a para WLAN opera na banda de 5 GHz e atinge taxas de transmissão de até 54 Mbps. O esquema escolhido de modulação para a camada física foi o OFDM, devido a seu bom desempenho em canais altamente dispersivos, como cenários *indoor*, onde o padrão é usado. Nos padrões WLAN, o sinal banda base OFDM é construído utilizando-se uma FFT de 64 pontos. Em seguida, um intervalo de guarda de 16 amostras (também chamado de prefixo cíclico) é adicionado com o propósito de robustecer o sistema contra o multipercurso e prevenir interferência entre símbolos (ISI). Com uma frequência de amostragem de 20 MHz, cada símbolo

tem duração de 4 μs (80 amostras), onde se inclui o intervalo de guarda de 800 ns. Com o intuito de oferecer uma banda de guarda ao canal adjacente, só 52 subportadoras são usadas: 48 para dados e 4 para portadoras-pilotos. Portanto, o espaçamento de subportadora é de 312.5 kHz e o espaçamento entre as duas subportadoras das extremidades é de 16.25 MHz. Na Tabela 2.1 são apresentados os principais parâmetros do padrão IEEE 802.11a utilizados ao longo deste projeto.

Tabela 2.1: Parâmetros do padrão IEEE 802.11a.

Parâmetro	**Símbolo**	**Valor**
Largura de banda	W	20 MHz
Período de amostragem	T_a	50 ns
Subportadoras no canal / Número de pontos da FFT	N_{FFT}	64
Subportadoras de dados	N_{dados}	48
Subportadoras de pilotos	$N_{pilotos}$	4
Distância entre subportadoras	$\Delta_F = W / N_{FFT}$	312.5 kHz
Período do símbolo	$T_s = 1 / \Delta_F$	3.2 μs
Número de amostras do símbolo	$N = T_s / T_a$	64

O pacote transmitido do padrão IEEE 802.11a é composto de um cabeçalho, seguido dos dados de carga útil. O cabeçalho contém informações acerca do pacote de que o receptor necessita (como duração do pacote e taxa de transmissão), além de conter um preâmbulo usado para sincronismo. A composição de um pacote no domínio do tempo é formada por um conjunto de campos transmitidos em sequência, cada um com uma função particular. Na Figura 2.7, observa-se que o primeiro intervalo do sinal é o trecho do preâmbulo, formado por símbolos curtos (STS) e longos (LTS). Depois, vem o *Signal Field* (SIG) e os blocos de dados. Na seção seguinte, são detalhadas as características e composição do preâmbulo, formado por um conjunto de símbolos que são usados para sincronizar o sistema.

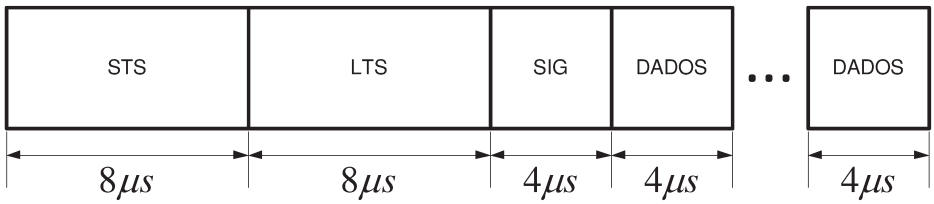


Figura 2.7: Estrutura de um pacote do padrão IEEE 802.11a.

2.6.1 Preâmbulo IEEE 802.11a

O preâmbulo é composto por dois conjuntos de símbolos: curtos (nomeados STS, do inglês *Short Training Sequence*) e longos (nomeados LTS, do inglês *Long Training Sequence*). Cada símbolo STS é composto de 16 amostras complexas e um símbolo LTS de 64 amostras. Antes

do primeiro símbolo LTS, é inserido um intervalo de guarda composto por 32 amostras do final do símbolo LTS. A Figura 2.8 apresenta o formato do preâmbulo.

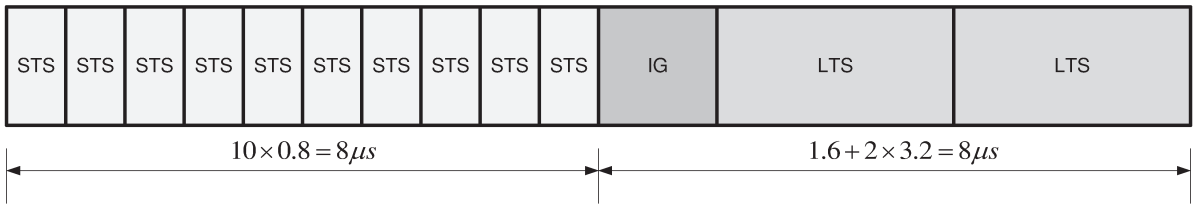


Figura 2.8: Preâmbulo IEEE 802.11a.

Por um lado, um símbolo STS tem a seguinte atribuição de subportadora:

$$\begin{aligned} S_{-26,26} = \sqrt{\frac{13}{6}}\{&0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,\\ &-1-j,0,0,0,1+j,0,0,0,\mathbf{0},0,0,0,-1-j,0,0,0,-1-j,\\ &0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0\}. \end{aligned} \tag{2.25}$$

No vetor acima, o elemento central realçado em negrito corresponde à componente DC. O STS utiliza 12 das 52 subportadoras disponíveis, sendo assim definido para resultar em um sinal com boas propriedades de correlação e uma baixa relação entre o pico e a média da potência. Essas características são fundamentais para a detecção, o sincronismo e o controle automático de ganho no receptor.

O sinal em (2.25) é levado para o domínio do tempo através da realização de uma IFFT, obtendo-se os STS transmitidos.

$$r_{short}(t) = w_{T_{short}}(t) \sum_{k=-N_{ST}/2}^{N_{ST}/2} S_k \, e^{kj2\pi\Delta f t}, \tag{2.26}$$

onde $T_{short} = 8\mu s$ é a duração total do STS, $N_{ST} = 52$ é o número de subportadoras e $\Delta f = 312.5$kHz corresponde ao espaçamento de frequência entre subportadoras. A equação $w(t)$, mostrada em (2.27), é uma função de janelamento no tempo, usada para suavizar a transição entre os símbolos OFDM [13].

$$w(n\,T_S) = \begin{cases} 1 & 1 \leqslant n \leqslant 159 \\ 0.5 & n = 0, 160 \\ 0 & \text{caso contrário} \end{cases} \tag{2.27}$$

onde T_S é o período de amostragem que corresponde a 50 ns. Essa janela é aplicada após encadear os 10 símbolos curtos.

O sinal temporal discreto é obtido através de uma IFFT de 64 pontos (i.e., N =64), o que gera um sinal de tempo discreto, também com 64 amostras. No entanto, conforme observado em (2.25), o sinal na frequência foi definido através de 52 subportadoras. Portanto, deve-se completar esse sinal de forma a conter o mesmo número de pontos utilizados na IFFT. Para

isto, são inseridos 11 zeros que equivalem às portadoras mantidas nulas como intervalo de guarda entre os canais. Dessa forma, aplicando-se a IDFT ao sinal da equação (2.25), obtém-se

$$r_n = \frac{1}{N}\sum_{k=0}^{N-1} S_k\, e^{\frac{j2\pi kn}{N_{FFT}}}, \quad n = 0, 1, ..., N-1. \tag{2.28}$$

O sinal criado possui uma duração de $3.2\mu s$, ou 64 pontos, no tempo discreto, a uma taxa de amostragem de 20 MHz. Embora o sinal calculado via IFFT tenha 64 amostras, ele exibe uma periodicidade que define 4 símbolos curtos. Como o símbolo curto tem periodicidade de $0.8\mu s$, então o sinal gerado contém 4 símbolos curtos. Para gerar o STS completo, esse sinal deve ser repetido duas vezes e meia, resultando em 10 símbolos curtos cuja duração total é de $8\mu s$ ou 160 amostras complexas.

Por outro lado, o símbolo LTS tem a seguinte atribuição de subportadora:

$$\begin{aligned} L_{-26,26} = \{&1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, \\ &1, 1, -1, 1, -1, 1, 1, 1, 1, \mathbf{0}, 1, -1, -1, 1, 1, -1, 1, -1, 1, \\ &-1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\}. \end{aligned} \tag{2.29}$$

No vetor acima, o elemento central realçado em negrito corresponde à componente DC. O sinal em (2.29) é levado para o domínio do tempo através da realização de uma IFFT, obtendo-se os LTS transmitidos.

$$r_{long}(t) = w_{T_{long}}(t) \sum_{k=-N_{ST}/2}^{N_{ST}/2} S_k\, e^{kj2\pi\Delta f(t-T_{GI2})}, \tag{2.30}$$

onde $T_{long} = 8\mu s$ é o comprimento total da sequência LTS e $T_{GI2} = 1.6\mu s$ é o comprimento do intervalo de guarda inserido antes do primeiro LTS.

Do mesmo modo, utiliza-se uma IFFT de 64 pontos para gerar o sinal no tempo. Portanto, os passos descritos anteriormente para o STS também são realizados. Dessa forma, aplicando-se a IDFT ao sinal da equação (2.29), obtém-se

$$r_n = \frac{1}{N}\sum_{k=0}^{N-1} L_k\, e^{\frac{j2\pi kn}{N_{FFT}}}, \quad n = 0, 1, ..., N-1. \tag{2.31}$$

Para gerar o LTS completo, deve-se repetir esse símbolo duas vezes, além de acrescentar o prefixo cíclico. Em outras palavras, o procedimento para criar o LTS consiste em gerar um símbolo sem o prefixo cíclico e repetir esse símbolo de forma a obter os dois símbolos completos. Em seguida, deve-se acrescentar o prefixo cíclico, que é inserido no começo do LTS com $1.6\mu s$ de duração (metade de um símbolo longo), sendo formado pelas últimas 32 amostras do símbolo original.

Os símbolos repetidos presentes no preâmbulo têm o objetivo de facilitar ao receptor o reconhecimento da presença de símbolos semelhantes, independentemente dos efeitos do canal. O padrão IEEE 802.11a especifica que os primeiros sete símbolos curtos sejam usados para a detecção do sinal, o Controle Automático de Ganho (AGC) e a seleção de diversidade (para sistemas MIMO). Os últimos três símbolos curtos devem ser utilizados para estimativa grosseira de frequência e o sincronismo de tempo. Os símbolos longos são destinados à estimação de canal e a estimativa fina de frequência. No entanto, também podem ser usados para refinar a estimativa de sincronismo de tempo.

2.7 Considerações para estimação e compensação

Para o sincronismo em um receptor OFDM, a questão é onde estimar e onde compensar: no domínio do tempo ou no domínio da frequência. Para resolver esta questão é necessário considerar os tipos de transmissão, recursos do sistema, latência, precisão em estimação e compensação, entre outros fatores [3].

Os tipos de transmissões são classificados em sistemas de transmissão de pacotes e transmissão de quadros. Nos sistemas baseados em pacotes, tais como o WLAN 802.11a/g/n, os dados são arranjados em pacotes. O comprimento máximo de cada pacote é limitado de modo que as deficiências do canal e parâmetros de sincronismo permanecem quase estacionários dentro de um pacote. Cada pacote contém no cabeçalho um preâmbulo que facilita o sincronismo. Depois do cabeçalho, a informação de usuário completa o pacote. Com tal estrutura de pacote, o receptor deve iniciar a detecção do sinal imediatamente após a recepção do preâmbulo. Para isso, é necessário que o bloco de sincronismo responda imediatamente, isto é, ele deve estimar e compensar os erros de sincronismo no sinal recebido quanto antes. Isto difere de sistemas OFDM baseados em quadros, onde o receptor pode necessitar de um tempo maior até encontrar a sincronização correta [14]. Devido ao fato de que o bloco da FFT requer vários ciclos para calcular o sinal no domínio da frequência, a maioria das tarefas de sincronismo do receptor são geralmente realizadas no domínio do tempo. Além disso, o preâmbulo é um sinal periódico que possui uma boa propriedade de autocorrelação, fazendo preferível o processamento do sinal no domínio do tempo.

Em sistemas 802.11a, nos quais a informação é enviada em pacotes, o receptor possui somente uma chance de realizar o sincronismo durante um curto intervalo de tempo. Caso o sincronismo não seja obtido o pacote será inteiramente perdido e a informação terá de ser reenviada. Por conseguinte, os algoritmos de sincronismo para sistemas de transmissão em pacotes necessitam de uma elevada taxa de acerto, além de não poderem ser complexos a ponto de exigir um alto custo computacional.

Portanto, os algoritmos escolhidos para a implementação são realizados no domínio do tempo, isto é, antes do bloco de FFT. Na próxima seção, serão apresentados os algoritmos de detecção do pacote, sincronismo de tempo e sincronismo de frequência.

2.8 Algoritmos de sincronismo

O sincronismo do pacote é dividido em várias fases, que são realizadas em sequência. Um grande número de algoritmos de sincronismo emprega os símbolos do preâmbulo. A primeira etapa é a detecção do pacote, seguida do sincronismo temporal. Em seguida, é realizado o sincronismo de frequência, que corrige possíveis desvios de frequência. Nas seções seguintes, são detalhados os algoritmos escolhidos para implementação em FPGA.

2.8.1 Detecção do pacote

A primeira etapa do sincronismo é a detecção do pacote. Em [15], aproveitando a repetição de símbolos no preâmbulo, a técnica usada é a autocorrelação junto a um detector de pico. A correlação é feita entre a sequência recebida e uma cópia atrasada no tempo da mesma sequência, com um atraso equivalente ao comprimento do símbolo. A operação de autocorrelação é efetuada por

$$R(d) = \sum_{m=0}^{L-1} \left(r^*_{d+m}\, r_{d+m+L} \right), \tag{2.32}$$

onde r_d representa o valor da d-ésima amostra recebida, r^*_d representa o conjugado de r_d, e no caso dos símbolos STS, $L = 16$ amostras.

Uma grande vantagem da autocorrelação é que esse algoritmo pode ser implementado utilizando uma equação recursiva, o que reduz o processamento requerido. Na equação (2.33), é definida a fórmula recursiva que requer duas multiplicações, uma soma e uma subtração para cada novo valor de saída do detector [16].

$$R(d+1) = R(d) + \left(r^*_{d+L}\, r_{d+2L} \right) - \left(r^*_d\, r_{d+L} \right). \tag{2.33}$$

Em [16] é desenvolvido um método de detecção de pacote no qual a autocorrelação é normalizada pela soma móvel da potência recebida, tal como é mostrado por

$$P(d) = \sum_{m=0}^{L-1} |r_{d+m+L}|^2. \tag{2.34}$$

$$M(d) = \frac{|R(d)|^2}{(P(d))^2}. \tag{2.35}$$

O valor resultante é comparado a um limiar, th, e se considera um pacote detectado se $M(d) > th$. O valor do limiar th deve ser escolhido com o fim de minimizar a ocorrência da detecção de falsos positivos e evitar falha na detecção de pacotes certos, que acontece quando o receptor é incapaz de detectar o preâmbulo.

2.8.2 Sincronismo de tempo

O sincronismo temporal consiste em determinar o tempo ótimo no qual a leitura dos símbolos deve ser feita, isto é, detectar o início de cada símbolo OFDM e encontrar a posição correta para a janela da transformada rápida de Fourier, conforme descrito na seção 2.4.1. Um sincronismo pouco preciso no tempo provoca interferência inter-simbólica, além de poder provocar também interferência inter-portadora.

O método de sincronismo proposto em [16] é baseado na métrica da equação (2.35), onde o valor máximo de $M(d)$ é encontrado juntamente com o índice da amostra na qual se dá aquele valor máximo, $d_{\max}$. O deslocamento de tempo é calculado como a diferença entre o valor $d_{\max}$ e a posição esperada desse máximo. Outra alternativa propõe buscar os dois pontos onde $M(d)$ é 90% do valor máximo $M(d_{\max})$, usando o ponto médio entre aquelas duas amostras para estimar o deslocamento. O primeiro método tem a desvantagem de carecer de precisão e o segundo tem uma implementação em hardware não muito simples.

Outra abordagem é introduzida em [17]. Esse método depende do cálculo de $R(d)$ e do cálculo de outra sequência de autocorrelação, com uma separação de amostras de $2L$

$$R_2(d) = \sum_{m=0}^{L-1} \left(r^*_{d+m}\, r_{d+m+2L} \right). \tag{2.36}$$

A diferença entre estas duas sequências é calculada

$$R_{dif}\,(d) = R\,(d) - R_2(d). \tag{2.37}$$

Esta sequência de diferença tem tipicamente um pico triangular durante o intervalo de guarda LTS, e o índice d_{MaxDif} deste pico pode ser utilizado para calcular o deslocamento de tempo.

Outra alternativa é o método da correlação cruzada entre a sequência recebida e os valores originais do preâmbulo, cujo cálculo é efetuado por

$$\Lambda\,(d) = \sum_{m=0}^{L-1} c_m^*\, r_{d+m}, \tag{2.38}$$

onde o termo c_m^* é o complexo conjugado dos valores das amostras do preâmbulo, L ($L = 16$ para STS e $L = 64$ para LTS) é o comprimento do símbolo e r_d é a sequência recebida.

A desvantagem principal deste algoritmo é que o cálculo não pode ser realizado recursivamente, o que aumenta a carga computacional requerida. Para cada valor de saída, é necessária a realização de L multiplicações complexas. Assim, para economizar recursos, pode ser aplicada uma quantização aos dados de entrada [18] sem afetar o desempenho do algoritmo.

O algoritmo de correlação cruzada usa os símbolos LTS e um detector de pico que procura o valor máximo de $\Lambda\,(d)$.

$$d_{xc\max} = \arg\max_d \left(|\Lambda\,(d)|\right). \tag{2.39}$$

Os símbolos STS também podem ser usados no algoritmo de correlação cruzada. Se os símbolos originais STS forem correlacionados com a sequência recebida, haverá um conjunto de picos em $\Lambda\,(d)$ a cada 16 amostras. O deslocamento de tempo é calculado no ponto em que aqueles picos de correlação deixam de ocorrer.

A vantagem da autocorrelação em relação à correlação cruzada é a simplicidade de implementação e economia de recursos multiplicadores usando a equação recursiva 2.33. No entanto, em termos de variância dos resultados das estimativas dos pontos de sincronismo, a correlação cruzada permite uma estimativa mais fina.

2.8.3 Sincronismo de frequência

O desvio de frequência é ocasionado pela diferença existente entre as frequências do oscilador do transmissor e do receptor, bem como pelo efeito Doppler, que ocorre quando existe um movimento relativo entre a fonte e o receptor [2]. Essa diferença na frequência pode destruir a ortogonalidade entre as subportadoras e causar interferência entre subportadoras (ICI). O sincronismo em frequência visa estimar esse desvio de frequência e corrigi-lo.

Se duas amostras semelhantes são transmitidas pelo canal, a diferença de fase entre elas no receptor é proporcional ao desvio de frequência, e também proporcional à separação entre os dois tempos de transmissão. O desvio de frequência f_Δ é proporcional ao desvio de fase, conforme mostrado por

$$\phi = 2\pi t f_\Delta. \tag{2.40}$$

Essa diferença de fase pode ser calculada através da observação das amostras recebidas, separadas pela duração de um símbolo. Se s_n é o símbolo curto de treinamento, o sinal discreto transmitido é definido por

$$y_n = s_n \exp\left(j2\pi\, f_{tx}\, n\, T_s\right), \tag{2.41}$$

onde f_{tx} é a frequência da portadora e T_s é o tempo de amostragem. O símbolo discreto em banda base recebido é definido por

$$r_n = s_n \exp\left(j2\pi\, f_{\Delta}\, n\, T_s\right), \tag{2.42}$$

onde o desvio de frequência da portadora $f_{\Delta} = f_{tx} - f_{rx}$ é a frequência da portadora recebida. A função de autocorrelação de r_n é derivada como

$$\begin{aligned} R\left(d\right) &= \sum_{m=0}^{L-1} r_{m+d}\, r^*_{m+d+L} \\ R\left(d\right) &= \sum_{m=0}^{L-1} \left[s_{m+d} \exp\left(j2\pi\, f_{\Delta}\left(m+d\right) T_s\right)\right] \left[s_{m+d+L} \exp\left(j2\pi\, f_{\Delta}\left(m+d+L\right) T_s\right)\right]^* \\ R\left(d\right) &= \exp\left(-j2\pi\, f_{\Delta}\, L\, T_s\right) \sum_{m=0}^{L-1} s_{m+d}\, s^*_{m+d+L} \end{aligned} \tag{2.43}$$

onde $L = 16$ para o caso dos STS.

A relação entre ϕ na equação (2.40) e o termo $R(d)$ da equação (2.43) é dada por

$$\phi \approx \arg\left[R(d)\right]. \tag{2.44}$$

Para $L = 16$ existe uma diferença de tempo total de $16\, T_s$, onde T_s é a duração de uma amostra. Essa fase é calculada com o algoritmo de CORDIC em modo vetorial. Portanto, depois a determinação da diferença de fase, a estimativa do desvio de frequência é aproximada por

$$f_{\Delta} \approx \frac{\arg\left[R(d)\right]}{2\pi \times 16\, T_s}. \tag{2.45}$$

Na implementação, o valor de $\arg\left[R(d)\right]$ é calculado empregando-se o algoritmo CORDIC. O valor de $\arg\left[R(d)\right]$ fica entre π e $-\pi$, e portanto os valores-limites do desvio de frequência serão

$$-625\ kHz \leqslant f_{\Delta} \leqslant 625\ kHz. \tag{2.46}$$

Assim, o desvio máximo identificável equivale a duas vezes o espaçamento entre cada subportadora.

No caso da estimativa fina do desvio de frequência, as equações (2.32) e (2.43) são calculadas usando-se os símbolos LTS, onde $L = 64$ amostras, melhorando assim a precisão por um fator de 4. Portanto, os valores-limites do desvio de frequência serão:

$$-156.25\ kHz \leqslant f_{\Delta} \leqslant 156.25\ kHz. \tag{2.47}$$

Assim, o desvio máximo é a metade do espaçamento entre as subportadoras.

O padrão IEEE 802.11a estabelece que a máxima tolerância para a frequência central é de ± 20 partes por milhão (ppm), o que corresponde a um desvio máximo de frequência de 200 kHz quando a frequência de portadora é 5 GHz.

Para se corrigir o desvio de frequência, é empregada a equação (2.48). O sinal é multiplicado por uma exponencial complexa, com frequência idêntica ao desvio estimado, mas de valor oposto.

$$s_d = r_d \, e^{-j2\pi f_\Delta n \, T_s} \, . \tag{2.48}$$

Para a geração das componentes de uma exponencial complexa, é usado o algoritmo CORDIC em modo rotacional. Uma revisão do algoritmo será apresentada na próxima seção. No CORDIC em modo rotacional, o vetor (a, b) é girado por um ângulo θ para obter um novo vetor (c, d)

$$(c, d) = (a\cos(\theta) - b\,\mathrm{sen}(\theta)\,, b\cos(\theta) + a\,\mathrm{sen}(\theta))\,. \tag{2.49}$$

O CORDIC gera simultaneamente a função seno e coseno da fase θ, estabelecendo $a = 1$ e $b = 0$ em (2.49). Assim, $c = \cos(\theta)$ e $d = \mathrm{sen}(\theta)$, o que permite gerar uma exponencial complexa, conforme a fórmula de Euler.

Capítulo 3

Ambiente de simulação

Nesta seção, são apresentados o ambiente de simulação, as ferramentas de software e a estrutura geral de um *test bench*. Da mesma forma, são apresentados o modelo de canal, valores dos símbolos curtos e longos e uma análise da quantização das amostras de entrada.

3.1 Software e Hardware

A implementação e a simulação dos algoritmos de sincronismo foram realizadas usando o ambiente Xilinx WebPACK com a FPGA Spartan 3 Started Kit. Com relação à programação da FPGA, o conjunto de ferramentas de software associado a um fluxo de projeto provê um nível de abstração que permite focar no algoritmo que se deseja implementar, em vez de se preocupar com os circuitos que serão implementados. Dessa forma, a programação do dispositivo pode ser feita através de uma linguagem de programação (VHDL) ou mesmo através da modelagem de sistemas (*System Generator*) [4]. A linguagem HDL escolhida neste trabalho foi VHDL, por ser uma linguagem de descrição de hardware que permite criar projetos independentes da tecnologia e que garante flexibilidade, re-utilização do código, escolha de ferramentas, fornecedores, bem como permite explorar em um nível mais alto de abstração diferentes alternativas de implementação e, através de simulações, verificar o comportamento do sistema digital.

Da mesma forma, uma simulação em MATLAB foi realizada para comparar e validar os dados processados na FPGA. Os dados de entrada ao circuito são gerados no MATLAB e armazenados em um arquivo de extensão *.txt* que é lido pelo *test bench* do projeto em VHDL. O *test bench* é um ambiente no qual o projeto, chamado de *design* ou *unit under test*, é verificado, por meio da aplicação de sinais ou estímulos e da monitoração de suas respostas. Em outras palavras, um *test bench* substitui o ambiente de projeto, de forma que o comportamento do circuito possa ser observado e analisado.

Da mesma forma, os dados resultantes processados pela FPGA são armazenados em um arquivo de texto *.txt* que contém os valores binários da simulação, os quais são transformados em formato decimal e analisado no MATLAB. O processo de simulação é mostrado na Figura 3.1.

3.2 Modelo de canal

O modelo de canal implementado é o denominado *tapped-delay*, que inclui os efeitos de desvanecimento por multipercurso, desvio de frequência (CFO) e ruído gaussiano aditivo branco

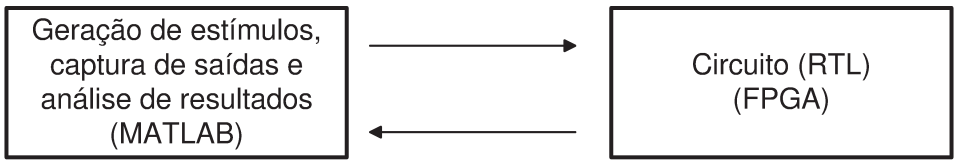


Figura 3.1: Estrutura geral de um *test bench*.

(AWGN). Um desvio Doppler de 52 Hz é assumido para cada *tap* [19]. A resposta ao impulso no tempo t é descrita por

$$h\left(t,\tau\right)=\sum_{r}h_{r}\left(t\right)\delta\left(\tau-\tau_{r}\left(t\right)\right). \tag{3.1}$$

Em (3.1) o atraso e o ganho do r-ésimo percurso são, respectivamente, $\tau_r(t)$ e $h_r(t)$. A partir de (3.1) para o exemplo de uma única parede refletora, $\tau_r^{'}=\upsilon_r/c$ onde υ_r é a velocidade com a qual o r-ésimo comprimento de percurso é aumentado. Assim, o desvio Doppler sobre o r-ésimo percurso é $-f\tau_r^{'}(t)$.

Realizando-se a convolução do sinal transmitido $x(t)$ com a resposta ao impulso do canal e adicionando-se o ruído do canal, $v\left(\tau\right)$, o sinal em banda base recebido é dado por

$$z\left(\tau\right)=\sum_{r}h_{r}\left(t\right)x\left(\tau-\tau_{r}\left(t\right)\right)+v\left(\tau\right). \tag{3.2}$$

Além disso, considerando-se que o canal seja quase estacionário, isto é, as atenuações $h_r(t)$ e os atrasos de propagação $\tau_r(t)$ não depende do tempo t, a resposta ao impulso é representada por

$$h(\tau)=\sum_{r}h_{r}\,\delta\left(\tau-\tau_{r}\right). \tag{3.3}$$

Os valores de τ_r e h_r seguem as características de um canal-padrão da indústria, em particular, o modelo de canal do *European Telecommunications Standards Institute* (ETSI).

Dentro desse padrão, os modelos de canais escolhidos foram ETSI A (com 50 ns de atraso de propagação) e ETSI C (com 150 ns de atraso de propagação), devido ao fato de que são modelos consolidados na literatura. O primeiro modelo simula o ambiente interno de um escritório. O segundo modelo simula um ambiente de espaço aberto. Os parâmetros desses tipos de canais são mostrados nas Tabelas 3.1 e 3.2 com 18 multipercursos [20].

Da mesma forma, é introduzido um desvio de frequência na sequência de entrada, detalhado em (3.4). O valor de cada desvio simula o caso com condições ótimas ($\Delta f = 0$ kHz), moderadas ($\Delta f = 100$ kHz) e severas ($\Delta f = 200$ kHz).

$$r_{n}=z\left(t\right)e^{j2\pi\Delta ft}\big|_{t=n\,T_{s}}. \tag{3.4}$$

O esquema do modelo de canal é exibido na Figura 3.2.

Tabela 3.1: Modelo de canal ETSI A: atraso e potência.

Atraso(μs)	**Potência (dB)**	**Atraso(μs)**	**Potência (dB)**
0.00	0.00	0.09	-7.80
0.01	-0.90	0.10	-4.70
0.02	-1.70	0.14	-7.30
0.03	-2.60	0.17	-9.90
0.04	-3.50	0.20	-12.50
0.05	-4.30	0.24	-13.70
0.06	-5.20	0.29	-18.00
0.07	-6.10	0.34	-22.40
0.08	-6.90	0.39	-26.70

Tabela 3.2: Modelo de canal ETSI C: atraso e potência.

Atraso(μs)	**Potência (dB)**	**Atraso(μs)**	**Potência (dB)**
0.00	-3.30	0.23	-3.00
0.01	-3.60	0.28	-4.40
0.02	-3.90	0.33	-5.90
0.03	-4.20	0.40	-5.30
0.05	0.00	0.49	-7.90
0.08	-0.90	0.60	-9.40
0.11	-1.70	0.73	-13.20
0.14	-2.60	0.88	-16.30
0.18	-1.50	1.05	-21.20

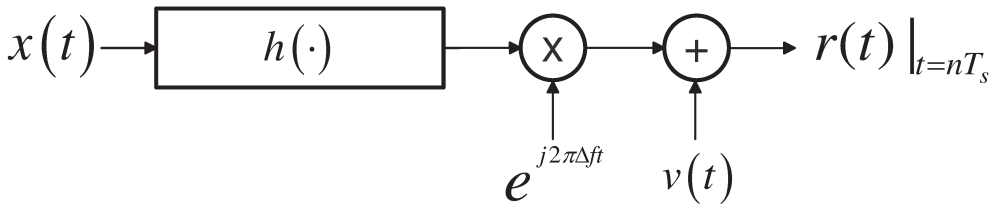


Figura 3.2: Modelo de canal.

3.3 Sequência de entrada

Na implementação, foram considerados apenas os valores do preâmbulo (320 amostras), já que processar todos os valores do pacote (especificados nas Tabelas G.4 e G.6 em [13]) fornece informação não necessária para a detecção do pacote, e para os sincronismos temporal e de frequência. A Tabela 3.3 contém as 16 amostras dos valores para um símbolo STS, resultado de (2.26). A Tabela 3.4 mostra os resultados provenientes de (2.30) e contém os valores das 64 amostras para um símbolo LTS.

Uma vez geradas, as amostras dos símbolos STS e LTS são arranjadas para formar o preâmbulo. Essa sequência é convoluída com a resposta ao impulso do canal para logo ser multiplicada

Tabela 3.3: Amostras do símbolo STS.

N	Re	Im	N	Re	Im
1	0.046	0.046	9	0.046	0.046
2	-0.132	0.002	10	0.002	-0.132
3	-0.014	-0.079	11	-0.079	-0.014
4	0.143	-0.013	12	-0.013	0.143
5	0.092	0.000	13	0.000	0.092
6	0.143	-0.013	14	-0.013	0.143
7	-0.014	-0.079	15	-0.079	-0.014
8	-0.132	0.002	16	0.002	-0.132

Tabela 3.4: Amostras do símbolo LTS.

N	Re	Im	N	Re	Im	N	Re	Im	N	Re	Im
1	0.156	0.000	17	0.063	-0.063	33	-0.156	0.000	49	0.063	0.063
2	-0.005	-0.120	18	0.037	0.098	34	0.012	-0.098	50	0.119	0.004
3	0.040	-0.111	19	-0.057	0.039	35	0.092	-0.106	51	-0.023	-0.161
4	0.097	0.083	20	-0.131	0.065	36	-0.092	-0.115	52	0.059	0.015
5	0.021	0.028	21	0.082	0.092	37	-0.003	-0.054	53	0.025	0.059
6	0.060	-0.088	22	0.070	0.014	38	0.075	0.074	54	-0.137	0.047
7	-0.115	-0.055	23	-0.060	0.081	39	-0.127	0.021	55	0.001	0.115
8	-0.038	-0.106	24	-0.057	-0.022	40	-0.122	0.017	56	0.053	-0.004
9	0.098	-0.026	25	-0.035	-0.151	41	-0.035	0.151	57	0.098	0.026
10	0.053	0.004	26	-0.122	-0.017	42	-0.057	0.022	58	-0.038	0.106
11	0.001	-0.115	27	-0.127	-0.021	43	-0.060	-0.081	59	-0.115	0.055
12	-0.137	-0.047	28	0.075	-0.074	44	0.070	-0.014	60	0.060	0.088
13	0.025	-0.059	29	-0.003	0.054	45	0.082	-0.092	61	0.021	-0.028
14	0.059	-0.015	30	-0.092	0.115	46	-0.131	-0.065	62	0.097	-0.083
15	-0.023	0.161	31	0.092	0.106	47	-0.057	-0.039	63	0.040	0.111
16	0.119	-0.004	32	0.012	0.098	48	0.037	-0.098	64	-0.005	0.120

por uma exponencial complexa com frequência de 0, 100 e 200 kHz. Finalmente, é adicionado ruído AWGN.

A seguinte seção apresenta o modo de conversão de um formato decimal da amostra a processar a um formato binário em complemento de 2 que será processado na FPGA.

3.3.1 Quantização das amostras

Considerando que um número racional com sinal em ponto fixo é um valor da forma $B_i \,/\, 2^b$, onde B_i e b são inteiros, com uma faixa de $-2^{M-1} \leq B_i \leq 2^{M-1}-1$, e M é o comprimento da palavra binária usado para representar cada amostra, a aproximação $b_i^{'}$ da amostra b_i escolhendo um valor de b para calcular B_i é representada por

$$B_i = \lfloor b_i \times 2^b \rceil, \tag{3.5}$$

onde $\lfloor \rceil$ é a função inteiro mais próximo.

Portanto

$$b_i^{'} = \frac{B_i}{2^b}. \tag{3.6}$$

Em geral, o coeficiente $b_i^{'}$ é só uma aproximação de b_i devido à operação de arredondamento, que é o coeficiente de quantização. Neste sentido, está-se realizando uma quantização das amostras em amplitude de forma análoga à de um conversor A/D que faz uma quantização de um sinal analógico. O erro de quantização e_i entre a aproximação e o valor real é dado por

$$e_i = b_i^{'} - b_i\,. \tag{3.7}$$

Para determinar o valor ótimo de b, considera-se o erro máximo de quantização dado em (3.8). Note-se que o erro diminui conforme b aumenta.

$$e_{i\,\max} = \frac{2^{-b}}{2}. \tag{3.8}$$

Considerando uma palavra binária de M bits, um número em complemento de 2 possui uma magnitude máxima de 2^{M-1}. Portanto, deve ser escolhido um valor de b garantindo que o valor de B_i não seja maior que 2^{M-1} para evitar o *overflow*. Assim, o valor ótimo de b é calculado por

$$b = \left\lfloor \log_2 \left(\frac{2^{M-1}-1}{\max\left(|b_i|\right)} \right) \right\rfloor, \tag{3.9}$$

onde $\lfloor x \rfloor$ é o inteiro menor ou igual que x. O valor b_i representa as amostras geradas para a simulação.

Em resumo, o valor ideal para b é o valor máximo que pode ser usado sem provocar *overflow* nas amostras, desde que proporcione o erro mínimo de quantização [21].

Na implementação foram considerados dois tipos de canais, cada um com três valores de Δf. Para cada modelo de canal, formam criados 200 sinais em cada valor de desvio de frequência, gerando um total de 1200 sinais de testes. Em cada sinal de teste, buscou-se o valor absoluto máximo tanto da parte real como da imaginária. Do total de sinais de teste, com uma SNR=10 dB, o valor máximo foi de 0.536654640; portanto, aplicando-se (3.9) com $M = 12$, o valor de b é 11. Dessa forma, cada amostra do sinal de teste é multiplicada por 2^{11}. Depois, o valor resultante é arredondado e convertido para um número binário de $M = 12$ bits, garantindo um erro mínimo de quantização e evitando que aconteça *overflow*. Se o valor da amostra é um número negativo, a conversão para binário inclui uma operação de complemento de 2. No MATLAB, o preâmbulo é armazenado em dois tipos de arquivos: um de formato *.mat* para as simulações e um arquivo *.txt*, que contém os valores binários de cada amostra do preâmbulo que será lido no *test bench*.

Inicialmente, a implementação foi desenvolvida com um sinal sem ruído nem desvio de frequência com o intuito de verificar o resultado e a precisão das operações envolvidas no sincronismo. Uma vez verificado o correto funcionamento, cada algoritmo foi testado para um total de 1200 sinais diferentes. Portanto, esta tarefa foi realizada no Xilinx ISE com o modo de linha de comandos, sendo possível, assim, diminuir notavelmente o tempo de verificação, uma vez que não é necessário usar o Xilinx ISE em modo de interface gráfica [22].

Capítulo 4

Implementação dos algoritmos de sincronismo

Esta seção apresenta a implementação dos algoritmos de sincronismo, arquiteturas, componentes e recursos da FPGA. Conforme explicado na seção anterior, a primeira etapa de sincronismo é a detecção do pacote, seguida do sincronismo de tempo, da estimativa e da correção do desvio de frequência da portadora. Cada uma dessas etapas realiza operações de deslocamento de bit, registros de atraso, multiplicação complexa, acumulação, estimativa de fase e geração das funções seno e coseno para o cálculo da autocorrelação e da correlação cruzada, necessárias para determinar a presença do pacote e definir a amostra do início do pacote e o desvio de frequência da portadora.

Na quantificação da precisão de um algoritmo de sincronismo de tempo, uma métrica útil é a variância das estimativas produzidas pelo algoritmo. As diferentes condições simuladas no canal irão produzir diferentes estimativas para o desvio de tempo e o deslocamento de frequência. A variância dentro de uma série de estimativas é indicativa de quão bem um algoritmo pode suportar o ruído aleatório introduzido pelo canal. A variância será não nula, porque o desvio de tempo não será idêntico em todos os casos. Uma variância menor é desejável.

A eficiência de um sincronizador, em termos de hardware, é determinada pela quantidade de recursos que ele ocupa na FPGA (número de *slices*) e a frequência de operação. As contagens de recursos são tomadas após a síntese em hardware, mas antes do *Place&Route*, porque os algoritmos de roteamento muitas vezes sacrificam área para melhorar a velocidade do circuito.

Assim, as métricas utilizadas no presente trabalho para avaliar e comparar os algoritmos de sincronismo de tempo são a variância, número de *slices* e frequência de operação.

Por outro lado, no sincronismo de frequência o bloco modular é o CORDIC. Neste caso, o importante é determinar a relação do número de iterações do algoritmo com o erro no cálculo da fase e função seno e coseno. Além disso, pelo fato de que o algoritmo repete a estrutura básica de uma iteração que contém três somadores algébricos, foi testado o circuito com dois somadores diferentes. De igual forma, duas arquiteturas (paralela e *pipelined*) foram testadas e validadas.

4.1 Detecção do pacote

A detecção do pacote é baseada em uma operação de autocorrelação normalizada pela potência recebida, conforme descrito na seção 2.6.1. Quando o valor de $M(d)$ é maior que um

limiar th, considera-se um pacote detectado. Conforme mostrado em [23], [17] e [24], o resultado de $|R(d)|^2$ pode ser simplificado a $|R(d)_{Real}| + |R(d)_{Imag}|$. Com o intuito de economizar a operação de divisão, realiza-se uma comparação entre $|R(d)|^2$ com o valor de $0.5 \times P\,(d)$, conforme mostrado na equação (4.1). O valor de 0.5 proporciona um bom desempenho e permite implementar a multiplicação por $P(d)$ com uma operação de deslocamento [25]. Os circuitos necessários para a detecção do pacote são um registro de atraso, multiplicador complexo, acumulador, comparador e um registro de deslocamento de bit.

$$|R\,(d)|^2 > 0.5 \times P\,(d). \tag{4.1}$$

A sequência de entrada ao circuito de detecção do pacote é a saída de um circuito AGC. No entanto, para o isolamento e o estudo dos algoritmos de sincronismo de interesse neste trabalho, o AGC é ignorado e a entrada vem diretamente da saída do canal amostrado. A saída do circuito de detecção do pacote inclui um sinal de controle que indica que o pacote foi detectado, como também os valores da autocorrelação e da potência calculados. O diagrama de blocos para o circuito detector de pacotes é mostrado na Figura 4.1.

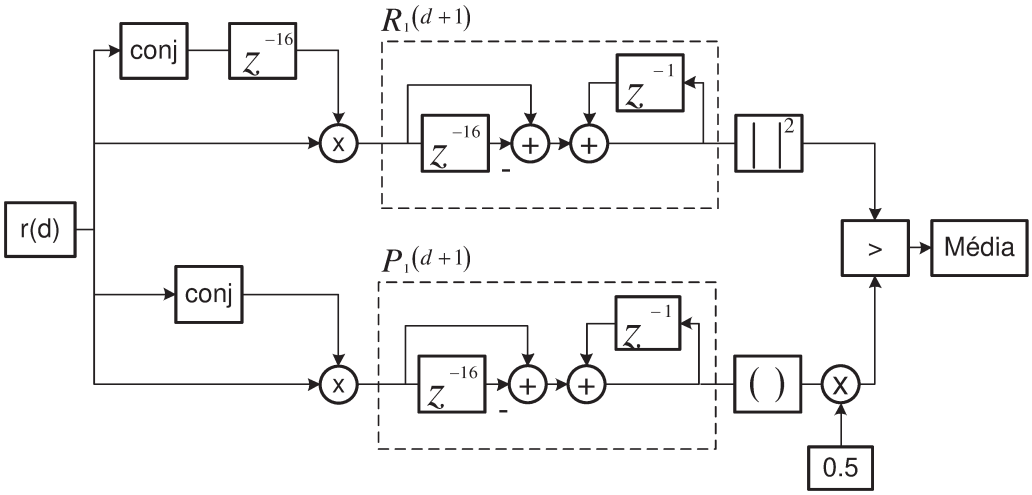


Figura 4.1: Diagrama de blocos do detector de pacotes.

Um FF de tipo D fornece armazenamento para um bit. Um registro é uma coleção de FFs de tipo D que são agrupados para armazenar vários bits. Um arquivo de registro é uma coleção de registros. Assim, o arquivo de registro que implementa o atrasador de 16 amostras é composto de L registradores em cascata, cada um com N bits de entrada e um caminho para o sinal real e outro para o sinal imaginário. A arquitetura deste bloco é mostrada na Figura 4.2. Na borda de subida do relógio, uma nova amostra é registrada e a amostra mais antiga é descartada. O circuito recebe os sinais digitais reais e imaginários das amostras OFDM e armazena as 16 amostras mais novas que são colocadas na saída, depois de 16 pulsos de relógio. O bloco multiplicador usará essas amostras atrasadas, juntamente com as novas amostras de entrada, para efetuar uma multiplicação complexa.

A multiplicação complexa usa quatro multiplicadores reais e dois somadores algébricos, conforme mostrado na Figura 4.3. Cada entrada possui comprimento de 12 bits e produz uma saída com resolução de 25 bits (24 bits resultantes da multiplicação mais 1 bit de *carry* resultante da soma binária).

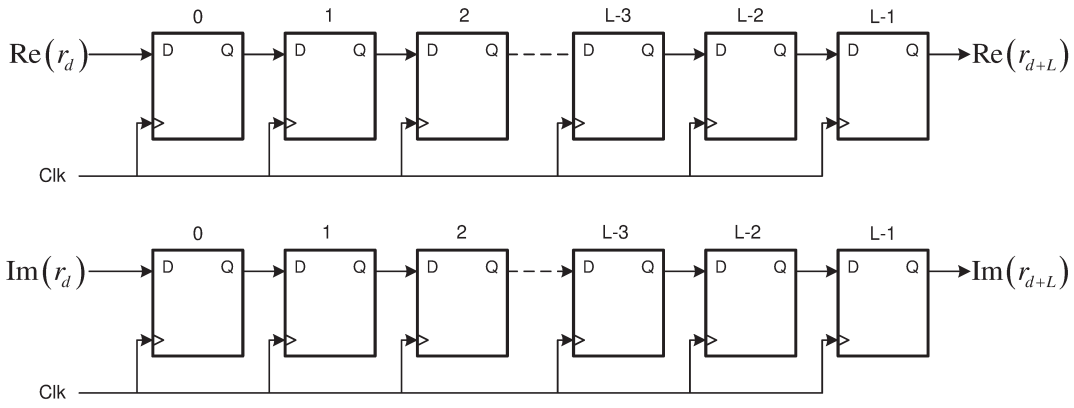


Figura 4.2: Registro de deslocamento: implementação do atrasador de 16 amostras.

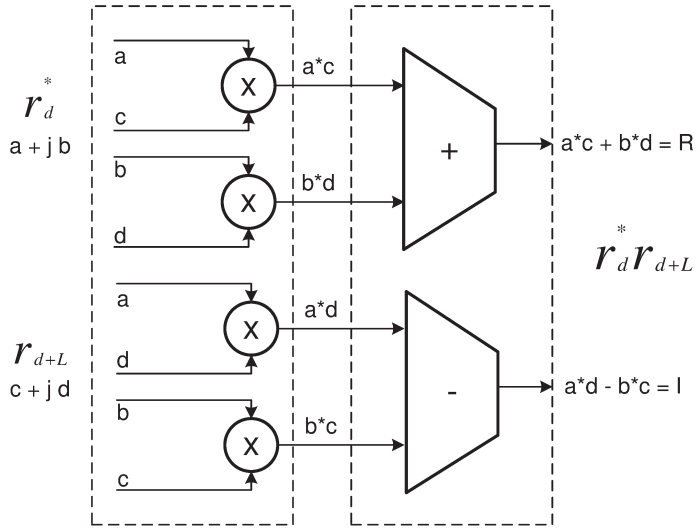


Figura 4.3: Multiplicador complexo.

Se o hardware disponível puder realizar três adições mais rápido do que o tempo de uma multiplicação, há uma maneira de acelerar a operação de multiplicação complexa [26]. A multiplicação de dois números complexos, $a + jb$ e $c + jd$, resulta no produto complexo mostrado por

$$R + jI = (a + jb)(c + jd) = (ac - bd) + j(bc + ad). \tag{4.2}$$

A equação (4.2) requer quatro multiplicações e duas adições. (Do ponto de vista computacional, pode-se assumir que uma subtração é equivalente a uma adição.) Em vez de usar a equação (4.2), pode-se calcular os seguintes valores intermediários:

$$\begin{aligned} k1 &= a(c + d) \\ k2 &= d(a + b) \\ k3 &= c(b - a) \end{aligned} \tag{4.3}$$

Em seguida, realizando as operações de (4.4), chega-se ao valor final de R e I.

$$\begin{aligned} R &= k1 - k2 \\ I &= k1 + k3 \end{aligned} \tag{4.4}$$

Os valores intermediários em (4.3) necessitaram de três adições e três multiplicações, enquanto que nos resultados nas equações (4.4) foram necessárias mais duas adições. Assim, troca-se uma das multiplicações necessárias na equação (4.2) por três operações de adição nas equações (4.3) e (4.4). Se o hardware utiliza menos ciclos de relógio para executar três adições do que uma única multiplicação, pode-se ganhar velocidade global de processamento usando-se as equações (4.3) e (4.4), em vez de equação (4.2) para uma multiplicação complexa.

Embora exista a economia de um multiplicador, o problema é que, do ponto de vista da latência, há uma multiplicação e dois somadores em série. A equação original tem apenas uma multiplicação e uma soma. Então, dependendo dos recursos do hardware e das restrições de latência, esta simplificação pode ser implementada.

Depois que a multiplicação complexa é realizada, a etapa seguinte é implementar a equação recursiva (2.33), conforme é mostrado na Figura 4.4. Em cada pulso de relógio um novo valor resultante da multiplicação, tanto da parte real como na imaginária, é ingressado ao registro de deslocamento, o qual realiza um atraso de 16 pulsos de relógio.

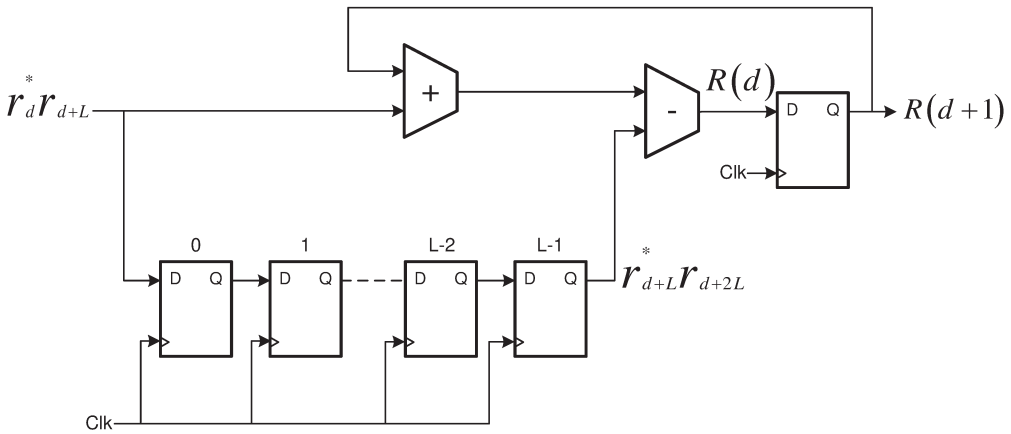


Figura 4.4: Arquitetura do bloco acumulador.

A operação de divisão requerida na equação (2.35) é evitada por meio da escolha de um limiar que seja potência de dois. O uso de um limiar de 0.5 permite que a divisão seja implementada usando-se uma operação de deslocamento de bit, o que resulta em uma economia de hardware [27], conforme mostrado na Figura 4.5.

Considerando-se o problema dos picos momentâneos verificados nos valores do cálculo por meio de (2.35) provocados por efeitos do canal, um circuito de média foi introduzido. Este circuito terá na saída um valor não nulo apenas se o sinal de detecção de pacote for não nulo para um determinado número M de ciclos de relógio. O valor usado na implementação foi $M = 32$, equivalente ao comprimento de dois símbolos curtos [24].

A Figura 4.7 mostra o resultado do circuito de autocorrelação e potência quando o preâmbulo está presente na entrada. Note-se que, após 160 amostras (10 símbolos STS), o sinal da

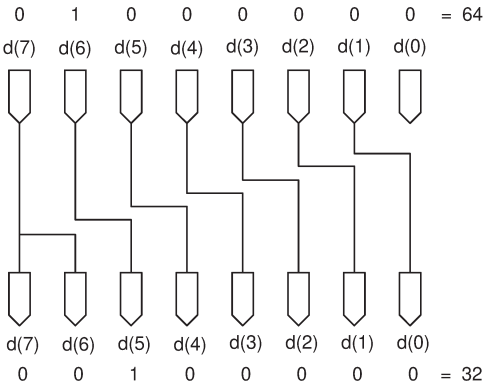


Figura 4.5: Deslocamento à direita que produz uma divisão por 2.

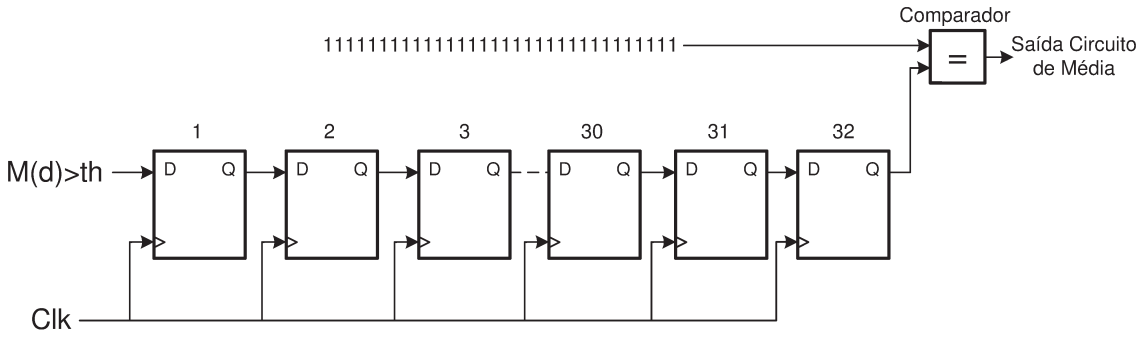


Figura 4.6: Arquitetura do circuito de média.

autocorrelação decai; no entanto, a potência do sinal é mantida, devido ao fato de que dos símbolos STS e LTS possuírem a mesma potência [14].

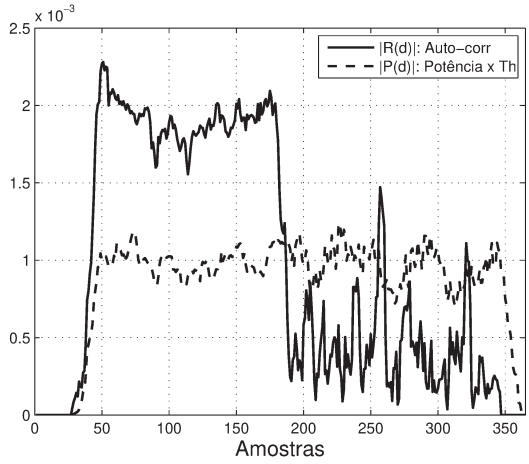
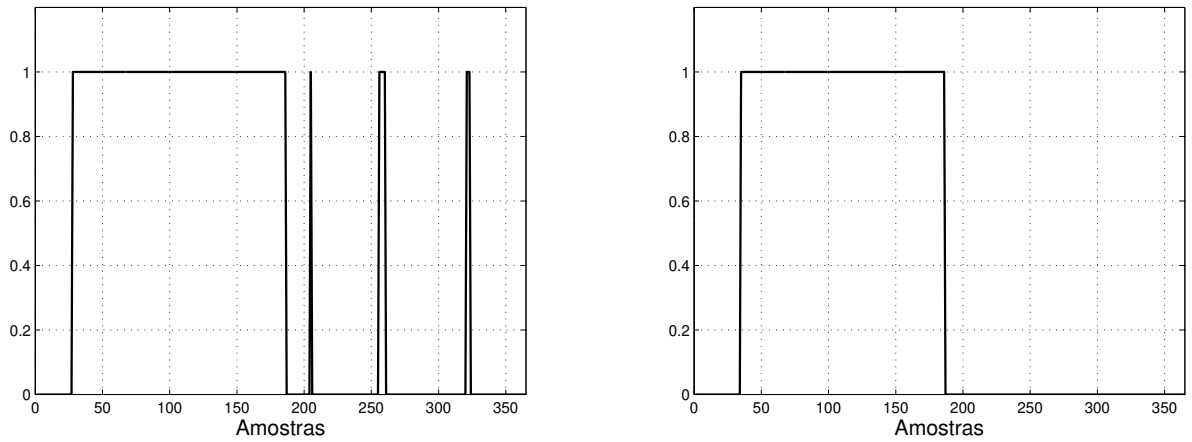


Figura 4.7: Autocorrelação e potência.

A Figura 4.8(a) mostra a saída do circuito quando a amplitude da autocorrelação é maior que a metade da potência recebida. Note-se também a presença de picos espúrios provocados pelo ruído. A Figura 4.8(b) mostra a saída do circuito de média. Este circuito só produz uma saída de valor 1 quando 32 amostras consecutivas da autocorrelação são maiores que a metade da potência. É possível observar a presença de um degrau entre as amostras 45 e 188, indicando a presença do STS.



(a) (b)

Figura 4.8: Métrica de comparação: (a) $M(d) > limiar$ (b) Circuito de média.

Com uma entrada de 12 bits, após a multiplicação complexa, obtém-se uma palavra binária de 25 bits. Na saída do acumulador (equação recursiva), obtém-se 29 bits devido ao fato de que são realizadas 16 somas nas quais é necessário aumentar 4 bits ($\lceil \log_2 (N) \rceil$) para se evitar o *overflow*. Finalmente, o cálculo do valor absoluto acrescenta mais um bit, gerando uma palavra binária com resolução de 30 bits.

Na Figura 4.9 são mostrados os sinais resultantes da simulação no *test bench*. O sinal de relógio (clk) foi estabelecido em 50 MHz [28].

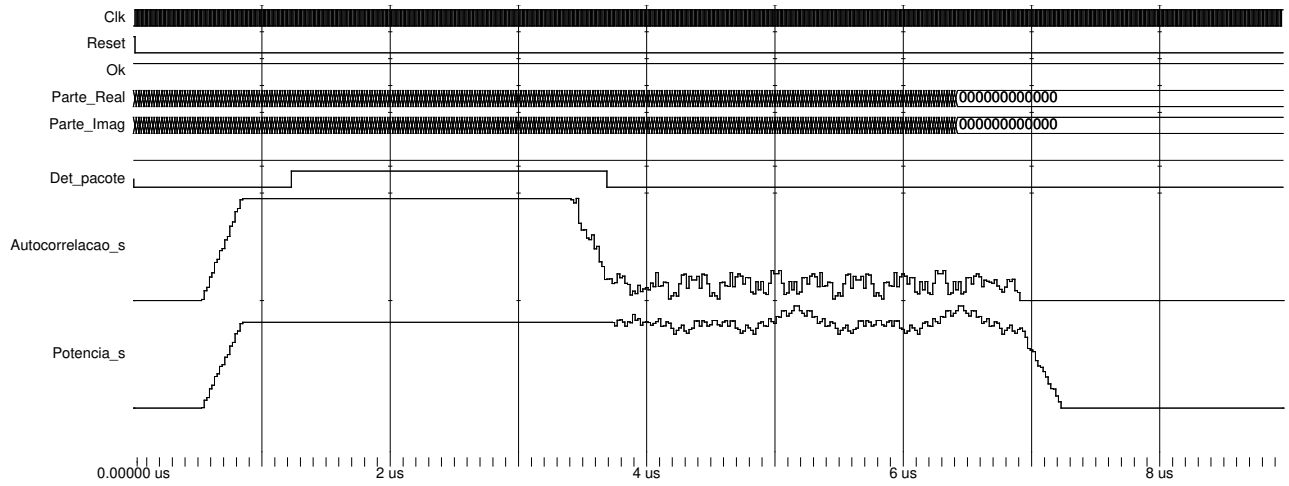


Figura 4.9: *Test bench* do detector do pacote.

O comprimento do STS e do LTS é de 160 amostras em cada um. Portanto, o número de amostras total do preâmbulo é de 320. Em cada sequência de entrada, houve o aumento de 128

amostras nulas com o intuito de obter uma melhor visualização do final do preâmbulo no *test bench*. Assim, em cada uma das figuras de *test bench* desta seção, a duração do preâmbulo vai até 6.4 μs, considerando um sinal de relógio de 20 ns. Da mesma forma, a função do sinal de *reset* em cada implementação é a de realizar a inicialização do sistema. Por exemplo, é gerado um pulso de *reset* de 10 ns para forçar o sistema para um estado inicial depois de se ligar a energia.

O resultado dos recursos da FPGA da implementação do algoritmo é apresentado na Tabela 4.1[1].

Tabela 4.1: Detecção do pacote com atrasador de 16 amostras.

Slice Logic Utilization	Used
Number of Slice Flip Flops	1176
Number of 4 input LUTs	1077
Number of Slices	820
MULT18X18SIOs	6
Maximum frequency	138.389MHz

Em [27] é apresentada uma alternativa ao detector do pacote, que consiste em mudar o valor do deslocamento do acumulador da Figura 4.1 de z^{-16} para z^{-144}. O resultado é um sinal em forma triangular (uma vez que em somente em um instante de tempo a janela de autocorrelação estará totalmente preenchida), no qual, usando a mesma normalização pela metade da potência, é determinada a presença do pacote. O resultado da autocorrelação pode ser utilizado para o sincronismo de tempo e para a estimação grosseira de frequência [27]. Na Figura 4.10 é apresentado o resultado dessa nova autocorrelação, com um sinal sem ruído nem desvio de frequência. Note-se que o valor do pico máximo está na amostra 173.

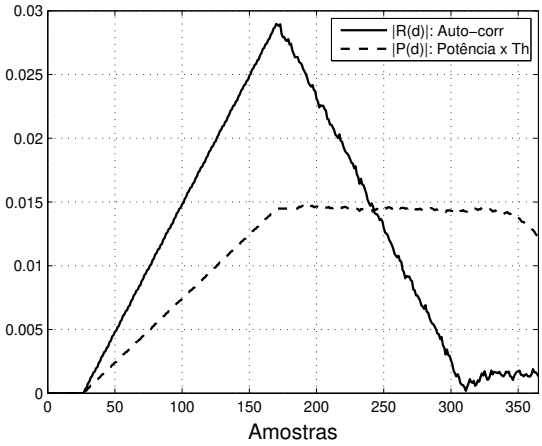


Figura 4.10: Autocorrelação e potência com atrasador de 144 amostras no acumulador.

O resultado dos recursos da FPGA da implementação do algoritmo é apresentado na Tabela 4.2. A frequência de operação diminui ligeiramente, enquanto os recursos da FPGA aumentam,

[1]Nas tabelas de uso de recursos da FPGA o texto foi mantido em inglês.

se comparados com a implementação anterior. No entanto, essa implementação serve também como circuito de sincronismo de tempo, adicionando na saída da autocorrelação um circuito detector de pico, que possui uma menor variância na detecção e justifica o uso de mais recursos. Este resultado será apresentado na seção seguinte.

Tabela 4.2: Detecção do pacote com atrasador de 144 amostras.

Slice Logic Utilization	Used
Number of Slice Flip Flops	1432
Number of 4 input LUTs	1677
Number of Slices	1299
MULT18X18SIOs	6
Maximum frequency	138.044MHz

4.2 Sincronismo de tempo

Esta seção apresenta a implementação de algoritmos de sincronismo de tempo que buscam encontrar a amostra de início do pacote. Para caracterizar cada algoritmo de sincronismo foi realizado o histograma dos resultados obtidos. O número de sinais de teste foi 1200, conforme foi descrito no capítulo 3. Além disso, são apresentados os recursos da FPGA usados, assim como a frequência de operação. As implementações são baseadas em operações de autocorrelação e de correlação cruzada.

4.2.1 Método de autocorrelação básica

Neste método, as entradas para o sincronizador são o resultado da autocorrelação e potência da Figura 4.1. Este detector inicia um contador quando o pacote é detectado, e para de contar no instante em que o sinal da comparação retorna a zero.

A Figura 4.11 mostra os sinais de saída quando na entrada está presente o preâmbulo. Na figura, o valor da variável *Contador* inicia quando o pacote é detectado e termina quando o nível da autocorrelação está por baixo do limiar de $0.5 \times P(d)$. Nesse instante, o sinal Td é ativado, indicando o limite do pacote.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.3. Nota-se que os recursos usados são quase iguais aos do circuito de detecção do pacote, acrescidos pela inclusão do circuito de contagem.

Em cada um dos testes, foi mantido o valor de SNR=10 dB (caso considerado crítico), mudando-se o desvio de frequência para cada um dos tipos de canais implementados. O histograma do algoritmo de autocorrelação básica é mostrado nas Figuras 4.12(a) e 4.12(b) e exibe o comportamento do circuito sincronizador, sob os canais modelados, indicando o valor da amostra em que a saída de autocorrelação cai abaixo do limiar $0.5 \times P(d)$.

Usando o resultado da autocorrelação da Figura 4.10, a posição do valor do pico máximo é determinada com o circuito apresentado na Figura 4.13. O circuito detector de pico devolve um índice que indica a posição da amostra em que ocorre um valor máximo. Esse circuito é composto de um contador, dois registros e um comparador.

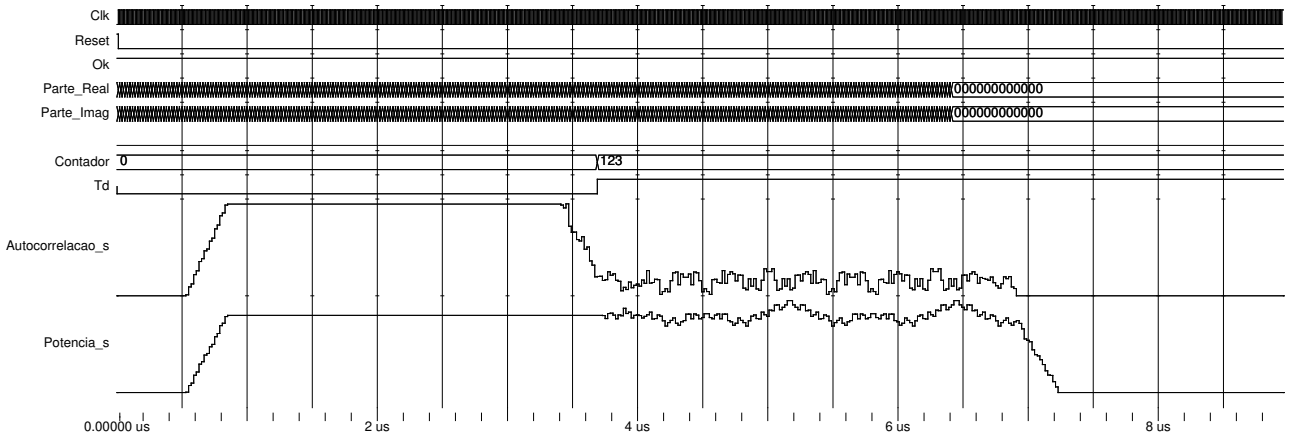
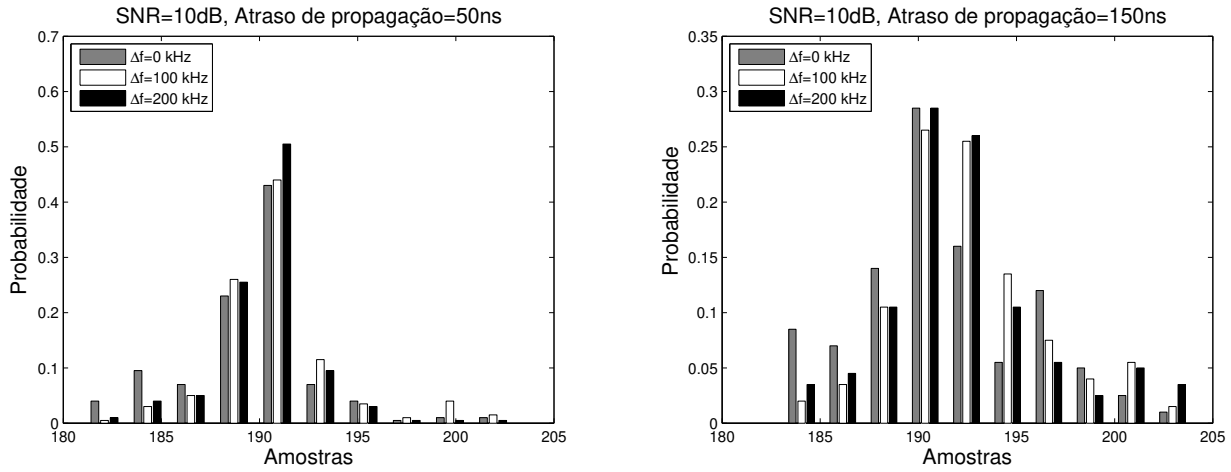


Figura 4.11: *Test bench* do algoritmo de autocorrelação básica.

Tabela 4.3: Autocorrelação básica com atrasador de 16 no acumulador.

Slice Logic Utilization	Used
Number of Slice Flip Flops	1186
Number of 4 input LUTs	1087
Number of Slices	825
MULT18X18SIOs	6
Maximum frequency	138.485MHz



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.12: Amostra em que a saída de autocorrelação cai por debaixo do limiar $0.5 \times P(d)$. Caso ideal $Td = 184$.

A Figura 4.14 mostra os sinais de saída no momento em que o preâmbulo está presente na entrada. Na figura, o sinal Td contém o valor da posição da amostra do valor máximo. Em condições ideais, o valor de Td é 173 amostras. No entanto, devido a efeitos do canal, esse valor muda para cada sinal de entrada.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.4.

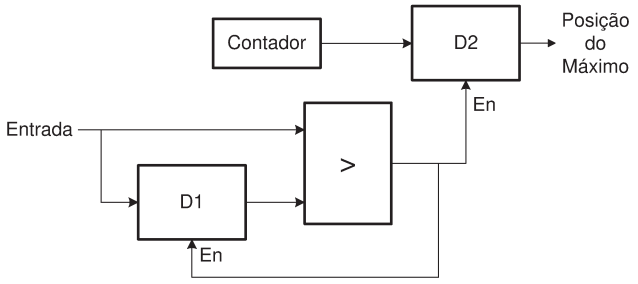


Figura 4.13: Detector de picos.

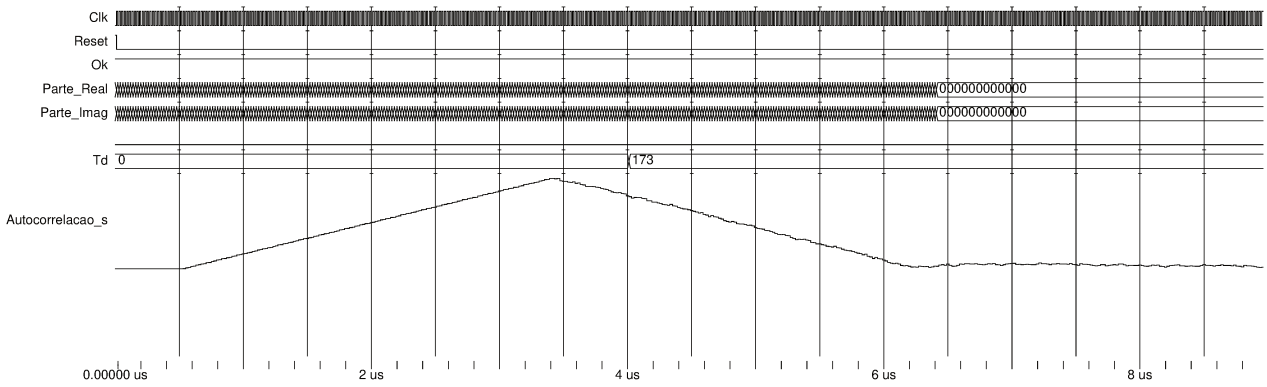


Figura 4.14: Saída do circuito de sincronização de tempo com atrasador de 144 amostras no acumulador.

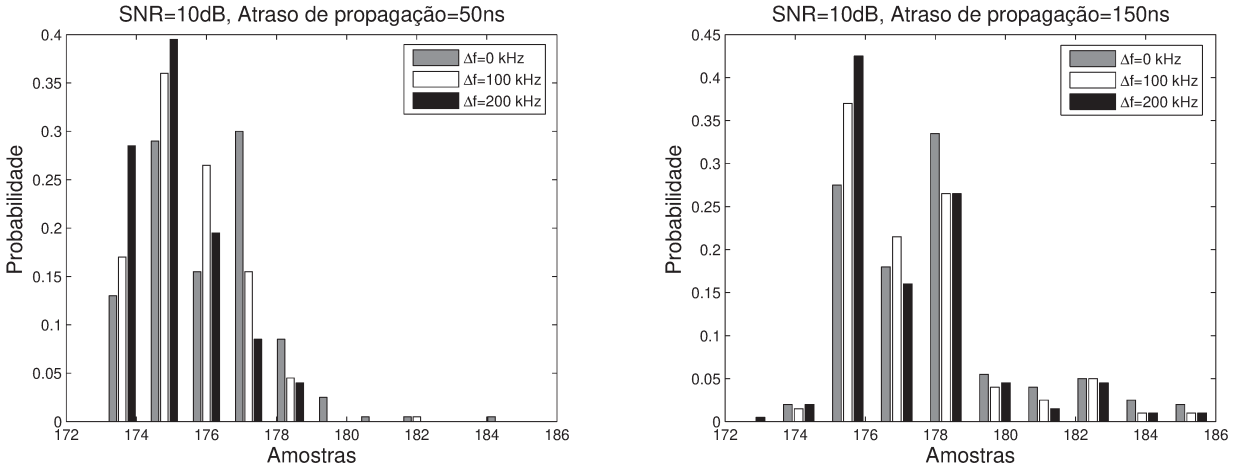
Tabela 4.4: Autocorrelação básica com atrasador de 144 no acumulador.

Slice Logic Utilization	Used
Number of Slice Flip Flops	1432
Number of 4 input LUTs	1677
Number of Slices	1299
MULT18X18SIOs	6
Maximum frequency	138.485MHz

O histograma do algoritmo é mostrado nas Figuras 4.15(a) e 4.15(b). Foi mantido um valor de SNR=10 dB, introduzindo-se vários desvios de frequência. Os gráficos apresentam o comportamento do circuito sincronizador, sob os canais modelados, indicando o valor da amostra no qual o pico do sinal ocorre.

4.2.2 Método de diferença de autocorrelação

Neste método, uma outra expressão de autocorrelação $R_2(d)$ é introduzida, sendo realizada com um atraso de 32 amostras e empregando uma arquitetura semelhante àquela exibida na Figura 4.1. Reutilizando-se o resultado de $R_1(d)$ e subtraindo do valor de $R_2(d)$, esse sinal resultante serve de entrada a um bloco de detecção de pico, conforme mostrado na Figura 4.16.



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.15: Amostra em que o máximo da autocorrelação acontece. Caso ideal $Td = 173$.

Esse valor é usado para determinar o início do pacote.

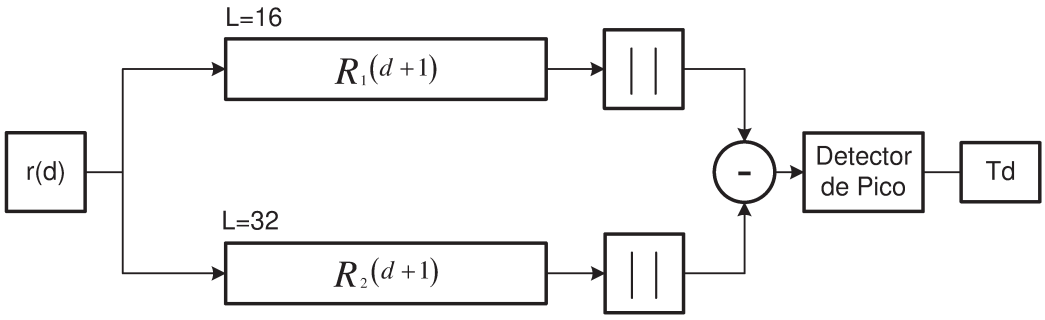


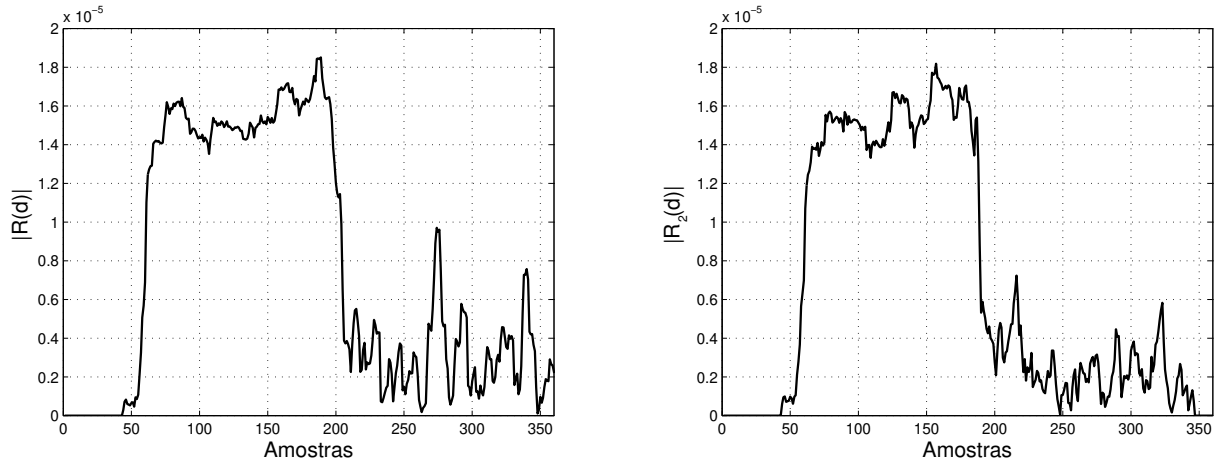
Figura 4.16: Algoritmo de diferença de autocorrelação.

A Figura 4.17(a) mostra o resultado da autocorrelação com um atraso de $L = 16$ amostras, enquanto a Figura 4.17(b) mostra o resultado da autocorrelação com um atraso de $L = 32$ amostras.

A Figura 4.18 mostra o resultado da diferença das autocorrelações. No sinal resultante é buscado o pico máximo, cuja posição determina o início do pacote.

Com $L = 16$ e uma entrada de 12 bits, após a multiplicação complexa é obtida uma palavra binária de 25 bits (24 bits da multiplicação mais 1 bits da soma). Na saída do acumulador (equação recursiva), obtém-se uma grandeza de 29 bits, devido ao fato de que são realizadas 16 somas onde é necessário aumentar 4 bits ($\lceil \log_2(N) \rceil$) para evitar o *overflow*. Finalmente, após o cálculo do valor absoluto, obtém-se uma palavra binária de 30 bits na saída. Quando $L = 32$, o resultado final tem um bit a mais, devido ao fato de que o número de somas é de 32, sendo necessários 5 bits para evitar o *overflow* na implementação da equação recursiva. Portanto, antes de se subtrair o valor das autocorrelações, ao valor de $R_1(d)$ é acrescentado um bit do sinal na posição do MSB.

A Figura 4.19 mostra os sinais de saída no momento em que o preâmbulo está presente na entrada. Na figura, os valores das variáveis *Autocorr_16* e *Autocorr_32* correspondem às



(a) (b)

Figura 4.17: Autocorrelação: (a) L=16 (b) L=32.

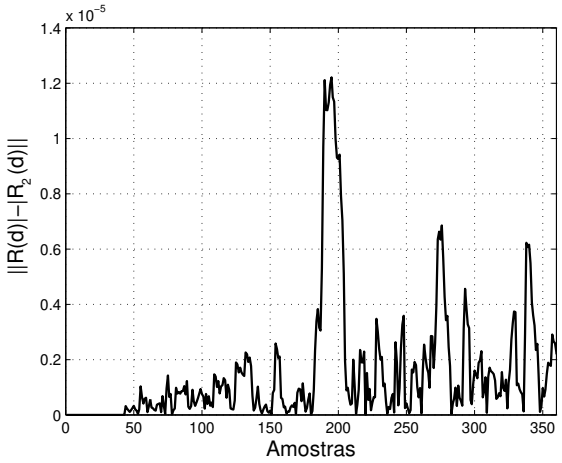


Figura 4.18: Diferença de autocorrelação.

duas autocorrelações. O detector de pico determina a posição do valor máximo do resultado da diferença das correlações, representado por Td no gráfico.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.5.

Tabela 4.5: Diferença de autocorrelação.

Slice Logic Utilization	Used
Number of Slice Flip Flops	1623
Number of 4 input LUTs	1706
Number of Slices	1160
MULT18X18SIOs	6
Maximum frequency	132.429MHz

O histograma do algoritmo de diferença de autocorrelação é mostrado na Figura 4.20. No-

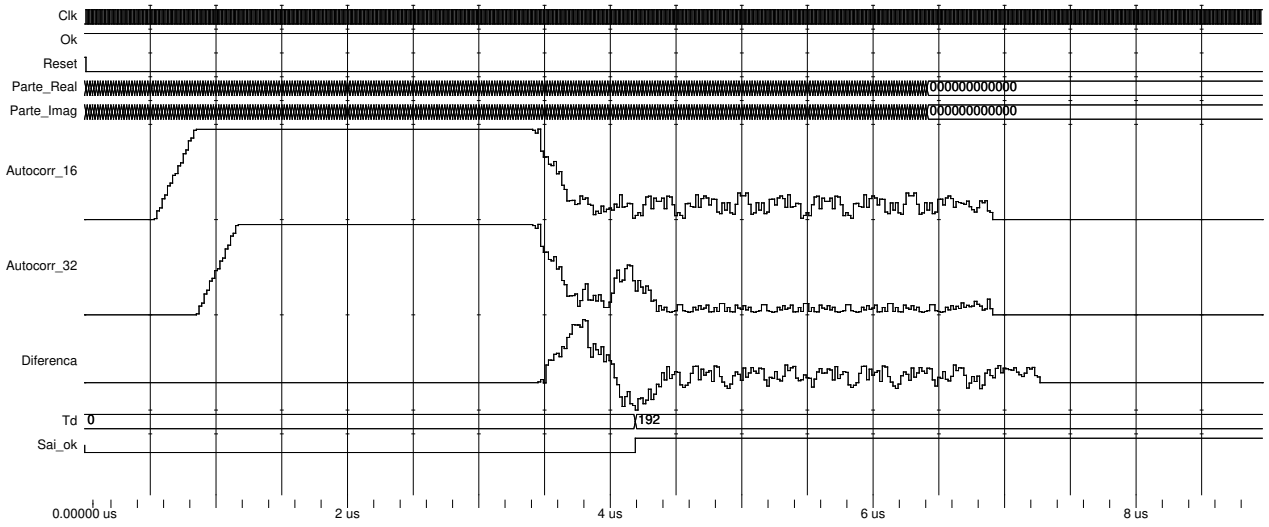
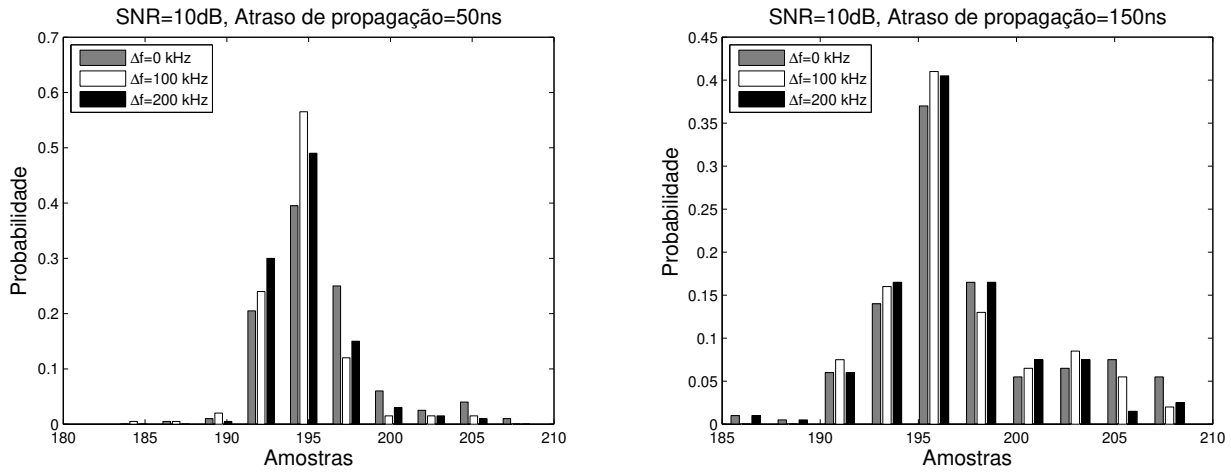


Figura 4.19: *Test bench* da diferença de autocorrelação.

vamente, foi mantido um valor de SNR=10 dB, introduzindo-se vários desvios de frequência. Os gráficos apresentam o comportamento do circuito sincronizador, sob os canais modelados, indicando o valor da amostra na qual o pico do sinal ocorre.



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.20: Amostra em que o máximo da diferença de correlação ocorre. Caso ideal $Td = 192$.

4.2.3 Correlação cruzada quantizada: uso de LTS

Nos sistemas WLAN, o preâmbulo é conhecido pelo receptor, o que permite o uso de uma correlação cruzada para estimar o atraso temporal. A correlação cruzada multiplica o sinal que chega pelo canal pelo símbolo original transmitido [3]. Os valores do símbolo original são armazenados em constantes dentro do programa.

Os elementos necessários para esta implementação são multiplicadores, registros de atraso e somadores. O resultado é um circuito com 64 tipos semelhantes de operações, conforme apresentado na Figura 4.21. Esse tipo de implementação é denominada filtro casado. Na

figura, r_d representa a amostra de entrada, c_n^* é o complexo conjugado da amostra do símbolo armazenado em memória e $\Lambda(d)$ é o resultado da equação (2.38).

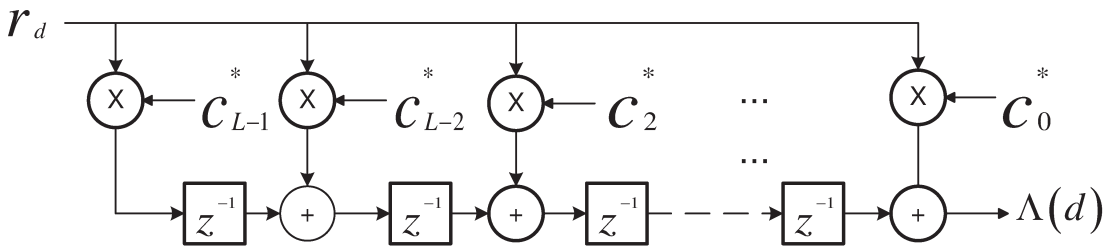


Figura 4.21: Correlação cruzada implementada com um filtro casado.

Visando economizar recursos multiplicadores, uma quantização é realizada com as amostras de entrada. Inicialmente, toma-se cada bit de sinal (MSB) para ser armazenado em um *buffer* de comprimento $L = 64$. Cada bit do *buffer* determina se o símbolo armazenado é somado ou subtraído. Por exemplo, em (4.5), o complexo conjugado do símbolo é multiplicado por um sinal cuja magnitude real e imaginária é um. O resultado será, na parte real, a soma das partes real e imaginária do símbolo armazenado; já a parte imaginária do resultado será a diferença das partes real e imaginária do símbolo armazenado. Portanto, considerando-se que o bit 0 é igual a 1 e o bit 1 é igual a -1, com as equações (4.5) é implementada uma correlação quantizada com um bit [18].

$$\begin{aligned}
(r_{real} + j\, r_{imag})^* \times (1 + j) &= (r_{real} + r_{imag}) + j\,(r_{real} - r_{imag}) \\
(r_{real} + j\, r_{imag})^* \times (1 - j) &= (r_{real} - r_{imag}) + j\,(-r_{real} - r_{imag}) \\
(r_{real} + j\, r_{imag})^* \times (-1 + j) &= (-r_{real} + r_{imag}) + j\,(r_{real} + r_{imag}) \\
(r_{real} + j\, r_{imag})^* \times (-1 - j) &= (-r_{real} - r_{imag}) + j\,(-r_{real} + r_{imag})
\end{aligned} \tag{4.5}$$

A saída do correlator é apresentada na Figura 4.22. Pode-se ver que o sinal resultante expõe claramente três picos: um ao final do intervalo de guarda de 32 amostras, outro ao final do primeiro LTS e um último pico ao final do segundo LTS.

Com uma entrada de 12 bits e um símbolo sem ruído de 16 bits armazenado em memória, na etapa de quantização é formado um vetor de 64 bits que contém os bits de sinal das amostras de entrada. Conforme foi mostrado, os símbolos armazenados em memória são somados ou subtraídos, gerando um sinal de 17 bits. Como a correlação cruzada tem uma janela de comprimento 64, obtém-se um resultado com 6 bits a mais. Em seguida, o valor absoluto desse resultado é calculado com uma soma binária, obtendo-se um resultado de correlação cruzada de 24 bits. Esse valor é ingressado ao bloco de detecção de pico, para determinar a amostra de início do pacote.

Na Figura 4.23 são mostrados os sinais resultantes da simulação.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.6.

O histograma do algoritmo de correlação cruzada é mostrado nas Figuras 4.24(a) e 4.24(b). Novamente foi mantido um valor de SNR=10 dB, introduzindo-se vários desvios de frequência.

A frequência da implementação anterior não cumpre os requisitos de tempo de processamento, que deve ser maior que 20 MHz. Sintetizando o código em uma FPGA de maior escala,

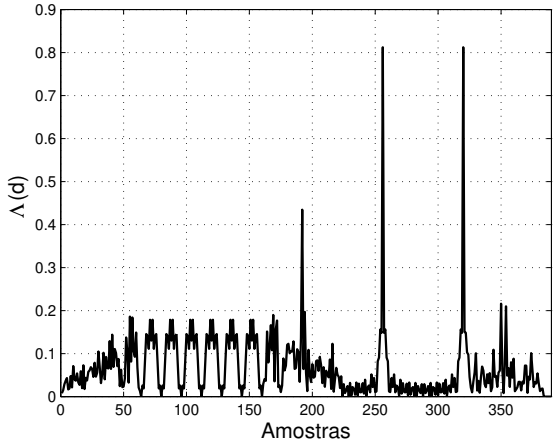


Figura 4.22: Correlação cruzada.

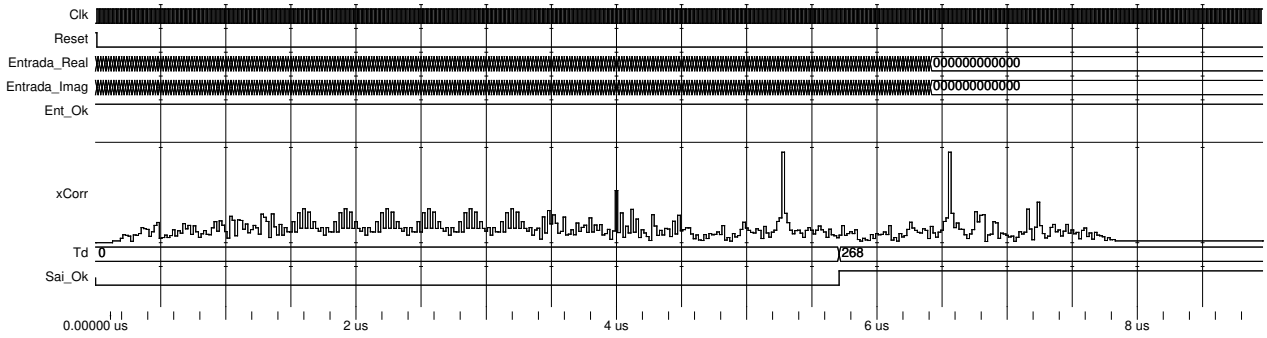


Figura 4.23: *Test bench* da correlação cruzada quantizada: LTS.

Tabela 4.6: Correlação cruzada quantizada: LTS.

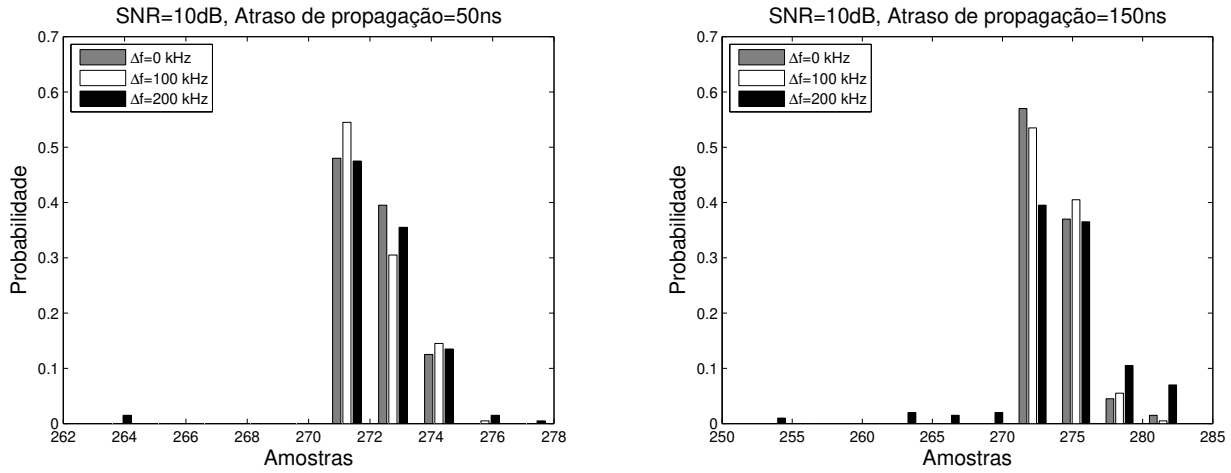
Slice Logic Utilization	Used
Number of Slice Flip Flops	778
Number of 4 input LUTs	3581
Number of Slices	2045
MULT18X18SIOs	0
Maximum frequency	13.059MHz

como a Virtex-4 FX ML405, a frequência de operação é de 21.448 MHz. No entanto, conforme é mostrado em [29] e [27], é possível diminuir os recursos da FPGA e aumentar a frequência de operação usando apenas 32 amostras do LTS. Na Figura 4.25, é exibido o resultado dessa nova correlação cruzada, pelo qual é fácil distinguir a presença de dois picos, cujas posições são usadas para determinar início do pacote. Em condições ideais, a posição do primeiro pico fica na metade do primeiro símbolo LTS.

Na Figura 4.26 são mostrados os sinais resultantes da simulação no *test bench*.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.7.

O histograma do algoritmo de correlação cruzada é mostrado nas Figuras 4.27(a) e 4.27(b). Novamente, foi mantido um valor de SNR=10 dB, introduzindo-se vários desvios de frequência.



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.24: Amostra em que o máximo absoluto da correlação cruzada ocorre. Caso ideal $Td = 268$.

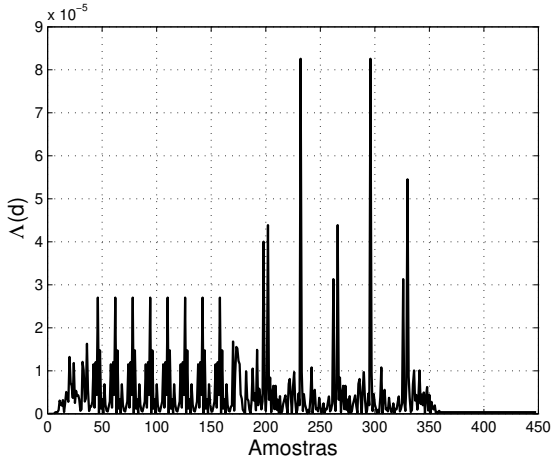


Figura 4.25: Correlação cruzada com 32 amostras do LTS.

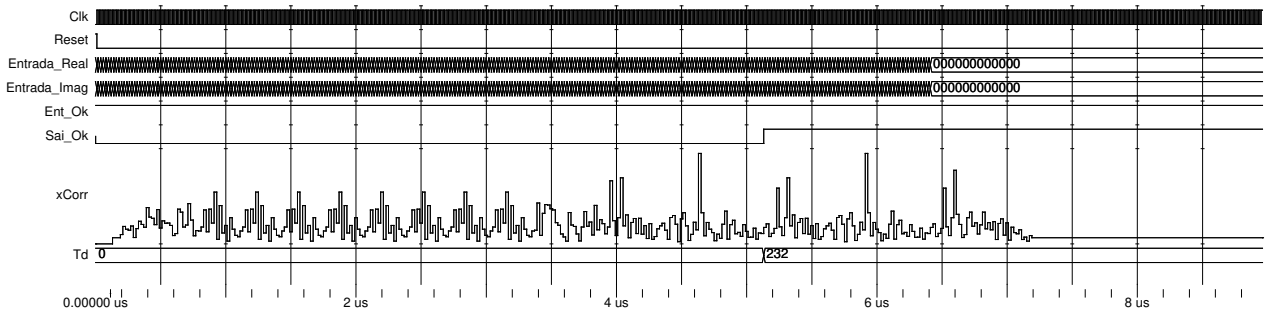
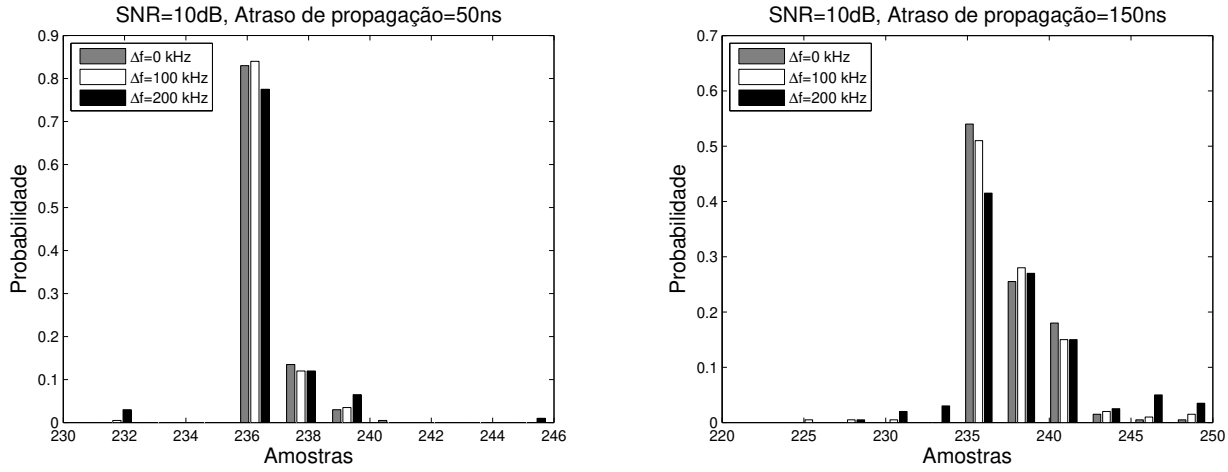


Figura 4.26: *Test bench* da correlação cruzada quantizada: LTS com 32 amostras.

Os gráficos apresentam o comportamento do circuito sincronizador, sob os canais modelados, indicando o valor da amostra na qual o pico do sinal ocorre.

Tabela 4.7: Correlação cruzada quantizada: LTS com 32 amostras.

Slice Logic Utilization	Used
Number of Slice Flip Flops	481
Number of 4 input LUTs	1935
Number of Slices	1109
MULT18X18SIOs	0
Maximum frequency	22.432MHz



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.27: Amostra em que o máximo absoluto da correlação cruzada ocorre. Caso ideal $Td = 232$.

4.2.4 Correlação cruzada quantizada: uso de STS

De forma semelhante à implementação proposta na seção anterior, é possível casar as amostras dos símbolos de entrada com as amostras dos símbolos STS sem ruído. O resultado é um sinal com vários picos de correlação localizados ao final de cada símbolo curto, conforme mostrado na Figura 4.28.

Com uma entrada de 12 bits e um símbolo sem ruído de 16 bits armazenado em memória, na etapa de quantização é formado um vetor de 16 bits que contém os bits de sinal das amostras de entrada. Conforme foi mostrado no método anterior, os símbolos armazenados em memória são somados ou subtraídos, dependendo do bit de sinal, gerando-se um sinal de 17 bits. Como a correlação cruzada tem uma janela de comprimento 16, um resultado com 4 bits a mais é obtido. Depois, é calculado o valor absoluto desse resultado com uma soma binária, obtendo-se um resultado de correlação cruzada de 22 bits. Esse vetor é ingressado ao bloco de detecção de pico, para determinar a amostra do limite do pacote.

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.8.

Na Figura 4.29 são mostrados os sinais resultantes da simulação.

O histograma do algoritmo de correlação cruzada com STS é mostrado na Figura 4.30. Novamente, foi mantido um valor de SNR=10 dB, introduzindo-se vários desvios de frequência.

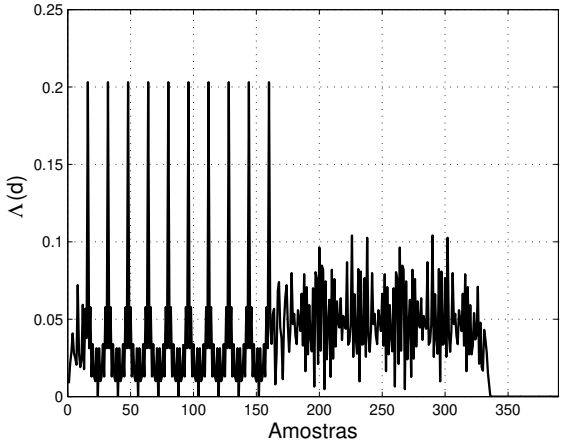


Figura 4.28: Correlação cruzada quantizada com STS.

Tabela 4.8: Correlação cruzada quantizada: STS.

Slice Logic Utilization	Used
Number of Slice Flip Flops	406
Number of 4 input LUTs	1284
Number of Slices	727
MULT18X18SIOs	0
Maximum frequency	37.397MHz

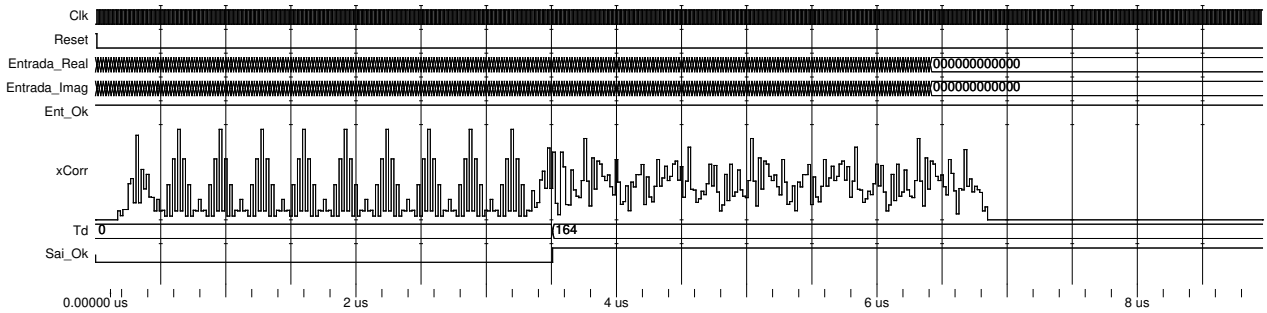


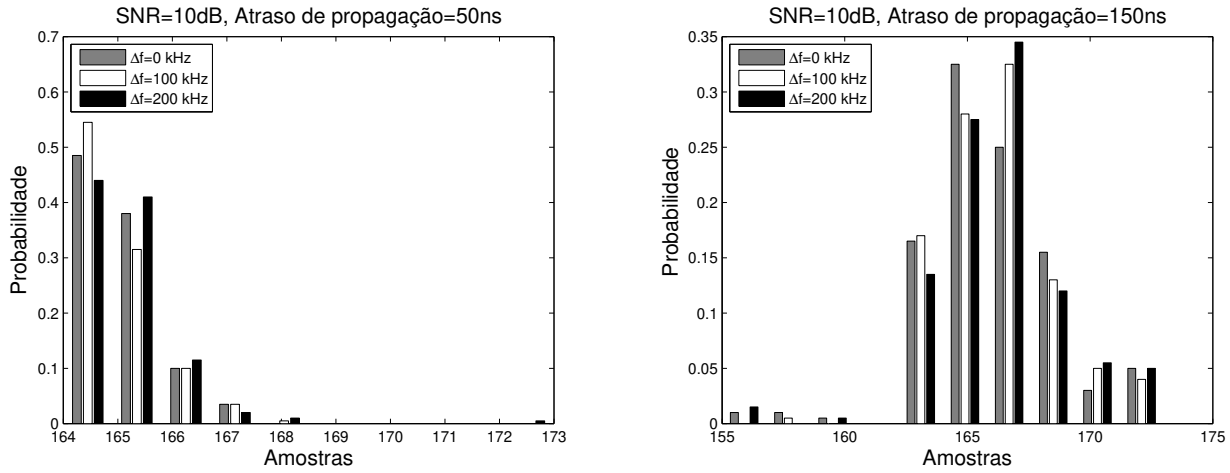
Figura 4.29: *Test bench* da correlação cruzada quantizada: STS.

4.2.5 Desempenho dos algoritmos de sincronismo de tempo

Na Figura 4.31 estão plotados os resultados das implementações com os algoritmos em questão, indicando a variância de seus estimadores temporais em diferentes níveis de SNR.

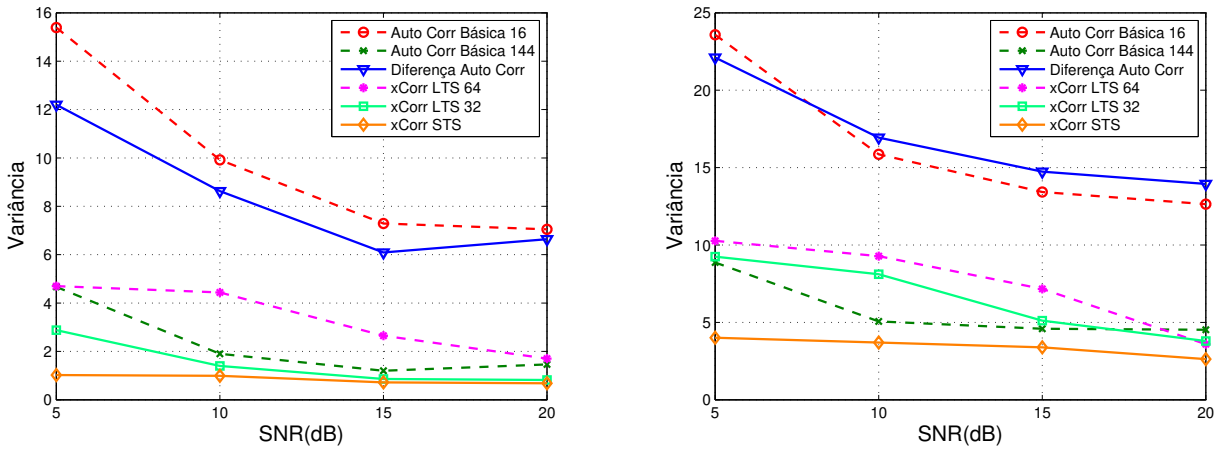
As Tabelas 4.9 e 4.10 apresentam o resumo de resultados para cada um dos algoritmos de sincronismo de tempo. Conforme descrito no início desta seção, os indicadores de comparação são a variância, o número de recursos da FPGA e a frequência de operação. Além disso, é apresentado o valor da média das estimativas de cada algoritmo na Tabela 4.11.

Os algoritmos de autocorrelação básica e de diferença de autocorrelação apresentam a maior variância e uma frequência de operação semelhante. No entanto, o algoritmo de autocorrelação básica possui um menor consumo de *slices*. O algoritmo de autocorrelação básica com atrasador



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.30: Amostra em que o máximo absoluto da correlação cruzada ocorre. Caso ideal $Td = 164$.



(a) 50 ns (b) 150 ns

Figura 4.31: Variância da estimativa temporal para diversos valores de SNR.

Tabela 4.9: Valores de variância da estimativa temporal dos diversos algoritmos de sincronismo.

SNR / **Algoritmo**	**5 dB**	**10 dB**	**15 dB**	**20 dB**
Autocorrelação básica 16	18,36	13,80	11,47	10,97
Autocorrelação básica 144	8,09	4,40	3,78	3,74
Diferença de autocorrelação	15,3	13,99	11,67	11,27
Correlação cruzada com LTS (64)	9,04	7,28	5,24	3,74
Correlação cruzada com LTS (32)	6,49	5,20	3,33	2,79
Correlação cruzada com STS	3,63	2,60	2,32	2,02

Tabela 4.10: Slices e frequência de operação.

Algoritmo	**Freq. (MHz)**	**Slice**
Autocorrelação básica 16	138,49	825
Autocorrelação básica 144	138,49	1299
Diferença de autocorrelação	132,43	1160
Correlação cruzada com LTS (64)	13,06	2045
Correlação cruzada com LTS (32)	22,43	1109
Correlação cruzada com STS	37,40	727

Tabela 4.11: Resumo dos algoritmos de sincronização de tempo: média.

SNR / **Algoritmo**	**5 dB**	**10 dB**	**15 dB**	**20 dB**
Autocorrelação básica 16	189,43	191,19	191,67	191,82
Autocorrelação básica 144	176,85	176,60	176,45	176,54
Diferença de autocorrelação	195,74	196,20	196,27	196,36
Correlação cruzada com LTS (64)	273,18	273,17	273,26	273,23
Correlação cruzada com LTS (32)	237,46	237,45	237,40	237,28
Correlação cruzada com STS	169,24	169,26	169,30	169,24

de 144 amostras possui uma variância menor mantendo a frequência de operação, porém usa mais recursos de *slices*. Já entre os algoritmos baseados em correlação cruzada com os LTS, o melhor resultado em cada um dos parâmetros foi aquele que usava 32 amostras do símbolo. A correlação cruzada com os símbolos STS possui a menor variância, a maior frequência de operação e um menor consumo de *slices* se comparada com os algoritmos que usam a correlação cruzada com os LTS.

4.3 Sincronismo de frequência

O desvio de frequência é causado pela diferença existente entre a frequência do oscilador do transmissor e a do receptor, bem como pelo efeito Doppler. Essa diferença na frequência pode destruir a ortogonalidade entre as subportadoras e causar ICI. O sincronismo em frequência tem como objetivo encontrar esse desvio de frequência e corrigi-lo.

A arquitetura do circuito de estimação e compensação é apresentada na Figura 4.32. A função do bloco CMA inclui o cálculo da função exponencial, conjugado, a multiplicação e a acumulação. O bloco CMV é o CORDIC em modo vetorial e o CMR é o CORDIC em modo rotacional.

O desvio de frequência da portadora é estimado da seguinte forma: realiza-se uma autocorrelação com os símbolos STS e calcula-se a fase desta autocorrelação (estimativa grosseira). Com essa fase, são compensados os símbolos LTS. Em seguida, uma nova autocorrelação dos símbolos LTS é realizada. A fase desta autocorrelação (estimativa fina) é calculada e somada à primeira estimativa de fase.

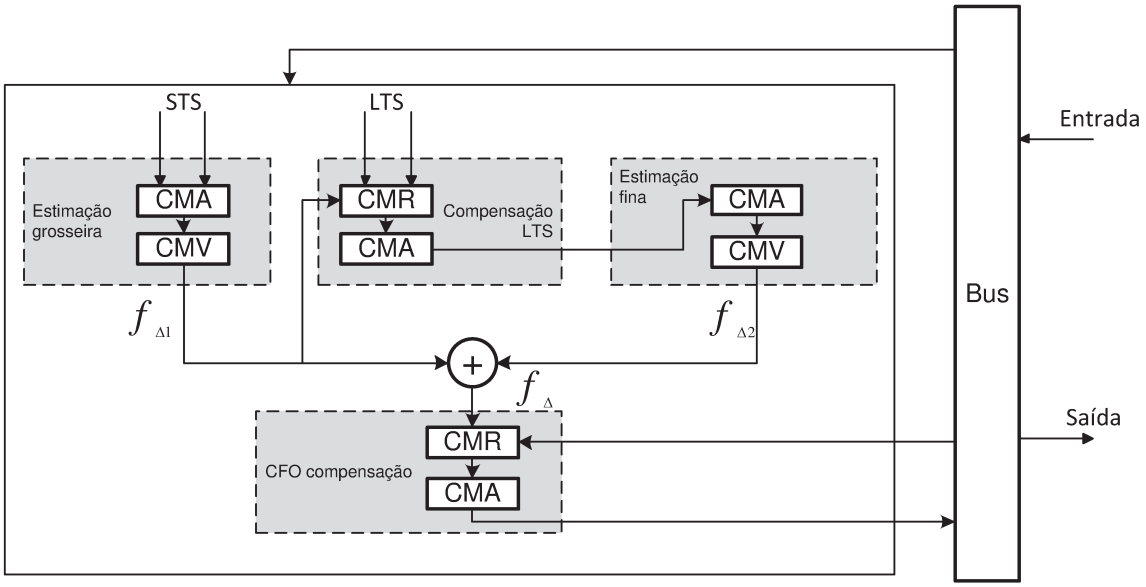


Figura 4.32: Arquitetura do circuito de compensação e estimação de CFO.

Os requisitos de hardware para esta parte do sincronizador reutilizam o autocorrelator da detecção de pacote (Figura 4.1). A saída da autocorrelação entra no bloco de cômputo de fase, que é baseado no algoritmo CORDIC em modo vetorial. A compensação é realizada com o CORDIC em modo rotacional. Na seção seguinte, apresenta-se a implementação do algoritmo CORDIC empregada nesta parte do sincronismo.

4.3.1 Implementação do CORDIC

CORDIC é um algoritmo iterativo que possui os mesmos componentes em cada etapa de pseudo-rotação: três somadores algébricos, duas unidades de deslocamento, um negador e uma LUT que armazena o valor do ângulo de rotação de cada iteração.

O CORDIC funciona para ângulos entre $-\pi/2$ e $\pi/2$ devido ao fato de que a primeira rotação é $\mathrm{tg}^{-1}(2^0)$. Para o aumento da faixa de operação do algoritmo CORDIC, é necessária uma rotação inicial do vetor, de forma a colocá-lo nos quadrantes I e IV. O diagrama da unidade de rotação é apresentado na Figura 4.33. São necessários três somadores algébricos e um negador. Na implementação, o somador da fase inicial de $+90^o$ é trocado pelos sinais de -90^o e $+90^o$ já armazenados em memória cuja saída depende do bit mais significativo (bit de sinal) do componente y_i. Para processar os sinais x_i e y_i, o projeto possui uma unidade para calcular o complemento de 2 que produz os sinais de x_{i+1} e y_{i+1}.

O deslocamento de um vetor binário à direita produz a divisão por dois. Para cada iteração é preciso uma unidade de deslocamento distinta; portanto, tem-se uma unidade de deslocamento conectada por trilhas. A extensão do bit de sinal é realizada atribuindo o bit mais significativo aos bits mais significativos do resultado, conforme o número de deslocamentos. A Figura 4.34 ilustra a unidade de deslocamento para o caso de um e três deslocamentos com um vetor de 8 bits.

A unidade de pseudo-rotação (também chamada de micro-rotação) tem a finalidade de girar o vetor, em cada uma das iterações, em um valor determinado por α_i. A Tabela 4.12 contém os

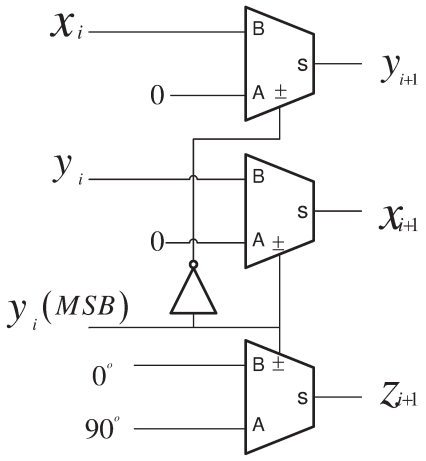
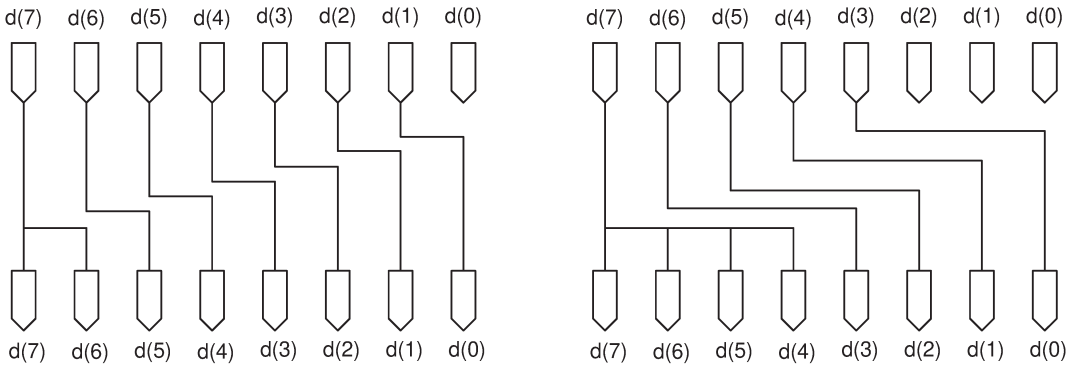


Figura 4.33: Diagrama funcional da unidade de rotação de $\pm 90^o$.



(a) (b)

Figura 4.34: Ilustração da unidade de deslocamento: (a) para 1 deslocamento, (b) para 3 deslocamentos sobre um operador de 8 bits.

ângulos $\alpha_i = \text{tg}^{-1}(2^{-i})$ que são pré-armazenados na FPGA [30]. Essa unidade é composta pelos seguintes blocos: duas unidades de deslocamento, três somadores algébricos e um negador, conforme mostrado na Figura 4.35. Para determinar se a operação é de soma ou subtração, considera-se o bit mais significativo do sinal y (o bit de sinal).

Dependendo do modo de realizar cada uma das iterações do CORDIC, há duas classes de processadores: paralelo e *pipelined*. No processador paralelo, todas as operações de rotação são realizadas em apenas um ciclo de relógio. Cada elemento de processamento é dedicado a uma iteração. Assim, duas simplificações importantes podem ser aplicadas à implementação: o deslocamento de cada iteração é fixo, sendo implementado diretamente nas conexões entre os elementos e os valores do deslocamento do ângulo, sendo distribuídos como constantes para cada iteração. A Figura 4.36 detalha esta arquitetura [31].

Para a avaliação da arquitetura proposta, as entradas foram dez vetores com ângulos igualmente espaçados em 30 graus (colocando vetores nos quatro quadrantes). O código foi testado variando-se o valor das iterações, de 4 a 16. Os valores resultantes da simulação foram com-

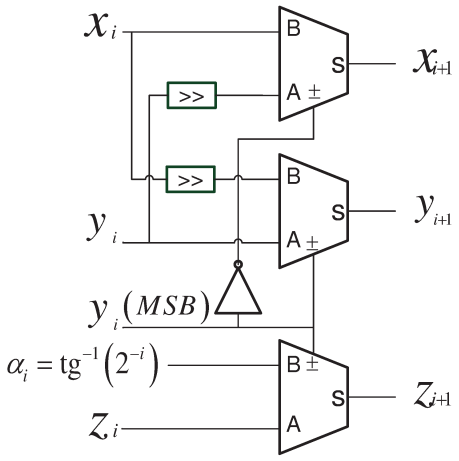


Figura 4.35: Diagrama funcional da unidade de pseudo-rotação.

Tabela 4.12: Valores de $\alpha_i = \mathrm{tg}^{-1}(2^{-i})$ e sua representação em formato binário U(0,16).

Iteração	**Valor decimal (rad)**	**Ponto fixo**
0	0.785400390625000	1100100100010000
1	0.463653564453125	0111011010110010
2	0.244979858398438	0011111010110111
3	0.124359130859375	0001111111010110
4	0.062423706054688	0000111111111011
5	0.031234741210938	0000011111111111
6	0.015625000000000	0000010000000000
7	0.007812500000000	0000001000000000
8	0.003906250000000	0000000100000000
9	0.001953125000000	0000000010000000
10	0.000976562500000	0000000001000000
11	0.000488281250000	0000000000100000
12	0.000244140625000	0000000000010000
13	0.000122070312500	0000000000001000
14	0.000061035156250	0000000000000100
15	0.000030517578125	0000000000000010

parados com os valores obtidos em MATLAB. Para o cálculo do erro absoluto empregou-se a seguinte equação:

$$Erro = |Cordic_{Matlab} - Cordic_{Vhdl}| . \tag{4.6}$$

O resultado do erro absoluto é apresentado na Figura 4.37.

Uma vantagem dos processadores paralelos é que entre cada iteração podem ser inseridos registros de *pipelining*, o que aumenta a frequência de operação da FPGA [32]. Em muitas FPGAs há registradores em cada célula lógica fazendo com que o *pipelining* não consuma recursos adicionais. A Figura 4.38 mostra um processador CORDIC *pipelined*.

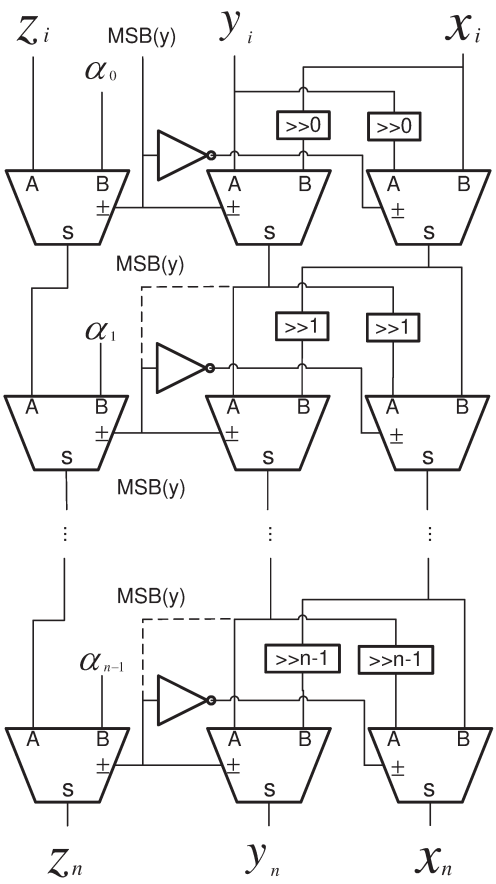
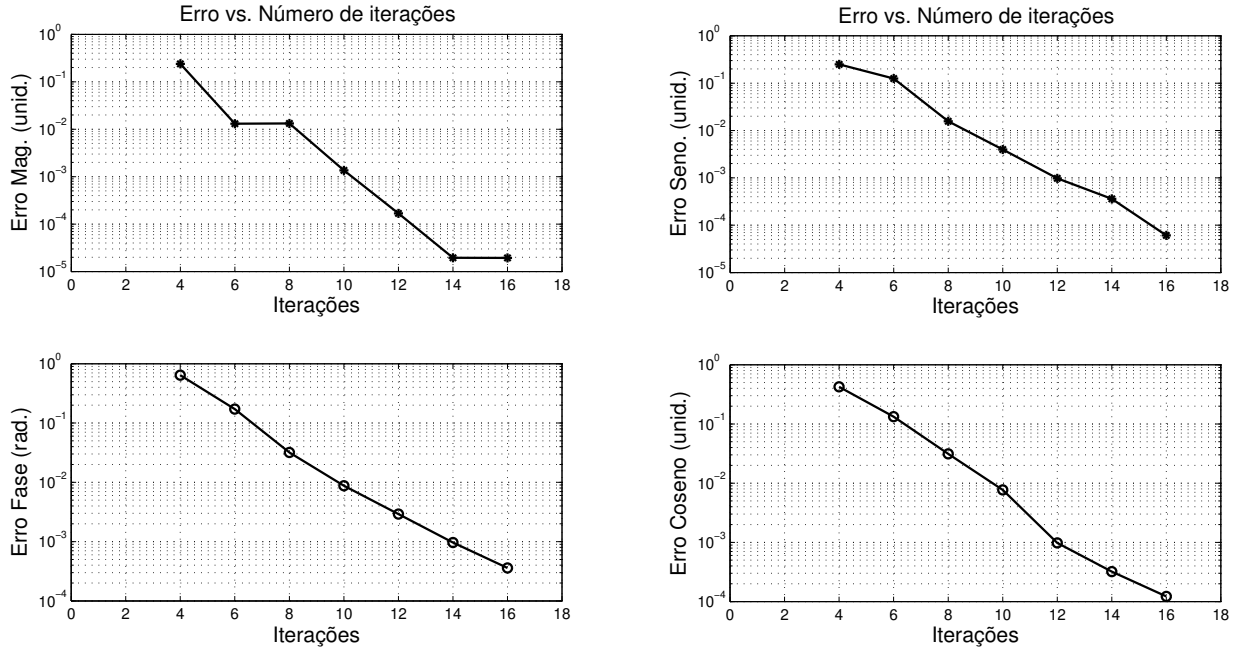


Figura 4.36: Processador paralelo CORDIC.

Devido ao fato de que o CORDIC precisa executar uma soma binária várias vezes ao longo do processamento, implementaram-se dois tipos de somadores com objetivo de se quantificar o efeito no tempo de processamento e consumo de recursos em FPGA. Com uma combinação de somadores completos (*full-adder*) é realizada a soma de dois números binários de n bits [33]. Enquanto o somador *Ripple-Carry Adders* (RCA) possui o projeto mais compacto entre todos os tipos de somadores ($O(n)$ área), ele é o mais lento dos tipos de somadores ($O(n)$ tempo). Por outro lado, o somador *Carry Look-ahead Adder* (CLA) é o mais rápido ($O(\log(n))$ tempo), mas é também o pior do ponto de vista da área ($O(n \log(n))$ área). O somador *Carry-Select Adder* (CSA) é considerado uma solução de compromisso entre RCA e CLA, porque oferece um bom equilíbrio entre a área compacta de RCA e o pequeno atraso do CLA ($O(\sqrt{n})$ tempo e $O(2n)$ área). Portanto, a arquitetura escolhida foi CSA [32]. A Figura 4.39 apresenta essa arquitetura.

As operações de adição e subtração podem ser combinadas em um único circuito incluindo uma porta OU exclusivo com cada somador completo. A entrada de M da Figura 4.39 controla a operação. Quando $M = 0$ o circuito é um somador, e quando $M = 1$ o circuito se torna um subtrator. Cada porta OU exclusivo recebe na entrada o valor de M e o valor de Y. Quando $M = 0$, temos $Y \oplus 0 = Y$. Os somadores completos recebem o valor de Y, o *carry* de entrada é 0 e o circuito realiza a operação X mais Y. Quando $M = 1$, temos $Y \oplus 1 = Y'$ e o *carry* de entrada é 1. A entrada Y é complementada e um valor de 1 é adicionado através do *carry* de



(a) (b)

Figura 4.37: Relação do erro absoluto com o número de iterações: (a) Magnitude e Fase (b) Seno e Coseno.

entrada. O circuito realiza a operação X mais o complemento de 2 de Y, ou seja, uma subtração binária. A Figura 4.40 apresenta os resultados da implementação das arquiteturas paralela e *pipelined* e os somadores CSA e RCA.

Pode-se ver na Figura 4.40 que a arquitetura *pipelined* possui um menor tempo de processamento, que é aproximadamente 12 vezes menor, indistintamente do somador projetado. O uso do somador CSA diminui o tempo de operação em 15.3% para 16 iterações no caso da arquitetura paralela e em 20.17% na arquitetura *pipelined*. No entanto, devido ao fato de que o somador CSA usa 50% mais somadores que o somador RCA, o primeiro emprega um número 7.7% maior de *slices* do que o segundo em uma arquitetura paralela, e 20.9% maior no caso de arquitetura *pipelined* com 16 iterações e 16 bits de palavra binária de valor de entrada.

A relação lógica/velocidade vai determinar o enfoque a ser escolhido para projetar o algoritmo CORDIC em uma FPGA. A inserção de registros entre cada iteração faz com que o algoritmo compute o resultado mais rapidamente; porém, o valor ficará pronto depois de N ciclos de relógio após o ingresso do primeiro dado, enquanto a arquitetura paralela terá pronto o resultado após um ciclo de relógio. O número de *slices* empregados em cada arquitetura não muda da mesma forma do que o tempo de processamento, devido ao fato de que cada *slice* tem registros que podem ser usados para projetar uma arquitetura *pipelined*. Portanto, a arquitetura escolhida para o CORDIC na etapa de sincronismo de frequência possui uma arquitetura *pipelined* com o somador CSA, usando 16 iterações.

4.3.2 Estimação e compensação usando CORDIC

O sinal complexo resultante da autocorrelação possui uma fase proporcional ao desvio de frequência da portadora, conforme (2.40). Essa frequência serve como primeira estimativa e é

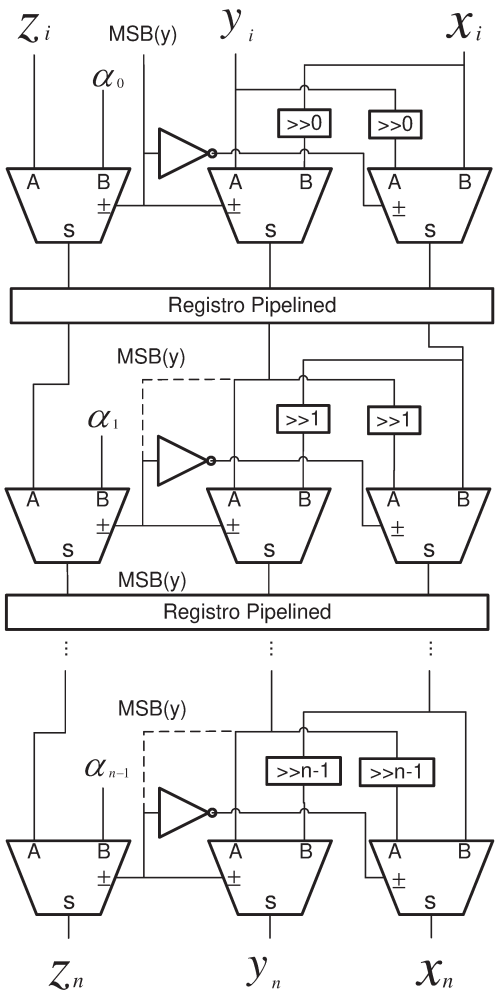


Figura 4.38: Processador CORDIC *pipelined*.

uma estimativa grosseira. Logo, com a finalidade de compensar o desvio de fase, os símbolos LTS são multiplicados por uma exponencial complexa com fase inversa à estimada com os símbolos STS. Essa operação é realizada com o CORDIC em modo rotacional. Mais uma autocorrelação com os símbolos LTS é realizada. Novamente é calculada a fase com o CORDIC em modo vetorial. Essa estimativa é denominada estimativa fina.

A compensação é realizada com o CORDIC em modo rotacional, tendo como base um circuito de Síntese Digital Direta (DDS). DDS é um método que gera formas de onda diretamente no domínio digital. O DDS é composto por um acumulador de fase e um conversor fase-amplitude [34], conforme mostrado na Figura 4.41. Neste projeto, o objetivo é gerar as funções seno e coseno necessárias para girar um vetor.

Por um lado, em um DDS convencional baseado em LUT, o acumulador de fase é um acumulador inteiro de N bits cuja saída indexa as direções da LUT diretamente onde os valores de amplitude das funções seno e coseno são armazenados. O valor máximo do acumulador $(2^N - 1)$ representa a fase 2π da função seno ou coseno. O acumulador gera um sinal rampa que é incrementado por um valor fixo. Assim, um sinal periódico é obtido na saída do conversor fase-amplitude, conforme mostrado na Figura 4.42.

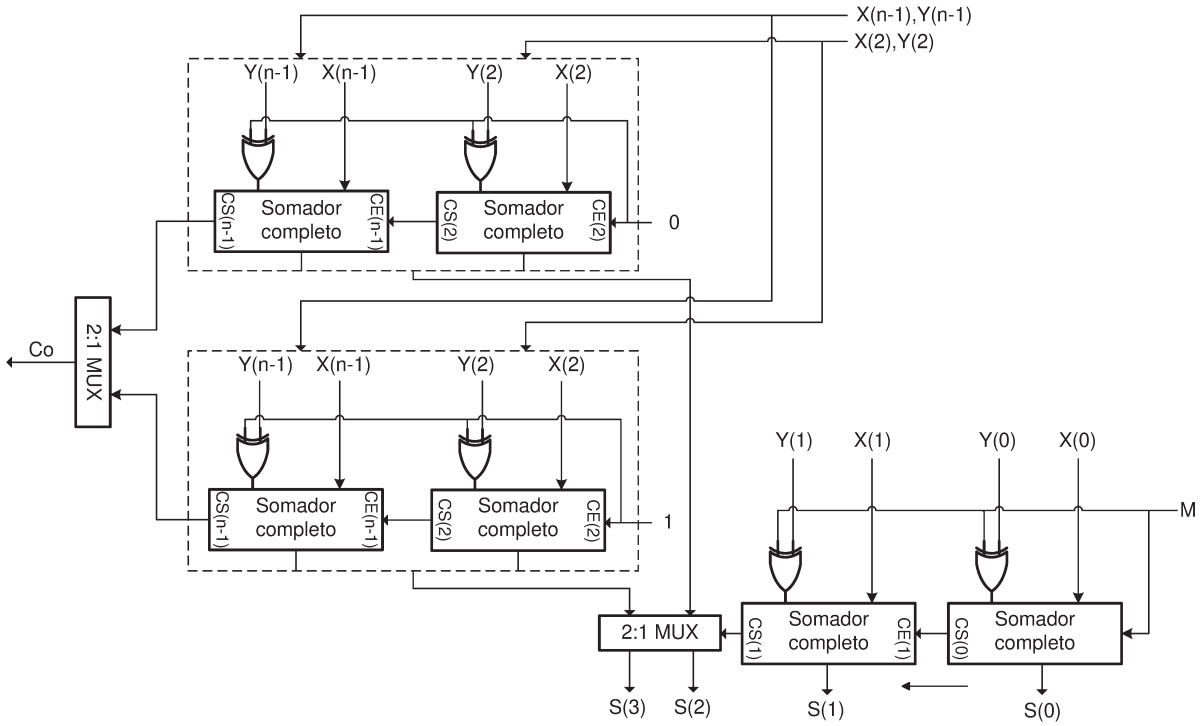
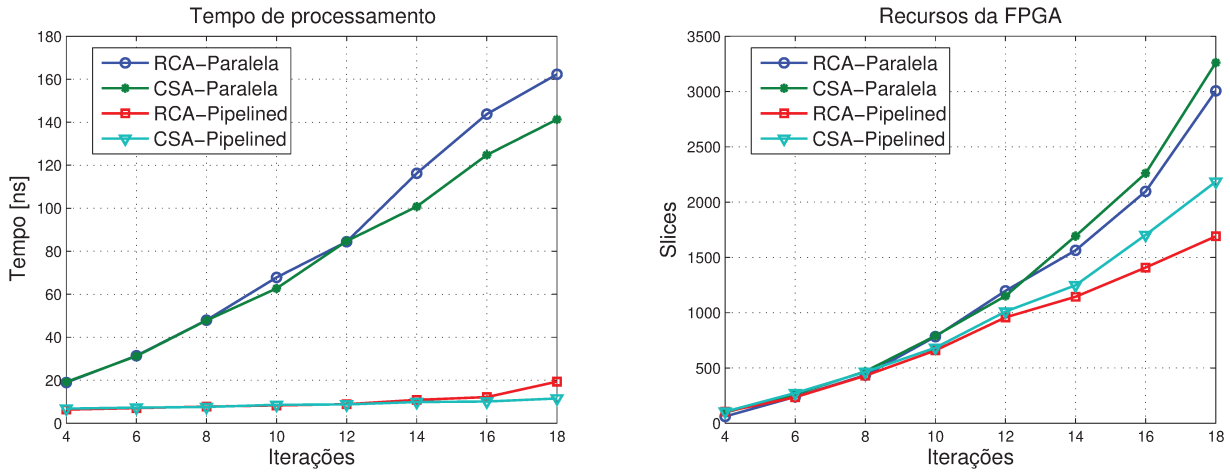


Figura 4.39: Arquitetura do somador *carry select adder*.



(a) (b)

Figura 4.40: Relação de iterações, tempo de processamento e recursos da FPGA com arquitetura paralela, pipelined e os dois somadores implementados: (a) Iterações vs. tempo de processamento (b) Iterações vs. recursos da FPGA.

Por outro lado, o algoritmo CORDIC configurado em modo rotacional pode operar como um conversor fase-amplitude que gera diretamente a onda seno e coseno [35]. A principal vantagem da utilização do DDS baseado em CORDIC com relação ao método baseado em LUT é que pode atingir uma alta resolução de fase e uma alta precisão com baixo custo de hardware [36].

Uma diferença entre os dois métodos é que o acumulador de fase no método baseado em LUT

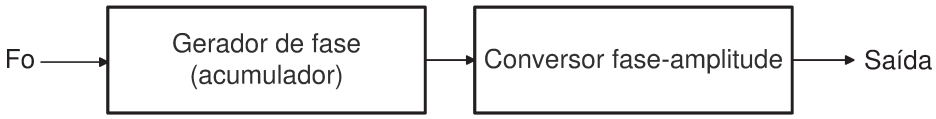


Figura 4.41: Diagrama de blocos do DDS.

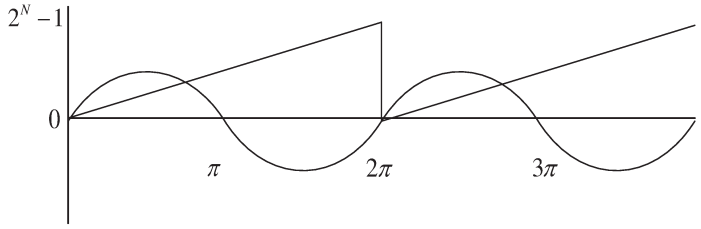


Figura 4.42: Formas de onda do método baseado em LUT.

produz um valor inteiro que direciona uma LUT, enquanto o método baseado em CORDIC gera um ângulo. Portanto, nesse último caso, um sinal rampa no intervalo $[-\pi, \pi]$ deve ser obtido pelo acumulador, conforme mostrado na Figura 4.43.

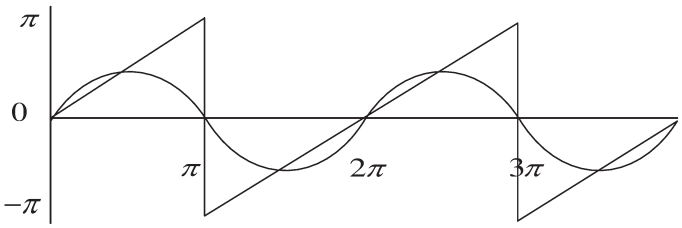


Figura 4.43: Formas de onda do método baseado em CORDIC.

Com um sinal de teste com desvio de frequência de 100 kHz, é obtida uma estimativa grosseira de 108.3 kHz, conforme mostrado na Figura 4.44. Note-se que essa estimativa não é tão precisa, sendo necessária uma estimativa fina.

Uma vez que a estimativa grosseira é determinada, os símbolos LTS são multiplicados por uma exponencial complexa com fase oposta a essa estimativa. Cada amostra do símbolo LTS é multiplicada por um sinal seno e coseno gerado pelo CORDIC, cuja fase vem do acumulador do DDS. Na Figura 4.45 é mostrado o sinal do acumulador, cujos limites são $-\pi$ e π. Neste trabalho, a fase calculada com o CORDIC foi acumulada até um valor de $\pm\pi$. Nesse instante, é adicionado um valor de $\mp 2\pi$, para conseguir meio ciclo da onda seno ou coseno com o CORDIC em modo rotacional.

A Figura 4.46 mostra as funções seno e coseno geradas no circuito.

O resultado do *test bench* é apresentado na Figura 4.47. O circuito final, possui além das entradas real e imaginária e do relógio, um sinal de *reset* e ativação.

O resultado da estimativa fina é apresentado na Figura 4.48. Para o caso particular deste teste, a estimativa final resultante da soma das estimativas grosseira e fina foi de 101.0048 kHz.

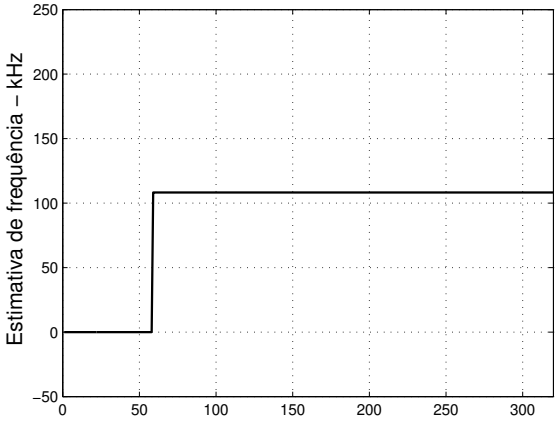


Figura 4.44: Estimativa grosseira.

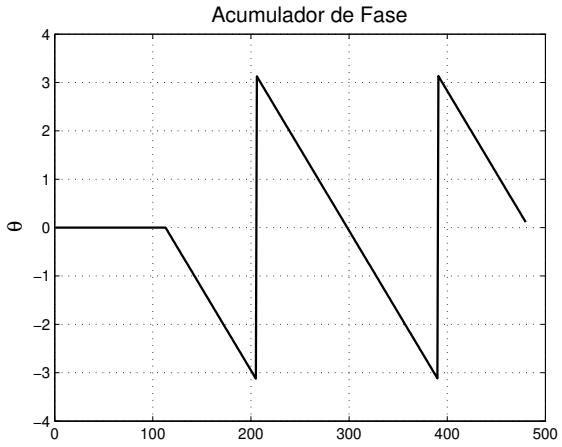
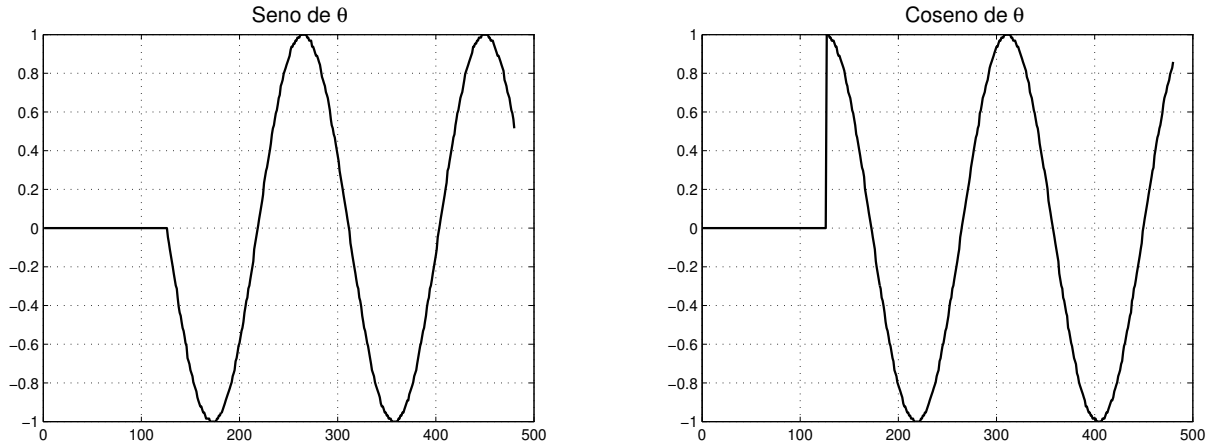


Figura 4.45: Acumulador de fase.



(a) (b)

Figura 4.46: (a) Função seno (b) Função coseno.

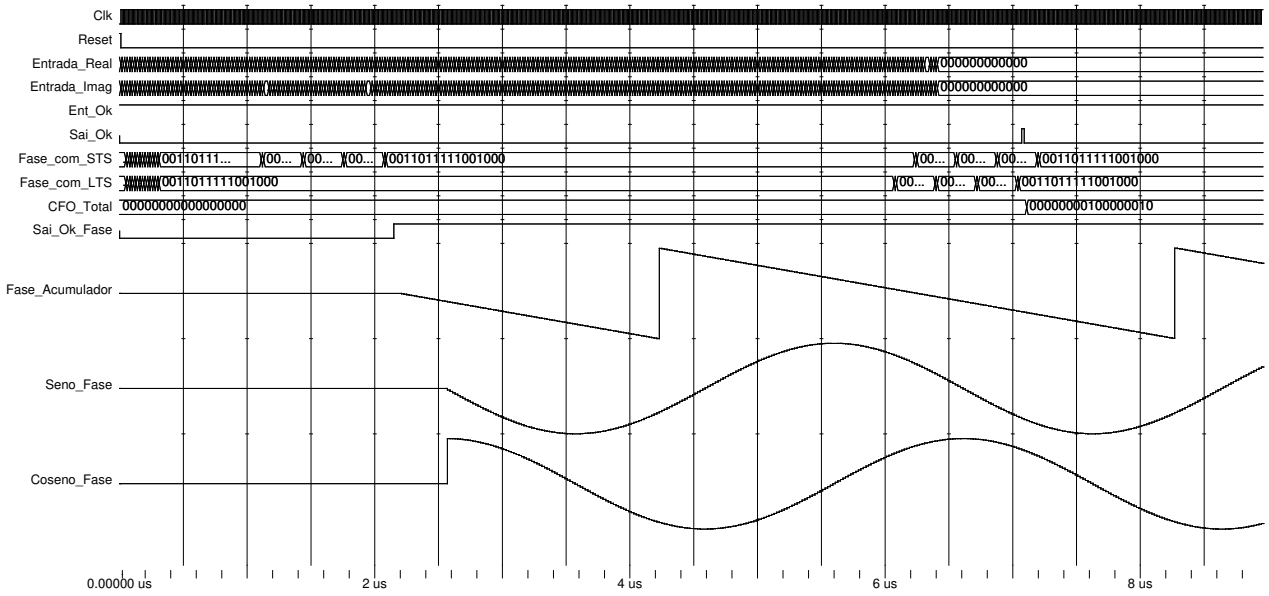


Figura 4.47: *Test bench* do circuito de estimação e compensação.

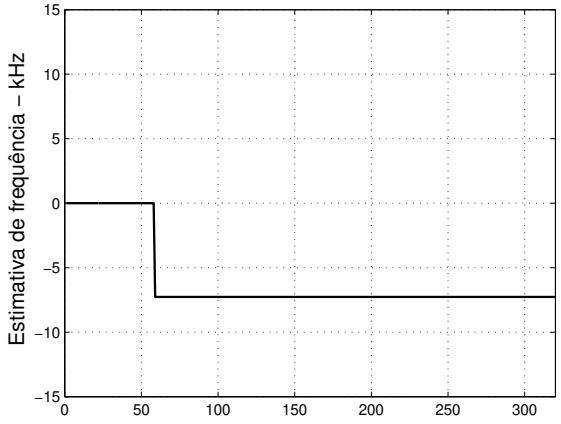
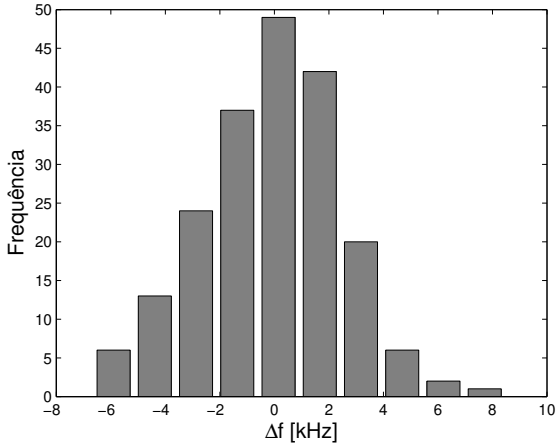


Figura 4.48: Estimativa fina.

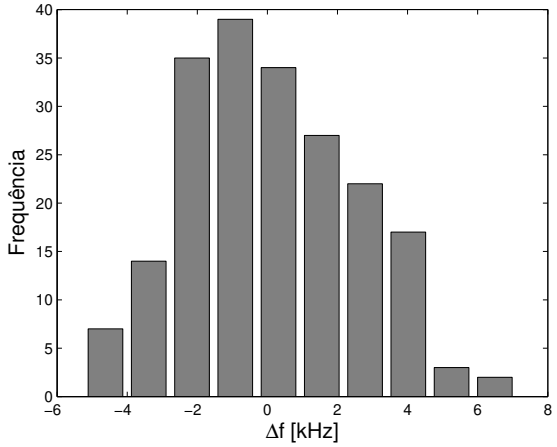
Para testar o circuito e analisar o comportamento em várias condições de canal, uma série de testes foram realizados. A SNR foi mantida constante em 10 dB e o desempenho do circuito foi examinado em condições de vários desvios de frequência e atraso de canal. O circuito foi testado usando-se 1200 sinais de entrada com desvios de frequência de 0, 100 e 200 kHz. As Figuras 4.49, 4.50 e 4.51 apresentam os resultados da estimação total para os desvios de frequência introduzidos.

Na implementação, um contador permite separar os últimos símbolos curtos cujos valores baseiam o calculo da fase de autocorrelação. Da mesma forma, usando-se um contador, são isolados os símbolos longos antes de que sejam compensados.

Com uma entrada de 12 bits, é obtido um resultado de 25 bits após a multiplicação complexa (24 bits da multiplicação mais 1 bit da soma). O acumulador (equação recursiva) tem um comprimento de janela de 16 e, portanto, é obtida uma palavra binária de 29 bits na saída do bloco de acumulação. Esse valor é ingressado ao CORDIC em modo vetorial para calcular a

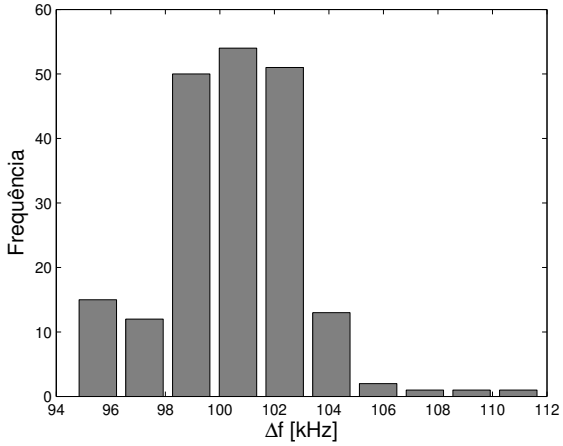


(a) Atraso de 50 ns

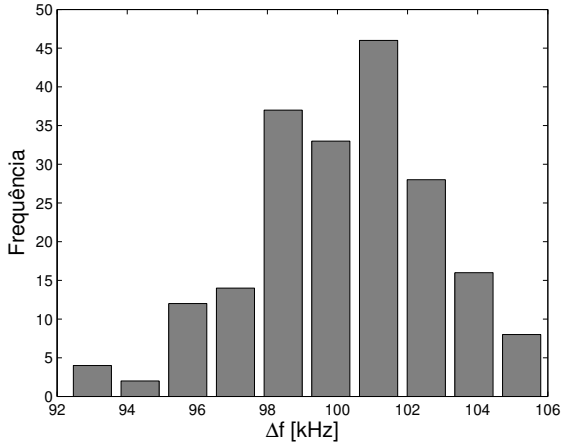


(b) Atraso de 150 ns

Figura 4.49: Histograma de frequência com 0 kHz.



(a) Atraso de 50 ns

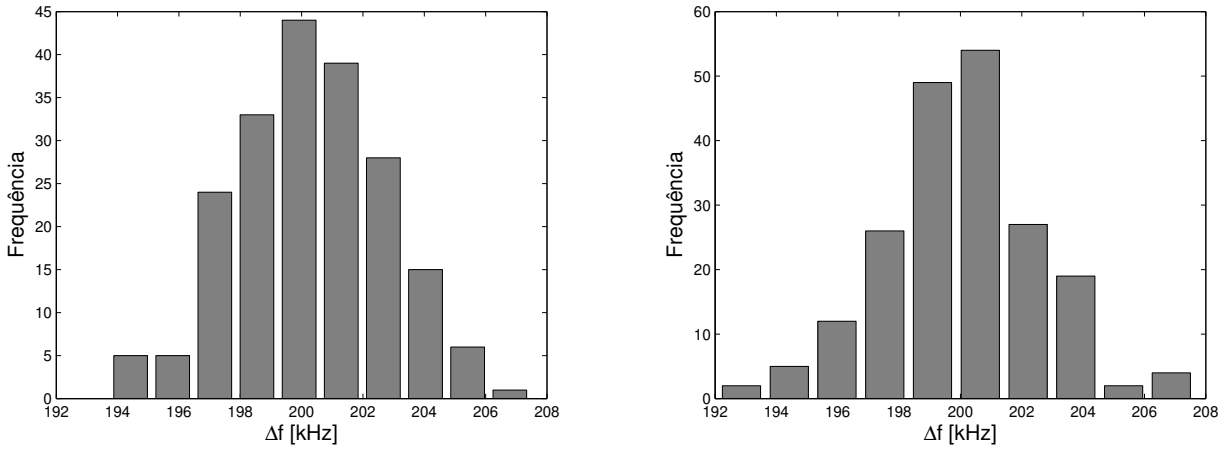


(b) Atraso de 150 ns

Figura 4.50: Histograma de frequência com 100 kHz.

fase da autocorrelação. Em [31] foi calculada a relação entre o número de iterações e o erro do resultado do CORDIC, no qual foi determinado que, com 16 iterações e para uma palavra binária de 16 bits, obteve-se um cálculo com boa precisão da fase. Assim, a saída do acumulador pode ser reduzida a uma palavra binária de 16 bits com um circuito de deslocamento à direita. O CORDIC calcula uma fase de comprimento de 16 bits. Essa fase é proporcional ao desvio de frequência da portadora.

As 128 amostras dos símbolos LTS são compensadas com a fase da autocorrelação dos STS. Essa fase é acumulada, para o caso particular de uma frequência estimada positiva, até um valor de $-\pi$. Quando o acumulador de fase é menor que $-\pi$, soma-se a ele um valor de 2π. Dessa forma, é formado o sinal de acumulação de fase que gera um sinal seno e coseno de tamanho binário igual ao símbolo LTS. Depois que cada símbolo LTS é compensado através de uma multiplicação complexa, cada amostra é dividida por 64 com um deslocamento à direita de 6 bits, resultando num sinal com 11 bits. Esse símbolo LTS compensado é autocorrelacionado, de modo que se pode encontrar a estimativa fina de frequência com um processo semelhante ao da



(a) Atraso de 50 ns (b) Atraso de 150 ns

Figura 4.51: Histograma de frequência com 200 kHz.

estimativa grosseira.

Os resultados com os valores máximos e mínimos estimados são apresentados na Tabela 4.13.

Tabela 4.13: Estimação do desvio da frequência da portadora.

Δf **(kHz)**	**Atraso (ns)**	**Máx. Estimado (kHz)**	**Mín. Estimado (kHz)**	**Média**
0	50	8,4740	-6,6541	-0,1233
0	150	7,1080	-5,2365	0,1264
100	50	111,8056	94,6664	100,4710
100	150	105,9347	92,3259	100,1472
200	50	207,4753	193,7360	200,2996
200	150	207,6560	192,0816	200,0230

O resultado do custo de implementação do algoritmo é apresentado na Tabela 4.14.

Tabela 4.14: Recursos para estimação e compensação do CFO.

Slice Logic Utilization	Used
Number of Slice Flip Flops	4881
Number of 4 input LUTs	9617
Number of Slices	5135
MULT18X18SIOs	11
Maximum frequency	77.561MHz

Capítulo 5

Conclusões

Este trabalho apresentou a implementação em FPGA de vários algoritmos de sincronismo para um receptor OFDM baseado no padrão IEEE 802.11a. Usando uma linguagem de descrição de hardware (VHDL), foi possível desenvolver, implementar e testar as operações matemáticas empregadas por esses algoritmos, as quais são: registro de atraso, deslocamento de bit, multiplicação complexa, acumulação, cálculo de fase e geração de funções seno e coseno.

Conforme explicado no Capítulo 2, a primeira tarefa de sincronismo é a detecção do pacote, realizada com o aproveitamento da estrutura do preâmbulo. A parte do preâmbulo correspondente aos símbolos curtos (compostos de 160 amostras complexas com uma duração de 50 ns por amostra) é correlacionada com a cópia atrasada de si mesma, gerando um sinal em forma de degrau que indica a presença do pacote. No entanto, conforme explicado na seção 2.8, para que essa detecção seja independente da potência do sinal recebido, é introduzida uma comparação de amplitude entre o sinal resultante da autocorrelação e a metade da potência do sinal. Esse sinal apresenta, às vezes, picos de curta duração. Por conseguinte, foi introduzido um circuito de média que gera um sinal ativo sempre que o sinal da autocorrelação é 32 vezes consecutivas maior que a metade da potência.

Após a detecção da presença do preâmbulo do sinal OFDM, é realizada a estimativa de tempo de início do pacote. Existem vários métodos propostos para determinar a amostra de início, sendo implementados aqueles que são baseados na operação de autocorrelação e correlação cruzada das amostras dos símbolos do preâmbulo. Os algoritmos de autocorrelação básica e de diferença de autocorrelação apresentam a maior variância e uma frequência de operação semelhante. No entanto, o algoritmo de autocorrelação básica possui um menor consumo de *slices*. O algoritmo de autocorrelação básica com atrasador de 144 amostras possui uma variância menor, mantendo a frequência de operação. Porém, utiliza mais recursos de *slices*.

Já no caso de algoritmos baseados na correlação cruzada com os LTS, o melhor resultado em cada um dos parâmetros foi aquele que usa 32 amostras do símbolo. A correlação cruzada com os símbolos STS possui a menor variância, a maior frequência de operação e um menor consumo de *slices*, se comparada com a dos algoritmos que usam a correlação cruzada com os LTS.

Por conseguinte, a melhor escolha em relação à variância são os algoritmos baseados em correlação cruzada, embora sejam os que permitem menor frequência de operação.

O descasamento entre o cristal do transmissor e do receptor e a frequência Doppler introduz um desvio de frequência que causa ICI e ISI. Esse desvio de frequência é proporcional à fase da autocorrelação do sinal recebido. A estimativa do desvio de frequência é calculada com o algoritmo CORDIC em modo vetorial em um sistema coordenado circular. Essa estimativa

é denominada estimativa grosseira. Após esta estimativa, o sinal LTS é multiplicado por uma exponencial complexa com frequência igual e oposta ao valor da estimativa grosseira. Os valores que compõem a exponencial complexa vêm do CORDIC em modo rotacional, o qual gera a função seno e coseno. Posteriormente, uma nova autocorrelação é realizada com os símbolos LTS para novamente estimar o desvio de frequência. Essa estimativa é denominada estimativa fina. Conforme mostrado nos resultados, a arquitetura implementada consegue estimar e compensar desvios de 0, 100 e 200 kHz. Essa etapa de sincronismo emprega o CORDIC em arquitetura *pipelined* e um somador CSA com o intuito de melhorar a frequência de operação em FPGA.

Para trabalhos futuros, baseado nos resultados desta dissertação, os seguintes tópicos são sugeridos:

- Extensão dos algoritmos para sistemas MIMO-OFDM.
- Impacto do limiar na detecção do pacote.
- Aplicação a outros sistemas de comunicação sem fio que usam OFDM como WiMAX, LTE, 802.11p, televisão digital, etc.
- Otimização do CORDIC, para uma menor área ocupada na FPGA e uma maior velocidade de funcionamento.

Referências bibliográficas

[1] Y.G. Li and G.L. Stuber. *Orthogonal Frequency Division Multiplexing for Wireless Communications*. Signals and Communication Technology. Springer, 2006.

[2] Y.S. Cho, J. Kim, W.Y. Yang, and C.G. Kang. *MIMO-OFDM Wireless Communications with MATLAB*. Wiley, 2010.

[3] T.D. Chiueh, P.Y. Tsai, and I.W. Lai. *Baseband Receiver Design for Wireless MIMO-OFDM Communications*. Wiley, 2012.

[4] C. Cardoso, M. Fernandes, and D. Arantes. Fpga e fluxo de projeto. http://www.decom.fee.unicamp.br/čardoso/ie344b/Introducao_FPGA_Fluxo_de_Projeto.pdf/, Julho 2007.

[5] P.P. Chu. *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. Wiley, 2008.

[6] Abraham Peled and A. Ruiz. Frequency domain data transmission using reduced computational complexity algorithms. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '80.*, volume 5, pages 964–967, 1980.

[7] Jeich Mar, Chi-Cheng Kuo, and Shih-Hao Chou. FPGA implementation of SDR based CFO estimation and compensation circuit for OFDM system. *Journal of Signal Processing Systems*, 66:141–146, 2012. 10.1007/s11265-011-0621-y.

[8] Erick Rocha. SOUSA. Um estudo sobre a sincronização de sistemas WiMAX. Dissertação (mestrado), Universidade Estadual de Campinas, Faculdade de Engenharia Eletrica e de Computação, 2009.

[9] T. Pollet, M. Van Bladel, and M. Moeneclaey. BER sensitivity of OFDM systems to carrier frequency offset and Wiener phase noise. *Communications, IEEE Transactions on*, 43(234):191–193, 1995.

[10] N. Shahin, N.J. LaSorte, S.A. Rajab, and H.H. Refai. 802.11g channel characterization utilizing labview and NI-USRP. In *Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International*, pages 753–756, 2013.

[11] G. Maier, A. Paier, and C.F. Mecklenbrauker. Packet detection and frequency synchronization with antenna diversity for IEEE 802.11p based on real-world measurements. In *Smart Antennas (WSA), 2011 International ITG Workshop on*, pages 1–7, 2011.

[12] R.P. Fabris Hoefel and A. Michielin Camara. Interoperability between IEEE 802.11n and IEEE 802.11a/g devices: Analytical and simulation results. In *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on*, pages 001466–001469, 2011.

[13] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, 2012.

[14] André Michielin Câmara. Sincronização de redes de pacotes OFDM. Tese de graduação, Universidade Federal do Rio Grande do Sul. Escola de Engenharia. Curso de Engenharia Elétrica, 2007.

[15] T. Keller and L. Hanzo. Orthogonal frequency division multiplex synchronisation techniques for wireless local area networks. In *Personal, Indoor and Mobile Radio Communications, 1996. PIMRC'96., Seventh IEEE International Symposium on*, volume 3, pages 963–967 vol.3, 1996.

[16] T.M. Schmidl and D.C. Cox. Robust frequency and timing synchronization for OFDM. *Communications, IEEE Transactions on*, 45(12):1613–1621, 1997.

[17] K. Wang, J. Singh, and M. Faulkner. FPGA implementation of an OFDM-WLAN synchronizer. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 89–94, 2004.

[18] C. Dick and F. Harris. FPGA implementation of an OFDM PHY. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 1, pages 905–909 Vol.1, 2003.

[19] Anna Berno and Nicola Laurenti. Time and Frequency Synchronization for Hiperlan/2. In *Proceedings of the Second International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols*, pages 491–502, London, UK, UK, 2002. Springer-Verlag.

[20] Jonas Medbo and Peter Schramm. Channel models for HIPERLAN/2 in different indoor scenarios. *3ERI085B*, 1998.

[21] Randy Yates. Fixed-Point Arithmetic: An introduction. http://www.digitalsignallabs.com/fp.pdf, Julho 2009.

[22] Evgeni Stavinov. Using Xilinx Tools in Command-Line Mode. Online at http://outputlogic.com/xcell_using_xilinx_tools/74_xperts_04.pdf, Julho 2011.

[23] S. Johansson, M. Nilsson, and P. Nilsson. An OFDM timing synchronization ASIC. In *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on*, volume 1, pages 324–327 vol.1, 2000.

[24] Jinpeng Xie, Yingqiang Ding, Shouyi Yang, and Lin Qi. FPGA implementation of frame synchronization and symbol timing synchronization based on OFDM system for IEEE 802.11 a. In *Intelligent Signal Processing and Communication Systems (ISPACS), 2010 International Symposium on*, pages 1–4. IEEE, 2010.

[25] María José Canet, Felip Vicedo, Vicenç Almenar, Javier Valls, and Eduardo R. Lima. Hardware design of a FPGA-based synchronizer for Hiperlan/2. In Jurgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application*, volume

3203 of *Lecture Notes in Computer Science*, pages 494–504. Springer Berlin Heidelberg, 2004.

[26] Richard G. Lyons. *Understanding Digital Signal Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2st edition, 2004.

[27] María José Canet, Javier Valls, Vicenç Almenar, and José Marín-Roig. FPGA implementation of an OFDM-based WLAN receiver. *Microprocessors and Microsystems*, 36(3):232 – 244, 2012.

[28] Riza Dönmez. Digital implementation of ETSI OFDM symbol synchronizer based on sliding correlation. Master thesis, Sabanci University, 2003.

[29] M.J. Canet, I. Wassel, V. Almenar, and J. Valls. Performance evaluation of fine time synchronizer for WLANs. In *Proceedings of 13th European Conference on Signal Processing (EUSIPCO 2005)*, volume 2, pages II–341–4 vol.2, 2005.

[30] Ferney Amaya and Jaime Velasco. Diseño de la tangente inversa usando el algoritmo CORDIC. *Taller XII Iberchip*, 1:183 – 186, 2006. 10.1023/A:1020202417934.

[31] Diego Barragán, Karlo Lenzi, and Luís Meloni. Desempenho do Algoritmo Paralelo CORDIC em Implementação em FPGA. *XXX Simposio Brasileiro De Telecomunicações 12*, 2012.

[32] Diego Barragán and Luís Meloni. Comparison of Parallel and Pipelined CORDIC algorithm using RCA and CSA. *IWT International Workshop on Telecommunications*, 2013.

[33] Robert Joachim Schweers. Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC. http://biblioteca.universia.net/, Julho 2009.

[34] L. Cordesses. Direct digital synthesis: a tool for periodic wave generation (part 1). *Signal Processing Magazine, IEEE*, 21(4):50–54, 2004.

[35] J. Vankka. Methods of mapping from phase to sine amplitude in direct digital synthesis. In *Frequency Control Symposium, 1996. 50th., Proceedings of the 1996 IEEE International.*, pages 942–950, 1996.

[36] F. Cardells-Tormo and J. Valls-Coquillat. Area-optimized implementation of quadrature direct digital frequency synthesizers on LUT-based FPGAs. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 50(3):135–138, 2003.

[37] Scott Hauck and Andre DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[38] Ray Andraka. A survey of CORDIC algorithms for FPGA based computers. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, FPGA '98, pages 191–200, New York, NY, USA, 1998. ACM.

[39] Jack E. Volder. The CORDIC trigonometric computing technique. *Electronic Computers, IRE Transactions on*, EC-8(3):330 –334, sept. 1959.

[40] X. Hu, R.G. Harber, and S.C. Bass. Expanding the range of convergence of the CORDIC algorithm. *Computers, IEEE Transactions on*, 40(1):13 –21, jan 1991.

Apêndice

Apêndice A

Conceitos matemáticos do algoritmo CORDIC

Uma das abordagens mais úteis e flexíveis disponíveis para o desenvolvimento de computação em hardware de alto desempenho é o algoritmo CORDIC [3].

O CORDIC permite a implementação de diversas funções matemáticas por meio de poucas iterações, sua flexibilidade é verificada em uma arquitetura única de hardware com sobrecarga de controle mínima, tendo a capacidade de calcular seno, coseno, cosh, senh, arctg, raiz quadrada, e conversões polar-para-retangular e retangular-para-polar, por citar apenas algumas funções [37].

Todas as funções trigonométricas podem ser calculadas ou derivadas utilizando-se rotações de um vetor [38]. A rotação de um vetor pode também ser usada para conversão de um sistema polar a retangular e de retangular a polar. O algoritmo CORDIC fornece um método para se realizar rotações de um vetor em ângulos arbitrários, usando apenas deslocamentos e somas. Uma FPGA é muito eficiente na realização de somadores de precisão arbitrária e, assim, o algoritmo CORDIC é em muitos aspectos uma escolha natural para FPGA. O algoritmo, desenvolvido por Volder [39], é derivado das equações gerais de rotação de um vetor com coordenadas (x,y) por um ângulo ϕ qualquer, como é representado na Figura A.1:

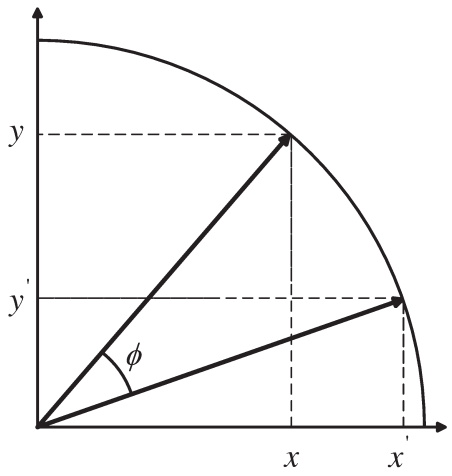


Figura A.1: Rotação vetorial.

Esta rotação vetorial pode ser expressa por:

$$x' = x\cos\phi - y\,\mathrm{sen}\phi. \tag{A.1}$$

$$y' = y\cos\phi + x\,\mathrm{sen}\phi. \tag{A.2}$$

que gira um vetor em um plano cartesiano pelo ângulo ϕ. As equações anteriores podem ser rearranjadas da seguinte forma:

$$x' = \cos\phi\,[x - y\,\mathrm{tg}\phi]\,. \tag{A.3}$$

$$y' = \cos\phi\,[y + x\,\mathrm{tg}\phi]\,. \tag{A.4}$$

Por enquanto, não há nenhuma simplificação. No entanto, se os ângulos de rotação são limitados de modo que $\mathrm{tg}\,(\phi) = \pm 2^{-i}$, a multiplicação pela tangente é simplificada a uma operação de deslocamento. Realizando-se pequenas rotações iterativas, é possível girar um vetor por um ângulo arbitrário qualquer. Se a decisão a ser tomada em cada iteração i é definir qual direção girar, em vez de se definir se deve girar ou não, o termo $cos(\delta_i)$ se torna constante (devido a $cos(\delta_i) = cos(-\delta_i)$). Assim, a rotação iterativa é definida como

$$x_{i+1} = K_i\left[x_i - y_i\,d_i\,2^{-i}\right]. \tag{A.5}$$

$$y_{i+1} = K_i\left[y_i + x_i\,d_i\,2^{-i}\right], \tag{A.6}$$

onde

$$K_i = \cos\left(\mathrm{tg}^{-1}\,2^{-i}\right) = \frac{1}{\sqrt{1+2^{-2i}}}. \tag{A.7}$$

$$d_i = \pm 1. \tag{A.8}$$

Sem se considerar a constante nas equações iterativas, tem-se um algoritmo para girar um vetor só com deslocamentos e somas. O tratamento do fator K_i pode ser considerado em outra parte do sistema ou considerado como ganho de processamento. Esse valor é de aproximadamente 0.6073 quando o número de iterações tende ao infinito. Portanto, o algoritmo de rotação tem um ganho de A_n, que é de aproximadamente de 1.647. O valor exato do ganho depende do número de iterações, conforme mostrado por

$$A_n = \prod_n \sqrt{1+2^{-2i}}. \tag{A.9}$$

O ângulo de rotação composto é definido exclusivamente pela sequência das direções das rotações elementares. Essa sequência pode ser representada por um vetor de decisão. O conjunto de todos os possíveis vetores de decisão é um sistema de medição angular com base em arco-tangentes binárias. No algoritmo, usa-se um somador-subtrator adicional que acumula os ângulos de rotação elementares em cada iteração. Os ângulos elementares podem ser expressos em qualquer unidade angular conveniente. Esses valores angulares são fornecidos por uma LUT (uma entrada por cada iteração) ou são diretamente conectados, dependendo da implementação. O ângulo acumulador adiciona uma terceira equação dada por

$$z_{i+1} = z_i - d_i\ \mathrm{tg}^{-1}\left(2^{-i}\right). \tag{A.10}$$

O algoritmo CORDIC pode ser empregado em dois modos de operação [38], ilustrados na Figura A.2.

Modo vetorial: a entrada é o vetor $\vec{r} = (x, y)$ e a saída é a magnitude R e o ângulo θ. Neste caso, o algoritmo gira o vetor para alinhá-lo ao eixo X (acumulando o ângulo necessário para fazer essa rotação, cujo acumulador é iniciado em zero) até reduzir a magnitude do eixo Y a 0. Este modo é utilizado para estimar o desvio de frequência da portadora.

Modo rotacional: as entradas são o vetor $\vec{r} = (x, y)$ e o ângulo de giro θ, e a saída é o vetor girado com novas coordenadas $\vec{r}' = (x', y')$. Desta forma, o vetor de entrada gira um ângulo específico, que é introduzido como parâmetro. Este modo é utilizado para compensar o desvio da frequência da portadora.

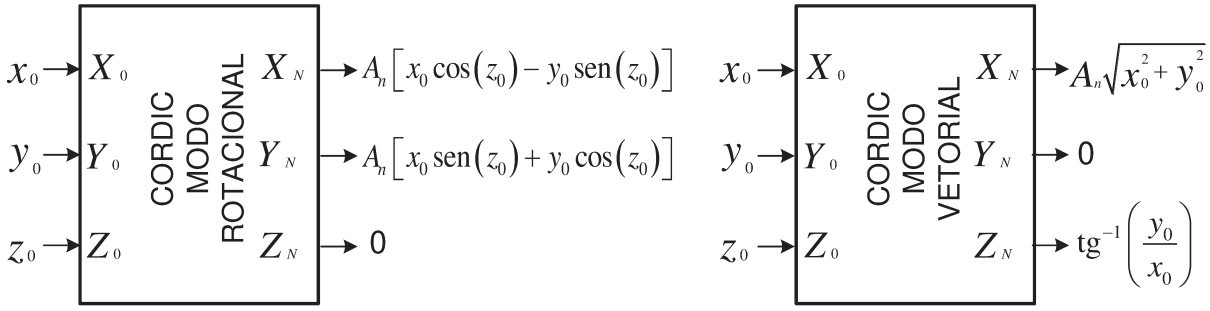


Figura A.2: CORDIC: modo rotacional e modo vetorial.

Usando-se as equações CORDIC, obtém-se rotações para ângulos menores que 90^o [40]. Para atingir rotações maiores que 90^o, deve-se realizar primeiro uma rotação de $\pm 90^o$ (para garantir que o vetor fique apenas nos quadrantes I ou IV), empregando-se as seguintes equações:

$$x_{i+1} = -\, y_i \, d_i \,. \tag{A.11}$$

$$y_{i+1} = x_i \, d_i \,. \tag{A.12}$$

$$z_{i+1} = d_i \cdot 90^\circ, \tag{A.13}$$

onde $d_i = -1$ para girar 90^o e $d_i = +1$ para girar -90^o.

Para se calcular a tangente inversa e módulo de um vetor, emprega-se o algoritmo CORDIC circular em modo vetorial (o vetor da entrada é girado para ser alinhado com o eixo Y). Neste caso, as coordenadas x_i e y_i do vetor $\vec{r}$ fazem a iniciação do algoritmo e, ao final das iterações, obtém-se, na variável z_{i+1}, o valor do ângulo θ do vetor $\vec{r}$, e na variável x_{i+1} obtém-se o valor da magnitude do vetor $\vec{r}$ multiplicada pelo fator A_n. As equações para esse modo são (com $z_0 = 0$ e n igual ao número de iterações):

$$x_n = A_n \sqrt{(x_0)^2 + (y_0)^2}. \tag{A.14}$$

$$y_n = 0. \tag{A.15}$$

$$z_n = z_0 - \mathrm{tg}^{-1}\left(\frac{y_0}{x_0}\right). \tag{A.16}$$

$$A_n = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}. \tag{A.17}$$

Portanto, o algoritmo tem duas etapas [38]: iniciação juntamente com rotação por $\pm\,90^o$ e pseudo-rotações iterativas que buscam levar a zero a variável y_i. O termo pseudo-rotação vem do fato de que o vetor aumenta a sua amplitude a cada iteração pelo valor de A_n.

Para se realizar a etapa de rotação de $\pm\,90^o$, utiliza-se o procedimento ilustrado na Figura A.3.

1. Inicialização: $x_i = x$; $y_i = y$; valores do vetor $\vec{r} = (x, y)$, $i = 0$.
2. Se $y_i \geq 0$, então $d_i = -1$, caso contrário, $d_i = 1$.
3. Rotação por $\pm\,90^o$:

$$\begin{aligned} x_{i+1} &= -\,y_i\, d_i \\ y_{i+1} &= x_i\, d_i \\ z_{i+1} &= d_i \cdot 90^\circ \end{aligned}$$

Figura A.3: Procedimento da etapa de inicialização e rotação, para garantir que o vetor fique apenas nos quadrantes I ou IV.

Para se realizar a etapa de N pseudo-rotações iterativas, utiliza-se o procedimento ilustrado na Figura A.4.

1. Se $y_i \geq 0$, então $d_i = -1$, caso contrário, $d_i = 1$.
2. Pseudo-rotações:

$$\begin{aligned} x_{i+1} &= x_i - y_i\, d_i\, 2^{-i} \\ y_{i+1} &= y_i + x_i\, d_i\, 2^{-i} \\ z_{i+1} &= z_i - d_i \mathrm{tg}^{-1}(2^{-i}) \end{aligned}$$

3. Incrementar i.
4. Se i é menor que n, ir para o passo 1.
5. Sair

Figura A.4: Procedimento para as etapas de pseudo-rotações.

As equações do algoritmo CORDIC (A.5), (A.6) e (A.10) para o modo rotacional têm o seguinte resultado final:

$$x_n = A_n \left[x_0 \cos z_0 - y_0 \text{ sen } z_0 \right]. \tag{A.18}$$

$$y_n = A_n \left[y_0 \cos z_0 + x_0 \text{ sen } z_0 \right]. \tag{A.19}$$

$$z_n = 0. \tag{A.20}$$

$$A_n = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}. \tag{A.21}$$

Fazendo com que $y_0 = 0$, é possível reduzir os resultados do modo de rotação a:

$$x_n = A_n \, x_0 \cos z_0 \, . \tag{A.22}$$

$$y_n = A_n \, x_0 \text{ sen } z_0 \, . \tag{A.23}$$

Impondo $x_0 = 1/A_n$, o processo de rotação gera o seno e o coseno com fase de z_0.

Para se realizar a etapa de N pseudo-rotações iterativas, utiliza-se procedimento igual ao das Figuras A.3 e A.4 mudando o passo 1 conforme mostrado em

$$d_i = 1 \; se \;\; z_i > 0, \;\; -1 \; se \;\; z_i < 0. \tag{A.24}$$