UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE ENGENHARIA ELETRÔNICA

ENGENHARIA ELETRÔNICA

ANDERSON CARLOS WOSS

IMPLEMENTAÇÃO DE UM ALGORITMO DE COMUNICAÇÃO EMBARCADO EM DISPOSITIVO LÓGICO PROGRAMÁVEL COM APLICAÇÃO DE TÉCNICA DE MULTIPLEXAÇÃO EM FREQUÊNCIA

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO

2014

ANDERSON CARLOS WOSS

IMPLEMENTAÇÃO DE UM ALGORITMO DE COMUNICAÇÃO EMBARCADO EM DISPOSITIVO LÓGICO PROGRAMÁVEL COM APLICAÇÃO DE TÉCNICA DE MULTIPLEXAÇÃO EM FREQUÊNCIA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Eletrônica, da Coordenação de Engenharia Eletrônica, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Paulo de Tarso Neves Júnior

TOLEDO

2014

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Toledo
Coordenação do Curso de Engenharia Eletrônica

TERMO DE APROVAÇÃO

Título do Trabalho de Conclusão de Curso Nº 006

Implementação de um Algoritmo de Comunicação Embarcado em Dispositivo Lógico Programável com Aplicação de Técnica de Multiplexação em Frequência

por

Anderson Carlos Woss

Esse Trabalho de Conclusão de Curso foi apresentado às 10:20 h do dia **04 de agosto de 2014** como requisito parcial para a obtenção do título **Bacharel em Engenharia Eletrônica**. Após deliberação da Banca Examinadora, composta pelos professores abaixo assinados, o trabalho foi considerado **APROVADO**.

_____
Prof. Dr. Alberto Yoshihiro Nakano
(UTFPR-TD)

_____
Prof. Dr. Felipe Walter Dafico Pfrimer
(UTFPR-TD)

_____
Prof. Dr. Paulo de Tarso Neves Junior
(UTFPR-TD)
Orientador

Visto da Coordenação

_____
Prof. M. Alessandro Paulo de Oliveira
Coordenador da COELE

Dedico este trabalho a todos aqueles que,
junto a mim, sentem-se orgulhosos e
felizes por minhas conquistas.

AGRADECIMENTOS

Agradeço à minha família e amigos por sempre terem me apoiado ao longo da minha vida acadêmica e por estarem presentes auxiliando-me nas dificuldades e compartilhando as alegrias.

Agradeço a todos os meus professores que não mediram esforços em transmitir seus conhecimentos e experiências, permitindo que eu chegasse até aqui, e em especial ao professor Doutor Paulo de Tarso Neves Júnior por ter me acompanhado e me auxiliado ao longo deste trabalho.

Agradeço a Universidade Tecnológica Federal do Paraná, campus Toledo, pelas excelentes condições de estudos que me proporcionaram.

A todos muito obrigado!

“Se enxerguei mais longe, foi por estar de
pé sobre ombros de gigantes”
(NEWTON, Isaac; 1676)

RESUMO

WOSS, Anderson Carlos. **Implementação de um Algoritmo de Comunicação Embarcado em Dispositivo Lógico Programável com Aplicação de Técnica de Multiplexação em Frequência**. 2014. 89 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná. Toledo, 2014.

Este trabalho relata o processo de implementação em VHDL (*VHSIC Hardware Description Language*) de um algoritmo de comunicação com aplicação da técnica OFDM (*Orthogonal Frequency-Divide Multiplexing*) e modulação QAM (*Quadrature Amplitude Modulation*). A modulação pode ser feita através da transformada de Fourier, ou da transformada rápida de Fourier (FFT), quando digitalmente. Desta forma, a FFT é implementada com base no algoritmo de Cooley-Tukey, na configuração Radix-2, com comprimento de oito pontos, utilizando números na representação com ponto flutuante de 32 bits. O código desenvolvido foi sintetizado no *software* Altera Quartus II e simulado no ModelSim. Como resultado, conseguiu-se transmitir um texto de 561 caracteres com sucesso, a partir da simulação funcional, a uma taxa de 34,5 MB/s utilizando um *clock* de 50 MHz, sendo possível sua recuperação sem erros no receptor, porém, devido às multiplicações com ponto flutuante e estrutura adotada para a FFT, o código sintetizado exigiu mais recursos que os dispositivos disponíveis continham e não foi possível embarcá-lo.

Palavras-chave: VHDL. OFDM. QAM. FPGA.

ABSTRACT

WOSS, Anderson Carlos. **Implementation of a Communication Algorithm Embedded on Programmable Logic Device with Frequency Multiplexing Technique Application**. 2014. 89 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Federal University of Technology - Paraná. Toledo, 2014.

This paper describes the process of implementation in VHDL (VHSIC Hardware Description Language) of a communication algorithm with applications of OFDM (Orthogonal Frequency-Divide Multiplexing) technique and QAM (Quadrature Amplitude Modulation). The modulation can be done by Fourier transform or the fast Fourier transform (FFT) when digitally. Thus, the FFT is implemented based on the Cooley-Tukey algorithm, in the Radix-2 configuration, with a length of eight points, using numbers in floating-point representation of 32 bits. The code developed was synthesized in Altera Quartus II and simulated in ModelSim. As a result, it was possible to transmit a text of 561 characters successfully, from functional simulation, at a rate of 34.5 MB/s using a clock of 50 MHz, with possible recovery without errors at the receiver, however, due to the floating-point multiplications and structure adopted for the FFT, the synthesized code demanded more resources than available devices contained and was not possible embed it.

Keywords: VHDL. OFDM. QAM. FPGA.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE SIGLAS E ACRÔNIMOS

LISTA DE SIGLAS

CFA	Algoritmos de fatores comuns
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
FDM	Multiplexação por divisão em frequências
FPGA	Arranjo de portas programável em campo
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
LUT	*Look-Up Table*
MCM	Modulação em múltiplas portadoras
OFDM	Multiplexação por divisão em frequências ortogonais
PFA	Algoritmos de fatores primos
PSK	Modulação digital em fase
RTL	Nível de transferência de registro
SCM	Modulação em portadora única
SDR	Rádio definidos por *software*
SPLD	Dispositivo lógico programável simples
VHDL	Linguagem de descrição de *hardware* para circuitos integrados de altíssimas velocidades

LISTA DE ACRÔNIMOS

ASK	Modulação digital em amplitude
BER	Taxa de erros de bits
CORDIC	Coordenação da rotação digital do computador
DARPA	Agência de Projetos e Pesquisas Avançadas de Defesa
LoS	Linha de vista
LUT	*Look-Up Table*
QAM	Modulação de amplitude em quadratura
QoS	Qualidade de serviço

LISTA DE SÍMBOLOS

$s(t)$	Sinal contínuo OFDM no domínio do tempo
ω	Frequência do sinal (Hz)
a	Termo em fase da modulação QAM
b	Termo em quadratura da modulação QAM
t	Tempo (s)
N	Número de canais utilizados na OFDM
k	Índice dos canais no domínio da frequência
T	Período do símbolo OFDM (s)
n	Índice dos canais no domínio do tempo
$s[n]$	Sinal discreto OFDM no domínio do tempo
z	Número complexo
j	Unidade imaginária
$\Re$	Parte real de um número complexo
$\Im$	Parte imaginária de um número complexo
$S[k]$	Sinal discreto OFDM no domínio da frequência
$X[k]$	Sinal, na frequência, obtido a partir da IDFT
$x[n]$	Sinal, no tempo, sobre o qual é calculado a IDFT
W_N^{nk}	n-ésimo fator de rotação da DFT de N pontos
λ	Índice da iteração do cálculo do CORDIC
ρ	Número de iterações do cálculo do CORDIC
Λ	Constante de congregação

SUMÁRIO

1 INTRODUÇÃO

Já nas primeiras décadas do século XX surgiu a necessidade da implementação de sistemas que permitissem múltiplas comunicações simultâneas. Na década de 1920, foram criados os equipamentos telefônicos com onda portadora para transmissão de dois ou quatro canais de voz (HAAS, 1977). Os primeiros modelos sofreram rápida evolução, ampliando de forma significativa a capacidade de transmissão.

O emprego de dispositivos semicondutores, a partir da década de 1950, deu grande impulso a este desenvolvimento, tornando os circuitos eletrônicos mais eficientes, confiáveis, compactos e com menor consumo de potência. A possibilidade destes componentes processarem sinais de frequências muito elevadas permitiu concentrar mais canais em uma mesma onda portadora em um único meio de transmissão, com emprego de sistemas poderosos de multiplexação (Ribeiro, 2012). Desta forma, o objetivo de ampliar a capacidade de comunicações vem sendo cumprido com bom desempenho através da aplicação das técnicas de multiplexação em dispositivos lógicos programáveis, os nomeados rádios definidos por *software* (SDR, *Software Defined Radio*).

Joseph Mitola escreveu em sua tese de mestrado, em 1991, o que é hoje considerada a primeira definição de rádios definidos por *software* (LOURENÇO, 2011).

> Um rádio definido por *software* é um rádio cuja forma de onda dos dados modulados é definida pelo *software*. Isto é, a forma de onda é gerada como sinais digitais, convertidos para analógicos por um conversor digital/analógico de banda larga e, em seguida, convertido da frequência intermediária para a frequência de rádio. O receptor, de forma semelhante, emprega um conversor analógico/digital de banda larga que capta todos os canais da onda transmitida, para, então, extrair e demodular a forma de onda de cada canal utilizando um *software* em um processador de propósito geral. (Mitola, 1995)

Resumidamente, este tipo de rádio pode ser entendido como uma única peça de *hardware* capaz de realizar diferentes funções em intervalos de tempos distintos através de mudanças no *software* que o compõe. Na comunidade científica, aceita-se por rádios programáveis dispositivos capazes de implementar através de *software* questões como modulação, filtros, correção de erros e compressão de dados, ou seja,

capazes de efetivar a comunicação conectados apenas aos acopladores de canal – conversor digital/analógico, conversor analógico/digital e estágios de potência – como visto na Figura 1.

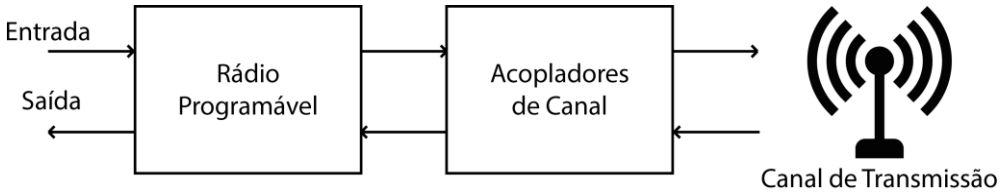


Figura 1 - Representação do modelo de um rádio programável

Os dispositivos de arranjo de portas programável em campo (FPGA, *Field Programmable Gate Array*) são, hoje, o que há de mais avançado no ramo de dispositivos lógicos programáveis e foram inicialmente comercializados pela empresa Xilinx Inc. no ano de 1983. Caracterizados por possuírem internamente blocos lógicos configuráveis, em que cada bloco contém capacidade computacional para implementar funções lógicas em forma de uma matriz bidirecional com chaves de interconexão para o roteamento, os FPGAs são ideais para a implementação de sistemas de transmissão em que se aplica as técnicas de multiplexação em frequência (SIQUEIRA, 2004).

A programação dos FPGAs se dá através do computador, por meio de *softwares* disponibilizados pelos fabricantes. A linguagem, denominada linguagem de descrição de *hardware* para circuitos integrados de altíssimas velocidades (VHDL, *Very High Speed Integrated Circuits Hardware Description Language*), foi originalmente desenvolvida na Agência de Projetos e Pesquisas Avançadas de Defesa (DARPA, *Defense Advanced Research Projects Agency*) do Departamento de Defesa dos Estados Unidos em meados da década de 1980, com o propósito de documentação de dispositivos lógicos que compunham os equipamentos vendidos às Forças Armadas norte-americanas e, em 1987, foi posta em domínio público, padronizada pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE, *Institute of Electrical and Electronis Engineers*), com novas tarefas, tais como a síntese, simulação, testes, verificação e compilação dos programas desenvolvidos (KAFIG, 2011).

Dentre as técnicas de transmissão por multiplexação na frequência, a que mais se destaca por sua eficiência e por ser amplamente utilizada é a multiplexação

por divisão em frequências ortogonais (OFDM, *Orthogonal Frequency-Division Multiplexing*). A técnica OFDM utiliza o conceito de múltiplas portadoras em diferentes frequências, moduladas e filtradas separadamente, ortogonais entre si para a transmissão de dados. A ortogonalidade entre as portadoras garante que não haja interferência entre elas e provê a redução da largura de banda total utilizada pelo sistema, permitindo elevadas taxas de transmissão (BHARDWAJ, GANGWAR, SONI, 2012). Um esboço do espectro na frequência de um sistema OFDM composto por cinco portadoras pode ser visto na Figura 2.

Ainda, o sinal que se propaga através de um sistema digital sofre diversas refrações, reflexões (*multipath*) e difrações, principalmente em comunicações sem fio ou até fibras ópticas e, assim, poderá sofrer com a interferência intersimbólica. A paralelização do fluxo de bits é uma medida para se compensar este efeito, porém, por vezes são necessárias outras soluções. A mais utilizada é a introdução de um intervalo de guarda no final (ou início) de cada símbolo OFDM transmitido que seja, no mínimo, igual ao máximo alargamento temporal do canal. Usualmente, o intervalo de guarda é preenchido com dados, uma cópia de parte do próprio símbolo, o que representa uma perda da eficiência da ocupação da banda disponível e de potência do sistema, porém, em sistemas reais, pode ser utilizado como ferramenta de sincronia entre transmissor e receptor, através da correlação de dados, e correção de erros (NETO, 2009).

Quando trata-se de transmissão de dados digitais através da técnica OFDM, a modulação mais aplicada no tratamento das portadoras é a modulação de amplitude em quadratura (QAM, *Quadrature Amplitude Modulation*). Nesta modulação, a mensagem digital a ser transmitida atua tanto sobre a fase quanto sobre a amplitude da portadora, ficando, assim, menos suscetível a ruídos e interferências quando comparada a outros modos de modulação (HANZO *et al*, 2004).

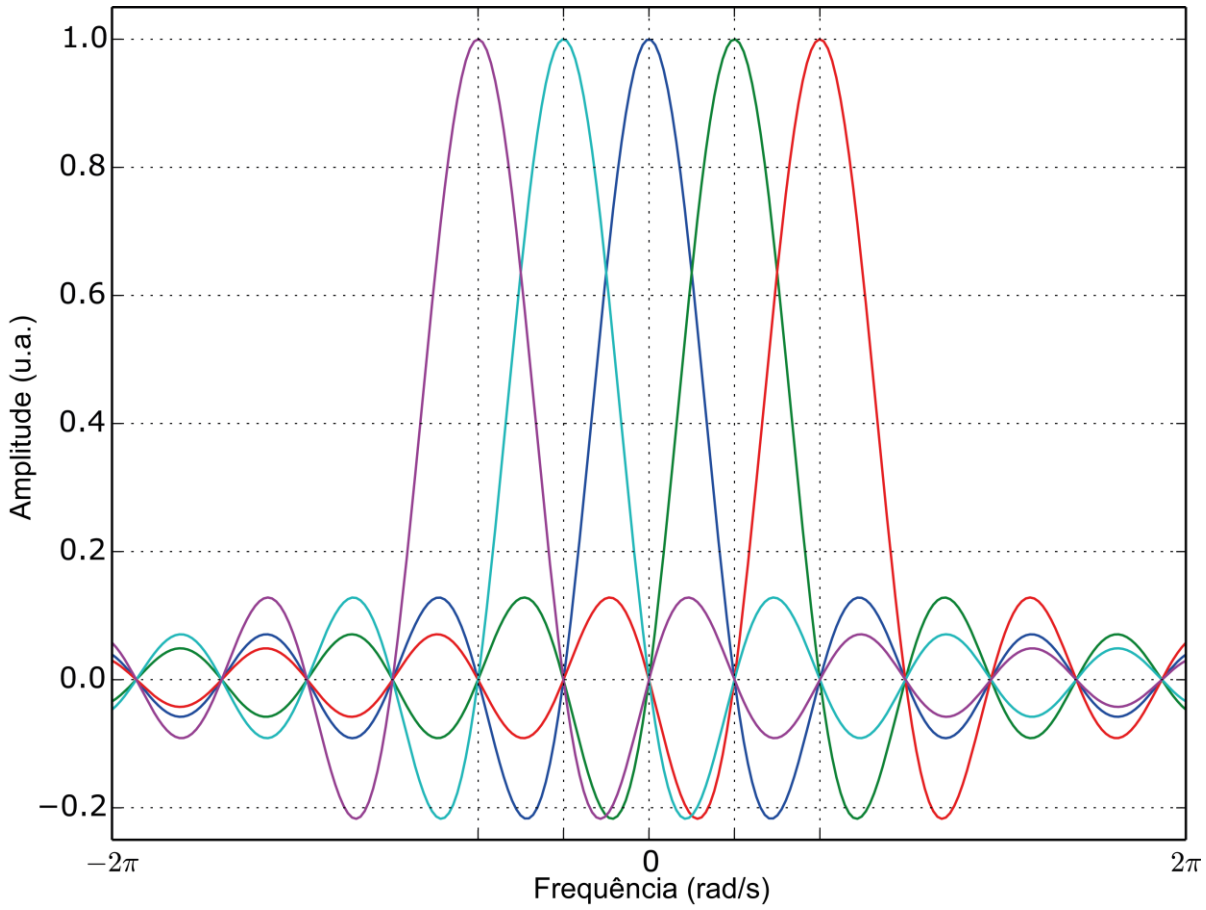


Figura 2 - Espectro na frequência de um sistema OFDM

1.1 OBJETIVO

O objetivo deste trabalho é a implementação em VHDL de um algoritmo de comunicação embarcado em um dispositivo lógico programável com a aplicação da técnica de multiplexação por divisão em frequências ortogonais, com as portadoras moduladas com QAM, sob a configuração *back-to-back*[1]. O diagrama de blocos do sistema pode ser visto na Figura 3.

[1] Um sistema é denominado *back-to-back* quando a saída do transmissor é acoplada diretamente à entrada do receptor, simulando um canal de transmissão ideal.

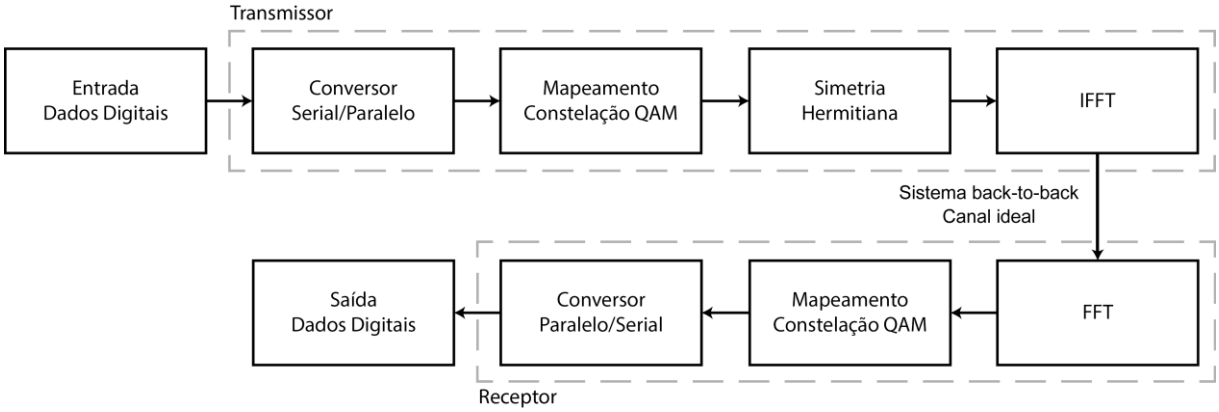


Figura 3 - Diagrama de blocos do sistema a ser implementado

A entrada de dados digitais atua como uma fonte dos dados a serem transmitidos pelo sistema. O conversor serial/paralelo tem como função a distribuição dos dados de entrada nos N canais utilizados na implementação OFDM, onde o comprimento da sequência binária atribuída a cada canal variará conforme a configuração da modulação QAM adotada. Os dados de cada canal, então, passam pelo mapeamento na constelação QAM, onde será gerado o fasor complexo que representa a sequência binária com os coeficientes de fase e quadratura. Aplica-se a simetria Hermitiana sobre a sequência de fasores complexos gerados pela modulação QAM para que possa ser utilizada a transformada inversa discreta de Fourier para consolidar a modulação dos dados. A saída da IFFT, no transmissor, é diretamente acoplada à FFT, no receptor, configurando-se como um sistema *back-to-back*. No receptor, efetua-se o processo inverso ao realizado no transmissor de forma a recuperar os dados transmitidos no bloco de saída de dados digitais.

Como em um sistema *back-to-back* não se considera os efeitos do canal de transmissão, o sistema não deverá apresentar problemas com interferência intersimbólica e, assim, faz-se desnecessário a implementação do intervalo de guarda.

1.2 ESTRUTURA DO TRABALHO

O corpo deste trabalho é dividido basicamente em três partes: conceitos fundamentais, desenvolvimento e resultados. Em conceitos fundamentais é feito uma análise matemática dos conceitos bases dos temas abordados pelo trabalho de forma a facilitar o entendimento quanto ao que se deve ser implementado e, principalmente,

as considerações feitas para que seja possível a implementação digital dos mesmos. Em desenvolvimento são descritos os métodos aplicados para a implementação de cada componente necessário, assim como os resultados individuais destes componentes, quando apropriado. Finalmente, em resultados são apresentados e discutidos os resultados obtidos a partir dos testes realizados com o sistema completo.

2 CONCEITOS FUNDAMENTAIS

Nesta seção, analisar-se-á matematicamente os conceitos fundamentais que envolvem o tema deste trabalho e suas considerações para a implementação digital dos mesmos.

2.1 O DISPOSITIVO FPGA

Um dispositivo FPGA consiste de um grande arranjo de blocos lógicos configuráveis contidos em um único circuito integrado. Cada bloco contém capacidade computacional para implementar funções lógicas e é possível realizar roteamento entre eles. Diferentemente dos dispositivos lógicos simples (SPLD, *Simple Programmable Logic Device*), que possuem planos compostos por portas lógicas “e” e “ou”, os dispositivos FPGAs possuem três componentes básicos: os blocos lógicos, os blocos de entrada e saída e as chaves de interconexão. Os blocos lógicos formam uma matriz bidirecional e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos. Esses canais de roteamento possuem chaves de interligação programáveis que permitem conectar os blocos lógicos de maneiras convenientes, em função da necessidade de cada projeto (BROWN, ROSE, 2002). O diagrama de blocos que representa a estrutura de um FPGA pode ser visto na Figura 4, onde os blocos denominados I/O representam os blocos de entrada e saída e as linhas entre os blocos representam as chaves de interligação.

No interior de cada bloco lógico de um dispositivo FPGA existem vários modos possíveis para implementação de funções lógicas. O mais utilizado comercialmente é através dos blocos de memórias LUT (*Look-Up Table*). Esse tipo de bloco lógico contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas, onde cada célula é capaz de armazenar um único valor lógico, zero ou um. Tipicamente, tais blocos lógicos possuem de quatro a cinco entradas, permitindo endereçar 16 ou 32 células de armazenamento, respectivamente.

Quando um circuito lógico é sintetizado em um FPGA, os blocos lógicos são programados para realizar as funções necessárias e os canais de interligação são estruturados de forma a realizar o roteamento necessário entre os blocos lógicos.

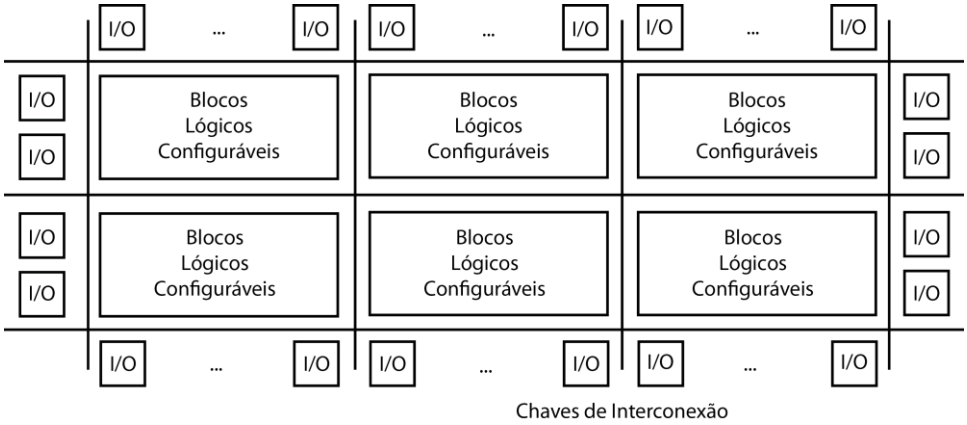


Figura 4 - Diagrama de blocos interno de um dispositivo FPGA

As células de armazenamento dos LUTs de um dispositivo FPGA são voláteis, o que implica perda de conteúdo armazenado quando o dispositivo é desligado. Desta forma, o FPGA deve ser reprogramado, geralmente, através de memórias Flash e EEPROM (*Electrically Erasable Programmable Read-Only Memory*), toda vez que for energizado.

Comumente caracteriza-se um dispositivo FPGA devido à complexidade apresentada em seus blocos lógicos, denominada granularidade (OLIVEIRA, 2011). Denomina-se grão a menor unidade configurável que compõe o FPGA e classifica-se em grande quando o dispositivo possui em seus blocos lógicos unidades lógicas aritméticas, pequenos microprocessadores e memórias, médio, quando os blocos do dispositivo possuem dois ou mais LUTs e dois ou mais *flip-flops*[2] e pequeno quando os blocos possuem unidades simples de duas entradas, multiplexadores 4x1 e um *flip-flop*.

2.2 A LINGUAGEM DE DESCRIÇÃO VHDL

VHDL é uma linguagem de descrição de *hardware* com um vocabulário rico o suficiente para abranger uma gama de projetos. Herda da linguagem Ada[3] as

[2] *Flip-flop* é o bloco primário utilizado na construção de unidades de armazenamento de dados em dispositivos eletrônicos, sendo o componente principal das memórias.
[3] Linguagem de programação Ada: http://www.adacore.com/adaanswers/about/ada

definições de tipo, a forte verificação de tipos e a sobrecarga de operadores e programas. Para separar as estruturas de programação de apoio do domínio do problema, VHDL também herda da Ada os conceitos de pacotes, popularizado por seu termo em inglês *packages*. Estes pacotes permitem a reutilização de rotinas e definições comuns. Enquanto Ada usa um elemento para descrever a simultaneidade em seu código, VHDL fornece vários elementos simultâneos que se aproximam mais do real funcionamento do circuito descrito, incluindo a descrição de hierarquias. VHDL fornece construções específicas para definir as estruturas ou conectividade interna de projetos de *hardware* hierárquicos, permitindo ao projetista configurar seu projeto com diferentes arquiteturas alternativas para seus subprojetos (COHEN, 1995).

Como resultado desta flexibilidade, VHDL é utilizada em várias fases do processo de descrição e verificação de circuitos. Estas etapas são geralmente classificadas como níveis de representação do aspecto de projeto. Há muitas interpretações, definições e opiniões sobre os níveis de representação, porém os principais são:

- Nível de sistema: Existem várias interpretações sobre o que é o sistema, dentre as quais o modelo estatístico. Este modelo representa símbolos transmitidos através de uma rede de Petri[4] para emular transações que fazem exigências sobre os recursos do sistema. O objetivo deste tipo de simulação é avaliar a eficiência de um projeto, em termos de tarefas impostas aos recursos. Esses recursos poderiam ser barramentos, processadores, memórias, etc. Outra interpretação da modelagem em nível de sistema incluem o nível algorítmico que define os algoritmos para o qual define o protocolo de comunicação entre as unidades.
- Nível de placa: Pode ser também referida como nível de sistema porque a placa representa normalmente uma função do subsistema. Simulação no nível de placa é normalmente realizada utilizando

[4] Uma rede de Petri é uma das várias representações matemáticas para sistemas distribuídos discretos.

componentes em VHDL, modelados em vários níveis, que simula o ambiente do sistema e interfaces com componentes.

- Nível de definição de instrução: Este nível é normalmente usado para definir e simular as operações de instruções de um processador. As instruções são buscadas e decodificadas com base no seu formato. As execuções são normalmente realizadas utilizando os operadores algébricos e lógicos.
- Nível de transferência de registro (RTL, *Register Transfer Level*): Este nível descreve os registros e as equações booleanas combinacionais entre eles. Ele é frequentemente usado como uma entrada para um sintetizador de lógica. Ele também é usado para definir máquinas de estado para projetar controladores onde os registros de estado podem ser tanto explicitamente como implicitamente definidos em função das declarações e estilo de codificação.
- Nível estrutural: Este nível representa as estruturas e interconexões de um projeto. Este nível pode ser gerado manualmente utilizando um editor de texto ou automaticamente a partir de uma ferramenta que converte um esquema de circuito em uma estrutura VHDL.
- Nível de porta: Este nível descreve as interconexões estruturais de elementos de baixo nível (portas e *flip-flops*) de um projeto. É geralmente uma saída VHDL de um sintetizador e é usado tanto como documentação na lista de conexões, como modelo VHDL da definição estrutural de um projeto para simulação.

2.3 A TÉCNICA DE MULTIPLEXAÇÃO OFDM

Os primeiros dispositivos utilizados para telecomunicações utilizavam a técnica de modulação em portadora única (SCM, *Single-Carrier Modulation*), que consiste em transmitir os dados de forma serial através de uma única portadora. Nesta técnica, o espectro de cada símbolo ocupava toda a largura de banda disponível para a transmissão (LOURENÇO, 2011). Porém, um dos principais problemas para os sistemas SCM é quando a resposta em frequência do canal não é linear, seja devido a perdas ou a ruídos, exigindo a necessidade de utilizar equalizadores. Dependendo,

ainda, da resposta em frequência do canal, estes equalizadores poderiam se tornar extremamente complexos. Nestes casos, havia a necessidade da utilização de equalizadores adaptativos, que exigia a transmissão de uma sequência de treinamento para a convergência do receptor e sincronização do sistema (Siqueira, 2004).

Como solução aos problemas apresentados nos sistemas SCM, surgiu a técnica de modulação em múltiplas portadoras (MCM, *Multi-Carrier Modulation*), que consiste em dividir o canal de transmissão em múltiplos canais menores, utilizando uma portadora, com frequência própria, em cada um dos canais. Dividindo-se em múltiplos canais, consegue-se fazer com que a resposta em frequência em cada canal seja praticamente linear (Figura 5) e, com isso, consegue-se utilizar equalizadores extremamente simples.

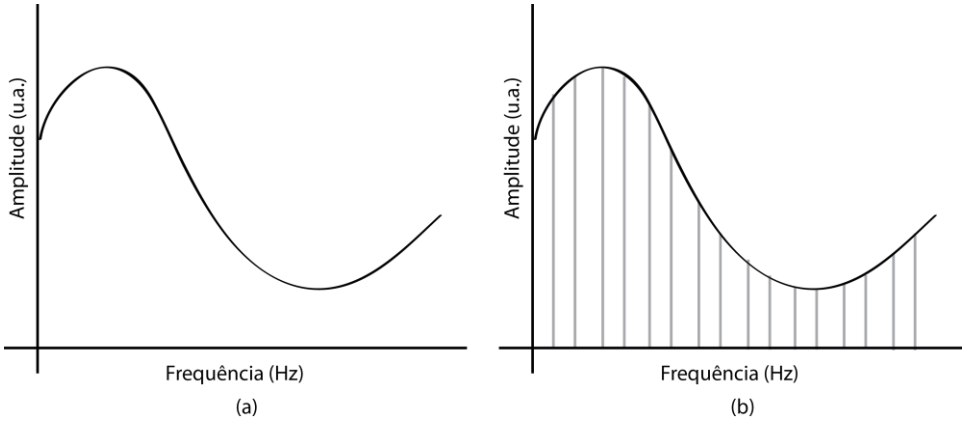


Figura 5 - Resposta em frequência de um canal (a) não linear (b) multiplexado na frequência

A principal técnica dentre todas as que implementam a MCM é a denominada OFDM, que consiste na divisão da frequência, onde múltiplos sinais são enviados em frequências diferentes, de forma que sejam ortogonais entre si. A técnica OFDM é uma evolução da técnica de multiplexação por divisão em frequências (FDM, *Frequency-Division Multiplexing*), que consiste, também, na transmissão por divisão em frequências, porém, implementando um intervalo de guarda entre as portadoras, de forma a evitar a interferência entre elas. Problema este que não ocorre na OFDM devido à ortogonalidade das portadoras, como pode ser visto na Figura 6.

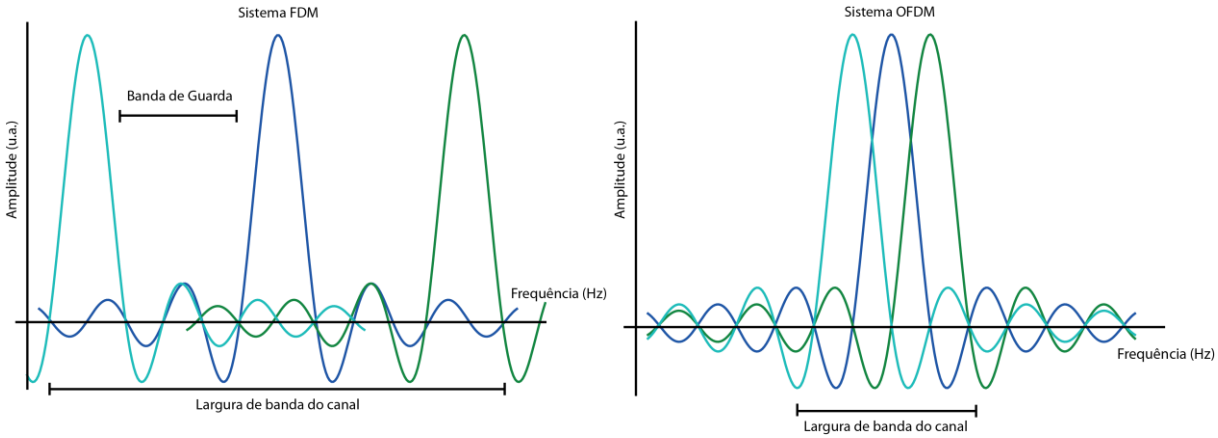


Figura 6 - Espectro de frequências de sistemas FDM e OFDM

Apesar de no espectro OFDM os canais estarem claramente sobrepostos, estes não causam interferência entre si devido à ortogonalidade, ou seja, o produto interno entre as portadoras é zero (VAN NEE, 2000).

Porém, o uso da técnica OFDM tem como consequência que o receptor tenha de sincronizar muito bem no tempo e na frequência, de modo a garantir a ortogonalidade das portadoras. Para ajudar na sincronização, costuma-se utilizar a técnica de introdução de portadoras piloto, que são divididas entre as portadoras de dados, como pode ser visto na Figura 7. Estas portadoras piloto servem como um padrão conhecido pelo receptor para ajudá-lo na sincronização.

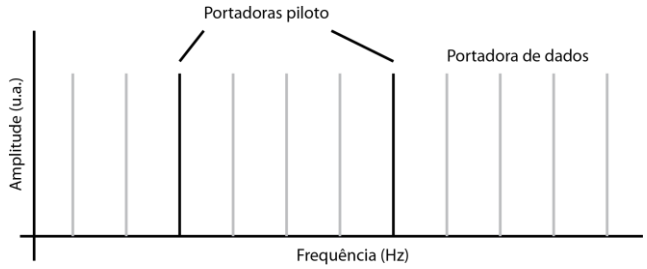


Figura 7 - Representação na frequência das portadoras piloto

Outro problema a ser resolvido, que se agrava nos sistemas de telecomunicações sem fio, é que o sinal que chega no receptor não é apenas o sinal de linha de vista (LoS, *Line of Sight*), que sai do transmissor e vai diretamente ao receptor, mas a soma de várias reflexões que ocorrem durante a transmissão, com diferentes ganhos e diferentes atrasos em cada percurso. Efeito este que denomina-se multi-percurso, ou, no inglês, *multipath*.

Dentre as consequências geradas pelo efeito de multi-percurso, aliado à resposta não linear do canal, cita-se o efeito de desvanecimento, em que cada portadora pode sofrer atenuações diferentes. Atenuação esta que pode variar com o tempo, o local e a própria frequência. Ainda, dá-se o nome de desvanecimento seletivo de frequência se a atenuação de uma ou mais das portadoras que compõe o sinal OFDM apresenta uma atenuação muito maior que as outras componentes. Se no caso de sistemas FDM, ocorrer o desvanecimento seletivo de frequência leva à inevitável perda do símbolo, nos sistemas OFDM, poucas portadoras são afetadas e, consequentemente, perde-se só poucos bits, que, se aliado à codificação e ao *interleaving*[5] de dados antes do processo de modulação, o efeito é minimizado. O efeito de desvanecimento seletivo de frequência pode ser evitado aumentando-se o número de portadoras.

Ao se transmitir um símbolo MCM imediatamente seguido do outro a ortogonalidade não será mais mantida, pois, devido ao efeito de multi-percurso, o final de um símbolo interferirá em *v* amostras do início do próximo símbolo, a denominada interferência intersimbólica. Para resolver este problema basta introduzir um período de guarda de *v* amostras entre um símbolo e outro, de tal forma que o final de um símbolo não interfira no próximo. Há, basicamente, três formas de se implementar este período de guarda (BINGHAM, 2000):

- Adicionar as últimas *v* amostras no início do sinal, fazendo assim um prefixo cíclico;
- Adicionar as primeiras *v* amostras no final do sinal, fazendo assim um sufixo cíclico;
- Não transmitir dados no intervalo de guarda e no receptor adicionar as últimas *v* amostras ao início;

A primeira das três é a mais utilizada e é representada na Figura 8.

[5] *Interleaving* é uma técnica onde os bits de um bloco de dados são entrelaçados com o uso de uma matriz. O entrelaçamento nada mais é que a mudança de ordem dos bits, com o objetivo de recuperar a mensagem mesmo com a ocorrência de ruídos devido aos desvanecimentos de curto prazo.

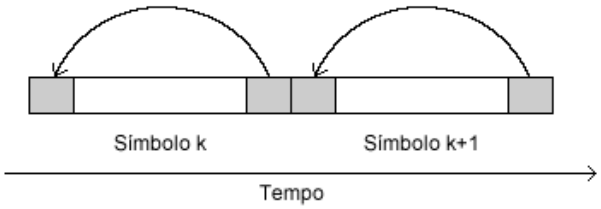


Figura 8 - Representação gráfica do prefixo cíclico

É natural que a eficiência na utilização do canal será degradada, pois está sendo introduzido um período de *v* amostras em que não se transmite dados úteis, porém, a melhoria que o período de guarda introduz é muito compensador, sendo praticamente impossível utilizar sistemas MCM sem ele, principalmente por possibilitar a sincronização entre transmissor e receptor através da correlação de dados. Com a utilização de equalizadores, pode-se alterar a resposta em frequência impulsiva do canal e com isso diminuir o tamanho do período de guarda, mas deve-se buscar o compromisso entre a eficiência e a complexidade do sistema na hora do projeto.

2.4 A MODULAÇÃO QAM

Dentre as técnicas de modulação de dados digitais, a mais utilizada atualmente é a modulação de amplitude em quadratura, QAM, pois herda as principais vantagens de duas predecessoras, a modulação em amplitude (ASK, *Amplitude-Shift Keying*) e a modulação em fase (PSK, *Phase-Shift Keying*), como mostra a Figura 9.

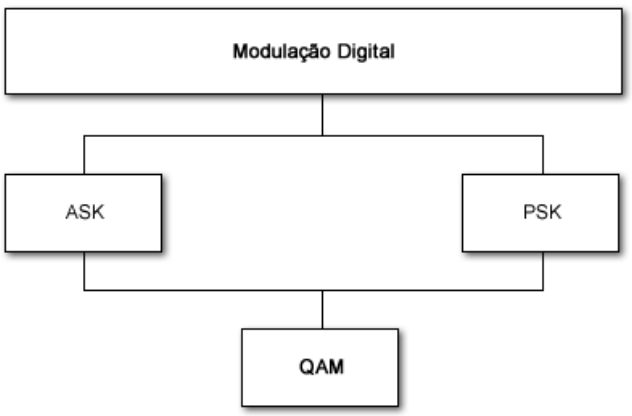


Figura 9 - Diagrama de blocos da modulação QAM

O sinal QAM é composto por duas portadoras de mesma frequência e diferença de fase de 90°, ou seja, as funções matemáticas cosseno e seno, ambas moduladas em amplitude, nos níveis de tensão *a* e *b*, respectivamente, determinados pela entrada binária, conforme mostra a Figura 10. Assim sendo, os níveis de tensão *a* e *b* possuirão valores discretos bem definidos, limitando o tamanho do alfabeto de transmissão.

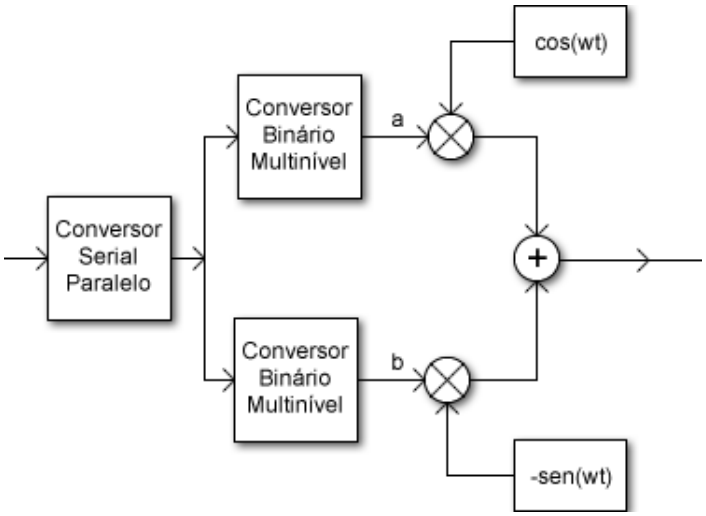


Figura 10 - Diagrama de blocos do modulador QAM

Devido à ortogonalidade das portadoras, é possível utilizar a mesma banda de frequência para a transmissão, em que cada portadora transmite metade dos bits de entrada. Por exemplo, em um modulador QAM de quatro bits, dois bits serão transmitidos pelo conversor em fase e dois serão transmitidos pelo conversor em quadratura (SIQUEIRA, 2004).

Percebe-se, ainda, através da Figura 10, que o símbolo transmitido por cada modulador QAM possui a seguinte forma matemática:

$$\begin{aligned} s(t) &= \mathrm{acos}(\omega t) - bsen(\omega t) \\ &= \sqrt{a^2 + b^2}\cos\left(\omega t - arctg\left(\frac{b}{a}\right)\right) \end{aligned} \qquad (1)$$

Pode-se ainda representar fasorialmente a equação (1), obtendo:

$$s(t) = \sqrt{a^2 + b^2} \angle - arctg\left(\frac{b}{a}\right)$$
$$= a + jb \qquad (2)$$

Onde j denota a unidade imaginária, definida por $j = \sqrt{-1}$. Como a e b só existem para alguns níveis discretos, pode-se indicar através de um diagrama todos os possíveis símbolos, conforme a equação (2), a que denomina-se diagrama de constelação (HANZO *et al*, 2004). A Figura 11 representa o diagrama de constelação de uma modulação QAM de quatro bits. A localização dos pontos e a forma do diagrama podem variar conforme a necessidade do sistema.

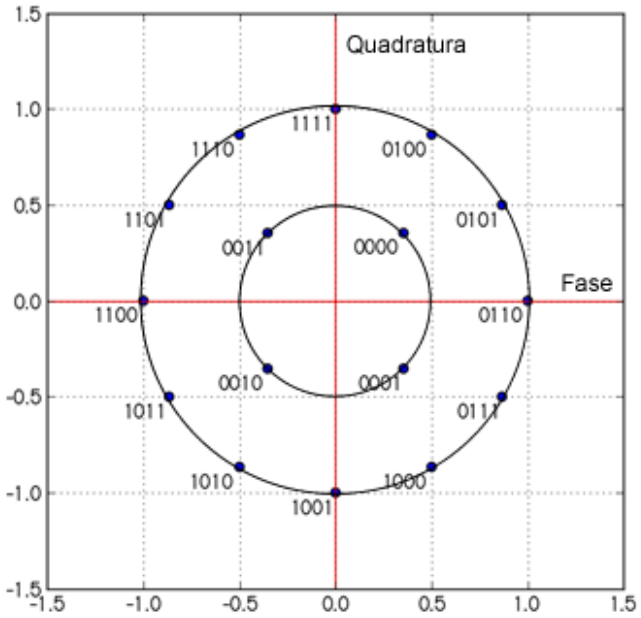


Figura 11 - Exemplo de diagrama de constelação para 16-QAM

É notável que quanto mais pontos o diagrama de constelação possuir, alcança-se uma taxa maior de transmissão, uma vez que cada símbolo transporta um número maior de bits, para uma mesma potência de transmissão. No entanto, quanto menor o número de pontos do diagrama, maior será a distância euclidiana entre eles, possibilitando uma melhor qualidade de serviço (QoS, *Quality of Service*), pois dificulta erros de interpretação por parte do receptor ao analisar o símbolo. Outra técnica que pode ser implementada no intuito de melhorar a comunicação é a denominada mapeamento de Gray, onde as sequências binárias que representam dois pontos adjacentes do diagrama de constelação diferem entre si em apenas um bit. Assim, se

o receptor ainda interpretar o símbolo recebido de forma equivocada, apenas um bit dentre todos da sequência estará errado, diminuindo a taxa de erro de bits (BER, *Bit Error Rate*) (HANZO *et al*, 2004).

2.5 MODULAÇÃO ATRAVÉS DA TRANSFORMADA DE FOURIER

Sabe-se, a partir da equação (1) que o sinal modulado QAM pode ser matematicamente expresso por:

$$s(t) = a\cos(\omega t) - bsen(\omega t),$$

onde a e b são, respectivamente, os termos em fase e quadratura do sinal e ω a frequência do sinal.

Sabe-se, também, que um símbolo transmitido pelo sistema OFDM é composto por vários sinais de frequências distintas sobrepostos e, assim, pode ser representado pela equação (3), onde N representa o número de sinais que compõe o símbolo

$$s(t) = \sum_{k=0}^{N-1} [a_k \cos(\omega_k t) - b_k sen(\omega_k t)]. \quad (3)$$

Para que seja possível a implementação digital, faz-se necessário a discretização da equação (3), substituindo nesta as seguintes relações:

$$\omega_k = \frac{2\pi k}{NT}, \quad t_n = nT,$$

onde T é o período do símbolo OFDM gerado. Desta forma, obtém-se

$$s[n] = \sum_{k=0}^{N-1} \left[a_k \cos\left(2\pi \frac{nk}{N}\right) - b_k sen\left(2\pi \frac{nk}{N}\right)\right]. \quad (4)$$

Para facilitar o entendimento dos passos seguintes, considera-se dois números complexos, z_1 e z_2, expressos por:

$$z_1 = x_1 + jy_1,$$
$$z_2 = x_2 + jy_2.$$

Sabe-se que o produto entre estes dois números complexos é dado por:

$$\begin{aligned} z_3 &= z_1 z_2, \\ &= (x_1 + jy_1)(x_2 + jy_2), \\ &= (x_1 x_2 - y_1 y_2) + j(x_1 y_2 + x_2 y_1). \end{aligned}$$

É visto que a parte real do resultado da multiplicação, z_3, possui a mesma forma do termo da somatória da equação (4), então pode-se relacioná-los por:

$$\Re(z_3) = a_k \cos\left(2\pi \frac{nk}{N}\right) - b_k sen\left(2\pi \frac{nk}{N}\right),$$
$$x_1 x_2 - y_1 y_2 = a_k \cos\left(2\pi \frac{nk}{N}\right) - b_k sen\left(2\pi \frac{nk}{N}\right),$$

de onde obtém-se:

$$x_1 = a_k,$$
$$x_2 = \cos\left(2\pi \frac{nk}{N}\right),$$
$$y_1 = b_k,$$
$$y_2 = \operatorname{sen}\left(2\pi \frac{nk}{N}\right).$$

Percebe-se, então, que o número complexo z_1 representa os coeficientes de fase e quadratura do canal k na modulação QAM e, a partir deste ponto, será

denominado d_k. O número complexo z_2 representa os fatores da modulação e pode ser representado em sua forma exponencial[6] obtendo-se, assim,

$$s[n] = \sum_{k=0}^{N-1} \left[\Re \left(z_3 = d_k e^{j2\pi \frac{nk}{N}} \right) \right]. \quad (5)$$

Das propriedades dos números complexos, sabe-se que a parte real de um número complexo z qualquer é numericamente igual a metade da soma dele com seu respectivo conjugado[7]

$$\Re(z) = \frac{1}{2}(z + z^*).$$

E, desta forma, pode-se reescrever a equação (5) para se obter

$$s[n] = \frac{1}{2} \left(\sum_{k=0}^{N-1} d_k e^{j2\pi \frac{nk}{N}} + \sum_{k=0}^{N-1} d_k^* e^{-j2\pi \frac{nk}{N}} \right). \quad (6)$$

Aplica-se a simetria Hermitiana (Anexo A) e a propriedade de simetria da transformada discreta de Fourier abaixo resumidas na equação (6)

$$\begin{aligned} S[0] = S[N+1] &= 0, \\ S[k] &= d_k \qquad \text{para } k \in [1, 2, \dots, N], \\ S[2N+2-k] &= d_k^*, \end{aligned}$$

obtém-se como resultado

[6] A forma exponencial de um número complexo pode ser obtida através da fórmula de Euler.
[7] O número complexo conjugado de um valor z é o seu valor simétrico no plano complexo em relação ao eixo real.

$$s[n] = \frac{1}{2}\left(\sum_{k=0}^{N-1} S[k]e^{j2\pi\frac{nk}{N}} + \sum_{k=N}^{2N+1} S[k]e^{j2\pi\frac{nk}{N}}\right),$$
$$= \frac{1}{2}\sum_{k=0}^{2N+1} S[k]e^{j2\pi\frac{nk}{N}}. \quad (7)$$

Observa-se, então, que o sinal de saída do sistema OFDM é proporcional à transformada discreta inversa de Fourier e, portanto, pode ser implementada digitalmente através do algoritmo da transformada rápida de Fourier (FFT).

2.6 TRANSFORMADA RÁPIDA DE FOURIER (FFT)

Para a análise da transformada rápida de Fourier, pode-se utilizar a terminologia introduzida por (BURRUS, 1977), que classifica todos os algoritmos FFT simplesmente por diferentes mapas (multidimensionais) de índices das sequências de entrada e saída. Esta terminologia baseia-se em uma transformação da DFT de comprimento N

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk} \quad (8)$$

em uma representação multidimensional $N = N_1N_2 \dots N_L$, onde L é o número de dimensões e $W_N^{\alpha} = e^{-j\frac{2\pi}{N}\alpha}$ representa o fator de rotação. Geralmente, isto é suficiente apenas para discutir o caso de dois fatores, $N = N_1N_2$, visto que dimensões maiores podem ser definidas de forma iterativa substituindo novamente um destes fatores.

Mapeia-se os índices de tempo n conforme a equação[8]

$$n = (An_1 + Bn_2) mod\ N \quad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1, \end{cases} \quad (9)$$

[8] mod representa o resto da divisão entre os operandos.

onde $N = N_1 N_2$, sendo $N_1, N_2 \in \mathbb{Z}$, e $A, B \in \mathbb{Z}$ são constantes que variam conforme o algoritmo em questão. Aplicando este mapeamento de índice, define-se uma relação de duas dimensões $f: \mathbb{C}^N \rightarrow \mathbb{C}^{N_1 \times N_2}$ da sequência, como descrito abaixo

$$\hat{x}[n_1, n_2] = x[(An_1 + Bn_2)\ mod\ N]. \quad (10)$$

Analogamente, pode-se fazer a mesmo mapeamento de índice no domínio da frequência, no índice k, conforme as equações

$$k = Ck_1 + Dk_2\ mod\ N \quad \begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1' \end{cases} \quad (11)$$

$$\hat{X}[k_1, k_2] = X[(Ck_1 + Dk_2)\ mod\ N], \quad (12)$$

onde $C, D \in \mathbb{Z}$ e são constantes que variam conforme o algoritmo em questão. Como a DFT é bijetora, deve-se escolher A, B, C e D de forma que o mapeamento de índices seja única, em outras palavras, seja uma projeção bijetora. Em (BURRUS, 1977) são descritas as condições para se determinar os valores destes coeficientes de acordo com os valores de N_1 e N_2 para que o mapeamento de índices seja bijetor.

Um ponto importante na distinção dos vários algoritmos da FFT, de acordo com (MEYER-BAESE, 2007), é saber se é permitido os coeficientes N_1 e N_2 possuírem fatores comuns ou se devem ser primos entre si[9]. Comumente, os algoritmos que permitem os coeficientes possuírem fatores comuns são denominados algoritmos de fatores comuns (CFA, *Common-Factor Algorithms*) e os que exigem que os coeficientes sejam primos entre si são denominados algoritmos de fatores primos (PFA, *Prime-Factor Algorithms*). O principal algoritmo dentre todos, o algoritmo de Cooley-Tukey, é classificado como CFA e será o adotado neste trabalho.

[9] Dois números são primos entre si quando o maior divisor comum entre eles é o número um.

2.6.1 O Algoritmo de Cooley-Tukey

O algoritmo de Cooley-Tukey é o principal dentre os algoritmos da FFT pois permite qualquer comprimento N da transformada, sendo $N \in \mathbb{N}$, porém, os mais comuns são os que possui o valor N como uma potência de r, i.e. $N = r^v$, para $r, v \in \mathbb{N}$, denominados radix-r.

Aplicando o mapeamento de índices descrito na equação (9) à transformação sugerida por Cooley-Tukey (e, anteriormente, por Gauss), obtém-se que $A = N_2$ e $B = 1$ (MEYER-BAESE, 2007). Ou seja:

$$n = N_2 n_1 + n_2 \quad \begin{cases} 0 \leq n_1 \leq N_1 - 1 \\ 0 \leq n_2 \leq N_2 - 1 \end{cases}. \tag{13}$$

De forma análoga, calcula-se o mapeamento dos índices na frequência, obtendo $C = 1$ e $D = N_1$. Ou seja:

$$k = k_1 + N_1 k_2 \quad \begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1 \end{cases}. \tag{14}$$

Pode-se, agora, reescrever o fator de rotação W_N^{nk} substituindo os resultados das equações (13) e (14), obtendo:

$$W_N^{nk} = W_N^{N_2 n_1 k_1 + N_1 N_2 n_1 k_2 + n_2 k_1 + N_1 n_2 k_2}. \tag{15}$$

Como W é da ordem de $N = N_1 N_2$, então tem-se que $W_N^{N_1} = W_{N_2}$ e $W_N^{N_2} = W_{N_1}$, o que simplifica a equação (15) para

$$W_N^{nk} = W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2}. \tag{16}$$

Substitui-se, então, a equação (16) na DFT definida na equação (8)

$$X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \underbrace{\left(W_N^{n_2 k_1} \underbrace{\sum_{n_1=0}^{N_1-1} x[n_1, n_2]\, W_{N_1}^{n_1 k_1}}_{\overline{X_1}} \right)}_{\bar{x}[n_2, k_1]} \tag{17}$$

$$= \underbrace{\sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2}\ \bar{x}[n_2, k_1]}_{\overline{X_2}}, \tag{18}$$

onde $\overline{X_1}$ representa uma transformada de N_1 pontos e $\overline{X_2}$ representa uma transformada de N_2 pontos.

Portanto, a partir das equações (17) e (18), pode-se descrever o algoritmo para o cálculo da FFT por Cooley-Tukey, para uma transformada de comprimento $N = N_1 N_2$, conforme o quadro 1.

Passo	Descrição
1	Calcular a transformação de índice da sequência de entrada conforme a equação (11)
2	Calcular as N_2 transformadas de comprimento N_1.
3	Multiplicar os fatores de rotação $W_N^{n_2 k_1}$ nos resultados obtidos no passo 2.
4	Calcular as N_1 transformadas de comprimento N_2.
5	Calcular a transformação de índice da sequência de saída conforme a equação (12).

Quadro 1 - Descrição do algoritmo de Cooley-Tukey

2.6.2 O Algoritmo Cooley-Tukey Radix-r

Como supracitado, um dos principais diferenciais do algoritmo de Cooley-Tukey perante os outros algoritmos FFT é o fato de possibilitar transformadas de comprimento N arbitrários. Porém, os mais populares são os que possuem N como um número potência de dois (radix-2) ou potência de quatro (radix-4), pois permitem simplificações nas multiplicações necessárias para a sua computação.

Para o algoritmo radix-2 com S estágios, o mapeamento de índices pode ser escrito como as equações

$$n = 2^{S-1} n_1 + 2^{S-2} n_2 + \cdots + 2 n_{S-1} + n_S, \tag{19}$$

$$k = k_1 + 2k_2 + \cdots + 2^{S-2}k_{S-1} + 2^{S-1}k_S. \quad (20)$$

O número de estágios S para os algoritmos radix-r pode ser calculado como $\log_r N$ e para cada grupo, ocorre o mesmo fator de rotação. Um exemplo do gráfico de fluxo desta transformada, para radix-2 com oito pontos, é mostrada na Figura 12.

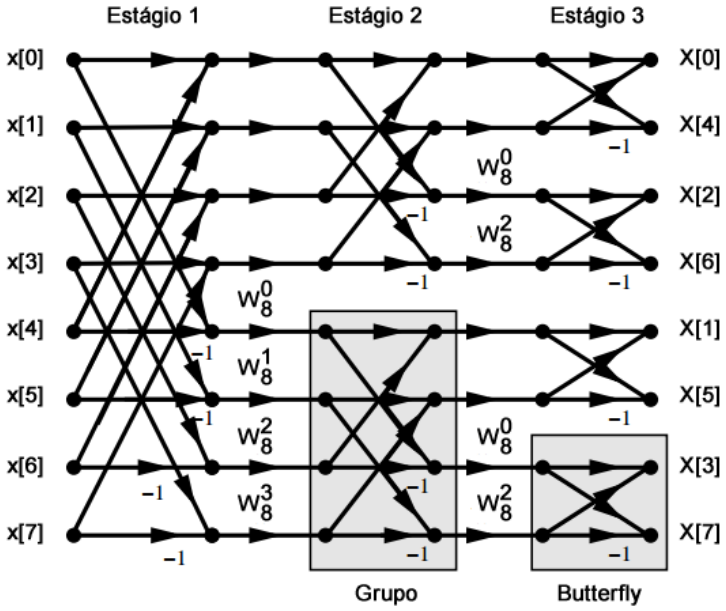


Figura 12 - Algoritmo de Cooley-Tukey radix-2 para 8 pontos

Uma das vantagens do algoritmo radix-2 é facilidade de implementação do mapeamento de índices, apresentado pelas equações (19) e (20), pois resume-se apenas a inversão de bits, porém, independente de qual algoritmo for implementado, há a necessidade de conhecer os fatores de rotação (*twiddle factors*) para efetivar a transformada e, para isso, é necessário conhecer os valores das funções trigonométricas seno e cosseno dos ângulos de rotação.

Basicamente, há dois métodos de se contornar este problema. O primeiro é calcular manualmente todos os valores necessários e armazená-los em memória, o que faz com que o cálculo da *butterfly* seja mais rápido, porém, restrito ao número de pontos, exigindo recalculá-los caso o comprimento da transformada seja alterado. O segundo método é efetuar o cálculo das funções trigonométricas no *hardware*, o que pode deixar o processamento da FFT um pouco mais lento, mas será independente

do comprimento da transformada. Uma forma de se implementar este cálculo é detalhada na próxima seção.

2.7 CÁLCULO DAS FUNÇÕES TRIGONOMÉTRICAS

Para calcular os valores das funções trigonométricas pode-se utilizar o algoritmo CORDIC (*Coordinate Rotation Digital Computer*), pois este baseia-se na rotação planar de um vetor, Figura 13, onde o vetor $P'(x', y')$ representa o vetor $P(x, y)$ rotacionado em um ângulo θ.

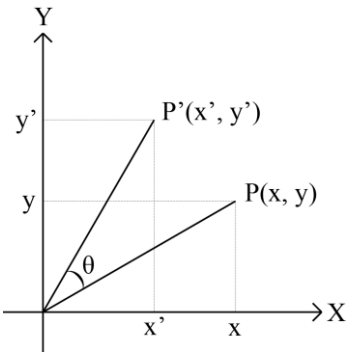


Figura 13 - Rotação de vetor calculada no algoritmo CORDIC

Matematicamente, pode-se representar este cálculo através da matriz de rotação, onde considera-se $P(x, y) = (x_i, y_i)$ e $P'(x', y') = (x_j, y_j)$, tem-se a equação

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}. \quad (21)$$

É facilmente visto que, quando aplicado como vetor de entrada o vetor unitário $(1, 0)$, o vetor de saída será dado por $(cos(\theta), sen(\theta))$ e, devido a isto, pode ser utilizado para calcular os valores das funções trigonométricas.

Pode-se, ainda, reduzir o valor do ângulo de rotação para facilitar sua implementação digital de forma a obter a rotação completa desejada através de múltiplas rotações sucessivas, em um processo iterativo.

Considera-se, então, um ângulo de rotação $\theta_\lambda \in \left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$ para reescrever a matriz de rotação conforme expresso em

$$\begin{pmatrix} x_{\lambda+1} \\ y_{\lambda+1} \end{pmatrix} = \begin{pmatrix} cos(\theta_\lambda) & -sen(\theta_\lambda) \\ sen(\theta_\lambda) & cos(\theta_\lambda) \end{pmatrix} \begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix}, \tag{22}$$

onde $\sum_\lambda \theta_\lambda = \theta$ e o vetor $(x_{\lambda+1}, y_{\lambda+1})$ representa o vetor obtido a partir da rotação em um ângulo θ_λ do vetor (x_λ, y_λ) na iteração λ.

Pode-se reduzir o número de multiplicações necessárias ao colocar o termo $cos(\theta_\lambda)$ em evidência, pois, assim, a diagonal da matriz torna-se unitária, obtendo

$$\begin{pmatrix} x_{\lambda+1} \\ y_{\lambda+1} \end{pmatrix} = cos\theta_\lambda \begin{pmatrix} 1 & -tg(\theta_\lambda) \\ tg(\theta_\lambda) & 1 \end{pmatrix} \begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix} \tag{23}$$

Simplifica-se ainda mais o processo quando seleciona-se um ângulo θ_λ de modo que sua tangente seja proporcional a uma potência negativa de dois, ou seja, $\theta_\lambda = \mathrm{tg}^{-1}(1/2^\lambda)$, pois, assim, como visto na equação (24)[10], a multiplicação pelo valor da tangente do ângulo θ_λ torna-se uma divisão por dois, o que, digitalmente, pode ser implementada por uma simples rotação de bits.

$$\begin{pmatrix} x_{\lambda+1} \\ y_{\lambda+1} \end{pmatrix} = \cos\left(tg^{-1}\left(\frac{1}{2^\lambda}\right)\right) \begin{pmatrix} 1 & -sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) \\ sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) & 1 \end{pmatrix} \begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix}. \tag{24}$$

Desta forma, representa-se a rotação completa através de

$$\begin{pmatrix} x_\rho \\ y_\rho \end{pmatrix} = \underbrace{\prod_{\lambda=0}^{\rho} cos\left(tg^{-1}\left(\frac{1}{2^\lambda}\right)\right)}_{\Lambda} \prod_{\lambda=0}^{\rho} \left[\begin{pmatrix} 1 & -sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) \\ sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) & 1 \end{pmatrix} \begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix} \right], \tag{25}$$

onde ρ representa o número de iterações utilizadas com $\rho \in \mathbb{N}$.

Como pode ser visto na Figura 14, o primeiro termo da equação (25), produto de cossenos, é uma função convergente quando utilizado um número suficiente de

[10] A função $sign$ determina o sinal de um número, retornando 1 quando positivo, 0 quando nulo e -1 quando negativo.

iterações (converge para o valor $0{,}607253$ para $\rho \geq 9$) e, portanto, pode-se evitar seu cálculo ao substituí-lo por esta constante, denominada constante de congregação, representada por Λ.

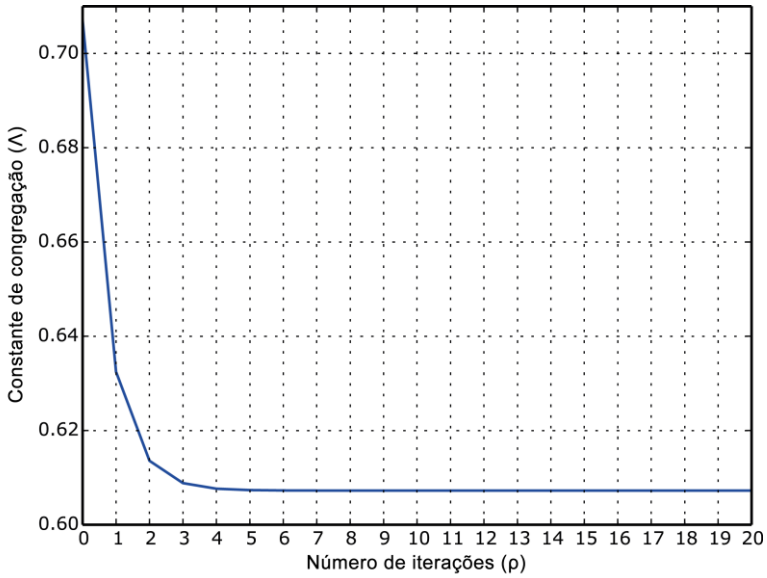


Figura 14 - Valores da constante de congregação pelo número de iterações

Portanto, a rotação completa é expressa por

$$\begin{pmatrix} x_\rho \\ y_\rho \end{pmatrix} = \Lambda \prod_{\lambda=0}^{\rho} \left[\begin{pmatrix} 1 & -sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) \\ sign(\theta_\lambda)\left(\frac{1}{2^\lambda}\right) & 1 \end{pmatrix} \begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix} \right], \tag{26}$$

sendo que em cada iteração, calcula-se o sistema representado na equação (27), lembrando que a divisão por dois é implementada digitalmente por uma rotação de bits, a implementação de cada iteração resume-se a adição, subtração e rotação de bits, operações facilmente implementadas.

$$\begin{cases} x_{\lambda+1} = x_\lambda - sign(\theta_\lambda)\, 2^{-\lambda}\, y_\lambda \\ y_{\lambda+1} = y_\lambda + sign(\theta_\lambda)\, 2^{-\lambda}\, x_\lambda \end{cases}. \tag{27}$$

3 DESENVOLVIMENTO

Nesta seção, detalhar-se-á o processo de implementação de cada componente do sistema, junto com os resultados obtido, ocupação do dispositivo e gráficos, quando necessário.

3.1 *SOFTWARES*

Utilizou-se o programa Quartus II Web Edition, da Altera, para a compilação, análise e síntese, dos códigos desenvolvidos, assim como o programa ModelSim-Altera Starter Edition para a simulação dos mesmos, ambos de licença gratuita, sem restrição de tempo. Para melhor visualização dos resultados, recriou-se as formas de ondas obtidas no ModelSim através da ferramenta WaveDrom (CHAPYNZHEKA, 2011).

Eventualmente utilizou-se *scripts* de autoria própria escritos na linguagem Python para a geração de gráficos de forma a facilitar a visualização dos resultados obtidos.

3.2 REPRESENTAÇÃO DE NÚMEROS COM PONTO FLUTUANTE

Para a representação de números com ponto flutuante utilizou-se uma biblioteca de terceiro, desenvolvida por David Bishop como proposta à IEEE para inclusão no VHDL-2008, denominada FPHDL (BISHOP, 2010). Como grande parte dos *softwares* de síntese ainda não suportam todas as alterações propostas no VHDL-2008, incluindo o Quartus II, Bishop desenvolveu, também, uma biblioteca adaptada para o VHDL-93 completamente sintetizável e esta foi utilizada neste trabalho.

A biblioteca permite configuração completa sobre todos os parâmetros utilizados por ela, incluindo a quantidade de bits utilizada para a representação do expoente e mantissa, o que define a precisão dos números. Com isso, adotou-se para este trabalho a precisão de 32 bits, definida no padrão IEEE 754, em que destina-se oito bits para a representação do expoente e 23 bits para a mantissa, além do bit de sinal.

Criou-se, também, uma extensão da biblioteca FPHDL para a representação de números complexos, com a sobrecarga dos operadores necessários, descrita através do código VHDL a seguir.

Algoritmo 1 - Cabeçalho da biblioteca de representação de números complexos

```
package complex_pkg is
      type complex is record
            real: float;
            imag: float;
      end record;
      type complex_vector
            is array (natural range <>) of complex;
      function "+" (l, r: complex) return complex;
      function "-" (l, r: complex) return complex;
      function "*" (l, r: complex) return complex;
      function conjugate (arg: complex) return complex;
      function "abs" (arg: complex) return float;
end package complex_pkg;
```

3.3 CONVERSORES SERIAL E PARALELO

A principal característica da técnica de transmissão OFDM é a capacidade de transmitir uma grande quantidade de dados distribuídos por múltiplos canais simultaneamente. Visando a implementação de uma comunicação serial entre dois computadores através do sistema desenvolvido neste trabalho, necessitou-se a implementação dos conversores serial/paralelo e paralelo/serial para a interface do sistema com os computadores.

3.3.1 Conversor Serial/Paralelo

O conversor serial/paralelo foi implementado através de uma estrutura de armazenamento de dados, denominada *buffer*, composta por uma lista de bits que permite o acesso simultâneo a vários endereços. Quando o *buffer* é completado com os bits de entrada, seu valor é atribuído à saída. As formas de ondas obtidas para um conversor serial/paralelo de oito bits e paridade par são mostradas na Figura 15 e na

Figura 16, com o bit de paridade correto e incorreto, respectivamente. Observa-se que há três sinais de controle, *busy*, *done* e *alert*, no qual, junto com os outros sinais, são descritos no Quadro 2.

Algoritmo 2 - Implementação das entradas e saídas do conversor serial/paralelo

```
entity SerialToParallel is
      generic (
            constant SIZE: natural := 8;
            constant PARITY: std_logic := '0'
      );

      port (
            signal clock: in std_logic;
            signal reset: in std_logic
            signal serial: in std_logic;
            signal parallel:
                  out std_logic_vector(SIZE-1 downto 0);
            signal busy: out std_logic;
            signal done: buffer std_logic;
            signal alert: buffer std_logic
      );
end entity SerialToParallel;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Serial	Entrada	Std_logic	Dados seriais com *start bit*, *stop bit* e bit de paridade
Parallel	Saída	Std_logic_vector(7 downto 0)	Saída paralela dos dados
Busy	Saída	Std_logic	Quando ‘1’, indica que há conversão em andamento
Done	Saída	Std_logic	Quando ‘1’, indica que a saída pode ser lida com segurança
Alert	Saída	Std_logic	Quando ‘1’, indica que houve erro e os dados precisam ser reenviados

Quadro 2 - Descrição das portas do conversor serial/paralelo

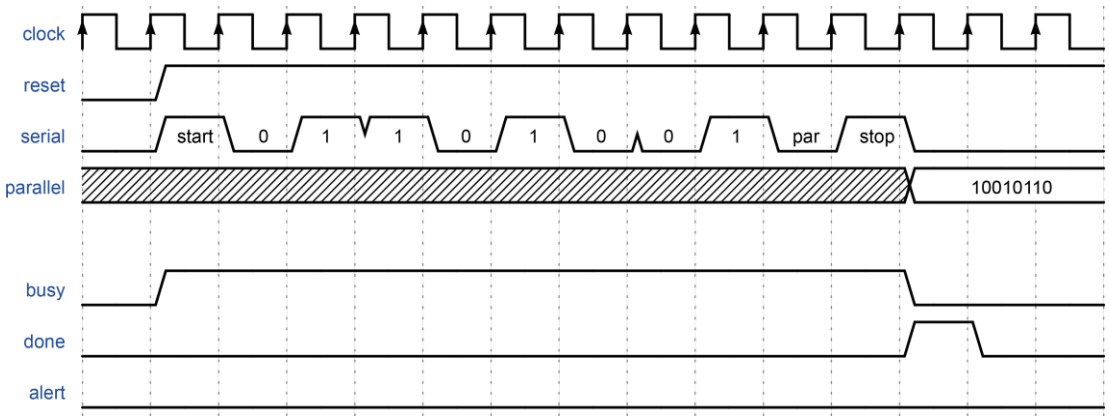


Figura 15 - Simulação do conversor serial/paralelo de 8 bits com paridade correta

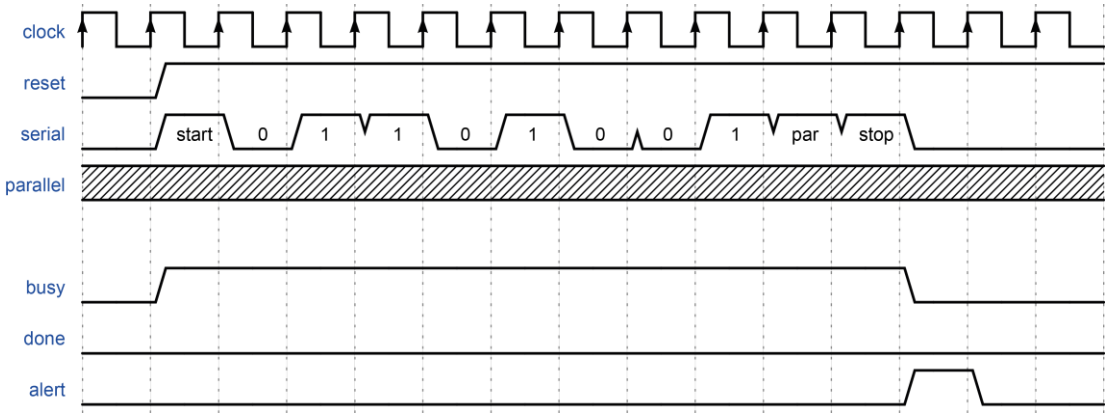


Figura 16 - Simulação do conversor serial/paralelo de 8 bits com paridade incorreta

A saída com os dados de forma paralela pode ser igualmente distribuída entre os canais OFDM para a transmissão, permitindo, assim, a transmissão simultânea da mensagem completa. Observa-se, ainda, que a conversão serial/paralelo insere no sistema um atraso relativo ao tamanho da mensagem que está convertendo, uma vez que a saída é atualizada somente ao final da conversão. Desta forma, a configuração desta conversão deve ser levada em conta quando feito o cálculo da velocidade de transmissão do sistema.

Os resultados da compilação do conversor serial/paralelo são mostrados no Quadro 3, onde observa-se uma utilização de recursos da FPGA inferior a 1%.

Parâmetro	Valor
Entidade	SerialToParallel
Família	Cyclone III
Dispositivo	EP3C16F484C6

Total de elementos lógicos	41 / 15.408 (< 1%)
Total de funções combinacionais	33 / 15.408 (< 1%)
Registradores lógicos dedicados	25 / 15.408 (< 1%)
Total de registradores	25
Total de bits de memória	0

Quadro 3 - Resultados da compilação do conversor serial/paralelo

3.3.2 Conversor Paralelo/Serial

De forma semelhante ao que ocorre no conversor serial/paralelo, os dados de saída do receptor OFDM devem, posterior à demodulação e decodificação, ser serializados, para permitir a comunicação com o computador. Armazena-se os dados de entrada em um *buffer* e despacha-se para a saída um bit por vez, gerando os bits de cabeçalho da mensagem serial, *start bit*, *stop bit* e bit de paridade, dinamicamente. As formas de ondas obtidas de um conversor paralelo/serial de oito bits e paridade par são mostradas na Figura 17. Os sinais amostrados são descritos no Quadro 4.

Algoritmo 3 - Implementação das entradas e saídas do conversor paralelo/serial

```
entity ParallelToSerial is
      generic (
            constant SIZE: natural := 8;
            constant PARITY: std_logic := '0'
      );
      port (
            signal clock: in std_logic;
            signal reset: in std_logic;
            signal start: in std_logic;
            signal parallel:
                  in std_logic_vector(SIZE-1 downto 0);
            signal serial: out std_logic;
            signal busy: out std_logic;
            signal done: buffer std_logic
      );
end entity ParallelToSerial;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando ‘1’, inicia a conversão
Parallel	Entrada	Std_logic_vctor(7 downto 0)	Entrada paralela dos dados
Serial	Saída	Std_logic	Saída serial dos dados, com *start bit*, *stop bit* e bit de paridade
Busy	Saída	Std_logic	Quando ‘1’, indica que há conversão em andamento
Done	Saída	Std_logic	Quando ‘1’, indica que a conversão foi concluída

Quadro 4 - Descrição das portas do conversor paralelo/serial

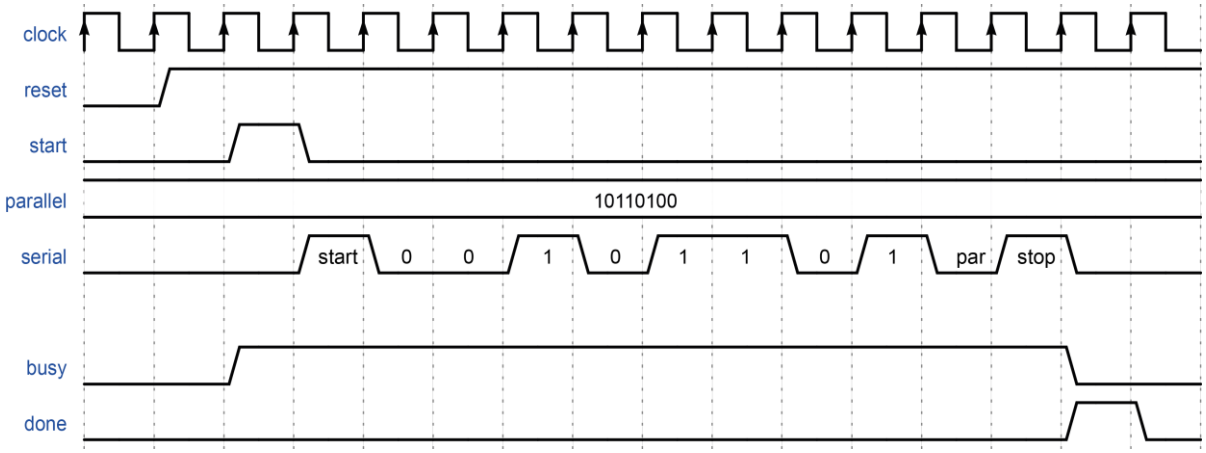


Figura 17 - Simulação do conversor paralelo/serial de 8 bits

Observa-se que, diferente do conversor serial/paralelo, a saída deve ser lida conforme o sinal de controle *busy*, pois o sinal *done* indica apenas o fim da conversão. Porém, o conversor paralelo/serial insere o mesmo atraso relativo que é inserido no conversor serial/paralelo devido ao tempo de conversão.

Os resultados da compilação do conversor paralelo/serial são mostrados no Quadro 5, podendo observar, também, uma ocupação inferior a 1% do dispositivo.

Parâmetro	Valor
Entidade	ParallelToSerial
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	28 / 15.408 (< 1%)
Total de funções combinacionais	28 / 15.408 (< 1%)
Registradores lógicos dedicados	17 / 15.408 (< 1%)

Total de registradores	17
Total de bits de memória	0

Quadro 5 - Resultados da compilação do conversor paralelo/serial

3.4 MODULAÇÃO QAM

Na modulação QAM o sinal transmitido sofre variações tanto em amplitude como em fase e, matematicamente, pode ser representado por um fasor complexo, com parte real e imaginária iguais aos coeficientes de fase e quadratura, respectivamente. Desta forma, é necessário fazer o mapeamento da sequência binária que será transmitida em cada canal OFDM para o seu respectivo fasor complexo e, para isso, armazena-se em memória todas as possíveis sequências de entrada, assim como os seus respectivos fasores complexos, em uma estrutura conhecida como *Look-Up Table* (LUT). O comprimento da sequência binária, assim como o número de fasores complexos a serem armazenados varia conforme a cardinalidade adotada para a modulação QAM.

Optou-se por utilizar a modulação QAM de cardinalidade 16 por ser a mais simples modulação comumente utilizada na prática. O diagrama de constelação que representa a modulação pode ser visto na Figura 18. Cada fasor complexo representa uma sequência binária de quatro bits, na qual diferencia-se em apenas um bit das sequências binárias adjacentes. Essa técnica é conhecida como mapeamento de Gray e reduz a taxa de erros por bit da transmissão.

3.4.1 Codificação QAM

Os sinais implementados no componente em VHDL para a codificação QAM são descritos no Quadro 6 e o diagrama de blocos do componente é mostrado na Figura 19.

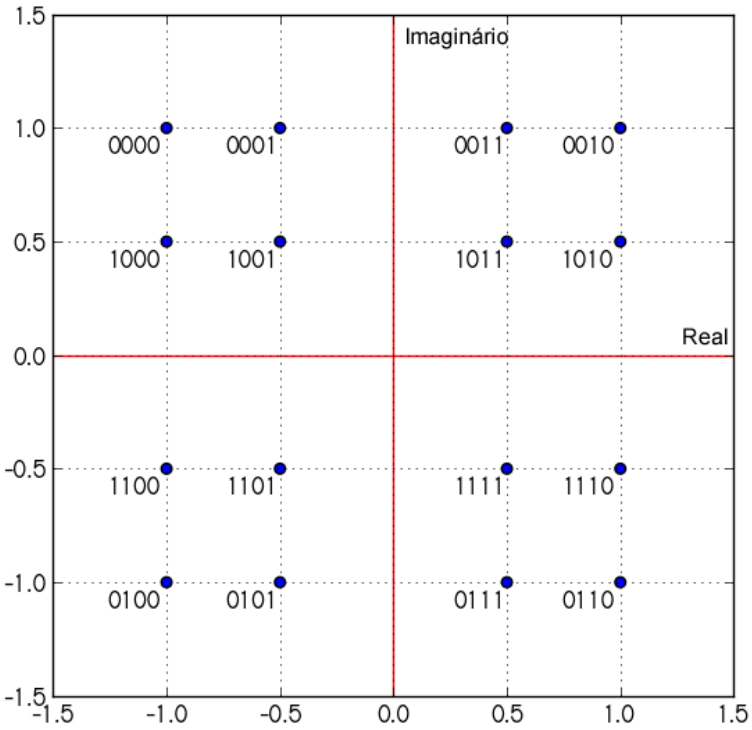


Figura 18 - Diagrama de constelação da modulação 16-QAM

Algoritmo 4 - Implementações das entradas e saídas do codificador QAM

```
entity Encoder is
      port (
            signal clock: in std_logic;
            signal reset: in std_logic;
            signal start: in std_logic;
            signal binary:
                  in std_logic_vector(LENGTH-1 downto 0);
            signal phasor: out complex
      );
end entity Encoder;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando ‘1’, inicia a codificação da sequência de entrada

Binary	Entrada	Std_logic_vector(3 downto 0)	Sequência binária a ser codificada
Phasor	Saída	Complex	Fasor complexo que representa a sequência de entrada

Quadro 6 - Descrição das portas do codificador QAM

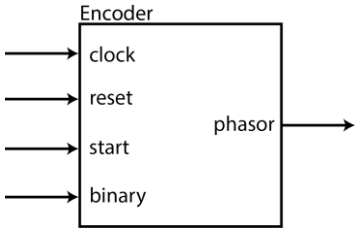


Figura 19 - Diagrama de blocos do codificador QAM

A implementação deste componente resume-se a atualizar a saída *phasor* com o valor armazenado na LUT na posição referente à entrada *binary*. Para uma entrada igual a sequência binária “1101” a saída receberá o valor armazenado na posição 9, o fasor complexo $-0{,}5 - j0{,}5$, conforme Figura 18, como mostrado na Figura 20. A posição pode variar conforme a implementação da LUT.

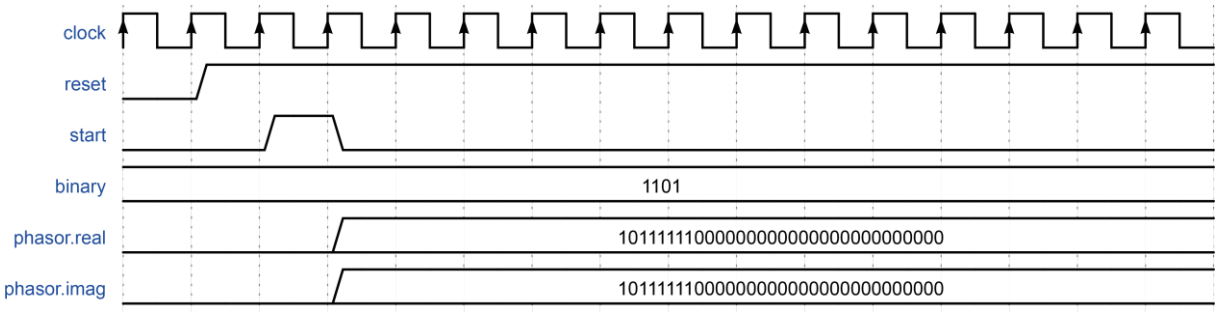


Figura 20 - Simulação do codificador QAM para a sequência "1101"

Os resultados da compilação do codificador QAM são apresentados no Quadro 7. Novamente, por se tratar se uma estrutura bastante simples, sua ocupação no dispositivo é inferior à 1%.

Parâmetro	Valor
Entidade	Encoder
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	5 / 15.408 (< 1%)

Total de funções combinacionais	3 / 15.408 (< 1%)
Registradores lógicos dedicados	5 / 15.408 (< 1%)
Total de registradores	5
Total de bits de memória	0

Quadro 7 - Resultados da compilação do codificador QAM

3.4.2 Decodificação QAM

Os sinais implementados no componente em VHDL para a decodificação QAM são descritos no Quadro 8 e o diagrama de blocos do componente é mostrado na Figura 21.

Algoritmo 5 - Implementações das entradas e saídas do decodificador QAM

```
entity Decoder is
      port (
            signal clock: in std_logic;
            signal reset: in std_logic;
            signal start: in std_logic;
            signal phasor: in complex;
            signal binary:
                  out std_logic_vector(LENGTH-1 downto 0)
      );
end entity Decoder;
```

A implementação do decodificador é exatamente o processo inverso ao feito na implementação do codificador. Isto é, atualiza-se a saída *binary* com o valor armazenado na LUT referente ao fasor complexo de entrada *phasor*. Para uma entrada de valor $1 + j$, a saída receberá a sequência binária “0010”, como mostrado na Figura 22, lembrando que a entrada *fasor* possui representação de 32 bits.

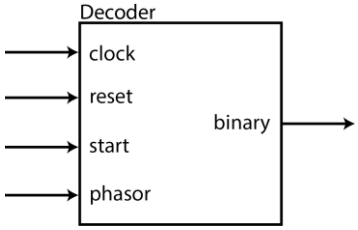


Figura 21 - Diagrama de blocos do decodificador QAM

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando ‘1’, inicia a codificação da sequência de entrada
Phasor	Entrada	Complex	Fasor complexo a ser decodificado
Binary	Saída	Std_logic_vector(3 downto 0)	Sequência binária respectiva ao fasor de entrada

Quadro 8 - Descrição das portas do decodificador QAM

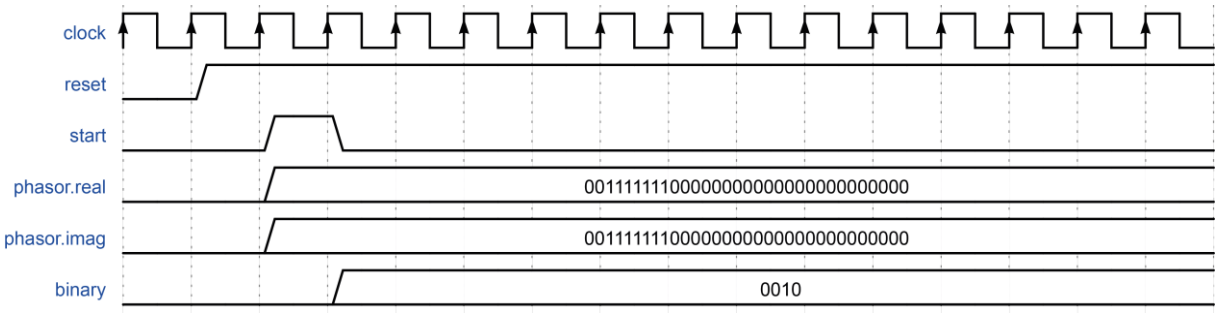


Figura 22 - Simulação do decodificador QAM para o fasor complexo (1,0; 1,0)

Os resultados da compilação do decodificador QAM são apresentados no Quadro 9. Embora consuma pouco mais recurso que o codificador, o total ainda se mantém inferior a 1%.

Parâmetro	Valor
Entidade	Decoder
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	29 / 15.408 (< 1%)
Total de funções combinacionais	29 / 15.408 (< 1%)
Registradores lógicos dedicados	4 / 15.408 (< 1%)
Total de registradores	4
Total de bits de memória	0

Quadro 9 - Resultados da compilação do decodificador QAM

3.5 SIMETRIA HERMITIANA

A simetria Hermitiana é necessária, como visto no item 2.5, para que a modulação dos dados possa ser feita através da transformada de Fourier e resume-

se a gerar, a partir de N valores complexos, um total de $2N + 2$ valores através de seus valores conjugados.

A descrição dos sinais implementados em VHDL para o componente encontra-se no Quadro 10 e o diagrama de blocos do componente é mostrado na Figura 23.

Algoritmo 6 - Implementações das entradas e saídas da simetria Hermitiana

```
entity HermitianSymmetry is
    generic (
        constant POINTS: natural := 7
    );

    port (
        signal Z_in: in complex_vector(0 to POINTS-1);
        signal Z_out: out complex_vector(0 to 2*POINTS+1)
    );
end entity HermitianSymmetry;
```

Porta	Direção	Tipo	Descrição
Z_in	Entrada	Complex_vector(0 to 6)	Lista de N valores complexos de entrada
Z_out	Saída	Complex_vector(0 to 15)	Lista de $2N + 2$ valores complexos de saída

Quadro 10 - Descrição das portas da simetria Hermitiana

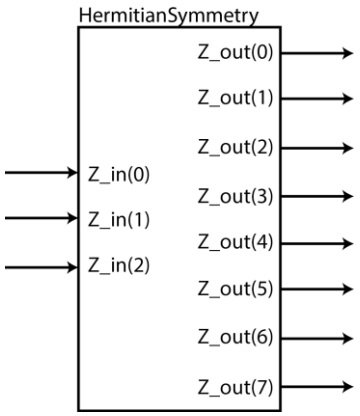


Figura 23 - Diagrama de blocos da simetria Hermitiana para N=3

O resultado da simulação do componente da simetria Hermitiana pode ser visto na Figura 24. Para facilitar a visualização dos gráficos, limitou-se a três valores

de entrada, nos quais são apresentados em sua forma real, sendo que em VHDL os mesmos possuem representação conforme a IEEE 754.

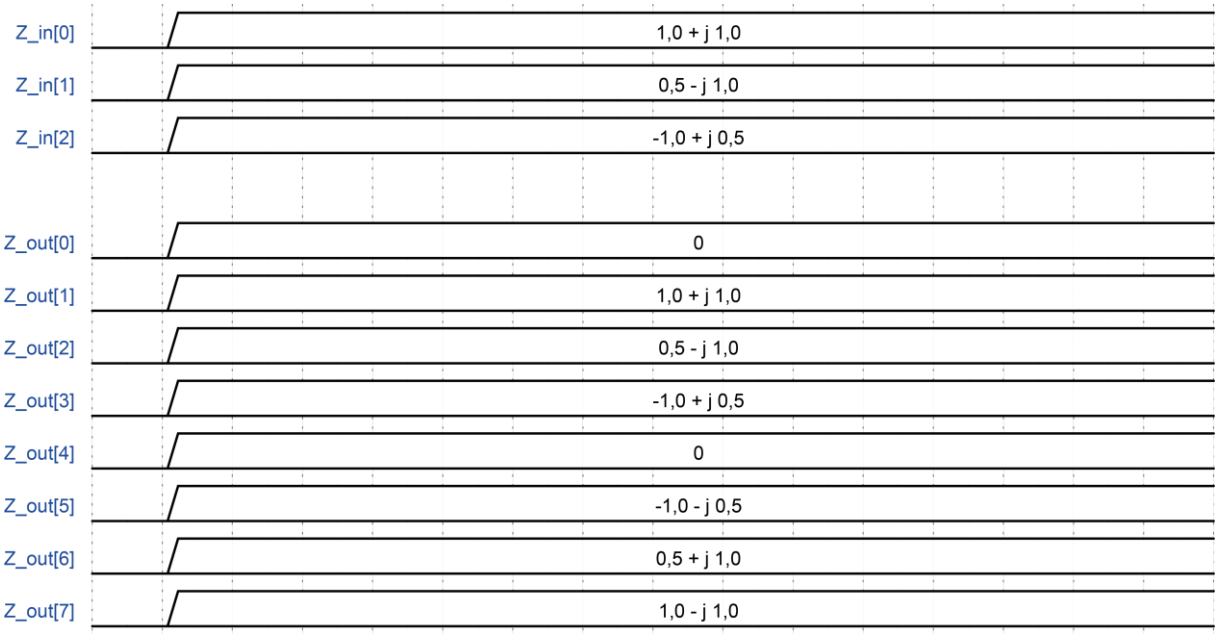


Figura 24 - Simulação da simetria Hermitiana para N=3

O código VHDL que efetua o cálculo da simetria Hermitiana é apresentado a seguir.

Algoritmo 7 - Geração da simetria Hermitiana em VHDL

```
architecture Behavior of HermitianSymmetry is
    alias N: natural is POINTS;
begin
     Z_out(0) <= (
            real => (others => '0'),
            imag => (others => '0')
      );

     Z_out(N+1) <= (
            real => (others => '0'),
            imag => (others => '0')
       );

       Z_out(1 to N) <= Z_in(0 to N-1);
```

```
       Symmetry: for i in N+2 to 2*N+1 generate begin
          Z_out(i) <= conjugate(Z_in(2*N + 1 - i));
      end generate Symmetry;
end architecture Behavior;
```

Os resultados da compilação do componente são apresentados no Quadro 11. É visto que, como o cálculo do conjugado de um valor complexo resume-se a apenas inverter o sinal da parte imaginária do mesmo e na representação IEEE 754 esta inversão é feita invertendo apenas o bit de sinal, o componente resume-se a poucas inversões de bits, o que explica o 0% obtido como ocupação do dispositivo, pois o processo é efetuado em elementos lógicos alocados anteriormente.

Parâmetro	Valor
Entidade	HermitianSymmetry
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	0 / 15.408 (0%)
Total de funções combinacionais	0 / 15.408 (0%)
Registradores lógicos dedicados	0 / 15.408 (0%)
Total de registradores	0
Total de bits de memória	0

Quadro 11 - Resultados da compilação da simetria Hermitiana

3.6 TRANSFORMADA DE FOURIER

Como demonstrado no item 2.6, a implementação da transformada de Fourier apresenta dois componentes principais: o de cálculo das funções trigonométricas, através do algoritmo CORDIC, e o de efetuar o cálculo da *Butterfly*. Ambas implementações são detalhadas abaixo.

3.6.1 Cálculo das Funções Trigonométricas por CORDIC

Uma das principais vantagens do algoritmo CORDIC é permitir que o cálculo das funções desejadas possa ser feito de forma iterativa. Porém, sendo VHDL uma linguagem paralela e concorrente, a implementação deste cálculo não é algo trivial.

Desta forma, a melhor maneira é utilizar um conceito herdado da teoria de sistemas operacionais, criado por Douglas Mcllroy (WIKIPEDIA, 2014), denominado *pipeline* (encadeamento), que consiste em encadear múltiplos processos através de seus fluxos padrão, de forma que a saída de um processo é utilizada como entrada do processo seguinte. Para um processo de três iterações, o componente implementado teria a forma apresentada na Figura 25.

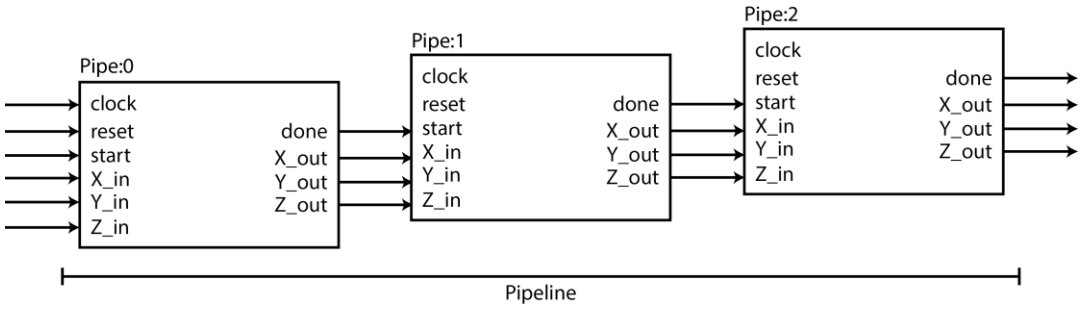


Figura 25 - Representação em blocos da pipeline do CORDIC

A descrição dos sinais implementados nos componentes que executam o cálculo do CORDIC é apresentada no Quadro 12.

Algoritmo 8 - Implementações das entradas e saídas do CORDIC

```
entity Cordic_Circular_Rotation is
    generic (
        constant ITERATIONS: natural := 12
    );

    port (
        signal clock: in std_logic;
        signal reset: in std_logic;
        signal start: in std_logic;
        signal X_in: in signed(14 downto 0);
        signal Y_in: in signed(14 downto 0);
        signal Z_in: in signed(19 downto 0);
```

```
        signal X_out: out signed(14 downto 0);
        signal Y_out: out signed(14 downto 0);
        signal Z_out: out signed(19 downto 0);
        signal done: out std_logic
    );
end entity Cordic_Circular_Rotation;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando “1”, inicia o cálculo da iteração
X_in	Entrada	Signed(14 downto 0)	Componente sobre as abcissas do vetor a ser rotacionado
Y_in	Entrada	Signed(14 downto 0)	Componente sobre as ordenadas do vetor a ser rotacionado
Z_in	Entrada	Signed(19 downto 0)	Ângulo de rotação
X_out	Saída	Signed(14 downto 0)	Componente sobre as abcissas do vetor rotacionado
Y_out	Saída	Signed(14 downto 0)	Componente sobre as ordenadas do vetor rotacionado
Z_out	Saída	Signed(19 downto 0)	Ângulo restante para completar a rotação

Quadro 12 - Descrição das portas de uma iteração do CORDIC

Como visto a partir da equação (24), consegue-se otimizar o cálculo do algoritmo CORDIC através da constante de congregação e, para isso, de forma a melhorar os resultados do componente VHDL, armazenou-se os valores apresentados no Quadro 13 em memória, conforme o número de iterações utilizado.

É natural que quanto mais iterações for utilizada, melhores serão os resultados, porém, ao mesmo tempo, consumirá mais recursos do dispositivo. Experimentalmente, observou-se que o algoritmo implementado apresentou exatidão de até quatro casas decimais nos valores das funções seno e cosseno a partir de doze iterações, então adotou-se esta quantia para o trabalho.

Número de iterações	Constante de Congregação (K)
0	0.707107
1	0.632456
2	0.613572
3	0.608834

4	0.607648
5	0.607352
6	0.607278
7	0.607259
8	0.607254
9 ou mais	0.607253

Quadro 13 - Valores da constante de congregação do algoritmo CORDIC

Devido a restrições na implementação do algoritmo, a faixa de convergência do mesmo é definida entre os ângulos 0 e $\pi/2$, apresentando, nesta faixa, os valores esboçados no gráfico da Figura 26. Pode-se observar que perto dos extremos de convergência do algoritmo, a função trigonométrica que teria seu valor próximo a um apresenta um comportamento inesperado, necessitando, assim, uma compensação manual no código da implementação, caso seja desejável aproximar mais o valor do real.

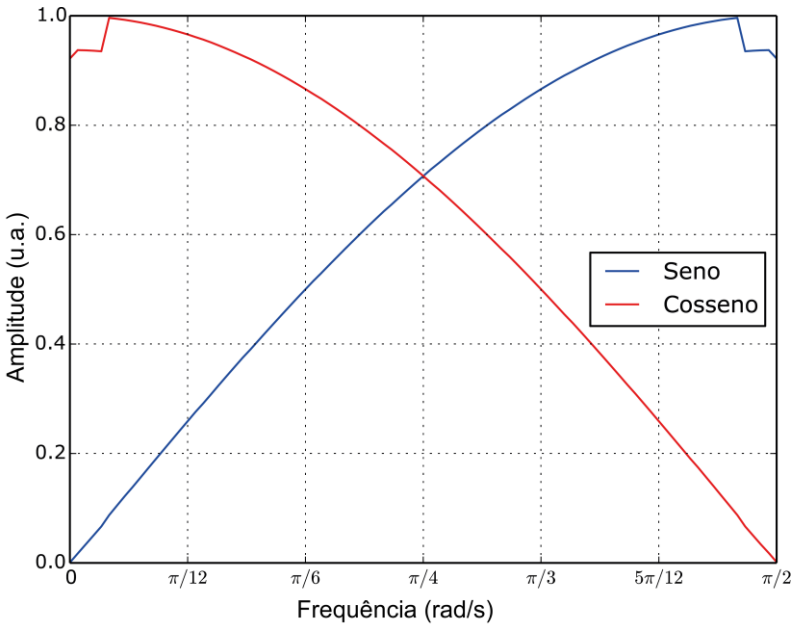


Figura 26 - Valores de seno e cosseno calculados por CORDIC

Para a utilização do algoritmo no sistema, é necessário, ainda, que este seja capaz de calcular os valores das funções trigonométricas fora de seu intervalo de convergência e, para contornar este problema de maneira simples, basta utilizar-se das relações de redução de ângulo ao primeiro quadrante. Ou seja, quando é necessário o cálculo dos valores de seno e cosseno de um ângulo superior a $\pi/2$,

reduz-se este ao primeiro quadrante, calcula-se o valor de seno e cosseno com CORDIC do ângulo reduzido e, posteriormente, corrige-se os sinais destes valores conforme o quadrante original do ângulo.

Adicionando no componente estas duas alterações: a compensação nos ângulos próximos aos limites de convergência e redução ao primeiro quadrante, obtém-se como resultado os valores esboçados no gráfico da Figura 27, onde as curvas em azul representam os valores do seno e as curvas em vermelho os valores do cosseno.

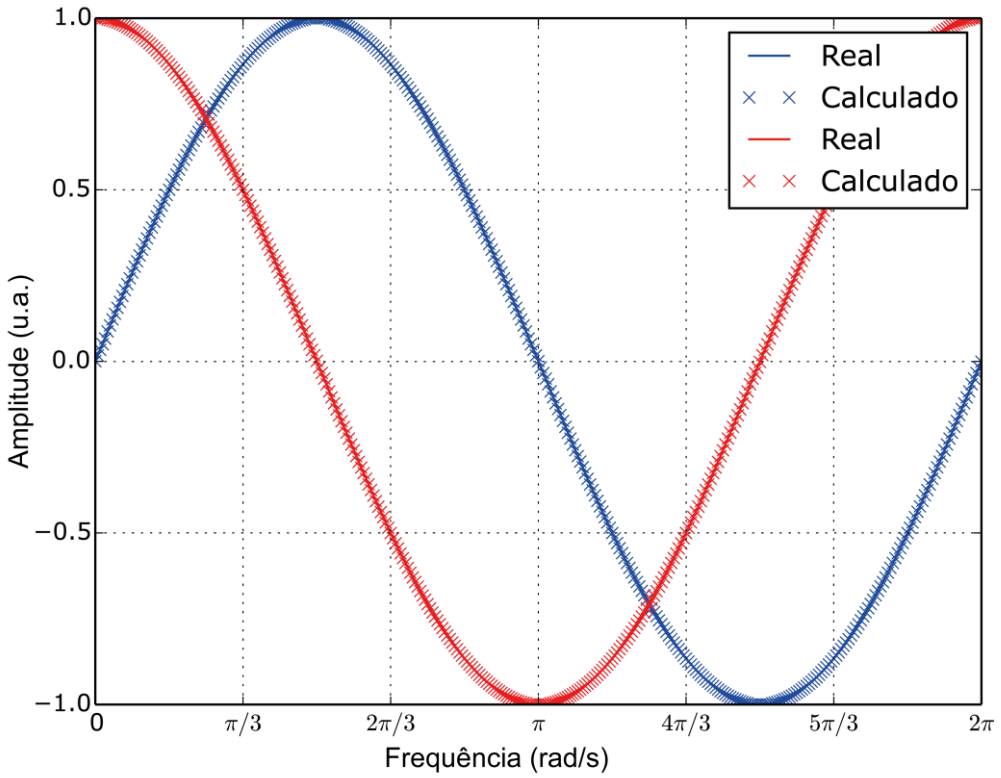


Figura 27 - Valores de seno e cosseno com CORDIC compensado

É notável a precisão que é obtida através deste algoritmo, utilizando-se apenas doze iterações.

Os resultados da compilação são mostrados no Quadro 14.

Parâmetro	Valor
Entidade	Cordic_Circular_Rotation
Família	Cyclone III
Dispositivo	EP3C16F484C6

Total de elementos lógicos	1.365 / 15.408 (9%)
Total de funções combinacionais	1.286 / 15.408 (8%)
Registradores lógicos dedicados	588 / 15.408 (4%)
Total de registradores	588
Total de bits de memória	0

Quadro 14 - Resultados da compilação do algoritmo CORDIC

3.6.2 Cálculo da *Butterfly*

Para a implementação da *Butterfly*, utilizou-se como base o diagrama apresentado na Figura 28, com duas entradas e duas saídas, configurando, assim a utilização do radix-2.

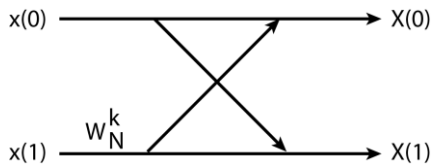


Figura 28 - Diagrama da *Butterfly* radix-2

As descrições dos sinais implementados do componente *Butterfly* são apresentadas no Quadro 15.

Algoritmo 9 - Implementações das entradas e saídas da Butterfly

```
entity Butterfly is
    generic (
        constant INVERSE: boolean := false
    );

    port (
        signal clock: in std_logic;
        signal reset: in std_logic;
        signal start: in std_logic;
        signal Z_a: in complex;
        signal Z_b: in complex;
        signal Wk: in complex;
        signal Z_c: out complex;
        signal Z_d: out complex;
        signal done: buffer std_logic
```

```
    );
end entity Butterfly;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando “1”, inicia o cálculo da *Butterfly*
Z_a	Entrada	Complex	Primeiro valor complexo de entrada (x(0))
Z_b	Entrada	Complex	Segundo valor complexo de entrada (x(1))
Wk	Entrada	Complex	Fator de rotação (*twiddle factor*)
Z_c	Saída	Complex	Primeiro valor complexo de saída (X(0))
Z_d	Saída	Complex	Segundo valor complexo de saída (X(1))
Done	Saída	Std_logic	Quando ‘1’, indica que o cálculo foi finalizado

Quadro 15 - Descrição das portas da Butterfly

O resultado da compilação do componente *Butterfly* é apresentado no Quadro 16 e, como pode ser visto, devido a existência de multiplicações entre valores com ponto flutuante, a ocupação do dispositivo sobe drasticamente. Como a transformada de Fourier de oito pontos exige a computação de doze *Butterfly* em seu cálculo, torna-se impossível sua execução embarcada na FPGA em questão, pois o sistema consumiria mais de 100% do dispositivo. Desta forma, os resultados obtidos a partir da *Butterfly* foram através de simulações.

Parâmetro	Valor
Entidade	Butterfly
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	9.334 / 15.408 (61%)
Total de funções combinacionais	9.334 / 15.408 (61%)
Registradores lógicos dedicados	129 / 15.408 (< 1%)
Total de registradores	129
Total de bits de memória	0
Elementos multiplicadores embarcados 9-bit	28 / 112 (25%)

Quadro 16 - Resultados da compilação da *Butterfly*

Sobre o componente *Butterfly*, desenvolveu-se o cálculo da IFFT de 8 pontos, encadeando, assim, um total de 12 *Butterfly*, divididas em três estágios, como mostra

a Figura 12. As descrições dos sinais deste componente são apresentadas no Quadro 17, o diagrama de blocos que o representa é mostrado na Figura 29, e o resultado da compilação do mesmo é apresentado no Quadro 18.

Algoritmo 10 - Implementações das entradas e saídas da FFT

```
entity FFT is
    generic (
        constant INVERSE: boolean := false;
        constant POINTS: natural := 8;
        constant STAGES: natural := 3
    );

    port (
        signal clock: in std_logic;
        signal reset: in std_logic;
        signal start: in std_logic;
        signal Z_in: in complex_vector(0 to POINTS-1);
        signal Z_out: out complex_vector(0 to POINTS-1);
        signal ready: out std_logic;
        signal done: buffer std_logic
    );
end entity FFT;
```

Porta	Direção	Tipo	Descrição
Clock	Entrada	Std_logic	Sinal de sincronização
Reset	Entrada	Std_logic	Sinal de reinicialização
Start	Entrada	Std_logic	Quando “1”, inicia o cálculo da *Butterfly*
Z_in	Entrada	Complex_vector(0 to 7)	Lista de valores complexos de entrada
Z_out	Saída	Complex_vector(0 to 7)	Lista de valores complexos de saída
Ready	Saída	Std_logic	Quando ‘1’, indica que o componente está inicializado
Done	Saída	Std_logic	Quando ‘1’, indica que finalizou o cálculo

Quadro 17 - Descrição das portas da FFT

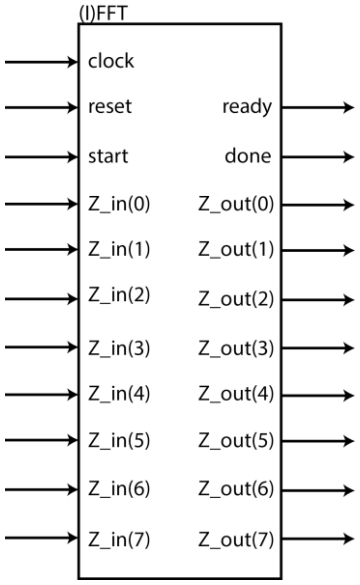


Figura 29 - Diagrama de blocos da transformada de Fourier

Para o teste do componente, foi inserido uma sequência de números complexos no qual sua transformada fosse previamente conhecida e esta foi comparada com os valores obtidos. Tal comparação é apresentada no Quadro 19. Vale ressaltar que a sequência de entrada foi escolhida de forma a satisfazer a simetria Hermitiana e, desta forma, a sequência de saída possui apenas valores reais. Mesmo que a saída obtida apresente componentes imaginárias, estas são muito menores que os componentes reais e podem ser desconsideradas.

Parâmetro	Valor
Entidade	FFT
Família	Cyclone III
Dispositivo	EP3C16F484C6
Total de elementos lógicos	130.384 / 15.408 (> 100%)
Total de funções combinacionais	130.164 / 15.408 (> 100%)
Registradores lógicos dedicados	2.347 / 15.408 (15%)
Total de registradores	2.347
Total de bits de memória	100
Elementos multiplicadores embarcados 9-bit	112 / 112 (100%)

Quadro 18 - Resultados da compilação da FFT

Índices	Sequências		
	Entrada	**Saída Esperada**	**Saída Obtida**
0	0	-1	-0,9981 – 7,3e-4 j
1	1 + j	3,1213	3,1245 – 1,8e-3 j
2	-1 + 0,5 j	6	5,9981 – 1,2e-3 j
3	-0,5 - j	-3,1213	-3,1216 – 2,4e-3 j
4	0	-3	-3,0010 – 3,6e-3 j
5	-0,5 + j	-1,1213	-1,1230 – 6,3e-4 j
6	-1 - 0,5 j	-2	-1,9989 + 5,6e-3 j
7	1 - j	1,1213	1,1202 – 6,5e-6 j

Quadro 19 - Comparação dos valores obtidos no cálculo da FFT

Inseriu-se, também, os valores da saída esperada do Quadro 19 como entrada do cálculo da transformada inversa, esperando obter como saída os valores de entrada do mesmo quadro. O resultado obtido é apresentado no Quadro 20.

Índices	Sequências		
	Entrada	**Saída Esperada**	**Saída Obtida**
0	-1	0	0
1	3,1213	1 + j	1,0017 + 1,0002 j
2	6	-1 + 0,5 j	-0,9992 + 0,4993 j
3	-3,1213	-0,5 - j	-0,5007 – 0,9984 j
4	-3	0	0
5	-1,1213	-0,5 + j	-0,4992 + 0,9998 j
6	-2	-1 - 0,5 j	-1,0004 – 0,5015 j
7	1,1213	1 - j	0,9997 – 1,0000 j

Quadro 20 - Comparação dos valores obtidos no cálculo da IFFT

3.7 INTERVALO DE GUARDA

Como já comentado, o fato de se trabalhar com um sistema *back-to-back* faz desnecessário a implementação do intervalo de guarda, porém, como este resume-se a transmissão da cópia de parte do símbolo, antes ou depois do mesmo, a implementação deste é facilmente feita retransmitindo certa quantidade dos valores de saída da IFFT, conforme o comprimento do intervalo de guarda desejado e as características do canal utilizado.

4 RESULTADOS

Nesta seção, detalhar-se-á os resultados obtidos após a implementação do sistema completo, composto do transmissor e receptor.

4.1 TRANSMISSOR

O transmissor foi implementado interconectando os componentes de codificação QAM, simetria Hermitiana e transformada inversa de Fourier, descritas na seção anterior, como mostrado na Figura 30. Definiu-se uma entrada de 12 bits, no qual é dividida em três sequências de quatro bits cada para a codificação QAM, de onde saem três fasores complexos. Gera-se, através da simetria Hermitiana, oito valores complexos a partir destes três valores e, sobre estes, efetua-se a transformada inversa de Fourier. Como resultado da transformada, obtém-se oito valores reais, no qual seriam os valores a serem transmitidos pelo canal. Sendo um sistema na configuração *back-to-back*, tais valores são enviados diretamente ao receptor.

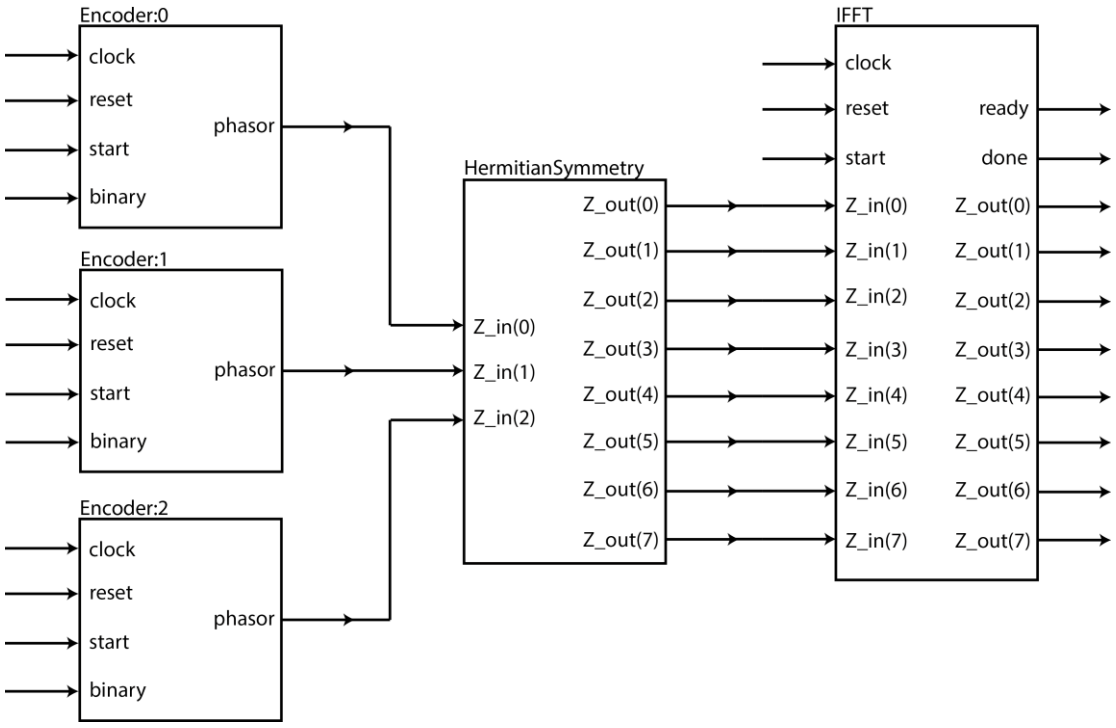


Figura 30 - Diagrama de blocos do transmissor

Como teste, inseriu-se a sequência binária “100111111010” como entrada do transmissor. Esta deverá ser dividida nas sequências “1001”, “1111” e “1010”, no qual, conforme o diagrama de constelação QAM, Figura 18, gerará os fasores $(-0{,}5 + 0{,}5j)$, $(0{,}5 - 0{,}5j)$ e $(1{,}0 + 0{,}5j)$. O resultado da transmissão, que é a saída da transformada de Fourier, é apresentado no Quadro 21, junto com os valores obtidos na simulação. As formas de ondas deste processo podem ser visualizadas no Anexo B.

Índices	**Sequências**		
	Entrada da IFFT	**Saída Esperada**	**Saída Obtida**
0	0	0,2500	0,2500
1	1 + 0,5 j	0,2134	0,2134 – 1,9e-4 j
2	0,5 – 0,5 j	-0,1250	-0,1250 – 9,1e-5 j
3	-0,5 + 0,5 j	-0,5669	-0,5669 + 8,0e-4 j
4	0	0	0
5	-0,5 – 0,5 j	0,0366	0,0365 – 1,7e-4 j
6	0,5 + 0,5 j	-0,1250	-0,1249 – 1,8e-4 j
7	1 – 0,5 j	0,3169	0,3169 – 3,6e-4 j

Quadro 21 - Resultado do teste do transmissor para a sequência "100111111010"

Nota-se, então, mesmo havendo resíduos das operações matemáticas nas componentes imaginárias, estas podem ser desprezadas devido pois aproximam-se bastante do zero.

O código VHDL implementado para o teste do transmissor é apresentado no Anexo D.

4.2 RECEPTOR

O receptor foi implementado interconectando os componentes da transformada de Fourier com os decodificadores da modulação QAM, como mostra a Figura 31. Vale ressaltar que os canais extras, gerados pela simetria Hermitiana no transmissor, são removidos no receptor, pois estes são desconsiderados e não levados em conta na hora da reconstrução da mensagem.

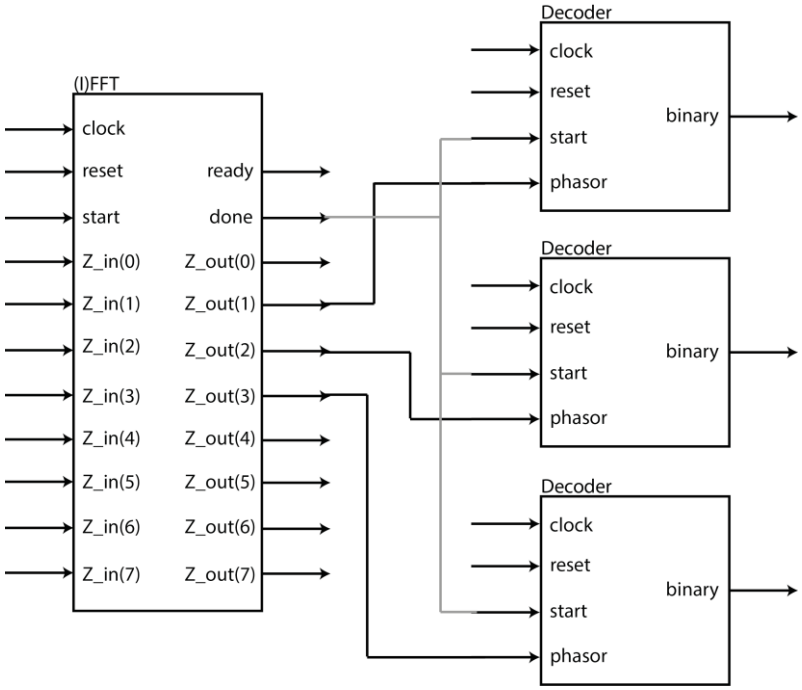


Figura 31 - Diagrama de blocos do receptor

Como teste, inseriu-se como entrada os valores obtidos no teste do transmissor, ignorando a parte imaginária, com o intuito de recuperar a mesma sequência binária de entrada deste. Ou seja, a saída do componente da FFT deve ser igual à segunda coluna do Quadro 21, já que esta foi definida como entrada da IFFT no transmissor. Os resultados obtidos são mostrados no Quadro 22. As formas de ondas deste processo podem ser visualizadas no Anexo C.

Índices	**Sequências**		
	Entrada da FFT	**Saída Esperada**	**Saída Obtida**
0	0,2500	0	-1,5e-7 – 1,9e-4 j
1	0,2134	1 + 0,5 j	1,0008 + 0,4992 j
2	-0,1250	0,4999 – 0,4999 j	0,4990 – 0,4996 j
3	-0,5669	-0,5 + 0,4998 j	-0,4992 + 0,5007 j
4	0	0,0002	0,0002 – 3,5e-4 j
5	0,0365	-0,5 – 0,4998 j	-0,5006 - 0,4990 j
6	-0,1249	0,4999 + 0,4999 j	0,5007 + 0,5001 j
7	0,3169	1 – 0,5 j	0,9990 – 0,5009 j

Quadro 22 - Resultado do teste do receptor

Percebe-se, então, que mesmo não havendo os efeitos do canal sobre o sinal, há a variação dos valores em torno do valor esperado devido às próprias operações

matemáticas realizadas e, desta forma, a sequência binária não pode ser recuperada de forma direta. Assim, necessitou-se alterar o código do decodificador QAM para que, ao invés de associar um ponto no plano complexo à uma sequência binária, associasse uma região circular centrado neste ponto, de raio definido conforme a máxima variação permitida. Desta forma, analisando as variações obtidas no Quadro 22, percebeu-se que um raio de 0,01 já seria suficiente para recuperar a mensagem, obtendo, assim, a sequência “100111111010” como saída – exatamente a mesma sequência definida como entrada do transmissor.

O código VHDL implementado para o teste do receptor é apresentado no Anexo E.

4.3 TESTE DE TRANSMISSÃO

Finalmente, de forma a testar o sistema completo, conectando o transmissor ao receptor, gerou-se, a partir do texto de Olavo Bilac, “Via Láctea” XIII, a partir da tabela ASCII, um arquivo binário de 4488 bits, no qual foi dividido em 374 sequências de 12 bits. Cada sequência foi transmitida pelo sistema, gravando em arquivo a saída obtida no receptor. Posteriormente, converteu-se o arquivo resultante para texto novamente e constatou-se que a mensagem foi transmitida com sucesso, sem nenhum caractere errado. A transmissão completa durou 15.510 ns, sendo 400 ns utilizados para o cálculo das funções trigonométricas e 15.110 ns para a transmissão em si, resultado em uma velocidade aproximada de 34,5 MB/s, para um *clock* de 50 MHz. Foi feita a simulação funcional do sistema e, assim, atrasos internos de propagação não foram considerados nos tempos supracitados.

O arquivo binário com o texto original foi gerado manualmente, sendo lido através de uma instância *file* no código VHDL, como pode ser visto no código apresentado no Anexo F. Foi analisado, também, os valores de saída da FFT no receptor, afim de verificar o espalhamento destes em relação aos valores definidos no diagrama de constelação QAM e este espalhamento, para o ponto $1 + j$, é mostrado na Figura 32. A partir destes valores, fez-se o cálculo do desvio padrão para cada ponto, obtendo os valores apresentados no Quadro X, a fim de se conhecer o raio da área circular que envolvesse todos os pontos, de forma a recuperar com sucesso a mensagem no receptor.

Ponto	Desvio Padrão
$-0{,}5 + j\ 1{,}0$	0,0009219
$0{,}5\ +\ j\ 1{,}0$	0,0007908
$-1{,}0\ +\ j\ 1{,}0$	0,0006976
$-0{,}5\ +\ j\ 0{,}5$	0,0005864
$-1{,}0\ +\ j\ 0{,}5$	0,0009060
$-0{,}5 - j\ 1{,}0$	0,0008978
$-0{,}5 - j\ 0{,}5$	0,0003988
$0{,}5\ +\ j\ 0{,}5$	0,0009061
$-1{,}0 - j\ 1{,}0$	0,0008459
$1{,}0 - j\ 0{,}5$	0,0003082
$0{,}5 - j\ 0{,}5$	0,0006364
$1{,}0\ +\ j\ 0{,}5$	0,0004177
$0{,}5 - j\ 1{,}0$	0,0007530
$1{,}0 - j\ 1{,}0$	0,0009287
$1{,}0\ +\ j\ 1{,}0$	0,0010493
$-1{,}0 - j\ 0{,}5$	0,0006866

Quadro 23 – Valores dos desvios padrão no receptor

Pela distribuição normal, sabe-se que um range equivalente a três desvios padrão é possível abranger 99,7% das amostras e adotando o maior desvio padrão obtido, 0,0010493, de forma que, com um mesmo valor, possa abranger as amostras em todos os pontos, percebe-se que uma região circular de raio 0,0031480 é suficiente para que a mensagem seja recuperada com sucesso no receptor. Considerou-se, então, de forma a facilitar a implementação, uma região circular de raio 0,0032, representada por um círculo vermelho.

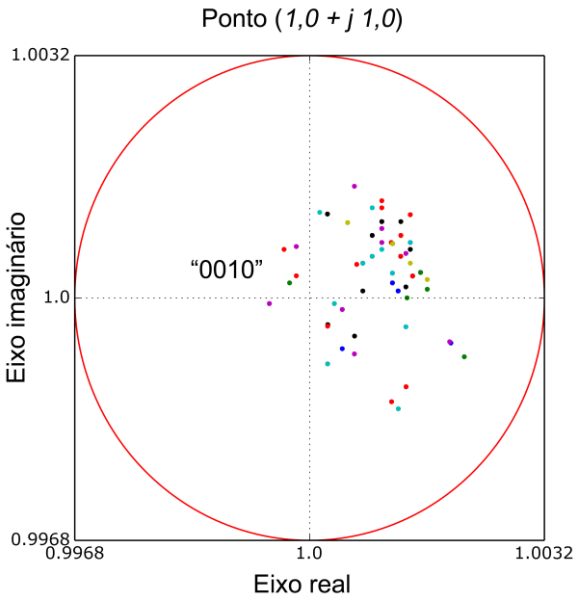


Figura 32 - Espalhamento dos resultados da FFT em relação ao ponto $1 + j$

5 CONCLUSÃO

O desenvolvimento das técnicas de transmissão digital, assim como dos dispositivos lógicos programáveis tem tornado realidade a possibilidade de se fazer sistemas de transmissão digital integrados em um único chip. Para fazer uso destas tecnologias modernas, foi desenvolvido uma estrutura completa para o transmissor e para o receptor OFDM, que utilizam a (I)FFT para fazer a modulação.

Como contribuição, este trabalho fornece um código completo e sintetizável, escrito em VHDL, para um modem (modulador/demodulador) OFDM com números com ponto flutuante, utilizando a (I)FFT de oito pontos, Radix-2 com CORDIC, que permite uma taxa de transmissão de aproximadamente 34,5 MB/s, para um *clock* de 50 MHz.

Para isso foram desenvolvidos e analisados os vários blocos funcionais desse sistema. Mostrou-se e comparou-se a ocupação em *hardware* das várias partes que o compõem, chegando-se a conclusão de que as partes críticas são o cálculo das funções trigonométricas através do CORDIC e o cálculo da FFT, sendo este último o de menor desempenho devido à sua complexidade. Porém, a própria empresa Altera tem, recentemente, anunciado projetos visando a fabricação de FPGAs com suporte nativo em *hardware* para números com ponto flutuante e, desta forma, aplicações como a desenvolvida neste trabalho podem se tornar viáveis em um futuro breve.

Mas ainda pode-se melhorar o desempenho do sistema revendo a implementação do cálculo da *Butterfly* na FFT e analisando a possibilidade do uso de memórias externas para armazenamento dos valores durante os cálculos buscando reduzir a quantidade de recurso demandado da FPGA, porém isto fica como sugestão para trabalhos futuros, assim como a inserção dos efeitos de um canal real durante a transmissão.

REFERÊNCIAS

BHARDWAJ, M.; GANGWAR, A.; SONI, D. **A Review on OFDM: Concept, Scope & its Applications**. 1. Ed. [S.l.]: IOSR Journal of Mechanical and Civil Engineering (IOSRJMCE), v. 1, 2012. 7-11 p.

BINGHAM, J. A. C. **ADSL, VDSL and Multicarrier Modulation**. 1ª edição. John Wiley & Sons, Inc. 2000.

BISHOP, D. W. **VHDL-2008 Support library**, 2010. Disponivel em: <http://www.vhdl.org/fphdl/>. Acesso em: Março 2014.

BROWN, S. ROSE, J. **Architecture of FPGAs and CPLDs: A Tutorial**. Toronto: Department of Electrical and Computer Engineering – University of Toronto, 2002.

BURRUS, C. **Index Mappings for Multidimesional Formulation of the DFT and Convolution**. [S.l.]: IEEE Transactions on Acoustics, Speech and Signal Processing, v. 25, 1977. 239-242 p.

CHAPYZHENKA, A. WaveDrom - Digital timing diagram in your browser. **WaveDrom**, 2011. Disponivel em: <http://wavedrom.github.io/>. Acesso em: 2014.

COHEN, B. **VHDL: Coding-Styles and Methodologies**. 1ª edição. 194 p. Kluwer Academic Publishers, Massachusetts, 1995.

HAAS, W. Technological Evolution in Transmission Systems. **Electrical Communications**, 1977. 283-288.

HANZO, L. L. et al. **Quadrature Amplitude Modulation: From Basics to Adaptive Trellis-Coded, Turbo-Equalised and Space-Time Coded OFDM, CDMA and MC-CDMA Systems**. [S.l.]: Wiley-IEEE Press, 2004. 1136 p.

KAFIG, W. **VHDL 101: Everything You Need to Know to get Started**. [S.l.]: Elsevier Inc, 2011. 217 p.

LOURENÇO, J. F. L. S. **Implementação em FPGA da Modulação/Demodulação OFDM 801.11p**. Aveiro: Universidade de Aveiro, 2011. 124 p.

MEYER-BAESE, U. **Digital Signal Processing with Field Programmable Gate Arrays**. 3. Ed. Nova Yorque: Springer Berlin Heidelberg, 2007. 795 p.

MITOLA, J. **The Software Radio Architecture**. [S.l.]: Communications Magazine, IEEE, v. 33, 1995. 26-38 p.

NETO, L. A. **Transmissão de sinais OFDM através de Fibras Ópticas Poliméricas (POFs) utilizando LEDS de Iluminação**. Niterói: Universidade Federal Fluminense, 2009.

OLIVEIRA, C. A. **Dispositivos Lógicos Programáveis**. Guaratinguetá: Universidade Estadual Paulista, 2011.

RIBEIRO, J. A. J. **Comunicações Ópticas**. 4ª. ed. São Paulo: Editora Érica, 2012.

SIQUEIRA, T. M. **Implementação de um Modem OFDM em FPGA**. Vitória: Universidade do Espírito Santo, 2004. 61 p.

VAN NEE, R. PRASAD, R. **OFDM for Wireless Multimedia Communications**. Artech House, 2000.

WIKIPEDIA. **Pipeline (Unix)**. 2014. Disponível em: <http://en.wikipedia.org/wiki/Pipeline_(Unix)>. Acessado em 2014.

ANEXO A – SIMETRIA HERMITIANA

Pela definição da transformada discreta inversa de Fourier, sabe-se que:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]\, e^{j\frac{2\pi nk}{N}} \quad (k \in \mathbb{Z} \,\forall\, 0 \leq k \leq N-1), \tag{A.1}$$

onde $X[k]$ representa o sinal na saída do bloco da IDFT, $x[n]$ representa o símbolo que modulará a n-ésima subportadora e N o número de subportadoras.

Na expansão de $X[k]$, obtem-se:

$$X[k] = \frac{1}{N}\Big(x[0] + x[1]e^{j\frac{2\pi}{N}k} + x[2]e^{j\frac{2\pi}{N}2k} + \cdots + x[N-2]e^{j\frac{2\pi}{N}(N-2)k} + x[N-1]e^{j\frac{2\pi}{N}(N-1)k}\Big). \tag{A.2}$$

Sabe-se, ainda, que para $\alpha \in \mathbb{N}$ e $0 < \alpha < N$:

$$e^{j\frac{2\pi}{N}(N-\alpha)k} = e^{-j\frac{2\pi}{N}\alpha k}.$$

Portanto, a saída da IDFT pode ser reescrita como:

$$X[k] = \frac{1}{N}\Big(x[0] + x[1]e^{j\frac{2\pi}{N}k} + x[2]e^{j\frac{2\pi}{N}2k} + \cdots + x[N-2]e^{-j\frac{2\pi}{N}2k} + x[N-1]e^{-j\frac{2\pi}{N}k}\Big). \tag{A.3}$$

É de se esperar que, se a entrada da IDFT, $x[n]$, for complexa, a saída, $X[k]$, poderá ser complexa, conforme a paridade da função, e, portanto, para a transmissão do símbolo OFDM, necessitarão de duas portadoras, uma em fase, para a transmissão da parte real, e outra em quadratura, para a transmissão da parte imaginária. Porém, pode-se simplificar o processo se a saída da IDFT for puramente real.

Da equação (A.3), pode-se observar que se:

$$x[0] = x\left[\frac{N}{2}\right] = 0,$$
$$x[N-\alpha] = x^*[\alpha] \quad \left(\alpha \in \mathbb{Z}\ \forall\ 1 \leq \alpha \leq \frac{N}{2} - 1\right), \tag{A.4}$$

onde o elemento $x[0]$ é denominado elemento (ou nível) de tensão contínua, pois está associado à subportadora de frequência zero e deve possuir valor puramente real. Como deseja-se não desperdiçar potência do transmissor em componentes contínuas, atribui-se o valor zero a este elemento. Então, obtém-se

$$X[k] = \frac{1}{N}\left(x[1]e^{j\frac{2\pi}{N}k} + x[2]e^{j\frac{2\pi}{N}2k} + \cdots + x^*[2]e^{-j\frac{2\pi}{N}2k} + x^*[1]e^{-j\frac{2\pi}{N}k}\right). \tag{A.5}$$

Agrupando cada termo com seu respectivo conjugado, tem-se

$$X[k] = \frac{1}{N}\left[\sum_{n=1}^{\frac{N}{2}-1}\left(x[n]e^{j\frac{2\pi n}{N}k} + x^*[n]e^{-j\frac{2\pi n}{N}k}\right)\right]. \tag{A.6}$$

Escrevendo os valores complexos $x[n]$ em sua forma polar, $|x[n]|e^{j\theta_n}$, onde θ_n é a fase do valor $x[n]$, tem-se

$$X[k] = \frac{1}{N}\left[\sum_{n=1}^{\frac{N}{2}-1}\left(|x[n]|e^{j\left(\theta_n+\frac{2\pi n}{N}k\right)} + |x^*[n]|e^{-j\left(\theta_n+\frac{2\pi n}{N}k\right)}\right)\right]$$
$$= \frac{1}{N}\left\{\sum_{n=1}^{\frac{N}{2}-1}|x[n]|\left[e^{j\left(\theta_n+\frac{2\pi n}{N}k\right)} + e^{-j\left(\theta_n+\frac{2\pi n}{N}k\right)}\right]\right\}. \tag{A.7}$$

Da forma exponencial das funções trigonométricas, sabe-se que

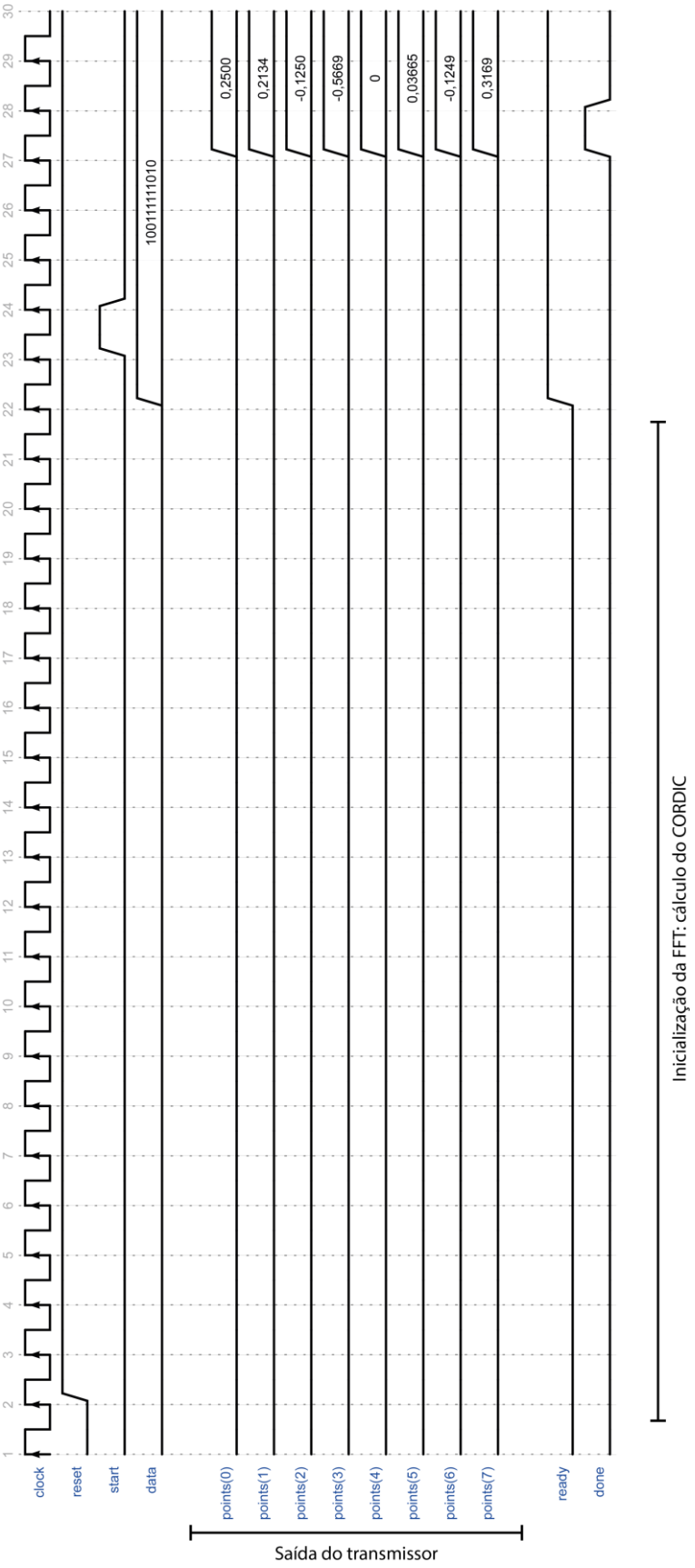
$$\cos(\theta) = \frac{e^{j\theta} + e^{-j\theta}}{2},$$

então, tem-se, a partir da equação (A.7):

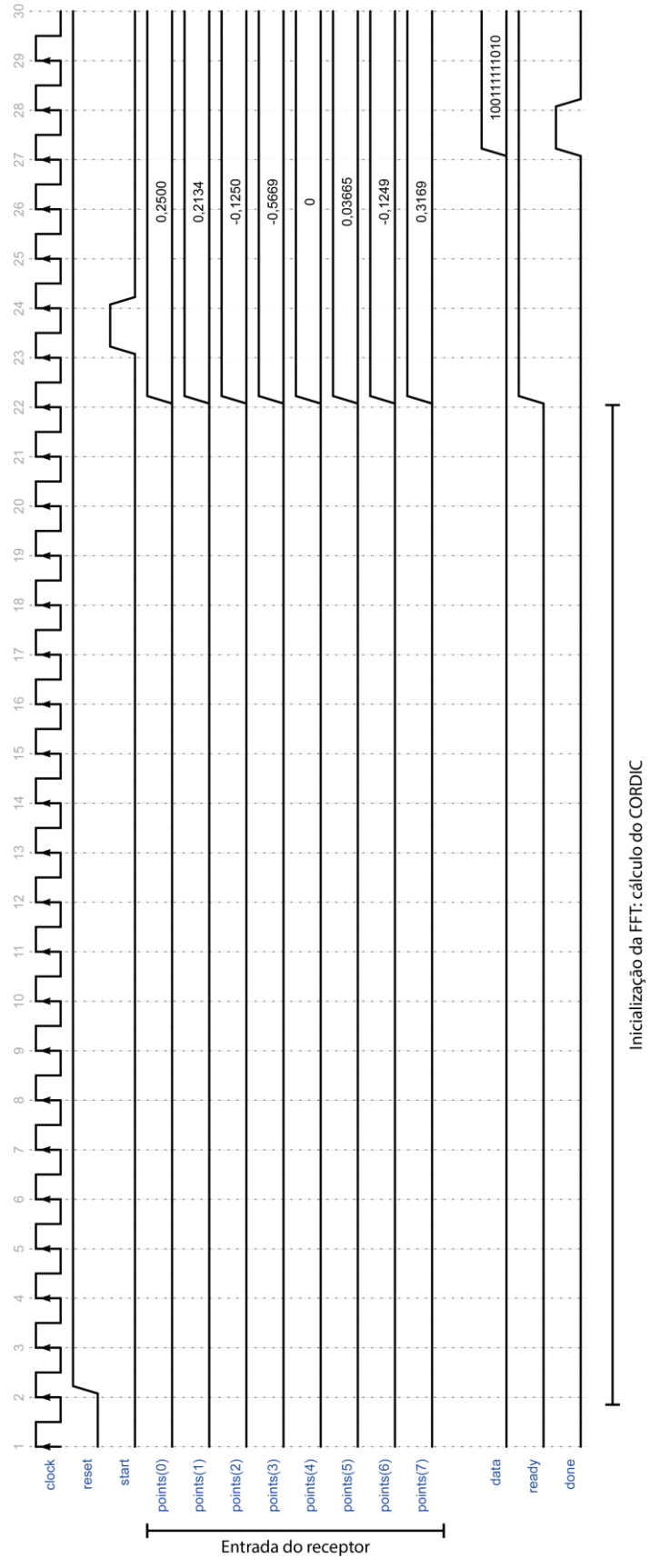
$$X[k] = \frac{1}{N}\left\{\sum_{n=1}^{\frac{N}{2}-1} |x[n]| \left[2cos\left(\theta_n + \frac{2\pi n}{N}k\right)\right]\right\}.$$

Ou seja, as operações matemáticas executadas pela IDFT sobre os termos de um vetor que satisfaz as condições impostas em (A.4) fazem com que as partes imaginárias se anulem, resultando em um vetor puramente real, simplificando o processo de transmissão, pois requererá apenas uma portadora. Tais condições definem a simetria Hermitiana. É visto ainda que para se gerar uma saída com N valores reais, é preciso $\frac{N}{2} - 1$ valores complexos de entrada, sendo que as outras entradas são definidas a partir da simetria. Ou melhor, dada uma entrada de comprimento N, pode-se aplicar a simetria Hermitiana para se gerar uma saída com $2N + 2$ valores reais.

ANEXO B – FORMAS DE ONDA DO TRANSMISSOR



ANEXO C – FORMAS DE ONDA DO RECEPTOR



ANEXO D – CÓDIGO VHDL (*TESTBENCH*) DO TRANSMISSOR

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.complex_pkg.all;

entity transmitter_tb is
    generic (
        constant CHANNELS: natural := 3
    );
end entity transmitter_tb;

architecture Behavior of transmitter_tb is
    signal clock: std_logic := '0';
    signal reset: std_logic := '0';
    signal start: std_logic := '0';
    signal data: std_logic_vector(0 to 4*CHANNELS-1)
          := (others => '0');
    signal points: complex_vector(0 to 2*CHANNELS+1);
    signal ready: std_logic;
    signal done: std_logic;
begin

    clock <= not clock after 10 ns;
    reset <= '1' after 20 ns;

    dut: entity work.Transmitter
        generic map (CHANNELS)
        port map
          (clock, reset, start, data, points, ready, done);
```

```
    process
    begin
        -- Aguarda a inicialização da FFT:
        wait until ready = '1';

        -- Dados a serem transmitidos:

        data <= "100111111010";

        wait until rising_edge(clock);

        -- Inicia a transmissão:
        start <= '1';
        wait until rising_edge(clock);
        start <= '0';

        -- Aguarda finalizar a transmissão:
        wait until done = '1';

    end process;
end architecture Behavior;
```

ANEXO E – CÓDIGO VHDL (*TESTBENCH*) DO RECEPTOR

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.float_pkg.all;
use work.complex_pkg.all;

entity receptor_tb is
    generic (
        constant CHANNELS: natural := 3
    );
end entity receptor_tb;

architecture Behavior of receptor_tb is
    signal clock: std_logic := '0';
    signal reset: std_logic := '0';
    signal start: std_logic := '0';
    signal points: complex_vector(0 to 2*CHANNELS+1);
    signal data: std_logic_vector(0 to 4*CHANNELS-1)
          := (others => '0');
    signal ready: std_logic;
    signal done: std_logic;
begin

    clock <= not clock after 10 ns;
    reset <= '1' after 20 ns;

    dut: entity work.Receptor
        generic map (CHANNELS)
        port map
          (clock, reset, start, points, data, ready, done);

    process
```

```
    begin
        -- Aguarda a inicialização da FFT:
        wait until ready = '1';

        -- Valores complexos de entrada do receptor:
        points(0) <= (to_float(0.2500),  to_float(0));
        points(1) <= (to_float(0.2134),  to_float(0));
        points(2) <= (to_float(-0.1250), to_float(0));
        points(3) <= (to_float(-0.5669), to_float(0));
        points(4) <= (to_float(0),       to_float(0));
        points(5) <= (to_float(0.0365),  to_float(0));
        points(6) <= (to_float(-0.1249), to_float(0));
        points(7) <= (to_float(0.3169),  to_float(0));

        wait until rising_edge(clock);

        -- Inicia o tratamento dos valores:
        start <= '1';
        wait until rising_edge(clock);
        start <= '0';

        -- Aguarda finalizar a recuperação dos dados:
        wait until done = '1';

    end process;
end architecture Behavior;
```

ANEXO F – CÓDIGO VHDL (*TESTBENCH*) DO SISTEMA

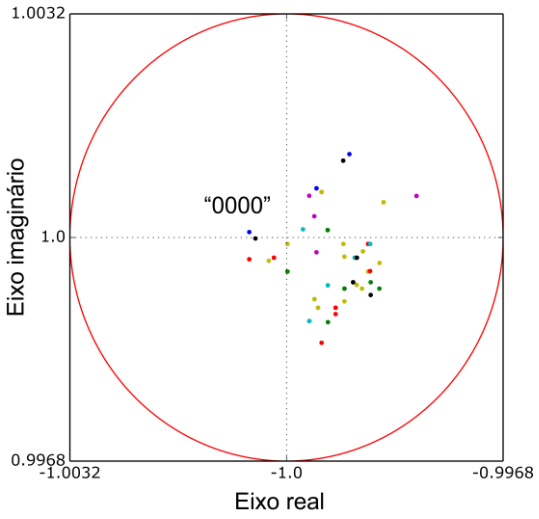
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;
use std.textio.all;

entity modem_tb is
    generic (
        constant CHANNELS: natural := 3
    );
end entity modem_tb;

architecture Behavior of modem_tb is
    -- Instâncias dos arquivos de entrada e saída:
    file input_file: text;
    file output_file: text;

    signal clock: std_logic := '0';
    signal reset: std_logic := '0';
    signal start: std_logic := '0';
    signal data_in:  std_logic_vector(4*CHANNELS-1 downto 0)
          := (others => '0');
    signal data_out: std_logic_vector(4*CHANNELS-1 downto 0);
    signal ready: std_logic;
    signal done: std_logic;
begin

    clock <= not clock after 10 ns;
    reset <= '1' after 20 ns;
```

```
-- Leitura do arquivo de entrada:
process
    variable input_line: line;
    variable input_data: std_logic_vector(data_in'range);
begin
    -- Aguarda a inicialização da FFT:
    wait until ready = '1';
    -- Abre o arquivo em modo de leitura:
    file_open(input_file, "input.txt", read_mode);

    while not endfile(input_file) loop
        -- Lê uma linha do arquivo:
        readline(input_file, input_line);
        read(input_line, input_data);
        -- Atribui o valor lido ao sinal de entrada:
        data_in <= input_data;
        -- Inicia a transmissão dos dados (símbolo):
        start <= '1';
        wait until rising_edge(clock);
        start <= '0';
        wait until rising_edge(clock);
    end loop;

    file_close(input_file);
end process;

-- Escrita do arquivo de saída:
process
    variable output_line: line;
    variable output_data:
           std_logic_vector(data_in'range);
begin
```

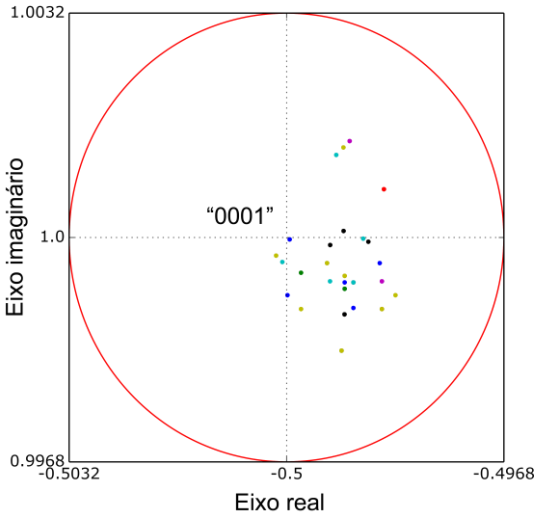
```
        -- Aguarda finalizar a recuperação dos dados:
        wait until done = '1';
        -- Abre o arquivo em modo de anexo:
        file_open(output_file, "output.txt", append_mode);

        -- Atribui os dados recuperados para escrita:
        output_data := data_out;

        -- Escreve os dados em uma linha do arquivo:
        write(output_line, output_data);
        writeline(output_file, output_line);

        file_close(output_file);

    end process;

    dut: entity work.Modem
        generic map (CHANNELS)
        port map
     (clock, reset, start, data_in, data_out, ready, done);

end architecture Behavior;
```

ANEXO G – GRÁFICOS DE ESPALHAMENTO DOS RESULTADOS

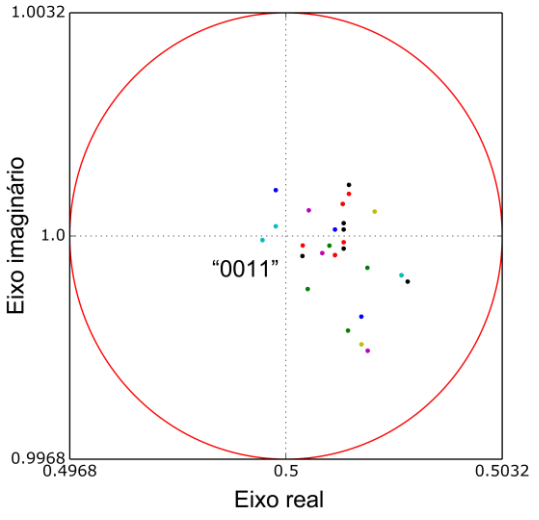
Ponto (*-1,0 + j 1,0*)



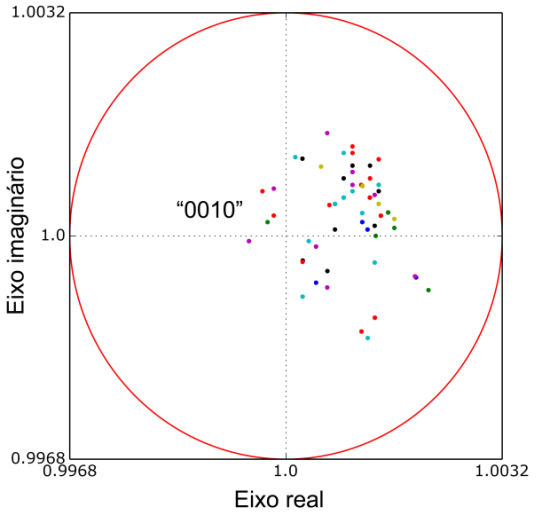
Ponto (*-0,5 + j 1,0*)



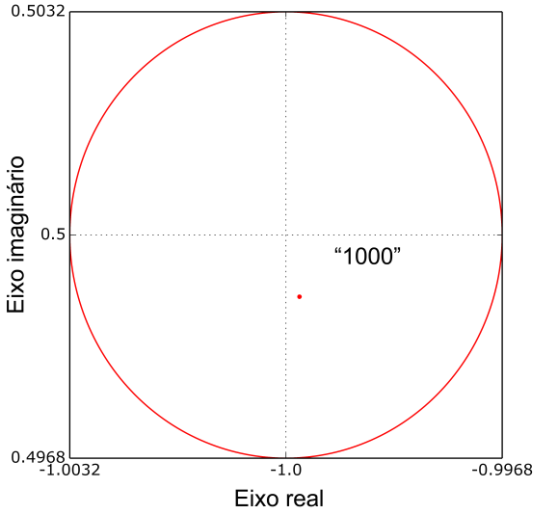
Ponto (*0,5 + j 1,0*)



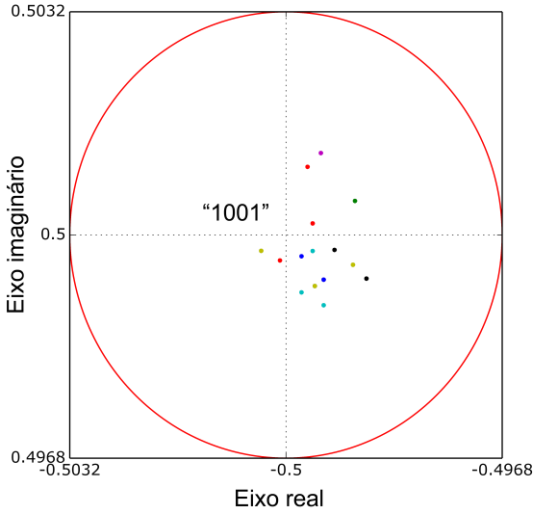
Ponto (*1,0 + j 1,0*)

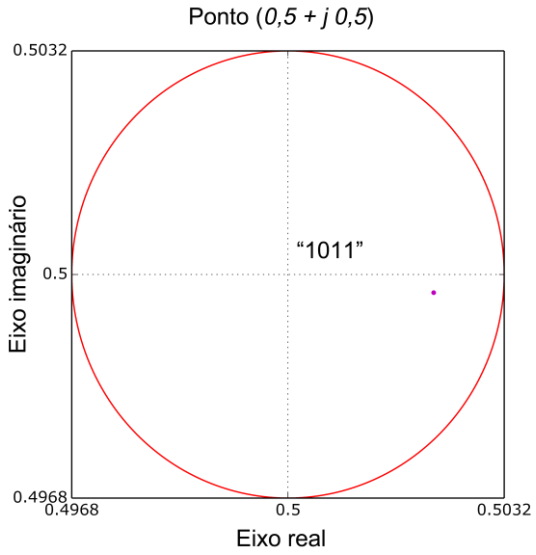


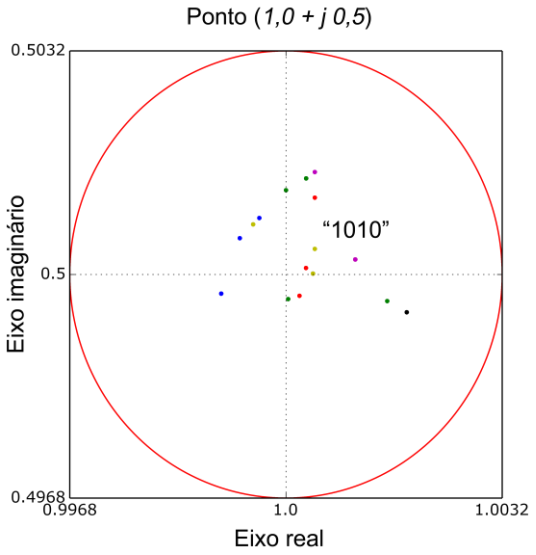
Ponto (*-1,0 + j 0,5*)

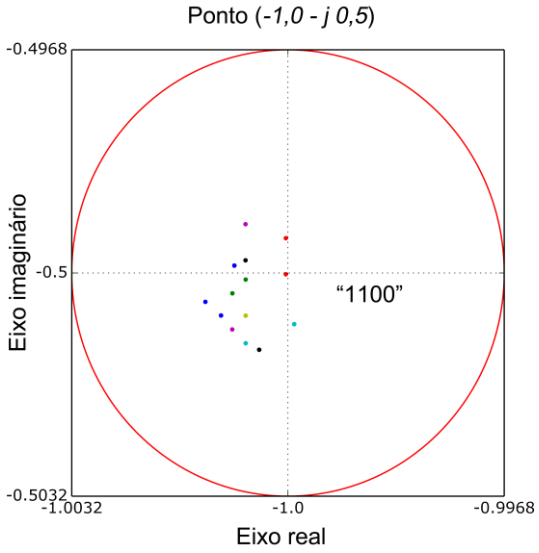


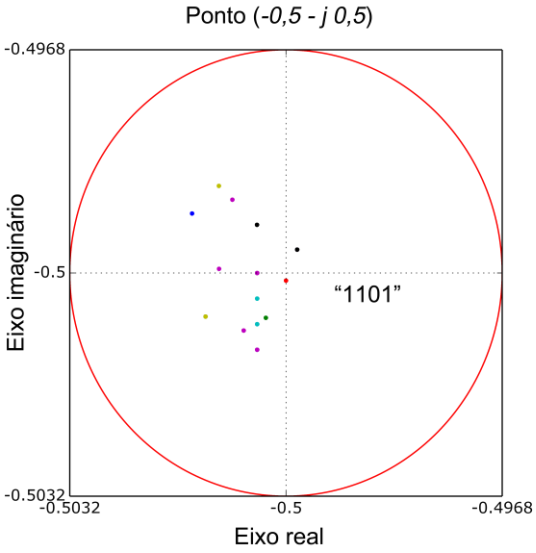
Ponto (*-0,5 + j 0,5*)

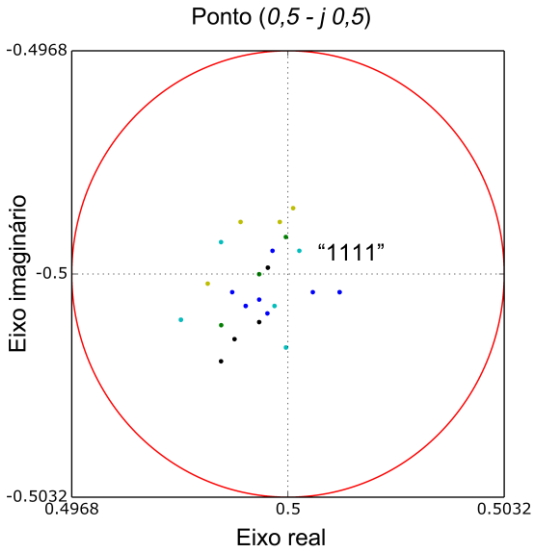


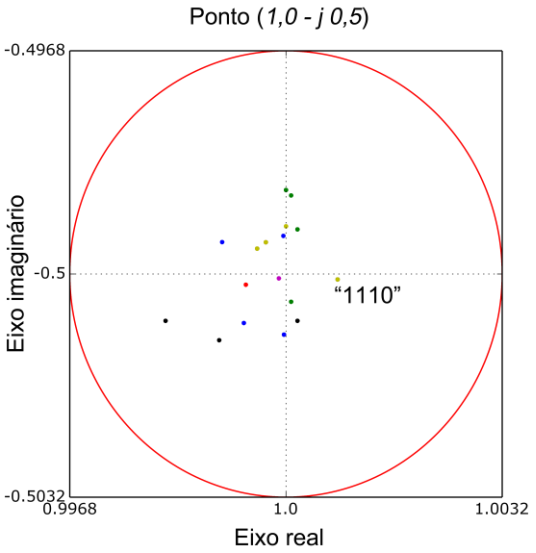

Ponto (0,5 + j 0,5)
0.5032
0.5
0.4968
Eixo imaginário
"1011"
0.4968
0.5
0.5032
Eixo real

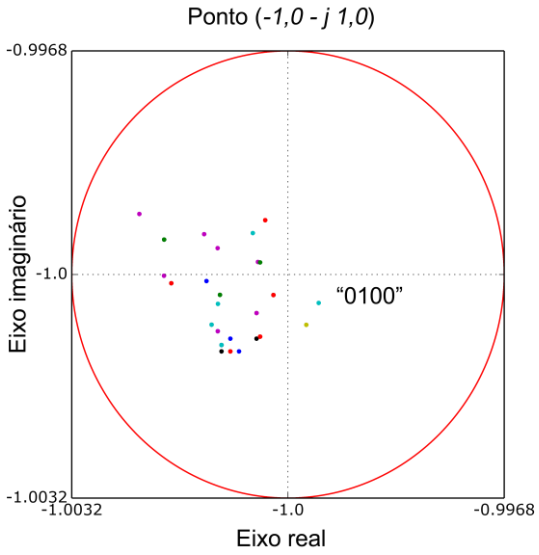

Ponto (1,0 + j 0,5)
0.5032
0.5
0.4968
Eixo imaginário
"1010"
0.9968
1.0
1.0032
Eixo real

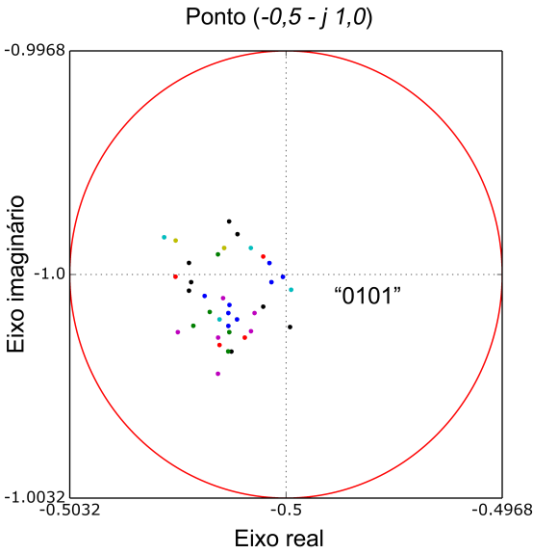

Ponto (-1,0 - j 0,5)
-0.4968
-0.5
-0.5032
Eixo imaginário
"1100"
-1.0032
-1.0
-0.9968
Eixo real

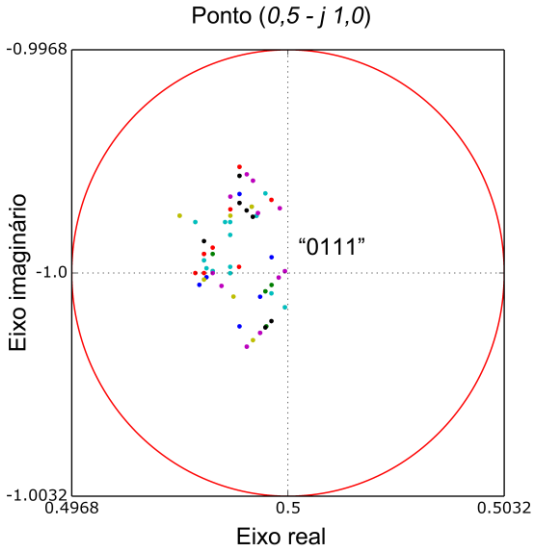

Ponto (-0,5 - j 0,5)
-0.4968
-0.5
-0.5032
Eixo imaginário
"1101"
-0.5032
-0.5
-0.4968
Eixo real

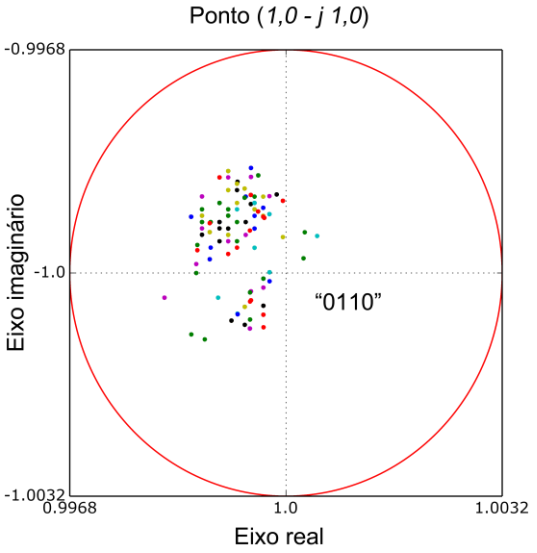

Ponto (0,5 - j 0,5)
-0.4968
-0.5
-0.5032
Eixo imaginário
"1111"
0.4968
0.5
0.5032
Eixo real


Ponto (1,0 - j 0,5)
-0.4968
-0.5
-0.5032
Eixo imaginário
"1110"
0.9968
1.0
1.0032
Eixo real


Ponto (-1,0 - j 1,0)
-0.9968
-1.0
-1.0032
Eixo imaginário
"0100"
-1.0032
-1.0
-0.9968
Eixo real


Ponto (-0,5 - j 1,0)
-0.9968
-1.0
-1.0032
Eixo imaginário
"0101"
-0.5032
-0.5
-0.4968
Eixo real


Ponto (0,5 - j 1,0)
-0.9968
-1.0
-1.0032
Eixo imaginário
"0111"
0.4968
0.5
0.5032
Eixo real


Ponto (1,0 - j 1,0)
-0.9968
-1.0
-1.0032
Eixo imaginário
"0110"
0.9968
1.0
1.0032
Eixo real