Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Engenharia Elétrica
Projeto de Graduação

Implementação de um modem OFDM em FPGA

Tonny Matos Siqueira

VITÓRIA

ABRIL/2004

Tonny Matos Siqueira

Implementação de um modem OFDM em FPGA

Parte manuscrita do Projeto de Graduação do aluno Tonny Matos Siqueira, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, para obtenção do grau de Engenheiro Eletricista.

Orientador:
Marcelo Eduardo Vieira Segatto, Ph. D.

Universidade Federal do Espírito Santo

VITÓRIA

ABRIL/2004

Tonny Matos Siqueira

Implementação de um modem OFDM em FPGA

Comissão Examinadora

Marcelo Eduardo Vieira Segatto, Ph. D.
Universidade Federal do Espírito Santo
Orientador

Evandro Ottoni Teatini Salles, D. Sc.
Universidade Federal do Espírito Santo
Examinador

Ailson Rosetti de Almeida, Ph. D.
Universidade Federal do Espírito Santo
Examinador

Vitória, 20 de Abril de 2004

Dedicatória

À minha família,
que me ajudou e me compreendeu durante todos esses anos.

Agradecimentos

Aos professores,
que me ajudaram a trilhar esse longo caminho.

À UFES,
pelo suporte e material utilizado.

Aos amigos do LABTEL,
que me fizeram rir bastante.

E a alguém em especial,
que sem ela esse trabalho não estaria pronto.

Lista de Figuras

1 Espectro SCM e MCM. 6

2 Modulações ASK, FSK, PSK e QAM 7

3 Constituição do QAM 7

4 Esquemático do modulador QAM 8

5 Diagramas de constelação ASK e PSK 9

6 Diagramas de constelação QAM 9

7 Espectro de um sinal QAM — $sinc(x)$ 10

8 Resposta em freqüência de um canal não linear 10

9 Divisão de um canal não linear em vários subcanais 11

10 Um transmissor MCM simplificado. 12

11 Espectro dos vários sistemas MCM 13

12 OFDM utilizando filtragem FDM 14

13 OFDM utilizando modulação SQAM 14

14 Resposta impulsiva do canal 19

15 Cyclic prefix 19

16 Modulador básico OFDM 21

17 Demodulador básico OFDM 22

18 A FPGA Spartan II 25

19 Constelação no receptor 27

20 Implementação da simetria Hermitiana 29

21 Número de operações da DFT e da FFT 30

22 Decimação da *butterfly* 32

23 Implementação da FFT de 8 pontos 32
24 Implementação da Butterfly 33
25 Comportamento do fator K do CORDIC 37
26 Butterfly Radix-4 39
27 Comparação entre a FFT radix-2 e radix-4 39
28 Diagrama de blocos de uma implementação básica da FFT 40
29 Diagrama de blocos de uma implementação da FFT com 3 RAMs 41
30 FFT radix-4 DF 43
31 A FFT integrada com a simetria hermetiana 44
32 FFT com a simetria e codificador 45
33 Transmissor OFDM 45
34 Receptor OFDM 46
35 Diagrama de blocos da FFT 47
36 Representação dos cálculos efetuados em um estágio 49
37 Representação da FFT 64 49
38 Simulação do CORDIC 50
39 O sistema OFDM testado 55
40 Disposição em hardware do modem teste 58

Lista de Tabelas

1 Tabela de conversão 4-QAM 27

2 Simetria hermetiana aplicada em 3 canais 28

3 Digit Reverse 42

4 Conversão para leitura da hermetiana 44

5 Tabela de conversão 4-QAM implementada 53

6 Comparação de utilização do hardware entre o CORDIC implementado e o gerado pelo COREGEN 56

7 Utilização do hardware da parte de cálculo da FFT 57

8 Comparação de utilização do hardware entre o FFT implementado e o gerado pelo COREGEN 57

9 Utilização do hardware dos modems OFDM de transmissão e recepção . . . 57

10 Utilização do hardware dos modem utilizado para teste 58

Lista de Siglas

ADSL	Asymmetric Digital Subscriber Line
ASIC	Application-Specific Integrated Circuit
ASK	Amplitude-Shift Keying
CLB	Configurable Logic Blocks
CORDIC	Coordinate Rotation Digital Computer
CPLD	Complex Programmable Logic Devices
DAB	Digital Audio Broadcast
DCM	Digital Clock Manager
DF	Decimação na freqüência
DFT	Discrete Fourier Transform
DLL	Delay-Locked Loops
DMT	Discrete MultiTone
DT	Decimação no tempo
DSL	Digital Subscriber Line
DSP	Digital Signal Processors
DVB	Digital Video Broadcast
FDM	Frequency-division multiplexing
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
FSK	Frequency-Shift Keying
HDL	Hardware Description Language
HDSL	High Bit Rate Digital Subscriber Line

ICI	Inter-Channel Interference
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
IIR	Infinite impulse response
IOB	Input Output Blocks
ISI	Inter-Symbol Interference
LC	Logic Cell
LUT	Look Up Table
MAC	Multiply And Accumulate
MCM	Multi-Carrier Modulation
OFDM	Orthogonal Frequency Division Multiplexing
OQAM	Offset QAM
PAL	Programmable Array Logic
PLA	ProgrammableLogic Array
PLC	Power Line Communication
PLD	Programmable Logic Devices
PROM	Programmable Read-Only Memory
PSK	Phase-Shift Keying
QAM	Quadrature Amplitude Modulation
RAM	Random access memory
ROM	Read-Only Memory
SCM	Single-Carrier Modulation
SPLD	Simple Programmable Logic Devices
SQAM	Staggered QAM
TCM	Trellis-Coded Modulation
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Sumário

Resumo . . . . . **xii**

1 Introdução **1**

2 Dispositivos lógicos programáveis **2**

2.1 FPGA 3

2.2 VHDL 4

3 OFDM — Orthogonal Frequency Division Multiplex **5**

3.1 Single e multi-carrier modulation 6

3.2 *Single-Carrier Modulation* 6

3.2.1 Modulação QAM 7

3.2.1.1 O modulador 8

3.2.1.2 Diagrama de constelação 8

3.2.1.3 Espectro QAM 9

3.2.2 Desvantagens do sistema SCM 10

3.3 *Multi-Carrier Modulation* 11

3.3.1 O principio básico do MCM 12

3.3.2 Implementações do MCM 13

3.3.2.1 Filtragem FDM 13

3.3.2.2 Modulador SQAM 14

3.3.2.3 Sobreposição do espectro QAM 15

3.4 Modulação via IDFT 15

3.5 Manutenção da ortogonalidade 18
3.6 A técnica OFDM 20
3.6.1 O transmissor OFDM 20
3.6.2 O receptor OFDM 21
3.6.3 O modem OFDM 23

4 Implementação do OFDM 24
4.1 O Hardware — A FPGA Spartan II XC2S200 24
4.2 Serial x paralelo 25
4.3 Mapeando a constelação 26
4.3.1 Codificador da constelação 26
4.3.2 Decodificador da constelação 27
4.4 A simetria hermetiana 28
4.5 A (I)FFT 30
4.5.1 Decimação no tempo e na freqüência 31
4.5.2 Implementação da *Butterlfy* 32
4.5.2.1 Tabela de senos 34
4.5.2.2 Cordic 35
4.5.3 Radix-2 38
4.5.4 Radix-4 38
4.5.5 Utilização da memória 40
4.5.6 Mapeamento da memória 41
4.6 O cyclic-prefix 42
4.7 Conversor digital-analógico e analógico-digital. 42
4.8 Melhorias 43
4.9 Quantidade de memória 46
4.10 A implementação 46

5 Análises 49

5.1 Fator de correção 49

5.2 Valores máximos 51

5.3 Ponto fixo 52

5.4 Taxas de transmissão 53

5.5 Análise da Implementação 54

5.6 Utilização do hardware 56

5.6.1 CORDIC 56

5.6.2 FFT 56

5.6.3 FFT com controlador 56

5.6.4 Modems OFDM de transmissão e recepção 57

5.6.5 Modem de teste 57

5.6.6 Floorplan do modem teste 58

6 Conclusões 59

Referências 60

Resumo

O desenvolvimento tanto de técnicas avançadas de transmissão digital de dados como dos atuais FPGAs (*Field programmable gate arrays*) tem tornado realidade o desenvolvimento de sistemas de comunicações totalmente digital, implementados em um único chip.

Para fazer uso dessas modernas tecnologias de transmissão de dados, é necessário que se desenvolva eficientes técnicas de modulação digital. A que mais tem se destacado é a OFDM (*Orthogonal Frequency Division Multiplexing*), principalmente por que utiliza a, extremamente eficiente, FFT para fazer a modulação.

Este trabalho descreve como se implementar um modem OFDM em FPGA utilizando VHDL. É feita um rápida descrição do OFDM, e mostra-se várias formas de se implementar esse em uma FPGA. Entre essas, escolheu-se a que utiliza uma FFT radix-4 com decimação em freqüência e utilizando o CORDIC para fazer os cálculos necessários.

É analisada várias características do modem OFDM desenvolvido, mostrado e comparado a ocupação em hardware das várias partes que o compõem. Chegando-se a conclusão de que as partes principais desenvolvidas para o sistema é mais eficiente do que o disponibilizado pelo fabricante do FPGA, a Xilinx, chegando a ter um economia de hardware de até 25%. Sendo possível implementar esse modem em um FPGA de baixo custo, como a SPARTAN II, ao invés de se utilizar grandes e caras FPGAs como os da família VIRTEX.

Palavras-chave: OFDM, FPGA, VHDL, FFT, CORDIC.

1 Introdução

Por um longo tempo o objetivo dos desenvolvedores de sistemas de comunicações era fazer o transmissor e o receptor totalmente digital, consistindo somente de antenas e um circuitos totalmente programável com filtros digitais, moduladores, demoduladores, codificadores e decodificadores de correção de erro, tudo em um único chip. Com o desenvolvimento tanto de técnicas de transmissão digital de dados[1] como dos atuais FPGAs (*Field programmable gate arrays*), com milhões de portas lógicas, isso tem se tornado realidade.

Os FPGAs estão no topo da revolução do processamento digital de sinais da mesma forma que os processadores programáveis, DSPs (*Digital Signal Processors*), estavam a duas décadas atrás. Muitos algoritmos de processamento digital de sinais, como as FFTs, filtros FIR ou IIR, para citar apenas alguns, anteriormente feitos com ASICs ou DSPs, estão sendo convertidos para FPGAs. Modernas famílias de FPGAs prevêem suporte para implementação de DSP com *fast-carry chain* (FCA) o qual são usados para implementar MACs (*multiply-accumulates*) em alta velocidade e baixo custo entre vários outros recursos interessantes.

Para fazer uso dessas modernas tecnologias para a transmissão de dados, é necessário que se desenvolva eficientes técnicas de modulação digital. Uma das técnicas que mais tem se destacado é a OFDM (*Orthogonal Frequency Division Multiplexing*), principalmente por que utiliza a, extremamente eficiente, FFT (*Fast Fourier Transform*) para fazer a modulação.

A técnica OFDM tem sido bastante utilizada ultimamente, principalmente em aplicações que requerem alta taxa de transmissão, como por exemplo: modems PLC (Power Line Communication) e ADSL, redes wireless como Wi-Fi (802.11) e HiperLAN/2, transmissões digitais de video, DVB (Digital Video Broadcast) e audio, DAB (Digital Audio Broadcast), entre outros.

[1] Onde toda a modulação é feita digitalmente.

2 Dispositivos lógicos programáveis

Toda função lógica, seja combinacional ou seqüencial, pode ser reduzida a uma simples representação na forma de soma de produtos. Dessa forma, qualquer função lógica complexa pode ser reduzida facilmente em lógicas discretas Es-OUs. Esta propriedade das funções lógicas possibilitou o desenvolvimento de componentes eletrônicos programáveis, chamados de dispositivos lógicos programáveis, PLD (*Programmable Logic Devices*).

Os primeiros dispositivos programáveis surgiram na década de 70 através do desenvolvimento da tecnologia de memórias ROM programáveis (PROM's). Desenvolveu-se, depois, dispositivos programáveis que continham arranjos uniformes de Es-OUs, ambos com as conexões programáveis de uma maneira clara e eficiente, para a obtenção da função desejada, chamados de PLA[1] (*Programmable Logic Array*). Contudo, essa configuração tornava o tempo de propagação do sinal no chip muito grande. Para resolver esse problema foi desenvolvido o PAL (*Programmable Array Logic*), que é um PLD com a lógica OU fixa, sendo possível apenas configurar as conexões da lógica E, perdendo assim um pouco da generalidade porém diminuindo o tempo de propagação.

Com a finalidade de aumentar a capacidade de implementação lógica nos dispositivos programáveis, surgiram novas arquiteturas que incorporam flip-flops em suas estruturas para facilitar o desenvolvimento de máquinas de estado e circuitos com lógica síncrona como os SPLDs (*Simple PLD*) e os CPLDs (*Complex PLD*).

Em geral, os PLDs são circuitos que são configurados pelo usuário para fazer um função lógica. Por não apresentarem uma função lógica definida, até que sejam configurados, diferem dos circuitos integrados personalizáveis por máscaras (ASIC, CUSTOM ou SEMICUSTOM), tornando o desenvolvimento muito mais rápido, barato e eficiente. Perde-se, porém, em generalidade e velocidade.

Os dispositivos de programação utilizados nos PLDs são elementos de programação

que permitem conectar, ou não, dois pontos do circuito. Comumente estes elementos são elos fusíveis, anti-fusíveis, células SRAM, transistores EPROM e EEPROM. Assim, alguns destes dispositivos podem ser reprogramados e outros são de programação única.

2.1 FPGA

Em 1985, a Xilinx introduziu uma idéia completamente nova[2], combinar o controle do usuário e a rapidez de desenvolvimento dos PLDs com a densidade e benefícios de custo dos *gate arrays*, criando assim o FPGA (*Field Programmable Gate Array*).

Um FPGA possui uma estrutura regular de células lógicas SRAM e interconexões em que o desenvolvedor possui total controle. Assim, o usuário pode desenvolver, programar e alterar o seu circuito quantas vezes quiser. Os mais recentes FPGAs possuem mais de 10 milhões de portas lógicas (Xilinx Virtex II). Dessa forma, é possível desenvolver grandes projetos em um único chip.

Nas células lógicas SRAM, ao invés de se usar as portas lógicas convencionais, é utilizado um LUT (*Look Up Table*) que determina a saída baseado nos valores de entrada. Foi incluído também flip-flops e portas tri-state nessas células para torná-las versáteis e completas.

Além das células lógicas, na maioria dos FPGAs existem outros componentes que aumentam suas funcionalidades, como por exemplo:

- Blocos de memórias RAM
- DLLs
- DCMs
- Multiplicadores
- Entre outros.

Recentemente foi lançado um novo FPGA (Virtex II Pro[1] da Xilinx) que, além de todos esses componentes, ainda possui um processador PowerPC 405 no seu hardware.

O FPGA é uma alternativa superior aos circuitos integrados personalizáveis por máscaras (ASIC, CUSTOM ou SEMICUSTOM). O FPGA evita o custo inicial, o grande

[1] http://www.xilinx.com/virtex2pro/

tempo de desenvolvimento, e o risco inerente dos convencionais ASICs. Também, a programabilidade dos FPGAs permite *upgrades* no campo sem a necessidade de troca do hardware, qualidade impossível para os ASICs.

2.2 VHDL

Com o aumento da quantidade de recursos dos PLDs e dos FPGAs tornou-se extremamente difícil programá-los.

Para resolver esse problema criou-se meios de se projetar a função lógica desejada, destacando-se: diagramas esquemáticos, tabelas verdade, equações booleanas, linguagem de descrição de hardware, HDL (*Hardware Description Language*), entre outros.

A tendência em desenvolvimento em hardware é a migração dos desenvolvimento gráfico para linguagem de descrição de hardware, HDL. Embora muitas lógicas possam ser descritas com os diagramas esquemáticos ou qualquer outro meio de entrada gráfica, tem sido percebido que o reuso de código é muito maior com sistemas baseados em HDL do que com os de entrada gráfica.

O VHDL[3] (*VHSIC Hardware Description Language* onde VHSIC significa *Very High Speed Integrated Circuit*) é um HDL padrão da indústria usada para descrever o hardware de uma forma abstrata do seu nível concreto. VHDL resulta de um trabalho feito na década de 70 e início de 80 pelo Departamento de Defesa dos Estados Unidos, sendo baseado na linguagem ADA.

A linguagem VHDL teve um rápido crescimento desde sua criação e tem sido usada por milhares de engenheiros no mundo para criar sofisticados produtos eletrônicos.

Em 1986, o VHDL foi proposto como um padrão do IEEE. Após um grande número de revisões e mudanças, finalmente foi adotada como o padrão 1076 do IEEE em dezembro de 1987 como o padrão VHDL 1076-1987 do IEEE. Em 1993 foi lançado um novo padrão, o 1076-1993.

3 OFDM — Orthogonal Frequency Division Multiplex

Basicamente, a técnica OFDM (*Orthogonal Frequency Division Multiplexing*) consiste em dividir o fluxo de dados de entrada a ser transmitidos entre vários canais paralelos e ortogonais. É devido a isso que se dá o nome OFDM que em inglês significa multiplexação por divisão em freqüência ortogonal. Em cada um desses canais é feito uma modulação digital, cada uma com uma portadora com freqüência diferente, de tal forma que um canal não interfira em outro, mantendo assim a ortogonalidade.

A forma de se implementar e garantir a ortogonalidade de um sistema desse tipo pode variar bastante, indo desde filtros passa faixa em cada freqüência até técnicas mais sofisticadas como a utilização da FFT com intervalo de guarda, que é a utilizada pelo OFDM.

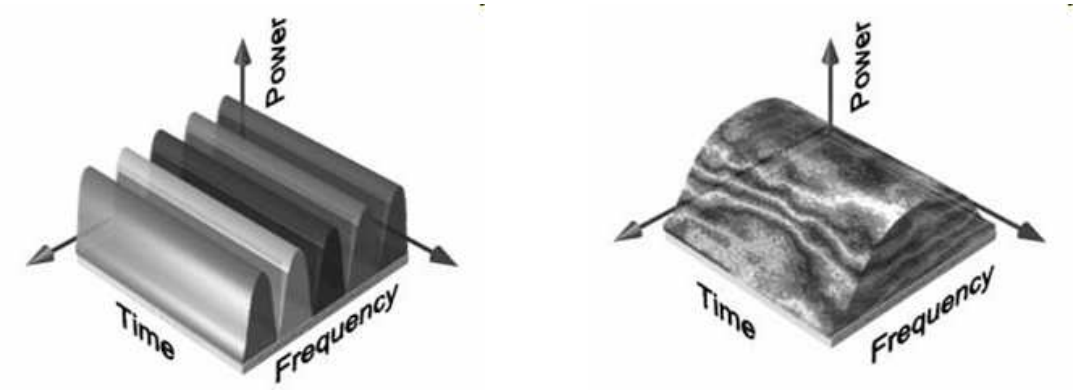
Na literatura, essa técnica, com seus vários tipos de implementação possíveis, tem sido chamada por vários nomes como *orthogonally multiplexed QAM* (O-QAM)[4], *parallel quadrature AM*[5] entre vários outros. Porém as duas que mais se destacaram foram a OFDM para uso sem-fio (*wireless*) e DMT (*Discrete Multi Tone*) para sistemas DSL (*Digital Subscriber Line*). Entretanto, como todas as técnicas basicamente utilizam o mesmo princípio, iremos nos referir a elas por um nome genérico[1]: MCM (*Multi-Carrier Modulation*), que em inglês significa modulação por multi-portadora.

Dessa forma MCM significará a técnica no sentido geral, e OFDM significará a implementação da técnica MCM utilizando a FFT.

[1] Como utilizado em [6]

3.1 Single e multi-carrier modulation

A técnica MCM, como explicado acima, divide o espectro disponível em vários canais ortogonais (Figura 1(a)). Em contrapartida pode-se utilizar todo o canal disponível para transmitir os dados, sem dividí-los em vários canais (Figura 1(b)). Essa segunda implementação é chamada de SCM (*Single-Carrier Modulation*), ou modulação por portadora única.



(a) *Espectro MCM.* **(b)** *Espectro SCM.*

Figura 1: *Espectro SCM e MCM.*

3.2 *Single-Carrier Modulation*

A técnica SCM, em que é utilizada apenas uma única portadora utilizando toda a banda disponível (Figura 1(b)) para modular o sinal a ser transmitido, tem sido usada por muitas décadas.

Nos moduladores SCM, os dados são transmitidos serialmente, ou seja, os símbolos gerados são transmitidos seqüencialmente, com o espectro utilizado por cada símbolo ocupando toda a banda disponível.

Entre as técnicas mais usadas, podemos destacar[7]:

ASK (*Amplitude-Shift Keying*)

Nesse processo de modulação digital, a amplitude da portadora senoidal é alterada de acordo com o sinal digital $x(t)$ a ser transmitido. (Figura 2(a))

FSK (*Frequency-Shift Keying*)

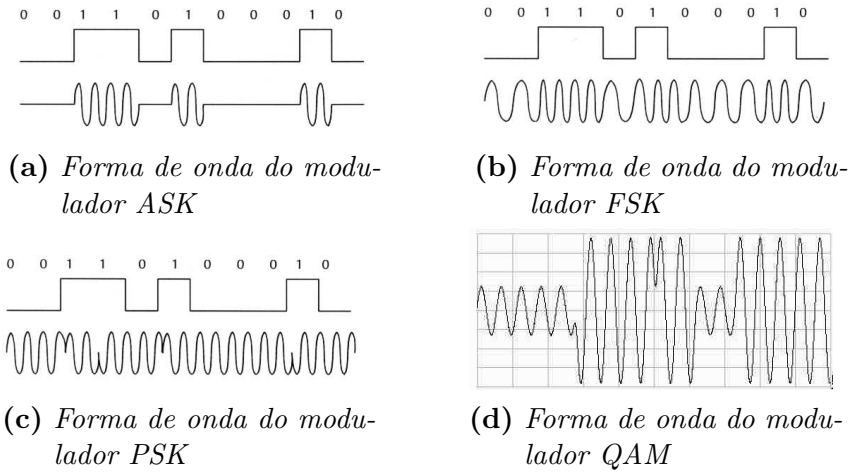
Nesse processo de modulação digital, o sinal digital a ser transmitido atua sobre a freqüência da portadora senoidal. (Figura 2(b))

PSK (*Phase-Shift Keying*)

Nesse processo de modulação digital, o sinal digital a ser transmitido atua sobre a fase da portadora senoidal. (Figura 2(c))

QAM (*Quadrature Amplitude Modulation*)

Nesse processo de modulação digital, o sinal digital a ser transmitido atua tanto sobre a fase quanto sobre a amplitude da portadora senoidal (Figura 2(d)).



(a) *Forma de onda do modulador ASK*

(b) *Forma de onda do modulador FSK*

(c) *Forma de onda do modulador PSK*

(d) *Forma de onda do modulador QAM*

Figura 2: *Modulações ASK, FSK, PSK e QAM*

3.2.1 Modulação QAM

Dentre essas formas de modulação digital a mais interessante e utilizada é a QAM. Nesse sistema (como dito), o sinal digital a ser transmitido atua tanto sobre a fase quanto sobre a amplitude da portadora senoidal. Portanto, o QAM é um misto da modulação ASK com a PSK (Figura 3).

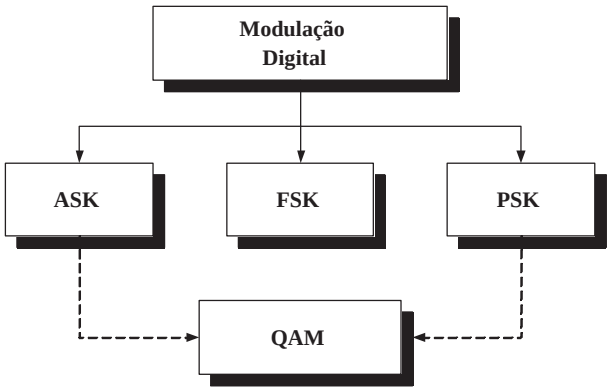


Figura 3: *Constituição do QAM*

3.2.1.1 O modulador

Para que isso seja possível, duas portadoras de mesma freqüência e com a fase defasadas em 90 graus (seno e cosseno) são modulados em amplitude por um sinal multinível (Figura 4), que através de um conversor binário multinível gera os níveis de tensão a e b de acordo com o número binário de entrada. Com isso eles terão um alfabeto finito, ou seja, só existirão para alguns níveis discretos.

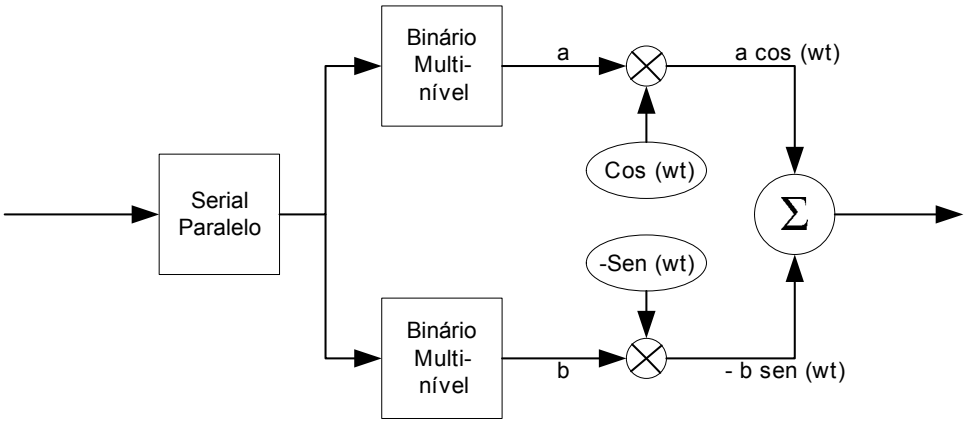


Figura 4: *Esquemático do modulador QAM*

Por causa da ortogonalidade das portadoras é possível utilizar a mesma banda de freqüência, onde cada portadora pode ser modulada independentemente e transmitida através da mesma freqüência.

Por exemplo, em um modulador QAM de dois bits, um bit é enviado para o conversor em fase, e o outro bit é enviado para o conversor em quadratura. Já para um modulador de quatro bits, dois bits são enviados para o conversor em fase e os outros dois são enviados para o conversor em quadratura.

3.2.1.2 Diagrama de constelação

O resultado da soma desses sinais é um sinal modulado tanto em amplitude quanto em fase (Figura 2(d)):

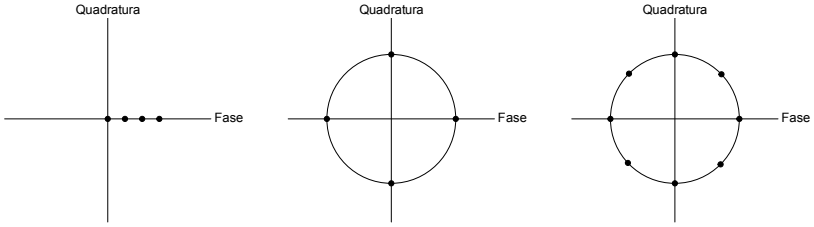
$$\begin{aligned} S(t) &= a\cos(\omega t) - b\sin(\omega t) \\ &= \sqrt{a^2+b^2}\cos\left(\omega t - \arctan\left(\frac{b}{a}\right)\right) \end{aligned} \tag{3.1}$$

É possível utilizar uma representação fasorial desse sinal. A partir da equação (3.1),

representa-se o fasor com um determinado valor de amplitude e de fase:

$$\begin{aligned} S(t) &= \sqrt{a^2 + b^2} \angle - \arctan\left(\frac{b}{a}\right) \\ &= a + jb \end{aligned} \tag{3.2}$$

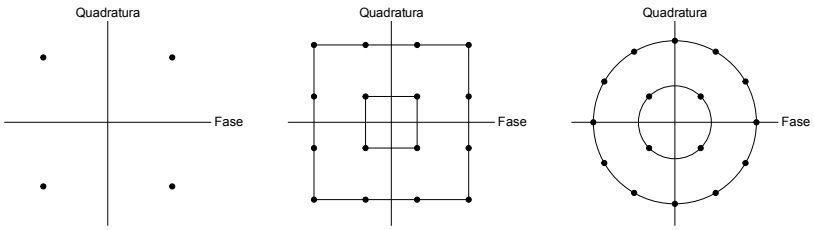
Como a e b só existem para alguns níveis discretos, então podemos indicar num diagrama todas as possíveis combinações de a e b, chamado de Diagrama de constelação. A Figura 5 mostra o diagrama de constelação para as modulações ASK e PSK.



(a) *4-ASK* **(b)** *4-PSK* **(c)** *8-PSK*

Figura 5: *Diagramas de constelação ASK e PSK*

O diagrama de constelação de um modulador QAM é montado de acordo com os níveis discretos produzidos pelo conversor. Assim, em um modulador QAM de dois bits, temos apenas quatro pontos na constelação, por isso esse modulador é chamado de 4 QAM (Figura 6(a)). Da mesma forma no modulador QAM de 4 bits será um 16 QAM e assim por diante. Observe que existem várias formas de se implementar a constelação QAM (Figura 6(b) e Figura 6(c)). O formato da constelação é determinado pelo conversor de tensão.



(a) *4-QAM* **(b)** *16-QAM* **(c)** *16-QAM*

Figura 6: *Diagramas de constelação QAM*

3.2.1.3 Espectro QAM

A Figura 7 mostra o espectro de um sinal QAM não filtrado. Ele possui o formato da função $sinc(x)$, com cruzamentos em zero nos múltiplos de $1/T_s$, onde T_s é o período do

símbolo QAM.

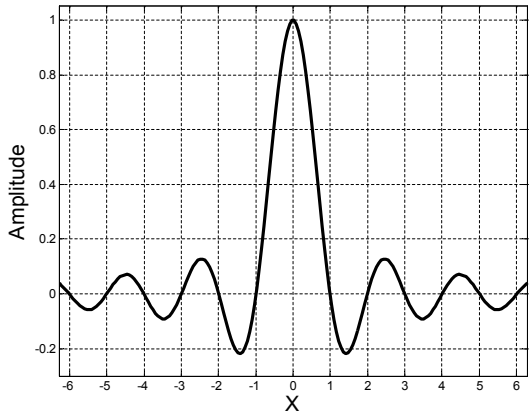


Figura 7: *Espectro de um sinal QAM — sinc(x)*

3.2.2 Desvantagens do sistema SCM

Um dos principais problemas para os sistemas SCM é quando a resposta em freqüência do canal não é linear (Figura 8), seja devido a perdas ou a ruídos. Para contornar esse problema é necessário utilizar eqüalizadores, e dependendo da resposta do canal esses eqüalizadores podem se tornar extremamente complexos. Outro problema é quando há multipercurso no canal. Nesse caso é necessário utilizar eqüalizadores adaptativos, e devido a isso transmitir uma seqüencia de treinamento para a convergência do eqüalizador e sincronização do sistema.

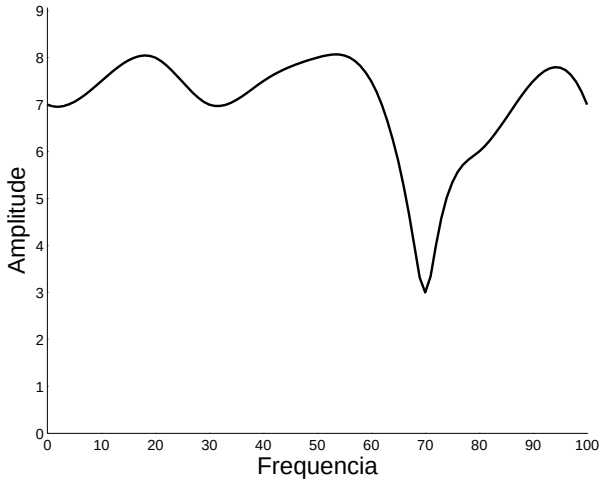


Figura 8: *Resposta em freqüência de um canal não linear*

Perceba que para sistemas de banda larga e alta taxa de transmissão os eqüalizadores,

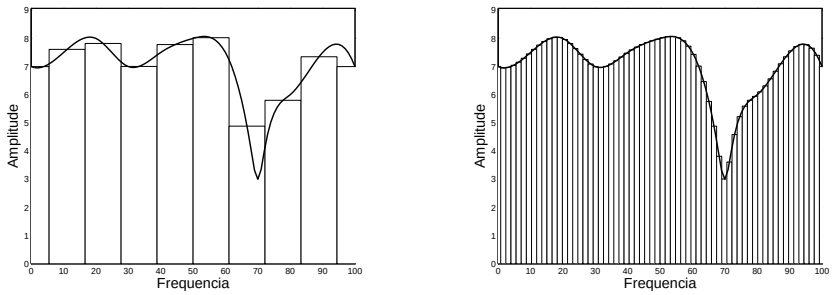
cada vez mais complexos e difíceis de implementar, se tornam um ponto de grande importância no desenvolvimento do sistema, pois dependerá dele o desempenho do sistema.

3.3 *Multi-Carrier Modulation*

O princípio de transmitir dados dividindo-os em vários canais, ou seja modulando várias portadoras, foi usado há mais de 45 anos (em 1957) no sistema Kineplex[8], porém, devido as complexidades de implementar tal sistema manteve-se essa técnica pouco explorada. Agora, entretanto, o interesse tem aumentado devido ao avanço da tecnologia, que permite a implementação do mesmo princípio utilizado nos primórdios do MCM para a transmissão de dados, porém utilizando técnicas mais eficientes.

Como dito, a técnica MCM divide o espectro disponível em vários subcanais ortogonais (Figura 1(a)) e transmite os dados paralelamente modulados em subportadoras[2].

O fato de dividir o espectro consegue resolver alguns problemas apresentados no SCM, como a necessidade de complexos eqüalizadores, pois dividindo-se o canal disponível em vários subcanais (Figura 9), a resposta em cada subcanal será quase linear, podendo ser utilizado eqüalizadores extremamente simples. Quanto ao problema de multipercurso, como veremos, é contornado com a utilização de um intervalo de guarda, que é extremamente simples de se implementar.



(a) *Canal com 10 subcanais* **(b)** *Canal com 64 subcanais*

Figura 9: *Divisão de um canal não linear em vários subcanais*

Outras razões para o interesse no MCM se deve ao fato de, entre suas propriedades, o sinal MCM poder ser processado no receptor sem melhorias do ruído/interferência e o longo tempo de símbolo gerado pelo MCM produz uma grande imunidade ao ruído impulsivo e rápidas mudanças no sistema.

[2]Será tratado como subportadora, a portadora em si, sintonizada em uma dada freqüência, e subcanal o espectro utilizado por essa subportadora.

Devido a essas características, o MCM tem se mostrado extremamente interessante para sistemas de banda larga e alta taxa de transmissão, pois a complexidade do sistema não cresce exageradamente.

3.3.1 O principio básico do MCM

Um esquema muito simplificado de um sistema MCM é mostrado na Figura 10.

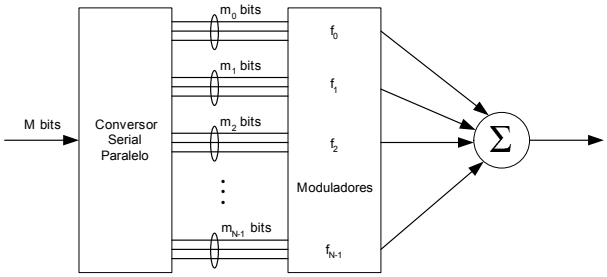


Figura 10: *Um transmissor MCM simplificado.*

Os dados na entrada do transmissor MCM são agrupados em blocos de M bits. Esses M bits são dividido em N subblocos de m_n bits, onde $M = \sum_{n=0}^{N-1} m_n$, que são usados para modular N subportadoras, cada uma com banda igual e com freqüência central $f_n = n\Delta f$, $n = 0, \ldots, N-1$, espaçadas de Δf uma da outra. Esse espaçamento Δf das subportadoras é cuidadosamente selecionado de tal forma que o espectro de uma não interfira no espectro das outras. As subportadoras moduladas são então somadas e transmitidas, gerando uma portadora MCM.

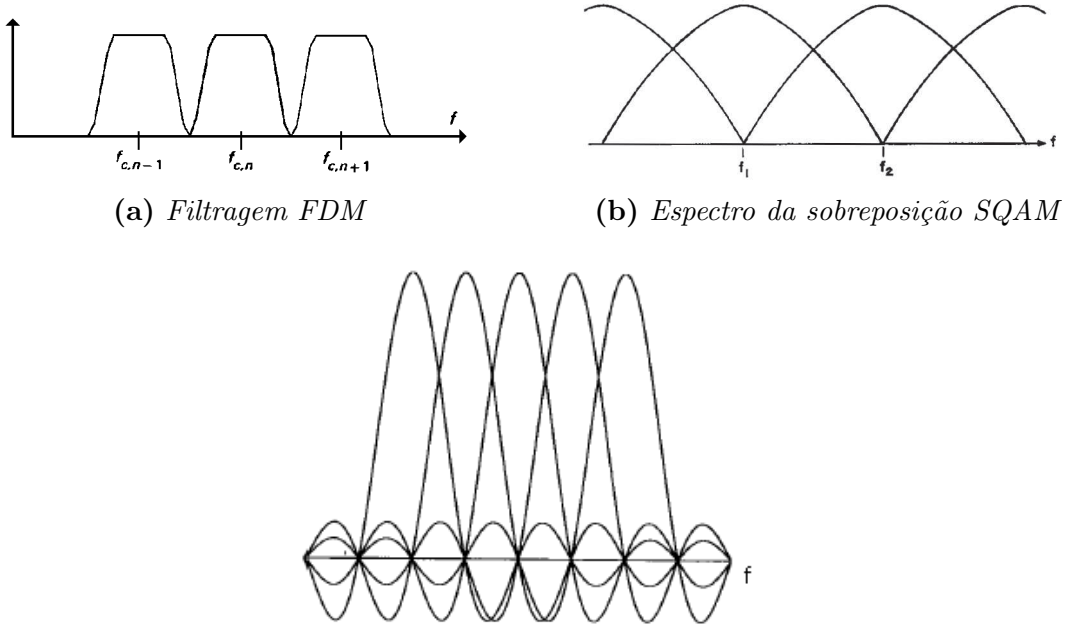
Em cada um desses N subcanais são modulados m_n bits utilizando, geralmente, um modulador QAM, porém pode-se utilizar qualquer uma das modulação digitais mostradas anteriormente. Portanto, a grosso modo, o que o sistema MCM faz é utilizar vários subcanais SCM para transmitir os dados paralelamente.

No sistema MCM, em oposição ao FDM convencional, o número de bits, m_n, de entrada que é alocado para modular as diferentes subportadoras pode ser diferente e ajustado com o tempo. A alocação dos bits para cada subcanal é coordenada pelo MCM para maximizar o desempenho. Dessa forma os subcanais que terão menos atenuação e/ou menos ruído poderão transmitir mais bits de informação.

No receptor é feito o processo reverso. O sinal recebido é separado em N subcanais, cada um centrado em f_n, em que é feita a demodulação dos m_n bits, e por fim, eles são re-arranjados para formar os M bits transmitidos.

3.3.2 Implementações do MCM

Ao longo do tempo foram propostas várias formas de se implementar o sistema MCM. Os pontos principais de distinção entre os sistemas está em como foi implementado a modulação e como foi feita a separação dos subcanais, de tal forma que elas se mantenham ortogonais. Podemos destacar três métodos[9]: filtragem FDM, modulador SQAM e sobreposição do espectro QAM.



(a) *Filtragem FDM*

(b) *Espectro da sobreposição SQAM*

(c) *Espectro da sobreposição de QAM*

Figura 11: *Espectro dos vários sistemas MCM*

3.3.2.1 Filtragem FDM

Inicialmente, como no sistema Kineplex, os modems MCM usavam as técnicas convencionais de FDM (*Frequency Division Multiplexing*), usando filtros para separar completamente os subcanais. O espectro de potência transmitido para somente três subcanais de um sistema MCM desse tipo é mostrado na Figura 11(a).

Um sistema básico desse tipo de modulação é mostrado na Figura 12. Esse sistema utiliza vários moduladores QAM, cada um com um freqüência central diferente, necessitando, assim, ser gerado e mantido vários sinais senoidais diferentes. No receptor é necessário recuperar a portadora para cada demodulador QAM. Além disso, implementa-se vários filtros tanto no transmissor quanto no receptor, cada um com banda passante

diferente tornando a implementação desse sistema extremamente complexa, principalmente quando o número de subcanais aumenta. Outro complicador é a dificuldade de se implementar vários filtros tendo exatamente a largura de Nyquist, o que torna a eficiência de utilização da banda muito baixa, como visto na Figura 11(a).

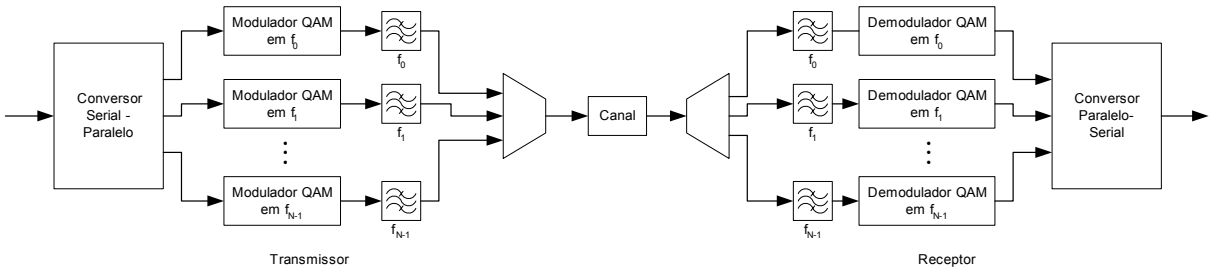


Figura 12: *OFDM utilizando filtragem FDM*

3.3.2.2 Modulador SQAM

Chang[10] mostrou as condições gerais para que se possa sobrepor os subcanais adjacentes, como visto na Figura 11(b), de forma que a eficiência de utilização da banda é aumentada para perto de 100%. Saltzberg[5] utilizou essas condições para implementar um sistema MCM que foi finalmente implementado por Hirosaki[4], como mostrado em Figura 13.

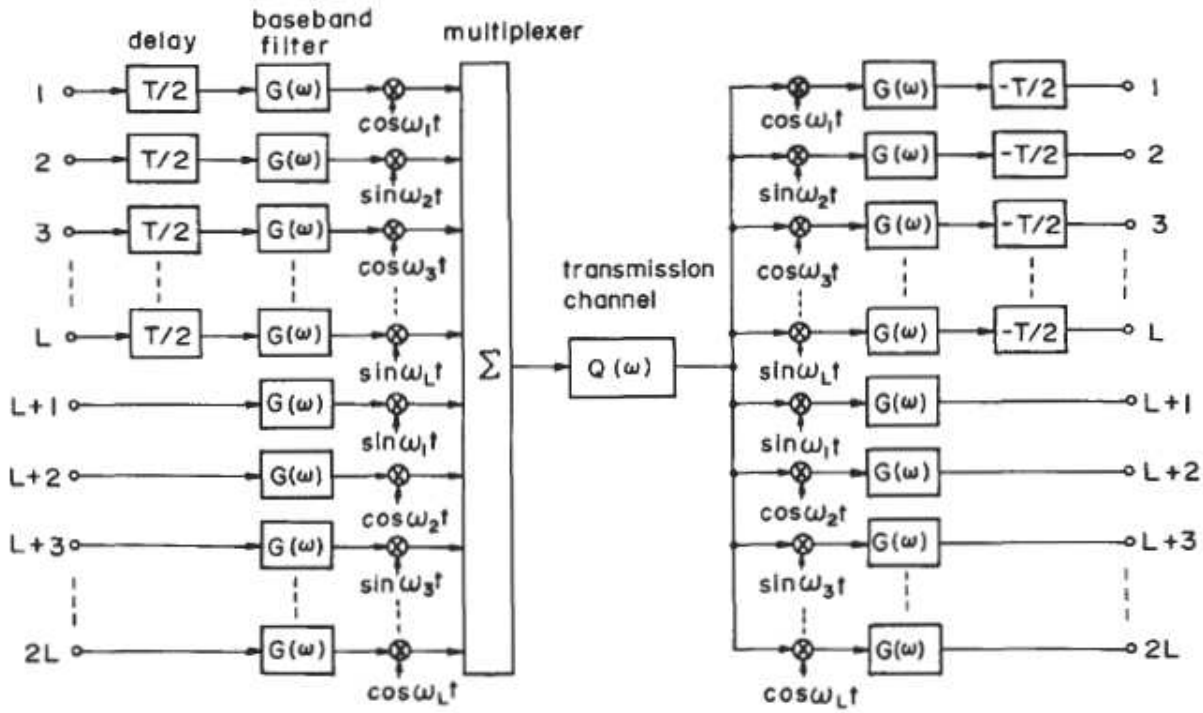


Figura 13: *OFDM utilizando modulação SQAM*

A diferença entre esse método e o anterior é a troca do banco de moduladores QAM por um banco de moduladores SQAM (*Staggered QAM*) ou, também chamado, OQAM (Offset QAM)[11]. Essa técnica consiste em atrasar o sinal em quadratura do QAM de meio período de símbolo em relação ao sinal em fase.

Apesar de Hirosaki ter implementado o banco de moduladores SQAM utilizando a DFT, ainda se faz necessário a utilização de bancos de filtros para se garantir a ortogonalidade. Porém, como dito, eles são sobrepostos.

3.3.2.3 Sobreposição do espectro QAM

Consegue-se uma grande melhoria tanto em eficiência de utilização da banda quanto em complexidade de implementação utilizando o formato do espectro QAM (Figura 7). Nesse sistema[12] os espectros individuais não têm mais a banda limitada. Embora haja sobreposição espectral entre as subportadoras, elas não interferem umas com as outras. Isso é conseguido fazendo com que a freqüência central de uma subportadora fique no cruzamento por zero de todas as outras subportadoras (Figura 11(c)). Então se o sinal for amostrado na freqüência da subportadora, não teremos interferências das outras subportadoras, pois elas serão zero. Em outras palavras, mantém-se a ortogonalidade espectral.

A divisão de freqüência é conseguida, não mais por filtragem de banda, mas no processamento, ou seja, não será mais necessário a implementação de um grande e complexo banco de filtros. Porém, para se manter a ortogonalidade é necessário se introduzir um período de guarda[12].

A grande vantagem desse método é que ambos, transmissor e receptor, podem ser **totalmente** implementados de forma digital utilizando a implementação rápida da DFT (*Discrete Fourier Transform*), a FFT(*Fast Fourier Transform*), como explicado a seguir.

3.4 Modulação via IDFT

Como dito, um sistema OFDM é uma forma de se implementar a técnica MCM vista na seção 3.3.2.3 utilizando a IDFT para fazer a modulação e a DFT para fazer a demodulação, ambas de forma **totalmente** digital. Como se pode fazer isso é explicado abaixo.

Como o sinal MCM consiste de N subportadoras QAM em paralelo, e como um

sinal QAM[3] consiste em duas portadoras em quadratura, $a\cos(\omega t) - b\sin(\omega t)$, que são modulados de acordo com a seqüencia de bits, teremos o seguinte sinal na saída do modulador MCM:

$$S(t_n) = \sum_{k=1}^{N} \left[a_k \cos(\omega_k t_n) - b_k \sin(\omega_k t_n)\right] \tag{3.3}$$

onde a_k e b_k são, respectivamente, os termos em fase e em quadratura do sinal QAM do canal k e ω_k é a freqüência da subportadora do canal k.

Discretizando, para que a modulação possa ser feita digitalmente, temos:

$$\begin{aligned} w_k &= \frac{2\pi k}{N\Delta t} \\ t_n &= n\Delta t \end{aligned} \tag{3.4}$$

onde Δt é o período em que cada símbolo MCM é gerado.

Aplicando (3.4) em (3.3) temos:

$$S_n = \sum_{k=1}^{N} \left[a_k \cos\left(2\pi\frac{nk}{N}\right) - b_k \sin\left(2\pi\frac{nk}{N}\right)\right] \tag{3.5}$$

Aplicando a propriedade da multiplicação de números complexos[4] na equação (3.5) teremos:

$$\begin{aligned} \left(a_k \cos\left(2\pi\frac{nk}{N}\right) - b_k \sin\left(2\pi\frac{nk}{N}\right)\right) &= \Re(Z) \\ &= (x_1x_2 - y_1y_2) \end{aligned}$$

Fazendo:

$$\begin{aligned} x_1 &= a_k \\ x_2 &= \cos\left(2\pi\frac{nk}{N}\right) \\ y_1 &= b_k \\ y_2 &= \sin\left(2\pi\frac{nk}{N}\right) \end{aligned}$$

[3]Ver seção 3.2.1, na pagina 7.

[4]Propriedade da multiplicação de números complexos:

$$Z = z_1z_2 = (x_1 + jy_1)(x_2 + jy_2) = (x_1x_2 - y_1y_2) + j(x_1y_2 + x_2y_1)$$

onde a parte real de Z é $\Re(Z) = (x_1x_2 - y_1y_2)$ e a parte imaginária de Z é $\Im(Z) = (x_1y_2 + x_2y_1)$

Teremos:

$$\begin{aligned} z_1 &= a_k + jb_k \\ z_1 &= d_k \\ z_2 &= \cos\left(2\pi\frac{nk}{N}\right) + j\sin\left(2\pi\frac{nk}{N}\right) \\ z_2 &= e^{+j2\pi\frac{nk}{N}} \end{aligned}$$

Então:

$$\begin{aligned} \left(a_k\cos\left(2\pi\frac{nk}{N}\right) - b_k\sin\left(2\pi\frac{nk}{N}\right)\right) &= \Re\left(Z = z_1 z_2\right) \\ &= \Re\left\{d_k e^{+j2\pi\frac{nk}{N}}\right\} \end{aligned} \tag{3.6}$$

Aplicando (3.6) em (3.5) teremos:

$$S_n = \sum_{n=1}^{N}\left[a_k\cos\left(2\pi\frac{nk}{N}\right) - b_k\sin\left(2\pi\frac{nk}{N}\right)\right] = \sum_{n=1}^{N}\Re\left\{d_k e^{+j2\pi\frac{nk}{N}}\right\}$$

Fazendo $F_k = d_k e^{+j2\pi\frac{nk}{N}}$, o seu conjugado[5] $F_k^* = d_k^* e^{-j2\pi\frac{nk}{N}}$ e aplicando a seguinte propriedade $F_k + F_k^* = 2\Re\left(F_k\right)$, temos

$$\begin{aligned} S_n &= \sum_{k=1}^{N}\Re\left\{F_k\right\} \\ 2S_n &= \sum_{k=1}^{N}2\Re\left\{F_k\right\} \\ &= \sum_{k=1}^{N}\left(F_k + F_k^*\right) \end{aligned}$$

Portanto,

$$\begin{aligned} S_n &= \frac{1}{2}\left[\sum_{k=1}^{N}F_k + \sum_{n=1}^{N}F_k^*\right] \\ &= \frac{1}{2}\left[\sum_{k=1}^{N}d_k e^{+j2\pi\frac{nk}{N}} + \sum_{k=1}^{N}d_k^* e^{-j2\pi\frac{nk}{N}}\right] \end{aligned} \tag{3.7}$$

[5]Propriedade do conjugado da multiplicação de números complexos: $\left(z_1 z_2\right)^* = z_1^* z_2^*$

Aplicando a simetria hermetiana[6] e a propriedade de simetria da DFT[7] podemos fazer:

$$\begin{aligned} X_0 &= X_{\overline{N}} = 0 \\ X_k &= d_k \\ X_{2\overline{N}-k} &= d_k^* \end{aligned} \tag{3.8}$$

para $k = 1, 2, \ldots, \overline{N} - 1$, onde $\overline{N} = N + 1$.

Então:

$$S_n = \frac{1}{2}\left[\sum_{k=0}^{\overline{N}-1} X_k e^{+j2\pi\frac{nk}{N}} + \sum_{k=\overline{N}}^{2\overline{N}-1} X_k e^{+j2\pi\frac{nk}{N}}\right]$$

$$S_n = \frac{1}{2}\left[\sum_{k=0}^{2\overline{N}-1} X_k e^{+j2\pi\frac{nk}{N}}\right]$$

Que é proporcional a IDFT de X_k.

Observe em (3.8), que o subcanal zero, X_0, será zero, $0+j0$, pois não se deve ter tensão DC no sinal. Outra coisa a ser notada é que para ser transmitido N canais, precisa-se de fazer uma IDFT de $2\overline{N}$ pontos, como $\overline{N} = N + 1$ temos $2(N + 1) = 2N + 2$ pontos, pois é necessário satisfazer a simetria hermetiana.

Com isso, percebe-se que a somatória de vários sinais QAM possui a mesma resposta que a IDFT de X_n definida em (3.8). Portanto, pode-se usar a implementação rápida da IDFT, a IFFT, para fazer a modulação, com a vantagem de poder implementá-la toda digitalmente, tendo simplesmente de fazer a conversão digital-analógico na saída do sistema digital.

3.5 Manutenção da ortogonalidade

Para que o sistema MCM utilizando sobreposição de espectro QAM (vista na seção 3.3.2.3) funcione é necessário que se garanta a ortogonalidade entre os vários subcanais, pois os mesmos não possuem a banda limitada.

Ao se transmitir um símbolo MCM imediatamente seguido de outro a ortogonalidade não será mais mantida, pois, devido a resposta do canal (Figura 14), o final de um símbolo interferirá em ν amostras do inicio de outro.

[6]Simetria Hermetiana: $X^*(\omega) = X(-\omega)$

[7]Propriedade de simetria da DFT: $X^*(k) = X(N - k)$

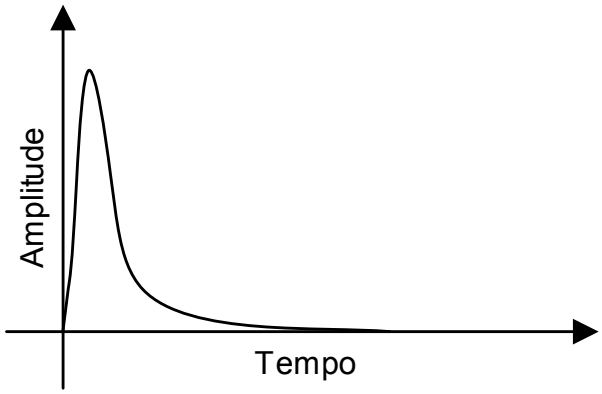


Figura 14: *Resposta impulsiva do canal*

Quando isso acontece um subcanal começa a interferir em outro, gerando a interferência inter-canal, ICI (*Inter-Channel Interference*). Outro problema que ocorre é quando há distorções lineares no canal, como atrasos devido a multipercursos e reflexões, que geram a interferência inter simbólica, ISI (*Inter-Symbol Interference*). Tanto a ICI quanto a ISI causam a perda da ortogonalidade do sistema.

Para resolver esse problema basta introduzir um período de guarda de ν amostras entre um símbolo e outro, de tal forma que o final de um símbolo não interfira no próximo. Basicamente há três formas de se fazer esse período de guarda[6]:

1. Adicionar as últimas ν amostras no inicio do sinal, fazendo assim um prefixo cíclico (*cyclic prefix*).
2. Adicionar as primeiras ν amostras no final do sinal, fazendo assim um sufixo cíclico.
3. Não transmitir nada no intervalo de guarda, e no receptor adicionar as ultimas ν amostras ao inicio.

Entre essas técnicas a mais utilizada é a de cyclic prefix, mostrada na Figura 15

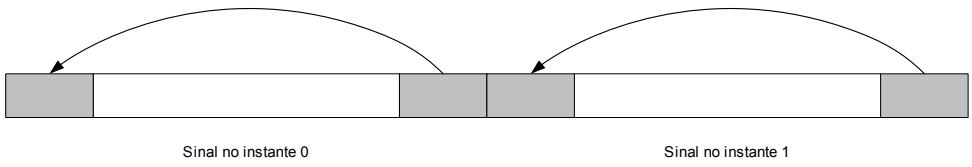


Figura 15: *Cyclic prefix*

A eficiência na utilização do canal será degradada, pois está sendo introduzido um período de ν amostras em que não se transmite nenhuma informação[8]. Pode-se transfor-

[8]Apesar de estar transmitindo dados, eles não contem nenhuma informação.

mar a resposta impulsiva do canal, e com isso diminuir o tamanho do período de guarda, com a inclusão de eqüalizadores. Portanto deve-se ter um compromisso entre a eficiência e a complexidade do eqüalizador na hora de se especificar o tamanho do período de guarda.

A melhoria que o período de guarda introduz é muito compensador, sendo praticamente impossível utilizar o sistema MCM sem ele. Além disso, tem-se a vantagem de facilmente resolver problemas como o multipercurso, fazendo assim a técnica MCM uma forte candidata a ser uma das principais modulação para *wireless*, principalmente as de tecnologia móvel, como os celulares (4G), o radiodifusão de TV digital (DVB) e rádio digital (DAB), redes de computadores sem fio (Wi-Fi) entre outros.

Outra boa aplicação para o MCM é na transmissão de dados pela rede elétrica. A rede elétrica de uma casa possui muitos ramais e bifurcações da linha de transmissão tornando o multipercurso um grande problema, contornável devido o intervalo de guarda utilizado pelo MCM.

3.6 A técnica OFDM

O OFDM é a implementação do MCM utilizando a IDFT para fazer a modulação e com o cyclic prefix para garantir a ortogonalidade. Ela foi primeiramente sugerida em [12] e implementada por Peled e Ruiz em 1980[13].

3.6.1 O transmissor OFDM

A função do transmissor OFDM é receber um bloco de M bits e gerar o sinal OFDM. Para isso, o transmissor OFDM (Figura 16) consiste basicamente de:

- Um conversor serial-paralelo
- Um Buffer
- Um codificador
- Um bloco que implementa a simetria hermetiana
- A IFFT
- Um adicionador de cyclic-prefix
- Um conversor paralelo-serial

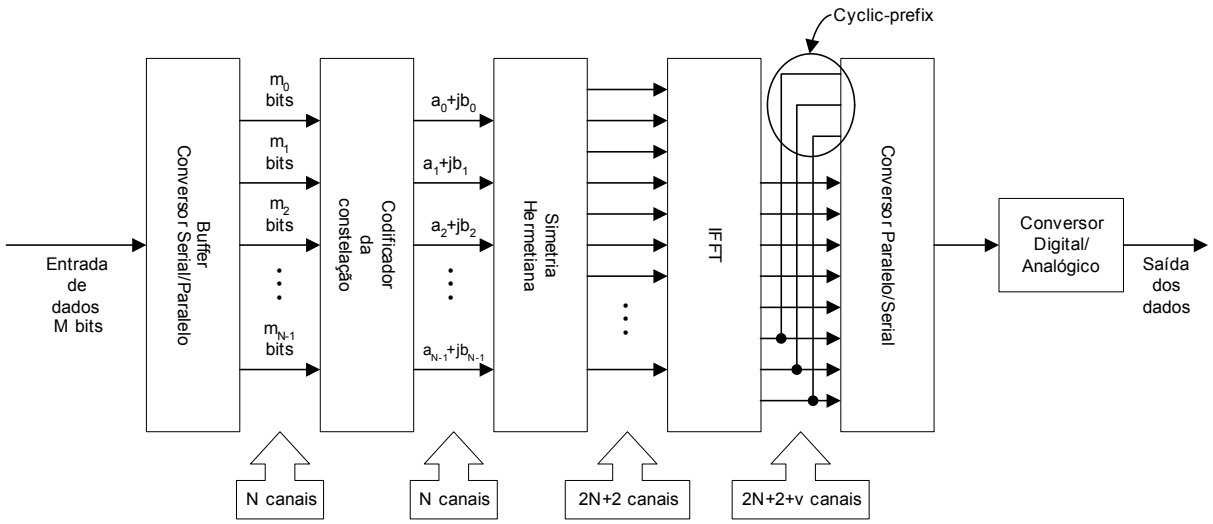


Figura 16: *Modulador básico OFDM*

- Um conversor digital-analógico.

O conversor serial-paralelo recebe os M bits seriais a serem transmitidos e os divide em N subblocos de m_n bits cada, que são então mapeados pelo codificador em um ponto $a_n + jb_n$ na constelação do modulador. Importante notar que nesse mapeamento é feito apenas uma conversão de bits para o fasor que ele representa, porém não é feita nenhuma modulação, como no caso do QAM, pois essa como vimos, é feita na IFFT. Após mapeados os N canais, é aplicado então a simetria hermetiana, de acordo com (3.8), gerando $2N+2$ canais complexos em simetria.

A IFFT recebe os $2N+2$ canais complexos gerados devido a simetria hermetiana e faz então a modulação propriamente dita. Ao final do processamento da IFFT, a saída só terá a parte real dos números. O conversor paralelo-serial serializa novamente os dados, porém ele serializa duas vezes as mesmas ν saídas da IFFT, fazendo o cyclic-prefix, e gerando $2N+2+\nu$ palavras para serem convertidos para analógico pelo conversor digital-analógico.

Perceba que a inclusão do cyclic-prefix é feita de forma extremamente simples pelo conversor paralelo-serial.

3.6.2 O receptor OFDM

Básicamente, o receptor desfaz tudo que foi feito no transmissor, portanto ele seria o "inverso" do transmissor (Figura 17), contendo:

- Um conversor analógico-digital.

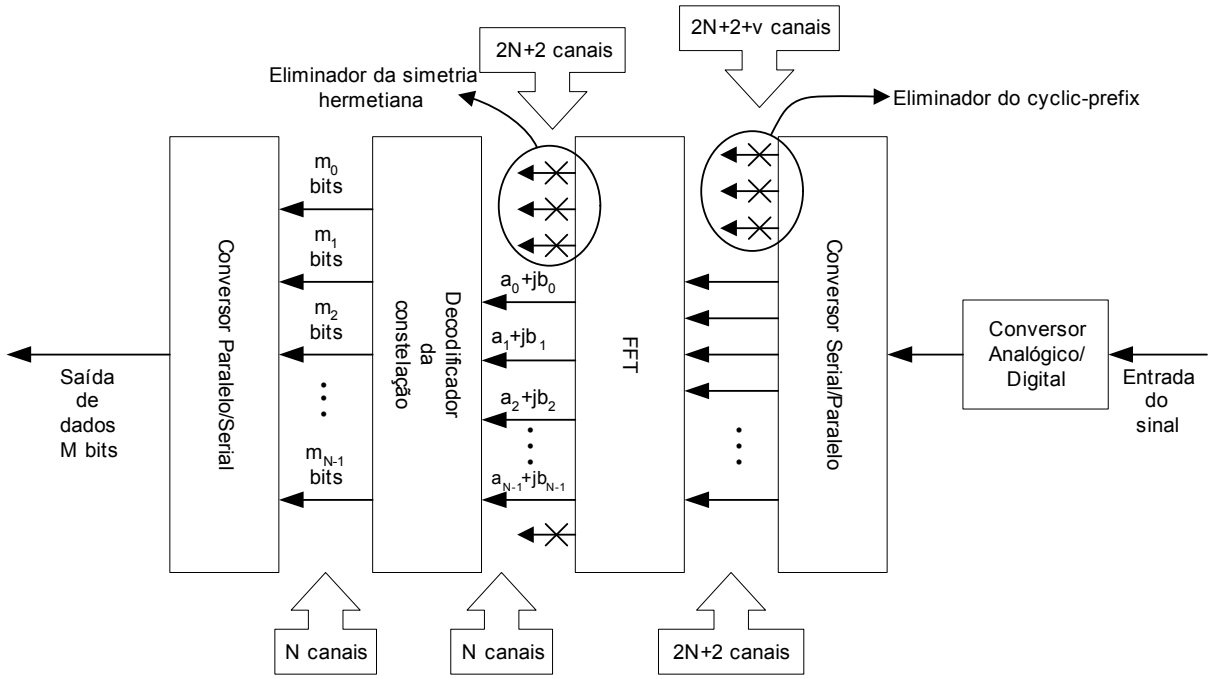


Figura 17: *Demodulador básico OFDM*

- Um conversor serial-paralelo
- Um eliminador do cyclic-prefix
- A FFT
- Um eliminador da simetria hermetiana
- Um decodificador
- Um Buffer
- Um conversor paralelo-serial

Nesse sistema o conversor analógico-digital amostra o sinal do canal, gerando $2N+2+\nu$ amostras seriais. Então o conversor serial-paralelo gera $2N + 2 + \nu$ canais paralelos.

A FFT recebe apenas $2N + 2$ canais reais, eliminando assim, de forma simples, os ν canais gerados devidos ao cyclic-prefix. A FFT demodula o sinal gerando também $2N + 2$ saídas complexas.

Ao fim do processamento da FFT, são enviados ao decodificador somente N canais complexos, eliminando, da mesma forma que se eliminou o cyclic-prefix, os canais gerados devido a simetria hermetiana. O decodificador faz a decisão sobre qual seqüencia binária o sinal recebido representa e então gera-se novamente o bloco de M bits transmitidos.

3.6.3 O modem OFDM

Integrando o transmissor e o receptor tem-se o modem OFDM. Nesse sistema, como foi visto, a modulação é feita de forma totalmente digital, e só se converte para analógico na hora de se transmitir o sinal. No receptor, a primeira coisa a se fazer é converter o sinal novamente para digital. A partir daí, todo o processo de demodulação é feito de forma digital.

Com isso, pode-se implementar todo o modem dentro de um chip, precisando apenas dos conversores analógico-digital e digital-analógico.

O projeto do modem tem de levar em conta a resposta do canal, para que se possa especificar o tamanho do cyclic-prefix.

4 Implementação do OFDM

Como visto na seção 3.6, o modem OFDM pode ser totalmente implementado de forma digital. Como os atuais FPGAs têm se mostrados bastantes interessantes para a implementação de sistemas digitais, de forma integrada em um único chip, será implementado o modem OFDM em um FPGA.

A escolha do FPGA levou em conta as características do sistema a ser implementado. Como será mostrado, há a necessidade de blocos de memória RAM de duas portas para a implementação da FFT, portanto foi escolhido os FPGAs da Xilinx, pois os mesmos contem as características exigidas. Mais especificamente foi utilizado a FPGA Spartan II XC2S200.

Será estudado agora, as características desse FPGA e a forma de se implementar em FPGA as várias partes do modem OFDM.

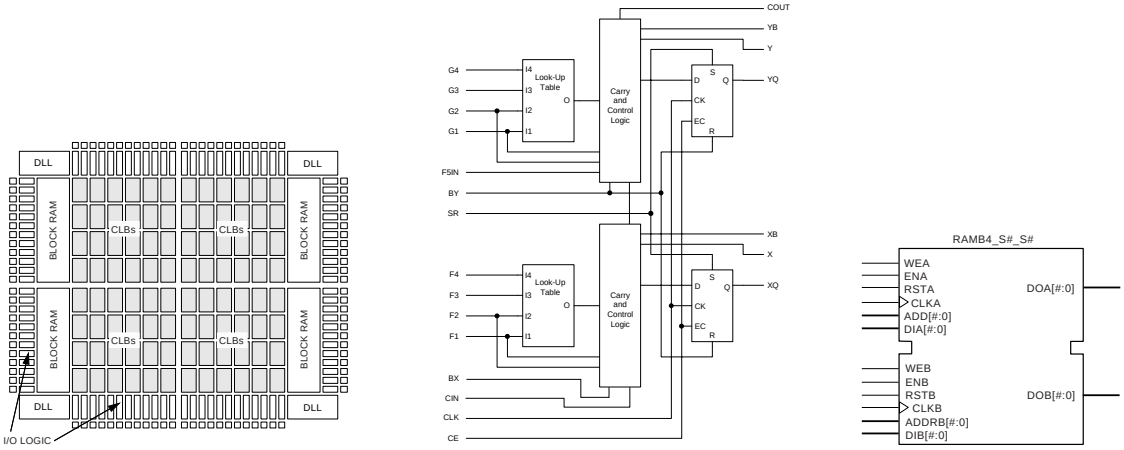
4.1 O Hardware — A FPGA Spartan II XC2S200

A família Spartan II [14] tem um arquitetura regular, flexível e programável de blocos lógicos configuráveis, CLB (*Configurable Logic Blocks*), como mostrado na Figura 18(a).

Os CLBs fornecem os elementos funcionais para a construção da maior parte da lógica. O bloco básico de um CLB é a célula lógica, LC (*Logic Cell*). Uma LC contém um gerador de funções, ou uma tabela de conversão, o LUT (*Look-Up Tables*), uma lógica de propagação rápida do carry (para operações matemáticas), e flip-flops para armazenar os dados. Cada CLB contém quatro LCs, organizados em dois *slices* similares. Um *slice* é mostrado na Figura 18(b)

Os CLBs são rodeado por blocos de Entrada e Saída programáveis, IOBs (*Input Output Blocks*). Os IOBs fornecem a interface entre os pinos do chip e a lógica interna.

Há quatro DLLs (*Delay-Locked Loops*), um em cada canto do chip, que compensam o



(a) *Diagrama de blocos básico* **(b)** *Um* slice *do CLB (São dois* slices *por CLB)* **(c)** *RAM de duas portas*

Figura 18: *A FPGA Spartan II*

atraso de propagação e mantém um baixo *clock skew* entre os sinais de clock distribuídos através do chip.

Existem duas colunas de blocos de RAM, em lados opostos do chip, ficando entre os CLBs e os IOBs. Os blocos de memória RAM (Figura 18(c)) aumentam as funcionalidades incorporadas ao Spartan II. Cada bloco de RAM é uma RAM de duas portas de 4096 bits totalmente síncrona tendo o tamanho do barramento de dados das duas portas variáveis, podendo ser configurados independentemente, e tendo o acesso independente.

Todos esses elementos são interconectados por uma poderosa e versátil hierarquia de canais de roteamento.

A FPGA Spartan II XC2S200 é a maior FPGA da família Spartan II. Ela contém 5.292 LCs que equivalem a até 200.000 portas lógicas, distribuídas em um arranjo de 28 x 42 CLBs, num total de 1.176 CLBs. Possui 284 portas para I/O e 14 blocos de RAM de 4096 bits cada, fornecendo um total de 56Kbits de RAM, além das quatro DLLs.

4.2 Serial x paralelo

Após conhecer as características da FPGA utilizada, será vista as formas de se implementar as várias partes do modem OFDM nela.

O conversor serial-paralelo do transmissor tem a função de receber os dados a serem transmitidos, montá-los em blocos de M bits, e então separá-los em N canais, com m_n bits cada. Lembre-se que a quantidade de bits em cada canal pode ser diferente .

Já o conversor serial-paralelo do receptor tem a função de receber os dados a serem demodulados, com a mesma estrutura do transmissor, apenas com a quantidade de canais a paralelizar diferentes.

Para armazenar os M bits é necessário um buffer. Pode ser utilizado o bloco de memória RAM, presentes na FPGA, para fazer esse buffer, armazenando cada entrada em uma posição de memória diferente.

Quando o conversor armazenar os primeiros M dados, o sistema terá de parar até que possa escrever novamente. Porém, como é desejável que o sistema funcione continuamente se faz necessário um buffer de tamanho $2M$. Então, quando o sistema completar a primeira metade do buffer, M dados, poderá continuar armazenando na segunda metade das $2M$ posições do buffer. Ao completar a segunda metade, começa a se escrever no início do buffer.

O conversor paralelo-serial é a implementação ao contrário do serial-paralelo. Ele lê o dado armazenado em um buffer de forma seqüencial, tornando o dado serial.

4.3 Mapeando a constelação

O codificador da constelação mapeia os m_n bits do canal em um ponto $a_n + jb_n$ na constelação do modulador. O decodificar recebe esse ponto e os remapeia como os m_n bits transmitidos.

4.3.1 Codificador da constelação

Como dito, é importante notar que nesse mapeamento é feito apenas uma conversão de bits para o fasor que ele representa, porém não é feita nenhuma modulação, como no caso do QAM, pois essa como mostrado, é feita pela IFFT.

É necessário especificar como será a constelação a ser mapeada, para se implementar esse bloco. Porém, independentemente do formato da constelação, o bloco codificador pode ser feito através de uma consulta a uma tabela de conversão, implementada pela LUT que existe nos LCs dos FPGAs.

Por exemplo, para um constelação 4-QAM[1], mostrada na Figura 6(a), de tal forma que a e b sejam números binários de 3 bits em complemento dois, e tenham o valor de

[1]Possui 2 bits de entrada

$+2_d$ ou 010_b para o bit 0 e -2_d ou 110_b para o bit 1 tem-se as possibilidades mostradas na Tabela 1.

Bits de entrada	Ponto da constelação $(a + jb)_b$
00	$010 + j010$
01	$010 + j110$
10	$110 + j010$
11	$110 + j110$

Tabela 1: *Tabela de conversão 4-QAM*

Atente que a entrada do codificador um número binário de m_n bits, e que a saída gera dois números binários, um em fase, a, e outro em quadratura, b, cujo tamanho é definido pela IFFT.

4.3.2 Decodificador da constelação

No receptor, o ponto da constelação transmitido (Figura 6(a)) poderá ter mudado (Figura 19) devido aos ruídos do canal de transmissão, erro no tempo de amostragem do receptor e várias outras causas.

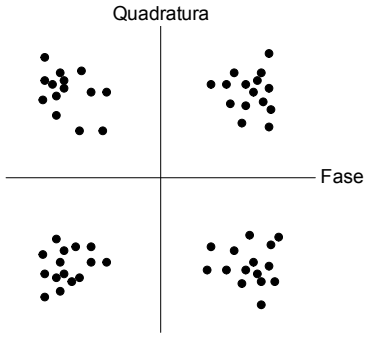


Figura 19: *Constelação no receptor*

Portanto é necessário definir um limiar para que possa ser feita a decisão sobre qual ponto na constelação o sinal recebido está representando. Essa é a função do decodificador.

Para o sistema exemplificado acima, observando a Tabela 1, o bit 0 é convertido para $\underline{0}10_2$ e o bit 1 para $\underline{1}10_2$. Nesse caso, o decodificador é implementado de forma simples, pega-se o bit mais significativo (que indica o sinal) para fazer a decodificação, gerando novamente um número binário de m_n bits.

Para sistemas em que o diagrama de constelação é maior, ou mesmo, o 4-PSK, mostrado em Figura 5(b), será necessário a implementação de métodos mais avançados,

como por exemplo, uma rede neural treinada para essa função.

4.4 A simetria hermetiana

Após mapeados os N canais é aplicado então a simetria hermetiana, de acordo com (3.8), para que a modulação possa ser feita por uma IFFT. Gera-se então $2N + 2$ canais em simetria.

No receptor, ao fim do processamento da FFT, são enviados ao decodificador somente N canais, eliminando-se os canais gerados devido a simetria hermetiana (Figura 17).

A simetria hermetiana é implementada de acordo com (3.8), reescrita aqui por conveniência:

$$\begin{aligned} X_0 = X_{\overline{N}} &= 0 \\ X_k &= d_k \\ X_{2\overline{N}-k} &= d_k^* \end{aligned}$$

para $k = 1, 2, \ldots, \overline{N} - 1$, onde $\overline{N} = N + 1$.

Por exemplo, para transmitir $N = 3$ canais (d_1, d_2, d_3), fazendo $\overline{N} = N + 1$, então $\overline{N} = 4$, $2\overline{N} = 8$ e $k = 1, 2, 3$. Aplicando na equação (3.8) obtém-se o resultado mostrado na Tabela 2.

k	X_k	X_{8-k}
1	$X_1 = d_1$	$X_7 = d_1^*$
2	$X_2 = d_2$	$X_6 = d_2^*$
3	$X_3 = d_3$	$X_5 = d_3^*$

Tabela 2: *Simetria hermetiana aplicada em 3 canais*

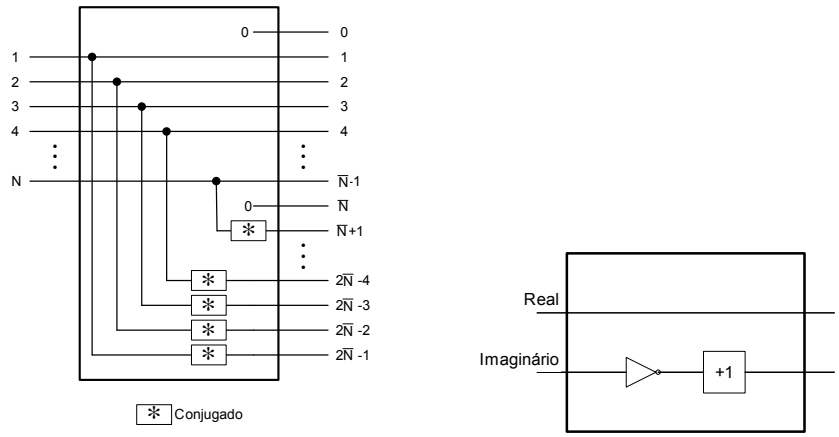
Sabendo que $X_0 = X_{\overline{N}=4} = 0 + j0$, obtém-se de acordo com a Tabela 2 o resultado

mostrado na Equação (4.1).

$$\begin{aligned} X_0 &= 0 \\ X_1 &= d_1 \\ X_2 &= d_2 \\ X_3 &= d_3 \\ X_4 &= 0 \\ X_5 &= d_3^* \\ X_6 &= d_2^* \\ X_7 &= d_1^* \end{aligned} \quad \text{Conjugado} \tag{4.1}$$

Dessa forma, pode-se manter X_0 e $X_{\overline{N}}$ sempre em zero. Enquanto se faz $X_{\overline{N}+1}$ até $X_{2\overline{N}-1}$ igual ao conjugado de $X_{\overline{N}-1}$ até X_1(Figura 20(a)).

Sendo o conjugado de $Z = a + jb$ igual a $Z^* = a - jb$, então para fazer a operação de conjugado deve-se mudar o sinal da parte imaginária, ou seja, fazer um inversor de sinal. Em hardware, o inversor de sinal de um número binário em complemento dois, b, é implementado invertendo-se todos os bits do número, $\overline{b}$, e somando um, $-b = \overline{b} + 1$ (Figura 20(b))



(a) *Implementação* **(b)** *Conjugado*

Figura 20: *Implementação da simetria Hermitiana*

4.5 A (I)FFT

Como visto na seção 3.6, a modulação MCM pode ser feita por meio de uma IDFT. Pode-se usar a implementação rápida da IDFT, a IFFT[2], diminuindo o tempo de processamento e o hardware utilizado (Figura 21). A demodulação, da mesma forma, pode ser feita pela DFT, ou melhor, pela FFT, que é a sua implementação eficiente.

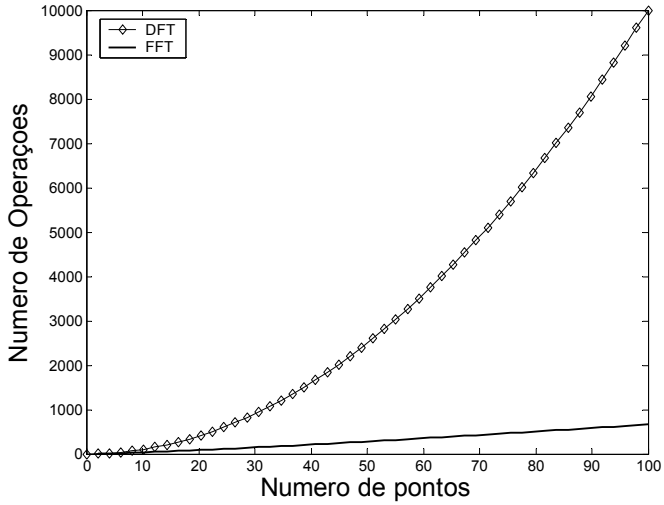


Figura 21: *Número de operações da DFT e da FFT*

A DFT de um vetor x_n de N pontos complexos no tempo é definida como[15]:

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{nk} \tag{4.2}$$

para $k = 0, \ldots, N-1$.

A inversa da DFT (IDFT) de uma seqüencia X_k de N pontos complexos na freqüência é definida como:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k W_N^{-nk} \tag{4.3}$$

para $n = 0, \ldots, N-1$.

Onde:

$$W_N = e^{-j2\pi/N} \tag{4.4}$$

O número de operações de multiplicação complexa para computar a DFT de um sinal com N pontos usando diretamente as formulas acima é proporcional a N^2.

[2]O D, da IDFT, indica o algoritmo enquanto o F indica a implementação, porém será usado a IFFT para representar ambas.

A FFT calcula a DFT com uma grande redução na quantidade de operações, retirando-se várias redundâncias existentes no cálculo direto da DFT. Essa eficiência é conseguida ao custo de um passo adicional para se re-ordenar os dados a fim de se determinar o resultado final. Esse passo adicional, desde que implementados eficientemente, não aumentarão de forma significativa a complexidade computacional do cálculo. Como resultado, a FFT é um algoritmo extremamente eficiente que proporciona uma boa implementação em hardware.

Para grandes valores de N, a eficiência computacional é conseguida quebrando-se sucessivamente a DFT em cálculos menores. Isso pode ser feito tanto no domínio do tempo, como no domínio da freqüência, como discutido adiante.

A principal observação a se fazer é que quando N não é um número primo, a DFT pode ser decomposta em várias DFTs de tamanho menores. Quando N é uma potência de r, pode-se dividir a DFT em $\log_r N$ estágios. Nesse caso a FFT é chamada de radix-r. Os algoritmos mais populares são as FFT radix-2 e radix-4.

Para o caso da radix-2 de N pontos, o número de operações de multiplicação é reduzido para a ordem de $(N/2)\log_2 N$. Veja a comparação entre a FFT e a DFT na Figura 21.

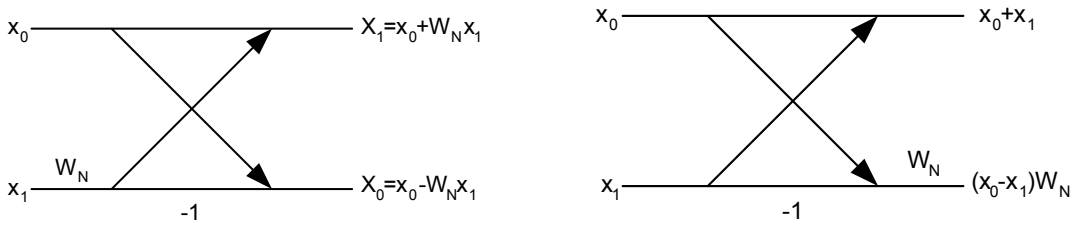
4.5.1 Decimação no tempo e na freqüência

Como mencionado, é possível dividir sucessivamente a entrada da FFT gerando pequenas seqüencia no domínio do tempo, por isso o nome decimação no tempo, DT. Também é possível se dividir a seqüencia de saída da FFT sucessivamente, ao invés de se dividir a entrada. Essa implementação é chamada de decimação na freqüência, DF. A decimação pode ser utilizada tanto para a FFT como para a IFFT.

É possivel repetir o processo até atingir o nível máximo possível de divisão. Nesse ponto a decimação gera-se um bloco básico que será utilizado em toda a FFT, chamado de *butterfly* devido ao seu formato. Um exemplo da *butterfly* da FFT radix-2 DT está na Figura 22(a).

Um exemplo de uma FFT radix-2 DT, para $N = 8$ é mostrada na Figura 23(a). Percebe-se que é necessário uma alteração na entrada de dados, pois tem de se separar a parte par da parte ímpar.

Da mesma forma que a decimação no tempo gera uma *butterfly*, a decimação na freqüência gera uma *butterfly* correspondente. Um exemplo para a *butterfly* gerada pela

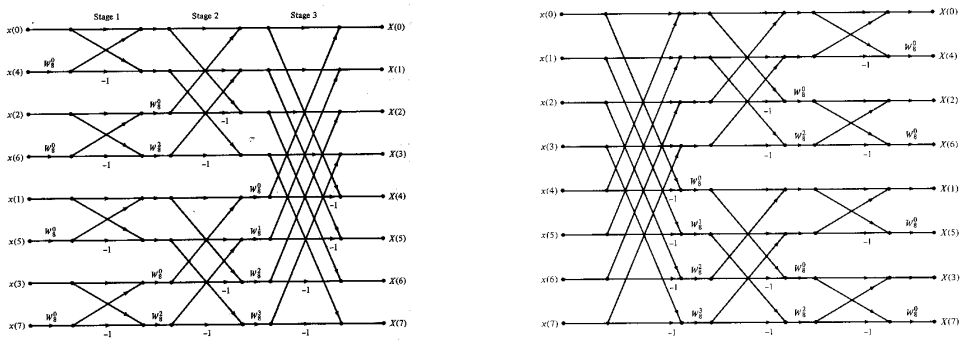


(a) *Butterfly DT* **(b)** *Butterfly DF*

Figura 22: *Decimação da* butterfly

FFT radix-2 DF está na Figura 22(b). Porém a alteração no fluxo de dados terá de ser feito na saída, e não mais na entrada.

Um exemplo de uma FFT radix-2 DF, para $N = 8$ é mostrada na Figura 23(b).



(a) *FFT* **(b)** *FFT*

Figura 23: *Implementação da FFT de 8 pontos*

4.5.2 Implementação da *Butterlfy*

Na *butterfly* da FFT a multiplicação complexa é feita entre um número complexo, $(a + jb)$, e W_N^q, W_N é definida em (4.4). Expandindo[3]:

$$(a + jb)\left(\cos\left(2\pi\frac{q}{N}\right) - j\sin\left(2\pi\frac{q}{N}\right)\right) \tag{4.5}$$

Resolvendo:

$$a\cos\left(2\pi\frac{q}{N}\right) - ja\sin\left(2\pi\frac{q}{N}\right) + jb\cos\left(2\pi\frac{q}{N}\right) - j^2b\sin\left(2\pi\frac{q}{N}\right)$$

Obtém-se:

$$\left(a\cos\left(2\pi\frac{q}{N}\right) + b\sin\left(2\pi\frac{q}{N}\right)\right) + j\left(-a\sin\left(2\pi\frac{q}{N}\right) + b\cos\left(2\pi\frac{q}{N}\right)\right) \tag{4.6}$$

[3]Fórmula de Euler: $e^{j\theta} = \cos(\theta) + j\sin(\theta)$

Da mesma forma, na IFFT a multiplicação complexa é feita entre um número complexo e W_N^{-q}, obtendo:

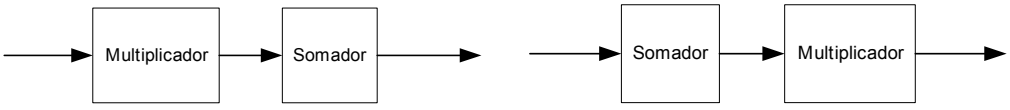
$$\left(a\cos\left(2\pi\frac{q}{N}\right) - b\sin\left(2\pi\frac{q}{N}\right)\right) + j\left(a\sin\left(2\pi\frac{q}{N}\right) + b\cos\left(2\pi\frac{q}{N}\right)\right) \tag{4.7}$$

A soma utilizada na *butterfly* possui o mesmo algoritmo, tanto para a FFT quanto para a IFFT, $a + b$ e $a - b$. Para não haver perigo de *overflow* devido a soma em complemento dois, se faz a extensão do sinal no número binário, repetindo o bit mais significativo, assim ao se somar 2 números binários de 10 bits, teremos de primeiro fazer a extensão do sinal, obtendo assim, dois números de 11 bits, para fazer a soma, onde o resultado também será de 11 bits. Esse procedimento tem de ser feito a cada vez que for somar ou subtrair um número.

Já para a multiplicação, a saída tem de ser do tamanho da soma do número de bits dos dois multiplicandos. Dessa forma, para fazer a multiplicação de dois números de 10 bits, teremos de ter uma saída de 20 bits. Esse procedimento tem de ser feito a cada multiplicação.

Se não houver impedimento, pode-se fazer uma rotação para direita (divisões por dois) nos números e reduzir o seu tamanho, desde que no final do procedimento seja aplicado um multiplicador correspondente.

A ordem como será feita a *butterfly* é definida pela decimação do radix. Se for DT, primeiro se faz multiplicação e depois a soma (Figura 24(a)). Se for DF, primeiro se faz a soma e então se faz a multiplicação(Figura 24(b)).



(a) *Butterfly DT* **(b)** *Butterfly DF*

Figura 24: *Implementação da Butterfly*

A multiplicação existente na *butterfly* mostrada em (4.6) e (4.7), exige uma certa atenção, pois, se não for implementada eficientemente, irá degradar muito o desempenho da FFT. Basicamente existe dois métodos para se fazer a multiplicação: armazenar em um tabela os valores de seno e cosseno ou calcular na hora o seu valor através do CORDIC.

4.5.2.1 Tabela de senos

A maneira mais direta de se fazer a multiplicação complexa mostrada em (4.6) e (4.7) é armazenar, ou em RAM ou nos LUTs dos CLBs, os valores de seno e cossenos para todos os ângulos desejáveis e implementar os quatro multiplicadores reais (dois para a parte real e dois para a parte imaginária) e dois somadores-subtratores (um para cada parte) necessários para o cálculo.

Entretanto, é possível construir o multiplicador complexo com somente três multiplicadores reais e três somadores-subtratores, pois uma operação é pré-computada[16].

A multiplicação (4.5) pode ser simplificada da seguinte forma:

Para simplificar a notação, representa-se

$$(a + jb)\left(\cos\left(2\pi\frac{q}{N}\right) - j\sin\left(2\pi\frac{q}{N}\right)\right)$$

como

$$(a + jb)(c - js)$$

onde $c = \cos$ e $s = \sin$, obtém-se (Equação (4.6)):

$$(ac + bs) + j(-as + bc)$$

Somando e subtraindo de $bc + jac$, tem-se para a parte real:

$$\begin{aligned}\Re &= ac + bs + bc - bc \\ &= bc + ac + bs - bc \\ &= c(b + a) + b(s - c)\end{aligned}$$

e para a parte imaginaria:

$$\begin{aligned}\Im &= -as + bc + ac - ac \\ &= -ac - as + bc + ac \\ &= a(-c - s) + c(b + a)\end{aligned}$$

Então, fazendo $e = b + a$:

$$\begin{aligned}\Re &= ce + b(s - c) \\ \Im &= ce + a(-c - s)\end{aligned}$$

Se for armazenados cos, $-(\cos + \sin)$ e $(\sin - \cos)$ de forma pré-computada para os ângulos desejados, será necessário apenas três multiplicações e três adições (uma para o cálculo de e). Eliminando-se, assim, uma multiplicação tornando mais eficiente o cálculo.

O mesmo processo pode ser aplicado em (4.7), obtendo-se um resultado parecido.

4.5.2.2 Cordic

Outra forma de se implementar as multiplicações mostrada nas equações (4.6) e (4.7) é através do CORDIC (*Coordinate Rotation Digital Computer*).

A equação 4.6, reescrita a seguir,

$$(R + jI) = \left(a\cos\left(2\pi\frac{q}{N}\right) + b\sin\left(2\pi\frac{q}{N}\right)\right) + j\left(-a\sin\left(2\pi\frac{q}{N}\right) + b\cos\left(2\pi\frac{q}{N}\right)\right)$$

pode ser representada de forma matricial, separando a parte real da imaginária:

$$\begin{bmatrix} I \\ R \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix}$$

onde $\theta = 2\pi\frac{q}{N}$.

Da mesma forma, a equação 4.7:

$$(R + jI) = \left(a\cos\left(2\pi\frac{q}{N}\right) - b\sin\left(2\pi\frac{q}{N}\right)\right) + j\left(a\sin\left(2\pi\frac{q}{N}\right) + b\cos\left(2\pi\frac{q}{N}\right)\right)$$

também pode ser representada de forma matricial:

$$\begin{bmatrix} R \\ I \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

onde $\theta = 2\pi\frac{q}{N}$.

A matriz de transformação das equações acima:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.8}$$

é a mesma matriz de rotação de vetores de um ângulo θ. Portanto podemos considerar as multiplicações complexas em (4.6) e em (4.7) como uma rotação de $a + jb$ de um ângulo $\theta = 2\pi\frac{q}{N}$.

A rotação de um ângulo θ pode ser executado em vários passos, usando um método recursivo, em que cada passo faz uma pequena rotação, de ângulo θ_n, no vetor até fazer

a rotação completa.

O CORDIC[17] utiliza desse procedimento recursivo para fazer a rotação do vetor. Dessa forma, é possível usar apenas rotação, somadores/subtratores e comparadores, que podem ser implementados facilmente em hardware. Por essa razão ele pode ser implementado de forma bastante eficiente em uma FPGA[18]. O procedimento de como isso é feito é explicado abaixo.

Isolando $\cos(\theta_n)$ na matriz de rotação (4.8):

$$\cos\theta_n \begin{bmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{bmatrix} \tag{4.9}$$

e fazendo o ângulo de cada passo, θ_n, de tal forma que $\theta_n = \arctan\left(\frac{1}{2^n}\right)$, é possível substituir a tangente da matriz de rotação (4.9) por uma simples operação de rotação para direita do número binário, pois ela será uma potência de 2.

A soma de todos os passos de rotação geram uma rotação de ângulo θ, $\sum_{n=0}^{\infty} S_n\theta_n = \theta$, onde $S_n = \{-1; +1\}$, portanto obtém-se

$$\tan\theta_n = S_n 2^{-n} \tag{4.10}$$

Aplicando (4.10) em (4.9), obtém:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos\theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \tag{4.11}$$

onde X_{n+1} e Y_{n+1} são o proximo passo de X_n e Y_n, respectivamente.

Na equação (4.11) há ainda o $\cos\theta_n$, isolado, para se calcular, e como o cálculo é iterativo serão várias multiplicações, que podem ser agrupadas como:

$$K = \frac{1}{P} = \prod_{n=0}^{\infty} \cos\left(\arctan\left(\frac{1}{2^n}\right)\right) \tag{4.12}$$

onde n é o número de interações.

Efetuando esse cálculo verificá-se que a partir da 14ª iteração o valor de K tende a uma constate, como mostrado na Figura 25, que pode ser aproximado por 0.6073. O valor de P será, aproximadamente, 1.6468.

Portanto, a partir desse ponto podemos substituir a multiplicação de vários cossenos (4.12) por uma multiplicação pela constante P, 1.6468.

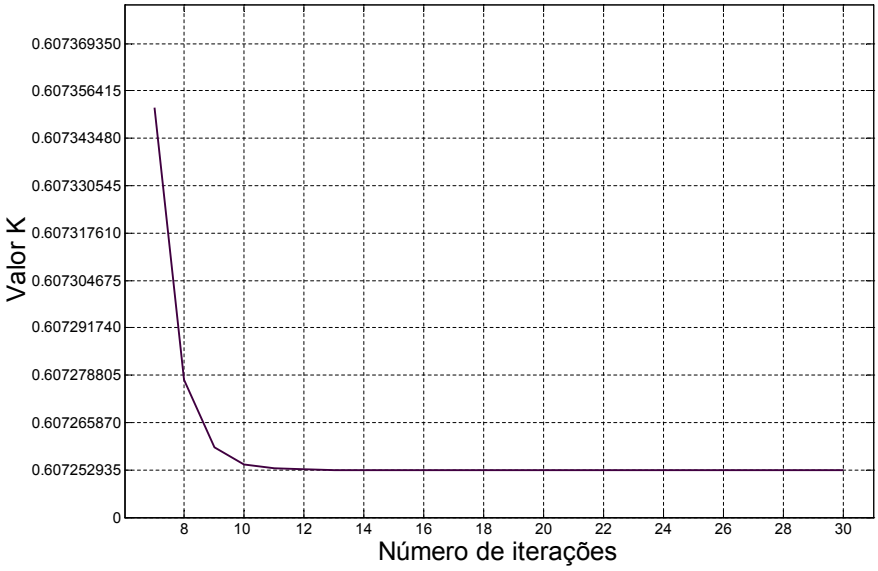


Figura 25: *Comportamento do fator K do CORDIC*

Desse forma, a matriz de rotação (4.8) se resume a rotações binárias para direita, somadores-subtratores e um multiplicador por uma constante.

Ao se dividir o ângulo de rotação, θ, em vários passos, tem de se ter a informação do quanto que falta para terminar a rotação, faz-se:

$$Z_{n+1} = \theta - \sum_{i=0}^{n} \theta_i$$

E para cada passo que se dá, S_n é recalculada em função do sinal de Z_n

$$S_n = \begin{cases} -1 \; if \; Z_n < 0 \\ +1 \; if \; Z_n \geq 0 \end{cases}$$

podendo ser implementado por um simples comparador.

Vários cálculos possíveis com o CORDIC, como seno, cosseno, transformação de polar para retangular e vice-versa, rotação de vetor, arco-tangente e várias outras são mostrado em [18]. Entre essa a que interessa para o cálculo da FFT é a de rotação de vetor.

Existem vários métodos de se implementar o CORDIC em uma FPGA[18], como por exemplo, de forma totalmente iterativa ou em pipeline, fazendo todas as operações de forma serial ou não.

Como o cálculo da FFT tem de ser efetuado o mais rápido possível, foi implementado o CORDIC em pipeline[17]. Na FFT, como se repete muitas vezes o cálculo com o

CORDIC, pode-se deixar para fazer a multiplicação pela constante P apenas no final, ou seja, ao terminar o calculo da CORDIC. Não se faz a multiplicação por P para a correção, ganhando eficiência e espaço na FPGA.

4.5.3 Radix-2

Na implementação radix-2, o bloco básico é uma *butterfly* de dois pontos, mostrada na Figura 22. Ela consiste de duas entradas e duas saídas. Internamente ela faz uma multiplicação complexa e duas somas complexas, que podem ser implementados pelos métodos discutidos anteriormente.

Se for feita a decimação no tempo, tem-se (Figura 22(a)):

$$\begin{aligned} X_1 &= x_0 + W_N x_1 \\ X_0 &= x_0 - W_N x_1 \end{aligned} \tag{4.13}$$

Se for feita a decimação na freqüência, tem-se: (Figura 22(b)):

$$\begin{aligned} X_1 &= x_0 + x_1 \\ X_0 &= (x_0 - x_1) W_N \end{aligned} \tag{4.14}$$

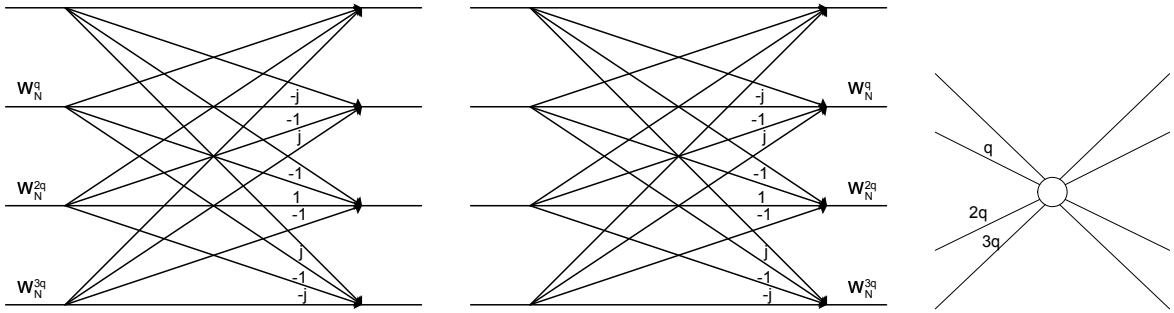
Mostra-se[16] que o cálculo da FFT utilizando radix-2 requer $(N/2)\log_2 N$ multiplicações complexas e $N \log_2 N$ adições complexas (Figura 27).

4.5.4 Radix-4

Na implementação radix-4, o bloco básico é uma *butterfly* de quatro pontos mostrada na Figura 26. Ela consiste de quatro entradas e quatro saídas. Internamente ela faz três multiplicações complexas e quatro somas complexas.

Há uma grande diferença, em relação ao radix-2, na implementação do algoritmo para se fazer a soma da butterfly (Figura 26(a)):

$$\begin{aligned} X_0 &= x_0 + x_1 + x_2 + x_3 \\ X_1 &= x_0 - jx_1 - x_2 + jx_3 \\ X_2 &= x_0 - x_1 + x_2 - x_3 \\ X_3 &= x_0 + jx_1 - x_2 - jx_3 \end{aligned}$$



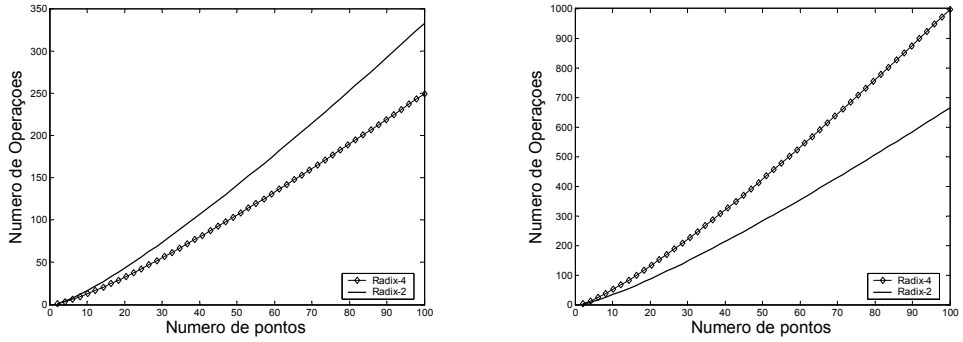
(a) *Decimação no tempo* **(b)** *Decimação na freqüência* **(c)** *Representação simplificada*

Figura 26: *Butterfly Radix-4*

Representando de forma matricial:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & +j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4.15}$$

Com a FFT utilizando radix-4, mostra-se[16] que o cálculo requer $(3N/8)\log_2 N$ multiplicações complexas e $(3N/2)\log_2 N$ adições complexas. Portanto houve uma redução de 25% no número de multiplicações (Figura 27(a)), mas o número de adições aumentou em 50% em relação à FFT radix-2 (Figura 27(b)).



(a) *Número de multiplicação* **(b)** *Número de adição*

Figura 27: *Comparação entre a FFT radix-2 e radix-4*

É interessante notar que se a soma da *butterfly*, equação (4.15), for feita em dois passos, o número de adições cai de 12 para 8. Isso é conseguido separando a matriz em

duas:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

fazendo o número de adições complexas ser reduzido para $N \log_2 N$, que é idêntico à FFT radix-2.

Portanto, com a FFT radix-4 consegue-se uma redução de 25% no número de multiplicações complexas necessárias, tornando o cálculo mais rápido.

A multiplicação complexa na *butterfly* do radix-4 é feita da mesma forma descrita na seção 4.5.2.

4.5.5 Utilização da memória

Basicamente, o algoritmo da FFT consiste em pegar os dados da memória (2 para radix-2 e 4 para radix-4), fazer a *butterfly* e retornar os dados calculados para a memória. Uma vez que os dados de entrada da *butterfly* já foram processados não há mais motivo de se guardá-los. Então podemos salvar os dados processados na mesma posição de memória. Com isso, a quantidade de memória necessária será fixa. A Figura 28 mostra a forma básica de se implementar essa FFT.

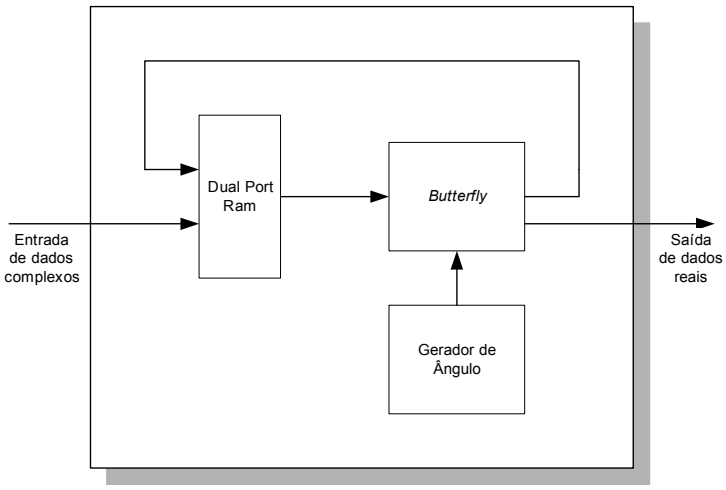


Figura 28: *Diagrama de blocos de uma implementação básica da FFT*

Essa implementação não é útil para o OFDM, pois nesse sistema, enquanto a FFT estiver processando, não é possível escrever os dados da entrada na memória. Outro problema é que a saída tem de ser armazenada para poder se fazer o cyclic-prefix.

Para resolver o problema do cyclic-prefix, pode-se colocar uma memória RAM na saída para armazenar os dados. E para que o sistema funcione de forma contínua, é necessário que a memória de entrada tenha o seu tamanho dobrado e o seu conteúdo separado em duas partes, para que enquanto a FFT estiver lendo de uma parte, seja possível escrever na outra e vice-versa. Também é necessário incluir outra memória RAM para que depois da primeira leitura a FFT possa armazenar e processar os dados sem interferir na memória de entrada. Essa configuração é mostrada na Figura 29.

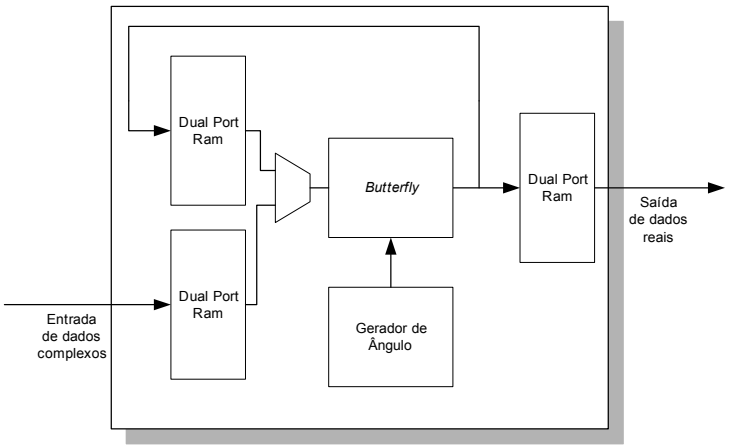


Figura 29: *Diagrama de blocos de uma implementação da FFT com 3 RAMs*

4.5.6 Mapeamento da memória

Na implementação da FFT utilizando o radix, percebe-se que é necessário o remapeamento da memória. Na implementação DT esse remapeamento é feito a partir da entrada, e na implementação DF é a partir da saída da FFT. Porém esse remapeamento pode ser feito de forma simples.

Por exemplo, para a FFT radix-2 DT, a entrada tem de ser escrita nos endereços de memória 0, 2, 4, 6, 1 3 , 5 e 7. Após processado um estágio tem de se escrever em 0, 4, 2, 6, 1, 5, 3, 7. Esse embaralhamento segue uma ordem bem definida. Se for observada essa seqüencia de forma binária, como na Table 3:

Observa-se que foi feita um mudança nas posições dos bits. No estágio 1 o bit mais significativo foi trocado com o menos significativo

$$b_3 b_2 b_1 \longrightarrow b_1 b_2 b_3$$

onde b_n significa o bit na posição n.

Seqüencia Binária		Estágio 1		Estágio 2		Estágio 3	
Decimal	Binário	Decimal	Binário	Decimal	Binário	Decimal	Binário
0	000	0	000	0	000	0	000
1	001	4	100	2	010	1	001
2	010	2	010	4	100	4	100
3	011	6	110	6	110	5	101
4	100	1	001	1	001	2	010
5	101	5	101	3	011	3	011
6	110	3	011	5	101	6	110
7	111	7	111	7	111	7	111

Tabela 3: *Digit Reverse*

Da mesma forma, no estágio 2 é observada o seguinte comportamento:

$$b_3b_2b_1 \longrightarrow b_2b_1b_3$$

E, no estágio 3:

$$b_3b_2b_1 \longrightarrow b_2b_3b_1$$

Pode-se portanto utilizar um contador, e implementar uma lógica na saída de tal forma que dependendo do estágio, se faça a conversão apropriada.

Uma análise similar pode ser feita para a FFT radix-4. Nessa implementação é trocado dois bits por vez.

4.6 O cyclic-prefix

A inclusão do cyclic-prefix no transmissor é feita de forma extremamente simples pelo conversor paralelo-serial. Basta o conversor ler duas vezes os ν bits que pertencem ao cyclic-prefix no buffer que é a memória RAM da IFFT, como explicado acima.

A exclusão do cyclic-prefix no receptor também é feita de forma simples, bastando não passar os ν bits que pertence ao cyclic-prefix para a FFT.

4.7 Conversor digital-analógico e analógico-digital.

Por fim, é feita a conversão para analógico, para que o sinal possa ser transmitido. Em [6, pgs. 154-155] mostra-se que para um conversor digital-analógico de 12 bits, consegue-se transmitir até 13 bits por canal.

Já no receptor, mostra-se em [6, pag. 168] que a resolução do conversor analógico-digital pode ser diminuída, onde, segundo [6], para a número de bits do conversor "10 é amplo, 9 provavelmente é suficiente, 8 será espremido".

4.8 Melhorias

Ao se analisar o sistema, com todos os componentes percebe-se um problema. Algumas partes foram implementados de forma paralela, ou seja, recebe-se todos os canais ao mesmo tempo, e transmite-se todos os canais ao mesmo tempo. Porém, o elemento de armazenagem principal utilizado é a memória RAM (Figura 18(c)), e o acesso a ela é de forma serial, se escreve e se lê em um endereço por vez[4]. Como a quantidade de memoria RAM na FPGA é limitada, é necessário adequar o sistema a essa situação.

A primeira observação a ser feita é quanto ao tipo da FFT, se é radix-2 ou se é radix-4, DT ou DF. Como vimos a Radix-4, apesar de ser mais limitada quanto às possibilidades de escolha do número do pontos, possui 25% a menos cálculos de multiplicação, fazendo-a a opção ideal. Quanto a decimação, se deve ser DT ou DF, ao se analisar a FFT, percebe-se que quando a FFT é implementada com decimação na freqüência, no último estágio de cálculo da *butterfly* todos os ângulos serão zero, fazendo a *butterfly* desnecessária. Com isso podemos ligar a memória RAM de saída direto ao somador, fazendo essa modificação o sistema fica como mostrado na Figura 29.

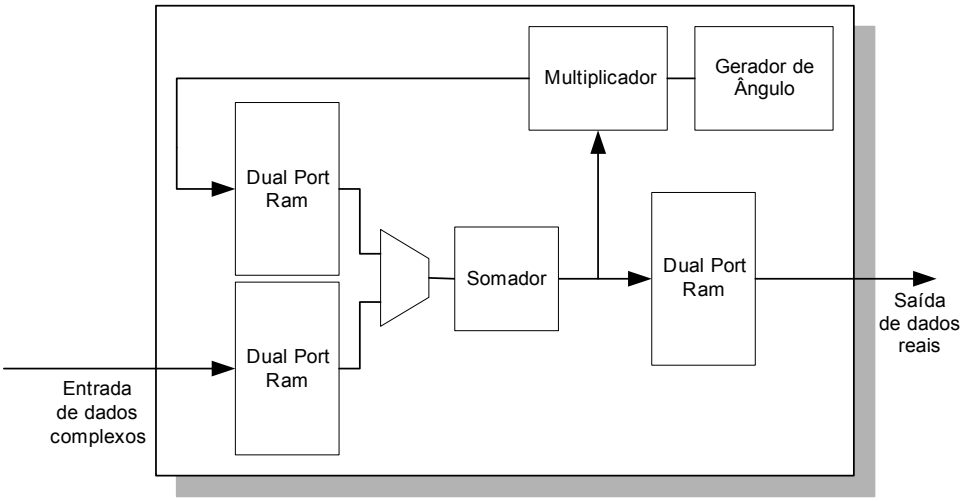


Figura 30: *FFT radix-4 DF*

A segunda observação a ser feita é que pode-se juntar a FFT com a simetria her-

[4]Apesar de se poder ler e escrever ao mesmo tempo, porém em endereços diferentes

metiana (Figura 31), colocando-se o bloco de conjugado (Figura 20(b)) entre a memória RAM de entrada da FFT e a *butterfly*. Dessa forma é necessário um controle do momento em que se deve fazer ou não o conjugado.

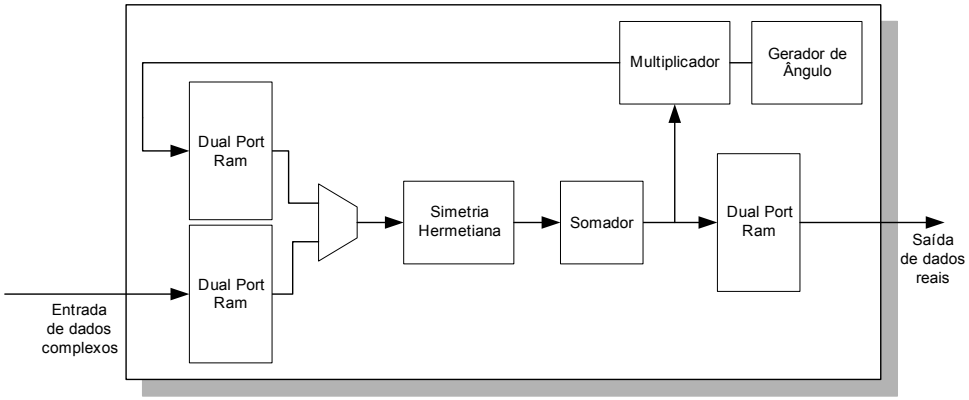


Figura 31: *A FFT integrada com a simetria hermetiana*

Para que possa ser feita a simetria hermetiana faz-se os endereços da memória de entrada 0 e N igual a zero, $0 + j0$, e nunca se escreve nesses endereços. A escrita e a leitura da RAM também devem ser ajustadas. Observa-se que se, por exemplo, para a primeira seqüencia de leitura de uma FFT radix-4 DF, mostrada na Tabela 4, os endereços de leitura forem tratados como números em complemento dois, e pegarmos o seu valor absoluto, estaremos lendo os endereços em simetria.

Decimal	Binário	Comp. 2	Abs.
0	000000	0	0
16	010000	16	16
32	100000	-32	32
48	110000	-16	16
1	000001	1	1
17	010001	17	17
33	100001	-31	31
49	110001	-15	15
2	000010	2	2
18	010010	18	18
34	100010	-30	30

Tabela 4: *Conversão para leitura da hermetiana*

A terceira observação é sobre o codificador. Como os dados serão armazenado seqüencialmente na memória RAM da FFT, podemos colocá-lo em série com essa memória de tal forma que sempre que for escrever um dado na memória, antes seja feita a codificação, comforme Figura 32.

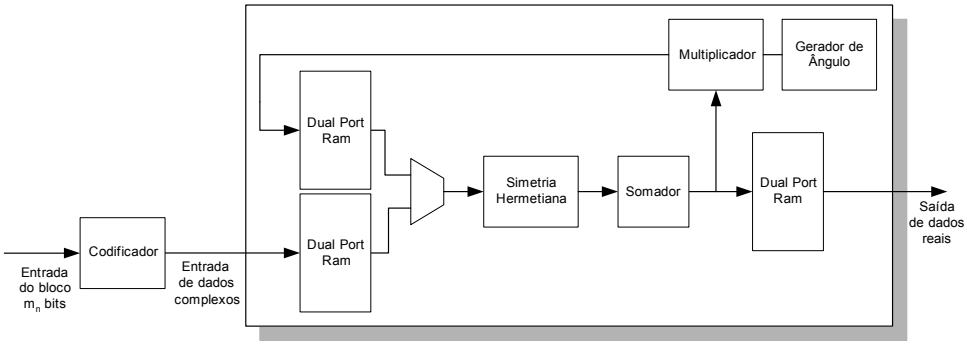


Figura 32: *FFT com a simetria e codificador*

Outra questão importante, discutida na seção 4.5.2, pagina 33, é sobre como se evitar o *overflow*. Para isso foi mostrado que aumentando-se o número de bits, porém como a quantidade de bits da memória é fixo, é necessário se reduzir a quantidade de números para esse tamanho de palavra. Porém, não se pode apenas rotacionar os dados, pois pode ocorrer erro de *overrun*, ou seja, ao se estourar a capacidade de representação do número binário ele volta a zero e recomeça a contagem.

Para se evitar isso se faz uma rotação com travamento de *overrun*. Se rotaciona alguns bits, e se mesmo assim sobrar bits a direita do limite especificado, se limita à máxima representação possível, tanto positiva quanto negativa. Dessa forma temos o transmissor OFDM mostrado na Figura 33.

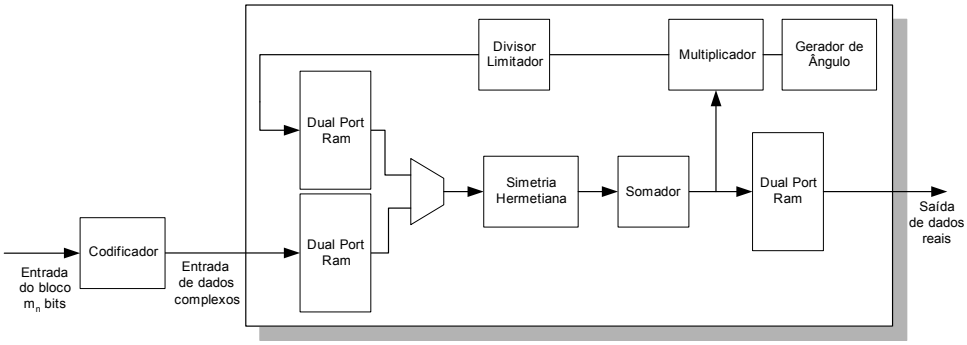


Figura 33: *Transmissor OFDM*

O receptor é mostrada na Figura 34. Observe que não existe o bloco correspondente a simetria hermetiana, pois a mesma é feita eliminando os canais que saem da FFT.

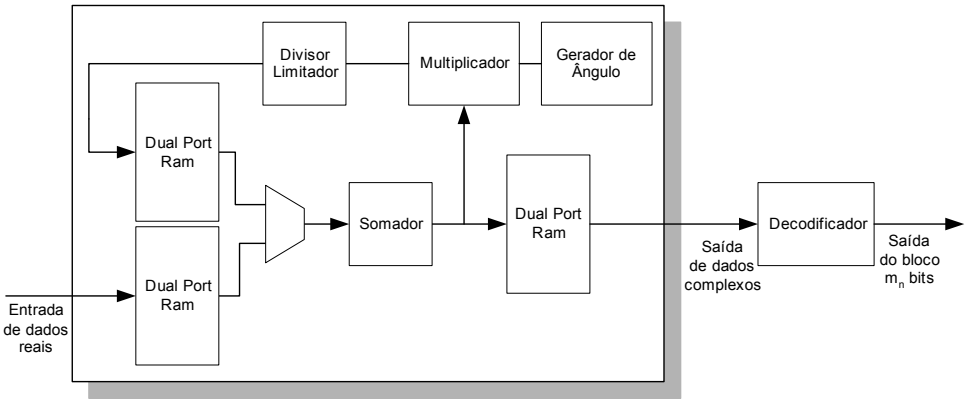


Figura 34: *Receptor OFDM*

4.9 Quantidade de memória

A quantidade de memória a ser utilizada é um ponto crucial. Pois é nela que será armazenado todos os dados a serem processados. Por isso, temos de especificar um FPGA que tenha a quantidade necessária de memória RAM.

Analisando-se as memórias utilizadas pelo transmissor, e sabendo-se que o codificador gera N números complexos, que a simetria hermetiana é feita na própria FFT, e sabendo que a memória de entrada do FFT é o dobro do que ela precisa armazenar então será necessário duas memórias RAM de $2N+2$ (dois endereços iguais a zero, para a hermetiana) de capacidade de endereçamento.

A memória de processamento também armazena números complexos e deve ter o comprimento igual a $2N + 2$. Portanto, será necessário mais duas memórias RAM de comprimento $2N + 2$. Já na memória de saída, que o número armazenado é real, só será necessário um memória RAM de comprimento $2N + 2$.

A mesma análise pode ser feita para o receptor, chegando-se a mesma quantidade de memória.

No total será necessário então 5 memórias de comprimento $2N+2$ tanto no transmissor quanto no receptor.

4.10 A implementação

Como discutido, pretende-se implementar o sistema OFDM mostrado na Figura 33 e Figura 34 utilizando uma FPGA. As características do OFDM a ser implementado são: codificador e decodifcador de QAM-4, IFFT e FFT de 64 pontos, 12 bits de precisão de

entrada, implementada usando radix-4 DF por CORDIC.

Como base para a implementação, foi adquirido um código aberto de uma FFT[5]. A Figura 35 mostra o diagrama de blocos dessa FFT. Analisando a FFT adquirida, percebe-se que é implementado uma FFT radix-4 DF por CORDIC. O número de pontos e o número de bits por palavra é configurável. Essas características correspondem com as que do OFDM que se pretende implementar. Porém a utilização da memória está conforme mostrado na Figura 28, sendo necessário então alguns ajustes.

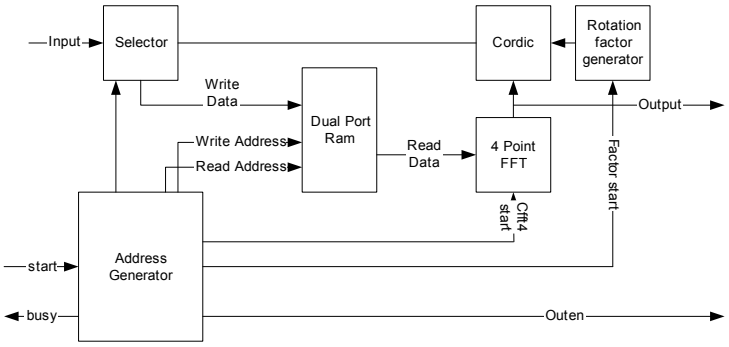


Figura 35: *Diagrama de blocos da FFT*

Foram reutilizados os blocos:

Dual Port Ram Sintetiza os blocos de memória RAM disponíveis na FPGA. Ele foi implementada de forma a ser totalmente configurável.

4 Point FFT Implementa a somatória realizada pela *butterfly* como o descrito em (4.15).

Cordic Reutiliza o código do CORDIC disponível também no Opencores[17]. Esse código implementa uma CORDIC em pipeline de tamanho fixo.

Rotation factor generator Implementa o gerador de ângulo para funcionar com o CORDIC.

Divisor-limitador Apesar de não aparecer na Figura 35, há um bloco que implementa o divisor-limitador discutido para evitar o *overrun*.

Em adição a esses blocos, foram desenvolvidos todos os outros que estão faltando para se ter o sistema OFDM planejado, como o multiplexador, o conjugado, as memórias para número complexos, o codificador e o decodificador e o conversor serial/paralelo.

[5]Adquirido em http://www.opencores.org/projects/cfft/

O transmissor foi estruturado como mostrado na Figura 33 e o receptor como mostrado na Figura 34.

O sistema foi desenvolvido em VHDL, separando-se a parte do cálculo da parte de controle. A parte de cálculo foi implementada de forma totalmente ajustável, tanto em precisão da FFT, quanto no número de pontos. A parte de controle, entretanto, está otimizado para operar em 64 pontos.

Como foi planejado uma IFFT de 64 pontos, o sistema terá $2N + 2 = 64 \rightarrow N = 31$ canais disponíveis para transmissão. Com isso serão necessários 10 memórias de 64 endereços para o modem OFDM. A Spartan II escolhida é suficiente para essa implementação.

O codificador escolhido foi o 4-QAM devido, principalmente, a simplicidade de se fazer o decodificador. Com isso, poderemos transmitir 2 bits por canal, num total de 31 canal, tendo 62 bits por símbolo OFDM.

5 Análises

Para se efetuar a FFT é preciso $\log_r N$ estágios, e em cada estágio se lê a memória, calcula-se a somatória e a multiplicação pelo CORDIC, da *butterfly*, e faz a divisão-limitação por quatro para então podermos rearmazernar na memória. Pode-se representar graficamente esse processo como mostrado na Figura 36.

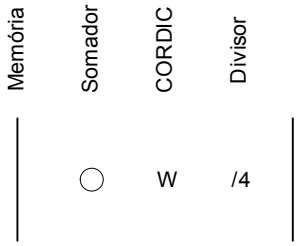


Figura 36: *Representação dos cálculos efetuados em um estágio*

Para o caso proposto, FFT radix-4 de 64 pontos, teremos $\log_4 64 = 3$ estágios, como mostrado na Figura 37. Perceba que o último estágio não contem o CORDIC e o divisor, pois após o somador os dados são armazenados diretamente na memória de saída.

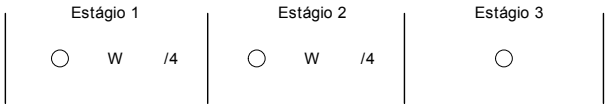


Figura 37: *Representação da FFT 64*

5.1 Fator de correção

Como dito, não será aplicado o fator de correção P do CORDIC durante o processamento. Portanto se faz necessário a análise do fator de correção que deve ser aplicado ao sistema, no final de todo o processo.

Para isso será utilizado uma entrada no CORDIC, porém sem sofrer rotações, ou seja com ângulo zero. Com isso será medido o ganho dado pelo CORDIC implementado.

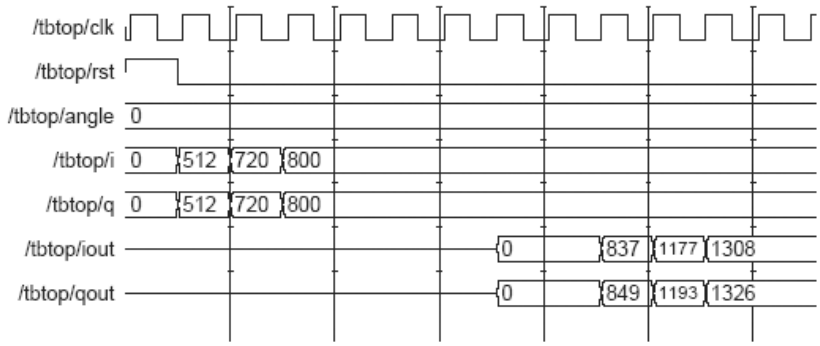


Figura 38: *Simulação do CORDIC*

Por exemplo (Figura 38), entrando-se com o número $512 + j512$ obtém-se $837 + j849$. Considerando o ganho como $w = saida/entrada$, obtém-se:

$$\begin{aligned} w &= \frac{837 + j849}{512 + j512} \\ &= 1.646484375 + j0.01171875 \end{aligned}$$

Considerando um estágio de cálculo, como mostrado na Figura 36, e todos os valores de entrada como A, a saída do somador será $4A$. Ao passar pelo CORDIC será então $4Aw$ e quando passar pelo divisor por quatro resultará em $4Aw/4 = Aw$, ou seja, cada estágio, com exceção do último, da um ganho de w a entrada.

Para a FFT de $N = 64$ pontos são necessários 3 estágios. Tem-se no fim do segundo estágio Aww. No terceiro e último estágio, após o somador, obtém-se $4Aww$, que o resultado final. Aplicando um fator de correção c, devido ao fato de não ter sido colocado um multiplicador na saída do CORDIC e como foi considerado que todas as entradas possuem valores iguais a A, tem-se

$$4Awwc = NA$$

onde c é o fator de correção. Então, isolando c acha-se o fator de correção que tem de ser aplicado à FFT, em função do número de pontos:

$$c = NA/4Aww = N/4ww$$

Para esse caso, $N = 64$, tendo então $c = 16/w^2$, resultando num fator de correção de $c = 5.9012 - j0.0840$.

Já para a IFFT de $N = 64$ pontos, a saída deveria ser $\frac{1}{N}NA = A$, portanto

$$4Awwc = A$$

Isolando c, obtém-se

$$c = 1/4ww$$

Então, para a IFFT de $N = 64$ pontos, $c = 1/4w^2 = 0.0922 - j0.0013$.

5.2 Valores máximos

Para se evitar que ocorra overflow em alguma parte do cálculo, deve-se analisar o sistema na pior situação, quando todas as entradas forem iguais a uma constante A, que será o máximo valor possível na entrada.

Como o sistema está sendo projetado para 64 pontos, e 12 bits de entrada em complemento dois, teremos na saída 14 bits em complemento dois, pois adiciona-se mais dois bits para evitar *overflow* no cálculo da somatória. O valor máximo na saída será $2^{13}-1 = 8191$, como a saída é um número complexo, $MS = 8191 + 8191i$, onde MS é o valor da máxima saída possível.

Como obtido anteriormente para a (I)FFT, a saída terá o valor de $4Aww$, então

$$4Aww = MS \rightarrow A = MS/4ww$$

Com isso, a entrada máxima será de

$$\begin{aligned} A &= 766.0102_d + j744.5071_d \\ &= 001011111110_2 + j001011101001_2 \end{aligned}$$

Para se ter a maior representação positiva com todos os bits um é necessário diminuir um pouco o valor máximo da entrada para:

$$000111111111_2 + j000111111111_2 = 511_d + j511_d = 1FF_h + 1FF_h$$

Portanto o máximo valor que pode ser aplicado na entrada da (I)FFT para não ocorrer overflow será de $511_d + j511_d$

5.3 Ponto fixo

Tendo sido o limite máximo estipulado na seção anterior, faz-se necessário analisar introdução do ponto fixo.

Agora, considerando a entrada com o ponto fixo na quarta casa, de tal forma que o valor máximo seja $1.FF_h$, obtém-se:

$$1.FF_h + j1.FF_h = 1.99609375_d + j1.99609375_d = 0001.11111111_b + j0001.11111111_b$$

Fazendo a FFT desse valor no Matlab[1], obtém-se:

$$127.75_d + j127.75_d = 01111111.11_b + j01111111.11_b$$

Dividindo por c temos:

$$21.33571904897690_d + j21.95194476842880_d$$
$$010101.010101011111000110101111_b + j010101.111100111011001010100111_b$$

Comparando com a saída da FFT implementada:

$$01010101011100_b + 01010111110000_b$$

Obtém-se para a parte real:

$010101.010101011111000110101111_b$	=	21.33571904897690_d	$(Matlab)$
010101.01011100_b	=	21.359375_d	$(FPGA)$
010101.01010110_b	=	21.3359375_d	$(Matlab\ arredondado)$

e para a parte imaginária:

$010101.111100111011001010100111_b$	=	21.95194476842880_d	$(Matlab)$
010101.11110000_b	=	21.9375_d	$(FPGA)$
010101.11110100_b	=	21.953125_d	$(Matlab\ arredondado)$

Concluindo-se então que o ponto fixo da entrada será na quarta casa e o ponto fixo da saída será na sexta casa.

[1]Fazendo todos os canais do modulador com esse valor e obedecendo a configuração da simetria hermetiana.

Agora podemos especificar o codificador, que já foi definido como um 4-QAM. Sabendo que esse poderá representar o seu valor máximo como $0001.11111111_b = 1.FF_h$ e o mínimo como $1110.00000001_b = E.01_h$, monta-se a seguinte tabela de conversão:

Bits de entrada	Ponto da constelação $(a + jb)_h$
00_b	$1.FF_h + j1.FF_h$
01_b	$1.FF_h + jE.01_h$
10_b	$E.01_h + j1.FF_h$
11_b	$E.01_h + jE.01_h$

Tabela 5: *Tabela de conversão 4-QAM implementada*

5.4 Taxas de transmissão

Para o OFDM planejado, de 31 canais modulados a 4-QAM por uma IFFT de 64 pontos, pode-se fazer a seguinte análise.

Sabe-se que para essa IFFT será necessário 3 estágios fazendo 64 cálculos[2] em cada, fazendo um total de 192 cálculos. Como o sistema está todo implementado em pipeline, ao final de 192 clocks já se começa a ter resposta. Portanto no mínimo cada frame OFDM terá a duração de 192 clocks/frame.

Porém como se tem que dividir o frame para os 31 canais, não se consegue uma conta exata, $192/31 = 6.1935$ clocks/canal, arredondando-se para cima, cada canal terá 7 clocks de duração, com isso a duração do frame aumenta para $7 \times 31 = 217$ clocks/frame.

Definindo como F a freqüência de trabalho da FPGA, a duração do clock será de $1/F$ segundo, portanto a duração do frame será de $217/F$ segundo/frame, ou $F/217$ frame/segundo. Como está sendo usado um modulador 4-QAM, tem-se 2 bits/canal e 31 canais/frame, resultando em 62 bits/frame. Portanto a taxa será de $62F/217$ bits por segundo.

Considerando um clock de 50MHz, a taxa de transmissão seria de 14.286Mbit/s. Porém foi considerado para o cálculo que o intervalo de guarda tem valor nulo. Essa taxa serve como uma referência para a taxa de transmissão máxima.

Pode-se generalizar essa equação e obter:

$$bF/n, \quad n \geq 7 \tag{5.1}$$

[2]Um cálculo para cada ponto da IFFT.

Onde b é o número de bits por canal, F é a freqüência de operação do FPGA e n é o número inteiro de clocks por canal.

5.5 Análise da Implementação

Para poder validar todo o sistema foi implementado um modem OFDM em um chip num sistema *back-to-back*, como ilustrado na Figura 39. Ele modula e demodula o sinal no próprio chip sem passar pelos conversores digital-analógico, analógico-digital.

Para o teste, foi implementado um amostrador-codificador de uma entrada serial. Esse amostrador foi desenvolvido com dois flip-flops, para poder se evitar a meta-estabilidade. Como o tempo de cada canal tem de ser maior que 7 clocks, o amostrador foi implementado para durar 8 clocks por canal. Leva-se 4 clock para amostrar e codificar o sinal em fase e mais 4 para amostrar e codificar o sinal em quadratura.

Após o dado ter sido totalmente mapeado, após 8 clocks, é então gravado em um endereço da memórida de entrada da IFFT que faz a simetria hermetiana e modula o sinal.

Foi desenvolvido também um sincronizador entre o transmissor e o receptor. Quando a IFFT terminar de processar, o sincronizador lê os dados e então os escreve na entrada do demodulador.

Ao final da demodulação, o decodificador lê a memória de saída e, como explicado, decodifica o sinal pegando o bit mais significativo, gerando-se assim o dado original. Após isso, é feita uma conversão paralelo-serial, com duração de 4 clocks para cada bit para ser enviada novamente à porta serial.

O sistema foi testado utilizando um clock de 50MHz. Como a amostragem é realizada a cada quatro intervalos de clocks, a taxa de amostragem será, de acordo com (5.1), de 12.5 Mega amostras por segundo [MA/s]. Com isso, a máxima taxa de entrada teórica será de 6.25MA/s.

Foi implementado e testado através da porta serial de um computador. O sistema não apresentou nenhum erro. Está para se fazer o teste usando o medidor de taxa de erro, porém existem algumas incompatibilidades que têm de ser corrigidas.

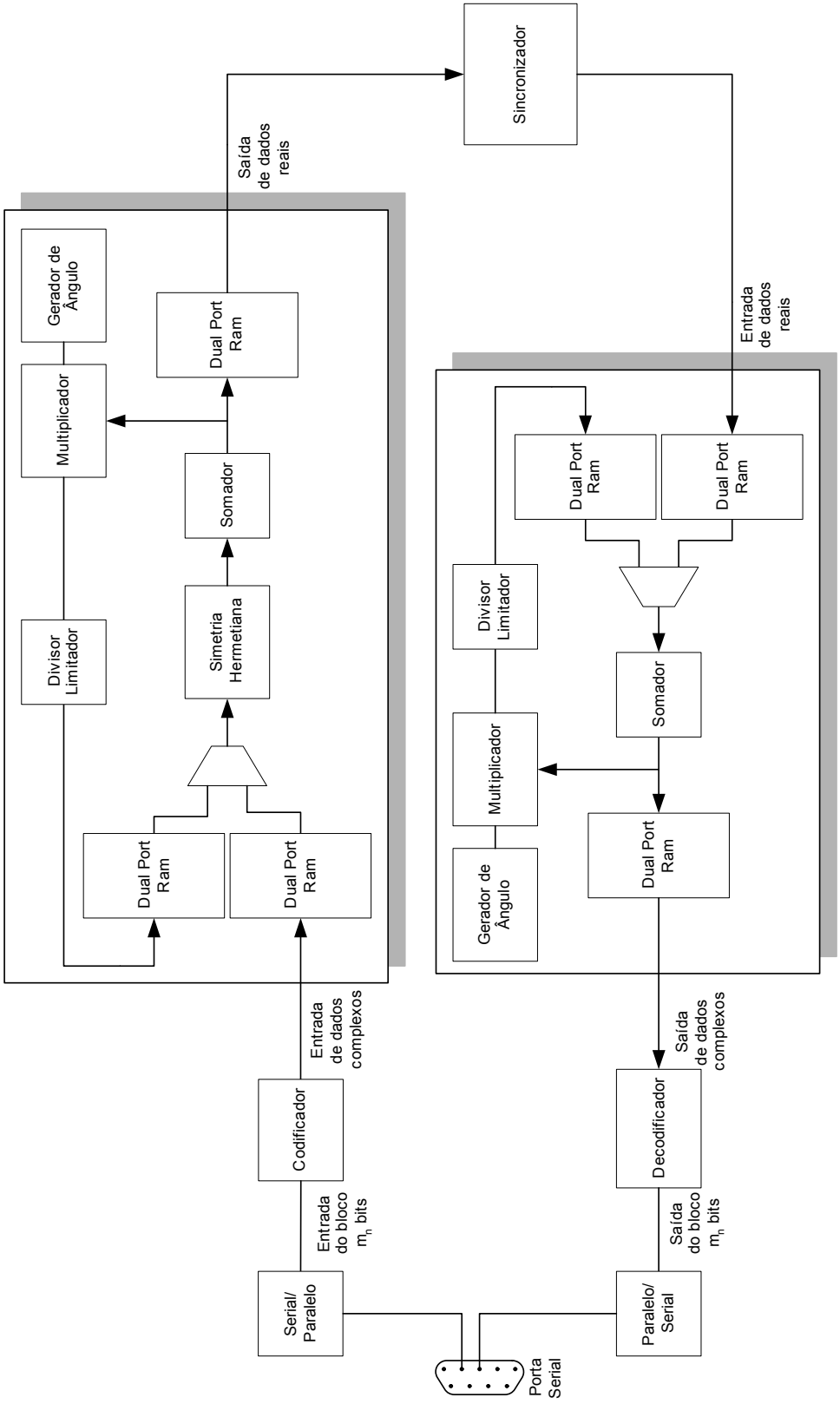


Figura 39: *O sistema OFDM testado*

5.6 Utilização do hardware

Nessa seção, um importante parâmetro no desenvolvimento do modem, a ocupação do sistema no FPGA, é analisada para o FPGA utilizado, a SPARTAN II XC2S200.

Será analisada a utilização do hardware do CORDIC, da parte de cálculo da FFT, da FFT completa, dos modems de transmissão e de recepção e do modem teste descrito na seção 5.5.

5.6.1 CORDIC

A Tabela 6 mostra a ocupação do CORDIC na FPGA e compara com um CORDIC equivalente gerado pelo COREGEN da Xilinx. O CORDIC desenvolvido utiliza muito menos hardware e possui uma freqüência máxima maior do que a do COREGEN. Portanto, o CORDIC desenvolvido é mais eficiente.

Parametros	Implementado	%	Coregen	%
Número de Slices	678	28	821	34
Número de Slice Flip Flops	435	9	1468	31
Freqüência máxima	103.348MHz		73.795MHz	

Tabela 6: *Comparação de utilização do hardware entre o CORDIC implementado e o gerado pelo COREGEN*

5.6.2 FFT

A parte que calcula a FFT foi divida em duas: uma que faz os cálculos e outra que faz o controle do sistema. A Tabela 7 mostra a utilização do hardware da parte que faz os cálculos. Como mostrado, nesse sistema está incluso o bloco do conjugado necessário para se fazer a simetria hermetiana.

A maior parte da FFT é ocupado pelo CORDIC, mais precisamente, 85% da FFT, e os outros 15% são utilizados para se fazer todo o cálculo restante da FFT.

5.6.3 FFT com controlador

Ao se interligar a parte que faz o cálculo com a parte de controle da FFT, percebe-se uma grande redução na freqüência máxima, como mostrado na Tabela 8. Isso se deve

Parâmetros	Implementado	%	Coregen	%
Número de Slices	1024	43	1622	68
Número de Slice Flip Flops	756	16	2564	54
Número de blocos de RAM	5	35	6	42
Freqüência máxima	103.767MHz		47.070MHz	

Tabela 7: *Utilização do hardware da parte de cálculo da FFT*

a grande complexidade do controlador do sistema. Será necessário uma reestruturação desse controlador para que se possa atingir maiores freqüência.

Parâmetros	Implementado	%	Coregen	%
Número de Slices	1059	43	1622	68
Número de Slice Flip Flops	799	16	2564	54
Número de blocos de RAM	5	35	6	42
Freqüência máxima	70.254MHz		47.070MHz	

Tabela 8: *Comparação de utilização do hardware entre o FFT implementado e o gerado pelo COREGEN*

5.6.4 Modems OFDM de transmissão e recepção

A Tabela 9 mostra a utilização do hardware pelos modems de transmissão e de recepção. Nesses modems estão incluídos a FFT com a simetria hermetiana, o codificador e o decodificador, o conversor serial-paralelo e o paralelo-serial.

Parâmetros	TX	%	RX	%
Número de Slices	1086	46	1059	45
Número de Slice Flip Flops	828	17	810	17
Número de blocos de RAM	5	35	5	35
Freqüência máxima	70.254MHz		70.701MHz	

Tabela 9: *Utilização do hardware dos modems OFDM de transmissão e recepção*

5.6.5 Modem de teste

A Tabela 10 mostra a utilização pelo sistema de teste mostrado na Figura 39. Salienta-se a utilização de 92% do hardware. Além de, após várias otimizações, conseguir uma freqüência máxima de operação de 93.529MHz.

Parâmetros	Utilização	%
Número de Slices	2180	92
Número de Slice Flip Flops	1716	36
Número de blocos de RAM	10	71
Freqüência máxima	93.529MHz	

Tabela 10: *Utilização do hardware dos modem utilizado para teste*

5.6.6 Floorplan do modem teste

A Figura 40 mostra a disposição do modem teste no FPGA SPARTAN II. A metade superior da FPGA corresponde ao modem de transmissão, a outra metade corresponde ao modem de recepção. No canto esquerdo ao meio está o bloco sincronizador entre o transmissor e o receptor. O sistema foi assim distribuído para tentar atingir uma maior freqüência de operação, porém não foi percebida qualquer melhoria.

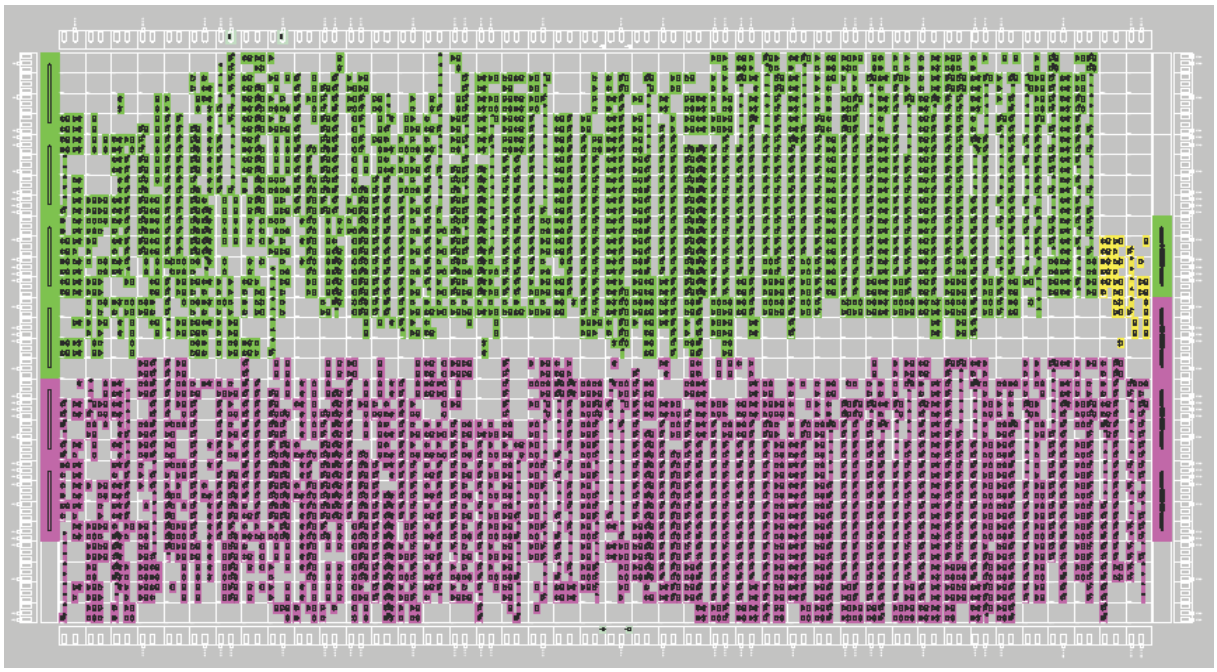


Figura 40: *Disposição em hardware do modem teste*

6 Conclusões

Com o desenvolvimento tanto de técnicas de transmissão digital de dados como dos atuais FPGAs tem tornado realidade a possibilidade de se fazer sistemas de transmissão digital integrados em um único chip. Para fazer uso dessas modernas tecnologias foi desenvolvido a estrutura completa para o transmissor e para o receptor OFDM, que utilizam a (I)FFT para fazer a modulação. Foram analisadas várias formas de se implementar esse modem OFDM em uma FPGA, e escolhidas as mais eficientes e versáteis.

Como contribuição, o presente trabalho, fornece um código em VHDL completo e funcional para um modem OFDM com 31 subportadoras, utilizando uma FFT de 64 pontos, radix-4 com CORDIC, a uma taxa de 12.5 Mbit/s.

Para isso foram desenvolvidos e analisados os vários blocos funcionais desse modem. Mostrou-se e comparou-se a ocupação em hardware das várias partes que o compõem. Chegando-se a conclusão de que as partes principais desenvolvidas para o sistema são mais eficiente do que o disponibilizado pelo fabricante do FPGA, a Xilinx, chegando a ter um economia de hardware de até 25%.

Dessa forma, é possível implementar esse modem em um FPGA extremamente barato como a SPARTAN II, ao invés de se utilizar grandes e caras FPGAs como os da família VIRTEX.

Nessa análise, uma das partes que mais se destaca é o controlador que organiza e manipula os dados para que se faça a modulação de forma correta. Ele, por ser extremamente complexo, não obteve um bom desempenho, tendo de ser reanalisado para que se possa funcionar em uma freqüência maior.

Além disso, há muito ainda a ser feito. Pode-se incluir vários tipos de codificação para controle de erro tanto em blocos como convolucionais, principalmente CRC, Reed-Solomon. Pode-se substituir o codificar da constelação por um TCM (*Trellis-Coded Modulation*), dando um ganho de até 7dB ao transmissor. Ao incluir o TCM, é necessário que se faça o decodificador do TCM, o principal deles é o Viterbi.

Referências

1 BROWN, S.; ROSE, J. **Architecture of FPGAs and CPLDs: A Tutorial**.

2 PARNELL, K.; MEHTA, N. **Programmable Logic Design Quick Start Hand Book**. 2. ed. Xilinx, 2002. Disponível em: <www.xilinx.com>.

3 SOCIETY, I. C. **IEEE Standard 1076: VHDL Language Reference Manual**. Nova York, EUA: The Institute of Electrical and Electronics Engineers, Inc., Maio 2002.

4 HIROSAKI, B. An orthogonally multiplexed qam system using the discrete fourier transform. **IEEE Transactions on Communications**, v. 28, p. 982–989, Julho 1981.

5 SALTZBERG, B. R. Performance of an efficient parallel data transmission system. **IEEE Transactions on Communications**, COM-15, p. 805–811, Dezembro 1967.

6 BINGHAM, J. A. C. **ADSL, VDSL and Multicarrier Modulation**. 1. ed. [S.l.]: John Wiley & Sons, Inc., 2000.

7 CARVALHO, R. M. **Princípios de comunicações**. 3. ed. Vitória: [s.n.], 2001.

8 DOELZ, M. L.; HEALD, E. T.; MARTIN, D. L. Binary data transmission techniques for linear systems. **Proc. IRE**, p. 656–661, 1957.

9 BINGHAM, J. A. C. Multicarrier modulation for data transmission: An idea whose time has come. **IEEE Communication Magazine**, p. 5–14, Maio 1990.

10 CHANG, R. W. High-speed multichannel data transmission with bandlimited orthogonal signals. **Bell System Technology Journal**, v. 45, p. 1775–1796, December 1966.

11 HANZO, L.; WEBB, W.; KELER, T. **Single- and Multi-carrier Quadrature Amplitude Modulation: principles and applications for personal communications, WLANs and broadcasting**. 1. ed. West Sussex, Inglaterra: John Wiley & Sons, Inc., 2000.

12 WEINSTEIN, S. B.; EBERT, P. M. Data trasmission by frequency-division multiplexing using the discrete fourier transform. **IEEE Transactions on Communications Technology**, v. 19, n. 5, p. 628–634, Outubro 1971.

13 PELED, A.; RUIZ, A. Frequency domain data trasmission using reduced computational complexity algorithms. **International Conference on Acoustics, Speech, and Signal Processing**, p. 964–967, Abril 1980.

14 XILINX. **The Spartan-II Family Data Sheet**. Novembro 2001.

15 PROAKIS, J. G.; MANOLAKIS, D. G. **Digital Signal Processing: Principles, Algorithms and Applications**. 3. ed. Nova Jersey, EUA: Prentice Hall, 1996.

16 BAESE, U. M. **Digital Signal Processing with Field Programmable Gate Arrays**. Berlin, Germany: Springer, 2001.

17 HERVEILLE, R. **Cordic Core Specification**. Dezembro 2001. Disponível em: <www.opencores.org>.

18 ANDRAKA, R. **A survey of cordic algorithms for fpga based computers**. 1998. Disponível em: <citeseer.ist.psu.edu/andraka98survey.html>.