

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
DEPARTAMENTO ACADÊMICO DE ELÉTRICA  
CURSO DE ENGENHARIA ELÉTRICA

CALLEBE SOARES BARBOSA

**IMPLEMENTAÇÃO DO ALGORITMO RADIX-2 PARA  
CÁLCULO DA FFT EM FPGA**

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO

2018

CALLEBE SOARES BARBOSA

## **IMPLEMENTAÇÃO DO ALGORITMO RADIX-2 PARA CÁLCULO DA FFT EM FPGA**

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia Elétrica da Coordenação de Engenharia Elétrica - CO-ELT - da Universidade Tecnológica Federal do Paraná - UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Fábio Luiz Bertotti

PATO BRANCO

2018

## **TERMO DE APROVAÇÃO**

O Trabalho de Conclusão de Curso intitulado **IMPLEMENTAÇÃO DO ALGORITMO RADIX-2 PARA CÁLCULO DA FFT EM FPGA** do acadêmico **Callebe Soares Barbosa** foi considerado **APROVADO** de acordo com a ata da banca examinadora **Nº 123 de 2018.**

Fizeram parte da banca examinadora os professores:

**Prof. Dr. Fábio Luiz Bertotti**

**Prof. Dr. Stephen Hawking**

**Dr. Brian Greene**

**Prof. Dr. Michio Kaku**

**Prof. Richard Feynman**

Aqui vai o texto da dedicatória. Aqui vai o texto da dedicatória. Aqui vai o texto da dedicatória.

*Acreditar é mais fácil do que pensar. Daí existem muito mais crentes do que pensadores.*

Bruce Calvert

## **AGRADECIMENTOS**

Aqui são os agradecimentos.

## **RESUMO**

Escreva aqui o texto de seu resumo... UTFPR<sup>T</sup><sub>E</sub>X

**Palavras-chave:** L<sup>A</sup>T<sub>E</sub>X, FFT, Transformada Rápida de Fourier, Sistemas Digitais, FPGA, Zyn.

## **ABSTRACT**

Write here the English version of your Resumo...

**Keywords:** L<sup>A</sup>T<sub>E</sub>X, FFT, Fast Fourier Transform, Digital System, FPGA, Zynq, Digital Signal Process.

## LISTA DE FIGURAS

Figura 1:	Batimento Cardíaco em 3D . . . . .	16
Figura 2:	Índice BM & FBOVESPA, São Paulo - Brasil . . . . .	16
Figura 3:	Temperatura de um Volume de Água em um Ebulidor Controlado . . . . .	17
Figura 4:	Modos Normais de uma Corda Vibrante . . . . .	18
Figura 5:	Ilustração da Propriedade de Autofunção de Sistemas Lineares . . . . .	20
Figura 6:	Aproximação do Sinal de Onda Quadrada por um Termo Senoidal . . . . .	21
Figura 7:	Aproximação do Sinal de Onda Quadrada por Soma de 3 Termos Senoidais . . . . .	22
Figura 8:	Aproximação do Sinal de Onda Quadrada por Soma de 8 Termos Senoidais . . . . .	22
Figura 9:	Aproximação do Sinal de Onda Quadrada por Soma de 8 Termos Senoidais . . . . .	26
Figura 10:	Butterfly do Fluxo do Sinal . . . . .	31
Figura 11:	FFT de 8 Pontos . . . . .	32
Figura 12:	Rotação de $H_r$ pelo ângulo de $2\pi r/N_0$ . . . . .	34
Figura 13:	Ângulos Elementares - Cordic Tradicional com N=4 Fonte: . . . . .	38
Figura 14:	EEAS com N=2 e S=4 . . . . .	41
Figura 15:	Exemplo de execução do Algoritmo TBS . . . . .	44
Figura 16:	Exemplo de execução do Algoritmo TBS . . . . .	45
Figura 17:	MSR com I=2, J=1 e N=2 Fonte: (LIN; WU, 2005) . . . . .	48
Figura 18:	MSR com I=2, J=1 e N=2 Fonte: (LIN; WU, 2005) . . . . .	49
Figura 19:	Arquitetura Tipica de uma FPGA . . . . .	51
Figura 20:	Arquitetura de uma CLB com 4 BLEs . . . . .	53
Figura 21:	Arquitetura de uma BLE ( <i>Basic Logic Element</i> ) . . . . .	54
Figura 22:	Diagrama Lógico Full Adder 4 Bits - ISE Design Suite 14 . . . . .	55

Figura 23: CPU MicroBlaze e Coprocessador para FFT com Interface UART - Vivado 2017.4	57
Figura 24: ZynqBery - TE0726	58
Figura 25: Arquitetura Simplificada - Zynq-7000	60
Figura 26: Arquitetura Simplificada de um Sistema Digital	61
Figura 27: Arquitetura Simplificado do um Sistema Digital Mapeado para o Zynq	62
Figura 28: Visão Geralda Arquitetura - Zynq 7000	63
Figura 29: Arquitetura Implementada FFT de 16 Pontos	65
Figura 30: Arquitetura Implementada FFT de 16 Pontos	66

## **LISTA DE TABELAS**

Tabela 1: Relação entre propriedade do tempo de um sinal e a representação de Fourier adequada . . . . .	23
Tabela 2: Matriz $\phi$ para o dado Exemplo . . . . .	45
Tabela 3: Exemplo de Tabela . . . . .	67

## **LISTA DE SÍMBOLOS**

$\phi$

$\vec{\alpha}$  alpha

$v\omega\psi_{n-1}^{jk}$  Função de teste da lista de símbolos. Está é uma descrição longa para um único símbolo

$\nabla$  Gradiente

$v\omega\psi_{n-1}^{jk}$  teste

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>13</b>
1.1 JUSTIFICATIVA .....	13
<b>2 REVISÃO DE LITERATURA .....</b>	<b>15</b>
2.1 REPRESENTAÇÃO DE FOURIER PARA SINAIS .....	17
2.1.1 Resposta do Sistemas LTI a Entrada Senoidal .....	19
2.1.2 Série de Fourier .....	24
2.1.3 Espectro de Fourier .....	25
2.2 SÉRIE DE FOURIER EM TEMPO DISCRETO .....	26
2.3 TRANSFORMADA RÁPIDA DE FOURIER.....	29
2.4 ALGORITMO CORDIC.....	33
2.4.1 CORDIC Tradicional .....	35
2.4.2 EEAS-CORDIC .....	37
2.4.2.1 Algoritmo TBS.....	41
2.4.3 MSR-CORDIC .....	46
2.5 FPGA.....	50
2.5.1 Aspectos Construtivos da FPGA.....	51
2.5.2 Programação na FPGA .....	55
2.5.3 ZynqBerry TE0726-03M .....	58
2.5.3.1 Zynq-7000 .....	59
<b>3 IMPLEMENTANDO PROCESSADOR CORDIC .....</b>	<b>64</b>
<b>4 IMPLEMENTANDO A FFT NO ZYNQBERRY .....</b>	<b>65</b>
4.0.1 Implementando FFT 32 Pontos .....	65
4.0.2 Implementando FFT 1024 Pontos.....	65
<b>5 ANÁLISE DOS RESULTADOS .....</b>	<b>66</b>
5.0.1 Implementando FFT 32 Pontos .....	66
5.0.2 Implementando FFT 1024 Pontos.....	66

<b>6 CONCLUSÃO .....</b>	<b>67</b>
6.0.1 Testes e exemplos de lista de siglas .....	67
6.0.2 Testes e exemplos de lista de símbolos .....	67
6.0.3 Exemplos de citação de referências bibliográficas.....	68
<b>7 NÍVEL 1 - TESTES DE CAPITULAÇÃO - PRIMÁRIO .....</b>	<b>69</b>
7.1 NÍVEL 2 - SECUNDÁRIO .....	69
7.1.1 Nível 3 - Terciário .....	69
7.1.1.1 Nível 4 - Quaternário .....	69
<b>ANEXO A - STELLAR MYSTERY SOLVED, EINSTEIN SAFE .....</b>	<b>73</b>
A.1 EXEMPLO DE SEÇÃO ANEXO .....	74

## 1 INTRODUÇÃO

A Transformada Discreta de Fourier ou DFT (*Discrete Fourier transform*), segundo Bingham (1990), é um recurso amplamente utilizado em aplicações como processamento de imagens, comunicação rede de área local sem-fio ou WLAN (*Wireless Local Area Network*), multiplexação por divisão de frequências ortogonais ou OFDM (Orthogonal Frequency Division Multiplexing), e medições de diferentes espectros.

O cálculo da DFT é uma tarefa que exige muitos recursos computacionais e que requer um projeto preciso para uma implementação eficiente (WANG *et al.*, 2010). A DFT tem como base a própria série trigonométrica de Fourier discretizada. Segundo Lathi (2007, p. 719), graças a um algoritmo desenvolvido por Cooley e Tukey o número de cálculos para executar uma DFT foi drasticamente reduzido, possibilitando um tempo de execução aceitável.

Segundo Zhou *et al.* (2009) os dispositivos conhecidos como FPGA (*Field Programmable Gate Array*) são cada vez mais usados em implementação de hardware para telecomunicações, por exemplo, devido a sua capacidade de alcançar um elevado desempenho, aliado a sua flexibilidade de configuração. Desta forma, o uso de FPGA para implementar um hardware específico para o cálculo da DFT torna-se relevante.

### 1.1 JUSTIFICATIVA

Como afirma He e Guo (2008), o algoritmo da FFT é utilizada em larga escala como componente chave em sistemas de processamentos de sinais. De tal forma que a FFT não é somente usada em aplicações de telecomunicações e processamento de dados audiovisuais (BINGHAM, 1990), mas é o algoritmo numérico mais comum em diversas áreas, como engenharia, medicina, física e matemática (VANMATHI *et al.*, 2014).

No campo da engenharia biomédica, a FFT tem sido empregada na avaliação de parâmetros biológicos, como a bioimpedância (AMARAL *et al.*, 2011). Segundo Martinsen e Grimnes (2011), a análise da bioimpedância já é usada para monitorar atividades bioelétricas cerebrais, diagnosticar câncer de pele, dermatite, hiperidrose, ava-

iliar a composição corporal, avaliar condição nutricional, detectar processo de rejeição de órgãos transplantados e ainda monitorar recém-nascidos através tomografia de impedância elétrica ou EIT (*Electrical Impedance Tomography*). Sendo a EIT a única técnica de obtenção de imagens que não afeta o sistema imunológico de recém-nascidos (TRIANTIS *et al.*, 2011).

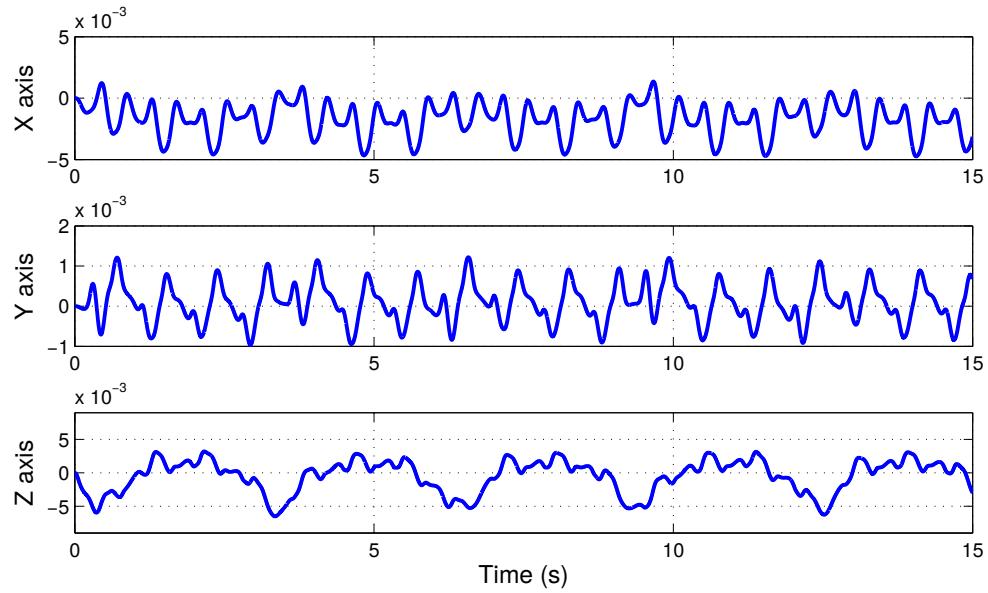
Para Ibrahim *et al.* (2016), a FPGA é uma boa escolha para a implementação do algoritmo da FFT devido a grande variedade de recursos de *hardware* sintetizáveis, além de possuir recursos de programação paralela que permite o processamento paralelo de sinais, conferindo assim uma maior rapidez na execução do algoritmo. Portanto implementar a FFT, uma ferramenta tão útil em diversas áreas da ciência, em uma plataforma vantajosa como a FPGA, é o principal motivador deste trabalho.

## 2 REVISÃO DE LITERATURA

Nos mais diversos campos da ciência os conceitos de sinais e sistemas possuem um papel importante, sendo utilizados nas aplicações mais variadas como acústica, sismologia, projeto de circuitos, sistemas de geração e distribuição de energia, controle de processos químicos e engenharia biomédica, entre outros. Indiferentemente da natureza física de uma aplicação, os sinais e sistemas que surgem destas possuem duas características básicas; os sinais contém informações sobre a natureza ou comportamento de um fenômeno, e os sistemas respondem a um sinal específico produzindo um comportamento desejado (OPPENHEIM; WILLSKY, 2010, Prólogo).

Um sinal é um conjunto de dados ou informações, que podem descrever diversos tipos de fenômenos, como o batimento cardíaco (Figura 1), a temperatura de um volume de aguá em um ebulidor controlado (Figura 3), o registro de vendas de uma empresa ou ainda os valores de fechamento de uma bolsa de valores (Figura 2). Já os sistemas são entidades que processam conjuntos de sinais (ditos entrada), e geram outro conjunto de sinais como resposta (ditos saídas). Os sistemas podem ser constituídos de componentes físicos, elétricos, mecânicos, hidráulicos ou apenas lógicos (LATHI, 2007, p. 75). O filtro de áudio, que modifica o sinal de entrada gerando um sinal de áudio com as características desejadas é um exemplo de sistema. Assim como o ebulidor controlado que a partir de um sinal de referência (entrada) gera um sinal de controle para o atuador da resistência de aquecimento, de modo a elevar a temperatura do líquido até a referência.

Segundo Oppenheim e Willsky (2010, p. 1) existe uma linguagem adequada para descrever sinais e um conjunto poderoso de ferramentas para analisá-los, capaz de se aplicar a problemas oriundos de diversos domínios. A série de Fourier e a Transformada Rápida de Fourier são ambas exemplos destas ferramentas, e serão apresentadas nos próximos capítulos.



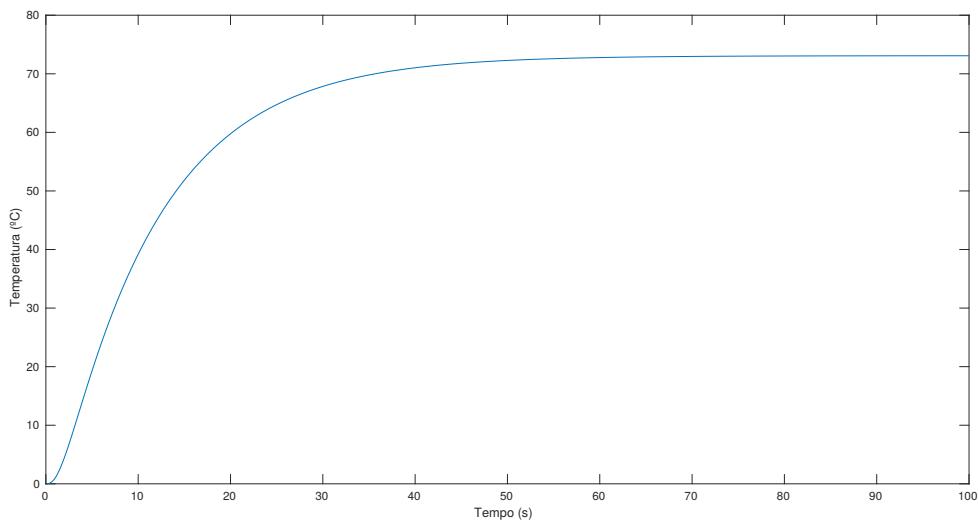
**Figura 1: Batimento Cardíaco em 3D**

Fonte: Liu *et al.* (2011)



**Figura 2: Índice BM & FBOVESPA, São Paulo - Brasil**

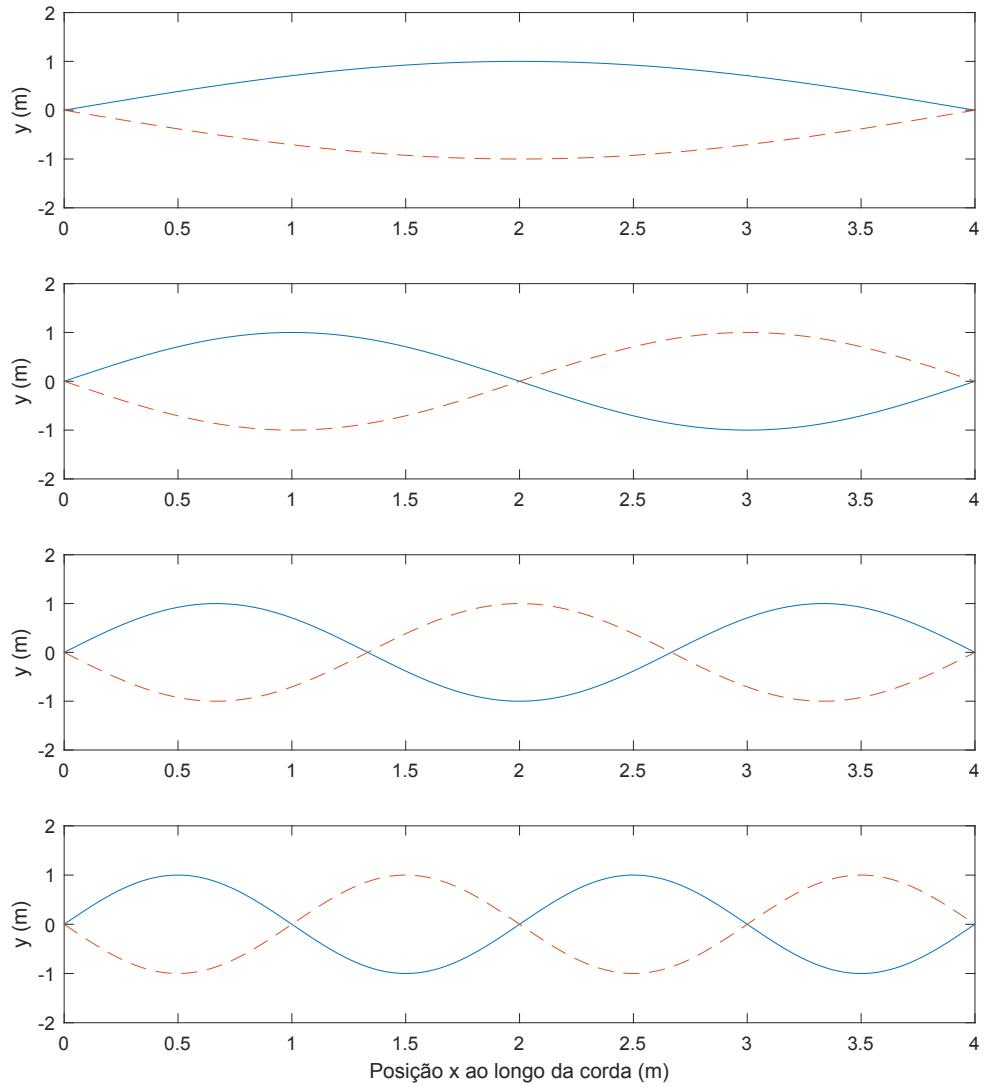
Fonte: TradingView (2017)



**Figura 3: Temperatura de um Volume de Água em um Ebulidior Controlado**  
Fonte: Autoria Própria

## 2.1 REPRESENTAÇÃO DE FOURIER PARA SINAIS

A série de Fourier tem como princípio os estudos das somas trigonométricas de senos e cossenos harmonicamente relacionados, com o intuito de descrever fenômenos periódicos. Tal estudo possui uma longa história que data pelo menos da época dos babilônicos, e que envolve o estudo de diferentes fenômenos físicos. Mas o marco moderno neste tema ocorre em 1748 com o matemático e físico suíço Leonhard Paul Euler (OPPENHEIM; WILLSKY, 2010, p. 104). Euler em seu estudo sobre ondas estacionárias através de cordas vibrantes observou que se a configuração da posição vertical  $y_0$  de um ponto horizontal  $x$  em uma onda estacionaria no tempo  $t_0$  for uma combinação linear dos modos normais da onda, o mesmo acontece com a configuração em qualquer valor de tempo  $t_s$  subsequente. Com base nesse estudo Euler demonstrou que é possível calcular diretamente os coeficientes da combinação linear em tempos futuros usando os coeficientes em tempos anteriores. (OPPENHEIM; WILLSKY, 2010, p. 104).



**Figura 4: Modos Normais de uma Corda Vibrante**  
Fonte: Adaptado de Oppenheim e Willsky (2010, p. 105)

O estudo de Euler se tornam ainda mais importantes quando aplicado a sinas e a sistemas LIT. Segundo HAYKIN e Veen (2001, p. 163), se a entrada de um sistema Linear Invariante no Tempo (LIT) for expressa por uma combinação linear ponderada de senóides ou exponenciais complexas, a saída do sistema será expressa como uma combinação linear ponderada da resposta do sistema a cada senóide ou exponencial complexa. Expressar sinais em termo de senoides ou exponenciais complexas não apenas leva a uma expressão alternativa útil para o comportamento da entrada e saída de um sistema LTI, como também fornece uma caracterização muito criteriosa dos sinais e sistemas.

Segundo Oppenheim e Willsky (2010, p. 105), meio século depois da divulgação do trabalho de Euler, o físico e matemático francês Jean-Baptiste Joseph Fourier (1768

- 1830), havia se envolvido no estudo sobre séries trigonométricas, com a motivação física de estudar o fenômeno da propagação e difusão de calor. Fourier conclui que séries senoidais harmonicamente relacionadas eram úteis na representação da distribuição de temperatura em um corpo, e que 'qualquer' sinal periódico poderia ser representado por tal série. Fourier ainda apresentou uma representação para sinais aperiódicos, não através de somas ponderadas de senoides harmonicamente relacionadas, mas como integrais ponderadas de senoides que não são necessariamente harmonicamente relacionadas(HAYKIN; VEEN, 2001, p. 163).

Como afirma Oppenheim e Willsky (2010, p. 106), muitas das ideias básicas por trás das contribuições de Fourier já eram conhecidas, e as condições precisas sob as quais a representação de sinais proposta era válida só foram apresentadas por P.L. Dirichlet em 1829. Porém foi Fourier que teve a clara percepção do potencial pra essa representação, e até certo ponto foi o seu trabalho e suas afirmações que estimularam grande parte do trabalho subsequente. Logo em sua homenagem o estudo de sinais e sistemas, usando representações senoidais, é denominado análise de Fourier. E as séries pelo qual é realizada a representação de sinais na forma de somas de senoides complexas é denominada série de Fourier.

### 2.1.1 RESPOSTA DO SISTEMAS LTI A ENTRADA SENOIDAL

Na análise de Fourier, os sinais de entrada senoidais são comumente usados para caracterizar a resposta de um sistema Linear e Invariante no Tempo, ou LTI (Linear Time-Invariant). A resposta senoidal em estado estacionário de um sistema LTI é obtido pela convolução entre a entrada senoidal e o sinal de impulso(HAYKIN; VEEN, 2001, p. 163).

Ao aplicar um sinal impulso ( $\delta(t)$ ) a entrada de um sistema LTI, é gerado um sinal de saída conhecido como resposta ao impulso  $\delta(t)$ . Através da resposta ao impulso é possível caracterizar de maneira completa o comportamento de um sistema. A resposta ao impulso também possibilita conhecer a resposta do sistema LTI a qualquer sinal de entrada, através da convolução deste sinal ao impulso (HAYKIN; VEEN, 2001, p. 108).

Assim realizando a convolução do impulso ao sinal senoidal, segundo HAYKIN e Veen (2001, p. 164), a saída de um sistema LTI dado uma entrada senoidal complexa  $x(t)$ , na forma exponencial  $e^{j\omega t}$ , é dada por:

$$y(t) = H(j\omega)e^{j\omega t} \quad (1)$$

Em que  $H(j\omega)$  é a resposta em frequência, definida em termos de resposta ao impulso  $\delta(t)$ . Assim;

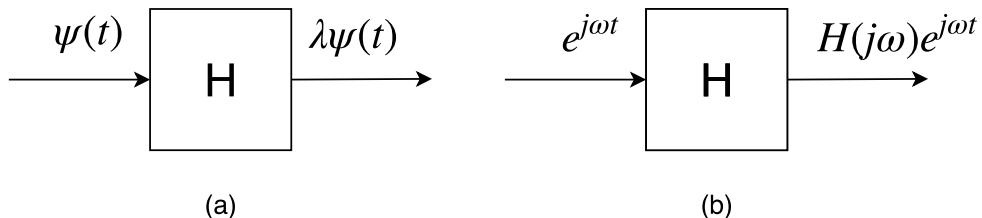
$$H(j\omega) = \int_{-\infty}^{\infty} \delta(t)e^{-j\omega t} dt \quad (2)$$

Logo a entrada senoidal complexa em um sistema LTI gera uma saída igual a entrada senoidal multiplicada apenas pela resposta em frequência do sistema  $H(j\omega)$ .

As equações (1) e (2) apenas consideram como entrada um sinal senoidal. Porém é de interesse obter uma expressão para a resposta do sistema LTI a quaisquer sinais arbitrários. Para tal HAYKIN e Veen (2001, p. 164) considera a senoide complexa  $\psi = e^{j\omega t}$  como uma autofunção do sistema  $H$  associando com o autovalor  $\lambda = H(j\omega)$ , de modo a satisfazer:

$$H\|\psi\| = \lambda\psi(t) \quad (3)$$

Como pode ser visto na Figura (5), a saída de um sistema dada a entrada de uma autofunção, é o produto da entrada por um número complexo. Se  $e_k$  for um autovetor de uma matriz  $A$ , e  $\lambda_k$  os autovalores associados a esta matriz, a autorrelação do problema tradicional do autovalor matricial é aplicável,  $Ae_k = \lambda_k e_k$  (HAYKIN; VEEN, 2001, p. 164).



**Figura 5: Ilustração da Propriedade de Autofunção de Sistemas Lineares**

(a) Autofunção geral  $\psi(t)$  e autovalor  $\lambda$ .

(b) Autofunção senoidal complexa  $e^{j\omega t}$  e autovalor  $H(j\omega)$ .

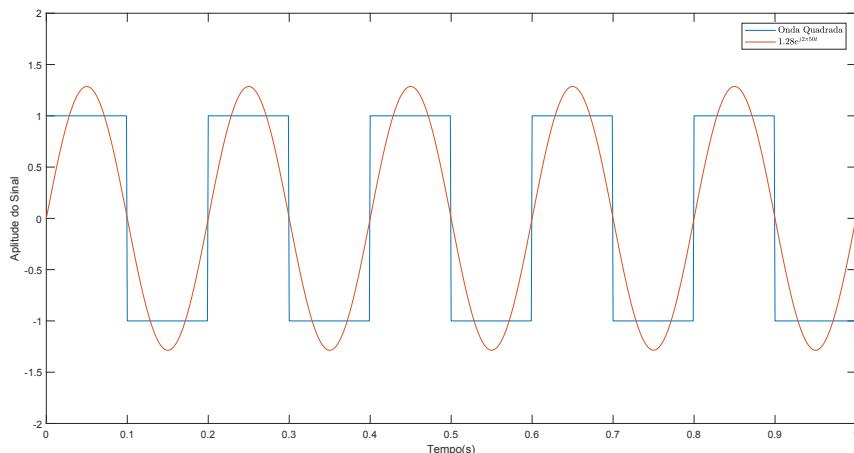
Fonte: Adaptado de HAYKIN e Veen (2001, p. 164)

A ideia principal aqui é utilizar a superposição ponderada de autofunções para representar um único sinal periódico. Para esse efeito HAYKIN e Veen (2001, p. 164) expressa a entrada de um sistema LTI como uma soma de  $N$  senoides complexas ponderadas, na forma:

$$x(t) = \sum_{k=1}^N a_k e^{j\omega_k t} \quad (4)$$

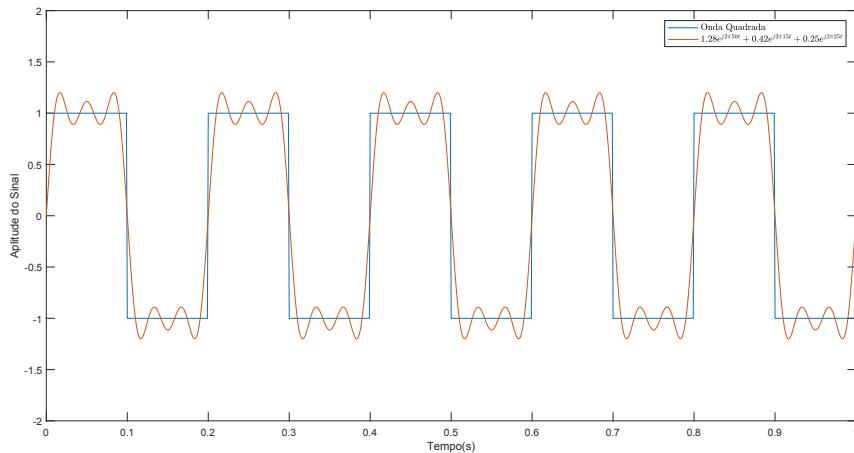
Ou seja na forma de série de Fourier.

Expressar a entrada de um sistema LTI a partir de uma soma ponderada de senoides complexas possui a intenção de realizar uma aproximação coerente do sinal de entrada, utilizando uma composição de funções básicas já bem conhecidas. Por exemplo considere o sinal de onda quadrada presente na Figura (6), o qual deseja-se aproximar utilizando uma soma de senoides complexas. Tomando uma senoide de amplitude 1.286 e uma frequência de  $50Hz$  é possível realizar uma aproximação grosseira, porém em fase com este sinal.

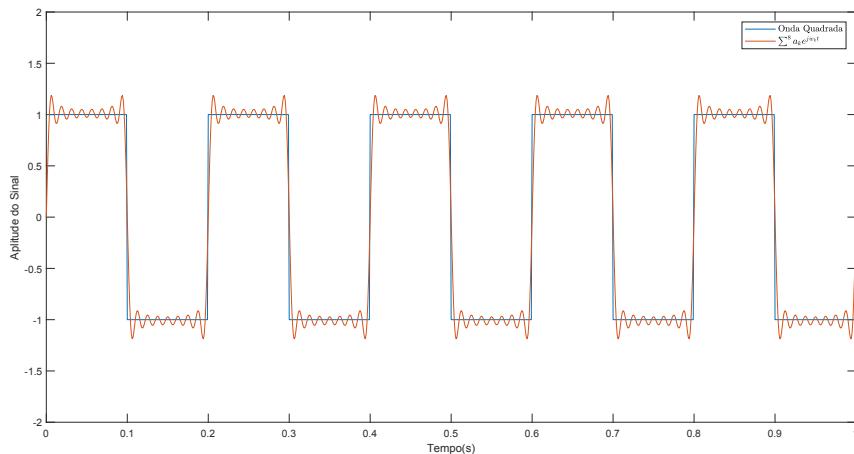


**Figura 6: Aproximação do Sinal de Onda Quadrada por um Termo Senoidal**  
**Fonte:** Autoria Própria

Para tornar melhorar a aproximação da representação em relação ao sinal de onda quadrada é possível adicionado mais termos senoidais ao somatório. Como pode ser visto nas Figura (7) e (8) as aproximações ficam mais suaves adicionando mais 2 ou mais 5 termos respectivamente. Quanto maior o número de termos senoidais complexas ponderados presentes no somatório da representação maior é a aproximação, sendo no limite perfeita.



**Figura 7: Aproximação do Sinal de Onda Quadrada por Soma de 3 Termos Senoidais**  
**Fonte:** Autoria Própria



**Figura 8: Aproximação do Sinal de Onda Quadrada por Soma de 8 Termos Senoidais**  
**Fonte:** Autoria Própria

Logo utilizando-se  $e^{j\omega_k t}$  for uma autofunção e  $H(j\omega_k)$  for o autovalor do sistema, aplicando-se a autorrelação apresentada anteriormente, o sinal de saída do sistema é dado por:

$$y(t) = \sum_{k=1}^N a_k H(j\omega_k) e^{j\omega_k t} \quad (5)$$

A saída, portanto nada mais é do que a soma ponderada das senoides complexas da entrada, sendo os pesos  $a_k$  ponderados pela resposta em frequência  $H(j\omega_k)$ . Por meio destes resultados é possível transformar a operação de convolução em uma operação de multiplicação dos termos  $a_k H(j\omega_k)$ . Segundo Oppenheim e Willsky (2010, p. 164) o fato da saída de um sistema LTI dada uma entrada represen-

tada como combinação linear de senoides complexas, ser também uma combinação linear dos mesmos sinais, foi uma descoberta de Euler, que motivou Fourier e os outros matemáticos após ele no estudo da extensão de classes de sinais que poderiam ser representados nesta forma de somatórios de exponenciais complexas ponderadas.

Além de tornar mais prático o cálculo da convolução de sinais, a representação em somais senoidais complexas ponderadas fornece uma interpretação alternativas para sinais e sistemas HAYKIN e Veen (2001, p. 166). Através da análise dos pesos  $a_k$  ponderados é possível descrever um sinal em função da frequência, ao invés do tempo.

A representação de sinais por séries de Fourier pode ser aplicada para diferentes tipos de sinais, com diferentes características. Há quatro classes de representações de Fourier, divididas de acordo com a sua periodicidade e sua continuidade, como pode ser visto na Tabela 1.

Para sinais periódicos a representação é feita como séries de Fourier, sendo que para sinais de tempo contínuo é aplicado a Série de Fourier (FS - *Fourier Series*), e para sinais de tempo discreto é usada as séries de Fourier de tempo discreto (DFS - *Discrete Fourier Serie*). Quando os sinais não são periódicos a representação é denominada como transformada, para o caso do sinal ser contínuo a representação é feita pela transformada de Fourier (FT - *Fourier Transform*), e no caso discreto é a transformada de Fourier de tempo discreto (DFT - *Discrete Fourier Trasform*).

Propriedade do Tempo	Periódico	Não Periódico
Continuo	Série de Fourier (FS)	Transformada de Fourier (FT)
Discreto	Série de Fourier de Tempo Discreto (DFS)	Transformada de Fourier de Tempo Discreto (DFT)

**Tabela 1: Relação entre propriedade do tempo de um sinal e a representação de Fourier adequada**

**Fonte:** Oppenheim e Willsky (2010, p. 166)

A representação de Fourier utilizada no desenvolvimento deste trabalho é a DFT, e portanto a mais importante a ser apresentada. As próximas seções deste trabalho passaram pela apresentação da FS depois pela DFS para em fim chegar na DFT.

### 2.1.2 SÉRIE DE FOURIER

Segundo Lathi (2007, p. 530) "Um sinal periódico  $x(t)$  com período  $T_0$  pode ser descrito como a soma de senoides de frequência  $f_0$  e todas as suas harmônicas", conforme apresentado na Equação (6). Esta é a chamada série de Fourier para sinais periódicos, ou apenas série de Fourier. Na expressão da Equação (6) a série de Fourier está na forma trigonométrica. Onde  $\omega_0$  é a frequência fundamental de  $x(t)$ , e  $a_0$ ,  $a_n$  e  $b_n$  são os coeficientes de amplitude das harmônicas que compõe  $x(t)$ , sendo  $a_0$  a harmônica zero (nível CC).

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t) \quad (6)$$

Os coeficientes  $a_0$ ,  $a_n$  e  $b_n$  da Equação (6) são determinados pelas seguintes equações:

$$a_0 = \frac{1}{T_0} \int_{T_0} x(t) dt \quad (7)$$

$$a_n = \frac{2}{T_0} \int_{T_0} x(t) \cos(n\omega_0 t) dt \quad (8)$$

$$b_n = \frac{2}{T_0} \int_{T_0} x(t) \sin(n\omega_0 t) dt \quad (9)$$

em que  $T_0$  representa o período relativo a frequência fundamental  $f_0$ .

A série de Fourier, além da forma trigonométrica, também pode ser apresentada na forma exponencial, em termos de  $e^{j\omega_0 t}$ , como apresentado na seção anterior. A forma exponencial, segundo Lathi (2007, p. 533), é dada através da Equação (10), em que o coeficiente  $C_n$  é análogo aos coeficientes  $a_n$  e  $b_n$  da série trigonométrica, sendo obtido por (11).

$$x(t) = \sum_{-\infty}^{\infty} C_n e^{jn\omega_0 t} \quad (10)$$

$$C_n = \frac{1}{T_0} \int_{T_0} x(t) e^{jn\omega_0 t} dt \quad (11)$$

Tanto a forma trigonométrica quanto a exponencial da série de Fourier consideram  $x(t)$  como sendo uma função qualquer, real ou complexa. Porém na maioria

das aplicações  $x(t)$  é real, como é o caso dos sinais neste trabalho. Segundo segundo Lathi (2007, p. 533), se o sinal de entrada do sistema LTI  $x(t)$  é real, isso significa que  $a_n$  e  $b_n$  também são reais para todos os valores de  $n$ , sendo portanto a série de Fourier representada na forma compacta:

$$x(t) = D_0 + \sum_{n=1}^{\infty} D_n \cos(n\omega_0 t + \theta_n) \quad (12)$$

Sendo:

$$D_0 = a_0 \quad (13)$$

$$D_n = \sqrt{a_n^2 + b_n^2} \quad (14)$$

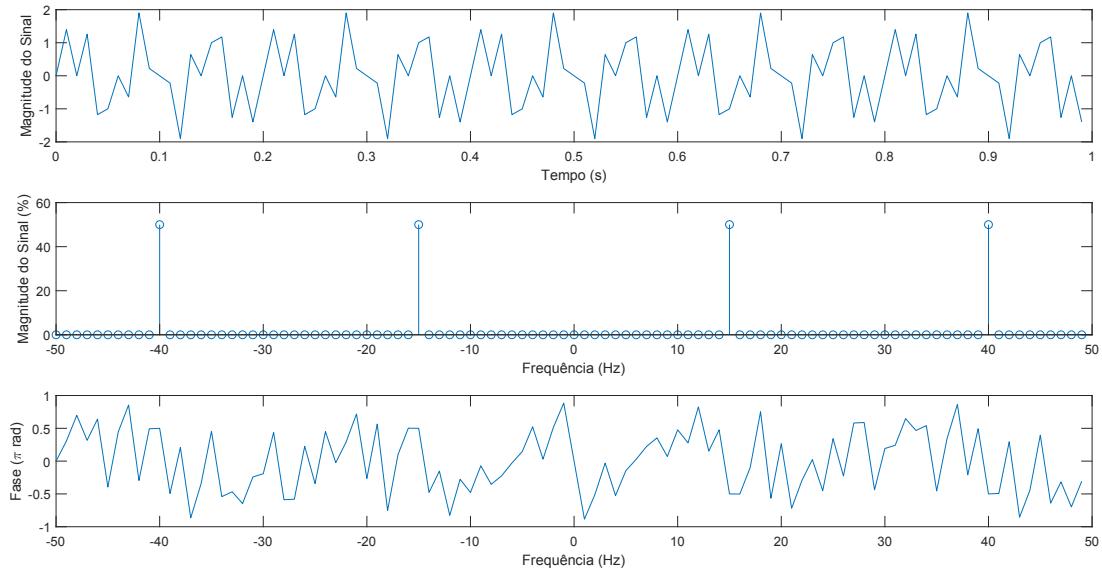
$$\theta_n = \tan^{-1} \frac{-b_n}{a_n} \quad (15)$$

Utilizar  $\cos$

### 2.1.3 ESPECTRO DE FOURIER

Por meio da série de Fourier na forma compacta, apresentada na Equação 12, conclui-se que um sinal real periódico  $x(t)$  pode ser descrito como uma soma de senoides de frequências  $n\omega_0$  e amplitudes  $D_n$  e fases  $\theta_n$ . Segundo Lathi (2007, p. 533), o espectro exponencial de Fourier é traçado a partir de  $D_n$  e  $\theta_n$  em função das frequências  $n\omega_0$ . Logo são traçados dois gráficos para o espectro exponencial de Fourier, um que relaciona  $C_n$  com  $n\omega_0$ , chamado espectro de magnitude. E outro que relaciona  $\theta_n$  com  $n\omega_0$ , chamado de espectro de fase.

Tomando como exemplo um sinal  $x(t) = \sin(2\pi 15t) + \sin(2\pi 40t)$ , expresso em um período igual a 1 segundo, como mostrado na Figura 9. Para este sinal é expresso o espectro de Fourier na mesma Figura (9), com o espectro de magnitude e fase.



**Figura 9: Aproximação do Sinal de Onda Quadrada por Soma de 8 Termos Senoidais**  
Fonte: Autoria Própria

Nota-se que o espectro da Figura (9) aparecem frequências negativas, as quais dividem a magnitude com suas frequências simétricas. Isso ocorre devido a simetria do angulo  $n\omega_0 t$  possuir no calculo dos coeficientes da série de Fourier. Para resolver este problema basta considerar apenas a parte positiva do espectro e multiplicar por 2 a magnitude das frequências no espectro de magnitude.

Para Lathi (2007, p. 533) os dois gráficos de magnitude e fase juntos formam o espectro de frequência, o qual revela os conteúdos de frequência do sinal  $x(t)$ , com suas amplitudes e fase. Conhecendo-se este espectro não só é possível analisar o sinal  $x(t)$  dentro do domínio da frequência, como também reconstruí-lo de forma fácil

## 2.2 SÉRIE DE FOURIER EM TEMPO DISCRETO

Ate aqui foi apresentada a forma continua da série de Fourier, porém para ser útil em uma aplicação computacional é necessário encontrar sua forma discreta, ou DFT (*Discrete Fourier Transform*). Segundo HAYKIN e Veen (2001, p. 314) a DFT é a única representação de Fourier que pode ser calculada por um computador, sendo amplamente usada para manipular sinais.

O primeiro passo para se obter uma DFT é considerar o teorema da Amos-tragem. Tal teorema afirma que um sinal real  $x(t)$ , cujo o espectro é limitado em  $\phi$  Hz, pode ser reconstruído a partir de suas amostras tomadas uniformemente a uma taxa

$f_s > 2\phi$  (LATHI, 2007, p. 679). Em seguida, a amostragem de  $x(t)$ , feita a uma frequência  $f_s$ , pode ser obtida pela multiplicação de  $x(t)$  por um trem de impulsos  $\delta(t)$ . Sendo tais impulsos unitários e periódicos, repetidos a cada  $T = 1/f_s$  segundos, por um numero total de amostras  $N_0$ , a amostragem pode ser definida por:

$$\bar{x}(t) = x(t)\delta_T(t) = \sum_{n=0}^{N_0-1} x(nT)\delta(t - nT) \quad (16)$$

Por conveniência, deseja-se obter um espectro do sinal amostrado  $x(t)$  em função de  $\omega$  ou expresso em termos de frequência. Para tal, segundo Lathi (2007, p. 681), o trem de impulsos  $\delta(t)$  é um sinal periódico que pode ser descrito pela série trigonométrica de Fourier da seguinte forma:

$$\delta_T(t) = \frac{1}{T}[1 + 2\cos(\omega_s t) + 2\cos(2\omega_s t) + 2\cos(3\omega_s t) + \dots] \quad (17)$$

Logo, multiplicando  $x(t)$  por  $\delta_T(t)$ , obtém-se:

$$\bar{x}(t) = x(t)\delta_T(t) = \frac{1}{T}[x(t) + 2x(t)\cos(\omega_s t) + 2x(t)\cos(2\omega_s t) + 2x(t)\cos(3\omega_s t) + \dots] \quad (18)$$

Segundo Oppenheim e Willsky (2010, p. 125), a transformada de Fourier do primeiro termo  $x(t)$ , em (18), é  $X(\omega)$ . Já a transformada de Fourier do segundo termo  $2x(t)\cos(\omega_s t)$  é  $X(\omega - \omega_s) + X(\omega + \omega_s)$ , e do terceiro termo  $2x(t)\cos(2\omega_s t)$  é  $X(\omega - 2\omega_s) + X(\omega + 2\omega_s)$ . E assim, semelhantemente a transformada de Fourier dos demais termos da serie que descreve (18), representam o espectro  $X(\omega)$  deslocado em  $n\omega_s$  e  $-\omega_s$ . Assim,

$$\bar{X}(\omega) = \frac{1}{T} \sum_{-\infty}^{\infty} X(\omega - n\omega_s) \quad (19)$$

Desde que a frequência de amostragem  $f_s$  garanta o critério do teorema da Amostragem, o sinal  $\bar{X}$  será constituído de repetições não sobrepostas de  $x(\omega_0)$ , a um intervalo de tempo  $T = 1/f_s$ . Logo tanto  $\bar{X}(\omega)$ , quanto  $\bar{x}(t)$  são periódicas e equivalentes, porém com representações distintas do aspecto amostrado. Sendo assim, através da propriedade de deslocamento no tempo da transformada de Fourier (20) e da (16), obtém-se (21) Oppenheim e Willsky (2010, p. 125):

$$\delta(t - nT) \longleftrightarrow e^{-jn\omega T} \quad (20)$$

$$\bar{x}(t) = \sum_{n=0}^{N_0-1} x(nT)e^{-jn\omega T} \quad (21)$$

Segundo Lathi (2007, p. 705), a transformada de  $\bar{x}(t)$  pode ser aproximada, considerando um certo *aliasing* negligenciável, para  $X(\omega)/T$ . Portanto:

$$X(\omega) = T \sum_{n=0}^{N_0-1} x(nT)e^{jn\omega T} \quad |\omega| \leq \frac{\omega_s}{2} \quad (22)$$

Analizando a propriedade periódica de  $x(t)$  e  $X(\omega)$ , e considerando  $x(nT)$  e  $X(r\omega_0)$  a n-ésima e r-ésima amostra de  $x(t)$  e  $X(\omega)$ , respectivamente, são definidas as seguintes variáveis:

$$x_n = Tx(nT) \quad (23)$$

$$x_n = \frac{T_0}{N_0}x(nT) \quad (24)$$

$$X_r = X(\omega) \quad (25)$$

$$\omega = r\omega_0 \quad (26)$$

$$X_r = X(r\omega_0) \quad (27)$$

$$\omega_0 = 2\pi f_0 = \frac{2\pi}{T_0} \quad (28)$$

Assim, substituindo (27) e (24) em (22), e fazendo  $\omega_0 T = \Omega_0 = 2\pi i/N_0$ , se obtém a seguinte expressão para a transformada discreta de Fourier (OPPENHEIM; WILLSKY, 2010, p. 125):

$$X_r = \sum_{n=0}^{N_0-1} x_n e^{j\omega_0 nr} \quad (29)$$

Onde:

$$\Omega_0 = \frac{2\pi}{N_0} \quad (30)$$

Para compactar a expressão de (29) se faz a substituição da expressão exponencial pela variável  $W$ , de modo que  $W_{N_0} = e^{j2\pi/N_0} = e^{-j\Omega_0}$ . Logo a expressão para DFT é dada por (31) (MEYER-BAESE, 2007, p. 344):

$$X_r = \sum_{n=0}^{N_0-1} x_n e^{j\omega_0 n r} \quad (31)$$

Onde:

$$0 \leq k \leq N_0 - 1 \quad (32)$$

## 2.3 TRANSFORMADA RÁPIDA DE FOURIER

Para se calcular uma DFT de  $N_0$  valores usando apenas (31), é necessário realizar um total de  $N_0^2$  multiplicações e  $N_0(N_0 - 1)$  somas utilizando números complexos. Deste modo, quando  $N_0$  assume um valor elevado, muitos recursos computacionais são necessários, até chegar ao ponto de que esse algoritmo se torna impraticável.

Para que se possa reduzir o numero de operações matemáticas necessárias para calcular a DFT é que surgiu o algoritmo criado por J.W. Cooley e John Tukey, conhecido como Transformada Rápida de Fourier ou FFT (*Fast Fourier Transform*) Lathi (2007, p. 719). Para reduzir o numero de cálculos, a FFT se utiliza da propriedade linear da transformada de Fourier. Já que, segundo Oppenheim e Willsky (2010, p. 119), a transformada de Fourier de um sinal pode ser dada pela combinação linear da transformada de Fourier de segmentos menores do mesmo sinal. Logo, é possível aplicar a DFT o paradigma da Divisão e Conquista, o qual é um recurso muito utilizado em algoritmos de ordenação.

Segundo Cormen *et al.* (2002, p. 21) um algoritmo de Divisão e Conquista realiza o desmembramento de um problema em vários subproblemas que são idênticos ao original, porém menores em sua faixa de ação, o que os torna mais simples de resolver. Em seguida, resolvem-se os subproblemas recursivamente e combinam-se essas soluções de modo a obter a solução para o problema original.

De modo muito semelhante o algoritmo da FFT prevê uma divisão recursiva

da DFT em dois blocos: bloco par e o bloco ímpar, como mostrado em (33) (CHU; GEORGE, 1999, p. 35). Nesta mesma equação os limites dos somatórios de ambas as parcelas ímpar e par foram redefinidas para englobar apenas metade dos  $N_0$  pontos, bem como os expoentes de  $W$  foram ajustados.

$$X_r = \underbrace{\sum_{n=0}^{\frac{N_0}{2}-1} x_{2n} W_{N_0}^{2nr}}_{\text{Parcela Par}} + \underbrace{\sum_{n=0}^{\frac{N_0}{2}-1} x_{2n+1} W_{N_0}^{(2n+1)r}}_{\text{Parcela Ímpar}} \quad (33)$$

Utilizando algumas das propriedades geométricas de  $W$ , já que o mesmo representa um numero complexo, pode-se realizar simplificações importantes em 34. Primeiro nota-se que  $W_{N_0/2} = W_{N_0}^2$ , logo:

$$X_r = \underbrace{\sum_{n=0}^{\frac{N_0}{2}-1} x_{2n} W_{N_0}^{2nr}}_{G_r} + \underbrace{\sum_{n=0}^{\frac{N_0}{2}-1} x_{2n+1} W_{N_0}^{(2n+1)r}}_{H_r} \quad (34)$$

Como  $G_r$  e  $H_r$  são DFTs com  $N_0/2$  pontos cada, então ambos possuem um período de  $N_0/2$ . Com base na propriedade periódica destas DFTs pode-se utilizar as simplificações (35) e (36) para reduzir o número de cálculos na DFT (LATHI, 2007, p. 721).

$$G_{r+\frac{N_0}{2}} = G_r \quad (35)$$

$$H_{r+\frac{N_0}{2}} = H_r \quad (36)$$

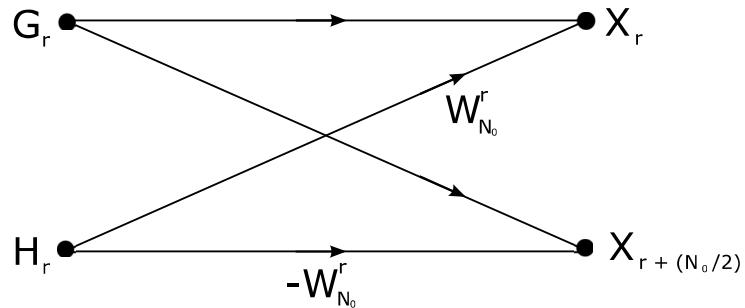
$$W_{N_0}^{r+\frac{N_0}{2}} = W_{N_0}^{\frac{N_0}{2}} = e^{-j\pi} W_{N_0} = -W_{N_0}^r \quad (37)$$

Alem disso, a expressão em (37) pode ser assumida para se reduzir o numero de cálculos da FFT. Portanto, usando (38) e (39) se obtém, respectivamente, os primeiros  $N_0/2$  pontos e os últimos  $N_0/2$  pontos da FFT.

$$W_{N_0}^{r+\frac{N_0}{2}} = W_{N_0}^{\frac{N_0}{2}} = e^{-j\pi} W_{N_0} = -W_{N_0}^r \quad (38)$$

$$W_{N_0}^{r+\frac{N_0}{2}} = W_{N_0}^{\frac{N_0}{2}} = e^{-j\pi} W_{N_0} = -W_{N_0}^r \quad (39)$$

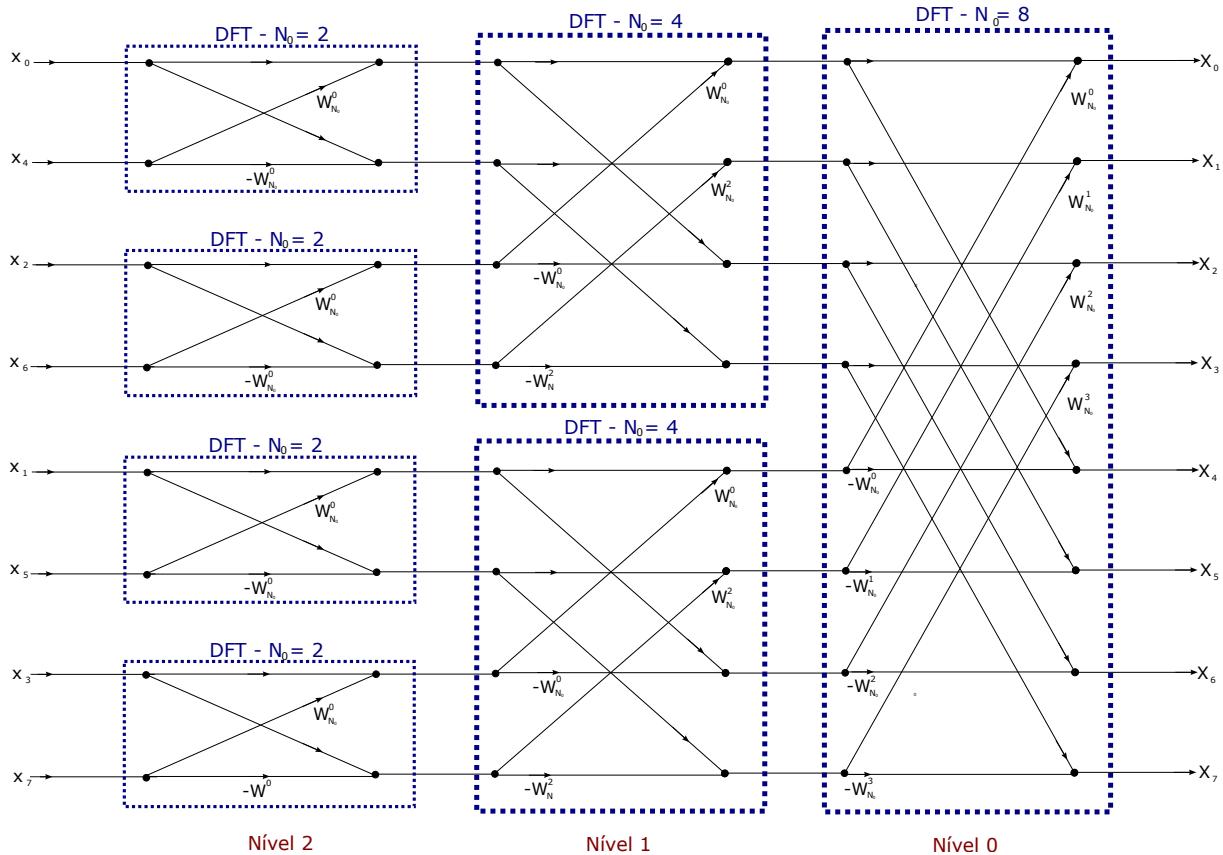
Portanto, uma DFT pode ser calculada combinando duas DFTs de  $N_0/2$ , tal como mostrado em (38) e (39). É comum na literatura representar este processo de cálculo de DFT feito pelo algoritmo da FFT pelo diagrama da Figura (10). Este diagrama é conhecido como *Butterfly* de Fluxo do Sinal (CHU; GEORGE, 1999, p. 36).



**Figura 10: Butterfly do Fluxo do Sinal**

Fonte: Lathi (2007, p. 721)

Aliando o conceito de divisão em conquista ao método de cálculo da DFT usando o diagrama *Butterfly*, a representação do algoritmo da FFT pode ser facilmente representado pelo diagrama da Figura (11) (LATHI, 2007, p. 722). Nesta figura, a FFT é feita para apenas 8 amostras de sinal  $X$ .



**Figura 11: FFT de 8 Pontos**  
Fonte: Lathi (2007, p. 722)

Apos se dividir uma DFT de tamanho  $N_0$  em duas DTFs de tamanho  $N_0/2$ , e subdividindo cada uma das DFTs de tamanho  $N_0/2$  em duas  $N_0/4$  (LATHI, 2007, p. 721). E assim o procedimento continua ate que se atinja um nível em que as DFTs tenham tamanho  $N_0/2^n = 2$ , ou seja quando se atinge DFTs que possuem um custo de cálculo mínimo.

Um fato importante sobre o algoritmo da FFT é que o valor  $N_0$  pode ser escolhido segundo a relação  $N_0 = r^n$ , onde  $n$  é o numero de níveis necessários para calcular a FFT, e  $r$  é o mínimo tamanho DFT. Os algoritmos de FFT mais usados possuem a base  $r$  igual a 2 ou a 4. Consequentemente, estes algoritmos são conhecidos respectivamente como Radix-2 e Radix-4 (MEYER-BAESE, 2007, p. 365). Neste trabalho o foco sera o algoritmo com a base  $r$  igual a 2, ou seja o Radix-2.

Na Figura 2 as DFTs estão agrupadas por níveis, onde cada nível abrange as DFTs de mesmo tamanho  $N_0/n$ , e no ultimo nível a esquerda há quatro DFTs de tamanho 2. O numero de níveis necessários em uma FFT de  $N_0$  pontos é  $\log_2 N_0$ . Os valores de  $X$  a direita estão ordenados de forma crescente, por ? em os valores de  $x$  a esquerda estão ordenados de forma diferente. Esta ordenação é conhecida como

*Bit-Reverse* (CHU; GEORGE, 1999, p. 51).

Segundo Chu e George (1999, p. 51), quando se divide o processo de cálculo de uma DFT em duas, sendo uma responsável pelos valores pares e a outra pelo valores ímpares, conforme as conexões entre os níveis vão ocorrendo há, permutações entre os sinais. O processo de *Bit-Reverse* prevê estas permutações, e através dele é possível saber qual a ordem adequada dos sinais na entrada da FFT. Para aplicar o conceito de *Bit-Reverse*, basta considerar um elemento  $x_k$  de ordem  $n$  e escrever-lo na base binária com  $\log_2 N_0$  bits. Em seguida, para determinar onde  $x_k$  será ocupado, basta converter novamente para base decimal o número binário obtido lendo esse na ordem inversa dos bits.

Ao final de todo o processo de simplificação do cálculo da DFT, o algoritmo da FFT necessita apenas realizar  $(N_0/2) \log_2 N_0$  multiplicações e  $N_0 \log_2 N_0$  somas complexas (LATHI, 2007, p. 720). Desta forma, reduz-se assim a complexibilidade do algoritmo, tornando a DFT praticável até para valores elevados de  $N_0$ .

## 2.4 ALGORITMO CORDIC

Na equação (38) nota-se que o cálculo da FFT depende essencialmente de um multiplicação complexa entre  $W_{N_0}^r$  e  $H_r$ , onde  $H_r$  representa um vetor complexo qualquer, e  $W_{N_0}^r$  é igual a  $e^{\frac{2\pi r}{N_0}}$ . Esta tarefa pode ser realizada através da representação de  $W_{N_0}^r$  e  $H_r$  na forma retangular, considerando para efeitos de exemplo  $W_{N_0}^r = x + iy$  e  $H_r = a + ib$ , logo:

$$W_{N_0}^r = x + iy \quad (40)$$

$$H_r = a + ib \quad (41)$$

$$H_r \cdot W_{N_0}^r = (x + iy)(a + ib) \quad (42)$$

$$H_r \cdot W_{N_0}^r = (xa - yb) + i(xb + ya) \quad (43)$$

Desta forma serão necessários utilizar 4 multiplicadores e 2 somadores, para completar esta operação (DESPAIN, 1974, p. 1). Porém em termos de complexidade hardware, multiplicadores são mais elaborados e em muitos dispositivos FPGA possuem apenas algumas dezenas, o que limita a utilização de multiplicadores em paralelo, reduzindo a velocidade de cálculo da FFT.

Para contornar o problema considere expressar o vetor  $H_r$  na forma polar

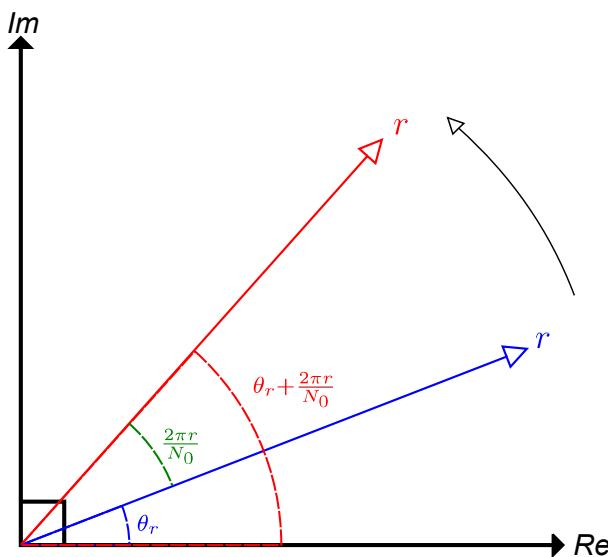
$re^{\theta_r}$  e manter  $W_{N_0}^r$  igual a  $e^{\frac{2\pi r}{N_0}}$ , logo:

$$H_r = Re^{(\theta_r)} \quad (44)$$

$$H_r \cdot W_{N_0}^r = re^{(\theta_r)} \cdot e^{\left(\frac{2\pi r}{N_0}\right)} \quad (45)$$

$$H_r \cdot W_{N_0}^r = re^{\left(\theta_r + \frac{2\pi r}{N_0}\right)} \quad (46)$$

Assim  $H_r \cdot W_{N_0}^r$  é equivalente a operação trigonométrica de rotacionar o vetor complexo  $H_r$  pelo ângulo de  $2\pi r/N_0$ . Como mostrado na Figura (12)



**Figura 12: Rotação de  $H_r$  pelo ângulo de  $2\pi r/N_0$**   
Fonte: Autoria Própria

Implementar operações trigonométricas sem utilizar multiplicadores, para evitar o gargalo que eles possam provocar, parece difícil, porém existe uma técnica bastante usada no desenvolvimento de circuitos lógicos em FPGA que o possibilita. Com o objetivo de oferecer uma solução para o cálculo de funções trigonométricas mais simples, utilizando o mínimo de recursos de tempo e *hardware*, Jack E. Volder(VOLDER, 1959) desenvolveu a técnica de computação trigonométrica CORDIC (*COordinate Rotation Digital Computer*).

### 2.4.1 CORDIC TRADICIONAL

O algoritmo CORDIC tem como base as micro-rotações do vetor alvo, de tal modo que cada micro-rotação possa ser feita por somas e deslocamentos de bits. Para tal considere o vetor  $H_r = x + iy$ , e a matriz de rotação  $A$  em função do ângulo  $\theta$ . A rotação do vetor é dada pela equação (47) (GARRIDO *et al.*, 2016).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (47)$$

Isolando o termo  $\cos(\theta)$  da equação (47), é obtido o sistema (48), o qual é a base da técnica CORDIC convencional (EL-MOTAZ *et al.*, 2014).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos(\theta) \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (48)$$

Segundo (VOLDER, 1959), ao invés de rotacionar completamente o vetor  $H_r$  pelo ângulo  $\theta$ , ou pelo ângulo  $2\pi r/N_0$  como mostrado na Figura (12), o algoritmo CORDIC rotaciona  $H_r$  por ângulos  $\theta_n$  muito menores, sendo estes frações de  $\theta$  de tal forma que:

$$\theta = \sum_{n=0}^{N-1} \mu_n \theta_n + \zeta \quad (49)$$

onde  $\mu_n$  representa o sentido da micro rotação  $\theta_n$ , se ela for no sentido horário é igual a 1, caso contrário é igual a -1. E  $\zeta$  é o erro acumulado da aproximação pelo somatório, aqui este será considerado suficientemente pequeno para ser ignorado. O primeiro ponto dessa abordagem é o fato de que se um ângulo  $\theta_n$  for suficientemente pequeno é possível afirmar que:

$$\theta_n \simeq \tan^{-1} \theta_n \quad (50)$$

Assim é possível substituir  $\theta$  no sistema da Equação (48), e obter a seguinte expressão:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos(\tan^{-1} \theta_n) \begin{bmatrix} 1 & -\tan(\tan^{-1} \theta_n) \\ \tan(\tan^{-1} \theta_n) & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (51)$$

O segundo ponto é que para satisfazer a Equação (49) so é necessário que a soma do conjunto de micro rotações  $\theta_n$  resulte no angulo  $\theta$ , o que abre a possibilidade de se escolher um conjunto de micro rotações baseadas em deslocamento de bits, como:

$$\theta_n = \tan^{-1} 2^{-n} \quad (52)$$

Logo aplicando a Equação(52) em (51), é obtido a expressão final para CORDIC:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \prod_{n=0}^{N-1} K_c \begin{bmatrix} 1 & -\mu 2^{-n} \\ \mu 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (53)$$

$$k_c = \cos(\tan^{-1}(2^{-n})) = \frac{1}{\sqrt{1 + 2^{-2n}}} \quad (54)$$

onde

$$\mu \in \{-1, 1\} \quad (55)$$

$K_c$  é calculado a cada micro-rotação  $n$ , assim seu valor total é dado pelo produto de todos os  $N$  ganhos. O número total de micro-rotações ( $N$ ) é escolhido de acordo com o SQNR (*Signal-to-Quantization-Noise Ratio*) admissível a cada rotação. Segundo (VORDER, 1959), considerando  $N \rightarrow \infty$ ,  $K_c$  é passa a ser constante e aproximadamente igual a 0,6073. O inverso de  $K_c$  é igual a 1,647, sendo conhecido como "Ganho CORDIC". Tal ganho é independente do angulo a ser rotacionado, e em muitos sistemas este ganho é só compensado fora do bloco lógico de cálculo do CORDIC.

Como pode ser observado o algoritmo CORDIC se resume a operações de deslocamento de *bit* e somas. A direção das micro-rotações é determinada por  $\mu$ , que depende diretamente sinal do ângulo  $\theta_n$  a se rotacionar. Em aplicações onde o ângulo a se rotacionar é previamente conhecido, que é o caso da FFT, as sequências de rotações  $\mu$  podem ser armazenado em uma ROM (EL-MOTAZ *et al.*, 2014). Tornando as micro-rotações como  $n$  interações, armazenando em  $z$  os ângulos rotacionados a cada interação de  $\theta_n$ , o algoritmo CORDIC é dado por:

$$x(n+1) = x(n) - [\mu 2^{-n}]y(n) \quad (56)$$

$$y(n+1) = y(n) + [\mu 2^{-n}]x(n) \quad (57)$$

$$z(n+1) = z(n) - \tan^{-1}[\mu 2^{-n}] \quad (58)$$

$$(59)$$

Onde:

$$H_r = x(0) + iy(0) \quad (60)$$

$$\theta = z(0) \quad (61)$$

$$n = \{0, 1, \dots, N-1\} \quad (62)$$

$$\mu = \begin{cases} 1 & z(n) \geq 0 \\ -1 & z(n) < 0 \end{cases} \quad (63)$$

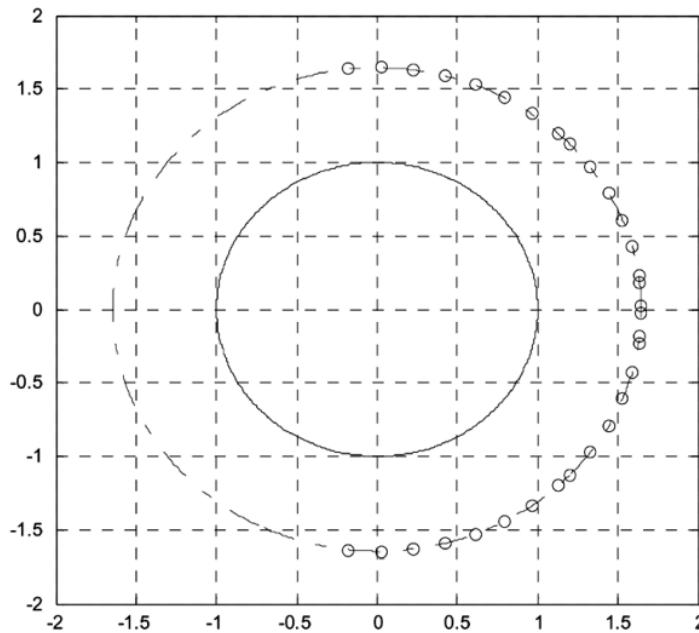
#### 2.4.2 EEAS-CORDIC

No algoritmo CORDIC cada ângulo de rotação  $\theta_n$  é necessariamente determinado de maneira sequencial após cada interação  $n$ , através de conjunto de ângulos elementares definidos como:

$$S_1 = \{\tan^{-1}(\mu 2^{-n})\} \quad (64)$$

$$: \mu \in \{-1, 1\}, n \in \{0, 1, \dots, N-1\} \quad (65)$$

A partir do ângulo inicial de  $H_r$ , o algoritmo Cordic realiza as  $N$  interações deslocando o vetor através do conjunto de ângulos elementares, de forma a reduzir a diferença entre o angulo atual e o ângulo. Para fins de comparação a Figura (13) apresenta a densidade combinacional dentro do circulo unitário do conjunto de ângulos elementares do Algoritmo CORDIC tradicional.



**Figura 13: Ângulos Elementares - Cordin Tradicional com N=4**

**Fonte:**

**Fonte: (LIN; WU, 2005)**

Segundo Wu e Wu (2001) o maior problema do CORDIC é a baixa velocidade computacional de calculo deste algoritmo, devido principalmente a necessidade de um grande número de interações  $N$  para atingir um erro de aproximação  $\zeta$  aceitável. O error de aproximação  $\zeta$  é dado por:

$$\zeta = \left| \theta - \sum_{n=0}^{N-1} \mu \theta_n \right| = \left| \theta - \sum_{n=0}^{N-1} \mu \tan^{-1}(2^{-n}) \right| \quad (66)$$

Onde:

$$\mu \in \{-1, 0, 1\} \quad (67)$$

Em aplicações onde os ângulos de rotação são conhecidos é possível relaxar as restrições da Equação (13), através do método *Angle Recoding* (AR) (WU; WU, 2001). Os AR tem como objetivo reduzir o número de interações CORDIC e o erro  $\zeta$ , para tal o AR que expande o conjunto de combinações lineares da Equação (65) adicionando zero ao conjunto de  $\mu$ . Obtendo assim uma melhor aproximação para certos valores de  $\theta$  e uma redução de até 50% no número de interações (MEHER *et al.*, 2009).

Por outro lado o método *Extended Elementary-Angle-Set Recoding* (EE-

ASR) apresenta um método baseado no AR que, além de expandir o conjunto de  $\mu$ , estende também o conjunto de ângulos elementares (*Elementary-Angle-Set*, EAS), para aumentar a possibilidades de decomposição do ângulo de rotação (WU *et al.*, 2003). Para perceber a modificação que o EEASR propõem nota-se primeiramente que o conjunto de ângulos elementares no CORDIC utilizando AR é definido como:

$$S_1 = \{\tan^{-1}(\mu 2^{-s})\} \quad (68)$$

$$: \mu \in \{-1, 0, 1\}, s \in \{0, 1, \dots, N-1\} \quad (69)$$

Como é possível notar em (69) os ângulos elementares dependem de apenas um termo potência de dois, ou *Signed Power of Two* (SPT). Segundo (WU *et al.*, 2003) para aumentar a precisão dos ângulos elementares e consequentemente reduzir o numero de interações pode-se adicionar mais um termo SPT em (69). Assim,

$$S_2 = \{\tan^{-1}(\mu_0 2^{-s_0} + \mu_1 2^{-s_1})\} \quad (70)$$

$$: \mu_0, \mu_1 \in \{-1, 0, 1\}, s_0, s_1 \in \{0, 1, \dots, S\} \quad (71)$$

Onde  $S$  é denominado como o número máximo de deslocamentos de *bits* a direita que podem ser realizados. Este valor esta diretamente relacionado com o quantidade de *bits* utilizados para representar um número dentro da arquitetura onde o Cordic é implementado. Por exemplo em uma aplicação em FPGA, onde a palavra binária utilizada para representar um número inteiro tenha apenas 16 *bits*, não faz sentido o valor de  $S$  ser maior do 15.

Portanto alterando a equação (51) com base em (71) é obtido a expressão para calculo interativo CORDIC utilizando o EEASR:

$$\begin{bmatrix} x(n+1) \\ y(n+1) \end{bmatrix} \begin{bmatrix} 1 & \mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)} \\ \mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)} & 1 \end{bmatrix} \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \quad (72)$$

$$: \mu_i, \mu_j \in \{-1, 0, 1\}, s_0, s_1 \in \{0, 1, \dots, S\} \quad (73)$$

$$\theta_n = \tan^{-1} (\mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)}) \quad (74)$$

$$z(n+1) = z(n-1) + \theta_n \quad (75)$$

É importante notar que ao adicionar mais termos a  $S_1$ , o ganho  $K_c$  também é modificado passando a ser definido por:

$$K(n)_c = \frac{1}{\sqrt{1 + [\mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)}]^2}} \quad (76)$$

Com a alteração de  $K_c$  o ganho passa a não ser mais constante, e varia de acordo com cada interação  $N$ . Sendo assim necessário calcular o ganho  $K_c$  para cada interação afim de realizar a compensação. O valor de  $K_c$  ao final de cada operação de cálculo CORDIC passa a ser definido por:

$$K_c = \prod_{n=0}^{N-1} K(n)_c \quad (77)$$

Para casos em que o ângulo  $\theta$  a ser rotacionado é conhecido, como é o caso da FFT, é possível escolher previamente o conjunto de valores  $\mu_0$ ,  $\mu_1$ ,  $s_0$  e  $s_1$ , e consequentemente através da Equação (76) determinar o valor de  $K_c$  a ser compensado a cada interação, e consequentemente por meio de (77) determinar a compensação de  $K_c$  a ser realizada após cada operação de rotação. Para tal é realizado, após as interações de rotação do algoritmo, uma correção no módulo do vetor resultante por meio da seguinte operação de rotação modificada:

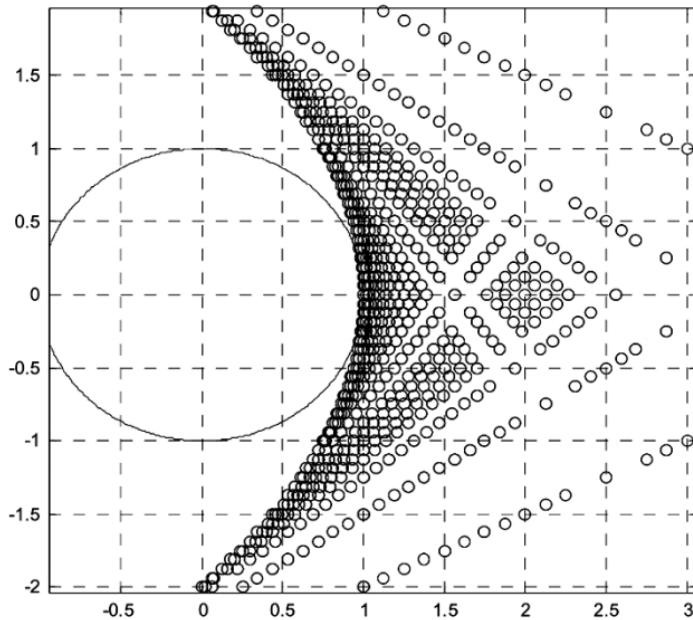
$$x(n+1) = x(n) - [\mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)}]x(n) \quad (78)$$

$$y(n+1) = y(n) - [\mu_0 2^{-s_0(n)} + \mu_1 2^{-s_1(n)}]y(n) \quad (79)$$

$$z(n+1) = z(n) \quad (80)$$

Onde  $\mu_0$ ,  $\mu_1$ ,  $s_0$  e  $s_1$  para esta operação são escolhidos de modo a minimizar o erro de compensação de  $K_c$ .

Com a relaxação das restrições introduzidas pelo método AR e pelo EE-ASR, a densidade combinacional dentro do círculo unitário do conjunto de ângulos elementares do Algoritmo CORDIC aumenta, se comparado ao CORDIC tradicional, como pode ser visto na Figura (14).



**Figura 14: EEAS com N=2 e S=4**  
Fonte: (LIN; WU, 2005)

#### 2.4.2.1 ALGORITMO TBS

A relaxação das restrições feita pelos métodos AR e EEASR tornaram o algoritmo iterativo CORDIC mais eficiente, porém ele abre uma questão crucial: a determinação dos conjuntos  $\mu$  e  $s$ . No algoritmo CORDIC tradicional o valor de  $\mu$  estava contido no conjunto  $\{-1, 1\}$ , e era determinado com base no sinal de  $z(n)$  a cada interação, porém no método AR  $\mu$  passa a estar contido no conjunto  $\{-1, 0, 1\}$ . Logo determinar o valor de  $\mu$  a cada interação torna-se uma tarefa de otimização, em relação a minimizar o erro  $\zeta$ , na forma:

$$\min \zeta = \left| \theta - \sum_{n=0}^{N-1} \mu \tan^{-1}(2^{-n}) \right| \quad (81)$$

Onde:

$$\mu \in \{-1, 0, 1\} \quad (82)$$

No algoritmo CORDIC convencional o conjunto dos ângulos elementares é definido por  $S_1$  na Equação (65), e a cada interação o deslocamento do vetor é feito com base no elemento  $n$  deste conjunto. O método EEASR adiciona mais um termo SPT a expressão do conjunto dos ângulos elementares  $S_2$ , definido na Equação (71),

o que possibilita a escolha de qualquer termo  $s_0$  e  $s_1$ . Logo além de agregar a mesma necessidade de determinar o conjunto otimizado  $\mu_0$  e  $\mu_1$  de AR, o EEASR também requer a determinação do conjunto otimizado  $s_0$  e  $s_1$ , a fim de também minimizar o erro  $\zeta$ . Portanto a função de minimização do erro  $\epsilon$  é dada pela seguinte expressão:

$$\min \zeta = \left| \theta - \sum_{n=0}^{N-1} \tan^{-1}(\mu_0 2^{-s_0} + \mu_1 2^{-s_1}) \right| \quad (83)$$

$$: \mu_0, \mu_1 \in \{-1, 0, 1\}, s_0, s_1 \in \{0, 1, \dots, S\} \quad (84)$$

Para otimizar a Função (84) é possível utilizar um variedade de algoritmos heurísticos ou não heurístico a fim de determinar um conjunto de parâmetros que minimize o erro  $\zeta$ , como por exemplo *Dijkstra*, 'Caminhos Mínimos' ou o mais comum Algoritmo *Greedy*. Wu *et al.* (2003) propõe um método de otimização para parâmetros EEAS chamado *Trellis-Based Search*(TBS).

A partir do numero máximo de interações  $N$ , do ângulo de rotação  $\theta$  e de  $W$ , que representa numero de bits de  $\theta$ , o TBS emprega um método de otimização para encontrar os melhores parâmetros  $s^0(n)$ ,  $s^1(n)$ ,  $\mu_0$  e  $\mu_1$  (WU *et al.*, 2003). TBS é baseado no efeito que os diferentes arranjos dos ângulos elementares  $S_2$  possíveis tem sobre  $\zeta$ . O algoritmo segue o seguintes passos:

1. *Inicialização*: Inicialmente é definido o vetor  $r(k)$ , o qual representa o conjunto de ângulos elementares presentes na Equação (84), obtidos através das possíveis combinação não redundantes de  $\mu_0$ ,  $\mu_1$ ,  $s_0$  e  $s_1$ . Em seguida é definido a matriz  $\phi(n, k)$  para  $(1 \leq n \leq N)$  e  $(1 \leq k \leq z(S_2))$ , representando a estrutura de acumulação do algoritmo utilizada para alcançar o melhor resultado.  $N$  é o número de interações desejadas, enquanto  $z(S_2)$  representa o número de elementos do vetor  $r(k)$ . Para então iniciar o algoritmo é  $\phi(1, k)$  é preenchido com o vetor  $r(k)$ .
2. *Acumulação*: Em cada interação  $n$  o algoritmo percorre os elementos  $k$  do vetor  $\phi(n+1, k^*)$ , atribuindo a cada valor  $k$  o menor resultado encontrado da expressão  $\|\phi(n, k^*) + r(k) - \theta\|$ . O índice  $i$  varia  $1 \leq i \leq N - 1$ , e  $k$  varia de  $1 \leq k \leq Z(S_2)$ .
3. *Determinação do Ótimo Global*: Ao final do processo de acumulação os elemento da coluna  $\phi(N, )$  apresentam os resultados globais. Logo dentro desta coluna é determinado o valor que mais se aproxima de  $\theta$ , defini-se este como  $\theta_{TBS}$ .

4. *Determinação do Caminho Solução:* A partir do elemento  $\theta_{TBS}$  da coluna  $\phi(N, \cdot)$  é traçado o caminho reverso até a coluna  $\phi(1, \cdot)$ . A começar por ( $k' = \theta_{TBS}$ ) e ( $i = N$ ) é escolhido o valor da coluna ( $i - 1$ ) que minimiza  $\|\phi(1 : z(S_2), i - 1) + r(k') - \theta\|$ , em seguida se atribui a  $k'$  a posição  $k$  do valor minimizante. Os valores  $k'$  são armazenados em um vetor e representam as combinações de ângulos elementares que minimizam o erro  $\epsilon$ .

Segue abaixo o pseudo-código que implementa o algoritmo TBS.

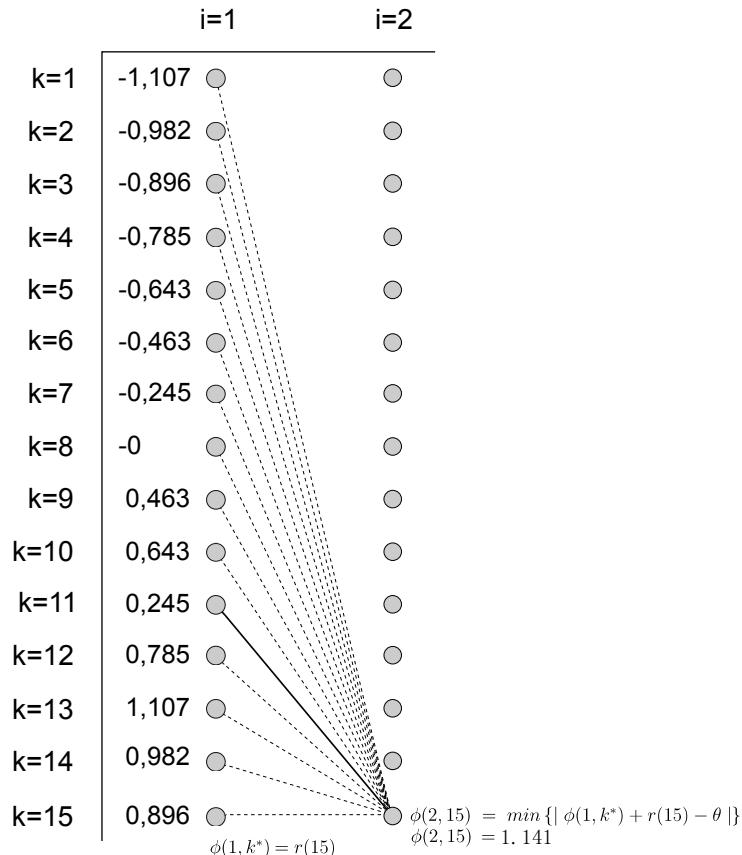
```

1 % Inicialização
2  $\phi(1, k) = r(k)$  para todo  $k$ ,
3
4 %Acumulação
5 FOR  $i = 1$  to  $N - 1$ 
6   FOR  $k = 1$  to  $Z(S_2)$ 
7      $\phi(i + 1, k) = \min \{|\phi(i, k^*) + r(k) - \theta| : 1 \leq k^* \leq Z(S_2)\}$ 
8   END
9 END
10
11 %Determinação do Ótimo Global
12  $\theta_{TBS} = \min \{|\phi(R_m, k^*) - \theta| : 1 \leq k^* \leq Z(S_2)\}$ 
13 Result( $N$ ) =  $k^*$ 
14
15 %Determinação do Caminho Solução
16 FOR  $i = N$  to  $2$ 
17   Encontra  $k'$  tal que  $\phi(i, k^*) = \min \{|\phi(i - 1, k') + r(k^*) - \theta| : 1 \leq k' \leq Z(S_2)\}$ 
18    $k^* = k'$ 
19   Result( $i - 1$ ) =  $k'$ 
20 END
21
```

Em (WU *et al.*, 2003) o autor utiliza um exemplo base para explicar o algoritmo TBS, o qual também será usado aqui. Para este exemplo os ângulos de rotação  $\theta$  será  $\pi/3$ , o número máximo de interações será  $N = 4$  e a resolução de  $\theta$  será de  $W = 4$  bits.

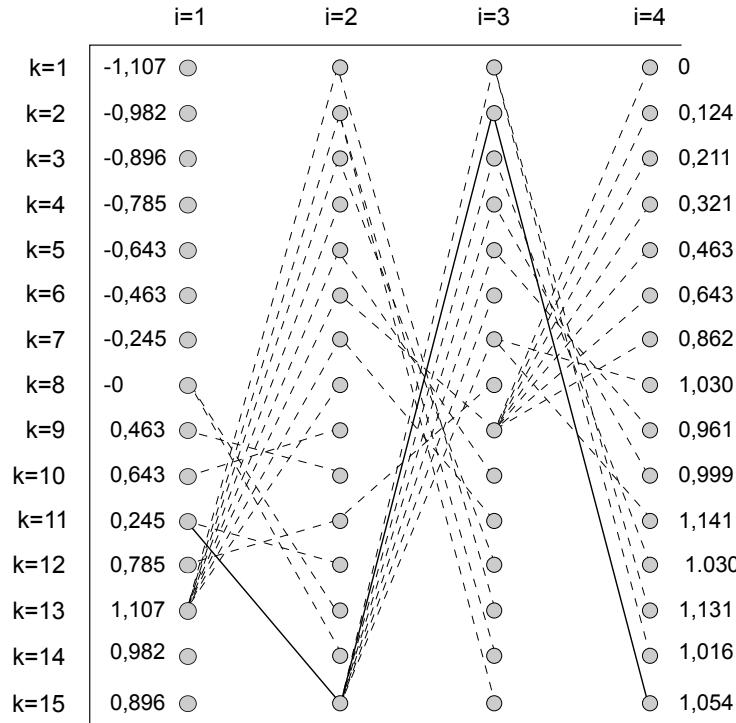
A primeira etapa do algoritmo então inicia com a definição do vetor  $r(k)$ , e o preenchimento da primeira coluna da matriz  $\phi(1, \cdot)$ , como pode ser visto na Figura (15). Na segunda etapa do algoritmo é realizada a acumulação, onde cada elemento das colunas 2, 3 e 4 são preenchidos com base na coluna anterior a partir da expressão  $\|\phi(n, k^*) + r(k) - \theta\|$ . Na Figura (15) o elemento  $\phi(2, 15)$  representa esta etapa, onde

o valor do elemento  $\phi(2, 15)$  é definido a partir da soma do ângulo representante da linha 15 ( $r(15)$ ), com o elemento da coluna 1 que minimiza o erro de aproximação do resultado desta soma com  $\theta$ , neste caso  $\pi/3$ . Este processo se estende até completar a coluna 4 da matriz  $\phi$ .



**Figura 15: Exemplo de execução do Algoritmo TBS**  
Fonte: Autoria Própria

Após a etapa de acumulação, a ultima coluna de  $\phi$  apresenta os resultados globais da acumulação. Logo para determinar o valor ótimo global é escolhido na coluna  $\phi(4, )$  o valor  $\theta_{TBS}$ , que mais se aproxima de  $\theta$  ou  $\pi/3$ . Após esta etapa é iniciada a determinação do vetor *Result*, o qual deve conter o caminho solução necessário para atingir o valor de ótimo global  $\theta_{TBS}$ . Na Figura (16) é apresentado os caminhos mínimos para cada coluna da matriz  $\phi$ , e consequentemente o melhor caminho encontrado pelo algoritmo.



**Figura 16: Exemplo de execução do Algoritmo TBS**  
Fonte: Autoria Própria

A tabela (2) apresenta os valores encontrados para  $\phi$  na execução do exemplo apresentado. Em destaque estão os valores do caminho solução, que são armazenados em *Result*.

	i=1	i=2	i=3	i=4
k=1	-1.1071	0	0.0339	0
k=2	-0.9828	0.1244	0.1582	0.1244
k=3	-0.8961	0.2111	0.2450	0.2111
k=4	-0.7854	0.3218	0.3556	0.3218
k=5	-0.6435	0.4636	0.4975	0.4636
k=6	-0.4636	0.6435	0.6774	0.6435
k=7	-0.2450	0.8622	0.8961	0.8622
k=8	0	1.1071	1.0304	1.0304
k=9	0.4636	1.1071	1.1071	0.9612
k=10	0.6435	1.1071	1.1071	0.9991
k=11	0.2450	1.0304	1.1071	1.1410
k=12	0.7854	1.0304	0.9965	1.0304
k=13	1.1071	1.1071	1.1071	1.1410
k=14	0.9828	0.9828	1.1071	1.0167
k=15	0.8961	1.1410	1.0204	1.0543

**Tabela 2: Matriz  $\phi$  para o dado Exemplo**  
Fonte: Autoria Própria

### 2.4.3 MSR-CORDIC

Como pode ser visto na Seção (2.4.1) o Algoritmo Cordic Tradicional possui a necessidade de realizar a correção do ganho  $k_c$  após o processo de rotação do vetor, além de requisitar um número elevado de interações a fim de reduzir o erro de aproximação do vetor. Para reduzir o número de interações Cordic o EEAS aumenta o número de termos SPT, porém acaba por impactar o ganho  $K_c$ , que passa a não ser mais constante e necessita ser compensado a cada iteração. Tanto no algoritmo tradicional quanto o EEAS a correção final de  $K_c$ , que se estabelece sempre abaixo da unidade, provoca uma degradação do SQNR.

Segundo Lin e Wu (2005) para evitar esta degradação é necessário manter o modulo do vetor de entrada o mais perto da unidade a cada interação. Assim Lin e Wu (2005) reformula o algoritmo Cordic para um novo esquema que passa a distribuir os termos SPT de forma diferente do EEAS, afim de dar mais liberdade ao ganho  $K_c$  e reduzir o número de interações necessárias para atingir um erro de aproximação admissível. Tal algoritmo pode ser expressado por:

$$\begin{bmatrix} x(n+1) \\ y(n+1) \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^J \mu_j 2^{-s_j(n)} & \sum_{i=1}^I \mu_i 2^{-s_i(n)} \\ \sum_{i=1}^I \mu_i 2^{-s_i(n)} & \sum_{j=1}^J \mu_j 2^{-s_j(n)} \end{bmatrix} \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \quad (85)$$

$$z(n+1) = z(n-1) + \theta_n \quad (86)$$

$$: \mu_i, \mu_j \in \{-1, 0, 1\}, s_i, s_j \in \{0, 1, \dots, S\} \quad (87)$$

A reformulação apresentada por Lin e Wu (2005) incrementa termos SPT nas parcelas  $x(n)$  de  $x(n+1)$  e  $y(n)$  de  $y(n+1)$ , afim de possibilitar o ganho  $K_c$  se tornar maior ou menor do que a unidade, dependendo da escolha dos parâmetros Cordic. Através desta alteração é possível realizar a operação de microrotação e a compensação do ganho  $K_c$  simultaneamente. Por tais motivos este algoritmo Coridc é denominado de *Mixed Scaling Rotation* (MSR). Consequentemente ao inserir mais termos na função Coridc, tanto  $\theta_n$  quanto  $K_c$  precisam ser corrigidos (KUO *et al.*, 2003). Portanto  $\theta_n$  e  $K_c$  passam a definidos por:

$$\theta_n = \tan^{-1} \left( \frac{\sum_{i=1}^I \mu_i 2^{-s_i(n)}}{\sum_{j=1}^J \mu_j 2^{-s_j(n)}} \right) \quad (88)$$

$$K(n) = \frac{1}{\sqrt{\left(\sum_{i=1}^I 2^{-s_i(n)}\right)^2 + \left(\sum_{j=1}^J 2^{-s_j(n)}\right)^2}} \quad (89)$$

$$K_c = \prod_{n=0}^{N-1} K(n) \quad (90)$$

Como pode ser percebido em (89) e (88), através da escolha adequada dos conjuntos de termos  $\mu_i$ ,  $\mu_j$ ,  $S_i$  e  $S_j$ , e dos valores de  $I$  e  $J$  é possível aproximar  $K_c$  da unidade a cada interação ao mesmo tempo em que o erro  $\zeta$  é reduzido. No MSR o conjunto de ângulos elementares é definidor por:

$$S_3 = \{\tan^{-1}(\mu_0 2^{-s_0} + \mu_1 2^{-s_1})\} \quad (91)$$

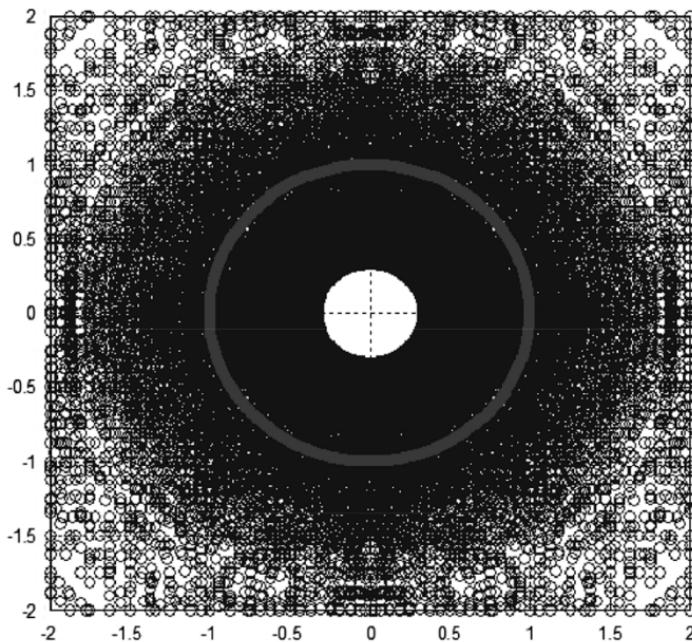
$$\therefore \mu_0, \mu_1 \in \{-1, 0, 1\}, s_0, s_1 \in \{0, 1, \dots, S\} \quad (92)$$

E consequentemente o erro  $\zeta$  é definido por:

$$\min \zeta = \left| \theta - \sum_{n=0}^{N-1} \tan^{-1}(\mu_0 2^{-s_0} + \mu_1 2^{-s_1}) \right| \quad (93)$$

$$\therefore \mu_0, \mu_1 \in \{-1, 0, 1\}, s_0, s_1 \in \{0, 1, \dots, S\} \quad (94)$$

Com mais graus de liberdade o MSR possui uma densidade combinacional dentro do círculo unitário maior do que o CORDIC tradicional e o EEAS-CORDIC (KUO *et al.*, 2003), como pode ser visto na Figura (17).



**Figura 17: MSR com I=2, J=1 e N=2**

Fonte: (LIN; WU, 2005)

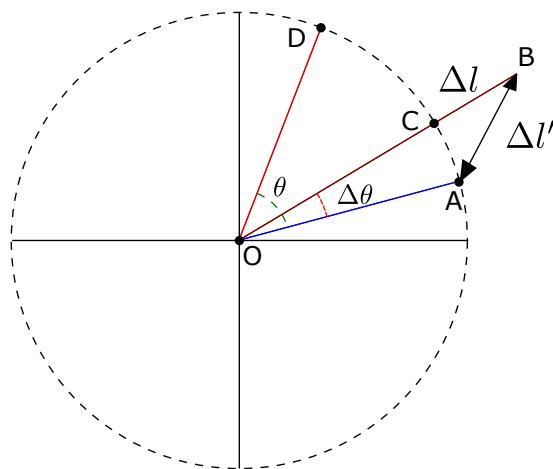
O incremento de mais termos SPT na definição do ângulo de rotação  $\theta_n$  acaba por dar ao conjunto dos ângulos elementares mais liberdade combinacional dentro do círculo unitário. No algoritmo Cordic tradicional e no EEAS havia uma dificuldade de realizar rotações maiores que  $\pi/4$ , devido a distribuição dos ângulos elementares, como pode ser visto nas Figura (13) e (14). Devido a esta dificuldade era necessário realizar uma pré-rotação no vetor de entrada, quando o ângulo de rotação era maior que  $\pi/4$ , afim de reduzir o ângulo para uma faixa mais aceitável de rotação. O que aumentava o custo computacional do algoritmo, e demandava mais recursos de hardware.

Porém no Algoritmo MSR o conjunto dos ângulos elementares no círculo unitário é maior e melhor distribuídos como visto na Figura (17). Isto possibilita ao MSR alcançar ângulos de rotação de  $0$  á  $2\pi$ , varrendo todo o círculo unitário.

O desempenho do algoritmo EEAS, como mostrado na seção anterior, é diretamente relacionado com a escolha dos parâmetros Coridc  $\mu_0$ ,  $\mu_1$ ,  $s_0$  e  $s_1$ . De maneira análoga é a determinação dos parâmetros Cordic  $J$ ,  $I$ ,  $\mu_i$ ,  $\mu_j$ ,  $s_i$  e  $s_j$  que garante a convergência e eficiência do algoritmo.

Para determinar os parâmetros Cordic do algoritmo MSR, Lin e Wu (2005) utiliza a análise do erro entre o vetor idealmente rotacionado, e o vetor que na prática do Algoritmo MSR consegue alcançar no rotacionamento. Para exemplificado con-

sidere que se deseja rotacionar um vetor  $\overrightarrow{OD}$  pelo ângulo  $\theta$  até a posição do ponto  $A$ , formando assim um novo vetor rotacionado  $\overrightarrow{OA}$ . Considerando que o MSR não consiga rotacionar o vetor  $\overrightarrow{OD}$  por exatamente  $\theta$ , e que devido ao ganho Cordic o módulo do vetor rotacionado seja diferente do vetor  $\overrightarrow{OD}$ . Assim a nova posição do vetor rotacionado seja  $\overrightarrow{OB}$ , como mostrado na Figura (18)(LIN; WU, 2005).



**Figura 18: MSR com I=2, J=1 e N=2**

Fonte: (LIN; WU, 2005)

Portanto analisando a Figura (18), o erro de aproximação  $\epsilon$  do algoritmo MSR é representado pelo trecho  $\overrightarrow{OB}$ , denominado  $\Delta l'$ . Já o erro angular é representado por  $\Delta\theta$ , e o erro do módulo é representado pelo trecho  $\overrightarrow{OC}$ , denominado  $\Delta l$ . Segundo Lin e Wu (2005) quando  $\Delta\theta$  é suficientemente pequeno é possível obter a seguinte expressão:

$$\Delta l'^2 = \overrightarrow{OA}^2 + \overrightarrow{OB}^2 - 2\overrightarrow{OA} \times \overrightarrow{OB} \cos(\Delta\theta) \quad (95)$$

$$\Delta l'^2 = 1 + (1 + \Delta l)^2 - 2 \times 1 \times (1 + \Delta l) \times \sqrt{1 - \sin^2(\Delta\theta)} \quad (96)$$

$$\Delta l'^2 \approx \Delta l^2 + \Delta\theta^2 \quad (97)$$

Por meio da Equação (97) é possível concluir que o erro angular  $\Delta\theta$ , e o erro modular  $\Delta l$  possuem o mesmo impacto no erro de aproximação do algoritmo MSR. Portanto afim de otimizar o desempenho do algoritmo MSR é necessário escolher um conjunto de parâmetros Cordic que privilegiem igualmente a redução de ambos os erros de aproximação (LIN; WU, 2005).

Assim determinar o conjunto dos melhores valores para os parâmetros Cordic  $J$ ,  $I$ ,  $\mu_i$ ,  $\mu_j$ ,  $s_i$  e  $s_j$  nada mais é do que uma tarefa de otimização, onde a equação

de minimização é a expressão do erro  $\epsilon$ , o qual pode ser definido como:

$$\min \epsilon = \min \sqrt{\Delta l^2 + \Delta \theta^2} \quad (98)$$

Para solucionar esta tarefa de otimização de  $\epsilon$  é possível utilizar diferentes tipos de algoritmos, e inclusive adaptar o algoritmo TBS já apresentado. Segundo Lin e Wu (2005) existe uma certa restrição quanto a utilização de algoritmos como o *greedy*, pois este tipo de algoritmo "guloso" não garante a melhor solução dentro do conjunto de soluções possíveis deste problema.

Um importante fator para a determinação do melhor conjunto de parâmetros Cordic para o MSR, além da minimização do erro  $\epsilon$ , é o efeito do ruído de arredondamento, ou também conhecido como *Roundoff Noise Analysis*(RNA). Nos dispositivos digitais o número de *bits* utilizados para representar sinais quantizados é limitado de acordo com a arquitetura. Este limite de representação acaba por introduzir ruídos nos sinais e consequentemente gera uma degradação do SQNR. w

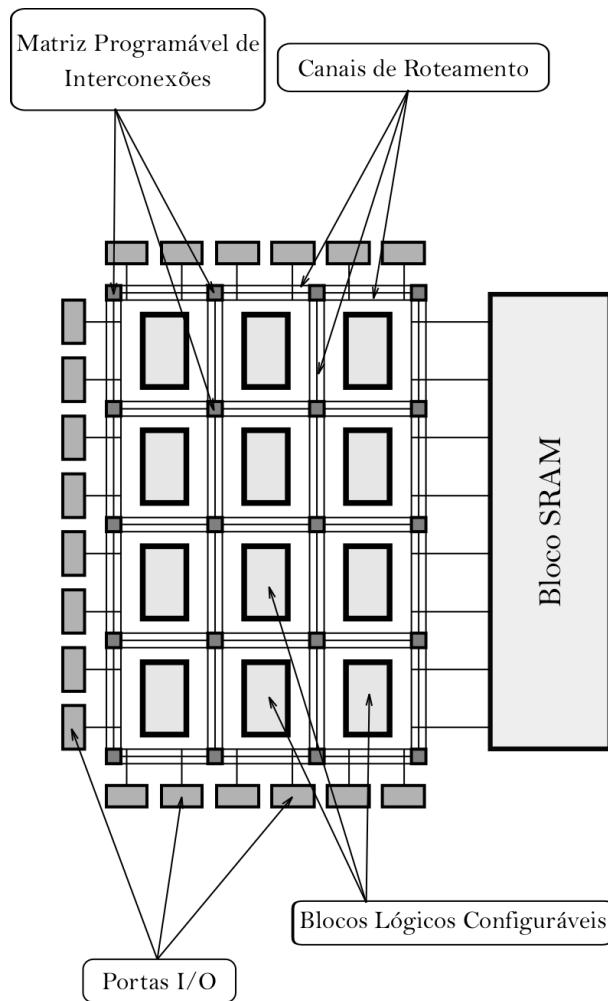
## 2.5 FPGA

Moore (2007, p. 4) define a FPGA como um dispositivo semicondutor capaz de ser totalmente redefinido após sua fabricação, permitindo ao programador reconfigurar produtos e funções já implementadas, adaptando o *hardware* a novas funções. De forma prática, a FPGA permite uma flexibilidade em um projeto, podendo mudar a forma como ele é implementado, sem a necessidade de se construir um *hardware* novo.

Para Moore (2007, p. 4), comparado com as outras formas de construir um hardware, a FPGA oferece duas grandes vantagens em uma aplicação. Primeiro, para uma aplicação ao invés de se utilizar um circuito integrado padrão comercial, que geralmente é super ou subdimensionado, ou ainda desenvolver um novo projeto de circuito integrado específico, consumindo tempo e recursos, a FPGA possibilita desenvolver um *hardware* exatamente dentro das especificações, personalizado e otimizado para a função destinada. Em segundo, porém tão importante quanto, é que essa capacidade de personalização de *hardware* possibilita a realização de operações de modo mais simplificado, rápido e energeticamente eficiente se comparado a um microprocessador.

### 2.5.1 ASPECTOS CONSTRUTIVOS DA FPGA

As FPGAs são baseadas em unidades lógicas elementares básicas, ou BLEs (*Basic Logic Elements*), dentro de uma hierarquia de interconexões reconfiguráveis que permitem que os LEs sejam fisicamente conectados uns aos outros de diferentes formas criando uma enorme variedade de componentes digitais. A arquitetura das FPGAs modernas são constituídas basicamente por conjunto de memórias de armazenamento em massa SRAM (*Static Random Access Memory*), Portas de Entrada/Saída, blocos lógicos configuráveis CLB (*Configurable Logic Blocks*) e sistema de roteamento, como pode ser visto na Figura (19) (MOORE, 2007, p. 5).

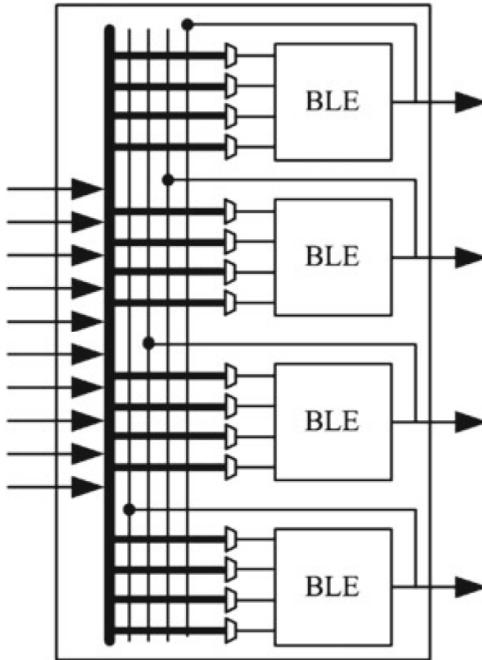


**Figura 19: Arquitetura Típica de uma FPGA**  
Fonte: Adaptado Meyer-Baese (2007, p. 6)

Os CLB são blocos que realizam operações lógicas básicas e armazenam pequenos volumes de dados. Comumente, as operações complexas, necessárias para

o processamento de uma aplicação, são divididas em processos mais simples para cada uma das CLBs selecionadas, de modo que a soma das tarefas de cada CLB seja equivalente a operação complexa, em uma estratégia de divisão e conquista. Para realizar operações lógicas básicas e ainda armazenar pequenos volumes de dados os CLBs tecnicamente poderiam ser apenas um pequeno circuito de transistores (granularidade fina), ou até mesmo um processador completo (granularidade grosseira). Se os CLBs fossem granularidade fina, para realizar tarefas complexas seria necessário um grande número de CLBs e um sistema de roteamento complexo para interconectá-los, o que resultaria em uma FPGA de baixa performance e um elevado consumo energético. Por outro lado de as CLBs forem de uma granularidade mais grosseira seria um desperdício de recurso utiliza-los em operações mais simples (FAROOQ *et al.*, 2012, p. 11). Assim a escolha do nível de complexabilidade, ou granulação, das CLBs de uma FPGA é um compromisso de otimização de recursos.

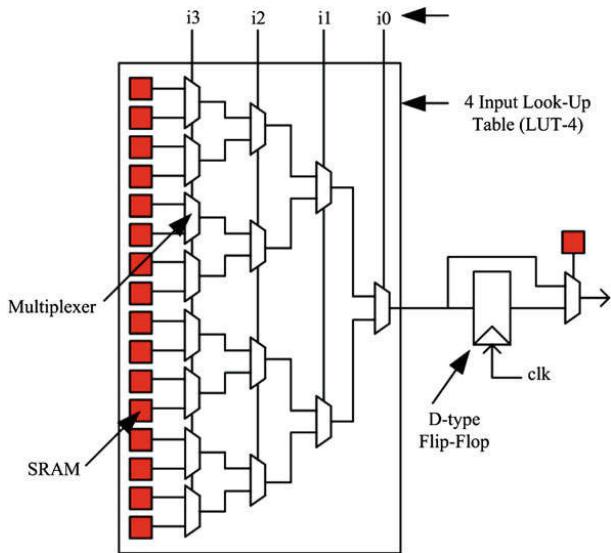
Ainda segundo Farooq *et al.* (2012, p. 11) dentro da grama de granulação das CLBs, algumas arquiteturas incluem o uso de portas NAND, interconexão de multiplexadores e tabelas de busca LUT (*Lookup Table*). Em especial fabricantes como a Xilinx utilizam CLBs baseadas em LUTs, já que CLBs baseadas em LUT oferecem uma boa relação de granulação, otimizando os recursos da FPGA para aplicações simples até as mais complexas. Este tipo de CLB pode incluir uma único elemento lógico básico BLE (*Basic Logic Element*), ou mesmo um *cluster* de BLEs interconectados, como mostrado na Figura (20).



**Figura 20:** Arquitetura de uma CLB com 4 BLEs

Fonte: Adaptado Farooq *et al.* (2012, p. 13)

Segundo Farooq *et al.* (2012, p. 11), um BLE mais simples consiste basicamente de um LUT e um *Flip-Flop* tipo D, como pode ser visto na Figura (21). Um LUT com  $k$  entradas pode implementar  $k$  funções booleanas utilizando os espaços de memória SRAM dentro da LUT. O exemplo apresentado na Figura (21) utiliza 16 bits de memória SRAM, os quais são conectadas a entrada do multiplexador que possui 4 bits de seleção, e cuja saída é ligada ao *flip-flop*. Esta configuração permite que a LUT tenha  $2^k$  combinações das  $k$  operações booleanas.



**Figura 21: Arquitetura de uma BLE (Basic Logic Element)**

Fonte: Adaptado Farooq et al. (2012, p. 13)

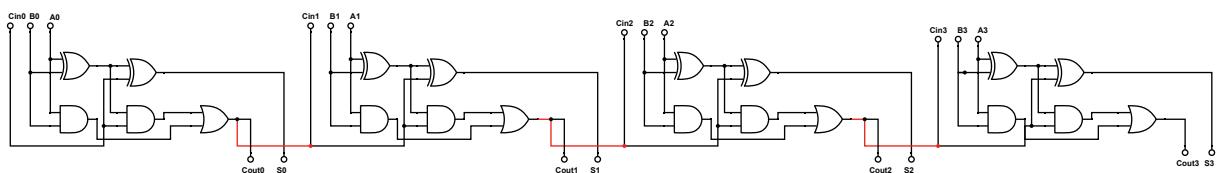
Um único BLE é capaz de realizar algumas operações booleanas básicas, porém em clusters as combinações de operações aumentam. FPGAs modernas tipicamente contém de 4 a 10 BLEs em um único cluster. Porém estas FPGAs não possuem apenas BLEs idênticas, na verdade há uma grande heterogeneia de blocos, sendo muitos deles desenvolvidos para propósitos específicos. Entre estes blocos de propósito específico estão multiplicadores, somadores, memórias e DSPs (*Digital Signal Processor*), entre outros. Estes blocos são desenvolvidos para otimizar o espaço, processamento, roteamento e demais recursos de *hardware* que seriam necessários para implementar as mesmas funções em BLEs comuns, sendo essenciais em certas aplicações Farooq et al. (2012, p. 10).

A implementação de qualquer circuito lógico é feita pela associação de diferentes blocos lógicos e pelas portas de entrada e saída da FPGA, os quais são conectados uns aos outros através da rede de roteamento programável, ou PLN (*Programmable Logic Network*). Na Figura (19) a PLN é representada pela Matriz Programável de Interconexões e pelos Canais de Roteamento. Para que a FPGA possa implementar qualquer circuito digital as interconexões de roteamento devem ser flexíveis para suportar a grande variedade de conexões demandada, otimizando sempre as distâncias das conexões e reduzindo a latência dos sinais. Portanto ao projetar um circuito a ser implementado na FPGA deve ser ter especial atenção a forma como o roteamento dos blocos lógicos é feito, buscando flexibilidade e eficiência Farooq et al. (2012, p. 13).

Nas FPGAs modernas além da unidades de armazenamento de Dados SRAM contido dentro das BLEs, mais especificamente nas LUTs, existe ainda grandes blocos SRAM isolados das BLEs, destinados a funcionar como o armazenamento de dados em massa. Estes blocos são importantes em aplicações digitais onde é necessário armazenar, como por exemplo, dados de amostragem ou mesmo dados pós-processamento que devem aguardar para serem passados para uma próxima etapa de processamento, ou mesmo transmitidos para fora da FPGA pelas portas de entrada e saída de dados. Estes bloco de memória é apresentada na Figura (19) como parte integrante da arquitetura tipica de uma FPGA.

### 2.5.2 PROGRAMAÇÃO NA FPGA

O desenvolvimento de uma aplicação em FPGA começa pela elaboração do Design de Referência, que nada mais é do que uma descrição lógica equivalente que deve ser programada na FPGA para implementação das operações lógicas desejadas. O Design de Referência pode ser feito utilizando diagrama de portas lógicas ou ainda usando qualquer linguagem de descrição de hardware como VHDL (*VHSIC Hardware Description Language*) ou Verilog. A maioria dos ambientes de desenvolvimento integrados (IDE - *Integrated Development Environment*), disponibilizados pelos fabricantes de FPGAs possuem a opção de programação visual utilizando portas lógicas. A Figura (22) apresenta o diagrama de um somador de 4 bits, feito no IDE ISE Design Suite 14, da fabricante Xilinx.



**Figura 22: Diagrama Lógico Full Adder 4 Bits - ISE Design Suite 14**  
Fonte: Autoria Própria

Programar um somador de 4 bits como apresentado na Figura (22), utilizando portas lógicas parece ser realmente simples. Porém para circuitos mais elaborados como um contador, ou até mesmo uma máquina de estados, pode ser tornar impraticável utilizar este método de desenvolvimento. Para circuitos mais elaborados é possível utilizar uma linguagem de descrição de hardware HDL (*Hardware Description Language*), para tornar o desenvolvimento mais fácil e intuitivo. Tendo em vista ainda que uma linguagem descritiva, como por exemplo VHDL, é visivelmente mais fami-

liar ao desenvolvedor que está acostumado a linguagens de programação como por exemplo Linguagem C. O Código (2.5.2) descrito abaixo apresenta o mesmo somador de 4 bits desenvolvido em VHDL.

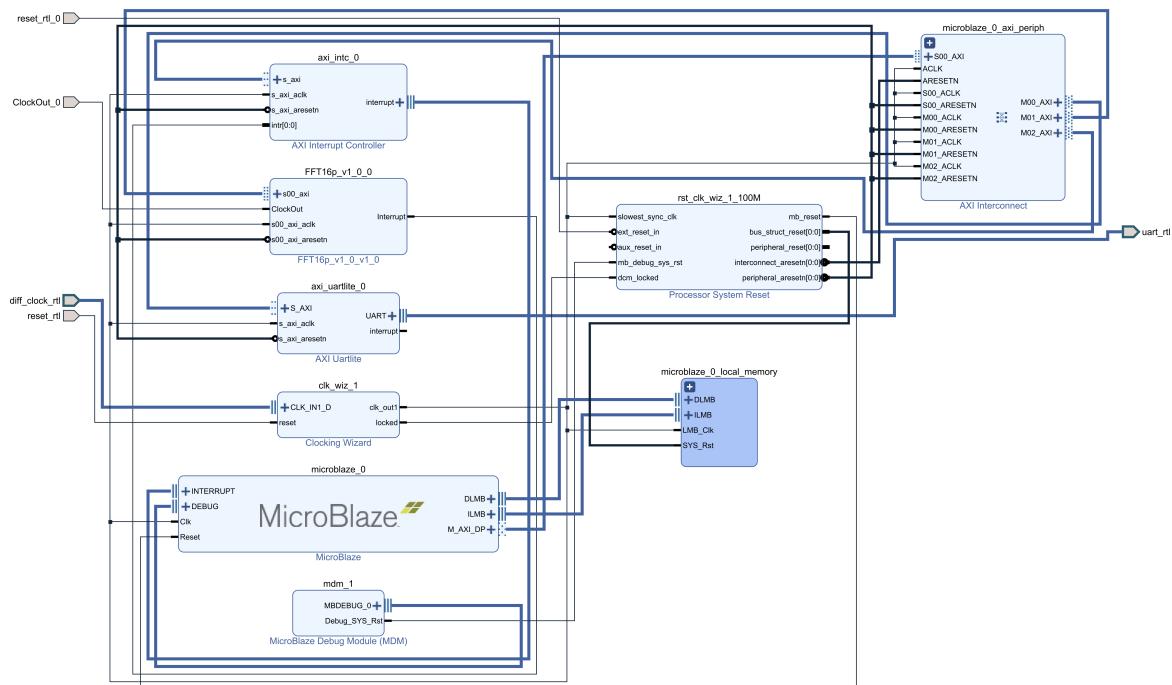
```
1 library IEEE;
2 use IEEE.STD.LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity FullAdder4Bits is
6   port(InputA : in unsigned(3 downto 0);
7        InputB : in unsigned(3 downto 0);
8        Result : out unsigned(3 downto 0);
9        CarryOut : out std_logic);
10 end entity;
11
12 architecture Behavioral of FullAdder4Bits is
13
14   signal Aux : unsigned(4 downto 0);
15
16 begin
17
18   Aux <= ("0" & InputA) + InputB;
19   Result <= temp(3 downto 0);
20   CarryOut <= Aux(4);
21
22 end architecture Behavioral;
23
```

Após definir o Design de Referência, via diagrama de portas lógicas ou mesmo por código HDL, o próximo passo é utilizar uma ferramenta de síntese da própria IDE para converter o design de referência em um conjunto de configurações de registradores, conexões e portas que serão usadas na FPGA para implementar as funcionalidades descritas no design de referência. Durante o processo de síntese a ferramenta verifica a sintaxe do código HDL, e a coerência entre as portas de externas, como sinais de *clock* e portas de entrada e saída, selecionadas no design de referência.

Ao desenvolver um design para implementação em FPGA é comum dividir as funcionalidades do sistema em pequenos blocos de modo a modularizá-lo, permitindo reaproveitar trechos de circuitos lógicos em diferentes aplicações. Segundo Moore (2007, p. 20) ao longo dos anos os fabricantes FPGA perceberam também que

vários dos sistemas implementados pelos desenvolvedores tinham funcionalidades muito comuns, como por exemplo processamento gráfico, interfaces de comunicação serial e até mesmo implementação de microprocessadores. Logo não fazia sentido o desenvolvedor desperdiçar tempo implementando um circuito extremamente comum. Assim os fabricantes passaram a oferecer bibliotecas circuitos lógicos modularizados para funcionalidades comuns, que passaram a ser chamados de IP (*Intellectual Property*).

A maioria dos IDEs mais recentes possuem uma interface gráfica de diagrama de blocos, onde cada bloco representa uma IP. Nesta interface é possível construir um Design de Referência utilizando a associação de IPs das bibliotecas do fabricante, ou de um repositório externo ou ainda utilizar uma IP própria, já que estes IDEs possibilitam a criação de IPs personalizadas. A Figura (23) apresenta o diagrama de blocos de um circuito lógico desenvolvido na IDE Vivado 2017.4, composto pelas seguintes IPs da Biblioteca padrão Xilinx: microprocessador MicroBlaze 10.0, bloco seu memória local, o controlador de periféricos MicroBlaze, a interface de comunicação UART, controle global de interrupções e controle de global de reset. E ainda IP *FFT16p\_V1\_0\_0* desenvolvida individualmente utilizando código VHDL.



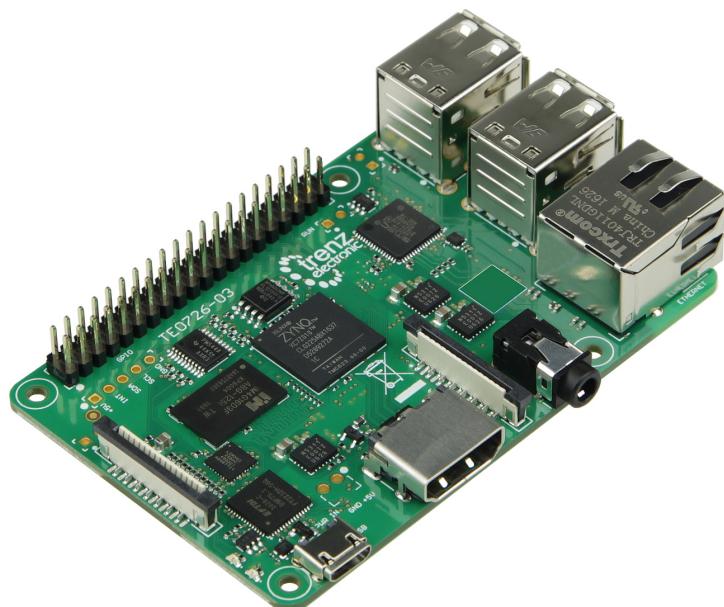
**Figura 23: CPU MicroBlaze e Coprocessador para FFT com Interface UART - Vivado 2017.4**  
Fonte: Autoria Própria

Para ??), a FPGA é uma boa escolha para a implementação do algoritmo

da FFT devido a grande variedade de recursos de hardware sintetizáveis, além de possuir recursos de programação paralela que permite o processamento paralelo de sinais, conferindo assim uma maior rapidez na execução do algoritmo. Como afirma Meyer-Baese (2007, Prefácio), muitos algoritmos de processamento de sinais, como FFT (*Fast Fourier Transform*) e os filtros FIR ou IIR, implementados anteriormente em Circuitos Integrados de Aplicação Específica ou ASIC (*Application Specific Integrated Circuits*), agora estão sendo implementados em FPGAs.

### 2.5.3 ZYNQBERRY TE0726-03M

O dispositivo escolhido para a implementação do algoritmo Radix-2 é o *ZynqBerry - TE0726* da fabricante *Trenz Eletronic®*, apresentado na figura (24). Este dispositivo é baseado no SoC (*System On Chip*) Raspberry Pi modelo 2, vem equipado com uma FPGA SoM (*System on Module*) XC7Z010-1CLG225C-REV3, da família Zynq-700 fabricado pela *Xilinx®* (ELETRONIC, 2018).



**Figura 24: ZynqBerry - TE0726**  
Fonte: Eletronic (2018)

A FPGA XC7Z010-1CLG225C tem a mesma arquitetura dos modelos da família Artix-7, também da Xilinx, e contendo recursos como: 28.000 células lógicas, 17.600 LUTs, 2.1 Mb de memória RAM divididos em blocos de 26Kb e um total de 36.200 *flip-flops*. Cada CLB, nesta FPGA, para implementar diferentes operações lógicas utiliza 16 *flip-flops*, 2 somadores de 4bits cascateáveis e ainda 8 LUTs. Sendo

possível configurar a memoria RAM das LUTs para 64x1 ou 32x2 bits, ou ainda como um *shift register (SRL)*. Além disso esta FPGA possuir 80 blocos DSP, cada um equipado com um multiplicador 18x25 simples, e um somador/acumulador de 48 bits. Todas estes recursos fazem do XC7Z010 um *hardware* competente para as mais diversas aplicações de processamento de sinais, como o calculo da FFT.

O processador utilizado no ZynqBerry TE0726-03M é um dual-core ARM Cortex-A9 de 866MHz é competente na execução eficiente de sistemas operacionais completos como o sistemas baseados em kernel Linux, que podem incluem interface gráfica sofistica. Cada núcleo deste processador conta ainda uma unidade NEON™Media Processing Engine (MPE), para alto desempenho em codificação e decodificação de áudio e vídeo, e uma unidade de ponto flutuante para incremento da precisão em operações matemáticas. Aliando a versatilidade da XC7Z010 e o poder de processamento do ARM Cortex-A9 é possível construir dispositivo que rode uma versão Linux, tirando proveito de toda a funcionalidade de tal sistema operacional pode prover, e que ainda disponha de um *hardware* acelerador personalizado para uma aplicação especifica, e que trabalhe em paralelo á uma alta frequência. Provendo assim uma solução engenhosa de alto desempenho para processamento de sinais.

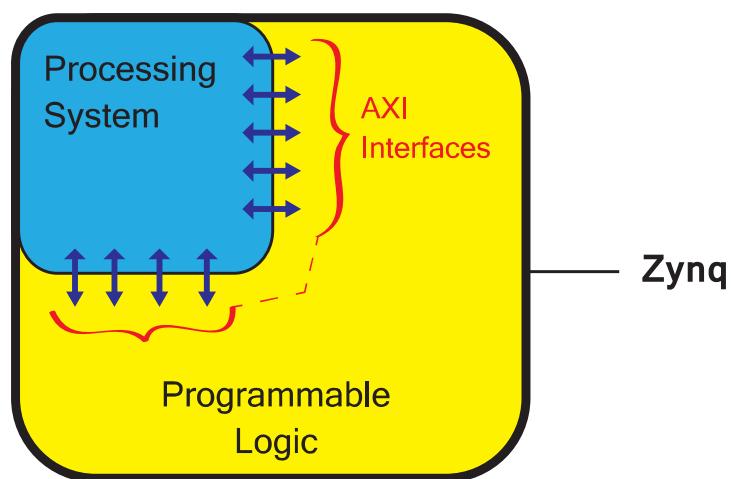
#### 2.5.3.1 ZYNQ-7000

Segundo a Crockett *et al.* (2014), Zynq-7000 é uma família de SoCs que integram a programabilidade em *software* de um processador ARM Cortex-A9, com a programabilidade em *hardware* de uma FPGA, possibilitando a integração entre CPU, DSPs e FPGA, agregando diversas funcionalidades em um único dispositivo. Zynq-7000 representa uma solução completa em processamento de sinais em um único equipamento, com um ótima relação performance/consumo energético.

A principal característica do Zynq-7000 é a forma com que ele combina um sistema de processamento (PS - *Processing System*), formado pelo entorno do processador ARM Cortex-A9, e um sistema de lógico programável (PL - *Programmable Logic*), caracterizado como um FPGA. Processadores dedicados já tem sido utilizados em conjunto com FPGAs em diferentes aplicações, porém não da mesma forma como é feita na familia Zynq-7000 Crockett *et al.* (2014, Introdução).

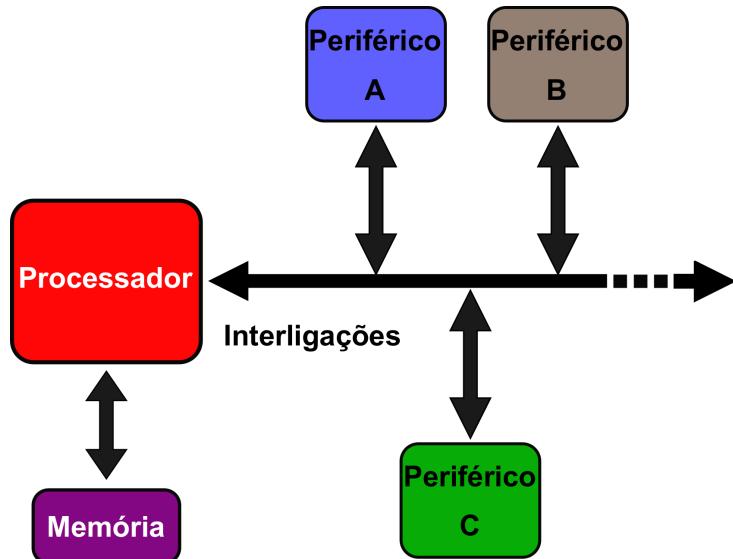
Segundo Para Crockett *et al.* (2014, p. 26) o PL é ideal para a implementação de operações lógicas de alta performance e sistemas de fluxo de dados contínuos. Por outro lado o PS é capaz de suportar rotinas de *software* e sistemas operacionais.

Qualquer aplicação pode ser partitionada em duas partes a serem implementadas uma em PL e outra em PS, afim de se tirar proveito do melhor dos dois mundos. Porém estas duas partes, mesmo que estando contidas dentro do encapsulamento do Zynq, como pode ser visto na Figura 25, são fisicamente distintas, e comumente estão operando em frequências diferentes. Para realizar a ponte de comunicação entre o PL e o PS, a família Zynq-7000 utiliza o padrão industrial conhecido como AXI (*Advanced eXtensible Interface*), ou interface extensível avançada. Esta interface permite estabelecer um fluxo de dados sincronizados entre PS e PL, de ambos os sentidos, suportando inclusive o disparo de interrupções através de ambos os sistemas.



**Figura 25: Arquitetura Simplificada - Zynq-7000**  
Fonte: Crockett *et al.* (2014, p. 26)

Para entender como os componentes de um sistema digital são mapeados dentro de um dispositivo Zynq, e como estes são divididos entre o PS eo PL, é necessário compreender como é a arquitetura de um sistema digital comum. Para Crockett *et al.* (2014, p. 27), o modelo básico do *hardware* de um sistemas digitais incorpora processadores, memórias, barramentos de interligação e os mais distintos periféricos. Como pode ser visto na Figura (26).



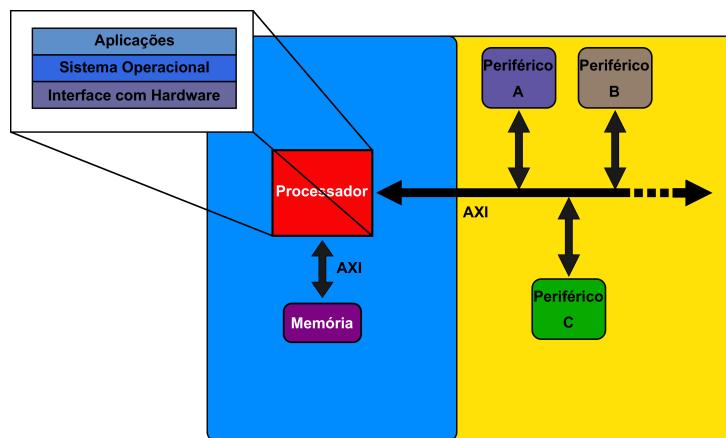
**Figura 26: Arquitetura Simplificada de um Sistema Digital**  
Fonte: Crockett et al. (2014, p. 27)

O processador é o elemento central deste modelo, pois é ele que executa o sistema de *software*, que compreende as camadas de mais alto nível como aplicações baseadas em sistemas operacionais, o próprio sistema operacional, e até o nível mais baixo como o *firmware* de interface com os periféricos do *hardware*. Já os periféricos são componentes funcionais externos ao processador, e que em geral são divididos em três tipos:

- **Coprocessadores** : Elementos que auxiliam o processador principal na realização de tarefas específicas, geralmente projetados para otimizar tal tarefa.
- **Interfaces de Comunicação**: Elementos responsáveis pela interação com interfaces externas, acionando gatilhos ou controlando portas digitais. Utilizando muitas vezes protocolos específicos de comunicação como UART ou SPI.
- **Elementos Adicionais de Memória** : Elementos exclusivamente destinados ao armazenamento de dados.

A Figura (27) apresenta a mesma arquitetura simplificada da Figura (26), porém mapeado para um dispositivo Zynq. A estrutura do sistema digital é dividida entre processador e memória para o lado PS, e os demais possíveis periféricos para o lado PL. Do lado PS a arquitetura é fixa, obedecendo a estrutura definida pelo fabricante, em total contraponto com o lado PS. No lado PS a estrutura é totalmente flexível, limitada apenas pelo número de CLBs disponíveis da FPGA, o que oferece

ao desenvolvedor um ambiente de "caixa de areia" para construir qualquer tipo de periférico.



**Figura 27: Arquitetura Simplificado do um Sistema Digital Mapeado para o Zynq**

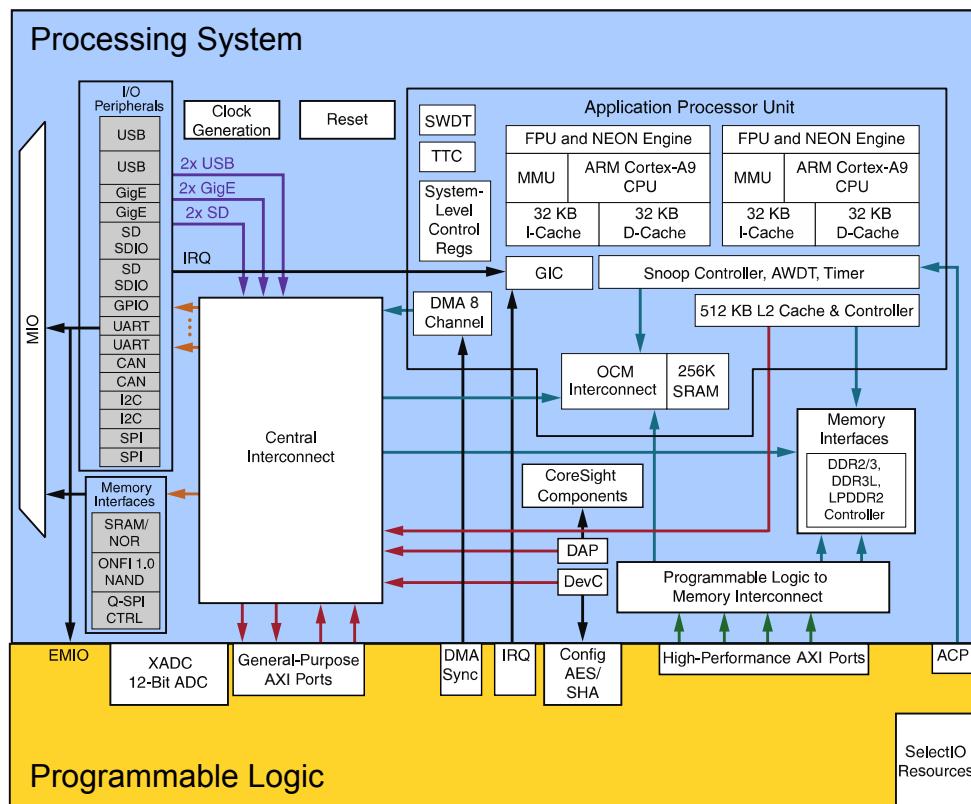
Fonte: Crockett et al. (2014, p. 27)

A interface AXI possui uma grande importância no desenvolvimento no Zynq-7000, já que por meio desta interface é que os periféricos em PL se comunicam com o processador em PS. A interface AXI faz parte da família de barramentos para microcontroladores ARM AMBA (*Advanced Microcontroller Bus Architecture*). O ARM AMBA é um protocolo *Open Standard* para conexões e gerenciamento de blocos funcionais dentro de dispositivos *Systems-on-Chip* (Soc), facilitando o desenvolvimento de designs com múltiplos processadores, e com grande número de controladores e periféricos (ARM, 2018). A família de SoCs Zynq-7000 utilizam a interface AXI versão 4, o qual obedece ao mais recentes padrões ARM AMBA 4.0 (XILINX, 2017). Existem no três tipos de interface AXI4:

- **AXI** : Interface destinada a transferências de dados em alta velocidade e de maior volume utilizando mapeamento de memória, utiliza endereços de memória para acessar dados e portanto consome recursos de memória para ser implementado. Interface mais complexa, e oferece maiores opções de controle de dados, inclusive transferência no modo *burst*.
- **AXI-Lite**: Interface destinada a transferências de dados em baixa velocidade utilizando mapeamento de memória, consome espaço de memória apenas para controlar dados da transferência, como destino, origem e status da transmissão. Interface mais simples do que a AXI, logo não oferece controle de dados como o modo *burst*.

- **AXI-Stream:** Interface para transferência de dados em alta velocidade, porém não utiliza mapeamento de memória. Toda a transferência nesta interface é feita por fluxo de dados contínuos, os quais não são armazenados pela interface.

Dentro do ambiente do PS no Zynq além do processador ARM Cortex-A9 existe ainda um conjunto de periféricos de memória, interconexão, comunicação e gerenciamento. A maioria destes periféricos utiliza a interface AXI, com 32 ou 64 Bits, inclusive a próprias conexões de fronteira entre PS e PL. Diferente de PS onde a arquitetura já está pronta e é estática, em PL o desenvolvedor pode criar todo uma gama de periféricos, e consequentemente conectá-los com uma quantidade enorme de interfaces AXI de qualquer tipo dos já citados. Porém a fronteira entre PS e PL é limitada e representa um gargalo na interação entre os dois lados. Os dispositivos da família Zynq-7000 possuem 2 interfaces AXI Mestre de 32-Bits, 2 interfaces AXI Escravo de 32-Bits e 4 interfaces 64-Bit/32-Bit configuráveis de alta velocidade. A Figura (28) apresenta uma visão geral da arquitetura do SoC Zynq-700, mostrando as conexões AXI dentro de PS e também as conexões na fronteira de PL.



**Figura 28: Visão Geral da Arquitetura - Zynq 7000**

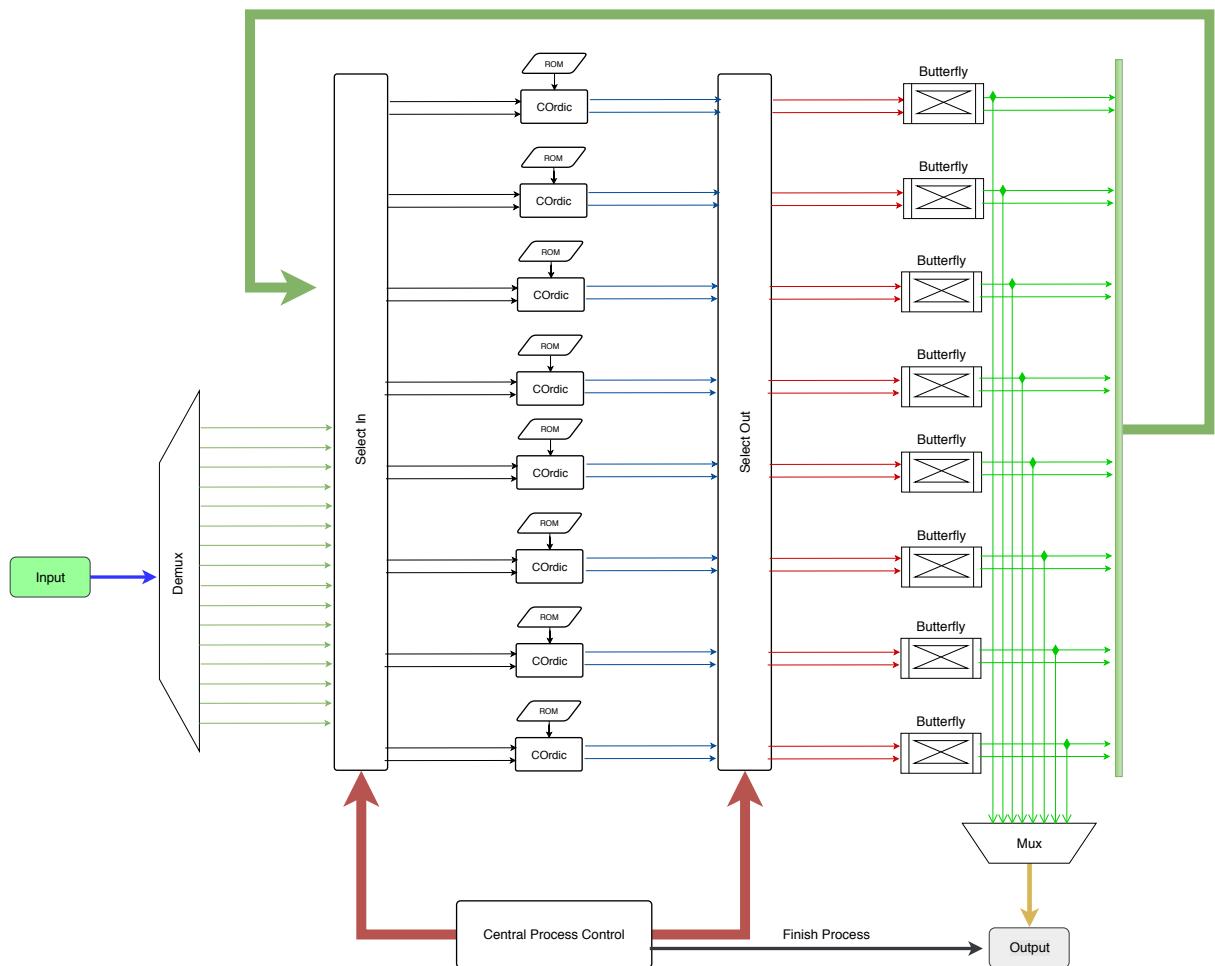
Legenda: **AXI 32-Bits/64-Bits**, **AXI 64-Bits**, **AXI 32-Bits**, **AHB 32-Bits**, **AXI 32-Bits**

Fonte: Adaptado de (XILINX, 2018)

### 3 IMPLEMENTANDO PROCESSADOR CORDIC

## 4 IMPLEMENTANDO A FFT NO ZYNQBERRY

### 4.0.1 IMPLEMENTANDO FFT 32 PONTOS

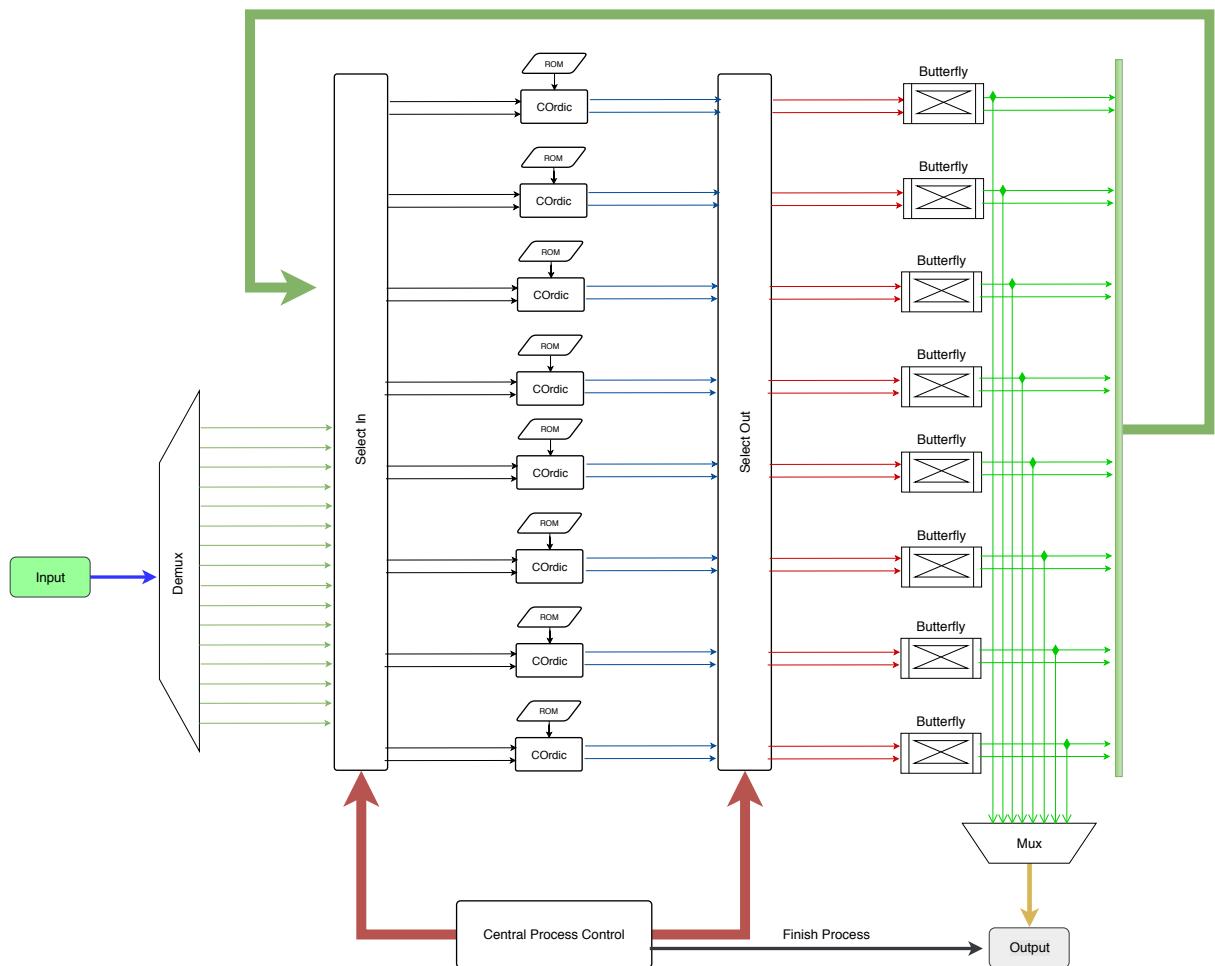


**Figura 29: Arquitetura Implementada FFT de 16 Pontos**  
Fonte: Autoria Própria

### 4.0.2 IMPLEMENTANDO FFT 1024 PONTOS

## 5 ANÁLISE DOS RESULTADOS

### 5.0.1 IMPLEMENTANDO FFT 32 PONTOS



**Figura 30: Arquitetura Implementada FFT de 16 Pontos**  
Fonte: Autoria Própria

### 5.0.2 IMPLEMENTANDO FFT 1024 PONTOS

## 6 CONCLUSÃO

**Tabela 3: Exemplo de Tabela**

Posição	Município	População
1	São Paulo	11.376.685
2	Rio de Janeiro	6.690.290
3	Salvador	2.710.968
4	Brasília	2.648.532
5	Fortaleza	2.500.194

**Fonte:** (??)

### 6.0.1 TESTES E EXEMPLOS DE LISTA DE SIGLAS

Testes da lista de siglas:

(TDLS1).

(EASREFER).

(ER).

Esta é uma sigla em que os parênteses não aparecem: SQPNA.

(DMA).

É preciso rodar duas vezes o Latex para que a lista de siglas seja atualizada.

### 6.0.2 TESTES E EXEMPLOS DE LISTA DE SÍMBOLOS

$$\phi = \vec{\alpha} \otimes v\omega\psi_{n-1}^{jk} \quad (99)$$

Onde:

$\phi$ : Angulo phi.

$\vec{\alpha}$ : Alfa.

$v\omega\psi_{n-1}^{jk}$ : Um simbolo grande.

Este símbolo ( $\nabla$ ) foi inserido no texto, fora do ambiente *equation*.

$$\nu\omega\psi_{n-1}^{jk}$$

### 6.0.3 EXEMPLOS DE CITAÇÃO DE REFERÊNCIAS BIBLIOGRÁFICAS

(??)

(????)

Conforme ??) a metodologia proposta é, bla bla bla...

(??)

## **7 NÍVEL 1 - TESTES DE CAPITULAÇÃO - PRIMÁRIO**

### **7.1 NÍVEL 2 - SECUNDÁRIO**

#### **7.1.1 NÍVEL 3 - TERCIÁRIO**

##### **7.1.1.1 NÍVEL 4 - QUATERNÁRIO**

###### **7.1.1.1.1 NÍVEL 5 - QUINÁRIO**

## REFERÊNCIAS

- AMARAL, Carlos Eduardo Ferrante do; LOPES, Heitor S; ARRUDA, Lúcia V; HARA, Marcos S; GONçALVES, Antonio J; DIAS, Adilson A. Design of a complex bioimpedance spectrometer using dft and undersampling for neural networks diagnostics. **Medical engineering & physics**, Elsevier, v. 33, n. 3, p. 356–361, 2011.
- ARM. **System IP AMBA Specifications**. 2018. <https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications>. Acessado em 13 de Agosto de 2017.
- BINGHAM, J. A. C. Multicarrier modulation for data transmission: an idea whose time has come. **IEEE Communications Magazine**, v. 28, n. 5, p. 5–14, May 1990. ISSN 0163-6804.
- CHU, Eleanor; GEORGE, Alan. **Inside the FFT black box: serial and parallel fast Fourier transform algorithms**. 1. ed. [S.I.]: CRC Press, 1999.
- CORMEN, Thomas H; LEISERSON, Charles E; RIVEST, Ronald L; STEIN, Clifford. **Algoritmos: teoria e prática**. 2. ed. [S.I.]: Editora Campus, 2002.
- CROCKETT, Louise H; ELLIOT, Ross A; ENDERWITZ, Martin A; STEWART, Robert W. The zynq book. **Strathclyde Academic Media**, 2014.
- DESPAIN, Alvin M. Fourier transform computers using cordic iterations. **IEEE Transactions on Computers**, IEEE, v. 100, n. 10, p. 993–1001, 1974.
- EL-MOTAZ, M. A.; NASR, O. A.; OSAMA, K. A cordic-friendly fft architecture. In: **2014 International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.I.: s.n.], 2014. p. 1087–1092. ISSN 2376-6492.
- ELETTRONIC, Inc Trenz. **TE0726 - ZynqBery**. San Jose, EUA, Julho 2018. V2.1.
- FAROOQ, Umer; MARRAKCHI, Zied; MEHREZ, Habib. **Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization**. [S.I.]: Springer Science & Business Media, 2012.
- GARRIDO, M.; KÄLLSTRÖM, P.; KUMM, M.; GUSTAFSSON, O. Cordic ii: A new improved cordic algorithm. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 63, n. 2, p. 186–190, Feb 2016. ISSN 1549-7747.
- HAYKIN, SIMON S; VEEN, Barry Van. **Sinais e sistemas**. [S.I.]: Bookman, 2001.
- HE, Hongjiang; GUO, Hui. The realization of fft algorithm based on fpga co-processor. In: **IEEE. Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on**. [S.I.], 2008. v. 3, p. 239–243.

- IBRAHIM, Muhammad; KAMAL, Mohsin; KHAN, Omar; ULLAH, Khalil. Analysis of radix-2 decimation in time algorithm for fpga co-processors. Computing, Electronic and Electrical Engineering (ICE Cube), 2016 International Conference on, 2016.
- KUO, Jen-Chih; WEN, Ching-Hua; LIN, Chih-Hsiu; WU, An-Yeu (Andy). Vlsi design of a variable-length fft/ifft processor for ofdm-based communication systems. **EURASIP Journal on Advances in Signal Processing**, v. 2003, n. 13, p. 439360, Dec 2003. ISSN 1687-6180. Disponível em: <<https://doi.org/10.1155/S1110865703309060>>.
- LATHI, Bhagwandas Pannalal. **Sinais e Sistemas Lineares**. 2. ed. [S.I.]: Brasil: Bookman, 2007.
- LIN, Chih-Hsiu; WU, An-Yeu. Mixed-scaling-rotation cordic (msr-cordic) algorithm and architecture for high-performance vector rotational dsp applications. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 52, n. 11, p. 2385–2396, Nov 2005. ISSN 1549-8328.
- LIU, C.; MOREIRA, P.; ZEMITI, N.; POIGNET, P. 3d force control for robotic-assisted beating heart surgery based on viscoelastic tissue model. In: **2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society**. [S.I.]: s.n., 2011. p. 7054–7058. ISSN 1094-687X.
- MARTINSEN, Orjan G; GRIMNES, Sverre. **Bioimpedance and bioelectricity basics**. [S.I.]: Academic press, 2011.
- MEHER, P. K.; VALLS, J.; JUANG, T. B.; SRIDHARAN, K.; MAHARATNA, K. 50 years of cordic: Algorithms, architectures, and applications. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 56, n. 9, p. 1893–1907, Sept 2009. ISSN 1549-8328.
- MEYER-BAESE, Uwe. **Digital signal processing with field programmable gate arrays**. 3. ed. [S.I.]: Springer, 2007.
- MOORE, Andrew. **FPGAs for Dummies Altera Special Edition**. [S.I.]: Wiley Brand, 2007.
- OPPENHEIM, Alan V; WILLSKY, Alan S. **Sinais e Sistemas**. 2. ed. [S.I.]: Brasil:Pearson, 2010.
- TRADINGVIEW. **Índice BM & FBOVESPA**. 2017. <https://www.tradingview.com/symbols/BMFBOVESPA-IBOV/>. Acessado em 14 de Agosto de 2017.
- TRIANTIS, Iasonas F; DEMOSTHENOUS, Andreas; RAHAL, Mohamad; HONG, Hongwei; BAYFORD, Richard. A multi-frequency bioimpedance measurement asic for electrical impedance tomography. In: IEEE. **ESSCIRC (ESSCIRC), 2011 Proceedings of the**. [S.I.], 2011. p. 331–334.
- VANMATHI, K; SEKAR, K; RAMACHANDRAN, Remya. Fpga implementation of fast fourier transform. In: IEEE. **Green Computing Communication and Electrical Engineering (ICGCC), 2014 International Conference on**. [S.I.], 2014. p. 1–5.
- VOLDER, J. E. The cordic trigonometric computing technique. **IRE Transactions on Electronic Computers**, EC-8, n. 3, p. 330–334, Sept 1959. ISSN 0367-9950.

WANG, B.; ZHANG, Q.; AO, T.; HUANG, M. **Design of Pipelined FFT Processor Based on FPGA**. Jan 2010. 432-435 p.

WU, Cheng-Shing; WU, An-Yeu. A novel trellis-based searching scheme for e eas-based cordic algorithm. In: **2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)**. [S.l.: s.n.], 2001. v. 2, p. 1229–1232 vol.2. ISSN 1520-6149.

WU, Cheng-Shing; WU, An-Yeu; LIN, Chih-Hsiu. A high-performance/low-latency vector rotational cordic architecture based on extended elementary angle set and trellis-based searching schemes. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 50, n. 9, p. 589–601, Sept 2003.

XILINX. **AXI Reference Guide**. 2017. [https://www.xilinx.com/support/documentation/ip\\_documentation/ug761\\_axi\\_reference\\_guide.pdf](https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf). Acessado em 15 de Agosto de 2017.

XILINX. **Zynq-7000 SoC Data Sheet: Overview**. 2018. [https://www.xilinx.com/support/documentation/data\\_sheets/ds190-Zynq-7000-Overview.pdf](https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf). Acessado em 16 de Agosto de 2017.

ZHOU, Bin; PENG, Yingning; HWANG, David. Pipeline fft architectures optimized for fpgas. **International Journal of Reconfigurable Computing**, Hindawi Publishing Corp., v. 2009, p. 1, 2009.

## ANEXO A - STELLAR MYSTERY SOLVED, EINSTEIN SAFE

For more than 30 years, Villanova University astronomer Ed Guinan has been plagued, puzzled, and perplexed by DI Herculis. On the surface, this binary star seems pretty much like any other binary star, with two stars going 'round and 'round each other in a predictable, orderly fashion. But there remained a nagging problem that as much as Guinan wanted, he couldn't just sweep under the rug: DI Her was not behaving in accordance with Einstein's general theory of relativity.

Every year, Guinan and his colleagues would observe the 8.5 - magnitude star. The two stars orbit each other in a plane lined up perfectly with Earth's line of sight, and they eclipse each other every 10.55 days. Thanks to these eclipses, which have been recorded since 1900, Guinan could make exceedingly precise measurements of the stars' masses, sizes, luminosities, and orbital characteristics.

Almost every known aspect of the system was hunky-dory. The stars are exactly as massive, large, and bright as theory predicts. But in a series of published papers, Guinan and his Villanova colleague Frank Maloney kept pointing out that the orbit was not behaving in accordance with general relativity, the cornerstone for modern science's understanding of gravity.

For years, Guinan looked for a third star in the system, or some other factor that could be throwing the orbit out of whack, but to no avail. The point in the orbit at which the two stars come closest continues to advance (precess) each orbital cycle at a rate only one-fourth the amount theory predicts. Guinan continually receives mail from armchair theorists who attempt to explain DI Her's anomalous precession with alternative theories of gravity.

Finally, after decades of frustration, a group led by Simon Albrecht (MIT) has solved the problem, and Einstein's theory has survived unscathed. "The monkey has been lifted off my back," jokes Guinan.

After being informed about DI Her by Guinan, Albrecht and his colleagues took detailed spectra in 2008 using a 1.93-meter telescope in France. These measurements, unlike earlier ones made by Guinan and others, were obtained with a high-resolution spectrograph. With more modern equipment, computers, and techniques, Albrecht revealed that the rotation axes of the two stars are tipped over on their sides with respect to the orbital plane (similar to Uranus's orientation with respect to the Sun), which makes DI Her an oddity among closely separated binary stars.

In this unusual arrangement, gravitational forces created by the misaligned

equatorial bulges of the stars give them an extra "kick" when they are closest in their elliptical (oval-shaped) orbits. This kick reduces the orbital precession to the observed rate, eliminating any discrepancies with Einstein's theory. "We consider the mystery of the anomalous orbital precession of DI Herculis to be solved," says Albrecht, whose paper will appear in tomorrow's *Nature*.

Guinan concurs, but he adds, "I'm relieved that the general relativity problem is solved. But it's been replaced with a new mystery."

DI Her's two members, both hot, massive, B-type stars, should have formed from a common disk of gas and dust. By feeding from the same disk, the spin axes of the two stars should be nearly perpendicular to the orbital plane, an arrangement seen in the large majority of binary stars, especially those like DI Her that have small separations.

"My guess is that there was a third-body interaction that perturbed the system, or the two stars formed separately and formed a binary through a capture process," says Guinan. "Another possibility is that our ideas of how binaries form may be off."

Together with MIT colleague Joshua Winn, Albrecht is observing other binary systems that display similar anomalies. "We want to understand if DI Her is a unique system or if misalignment is more common among close systems," says Albrecht. "We also want to understand what might cause this misalignment. Clearly, there is more going on than we guessed so far."

Even if Einstein's theory remains safe and sound, astronomers and physicists continue to subject it to more stringent tests. Given general relativity's incompatibility with quantum mechanics (the theory that describes the microworld of atoms and light with extraordinary accuracy), scientists expect that Einstein will not have the last word on gravity. Ultimately, it will be replaced by an even deeper "theory of everything."

## A.1 EXEMPLO DE SEÇÃO ANEXO

Esta é uma seção, dentro dos anexos.