UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO DE CIÊNCIAS SOCIAIS E TECNOLÓGICAS
MESTRADO EM ENGENHARIA ELETRÔNICA E COMPUTAÇÃO

ANDRÉ NEVES SAPPER

Exploração do Algoritmo CORDIC para a Redução de Potência em Arquiteturas de Transformada Rápida de Fourier (FFT)

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Engenharia Eletrônica e Computação

Orientador: Prof. Dr. Eduardo Antonio C. da Costa
Co-orientador: Prof. Dr. Sergio Bampi

Pelotas
2018

“Quem tem milho, tem tudo.”

— AUTOR DESCONHECIDO

Dados Internacionais de Catalogação na Publicação (CIP)

S241e Sapper, André Neves

Exploração do algoritmo CORDIC para a redução de potência em arquiteturas de transformada rápida de Fourier (FFT). / André Neves Sapper. – Pelotas: UCPEL, 2018.

75 f.

Dissertação (mestrado) – Universidade Católica de Pelotas, Programa de Pós-Graduação em Engenharia Eletrônica e Computação, Pelotas, BR-RS, 2018. Orientador: Eduardo Antônio C. da Costa; co-orientador: Sérgio Bampi.

1. CORDIC. 2. FFT. 3. arquitetura serial. 4. baixo consumo de potência. I. Costa, Eduardo Antônio C. da, or. II. Título.

CDD 621

Ficha catalográfica elaborada pela bibliotecária Cristiane de Freitas Chim CRB 10/1233

AGRADECIMENTOS

Um agradecimento aos colegas, professores e funcionários do Mestrado em Engenharia Elétrica e Computação da Universidade Católica de Pelotas pela troca de experiências que contribuíram para a realização deste trabalho.

Agradeço ao Leonardo Soares pelo empenho na consolidação do CORDIC. O Leonardo participou fervorosamente na concepção dos modelos matemáticos das primeiras versões do CORDIC e também foi responsável pelas sínteses Cadence destas versões.

Agradeço ao Guilherme Paim pelas sínteses Cadence das primeiras versões da FFT sequencial. O Guilherme recebeu esta difícil missão, aos quarenta e cinco do primeiro e segundo tempos, e as encarou com destreza.

Agradeço aos amigos Jeroen Vermeeren e Erasmo José Dias Chiappetta Filho pelas discussões e incentivos que se tornaram contribuições ao trabalho.

Agradeço a minha sobrinha querida, Heloísa, que me autorizou faltar ao seu aniversário de dois aninhos por demandas finais deste trabalho.

Agradeço aos professores Sergio José Melo de Almeida e Everton Granemann Souza pela revisão e sugestões quando da Qualificação deste trabalho.

Faço aqui um agradecimento especial ao meu orientador, o professor Eduardo Antonio César da Costa, por toda a ajuda, ensinamentos, compreensão e paciência desta jornada. Sem ele, este trabalho não seria possível.

Agradeço à minha família pelo apoio, compreensão, logística e suporte operacional.

Agradeço à CAPES, pelo apoio financeiro.

Agradeço a Deus.

RESUMO

Este trabalho tem por objetivo a exploração do algoritmo de rotação de coordenadas denominado CORDIC (*COordinate Rotation Digital Computer*) em arquiteturas dedicadas de Transformada Rápida de Fourier (FFT - *Fast Fourier Transform*). Arquiteturas totalmente sequenciais de diferentes tamanhos (diferentes números de pontos) para projetos específicos de algoritmos FFT são implementadas e comparadas. Nas arquiteturas FFT, o algoritmo CORDIC tem sido amplamente utilizado para a geração dos coeficientes (*twiddle factors*), visto que esse algoritmo elimina a necessidade de utilização de circuitos multiplicadores, além de reduzir os requerimentos de memória ROM (*Read Only Memory*). Visto que o *twiddle factor* é composto por fatores de cossenos e senos, o trabalho explora o relacionamento entre o número de bits e o número de iterações no algoritmo CORDIC para verificar o impacto no consumo de energia em arquiteturas ASIC para funções seno e cosseno. Verificou-se que há uma relação direta entre o número de bits e iterações utilizados pelo CORDIC e o impacto disto em hardware. O melhor ajuste em termos de precisão é dependente de aplicação, ou seja, o número de bits na entrada do circuito e o número de iterações que o CORDIC deve executar estão individualmente relacionados com a precisão exigida pela aplicação final. A partir do espaço de projeto das implementações do algoritmo CORDIC, explora-se o seu uso em arquiteturas FFT sequenciais. A principal finalidade é estabelecer uma metodologia de projeto de baixa potência para arquiteturas FFT a partir do uso do algoritmo CORDIC. Para tal finalidade, foram implementadas FFTs de 32, 64, 128 e 256 pontos com o CORDIC. Os principais resultados mostraram que a utilização do CORDIC é vantajosa quando comparada com a solução de armazenagem clássica explícita dos *twiddle factor* em todos os cenários. Entretanto, baseado nos resultados obtidos, observou-se que ainda havia oportunidades para melhorias. Foram desenvolvidas então três versões adicionais do CORDIC (v.2, v.3 e v.4) das quais a melhor foi utilizada numa releitura da FFT implementada inicialmente. Os resultados demonstram uma melhora significativa quando utilizadas as versões mais recentes do CORDIC e da FFT. Os resultados também demonstram que o CORDIC é uma alternativa vantajosa quando considerado seu custo área-*power* em frequências mais baixas de operação.

Palavras-chave: CORDIC. arquitetura serial. FFT. baixo consumo de potência.

Exploration of the CORDIC Algorithm for Power Reduction in Fourier Transform (FFT) Architectures

ABSTRACT

This work explores the use of CORDIC (COordinate Rotation Digital Computer) algorithm in dedicated Fast Fourier Transform (FFT) architectures. Fully sequential architectures of different sizes (different numbers of points) for specific FFT algorithm projects are implemented and compared. The CORDIC algorithm has been widely used in FFT architectures to generate the twiddle factors, since this algorithm eliminates the need to use multiplier circuits, besides reducing the ROM memory requirements. Since the twiddle factors are composed of factors of cosines and sines, the work explores the relationship between the number of bits and the number of iterations in the CORDIC algorithm. The main scope is to verify the impact on energy consumption in ASIC architectures for sine and cosine functions. By the obtained results it has been verified that there is a direct relationship between the number of bits and iterations used by CORDIC and the impact of this on hardware. The best fit regarding accuracy is dependent on the application. In other words, the number of bits at the circuit input and the number of iterations that CORDIC must perform are individually related to the accuracy required by the final application. From the design space enabled by the implementations of the CORDIC algorithm, is explored its use in sequential FFT architectures. The main goal is to establish a low power design methodology for FFT architectures based on the use of the CORDIC algorithm. For this purpose, FFTs of 32, 64, 128 and 256 sizes were implemented with CORDIC. The main results showed that the use of CORDIC is advantageous when compared to the explicit classical twiddle factor storage solution in all scenarios. However, based on the results obtained, it was observed that there were still opportunities for improvement. Three additional CORDIC versions (v.2, v.3 and v.4) were then developed, the best of which was used in a re-reading of the initially implemented FFT. The results demonstrate a significant improvement when using the most recent versions of CORDIC and FFT. The results also demonstrate that CORDIC is an advantageous alternative when considering its area-power cost at lower operating frequencies.

Keywords: CORDIC, serial architecture, FFT, low power consumption.

LISTA DE ABREVIATURAS E SIGLAS

AASI	Aparelho de Amplificação Sonora Individual
ASIC	Application-Specific Integrated Circuit
CMOS	Complementary Metal-Oxide-Semiconductor
CORDIC	Coordinate Rotation Digital Computer
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
GHDL	G Hardware Design Language
GHW	GHDL Waveform
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
LEF	Library Exchange Format
PIPO	Parallel-input, Parallel-output
PLE	Physical Layout Estimation
RTL	Register Transfer Level
SDF	Standard Delay Format
STFT	Short-Time Fourier Transform
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

LISTA DE SÍMBOLOS

Δf	Resolução em frequência
ΔT	Intervalo amostral
μm	$micro$ metro
f_{op}	Frequência de operação (Hz)
f_s	Frequência de amostragem (Hz)
f_{max}	Frequência máxima detectável
Hz	Hertz
j	$\sqrt{-1}$
kB	Quilobyte
kHz	Quilohertz
MHz	Megahertz
mW	$mili$ watt
N	Tamanho da FFT ou número de pontos
nm	$nano$ metro
T	Período em segundos
V	Volt
$x(n)$	Sequência das amostras de entrada
$X(m)$	m-ésimo componente de saída da DFT

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em uma época de Internet das Coisas (do inglês, *Internet of Things*, IoT), em plena Era da Informação, de grande oferta de produtos eletrônicos digitais e com importantes *foundries* como TSMC (TSMC, 2017) e Samsung (SAMSUNG, 2017) anunciando para os próximos anos a manutenção da Lei de Moore (MOORE, 1965) com tecnologias de 7 nm e 5 nm, a fácil aquisição de produtos eletrônicos se tornou realidade. Era de se esperar, no entanto, que os equipamentos portáteis mais populares, principalmente aqueles operados por bateria como *smartphones*, *smartwatches* e *notebooks* (entre diversos outros), tivessem uma autonomia de utilização maior. Por sinal, a crescente preocupação com consumo de energia dos dias atuais recorda a preocupação que houve, um dia, com desempenho. Além disto, este anseio por autonomia não diz respeito somente à energia elétrica ou a outras fontes de energia, mas faz parte de um processo muito maior de consumo eficiente e consciente de recursos naturais.

Na prática, cada novo dispositivo eletrônico é lançado com menor peso, menor espessura e mais rápido do que sua versão anterior, mas sem ganhos significativos na duração da bateria. Uma razão para isto é que, ao mesmo tempo em que novos métodos computacionais são desenvolvidos e aplicados na obtenção de mais eficiência, o crédito resultante destes avanços proporciona, além das reduções físicas citadas, um acréscimo na capacidade computacional nos dispositivos. De forma direta, é possível afirmar que questões de *design* também afetam o desempenho geral do produto e vice-versa. Além do conteúdo, a forma também faz parte da evolução de produtos tecnológicos, ou seja, estética e ergonomia são consideradas partes essenciais no desenvolvimento e no aperfeiçoamento destes produtos. Aliás, estes são conceitos amplamente dominados pela Apple (APPLE, 2017), detentora de diversas patentes e prêmios na área de *codesign* de *hardware* e *software*.

Uma classe de algoritmos que vem sendo bastante utilizada, com a proliferação de equipamentos portáteis operados por bateria é a de processamento digital de sinais (DSP - *Digital Signal Processing*). Em aplicações DSP, uma das operações mais amplamente utilizadas envolve o processamento dos sinais pela transformada discreta de Fourier (DFT - *Discrete Fourier Transform*) (COSTA, 2002). Essa classe de algoritmo tem uma larga utilização em áreas como processamento de imagens, processamento de áudio, engenharia biomédica, comunicações, entre outras. Nessas áreas de aplicação, os sinais são processados a partir de algoritmos de transformada rápida de Fourier (FFT - *Fast Fourier Transform*), que processam de forma eficiente a DFT, com a redução do grande número de operações aritméticas envolvido.

Na FFT, a borboleta (*butterfly*) representa o elemento central de cálculo, onde as amos-

tras são multiplicadas por coeficientes fixos (*twiddle factors*), que representam valores múltiplos de $2\pi/N$, onde N é o número de pontos da FFT. Um dos métodos que tem sido amplamente utilizado para a geração dos *twiddle factors* é o algoritmo CORDIC (*COordinate Rotation Digital Computer*) (QURESHI, 2012). A ideia central do uso do CORDIC na FFT é substituir os fatores cosseno e seno por rotações iterativas. Isso permite tanto reduzir os requerimentos de memória ROM (*Read-Only Memory*), quanto substituir as operações de multiplicação por operações mais simples de somas e deslocamentos.

Este trabalho explora o uso do algoritmo CORDIC em arquiteturas FFT. Para tal, são realizadas inicialmente implementações do algoritmo CORDIC na forma estendida (*unfolded*) com e sem o uso de *pipeline*. Nas arqutieturas implementadas são explorados os aspectos de variação do número de bits e do número de iterações para aproximações do resultado, visando a redução da dissipação de potência. Os resultados iniciais mostraram que existe um espaço de projeto a ser explorado. As aproximações que podem ser realizadas no circuito CORDIC, a partir das variações do número de bits e do número de iterações, provocam um erro, que pode ser controlado, dependendo da aplicação, em favor da redução da dissipação de potência.

Os bons resultados obtidos com as implementações do algoritmo CORDIC estimulam o seu uso em arquiteturas FFT. Para tal, foram implementadas arquiteturas FFT, totalmente sequenciais, com a geração dos $twiddle factors$ através do CORDIC. A arquitetura sequencial implementada é totalmente parametrizável e não foi baseada em nenhuma arquitetura específica disponível na literatura, mas sim, é resultado de um trabalho original, seguindo o expertise dos autores do trabalho. A implementação é baseada no uso de registradores de deslocamento, o que habilita o uso da técnica *clock gating*.

As sínteses realizadas das FFTs levaram conta as seguintes estratégias de implementação: (*i*) usando apenas *shift registers* circular, ou seja armazena integralmente os valores das partes real e imaginária dos coeficientes; (*ii*) usando *shift registers* e *clock gating*; (*iii*) usando *shift registers* e CORDIC (armazena somente os ângulos dos *twiddle factors* das referidas sequências, e, finalmente (*iv*) usando *shift registers*, *clock gating* e CORDIC.

Os resultados preliminares mostraram que o uso do CORDIC, na estrutura baseada em *shift registers*, juntamente com a técnica *clock gating*, promoveram a redução de dissipação de potência em arquiteturas de FFTs com tamanhos de 32, 64, 128 e 256 pontos para duas frequências alvo (16 kHz e 32 kHz).

1.1 Motivação

De acordo com Dongarra and Sullivan (2000), o algoritmo FFT é considerado um dos 10 algoritmos mais importantes de todos os tempos. Nas palavras do próprio autor, "*a FFT talvez seja o algoritmo mais ubíquo hoje em uso para analisar ou manipular dados discretos ou digitais*." Segundo Crandall, Klivington and Mitchell (2009), FFTs podem ser tão pequenas quanto 256 pontos[1] e tão grandes quanto 2^{30} pontos[2]. O autor Crandall, Klivington and Mitchell (2009) ainda destaca diversas aplicações em diferentes áreas de modo a destacar a versatilidade de aplicações da FFT, ainda que orientadas a grandes volumes de dados. Entre as aplicações é possível citar: geofísica, sismologia, processamento de imagens, processamento de áudio, tomografia, cosmologia, criptografia, genômica computacional, entre outras.

A importância histórica e atual da transformada de Fourier a coloca como *kernel* da função de processamento principal a ser explorada. Diversas derivações da FFT *Radix*-2 original proposta por Cooley e Tukey em 1965 (COOLEY; TUKEY, 1965) surgiram e ainda surgem, sempre, com o desafio de reduzir a complexidade e/ou o número de operações aritméticas. O espaço de exploração arquitetural de uma FFT é proporcional ao seu custo computacional. Além disto, está diretamente relacionado a diferentes métricas de desempenho em diferentes níveis de projeto exigidos pela aplicação final, o que amplia as possibilidades de pesquisa de estratégias originais.

Neste trabalho a principal motivação é o uso do algoritmo CORDIC em arquiteturas FFT visando a sua baixa dissipação de potência. Embora o algoritmo CORDIC tenha sido descoberto em 1959 por Jack E. Volder (VOLDER, 1959), é possível identificar espaço de projeto a ser explorado nesse algoritmo para diferentes aplicações. O uso do algoritmo CORDIC em arquiteturas FFT vem sendo explorado ao longo dos anos, visto a versatilidade desse algoritmo, que realiza de forma simples (apenas com somas e deslocamentos), o cálculo de funções trigonométricas, hiperbólicas e logarítmicas entre outras. Desta forma, esse algoritmo é explorado nesse trabalho como forma de identificar formas eficientes de geração dos *twiddle factors* nas borboletas da FFT.

O projeto de arquiteturas FFT de baixa dissipação de potência poderá habilitar o seu uso em aplicações que demandem o aumento do tempo de vida útil da bateria. No caso, esse trabalho deverá explorar, quando da sua finalização, o uso de arquiteturas FFT de baixa potência para aparelhos auditivos.

[1] para processamento de fala, por exemplo.
[2] 1.073.741.824 de pontos, para matemática experimental, por exemplo.

1.2 Objetivos

Com o estudo realizado neste trabalho foi possível verificar a possibilidade de exploração do algoritmo CORDIC em arquitetura FFT. Desta forma, o principal objetivo deste trabalho é a exploração de técnicas de redução de potência em arquiteturas FFT, que inclui a exploração do algoritmo CORDIC. Inicialmente foi desenvolvida uma arquitetura FFT totalmente sequencial parametrizável. Com isso, pôde-se explorar o relacionamento entre o número de pontos e o número de bits da FFT para uma aplicação alvo.

As aproximações possibilitadas pelas implementações do algoritmo CORDIC possibilitaram aproximações importantes na FFT, visando a baixa dissipação de potência. O objetivo foi realizar uma exploração arquitetural nas implementações da FFT. Para tal, foram estabelecidas implementações com níveis de paralelismo para atingir uma frequência alvo de operação para uma determinada aplicação. No caso, usou-se como estudo de caso aparelhos auditivos que necessitam de projeto de arquitetura FFT de baixa dissipação de potência para o aumento do tempo de vida útil da bateria.

1.3 Metodologia

O projeto e o desenvolvimento dos algoritmos e estruturas apresentados neste trabalho de pesquisa foram obtidos através da utilização coordenada de diferentes ferramentas de simulação e síntese de circuitos. Sinteticamente, os algoritmos foram modelados em linguagem interpretada de alto nível (GNU Octave[3]), codificados manualmente em Linguagem de Descrição de *Hardware* (do inglês, *Hardware Description Language*, HDL) independente de plataforma (VHDL[4]), de modo estruturado e plenamente sintetizáveis visando ASIC (*Application-Specific Integrated Circuit*). A descrição da metodolodia utilizada para realização das sínteses lógicas e consequentes estimativas de potência consta na seção 1.3.1. Com exceção das ferramentas utilizadas no fluxo de síntese lógica, as demais são *open-source*.

Inicialmente, um modelo das macrofunções foi criado em Octave. Após realizar a validação destas macrofunções, o resultado é a obtenção de *golden models* do sistema ou de partes dele. Na sequência, após a codificação dos blocos e macroblocos em VHDL, estes *golden models* são utilizados como referência no processo de validação das saídas das implementações VHDL, com base em *stimuli* comuns, de forma a garantir a transposição dos modelos para uma

[3](EATON; BATEMAN; HAUBERG, 1997)
[4]Compatível com IEEE 1076-2008, (COMMITTEE et al., 2008)

representação em baixo nível. Para concretizar esta fase de desenvolvimento, ferramentas específicas para compilação e simulação de circuitos em HDL são necessárias, as quais foram utilizadas, o GHDL e o GTKWave. O GHDL (*G Hardware Design Language*) (GINGOLD, 2011) é um simulador de código aberto para a linguagem VHDL. Esse simulador permite compilar e executar VHDL com suporte total às versões de 1987, 1993 e 2002 do padrão IEEE (*Institute of Electrical and Electronics Engineers*) e com suporte parcial à revisão de 2008. O GTKWave (BYBELL, 2010) é um visualizador de forma de onda com suporte a diversos formatos de arquivos, entre eles, VCD (*Value Change Dump*) e GHW (*GHDL Waveform*).

As entradas e saídas das etapas anteriormente descritas são valores com sinal em complemento de 2, e as entradas e saídas apresentam largura de *bits* parametrizável. Complementarmente, é indispensável destacar que a parametrização de modelos e blocos funcionais é de grande importância na exploração de arquiteturas. São os parâmetros que permitem ajustar, de maneira prática, os diferentes propósitos operacionais internos de um bloco de *hardware*, de modo a verificar objetivamente qual a melhor configuração para atender alguma (ou algumas) restrição de projeto.

1.3.1 Metodologia para Estimativa de Potência

A metodologia para estimativa de potência é mostrada na Figura 1.1. Uma síntese utilizando o RTL *Compiler* no modo PLE (*Physical Layout Estimation*) é executada para gerar o *netlist* em nível de portas VHDL e os arquivos SDF (*Standard Delay Format*). O RTL *Compiler* recebe: (*i*) o projeto VHDL, (*ii*) as bibliotecas de células de 45 nm, (*iii*) o arquivo de restrições de tempo, (*v*) os arquivos Macro e Tech LEF (*Library Excahnge Format*) e (*v*) o arquivo Tabela de Capacitâncias. Os valores dos cosenos de entrada e função senoidal foram usados e armazenados em arquivos de texto. Esses valores são fornecidos para a ferramenta de simulação que executa o $testbench$ com os arquivos de *design* das arquiteturas CORDIC e gera um arquivo de atividade de comutação em um formato VCD. O VCD e o *design* dos arquivos são entregues para a ferramenta de síntese que gera os relatórios de área, potência e atraso. As arquiteturas CORDIC foram descritas em VHDL e sintetizadas em células padrão Nangate de 45 nm com o compilador Cadence RTL. A dissipação de potência foi estimada considerando uma fonte de alimentação de 1,1 V e uma operação de frequência entre 10 MHz e 1,7 GHz.

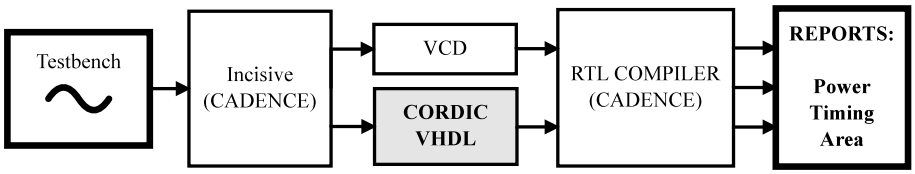

Figura 1.1 – Metodologia para estimação de potência

Fonte: O Autor

Vale destacar que o consumo de potência em circuitos CMOS (*Complementary metal-Oxide Semiconductor*) pode ser dividido fundamentalmente em consumo estático e consumo dinâmico de potência, conforme mostra a Figura 1.2. O consumo estático, proveniente da corrente de fuga, é a potência consumida pelas capacitâncias internas presentes no circuito. O consumo dinâmico é proveniente, basicamente, pela atividade de chaveamento do circuito, pelo curto-circuito durante o chaveamento dos transistores e pela atividade de chaveamento espúria (*glitching*), derivada da profundidade lógica do circuito. Neste trabalho serão apresentados resultados de potência relativos à potência dinâmica e à potência estática.

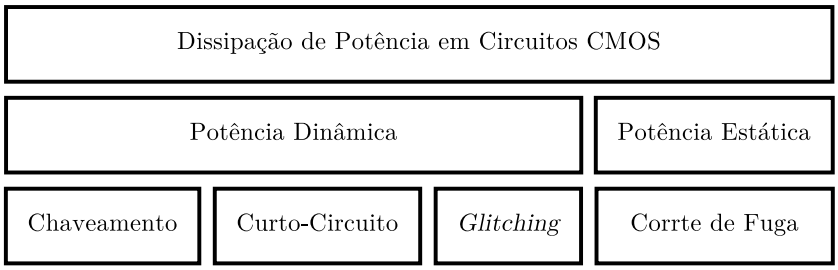

Figura 1.2 – Organização das fontes de consumo de potência em circuitos CMOS.

Fonte: O Autor

1.4 Contribuições do Trabalho

As principais contribuições deste trabalho, estão voltadas à implementação de uma arquitetura FFT totalmente sequencial, parametrizável, usando o algoritmo CORDIC. Verificou-se o impacto da redução de potência de arquiteturas do algoritmo CORDIC, a partir do relacionamento entre as variações no número de bits e no número de iterações. Verificou-se a vantagem do uso do algoritmo CORDIC na geração dos *twiddle factors* da arquitetura FFT implementada. Desta forma, as principais contribuições deste trabalho estão divididas em:

- (*i*) Implementação de uma arquitetura CORDIC estendida *unfolded*;
- (*ii*) Implementação de uma arquitetura CORDIC *unfolded full pipelined*;
- (*iii*) Exploração de arquiteturas CORDIC;

- (*iv*) Avaliação do impacto de potência das arquiteturas propostas em (*i*), (*ii*) e (*iii*);
- (*v*) Projeto e implementação de uma arquitetura FFT sequencial parametrizável;
- (*vi*) Projeto e implementação de um gerador de *twiddle factors*, a partir do algoritmo CORDIC para a arquitetura proposta em (*v*).

1.5 Organização do Trabalho

Este trabalho está estruturado em 5 capítulos. No Capítulo 2 são apresentados os conceitos básicos relacionados a Transformada Rápida de Fourier (FFT). Elementos estruturais e arquiteturais também fazem parte desta relação. O Capítulo 3 é dedicado ao algoritmo CORDIC e a sua implementação em *hardware*. O Capítulo 4 apresenta aspectos da implementação de uma arquitetura FFT sequencial e o uso do algoritmo CORDIC para a geração dos *twiddle factors*. Finalmente, no Capítulo 5, são apresentadas as conclusões do trabalho e propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA DA TRANSFORMADA RÁPIDA DE FOURIER

A Transformada de Fourier é apenas uma entre dezenas de transformadas matemáticas. O principal objetivo de uma transformada é mapear uma função em seu domínio original para outro domínio com a intenção de facilitar a visualização ou manipulação da função original neste novo domínio. Um exemplo típico é a transformada de coordenadas entre os sistemas cartesiano e polar.

Segundo Richard et al. (2004), a "*Transformada Discreta de Fourier (DFT) é um dos dois procedimentos mais comuns e poderosos encontrados no campo do processamento digital de sinais (filtragem digital é o outro)*". Isso devido ao fato de permitir analisar, manipular e sintetizar sinais de maneiras que não são possíveis com o processamento de sinais contínuos (analógico) no tempo.

De acordo com Richard et al. (2004), "*mesmo que a DFT esteja sendo utilizada em quase todos os campos da engenharia, aplicações para a DFT continuarão a surgir à medida que sua utilidade se tornar mais amplamente compreendida*. Para isto, uma sólida compreensão da DFT é fundamental para quem trabalha no campo de processamento digital de sinais.

2.1 Transformada Rápida de Fourier - FFT

A FFT é um algoritmo eficiente para o cálculo da Transformada Discreta de Fourier (DFT). Como lembrado por Richard et al. (2004), a FFT não é uma aproximação da DFT, mas sim representa a própria DFT, mas com um reduzido número de operações aritméticas.

Conforme descrito no artigo em Johnson and Frigo (2008), a diferença no número de operações aritméticas para FFTs de tamanho N, em potências-de-dois, desde a implementação *radix-2* por Cooley e Tukey (COOLEY; TUKEY, 1965) até o algoritmo estado-da-arte dos dias atuais (LUNDY; BUSKIRK, 2007), é de aproximadamente 25% $-$ respectivamente, $\sim 5Nlog_2N$ e $\sim \frac{34}{9}Nlog_2N$. Os autores Johnson e Frigo (JOHNSON; FRIGO, 2008) ainda lembram que, inicialmente, e durante muitos anos, o principal caminho para promover melhorias em relação ao algoritmo de Cooley e Tukey focava na redução do número de operações aritméticas. Segundo os autores, grandes esforços foram despendidos no sentido de encontrar novos algoritmos que reduzissem o número de operações aritméticas, o que resultou numa variação do algoritmo de Cooley e Tukey chamada *Split-Radix* (YAVNE, 1968) e no aparecimento do método de Winograd (WINOGRAD, 1978).

A DFT pode ser definida como uma sequência discreta no domínio da frequência. A

equação correspondente, nestre trabalho sendo representada por $X(m)$, onde $m = 0, 1, 2...N - 1$, pode ser escrita nos formatos exponencial e retangular. A DFT no formato exponencial pode ser observada na Equação 2.1. Observando a Identidade de Euler em 2.2, a DFT pode ser reescrita no formato retangular, de acordo com a Equação 2.3.

DFT no formato exponencial:

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} \quad , m = 0, 1, 2...N - 1 \tag{2.1}$$

Identidade de Euler:

$$e^{-j\theta} = \cos(\theta) - j\sin(\theta) \tag{2.2}$$

DFT no formato retangular:

$$X(m) = \sum_{n=0}^{N-1} x(n)[\cos(2\pi nm/N) - j\sin(2\pi nm/N)] \quad , m = 0, 1, 2...N - 1 \tag{2.3}$$

Onde:

- $X(m)$: m-ésimo componente de saída da DFT, $X(0)$, $X(1)$, $X(2)$ e etc.,
- m: índice da saída da DFT no domínio da frequência, $m = 0, 1, 2, 3, ..., N - 1$,
- $x(n)$: sequência das amostras de entrada $x(0)$, $x(1)$, $x(2)$ e etc.,
- n: índice das amostras de entrada da DFT, $n = 0, 1, 2, 3, ..., N - 1$,
- j: $\sqrt{-1}$, e
- N: número de amostras de entrada e de pontos em frequência na saída da DFT.

Para fins de simplificação, é possível considerar ainda que $W_N = e^{-j2\pi/N}$, obtendo-se assim a DFT da Equação 2.4, onde W_N são os *twiddle factors* ou coeficientes trigonométricos (raízes primitivas da unidade) da transformada.

DFT simplificada:

$$X(m) = \sum_{n=0}^{N-1} x(n)W_N \quad , m = 0, 1, 2...N - 1 \tag{2.4}$$

Como se observa, o número de cálculos necessários para realizar a DFT é de ordem $O(N^2)$. Cooley e Tukey, em 1965, "reinventaram" o algoritmo da DFT de forma que pudesse ser eficientemente processado computacionalmente (COOLEY; TUKEY, 1965). A ideia central

da abordagem deles foi processar, recursivamente, $N/2$ (divide e conquista) amostras. Com isto, a complexidade computacional pôde ser reduzida para $O(NlogN)$ operações. As variações mais simples e mais conhecidas da FFT de Cooley e Tukey são a FFT *Radix*-2 Decimada no Tempo e a *Radix*-2 FFT Decimada em Frequência. A complexidade computacional de ambas é a mesma, a diferença é que o processo de dividir e conquistar no tempo promove a divisão das amostras de entrada em amostras pares e ímpares. Por outro lado, o processo de dividir e conquistar em frequência resulta em uma divisão das amostras de entrada em dois blocos − primeiras $N/2$ amostras e $N/2$ amostras restantes. A Figura 2.1 é a representação gráfica de uma DFT de 2 pontos, ou seja, uma *butterfly* de 2 pontos.

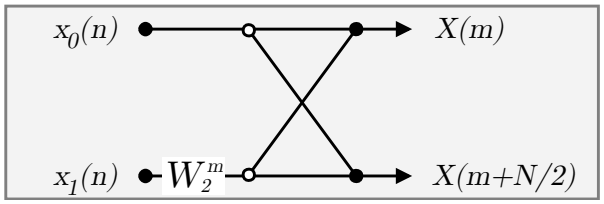

Figura 2.1 – DFT de 2 pontos (*butterfly*)
Fonte: O Autor. Inspirado por (RICHARD et al., 2004)

Conforme apresentado por (DIRLIK, 2013a), e de acordo com as Equações 2.1−2.4, definidas anteriormente, para uma DFT de 2 pontos, têm-se o estabelecimento das Equações 2.5 e 2.6.

$$X_0 = x_0 + x_1 \cdot W^0 \tag{2.5}$$

$$X_1 = x_0 + x_1 \cdot W^1 \tag{2.6}$$

De acordo com Richard et al. (2004), alguns autores gostam de explicar a redução aritmética da FFT pelas redundâncias inerentes nos *twiddle factors* W_N^m. Para isto, duas propriedades fundamentais dos *twiddle factors* são a simetria e a periodicidade, como mostram as Equações 2.7 e 2.8.

$$W_N^{(m+N/2)} = -W_N^m \tag{2.7}$$

$$W_N^{(m+N)} = W_N^m \tag{2.8}$$

A Figura 2.2 ilustra estas propriedades, mostrando as equivalências entre os coeficientes de uma DFT de 4 e 8 pontos.

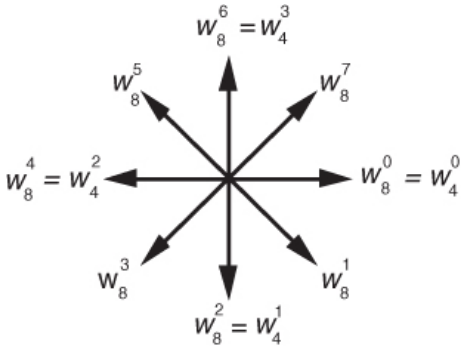

Figura 2.2 – Redundâncias cíclicas nos *twiddle factors* de uma FFT de 8 pontos.
Fonte: (RICHARD et al., 2004)

Desta forma, as Equações 2.5 e 2.6 podem ser reescritas na forma das Equações 2.9 e 2.10, ou seja, tem-se apenas uma multiplicação (complexa), diante duas da DFT na forma direta.

$$X_0 = x_0 + x_1 \cdot W^0 \tag{2.9}$$

$$X_1 = x_0 - x_1 \cdot W^0 \tag{2.10}$$

A Figura 2.3 representa uma FFT Decimada no Tempo com $N = 8$ pontos. Nota-se que, para este caso de decimação (tempo), as amostras de entrada (primeiro estágio) são divididas (decimadas) em pares e ímpares. No segundo estágio, as amostras são redivididas em pares e ímpares novamente, e assim em todos os estágios.

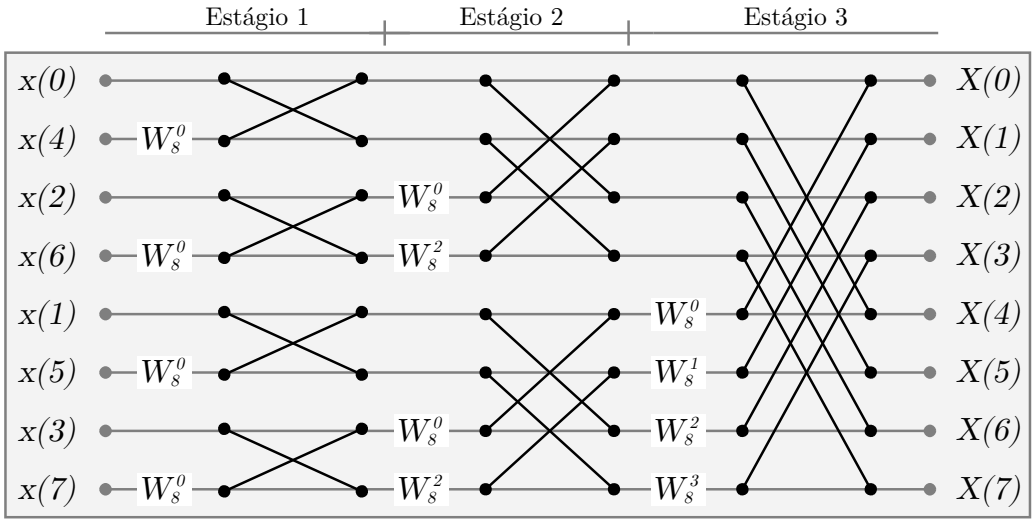

Figura 2.3 – FFT de 8 pontos (*butterfly*)
Fonte: O Autor. Inspirado por (RICHARD et al., 2004)

2.2 *Butterfly*

Conforme mencionado, a *butterfly* é o principal bloco aritmético de uma FFT. A Figura 2.4 apresenta a topologia da *butterfly radix*-2 utilizada neste trabalho. Ela é composta por um multiplicador complexo e dois somadores/subtratores complexos. O multiplicador complexo é composto por quatro multiplicadores reais e dois somadores/subtratores reais. O somador/subtrator complexo é composto por dois somadores/subtratores reais. O multiplicador complexo é o grande consumidor de recursos deste bloco. Existem algumas topologias e estratégias para a otimização deste circuito, as quais estão além do escopo deste trabalho. Os trabalhos desenvolvidos por Fonseca et al. (2010) e Neuenfeld (2016) discutem diversas soluções em diversos níveis de abstração para circuitos de *butterflies*. O trabalho desenvolvido por Costa (2002) discute, entre outras coisas, diversas técnicas de otimização para operadores aritméticos aplicados a circuitos DSP.

Supondo-se que $a+jb$ e $c+jd$ sejam dois números complexos, sua soma/subtração pode ser definida de acordo com a Equação 2.11. Considerando-se estes mesmos números complexos, a multiplicação entre eles pode ser definida de acordo com a Equação 2.12. O número total de operações aritméticas em uma butterfly é 10, sendo 4 multiplicadores e 6 somadores/subtratores.

$$(a + jb) + (c + jd) = (a + c) + j(b + d) \tag{2.11}$$

$$(a + jb) * (c + jd) = (a * c - b * d) + j(b * c + a * d) \tag{2.12}$$

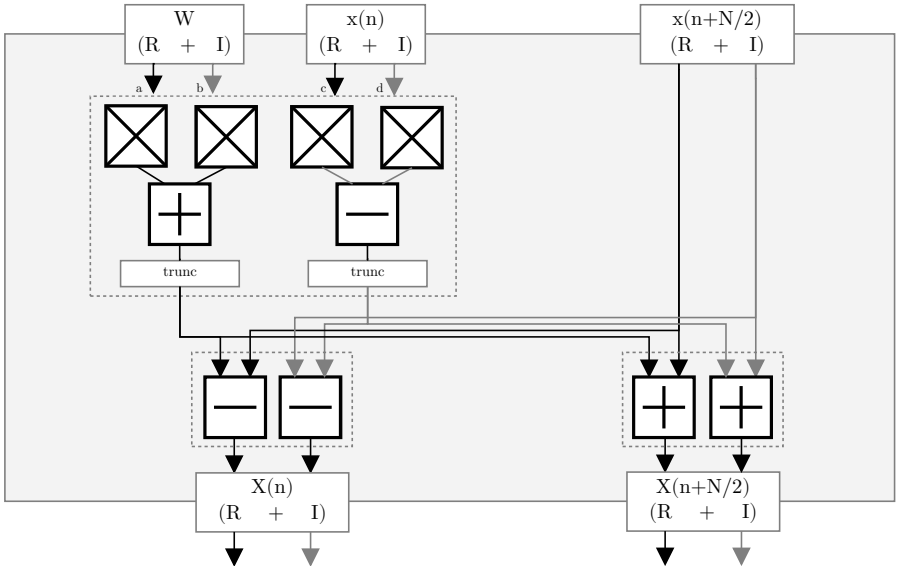

Figura 2.4 – *Butterfly* Decimada no Tempo

Fonte: O Autor

2.3 Resolução de uma FFT

Um parâmetro importante na especificação de uma FFT é a sua resolução em frequência, representada por Δf. Conforme se observa na Figura 2.5, diversas relações podem ser estabelecidas. O intervalo amostral, representado por ΔT, é o número total de amostras de entrada da FFT (N) vezes o perído entre uma amostra e outra, representado por f_s, conforme mostra a Equação 2.14.

A frequência de amostragem e o intervalo de captura determinam a maior frequência disponível e a resolução de cada banda do espectro. Como se observa, a frequência máxima detectável equivale à metade da frequência de amostragem (Teorema de *Nyquist*, Equação 2.13) e a resolução em frequência de cada banda é o recíproco do intervalo amostral (Equações 2.14 e 2.15).

Duas dicas práticas estabelecidas por Richard et al. (2004) como princípio geral de operação na utilização de uma FFT são (*i*) amostrar rápido o suficiente e (*ii*) amostrar por tempo suficiente.

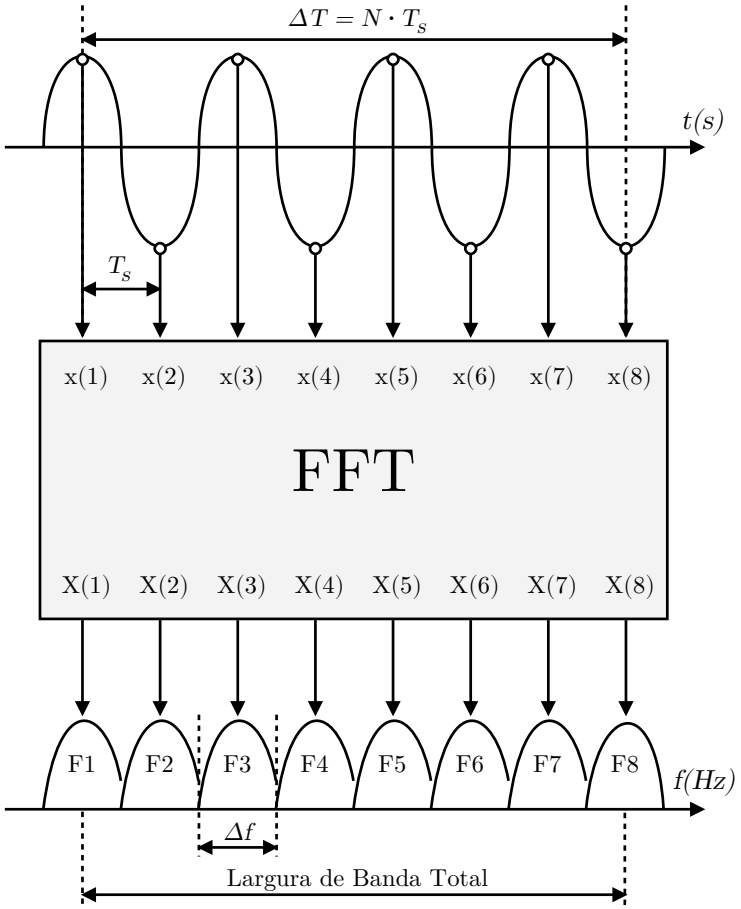

Figura 2.5 – Intervalo amostral

Fonte: O Autor

$$f_{max} = \frac{1}{2f_s} \tag{2.13}$$

$$\Delta T = N \cdot f_s \tag{2.14}$$

$$\Delta f = \frac{1}{\Delta T} \tag{2.15}$$

Onde:

N: Número de amostras (tamanho da FFT)

f_s: Frequência de amostragem

ΔT: Intervalo amostral

Δf: Resolução em frequência

2.4 Arquiteturas de FFTs - Histórico e Classificação

Na literatura, são encontradas diferentes classificações de arquiteturas de FFTs. Ao longo do tempo, alguns termos utilizados para classificar as diferentes arquiteturas foram sendo substituídos por outros mais atuais. Além disso, com soluções cada vez mais especializadas para problemas cada vez mais específicos, diversos novos termos e designações surgiram, de tal forma que não é incomum encontrar uma combinação de termos para classificar uma arquitetura. Deste modo, a principal contribuição deste capítulo é mostrar algumas terminologias que têm sido utilizadas ao longo do tempo.

Em 1969, apenas 4 anos depois da FFT de Cooley e Tukey, o trabalho de Bergland (BERGLAND, 1969b) já introduzia 4 diferentes organizações (famílias, segundo o autor) de máquinas: *(i)* processador sequencial, *(ii)* processador em cascata, *(iii)* processador paralelo iterativo e *(iv)* processador *array*. Neste mesmo trabalho, o autor observava que o tempo de execução e os requisitos de memória de um *software* eram suficientes para estimar a eficiência deste *software*, o que para implementações em *hardware* não era o bastante. Num trabalho complementar, Bergland (BERGLAND, 1969a) apresentou detalhes técnicos de projeto/desenvolvimento de FFTs de grandes empresas e instituições. Entre as 15 participantes estavam Bell, Departamento de Defesa, M.I.T., Texas Instruments e Divisão Aeroespacial Westinghouse, e cada uma delas recebera uma lista contendo 102 itens técnicos que deveriam ser preenchidos. Esta lista foi desenvolvida durante um Workshop em Processamento da Transformada Rápida de Fourier, organizado pela IEEE, e era composta por caracteristicas que serviriam para especificar uma FFT. Vale dizer que no segmento "Arquitetura" desta lista, além das 4 arquiteturas anteriormente citadas havia uma 5ª opção – *(v)* Outras – de certa forma, já prevendo espaço para possíveis exceções.

Em 1970, num trabalho de Groginsky and Works (1970), o termo "cascata" já era substituído por "*pipeline*". Embora o processador de sinais (FFT) descrito neste trabalho apresentasse uma estrutura em cascata, o autor preferiu a designação "*pipeline*" usada por *designers* de computadores porque, segundo ele, a estrutura em cascata permitia a aplicação direta de técnicas de *pipeline* em blocos aritméticos.

Em 1983, num trabalho de pesquisa de Thompson (THOMPSON, 1983) com foco em avaliar a relação Área-*Timing* em circuitos de Transformadas de Fourier em VLSI, novas possibilidades arquiteturais eram relacionadas e outros termos surgiam. Neste trabalho, Thompson reuniu nove estratégias de FFTs em VLSI: (*i*) 1-*cell* DFT, (*ii*) N-*cell* DFT, (*iii*) N^2-*cell* DFT, (*iv*) 1-proc DFT, (*v*) *Cascade*, (*vi*) FFT *Network*, (*vii*) *Perfect Shuffle*, (*viii*) CCC e (*ix*) *Mesh*.

A Figura 2.6 resume as principais topologias arquiteturais de FFTs encontradas na literatura. Nos dias atuais, existem variações das topologias mostradas na Figura 2.6, sendo que os seus elementos componentes característicos continuam os mesmos.

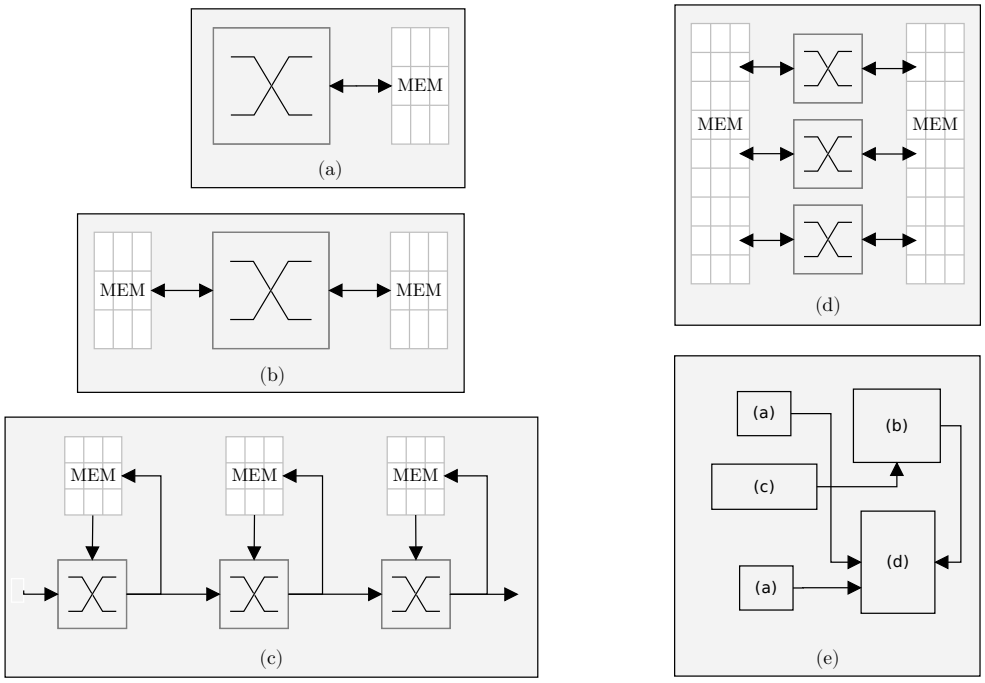

Figura 2.6 – Topologias de Arquiteturas de FFTs mais comumente encontradas – (a) sequencial e memória única; (b) sequencial e memória dupla; (c) cascata (*pipeline*); (d) paralela (*array*); (e) outras (*mesh*)

Fonte: O Autor

2.5 Resumo do Capítulo

Neste capítulo, alguns dos principais conceitos relacionados à compreensão da Transformada Discreta de Fourier (DFT) e da Transformada Rápida de Fourier (FFT) foram apresentados. Aspectos relativos à FFT, tais como os elementos constituintes (*butterflies e twiddle factors*), bem como um histórico e classificação de topologias de estruturas FFT foram apresentados. O próximo capítulo aborda os aspectos do algoritmo CORDIC, que darão subsídios à sua aplicação nas arquiteturas FFT propostas.

3 ALGORITMO CORDIC E SUA IMPLEMENTAÇÃO EM *HARDWARE*

Segundo Volder, (VOLDER, 1959), o Computador Digital para Rotação de Coordenadas (do inglês, *COordinate Rotation DIgital Computer*, CORDIC) é um operador multiuso que executa funções aritméticas de processamento de sinais como multiplicação e divisão, bem como funções trigonométricas, exponenciais e hiperbólicas. Nesse algoritmo, as multiplicações são realizadas por deslocamentos e somas para realizar a rotação do vetor. Desta forma, este operador é indicado para aplicações, como o algoritmo FFT (WANG; PIURI, 1997), cuja principal operação é executada pelo produto dos dados de entrada pelos coeficientes apropriados no seu componente *butterfly*.

Como o cálculo do CORDIC é inerentemente sequencial, a principal desvantagem deste algoritmo é a reduzida velocidade computacional. Na operação CORDIC, seu desempenho é dado de acordo com o número de iterações, que é igual ao tamanho da palavra interna (KAUR; SINGH, 2012). De fato, o número de iterações define a precisão do algoritmo. Para uma menor resposta de erro, um número maior de iterações é necessário. No entanto, ao considerar aplicações tolerantes a erros, onde a precisão pode ser relaxada, como no processamento de áudio ou vídeo, a implementação do algoritmo CORDIC com um número reduzido de iterações pode impactar diretamente na redução de dissipação de potência.

Neste capítulo, explora-se o relacionamento entre o número de bits e o número de iterações no algoritmo CORDIC para verificar o impacto no consumo de potência em arquiteturas ASIC para funções seno e cosseno. O objetivo principal é estabelecer uma relação ideal entre o número de iterações (para um erro aceitável) e o consumo de potência para uma determinada arquitetura. Para isso, foram implementadas arquiteturas estendidas (*unfolded*) de 8, 16 e 24 bits e sua potência dissipada é obtida, de acordo com a variação adequada do número de iterações. Em termos de arquiteturas propostas, duas abordagens são consideradas: (*i*) arquitetura totalmente *pipeline*, a qual registra todas as etapas da iteração e, (*ii*) arquitetura não *pipeline*, com registradores apenas na entrada e saída.

3.1 Visão Geral do Algoritmo

Esta seção resume as principais características do Algoritmo CORDIC, bem como os principais trabalhos da literatura relacionados ao escopo deste trabalho.

Segundo Volder (VOLDER, 1959), o CORDIC pode ser usado para resolver, em uma operação de computação e com igual velocidade: (*i*) as relações envolvidas na rotação de co-

ordenadas no plano; (*ii*) conversão de coordenadas retangulares em polares; (*iii*) multiplicação; (*iv*) divisão; ou (*v*) a conversão entre sistemas de base. Além disso, uma variedade de funções trigonométricas pode ser derivada dessas funções usando rotações vetoriais, como mostrado na Tabela 3.1. Como afirmado por Wang (WANG; PIURI, 1997), as áreas de aplicação do CORDIC foram expandidas para lidar com vários problemas de algoritmos DSP, tais como filtragem, equalização, FFT e processamento de imagens. A implementação eficiente do CORDIC é um ponto-chave para a realização efetiva de sistemas dedicados. O CORDIC fornece um método iterativo de executar rotações vetoriais, por ângulos arbitrários, usando apenas deslocamentos e adições. Devido à simplicidade das operações, é adequado para implementações em VLSI (*Very Large Scale Integration*).

O CORDIC é derivado da equação de rotação geral simplificada, conforme as Equações 3.3 e 3.4, que por sua vez são derivadas da Equação de rotação geral no plano, conforme definem as Equações 3.1 e 3.2. Substituindo $tan(\alpha)$ por $\pm 2^{(-i)}$, onde i é o índice da iteração, as multiplicações nas Equações 3.3 e 3.4 podem ser substituídas por operações de deslocamento. O valor da $tan(\alpha)$ é 1 para $i = 0$, 0.5 para $i = 1$ e assim por diante. Logo, 3.3 e 3.4 podem ser reescritos como 3.5 e 3.6, onde K_i é $cos(tan^{(-1)}2^{(-i)}))$, d_i é ± 1 e a Equação 3.7 é a direção na qual o vetor de entrada deve rotacionar, também chamado de acumulador de ângulos.

$$x' = x * cos\alpha - y * sin\alpha \tag{3.1}$$

$$y' = y * cos\alpha + x * sin\alpha \tag{3.2}$$

$$x' = cos\alpha * (x - y * tan\alpha) \tag{3.3}$$

$$y' = cos\alpha * (y + x * tan\alpha) \tag{3.4}$$

$$x_{i+1} = K_i[x_i - y * d_i * 2^{(-i)}] \tag{3.5}$$

$$y_{i+1} = K_i[y_i + x * d_i * 2^{(-i)}] \tag{3.6}$$

$$z_{i+1} = [z_i - d_i * tan^{(-i)}2^{(-i)}] \tag{3.7}$$

A Tabela 3.1 mostra os modos de operação do algoritmo CORDIC, levando-se em conta tanto os modos de rotação como de vetorização, para coordenadas lineares e hiperbólicas (WANG; PIURI, 1997). No modo de rotação, o vetor de entrada é rotacionado de acordo com um ângulo especificado (argumento de entrada). O modo de vetorização rotaciona o vetor de entrada no eixo x ao armazenar o ângulo necessário para fazer esta rotação. No modo de rotação, o acumulador de ângulo é inicializado para o ângulo desejado de rotação. A decisão do

sentido de rotação, a cada iteração, é feita para reduzir a magnitude residual do ângulo no acumulador de ângulos. Esta decisão baseia-se, portanto, no sinal do valor residual do ângulo. As primeiras simulações com o CORDIC foram realizadas para explorar o seu comportamento e a sua estrutura. Segundo Kaur (KAUR; SINGH, 2012), o algoritmo CORDIC tem cinco aspectos principais que afetam diretamente a sua implementação de *hardware*: (*i*) mapeador de quadrantes; (*ii*) inicialização; (*iii*) número de bits; (*iv*) número de iterações e (*v*) topologia arquitetural. Os trabalhos da literatura que levam em conta a maioria destes aspectos são apresentados na próxima subseção.

Tabela 3.1 – CORDIC: Modos de Operação

Coordenadas	**Saída**	**Modo Rotação**	**Modo Vetorização**
Circular $(m = 1)$	x	$K(xcosz - ysinz)$	$K\sqrt{x^2 + y^2}$
	y	$K(ycosz + xsinz)$	0
	z	0	$z + tan^{-1}(y/x)$
Linear $(m = 0)$	x	x	x
	y	$y + xz$	0
	z	0	$z + y/x$
Hiperbólico $(m = -1)$	x	$K'(xcoshz - ysinhz)$	$K\sqrt{x^2 + y^2}$
	y	$K'(ycoshz + xsinhz)$	0
	z	0	$z + tanh^{-1}(y/x)$

$K \approx 1.647$

Fonte: O Autor

3.2 Implementação de Arquiteturas do Algoritmo CORDIC

Este capítulo explora arquiteturas CORDIC e suas aproximações com vistas à sua implementação em ASIC (*Aplication-Specific Integrated Circuit*) CMOS (*Complementaty Metal-Oxide Semiconductor*). A redução no número de iterações e na largura de bits é sistematicamente avaliada para impulsionar a eficiência de energia em detrimento da ocorrência de erro de magnitude controlada para as funções trigonométricas de seno e cosseno. As versões aproximadas propostas da arquitetura CORDIC, são descritas em linguagem de descrição de *hardware* VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) e sintetizadas para uma tecnologia de 45 nm.

3.2.1 Trabalhos Relacionados

Ao longo dos anos, várias estratégias foram apresentadas na literatura para a implementação eficiente do algoritmo CORDIC. Como exemplo, em (MEHER et al., 2009) é apresentado um documento com uma revisão dos cinquenta anos de algoritmos e arquiteturas destinadas a acelerar o CORDIC. Por outro lado, uma revisão de implementações de algoritmos CORDIC em modo de rotação na versão estendida é apresentado em (LAKSHMI; DHAR, 2010).

Em relação à implementação, a maior parte das arquiteturas propostas são voltadas para solução em FPGA (*Field Programmable Gate Array*), como em (NEJI et al., 2013) que sugere uma implementação de FPGA do algoritmo CORDIC para sistemas de reconhecimento de impressões digitais. Em (SWAIN; PATNAIK, 2015), é proposto o projeto e a implementação do algoritmo CORDIC usando VHDL. Uma implementação de alta velocidade de ponto fixo do algoritmo CORDIC, com técnicas direcionadas ao ambiente Xilinx Virtex 5, é apresentada em (KAUR; SINGH, 2012). O trabalho em (LIU; FAN; MA, 2014) apresenta uma rotação de coordenada modificada para o CORDIC, através de uma implementação de arquitetura paralela para gerar formas de onda seno e cosseno. O CORDIC de 32 bits modificado é implementado em um dispositivo Spartan XC3S500E utilizando o pacote de projetos Xilinx ISE 12.3.

Embora as soluções mencionadas na literatura visem criar arquiteturas eficientes para os algoritmos CORDIC, nenhuma delas apresenta resultados de potência, já que todas visam soluções em FPGA. O trabalho em (TIWARI; GOEL, 2016) apresenta a implementação HDL do Algoritmo CORDIC Híbrido. A ferramenta *Synopsys Design Vision* foi utilizada para analisar as respostas da arquitetura em tecnologia de 90 nm. Segundo os autores, a arquitetura híbrida é rápida ao custo de precisão e consumo de potência. O trabalho em (MEHER; PARK, 2013) propõe projetos CORDIC para um ângulo fixo de rotação. Embora os projetos propostos tenham sido sintetizados pela ferramenta Synopsys Design Compiler, usando Biblioteca TSMC de 90 nm, não há resultados relativos à potência consumida. Diferentemente dos trabalhos mencionados da literatura (voltados para ASIC), as arquiteturas do CORDIC propostas neste capítulo são sintetizadas para uma tecnologia de 45 nm, e são apresentados resultados de potência para esta tecnologia.

No trabalho proposto, explora-se o impacto na redução de energia quando varia-se o número de iterações, onde o erro pode ser relaxado. Com esse objetivo, alguns trabalhos da literatura analisam o relacionamento entre o número de iterações e o erro computacional como em (HON; MARSONO, 2013), que propõe um *hardware* para a exploração espacial do algoritmo CORDIC para um tempo de execução em plataforma reconfigurável. No entanto, nenhum

resultado é mostrado em (HON; MARSONO, 2013) em relação ao impacto de variar o número de iterações, como é realizado neste trabalho de dissertação.

O cálculo de precisão do seno e funções cosseno, para diferentes ângulos alvo, é apresentado em (CHEN et al., 2015). Embora um algoritmo CORDIC com estratégia de rotação aprimorada seja proposta, para reduzir os tempos de iteração adicionais, não há indicações de redução de potência, visto que o trabalho não apresentou resultados de potência. A Tabela 3.2 resume as principais contribuições de trabalhos da literatura, bem como a nossa proposta.

Tabela 3.2 – Sumário dos Trabalhos Relacionados

#	Referência	***A***	***B***	***C***	***D***	***E***	***F***
1	(MEHER et al., 2009)	√	-	-	-	-	-
2	(LAKSHMI; DHAR, 2010)	√	-	-	-	-	-
3	(NEJI et al., 2013)	√	-	-	-	-	-
4	(SWAIN; PATNAIK, 2015)	√	-	-	-	-	-
5	(LIU; FAN; MA, 2014)	√	-	-	-	-	-
6	(TIWARI; GOEL, 2016)	√	-	√	-	-	-
7	(MEHER; PARK, 2013)	-	√	-	-	-	-
8	(HON; MARSONO, 2013)	-	√	√	-	-	-
9	(CHEN et al., 2015)	-	√	-	-	√	-
10	(EATON; BATEMAN; HAUBERG, 1997)	-	√	√	-	√	-
11	**(SAPPER et al., 2017)**	-	√	√	√	√	√

(A) Solução FPGA ***(B)*** Solução ASIC ***(C)*** Resultado de potência ASIC

(D) Metodologia de extração de potência ***(E)*** Variação do núm. de iterações

(F) Avaliação do impacto em potência e erro variando o núm. de iterações

Fonte: O Autor (SAPPER et al., 2017)

3.2.2 Arquiteturas CORDIC Propostas

As arquiteturas CORDIC propostas podem ser vistas na Figura 3.1. São implementadas duas abordagens arquiteturais diferentes usando o mesmo núcleo. Ambas as arquiteturas implementam o algoritmo CORDIC na forma estendida, ou seja, cada núcleo CORDIC, mostrado na Figura 3.1 (c), realiza uma iteração. A principal diferença entre estas duas estruturas CORDIC é que a da Figura 3.1 (a) não usa estágios de *pipeline*, enquanto que, a da Figura 3.1 (b),

usa estágios de *pipeline*. A estrutura apresentada na Figura 3.1 (c) representa o núcleo comum usado por ambas as soluções.

Ao implementar uma versão em *pipeline* do CORDIC, um maior *throughput* pode ser alcançado, visto que cada iteração pode ser realizada em cada ciclo de relógio. Note que o núcleo principal apresentado na Figura 3.1 (c) é composto de adições/subtrações e deslocamentos, o que pode ser facilmente implementado em *hardware*. As subtrações são realizadas em complemento de 2. Portanto, a combinação de uma estrutura básica simples com o uso de *pipeline* tende a maximizar o desempenho geral da arquitetura CORDIC. Além disso, o uso de *pipeline* é vantajoso para a redução da dissipação de energia, visto que há uma redução significativa da atividade de sinais espúrios (*glitching*).

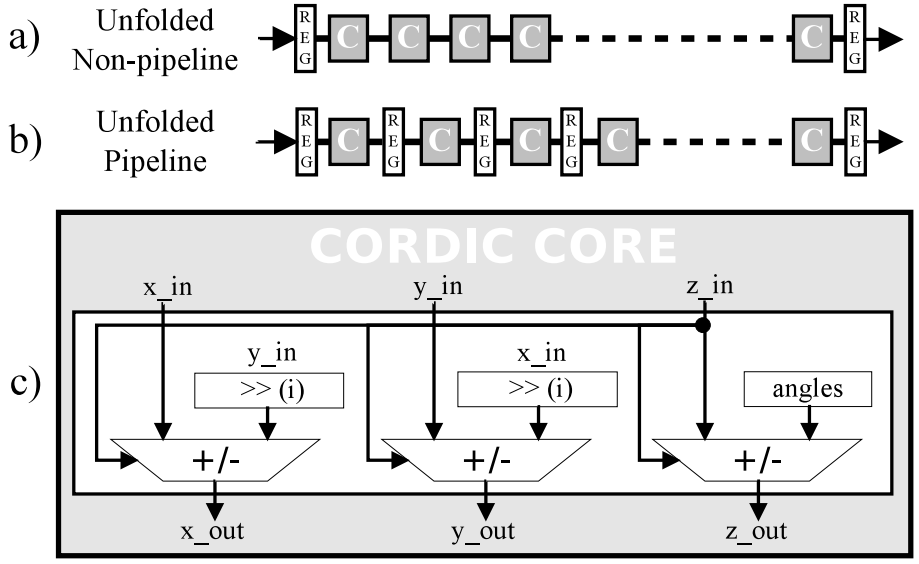

Figura 3.1 – Estrutura arquitetural do algoritmo CORDIC implementado: a) CORDIC totalmente paralelo; b) CORDIC totalmente paralelo com $n - 1$ estágios de *pipeline*, onde n é o número de iterações; c) *Core* do CORDIC utilizado nas versões dos itens a) e b).

Fonte: O Autor (SAPPER et al., 2017)

3.2.3 Resultados de Simulação e Implementação

A Tabela 3.3 mostra os resultados de síntese para as versões *pipeline* e não *pipeline* das estruturas CORDIC implementadas. As duas primeiras colunas da Tabela 3.3 mostram o cenário geral das simulações e sínteses realizadas. A primeira coluna é o tamanho da entrada em bits, e a segunda é o número de iterações. De imediato, percebe-se que o número de iterações nunca é maior do que o tamanho da entrada. Isto se deve ao fato de que os n bits da entrada não fornecem mais resolução do que n iterações. Isso representa que, cada nova iteração, apresenta

duas vezes mais precisão do que a iteração anterior. As colunas restantes são os resultados para ambas as estruturas CORDIC.

Tabela 3.3 – Resultados das Sínteses do CORDIC para 8, 16 e 24 bits

B	**I**	***Unfolded Pipeline***				***Unfolded Non-pipeline***				**MN**
		Frequência		**Potência**		**Frequência**		**Potência**		
		R	**E**	**R**	**E**	**R**	**E**	**R**	**E**	
	4			2.85	4.34		0.91	0.39	0.9	9.87
8	6		1.7	4.49	6.59		0.71	0.57	1.65	2.28
	8			5.36	7.08		0.61	0.99	3.75	0.60
	8			14.9	21.9		0.36	1.33	3.97	0.55
16	12	1.0	1.6	22.9	32.2	0.01	0.26	2.16	7.02	0.027
	16			29.4	38.1		0.21	5.51	23.8	0.003
	16			49.6	61.3		0.16	2.68	6.69	0.003
24	20		1.4	62.8	75.7		0.11	4.16	7.73	e-4
	24			76.4	87.8		0.11	7.12	18.0	e-6

(B) Bits **(I)** Iterações **(R)** Reduzida **(E)** Elevada **(MN)** Média Normalizada
do erro absoluto em percentagem(%). Frequência em GHz. Potência em mW.

Fonte: O Autor (SAPPER et al., 2017)

Sabe-se da literatura que, a técnica de *pipeline* reduz o caminho crítico do circuito, o que corresponde a uma maior frequência de operação. Deste modo, numa etapa de pré-síntese, as frequências alvo de 1,0 GHz e 2,0 GHz foram testadas para as versões sem *pipeline* e com *pipeline*. Observou-se que o tempo de folga (*slack time*) para essas sínteses das duas arquiteturas estava negativo mas com valores relativamente baixos. Isso representa que, as arquiteturas estavam próximas de uma possível frequência máxima de operação. Assim, a frequência inicial de 1,0 GHz com incrementos de 100 MHz foi escolhida para a versão *pipeline* (alto desempenho) e 10 MHz iniciais com incrementos de 50 MHz para as versões CORDIC sem *pipeline* (baixo desempenho).

Como resultado, e como esperado, as frequências máximas foram alcançadas pelas soluções que apresentavam o menor número de bits nas entradas. Para o primeiro caso (alto desempenho), foi obtida uma frequência máxima de operação de 1,7 GHz. Para o segundo caso (baixo desempenho), foi obtido um valor de 910 MHz como frequência máxima de operação. Para ambos os casos foram consideradas entradas com 8 bits de largura.

A última coluna da Tabela 3.3 está relacionada com a média do erro absoluto normalizado (MN) em porcentagem. Estes valores de erro foram obtidos, utilizando-se as funções seno e cosseno (`sin` e `cos`) internas do Octave, como referência aos valores obtidos através dos blocos CORDIC implementados. É possível observar na última coluna da tabela, que quanto maior o número de bits, menor é o erro para um número diferente de iterações. Para entradas com 16 e 24 bits de largura, por exemplo, as diferenças de erro não são significativas, e isso significa que, para aplicações tolerantes a erros, o número de iterações pode ser menor do que o número de bits. Este aspecto tem um impacto considerável na redução de energia, como pode ser comparado na Tabela 3.3, para diferentes frequências, e para ambos os casos, com e sem *pipeline*.

A Figura 3.2 favorece a visualização dos resultados listados na Tabela 3.3. Através desta figura é possível observar os intervalos mínimo e máximo dos resultados obtidos por cada uma das estratégias, as quais são plotadas individualmente nas Figuras 3.3 e 3.4. Embora os resultados de área (em μm^2) obtidos neste trabalho não sejam considerados para análise, pode-se afirmar que há um aumento significativo de área na estratégia com *pipeline*. Isso devido à introdução de circuitos registradores para a redução do caminho crítico.

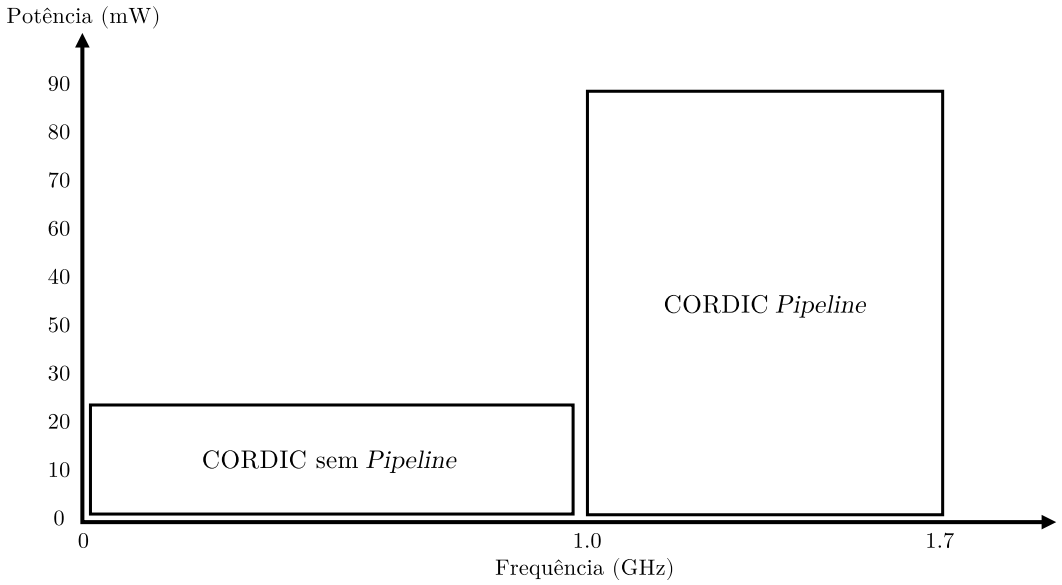

Figura 3.2 – Mapa geral dos resultados de síntese das estruturas CORDIC implementadas
Fonte: O Autor (SAPPER et al., 2017)

As Figuras 3.3 e 3.4 exibem, graficamente, as variações no número de bits da entrada (8, 16 e 24) e no número de iterações realizadas pelo CORDIC. Os gráficos mostram que é possível obter um ponto ótimo entre consumo de potência e o número de iterações, dependendo da frequência desejada (alto ou baixo desempenho). Vale ressaltar que nestas duas figuras o

erro absoluto máximo admitido não é levado em consideração, o que pode ser um parâmetro decisivo para a otimização do consumo de potência, conforme mostrado ao longo do capítulo. Vale ressaltar que, nos gráficos, `it` representa o número de iterações.

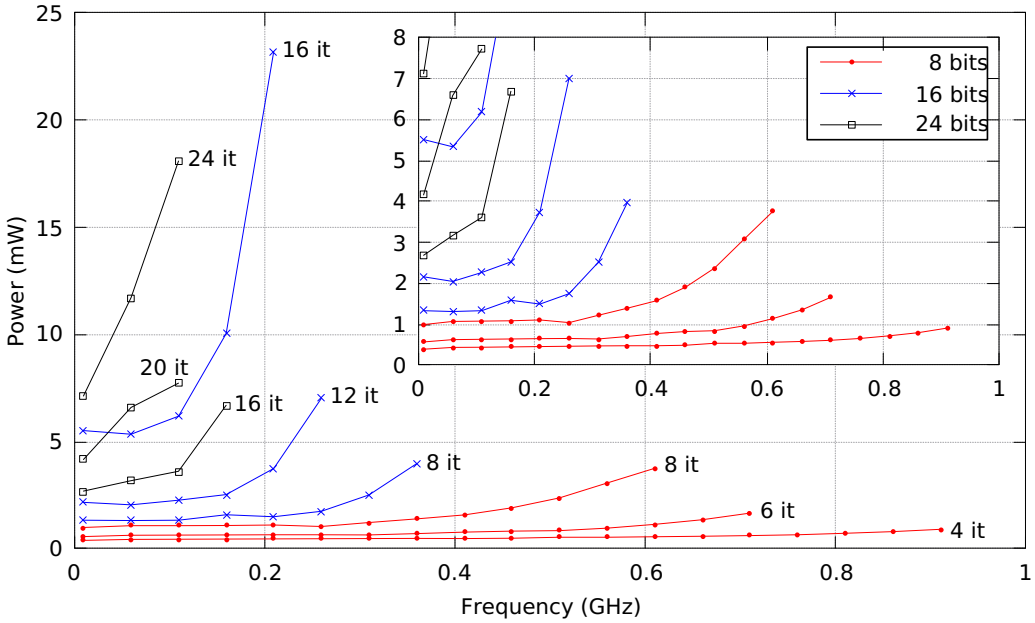

Figura 3.3 – Consumo de Potência X Frequência [CORDIC sem *Pipeline*]

Fonte: O Autor (SAPPER et al., 2017)

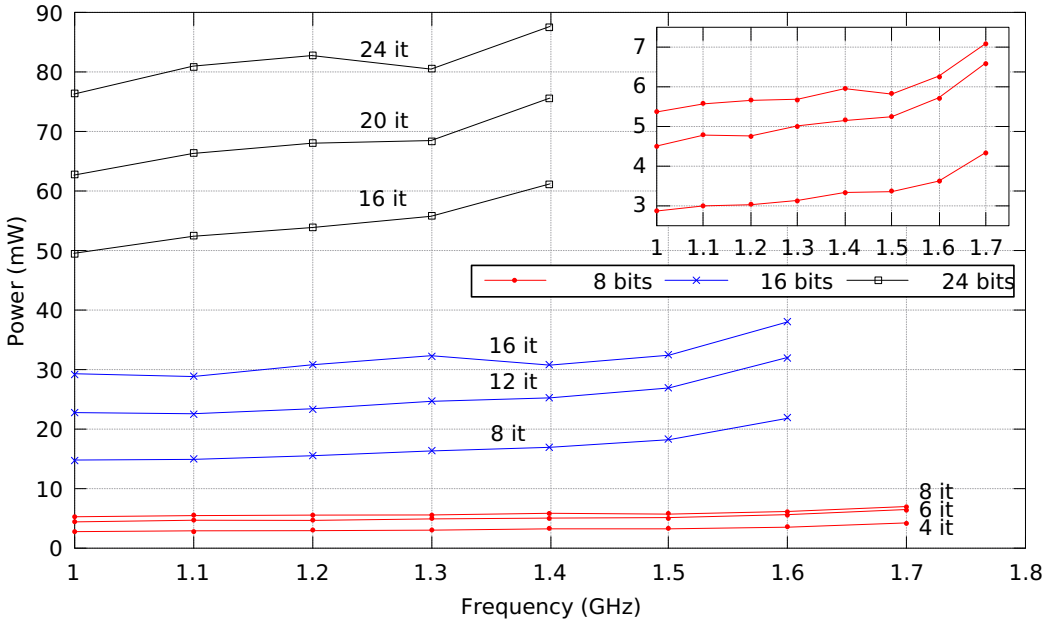

Figura 3.4 – Consumo de Potência X Frequência [CORDIC com *Pipeline*]

Fonte: O Autor (SAPPER et al., 2017)

Pelos resultados da Figura 3.5, é possível estabelecer relacionamentos ótimos entre restrições de energia e precisão. O gráfico da Figura 3.5 mostra que quanto maior o número de

iterações menor é o erro. Observa-se que, como os erros de 8, 16 e 24 bits estão próximos um do outro, apenas uma linha é mostrada. As linhas verticais na figura representam as faixas de potência mínimas e máximas para cada iteração. Supondo uma restrição de projeto de 10 mW e uma imprecisão máxima de 2,28% (6 iterações), portanto, os melhores candidatos para atender a estes requisitos podem ser os 8 bits de entrada e 6 iterações, dependendo da restrição de frequência do projeto.

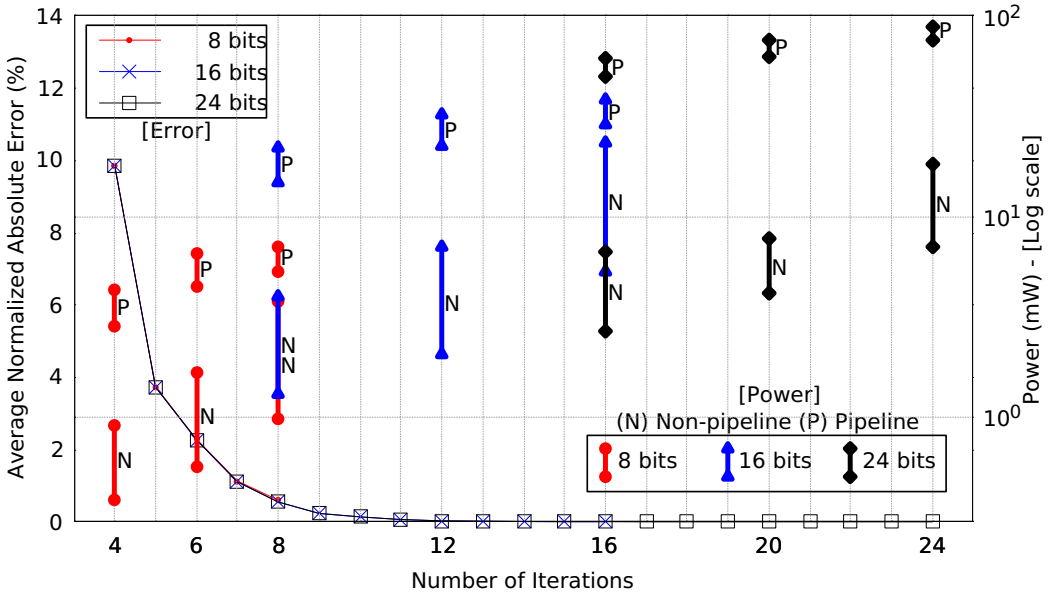

Figura 3.5 – Erro absoluto por iteração X Consumo de potência

Fonte: O Autor (SAPPER et al., 2017)

3.3 CORDIC versões 2.0, 3.0 e 4.0

Este capítulo apresentou, até aqui, o desenvolvimento do algoritmo CORDIC com vistas a uma utilização eficiente. A primeira versão permitiu vislumbrar algumas de suas principais características através das curvas dos resultados apresentados. Foi possível estabelecer uma metodologia de projeto para o CORDIC com a demonstração da variação do número de bits e do número de iterações para verificação do impacto no erro e na dissipação de potência.

Esta seção apresenta, especialmente, variações na implementação do CORDIC. Conforme exposto em seção introdutória (Tabela 3.1), o CORDIC pode ser utilizado, também, como rotacionador complexo. Isto se deve ao fato de que, em arquiteturas de FFT, os *twiddles* possuem módulo um (1). De modo a nortear as variações desenvolvidas, considera-se a Figura 3.6. Conforme se observa, foram desenvolvidas duas estratégias principais para implementar a multiplicação complexa: (*i*) multiplicador complexo tradicional e (*ii*) CORDIC. Foram desen-

volvidas duas versões de multiplicador complexo: (*i*) formado por multiplicadores genéricos ("*") e (*ii*) formado por multiplicadores do tipo Booth (COSTA, 2002). Foram desenvolvidas quatro versões do algoritmo CORDIC, sendo duas compostas pelo acumulador de ângulos e duas sem o mesmo. É imprescindível salientar que para que o CORDIC desempenhe uma multiplicação complexa corretamente, são necessários alguns blocos auxiliares conforme se observa na figura 3.6, quais sejam: (*i*) corretor de entrada, (*ii*) corretor de ganho e (*iii*) corretor de saída.

A sequência do capítulo apresenta em detalhes as principais diferenças entre essas quatro versões.

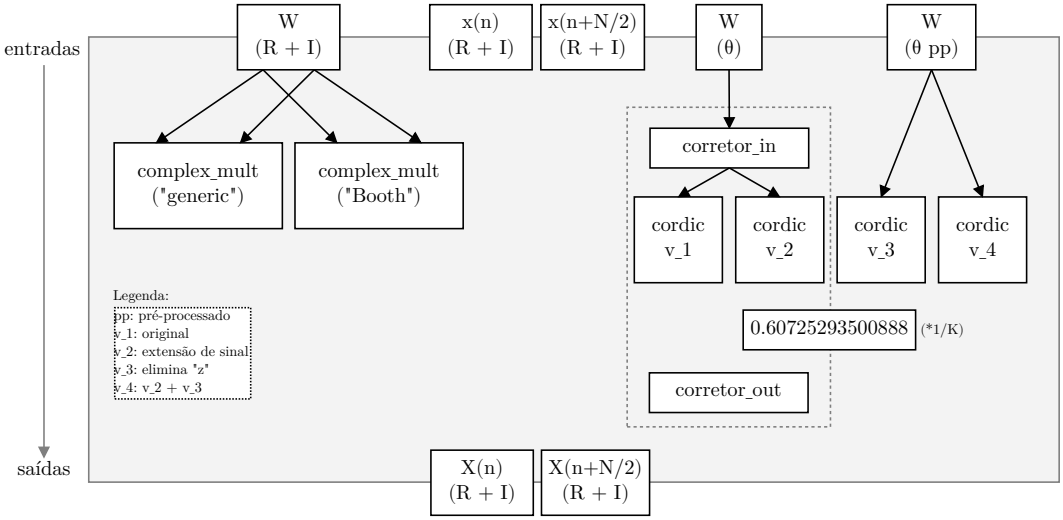

Figura 3.6 – Multiplicador complexo − Estratégia tradicional Vs. variações do CORDIC

Fonte: O Autor

A Figura 3.7 apresenta de forma completa todas as instâncias internas utilizadas no desenvolvimento de cada versão do CORDIC. De modo complementar, esta figura está organizada de modo hierárquico, onde se percebe que as versões 2.0, 3.0 e 4.0 derivam da versão 1.0. Mais especificamente, as versões 2.0 e 3.0 derivam da versão 1.0, e a versão 4.0 deriva das versões 2.0 e 3.0. A principal diferença entre as diferentes versões reside no fato de que as versões 3.0 e 4.0 não utilizam o acumulador de ângulos, o qual pode ser pré-processado quando o ângulo de entrada é previamente conhecido, que é o caso das arquiteturas implementadas neste trabalho. Esta característica na implementação visa garantir uma menor área e uma menor dissipação de potência.

Figura 3.7 – CORDIC – organização das instâncias das versões implementadas

Fonte: O Autor

As Figuras 3.8, 3.10, 3.12 e 3.13 a seguir apresentam, inicialmente, os detalhes internos de funcionamento em alto nível de abstração das quatro diferentes versões do CORDIC. O CORDIC v.1.0 representado genericamente pela Figura 3.8 é a versão original desenvolvida e apresentada na primeira parte deste capítulo. Esta figura é uma representação do algoritmo CORDIC clássico, sem quaisquer tipo de otimizações. O objetivo local é permitir um contraste das modificações realizadas nas versões subsequentes.

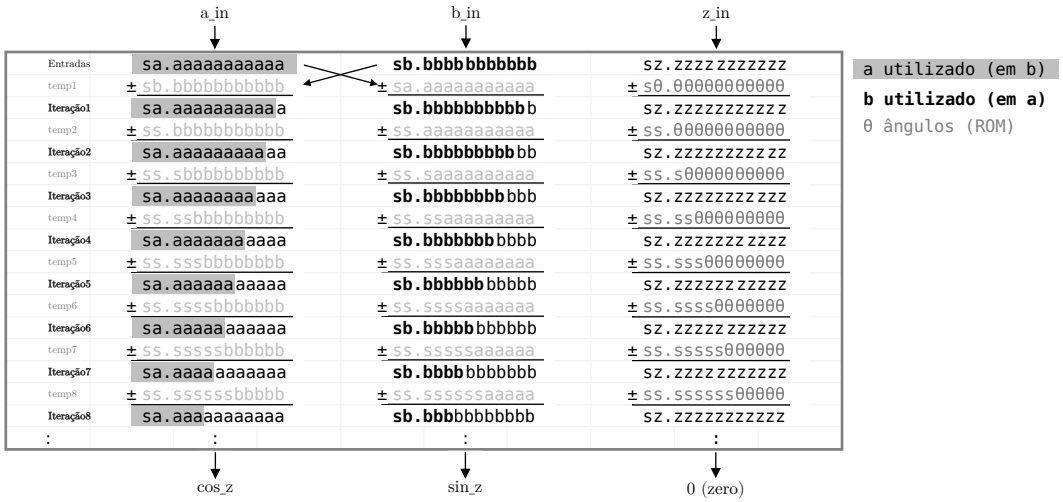

Figura 3.8 – CORDIC v.1.0 – esquema geral

Fonte: O Autor

A Figura 3.9 ilustra uma execução real com os valores de cada iteração de forma discriminada. As entradas em X, Y e Z são respectivamente $\sqrt{2}/2$ (~ 0.70711), $\sqrt{2}/2$ (~ 0.70711) e $\pi/4$, ou seja, deseja-se rotacionar um vetor com 45 graus de fase por 45 graus. Como o CORDIC possui natureza iterativa, a medida em que as iterações ocorrem, mais próximo do resultado final desejado.

```
>> [COS,SIN]=cordic(0.70711,0.70711,pi/4,9,9,0)

                                                                  d(+>0; -<0)
              X - (y * d)              Y + (x * d)                Z - Z_temp
-------------------------------------------------------------------------------
   INPUTS  X: 0010110101  0.7071   Y: 0010110101  0.7071   Z: 0011001001  0.7854
          -x: 0010110101  0.7071  +y: 0010110101  0.7071   z: 0011001001  0.7854
-------------------------------------------------------------------------------
   IT:  1  X: 0000000000  0.0000   Y: 0101101010  1.4142   Z: 0000000000 -0.0000
          +x: 0010110101  0.7071  +y: 0000000000  0.0000   z: 0001110111  0.4636
-------------------------------------------------------------------------------
   IT:  2  X: 0010110101  0.7071   Y: 0101101010  1.4142   Z: 0001110111  0.4636
          -x: 0001011011  0.3536  +y: 0000101101  0.1768   z: 0000111111  0.2450
-------------------------------------------------------------------------------
   IT:  3  X: 0001011011  0.3536   Y: 0110010111  1.5910   Z: 0000111000  0.2187
          -x: 0000110011  0.1989  +y: 0000001011  0.0442   z: 0000100000  0.1244
-------------------------------------------------------------------------------
   IT:  4  X: 0000101000  0.1547   Y: 0110100011  1.6352   Z: 0000011000  0.0943
          -x: 0000011010  0.1022  +y: 0000000010  0.0097   z: 0000010000  0.0624
-------------------------------------------------------------------------------
   IT:  5  X: 0000001101  0.0525   Y: 0110100101  1.6449   Z: 0000001000  0.0319
          -x: 0000001101  0.0514  +y: 0000000000  0.0016   z: 0000001000  0.0312
-------------------------------------------------------------------------------
   IT:  6  X: 0000000000  0.0011   Y: 0110100110  1.6465   Z: 0000000000  0.0007
          -x: 0000000111  0.0257  +y: 0000000000  0.0000   z: 0000000100  0.0156
-------------------------------------------------------------------------------
   IT:  7  X: 1111111010 -0.0246   Y: 0110100110  1.6465   Z: 1111111100 -0.0150
          +x: 0000000011  0.0129  +y: 0000000000 -0.0002   z: 0000000010  0.0078
-------------------------------------------------------------------------------
   IT:  8  X: 1111111101 -0.0118   Y: 0110100110  1.6467   Z: 1111111110 -0.0072
          +x: 0000000010  0.0064  +y: 0000000000 -0.0000   z: 0000000001  0.0039
-------------------------------------------------------------------------------
   IT:  9  X: 1111111111 -0.0054   Y: 0110100110  1.6468   Z: 1111111111 -0.0032
                COS = -0.0053517           SIN =  1.6468
```

Figura 3.9 – CORDIC v.1.0 – exemplo real

Fonte: O Autor

O CORDIC v.2.0 representado genericamente pela Figura 3.10 é a implementação que explora a redundância das extensões de sinal que são realizadas ao longo das iterações. Ao observar-se os resultados intermediários ao longo das iterações na Figura 3.9 é possível destacar que, da mesma forma que os valores tendem a convergir para o resultado final em decimal, o mesmo ocorre com os estes valores na sua representação binária. Desta forma, as versões 2.0 e 4.0 do CORDIC exploram esta redundância entre as iterações numa tentativa de acelerar parcelas já estabilizadas de uma iteração qualquer para a próxima.

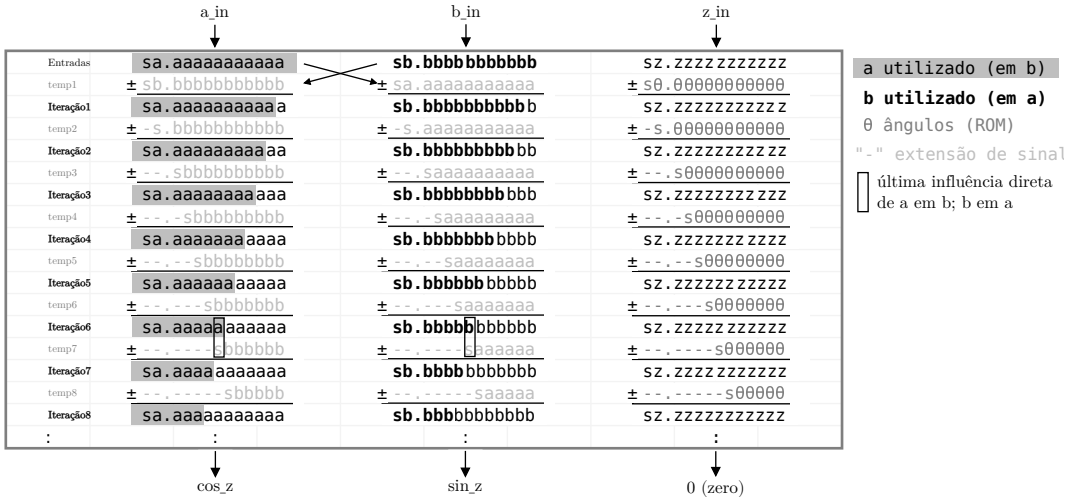

Figura 3.10 – CORDIC v.2.0 − esquema geral

Fonte: O Autor

A Figura 3.11 descreve o circuito completo de uma iteração n do CORDIC v.2.0, o qual explora a característica de redundância da extensão de sinal. Esta exploração foi feita de modo parcial, onde somente otimizou-se as iterações em que o valor intermediário da iteração anterior era positivo, ou seja, seu bit mais significativo era zero. Desta forma, foi possível desenvolver e aplicar o circuito da Figura 3.11 sem risco de perder algum bit de informação, já que somar ou subtrair de zero ou de uma sequência de zeros resulta no valor do próprio operando.

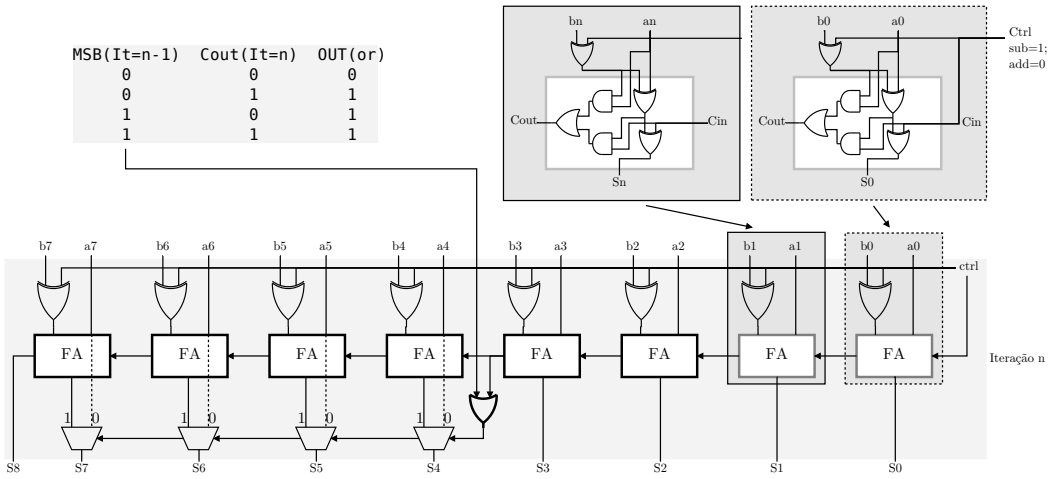

Figura 3.11 – CORDIC v.2.0 – exploração estrutural da redundância da extensão de sinal

Fonte: O Autor

O CORDIC v.3.0 representado genericamente pela Figura 3.12 é a versão que elimina completamente o processamento do acumulador de ângulos z. Isto se deve ao fato de que, para um conjunto de ângulos de entrada previamente conhecido, é possível pré-processar os sinais de controle que seriam resultantes do processamento em z e armazená-los (ROM) no lugar dos ângulos da mesma forma que estes o seriam.

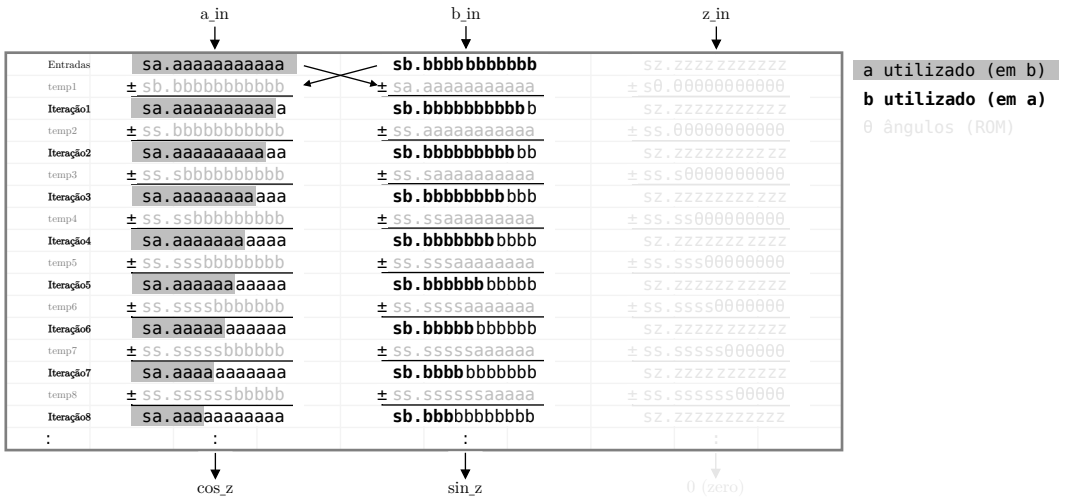

Figura 3.12 – CORDIC v.3.0 – esquema geral

Fonte: O Autor

O CORDIC v.4.0, representado genericamente pela Figura 3.13, é a versão que combina características das soluções exploradas nas versões 2.0 e 3.0 do CORDIC. Além do acumulador de ângulos ter sido completamente eliminado, também foi introduzida a técnica de exploração da redundância de sinal entre iterações.

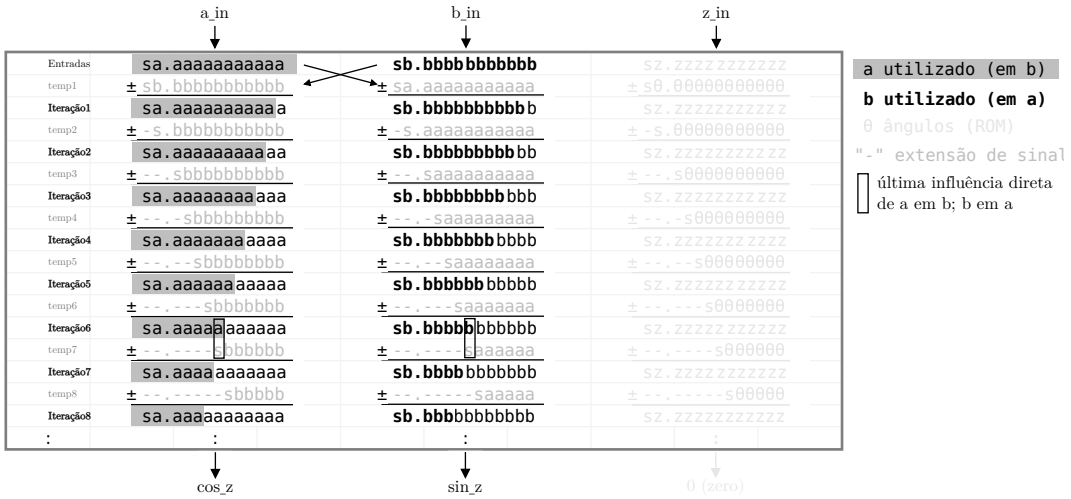

Figura 3.13 – CORDIC v.4.0 − esquema geral

Fonte: O Autor

3.3.1 Resultados de síntese para o CORDIC (v.2.0, v.3.0 e v.4.0)

Esta seção apresenta os resultados de síntese física obtidos para todos os possíveis candidatos a multiplicador complexo de uma arquitetura FFT. As Figuras 3.14 e 3.15 apresentam os resultados de consumo de potência e as Figuras 3.16 e 3.17 apresentam os resultados de área consumida em células. As sínteses foram feitas variando-se o período do ciclo de relógio de 15 ns a 300 ns com incrementos de 5 ns. Em frequência, portanto, estes valores variaram de 75 MHz a $\sim$3.33 MHz, nos gráficos, apresentados em ordem crescente de frequência.

A Figura 3.14 apresenta todos os resultados da síntese física de consumo de potência dos rotacionadores complexos desenvolvidos no trabalho. É notória a evidência de que, acima de 20 MHz, os multiplicadores complexos formados por multiplicadores tradicionais consomem menos. A principal razão para isto é o fato de que estes multiplicadores fazem parte da biblioteca de células nativa da ferramenta de síntese, isto é, a ferramenta possui conhecimento prévio e otimizado dos algoritmos que geram estes operadores aritméticos de forma a garantir sua eficiência. Vale destacar que para esta macroescala, a diferença no consumo de potência para valores inferiores a 20 MHz tende a ser minimizada como mostra a Figura 3.15 subsequente.

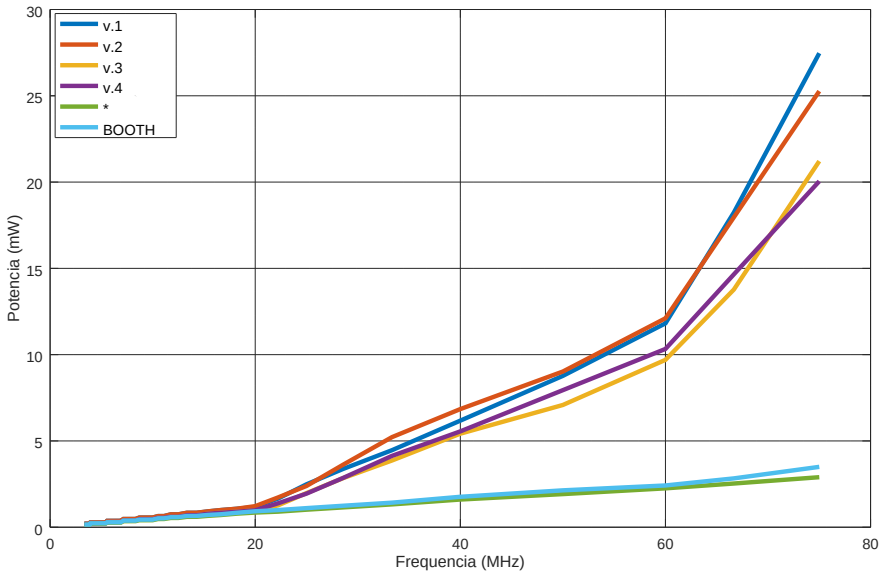

Figura 3.14 – Visão geral do consumo de potência total – CORDIC (todos) vs. multiplicador complexo (todos)

Fonte: O Autor

A Figura 3.15 expande a visualização da Figura 3.14 entre as faixas de 0 MHz a 25 MHz. Como se observa, os multiplicadores complexos (tradicionais) possuem uma variação praticamente constante no consumo de potência a medida em que se diminui (ou aumenta) a frequência de operação. Os multiplicadores complexos baseados no CORDIC possuem uma diminuição abrupta no consumo de potência até os 20 MHz, e após esta frequência, a diminuição do consumo de potência tende a ser constante da mesma forma que seu concorrente. Uma das razões para explicar este comportamento pode ser amparado pelos resultados de área consumida, apresentados nas Figuras 3.16 e 3.17.

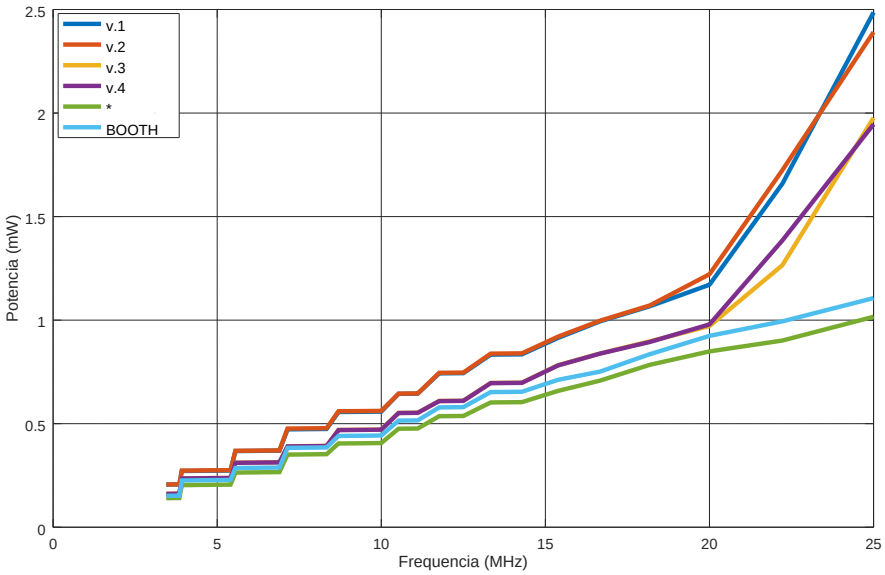

Figura 3.15 – Visão parcial do consumo de potência total − CORDIC (todos) vs. multiplicador complexo (todos)

Fonte: O Autor

A Figura 3.16 apresenta todos os resultados da síntese física para consumo de área dos rotacionadores complexos implementados. A exemplo do que ocorre para consumo de potência e pelos motivos já mencionados, é saliente que, acima de 20 MHz, os multiplicadores complexos tradicionais consomem menos área. Aliás, somente apresentam qualquer variação de área acima de 60 MHz, diferentemente do que ocorre com o consumo de potência. Para os casos CORDIC, o decréscimo de área acompanha o decréscimo de dissipação de potência até valores bem próximos a 20 MHz, conforme pode ser melhor visualizado na Figura 3.17.

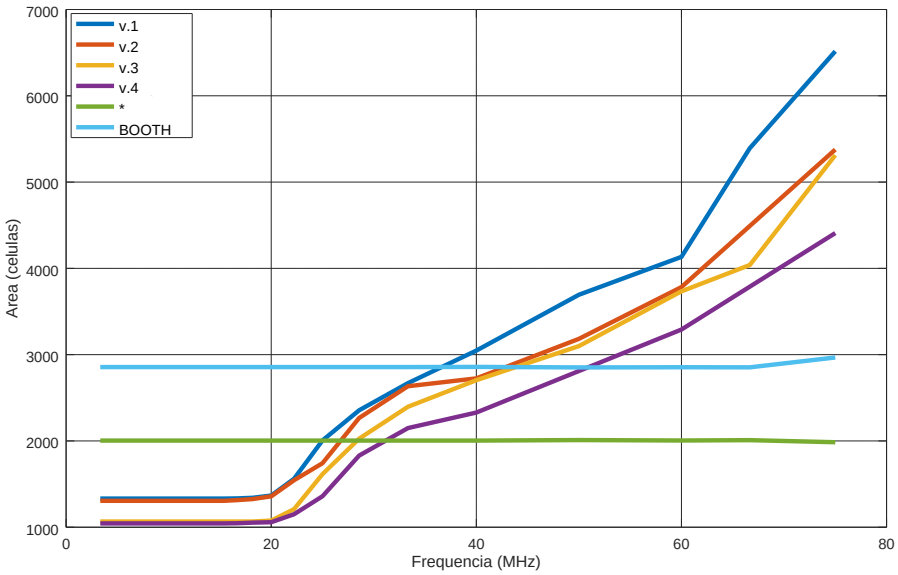

Figura 3.16 – Visão geral da área alocada – CORDIC (todos) vs. multiplicador complexo (todos)

Fonte: O Autor

A Figura 3.17 expande a visualização da Figura 3.16 entre as faixas de 0 MHz a 25 MHz de modo a proporcionar melhor percepção acerca da frequência exata em que o consumo de área torna-se praticamente constante. Este característica nos resultados de consumo de área permite inferir que a ferramente de síntese lógica atingiu valores muito próximos do limite inferior mínimo de portas lógicas (células). Vale exaltar que abaixo de 25 MHz, as quatro versões do CORDIC consomem menos área do que seus concorrentes, o que pode ser fator determinante para uma possível implementação de arquiteturas paralelas.

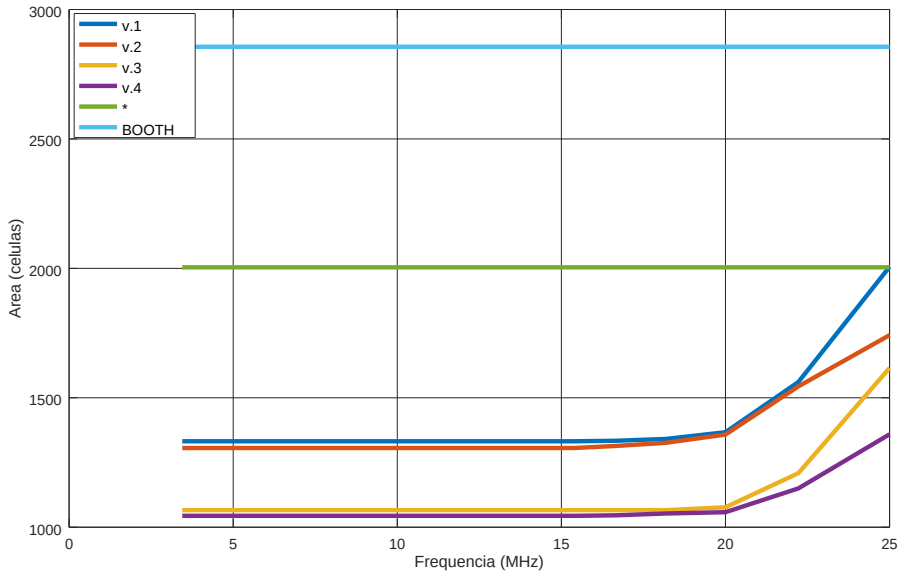

Figura 3.17 – Visão parcial da área alocada – CORDIC (todos) vs. multiplicador complexo (todos)

Fonte: O Autor

3.4 Resumo do capítulo

Este capítulo apresentou o desenvolvimento do algoritmo CORDIC com implementações deste algoritmo em *hardware* dedicado. Foram apresentados os detalhes das implementações propostas, bem como um resumo dos principais trabalhos da literatura relacionados ao tema. Por fim, foram apresentados os principais resultados obtidos com as implementações propostas. Foi possível estabelecer uma metodologia de projeto para o CORDIC com estabelecimento da variação do número de bits e do número de iterações para verificação do impacto no erro e na dissipação de potência. O capítulo a seguir mostra implementações de arquiteturas dedicadas FFT com o uso do algoritmo CORDIC.

4 PROJETO E IMPLEMENTAÇÃO DE UMA ARQUITETURA FFT SEQUENCIAL

Este capítulo descreve o projeto e a implementação de uma arquitetura FFT sequencial. Uma das principais contribuições de uma realização desta complexidade é a investigação de soluções segundo uma metodologia *top-down* de desenvolvimento. Para esta metodologia, o foco de atuação inicial ocorre mais próximo dos níveis algorítmico e sistêmico, o que permite identificar precocemente os principais gargalos do sistema. Uma outra contribuição importante é a validação de soluções para subsistemas específicos através da implantação destes subsitemas numa arquitetura hierarquicamente superior. Um exemplo prático disto é a utilização das técnicas e estruturas discutidas e implementadas no Capítulo 3 (CORDIC) na arquitetura FFT desenvolvida.

Com relação à arquitetura aqui descrita, embora sequencial, o *design* foi projetado e codificado de modo parametrizável, isto é, pode ser instanciado para implementar uma FFT *radix-2* de N pontos, apesar de apresentar uma única *butterfly*. É evidente que, para valores muito grandes de N (acima de 1024 pontos), a arquitetura pode não atingir o *throughput* e/ou a latência necessários pela aplicação. Além disto, a frequência de amostragem pode, por exemplo, ser maior do que a frequência de operação da borboleta, exigindo, para estes casos, um outro projeto de arquitetura.

Inicialmente, considerou-se a implementação de uma FFT com arquitetura *pipeline*. A principal característica deste tipo de arquitetura é apresentar uma *butterfly* dedicada por estágio. Uma vantagem desta solução é a regularidade da sua estrutura, o que facilita o processo de parametrização. Por outro lado, para FFTs relativamente pequenas (abaixo de 128 pontos), verificou-se que uma *butterfly* por estágio seria um potencial consumidor de recursos.

A Figura 4.1 apresenta um diagrama de blocos simplificado em alto nível da FFT sequencial. Podem ser observados 4 blocos principais: – (*i*) unidade de controle (FSM - *Finite State Machine*), (*ii*) *butterfly*, (*iii*) memória de dados e (*iv*) memória de coeficientes (*twiddle factors*). Este diagrama segue a classificação de topologias de FFT apresentadas no capítulo anterior.

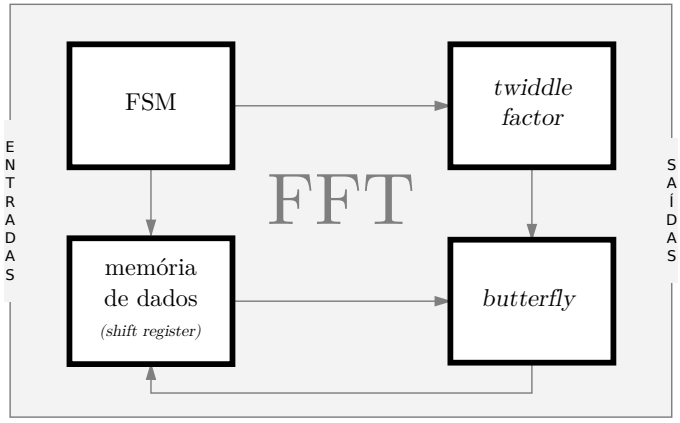

Figura 4.1 – FFT Totalmente Sequencial − Diagrama geral de blocos

Fonte: O Autor

A Figura 4.2 apresenta os elementos básicos para compreender a criação da arquitetura totalmente sequencial. A Figura 4.2 é composta de três subfiguras que são: 4.2 (a) estrutura direta de uma FFT *radix-2* de 8 pontos decimada no tempo com entradas invertidas; 4.2 (b) versão simplificada da arquitetura totalmente sequencial resultante e 4.2 (c) tabulação auxiliar do fluxo de entradas e saídas dos registradores de deslocamento (`sr0`, `sr1` e `sr2`) presentes em 4.2 (b).

Inicialmente, através da observação das características da FFT representada em 4.2 (a), a arquitetura totalmente sequencial pôde ser derivada. É possível notar que em 4.2 (a) há círculos que destacam as duplas de amostras que são processadas primeiro em cada estágio. Como se observa, há uma constância entre estes círculos que é equivalente a $N/2$ amostras. De fato, ao se analisar outras FFTs *radix*-2, o mesmo padrão é observado. Esta regularidade entre a ordem das amostras é uma das principais características que motivou o desenvolvimento da estrutura descrita neste trabalho.

Uma outra característica igualmente importante, e que pode ser observada em 4.2 (b), é a total ausência de codificação/decodificação de endereços de dados. Como se observa, só existe uma saída em cada instância de *shift register* (`sr0`, `sr1` e `sr2`). Isso significa que o acesso aos dados em processamento é feito de maneira totalmente sequencial, ou seja, com a utilização de *shift registers* como elemento de memória. Também, não existem atrasos de leitura/escrita no acesso aos dados que estão sendo processados. Deste modo, usa-se o termo “controle mínimo” para definir ou explicar a orientação geral de desenvolvimento desta versão de arquitetura.

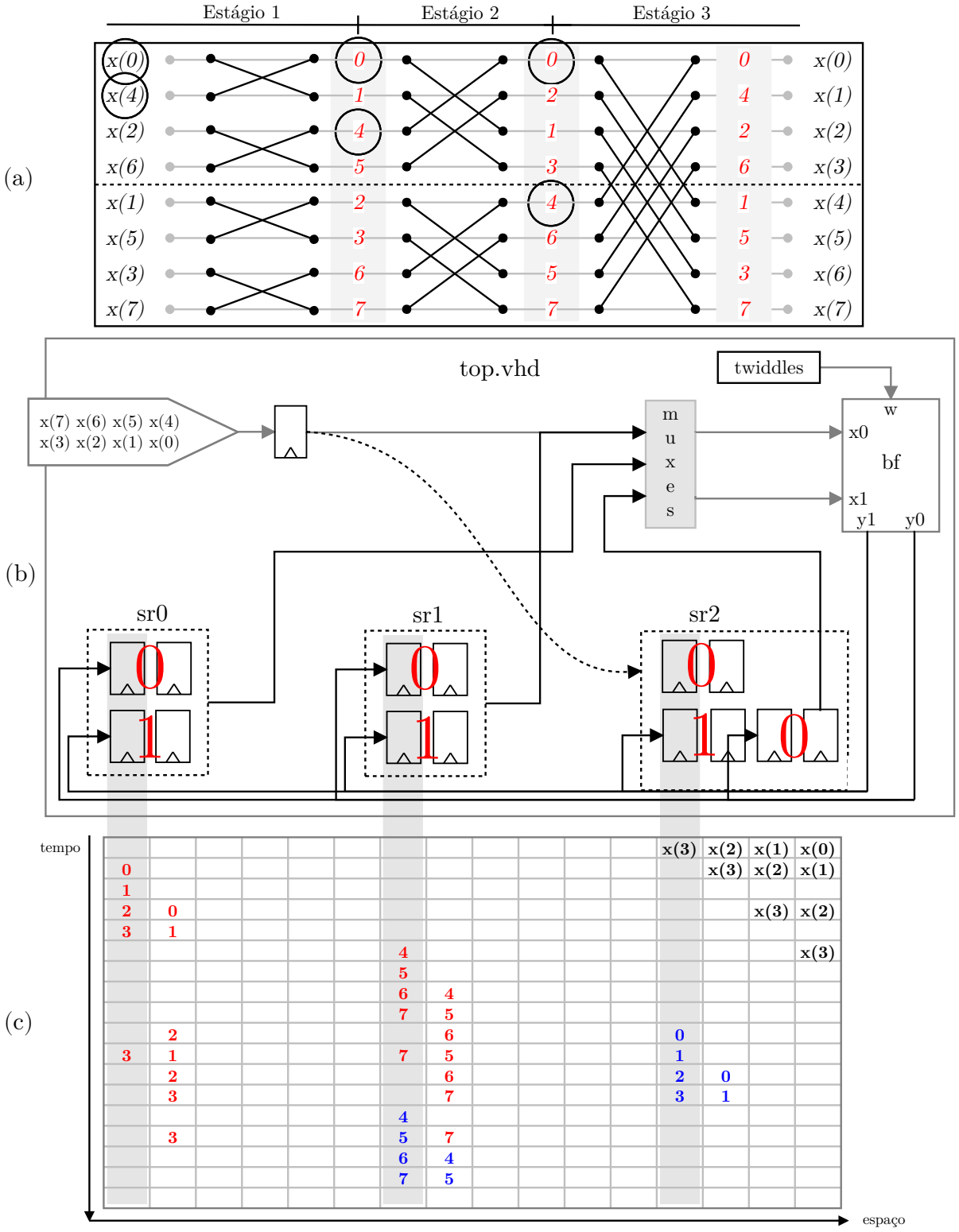

Figura 4.2 – FFT Totalmente Sequencial – Derivação da arquitetura parametrizável. (a) FFT $N = 8$ na forma direta (b) visão geral da arquitetura sequencial (c) tabulação para derivação da arquitetura proposta

Fonte: O Autor

4.1 Descrição Detalhada da FFT Sequencial

A Figura 4.3 apresenta um diagrama de blocos detalhado da FFT totalmente sequencial. Esta figura contém uma visão hierárquica do arquivo `top.vhd` de mais alto nível do sistema implementado. Uma das vantagens do desenvolvimento hierarquicamente estruturado é que diferentes soluções podem ser desenvolvidas para uma mesma subfunção. Este é um caso comum para os blocos *twiddle factors* e *butterfly*, temas constantes de pesquisa.

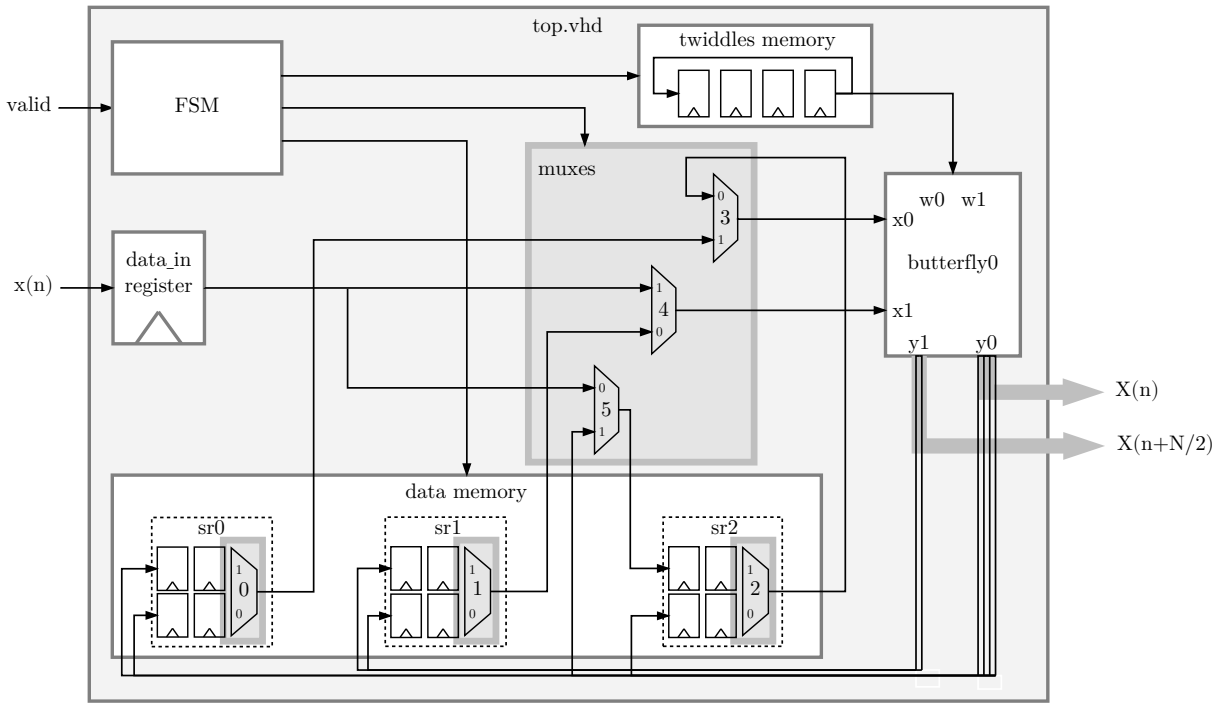

Figura 4.3 – FFT Totalmente Sequencial – Diagrama detalhado de blocos

Fonte: O Autor

A máquina de estados (FSM) controla um total de 6 multiplexadores (`muxes`) principais, organizados da seguinte maneira: os multiplexadores `mux0`, `mux1` e `mux2`, pertencentes à memória de dados, selecionam o modo de funcionamento (serial ou paralelo) dos *shift registers* de dados (`sr0`, `sr1` e `sr2`). Os multiplexadores `mux3` e `mux4` selecionam as entradas para processamento pela *butterfly*. Especificamente, o `mux3` seleciona entre as saídas de `sr0` e `sr2` para a entrada `x0` da *butterfly*. Equivalentemente, o `mux4` seleciona entre a saída de `sr1` e a saída de `data_in_register` para a entrada `x1` da *butterfly*. O multiplexador `mux5` seleciona entre `data_in_register` e a saída `y0` da *butterfly* para a entrada de `sr2`.

A memória de dados é composta de três *shift registers*: `sr0`, `sr1` e `sr2`. Cada um destes *shift registers* é formado por dois *shift registers* na configuração entrada paralela, saída paralela (do inglês, *Parallel Input Parallel Output*, PIPO). Detalhes sobre este bloco podem ser

vistos na seção 4.2.

4.2 Memória de Dados - *Shift Register*

A Figura 4.4 apresenta a memória de dados da FFT totalmente sequencial e seu princípio de funcionamento. Mais exatamente, a Figura 4.4 representa os dois modos de operação dos *shift registers* da memória de dados. Conforme descrito anteriormente, existem três instâncias de *shift registers*: duas instâncias compostas pelo modo representado pela Figura 4.4 (a) e uma instância composta pelo modo representado pela Figura 4.4 (b). Cada instância é composta de um *array* de *shift registers* na configuração "entrada paralela, saída paralela" (do inglês, *Parallel Input Parallel Output*, PIPO).

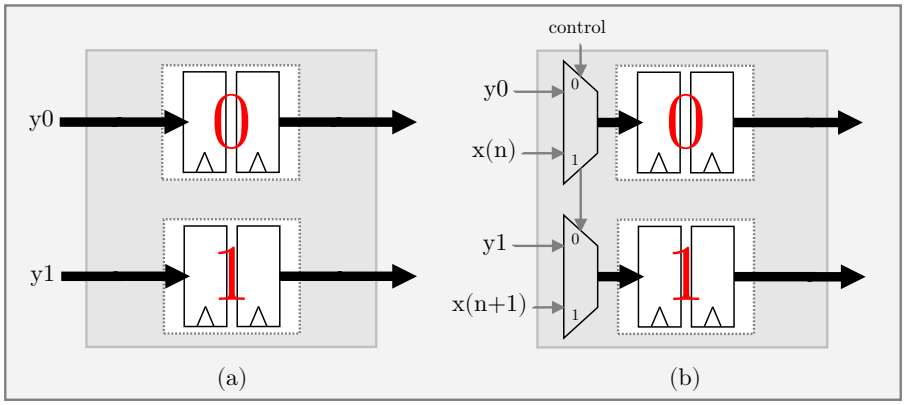

Figura 4.4 – FFT Totalmente Sequencial – Modos de operação do *Shift Register* de dados: (a) configuração paralela e (b) configuração híbrida

Fonte: O Autor

Os três *shift registers* que compõem a memória de dados da FFT são formados especificamente como: `sr0` e `sr1` por 4.4 (a) e o `sr2` por 4.4 (b). A principal diferença entre eles está no fato de que o `sr2` possui duplo comportamento. É ele o responsável por receber $N/2$ as amostras de entrada, portanto, sendo necessários multiplexadores na sua entrada. Vale ressaltar que os três *shift registers* possuem controle individual de `clock enable`, permitindo que otimizações de controle possam ser feitas.

4.3 Controle

Uma versão simplificada da máquina de estados da arquitetura sequencial implementada é apresentada através da Figura 4.5. A regularidade das operações aritméticas realizadas pela

arquitetura proposta permitiu inferir 4 estados de operação que representam satisfatoriamente o *modus operandi* geral da arquitetura, sendo, portanto, denominada de "Controle Mínimo". Esta nomenclatura não significa que esta arquitetura apresente o menor ou o mais simples controle entre as diversas implementações dsiponíveis na literatura, mas sim, servir como referência interna no contexto deste trabalho.

4.3.1 Máquina de Estados - FSM

Sob o ponto de vista funcional, o sinal de *reset* (`rst_n`) coloca a máquina de estados no estado inicial `IDLE` conhecido, bem como os registradores internos dos demais blocos. Quando em `IDLE`, o circuito está pronto para inicializar o processamento de amostras e, após o recebimento da primeira amostra válida (`valid=1`), o estado atual passa a ser `INPUTS`. O estado permanece em `INPUTS` até que todas as N amostras da FFT tenham sido recebidas e processadas, concluindo assim o primeiro estágio. A partir do segundo estágio, as amostras são processadas segundo a frequência de operação *clock* estipulada. Se o número de estágios ($log_2 N$) da FFT for par, o processamento termina no terceiro estágio (`SR0_SR1`), senão, no quarto e último estágio (`SR1_SR2`).

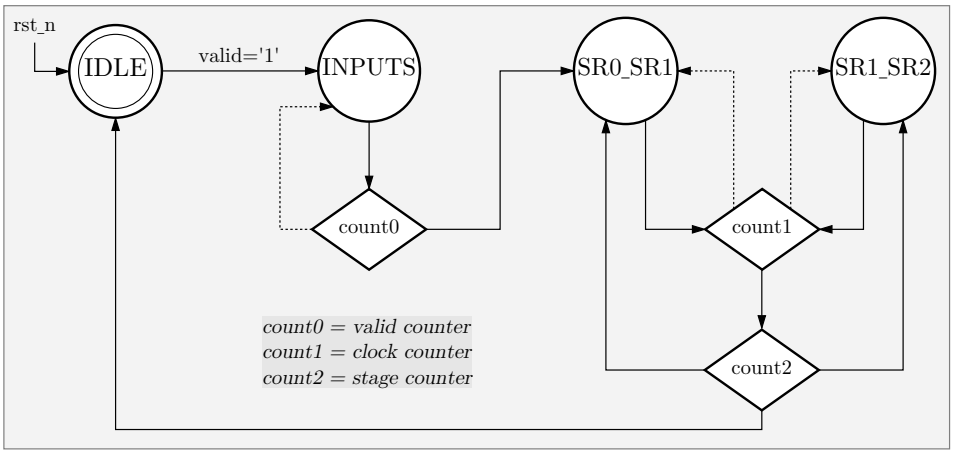

Figura 4.5 – FFT Totalmente Sequencial − FSM simplificada

Fonte: O Autor

Internamente, a máquina de estados funciona de acordo com os valores de três contadores (`count0`, `count1` e `count2`), conforme ilustrado na Figura 4.5. São estes valores que modificam os estados da máquina. O contador `count0` é responsável por contar as N amostras de entrada válidas. Após receber a última ($N-$ésima) amostra válida, a FFT começa a processar os estágios restantes de acordo com a frequência de *clock* (`count1`) do sistema, e a uma

taxa de duas amostras[1] por ciclo de relógio, ou seja, $N/2$ ciclos de relógio por estágio. Com isto, um terceiro contador se faz necessário – `count2` – responsável por contar o número de estágios processados. Ao atingir o número total de estágios, a máquina então se reinicializa e vai para o estado inicial `IDLE`, ficando pronta para o processamento de outro *frame* de dados.

4.3.2 Máquina de Estados - Gerador de Padrões

A máquina de estados (FSM) da FFT totalmente sequencial apresenta um circuito auxiliar, invisível a ela, que é responsável por habilitar e coordenar o chaveamento individual dos *shift registers* da memória de dados. Este circuito, denominado aqui de Gerador de Padrões, é formado também por um *shift register*. Mais especificamente, este gerador é formado por um contador binário do tipo Johnson, o qual pode ser percebido como um *shift register* já que é formado predominantemente por registradores. É imprescindível destacar que *shift registers*, de modo geral por sua natureza exclusiva de propagação de dados de forma sequencial, permitem aplicar técnicas específicas que explorem esta característica. Um exemplo são as técnicas de codificação de dados que aplicam estratégias eficientes para eliminação de redundâncias baseadas na correlação dos dados. (COSTA, 2002)

A Figura 4.6 representa um esquema geral simplificado do gerador de padrões da FFT. Esta figura é composta de três subfiguras 4.6 (a) 4.6 (b) e 4.6 (c) que são, pela ordem, uma derivação simplificada da memória de dados de uma FFT de 8 pontos e dos controles individuais necessários para o perfeito sincronismo dos dados em processamento através de controles individuais de linhas e colunas; uma derivação detalhada da memória de dados de uma FFT de 16 pontos onde é demonstrado o funcionamento individual dos padrões previamente percebidos em (a) e por último um exemplo de contador Johnson e suas saídas.

Conforme mencionado, a principal finalidade deste gerador de padrões é fornecer um controle individual sobre os registradores que compõem os *shift registers* da memória de dados através de um sinal `clock_enable`[2]. A grande vantagem deste sinal de controle é que somente os registradores que precisam ser atualizados são acionados, o que elimina chaveamentos desnecessários e proporciona redução de potência. Além disto, um outro efeito direto desta técnica é que a ferramenta de síntese consegue inferir, a partir dos sinais de *clock enable*, a geração de *clock gating*[3] de maneira automática.

[1] Uma *butterfly radix*-2 recebe duas amostras na entrada e entrega duas amostras na saída.

[2] Sinal de controle que (des)habilita a atualização do valor de um registrador.

[3] Técnica que desliga o sinal de *clock*, em pontos específicos do circuito, de forma a economizar potência. (MICHAEL et al., 2007)

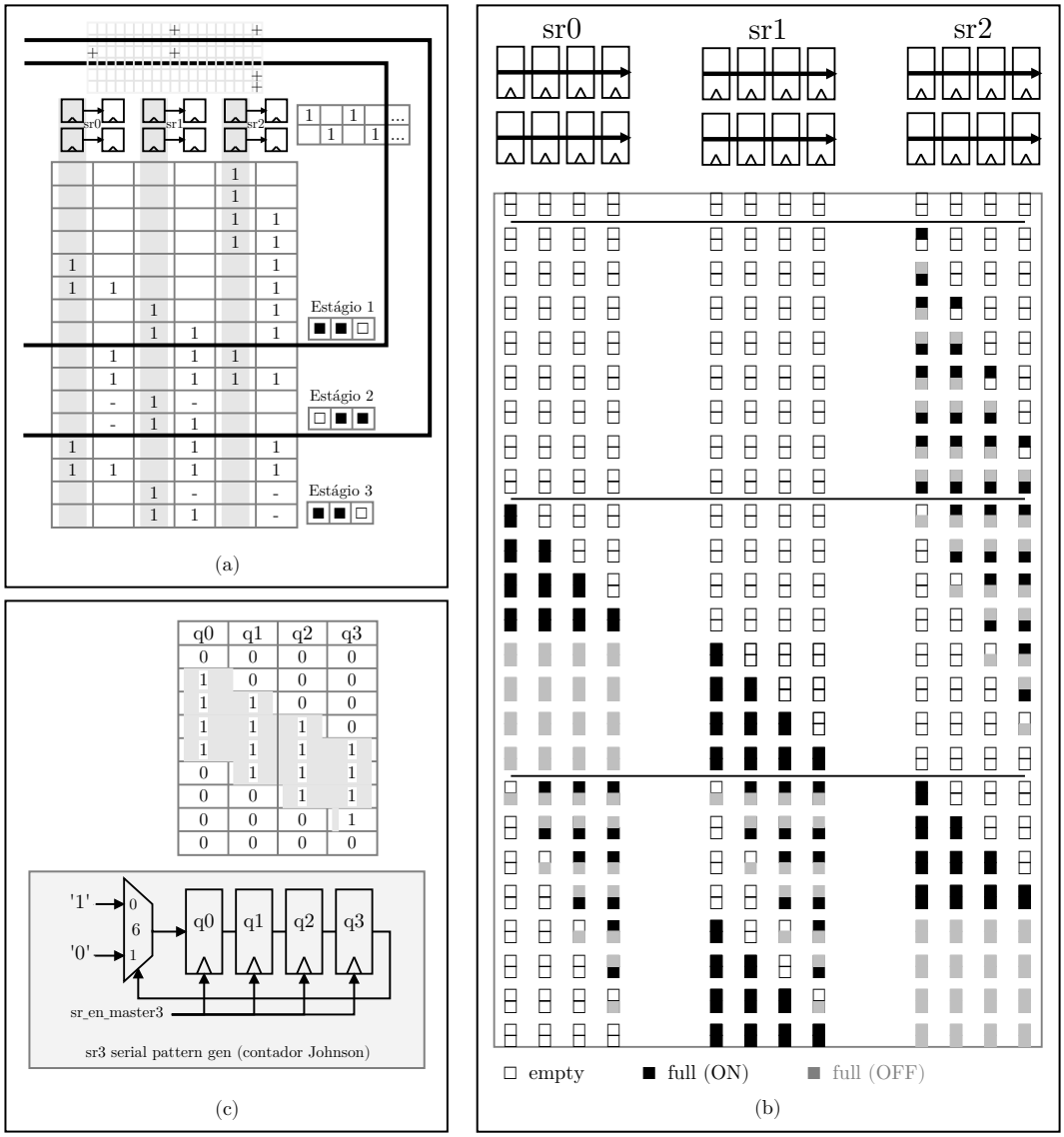

Figura 4.6 – FFT Totalmente Sequencial − Geração de padrões da FSM: (a) *shift register* da memória de dados e a derivação do seu controle em uma FFT de 8 pontos (b) o mesmo que (a), mas para uma FFT de 16 pontos (c) contador Johnson de $N/2$ bits

Fonte: O Autor

4.4 Coeficientes (*Twiddle factors*) para a arquitetura totalmente sequencial

Os coeficientes (*Twiddle Factors*) são valores constantes, e como tal, precisam ser armazenados. Armazenar dados em *hardware* requer algum tipo de memória, e, se o volume destes dados for elevado, o custo em memória também o será. Uma outra alternativa é a geração *on the fly* dos *twiddle factors* quando necessário, utilizando, por exemplo, o CORDIC. Conforme observado no capítulo 2, os valores dos *twiddle factors* são, na prática, componentes complexos (partes real e imaginária) que representam ângulos em sua forma cartesiana. Estes ângulos são

simétricos e espelhados ao longo de todo circulo unitário, ou seja, apresentam redundância, o que permite otimizações.

À primeira vista, a implementação mais direta dos *twiddle factors* é o seu armazenamento em memória. Supondo uma FFT de N pontos, seriam necessários $N/2 * 2 * n_{bits}$ bits de memória, sendo $N/2$ o total de *twiddle factors*, 2 componentes (real e imaginário) de cada *twiddle* e n_{bits} de largura de bits de cada *twiddle*. Para uma FFT de 128 pontos e 16 bits de resolução por componente do *twiddle*, ter-se-ia então $64 * 2 * 16$ totalizando 2048 bits de memória (2 kB), sem contar o *hardware* necessário para leitura/escrita. Como se observa, os requisitos de memória aumentam linearmente com o aumento do tamanho da FFT, o que para estruturas grandes pode ser um oneroso consumidor de recursos.

Na Figura 4.7, que apresenta uma estrutura de FFT explícita (direta) para um número de pontos $N = 8$, é possível perceber a ordem de processamento das *butterflies* conforme apontam, sequencialmente, `s1` (W_8^0, W_8^0, W_8^0 e W_8^0), `s2` (W_8^0, W_8^2, W_8^0 e W_8^2) e `s3` (W_8^0, W_8^1, W_8^2 e W_8^3).

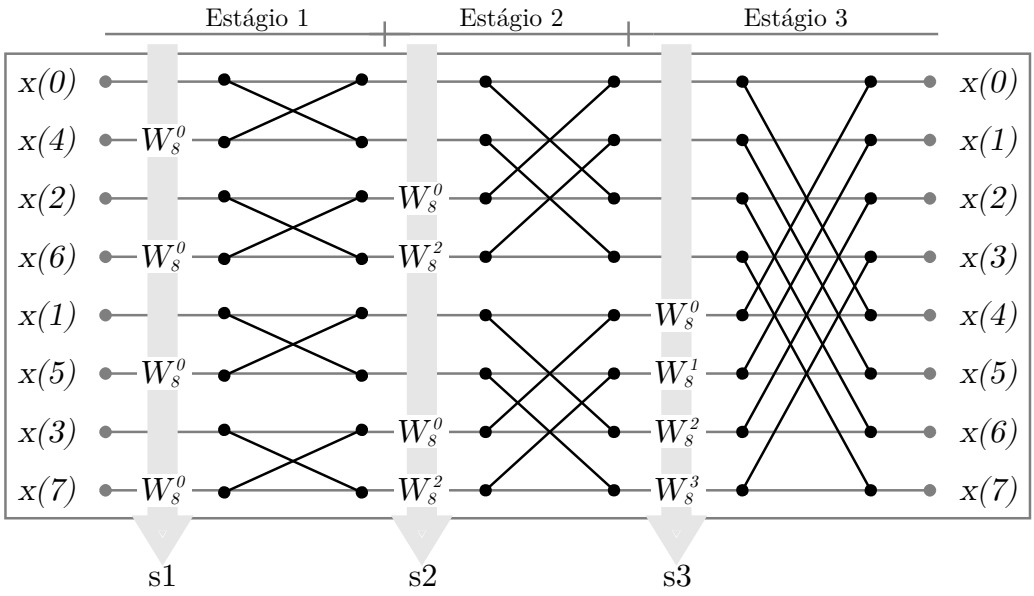

Figura 4.7 – Estrutura explícita (direta) de uma FFT com $N = 8$ pontos

Fonte: O Autor

Com base nas sequências exibidas na Figura 4.7, duas estratégias para o fornecimento dos coeficientes trigonométricos foram utilizadas, conforme demonstram os itens (a) e (b) da Figura 4.8. A estratégia (a) utiliza somente um *shift register* circular, com as sequências `s1`, `s2` e `s3` armazenadas explicitamente, ou seja, armazena integralmente os valores das partes real e imaginária. A estratégia (b) utiliza igualmente um *shift register* circular, e adicionalmente, um bloco CORDIC. O *shift register* desta estratégia armazena somente os ângulos dos *twiddle factors* das referidas sequências. Isso para fornecer estes ângulos para processamento pelo

CORDIC para a obtenção dos valores real e imaginário de cada coeficiente. A ideia central destas duas estratégias é avaliar o benefício de armazenar menos informações (somente os ângulos trigonométricos) ao custo da utilização do CORDIC.

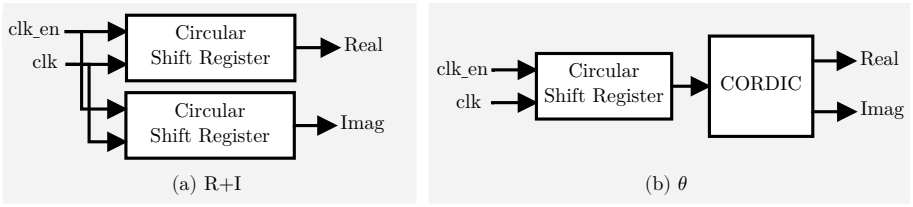

Figura 4.8 – FFT Totalmente Sequencial − Estratégias para geração de *Twiddle Factors*

Fonte: O Autor

4.5 Resultados de simulação e implementação

A Figura 4.9 apresenta um mapa com as variações das sínteses lógicas realizadas para a arquitetura totalmente sequencial. Quatro características orientam o mapa de sínteses: (*i*) estratégia para obtenção dos *twiddle factors*, (*ii*) *clock gating*, (*iii*) frequência de amostragem (f_s) e (*iv*) tamanho (N) da FFT. Duas frequências de amostragem foram consideradas: (*i*) 16 kHz e (*ii*) 32 kHz. Deste modo, as frequências máximas transformadas são de 8 kHz e 16 kHz, respectivamente. Além disto, quatro tamanhos de FFTs foram selecionados: (*i*) 32, (*ii*) 64, (*iii*) 128 e (*iv*) 256 pontos. Ao todo, trinta e duas (32) arquiteturas diferentes foram sintetizadas, e todas as variações afetam todos os blocos da arquitetura, com exceção do ítem (*i*), em que a variação é o próprio bloco. Mais detalhes sobre este ítem podem ser obtidos na seção 4.4.

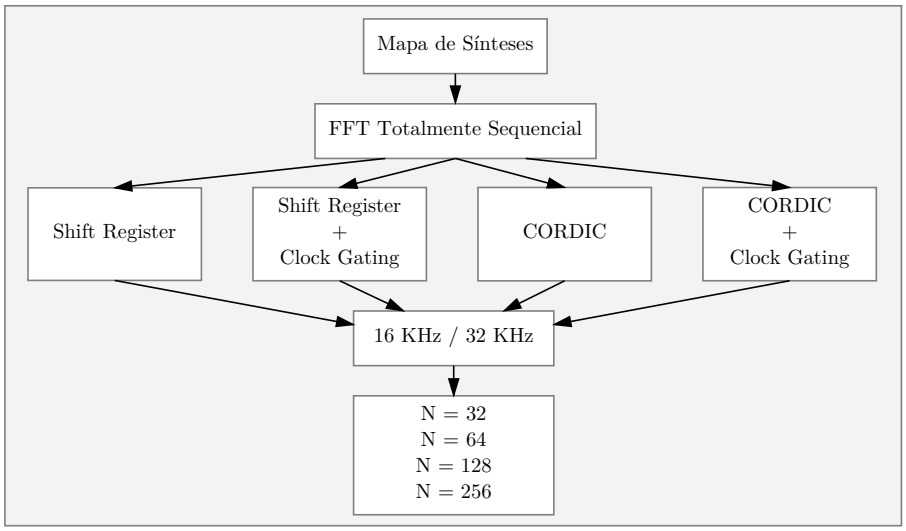

Figura 4.9 – FFT Totalmente Sequencial − Mapa de Sínteses

Fonte: O Autor

Para uma única unidade de processamento (*butterfly*), e considerando as frequências de amostragem fs mencionadas, a frequência de operação pode ser calculada diretamente como sendo o intervalo entre uma amostra e outra, divididos pelo número total de operações da FFT (menos as operações de um estágio $-$ o primeiro, $N/2$). O total de operações (rotações complexas) da FFT é o número de estágios vezes o número de operações por estágio. Assim sendo, a frequência de operação para a arquitetura FFT sequencial pode ser determinada de acordo com a Equação 4.1.

$$freq.operacao = \frac{1/f_s}{(log2(N) * N/2) - N/2} \tag{4.1}$$

De acordo com a Equação 4.1, e com os parâmetros de síntese definidos anteriormente, a frequência de operação mínima (ou período máximo de relógio) podem ser observados na Tabela 4.1. Como é possível observar, quando se tem uma frequência de amostragem (f_s) duas vezes maior, logo, duas vezes maior também será a frequência de operação. Visto que todo o processamento é realizado no intervalo entre uma amostra e outra ($1/F_s$), logo esta é a latência da FFT totalmente sequencial.

Tabela 4.1 – FFT Totalmente Sequencial $-$ Frequência de operação

	Número de Pontos (N) da FFT			
f_s	32	64	128	256
16 kHz	1.024 MHz	2.560 MHz	6.144 MHz	14.336 MHz
32 kHz	2.048 MHz	5.120 MHz	12.288 MHz	28.672 MHz

Fonte: O Autor

A Tabela 4.2 apresenta os resultados de potência total para diferentes parâmetros na arquitetura sequencial, em mW, para todas as arquiteturas de FFT totalmente sequencial desenvolvidas. Foram sintetizadas quatro diferentes estratégias onde a principal diferença entre elas é a utilização (ou não) do algoritmo CORDIC e a utilização (ou não) da técnica de *clock gating*.

Tabela 4.2 – FFT Totalmente Sequencial − Consumo de potência total (mW)

N	SR		SR+CG		SR+CORDIC		SR+CORDIC+CG	
	16 kHz	32 kHz	16 kHz	32 kHz	16 kHz	32 kHz	16 kHz	32 kHz
32	1.41	1.29	0.42	0.42	0.96	1.33	0.27	0.27
64	3.31	4.08	0.95	0.96	2.05	2.22	0.59	0.60
128	8.30	13.89	2.13	7.39	5.83	9.36	1.27	5.15
256	32.46	38.06	19.32	22.94	20.24	25.46	10.99	14.29

Fonte: O Autor

A Figura 4.10 coloca os dados da Tabela 4.2 em perspectiva gráfica. As linhas da tabela estão plotadas no eixo x da figura, ou seja, o consumo de potência.

Como pode ser observado, a potência aumenta de acordo com o número de pontos da FFT. Também, com o aumento da frequência, a potência também aumenta. Isso devido ao fato da principal componente de dissipação de potência (potência dinãmica), ser diretamente proporcional à frequência de operação. Observa-se também que, o uso do CORDIC na estratégia *shift register* habilita a redução da dissipação de potência. Entretanto, o uso do *clock gating* habilita ainda mais reduções de potência na estratégia *shift register*. Isso devido ao fato da estratégia *clock gating* desabilitar o relógio em partes estratégicas do circuito, o que evita atividades de transição desnecessárias. Desta forma, a implementação mais eficiente é aquela que usa a combinação do CORDIC e do *clock gating* na implementação *shift register*, como pode ser visto na última coluna da Tabela 4.2.

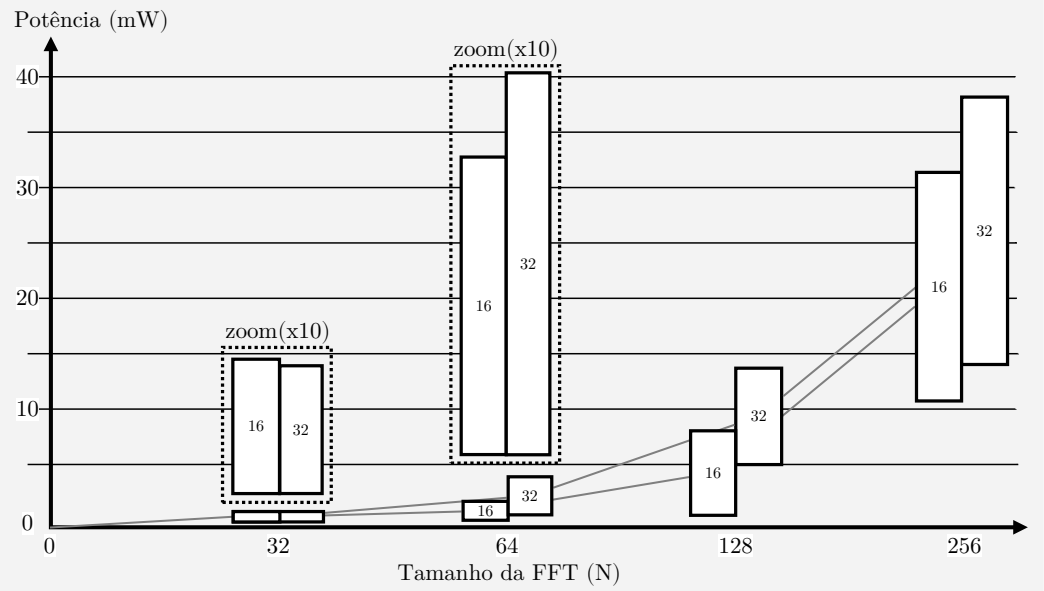

Figura 4.10 – FFT Totalmente Sequencial − Consumo de potência total (Tabela 4.10)

Fonte: O Autor

As tabelas a seguir apresentam os resultados individuais agrupados segundo a estratégia de implementação dos *twiddle factors* (SR ou CORDIC), frequência de operação (16 kHz e 32 kHz) e estratégia de síntese (com ou sem *clock gating*).

A Tabela 4.3 apresenta os resultados detalhados de síntese para cada um dos blocos da versão FFT sequencial, com *twiddle factors* explicitamente implementados em *shift register*. Como se observa, o armazenamento dos *twiddle factors* (partes real e imaginária − $\Re$ e $\Im$), é responsável pela maior parte do consumo de potência da FFT. Isso devido ao fato de precisar de elementos de armazenamento, que são mais complexos em termos de implementação em *hardware*.

Tabela 4.3 – FFT Totalmente Sequencial − Consumo de Potência (mW) − Versão *Shift Register*

	Shift Register (SR)							
Frequência	16 kHz				32 kHz			
N	32	64	128	256	32	64	128	256
total	1.41	3.31	8.30	32.46	1.29	4.08	13.89	38.06
twiddle (R+I)	0.95	2.52	6.74	26.59	0.89	3.12	11.22	28.66
sr0, sr1 e sr2	0.26	0.55	1.28	4.72	0.22	0.69	1.94	7.63
fsm, sr3 e outros	0.08	0.10	0.14	0.37	0.065	0.12	0.20	0.41
butterfly	0.12	0.13	0.13	0.78	0.12	0.15	0.53	1.35

Fonte: O Autor

A Tabela 4.4 apresenta resultados de síntese detalhados para a versão FFT sequencial na versão *shift register* com a técnica de *clock gating*. De acordo com os resultados apresentados, observa-se a importância do uso da técnica *clock gating* nos elementos que usam registradores. Para esses elementos, há uma redução significativa de potência, como pode ser comparado nas Tabelas 4.3 e 4.4.

Tabela 4.4 – FFT Totalmente Sequencial − Consumo de Potência (mW) − Versão SR+CG

		Shift Register (SR) + Clock Gating (CG)							
		16 kHz				32 kHz			
#	N	32	64	128	256	32	64	128	256
1	total	0.42	0.95	2.13	19.32	0.42	0.96	7.39	22.94
2	twiddle (R+I)	0.29	0.73	1.68	16.23	0.29	0.73	5.79	18.53
3	sr0, sr1 e sr2	0.09	0.19	0.40	1.69	0.09	0.19	0.69	2.74
4	fsm, sr3 e outros	0.03	0.03	0.04	0.25	0.02	0.03	0.09	0.31
5	butterfly	0.01	0.01	0.01	1.15	0.01	0.01	0.81	1.36

Fonte: O Autor

A Tabela 4.5 a seguir apresenta resultados de síntese detalhados para a versão FFT sequencial com o uso do CORDIC. Observa-se que, por si só, o CORDIC já habilita redução de potência na geração dos *twiddle factors* (twd CORDIC na Tabela 4.5), como pode ser comparado nas Tabelas 4.3 e 4.5. Isso atesta a importância da exploração do uso do algoritmo CORDIC nas implementações das arquiteturas FFT.

Tabela 4.5 – FFT Totalmente Sequencial − Consumo de Potência (mW) − Versão CORDIC

		CORDIC							
		16 kHz				32 kHz			
#	N	32	64	128	256	32	64	128	256
1	total	0.96	2.05	5.83	20.24	1.33	2.22	9.36	25.46
2	twd (CORDIC)	0.54	1.31	4.02	14.66	0.74	1.42	6.55	16.65
3	sr0, sr1 e sr2	0.23	0.49	1.43	4.94	0.31	0.54	2.29	6.93
4	fsm, sr3 e outros	0.07	0.09	0.15	0.37	0.09	0.09	0.23	0.42
5	butterfly	0.13	0.16	0.23	0.27	0.19	0.16	0.29	1.45

Fonte: O Autor

A partir dos resultados apresentados nas tabelas anteriores, chega-se à conclusão de que

o uso conjunto das estratégias CORDIC e *clock gating* pode habilitar significativas reduções de potência na implementação *shift register*. Isso pode ser comprovado nos resultados da Tabela 4.6, para ambas as frequências de operação, e, como pode ser comparado com as tabelas anteriores.

Tabela 4.6 – FFT Totalmente Sequencial − Consumo de Potência (mW) − Versão CORDIC+CG

		CORDIC + Clock Gating							
		16 kHz				32 kHz			
#	N	32	64	128	256	32	64	128	256
1	total	0.27	0.59	1.27	10.99	0.27	0.59	5.15	14.29
2	twd (CORDIC)	0.15	0.36	0.83	8.65	0.15	0.36	3.99	9.97
3	sr0, sr1 e sr2	0.09	0.19	0.39	1.82	0.09	0.19	0.76	2.97
4	fsm, sr3 e outros	0.02	0.03	0.04	0.17	0.02	0.03	0.14	0.38
5	butterfly	0.01	0.01	0.01	0.35	5.081	5.081	0.28	0.98

Fonte: O Autor

4.5.1 Trabalhos Relacionados da Literatura

De acordo com (DIRLIK, 2013b), a maneira mais comum de implementação de uma FFT é o uso da estratégia *pipelined* para o aumento do *throughput* da arquitetura. O trabalho aqui apresentado segue essa tendência com a proposição de uma arquitetura totalmente sequencial usando a estratégia *pipeline*.

O trabalho em (DIRLIK, 2013b) apresenta uma comparação entre as diferentes estruturas de implementação de arquiteturas FFT. O trabalho apresentado em (LIU; FENG, 2007), por exemplo, propõe um processador FFT reconfigurável de baixa potência e alta velocidade. O projeto consiste de um elemento de processamento *radix*-2 (PE), dois *radix*-4 PE e dois *radix*-8 PE, que são colocados juntos em uma arquitetura SDF (*Single-path Delay Feedback*) de *pipeline*. Entretanto, de acordo com (DIRLIK, 2013b), embora os autores afirmem terem proposto um projeto de baixa potência, os resultados não mostram isso, visto que a arquitetura apresenta 307.7 mW de dissipação de potência.

O trabalho apresentado em (ZHAO; ERDOGAN; ARSLAN, 2005), tem por objetivo obter o equilíbrio ideal entre baixa potência e alta flexibilidade. O sistema pode ser reconfigurado para executar FFTs de 16 a 1024 pontos usando apenas um bloco de borboleta. Entretanto, as soluções propostas não podem ser consideradas de baixa potência, visto que apresentam em

média 68,7 mW e 81,8 mW para as arquiteturas não reconfigurável e reconfigurável, respectivamente.

O trabalho em (ANBARASAN; SHANKAR, 2012) apresenta uma arquitetura usando uma estrutura reconfigurável com multiplicadores por constantes e multiplicadores paralelos de bits (usando o algoritmo de multiplicação de Booth) para gerar os *twiddle factors*. De acordo com os autores, isso reduz o custo do *hardware*, em comparação com o uso de uma grande memória ROM.

Outros trabalhos exploram algumas técnicas no sentido de reduzir o tamanho da memória ROM utilizada, como em (SRIDHARAN; VIJI, 2010), que para reduzir a área do circuito, usa a notação canônica CSD (*Digital Signal Digit*) e uma unidade sem multiplicador que não armazena todos os *twiddle factors* na memória ROM. Apenas alguns *twiddle factors* são armazenados e o resto pode ser derivado usando apenas operações de deslocamento e soma. Entretanto, para reduzir o consumo de energia, os autores levaram em conta que as entradas serão apenas reais. Desta forma, as borboletas do primeiro estágio são modificadas para ignorar a entrada imaginária. Isso pode ser um problema para algumas aplicações alvo que também requeiram os valores imaginários.

O trabalho apresentado em (SATHASIVAM; REDDY, 2016) usa multiplicação complexa para a geração dos *twiddle factors*, o que proporciona o armazenamento prévio dos mesmos na unidade ROM. Esses *twiddle factors* são metade do número de pontos da FFT e, com base no estágio utilizado, o número de coeficientes necessários também varia. A arquitetura proposta foi sintetizada em CMOS 0.18 μm, e funciona com um tamanho de entrada de 36 bits na frequência de 100 MHz, e consome 39 mW, em uma tensão de alimentação de 1,8 V.

Todos os trabalhos mencionados até o momento não fazem o uso do algoritmo CORDIC para a geração dos *twiddle factors*. Entretanto, como mostrado na literatura, bem como neste trabalho de dissertação, a forma mais eficiente da geração dos *twiddle factors* é com o uso do algoritmo CORDIC. Neste contexto, o trabalho em (CHIKHALIYA; DAVE; TIWARI, 2016) propõe uma arquitetura FFT baseada no algoritmo CORDIC para a geração dos coeficientes. Entretanto, apesar dos autores afirmarem que a arquitetura proposta baseada no CORDIC reduz amplamente a utilização de recursos de memória, o que é fato, não apresentam resultados de área e potência que suportem tal afirmação.

Em (PRINCY; KUMAR, 2017) é proposta uma arquitetura FFT binária, baseada no uso do algoritmo CORDIC. De acordo com os autores, trata-se da primeira arquitetura que computa os valores de sequência de saída, usando o algoritmo CORDIC. Apesar dos autores afirmarem que a solução proposta reduz os resquisitos de memória, bem como a quantidade de recursos de

hardware, não há resultados que confirmem tal afirmação.

O trabalho apresentado em (SHALINI; MANJUNATHA, 2018) propoõe uma arquitetura FFT de N pontos usando o algoritmo CORDIC. O multiplicador Vedic, baseado em multiplicação *butterfly*, é usado para melhorar a velocidade e, o processador CORDIC modificado é usado para otimmizar a utilização de área no cálculo dos *twiddle factors*. Embora os autores apresentem comparações e ganhos com outros trabalhos da literatura, os resultados apresentados são baseados apenas em plataforma FPGA, o que não é recomendável para avaliação de valores de dissipação de potência.

Embora os trabalhos mencionados de alguma forma procurem ao máximo otimizar os recursos de *hardware*, para uma implementação de uma FFT de baixa dissipação de potência, nenhum deles utiliza a estratégia de *clock gating*, juntamente com o algoritmo CORDIC, como é realizado nesta dissertação. Esse aspecto será ainda mais explorado, com possíveis novas otimizações propostas até o final desta dissertação. A Tabela 4.7 mostra as pricnipais contribuições da literatura, bem como a contribuição da nossa solução proposta.

Tabela 4.7 – Sumário dos Trabalhos Relacionados

#	Referência	***A***	***B***	***C***	***D***	***E***
1	(LIU; FENG, 2007)	-	√	√	-	-
2	(ZHAO; ERDOGAN; ARSLAN, 2005)	-	√	√	-	-
3	(ANBARASAN; SHANKAR, 2012)	-	√	√	-	-
4	(SRIDHARAN; VIJI, 2010)	-	√	√	-	-
5	(SATHASIVAM; REDDY, 2016)	-	√	√	-	-
6	(CHIKHALIYA; DAVE; TIWARI, 2016)	-	√	-	√	-
7	(PRINCY; KUMAR, 2017)	-	√	-	√	-
8	(SHALINI; MANJUNATHA, 2018)	√	-	-	√	-
9	**Solução proposta**	-	√	√	√	√

(A) Solução FPGA ***(B)*** Solução ASIC ***(C)*** Resultado de potência ASIC
(D) Uso do CORDIC ***(E)*** Uso do CORDIC e *clock gating*

Fonte: O Autor

4.6 FFT Totalmente Sequencial Otimizada

Com o desenvolvimento das primeiras versões do CORDIC e da FFT, foi possível perceber os blocos que demandavam mais consumo de potência. A partir disto, objetivou-se a

melhora específica destes blocos, de modo que estes resultados são apresentados e discutidos nesta seção. A Tabela 4.8 sumariza os resultados para uma FFT de 128 pontos e 16 KHz de frequência de amostragem.

Tabela 4.8 – FFT Totalmente Sequencial − Resultados das versões (1.0 e 2.0) implementadas (em mW)

	SR+CORDIC+CG (v.1)	CORDIC+ROM (v.2)	CORDIC+ROM+CG (v.2)
Total	1.27	1.75	0.76
Twiddle	0.83	0.008	0.0002
SR0+SR1+SR2	0.39	1.34	0.38
FSM+SR3+Outros	0.04	0.2	0.06
Butterfly	0.01	0.2	0.32

Fonte: O Autor

Para efeitos de comparação, a Tabela 4.8 apresenta os resultados de forma discriminada. É possível perceber que a utilização do CORDIC na v.1 para geração dos *twiddles* dissipa mais potência do que apenas armazenar estes valores em memória ROM. É observável que o maior consumidor de recursos da FFT versão 1.0 é a geração dos *twiddles* em combinação com *Shift Register*. Já a versão 2.0 da FFT implementada com CORDIC como multiplicador complexo em conjunto com a técnica de *clock gating* foi a arquitetura que obteve o menos consumo de potência entre as três. Nesta versão, os maiores consumidores de recurso passaram a ser então os *shift registers* e a *butterfly* implementada com o CORDIC.

5 CONCLUSÕES

Este trabalho apresentou um estudo sobre arquiteturas dedicadas do algoritmo CORDIC e a sua aplicação em arquiteturas dedicadas dos algoritmo FFT. Foram implementadas arquiteturas *unfolded* do algoritmo CORDIC nas versões com e sem *pipeline*. Em relação à FFT, foi proposta uma arquitetura totalmente sequencial parametrizável, com o uso do CORDIC para a geração dos *twiddle factors*.

O CORDIC é um algoritmo que calcula funções Trigonométricas e Hiperbólicas. É utilizado em estruturas de circuitos digitais para processamento de sinais. Por apresentar baixa complexidade computacional e realizar processos iterativos, baseados apenas em operações de soma/deslocamento para obtenção do resultado final, sua utilização se torna atrativa quando da implementação de circuitos visando baixa dissipação de potência.

Ainda que o CORDIC tenha sido apresentado à comunidade científica na década de 50, quando se fala em processamento eficiente de sinais, o CORDIC é um algoritmo a ser considerado. Isto se deve principalmente à sua baixa complexidade computacional, ou seja, o *hardware* que implementa o CORDIC utiliza somente circuitos de soma e subtração.

A partir do desenvolvimento de um modelo em alto nível do CORDIC, utilizando a ferramenta GNU/Octave, observou-se o comportamento funcional do algoritmo de forma a estabelecer o espaço de exploração arquitetural com vistas à otimização de potência. Com base neste modelo, foram inseridas funções de (des)quantização a fim de aproximar o modelo inicial de um modelo quantizado, ou seja, de um circuito real passível de ser implementado. Após, procedeu-se com a descrição do *hardware* em linguagem VHDL para validar a estrutura implementada.

Por fim, realizou-se a síntese física ASIC dos circuitos implementados e foram obtidos resultados para as métricas de interesse – área, atraso e potência, para CMOS 45 nm Nangate Open Cell Library utilizando a ferramenta Cadence RTL Compiler. Verificou-se que há uma relação direta entre o número de bits e iterações utilizados pelo CORDIC e o impacto disto em *hardware*. O melhor ajuste em termos de precisão é dependente de aplicação, ou seja, o número de bits na entrada do circuito e o número de iterações que o CORDIC deve executar estão individualmente relacionados com a precisão exigida pela aplicação final.

Os bons resultados obtidos com a exploração do algoritmo CORDIC em *hardware*, motivaram a sua utilização em arquiteturas FFT. Para tal, foram implementadas arquiteturas FFT sequenciais para 32, 64, 128 e 256 pontos para duas frequência de operação alvo (16 e 32 kHz). Num primeiro momento (FFT versão 1.0), foram usadas duas estratégias de implementação

para o armazenamento dos coeficientes. A primeira baseada totalmente no uso de *shift registers* e a segunda baseada em uma implementação híbrida usando *shift registers* e CORDIC (armazena somente os ângulos dos *twiddle factors*. O uso de *shift registers* habilitou o uso da técnica *clock gating*.

Os resultados mostraram que na FFT versão 1.0, a combinação da implementação híbrida (*shift register + CORDIC*) é a melhor estratégia para a redução de potência. Além disso, o uso do *clock gating* sempre habilita reduções de potência nas diferentes implementações. Desta forma, a implementação mais eficiente (em termos de redução de potência), é aquela que combina a implementação híbrida e o uso do *clock gating*.

Através da observação dos resultados obtidos pelas técnicas implementadas na arquitetura da FFT versão 1.0, uma nova bateria de explorações arquiteturais foi desenvolvida dando origem a uma FFT Totalmente Sequencial versão 2.0. A principal modificação foi feitas exatamente no bloco que consumia mais recursos (potência), que era justamente o bloco de geração dos *twiddles*. Além disto, quatro novas versões do CORDIC foram desenvolvidas, sendo que a melhor destas versões do CORDIC foi utilizada como multiplicador complexo. A geração dos *twiddles* utilizando-se o CORDIC combinado com *Shift Register* foi substituído por uma ROM. Os resultados obtidos demonstram que houve melhoras significativas no consumo total quando comparadas com a versão 1.0, sendo que a técnica de *clock gating* foi mantida por apresentar melhores resultados do que as demais técnicas de implementação.

5.1 Sugestões de Trabalhos Futuros

Diante dos resultados obtidos ao longo desta dissertação, com resultados expressivos obtidos com a exploração do algoritmo CORDIC na arquitetura FFT, abre-se uma perspectiva de novos trabalhos nesta linha. Desta forma, os novos desafios devem envolver:

1- Exploração de somadores compressores na estrutura da borboleta da FFT. Como a borboleta da FFT é composta por circuitos somadores e subtratores, novos operadores compressores deverão ser explorados visando a otimizações de dissipação de potência;

2- Exploração do paralelismo na arquitetura FFT, de acordo com uma aplicação alvo, aparelhos auditivos;

3- Fazer novas otimizações no CORDIC. A exploração aprofundada do CORDIC permitiu verificar novas oportunidades de otimizações, que deverão ser exploradas como trabalho futuro;

4- Realizar o algoritmo CORDIC e a FFT explorando a codificação de operandos. Em

[COS 2002] foram propostos novos operadores aritméticos operando em uma codificação diferente da binária. O objetivo é o de reduzir tanto a atividade de chaveamento quanto a lógica interna dos blocos. Esses novos operadores deverão ser explorados, tanto no CORDIC quanto na estrutura da FFT.

5.2 Publicação no Tema do Trabalho

Decorrente dos resultados obtidos nessa dissertação, o seguinte trabalho foi apresentado e publicado em congresso científico:

André N. Sapper, Leonardo Soares, Eduardo Costa e Sergio Bampi. "Exploring the Combination of Number of Bits and Number of Iterations for a Power-Efficient Fixed-Point CORDIC Implementation". In: IEEE International Conference of Electronics, Circuits, and Systems, 2017, Batumi, Georgia. Proceedings of the 24th ICECS, 2017. p.1-4.

REFERÊNCIAS

ANBARASAN, A.; SHANKAR, K. Design and implementation of low power fft/ifft processor for wireless communication. In: **International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME)**. [S.l.: s.n.], 2012. p. 152–155.

APPLE. 2017. <https://www.apple.com/>. Acessado em 01/07/2018.

BERGLAND, G. Fast fourier transform hardware implementations–a survey. **IEEE Transactions on Audio and Electroacoustics**, IEEE, v. 17, n. 2, p. 109–119, 1969.

BERGLAND, G. Fast fourier transform hardware implementations–an overview. **IEEE Transactions on Audio and Electroacoustics**, IEEE, v. 17, n. 2, p. 104–108, 1969.

BYBELL, T. Gtkwave. **URL: http://gtkwave. sourceforge. net**, 2010.

CHEN, K.-T. et al. A cordic algorithm with improved rotation strategy for embedded applications. **Journal of Industrial and Intelligent Information Vol**, v. 3, n. 4, 2015.

CHIKHALIYA, J.; DAVE, C.; TIWARI, J. Design and implementation of fft processor using cordic algorithm. **International Journal of Innovative and Emerging Research in Engineering**, Available online at www.ijiere.com, v. 3, n. 4, p. 143–148, 2016.

COMMITTEE, I. D. A. S. et al. Std 1076-2008, ieee standard vhdl language reference manual. **IEEE, New York, NY, USA**, 2008.

COOLEY, J. W.; TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. **Mathematics of computation**, JSTOR, v. 19, n. 90, p. 297–301, 1965.

COSTA, E. A. C. **Operadores Aritméticos de Baixo Consumo para Arquiteturas de Circuitos DSP. 2002**. Thesis (PhD) — Tese (Doutorado em Ciência da Computação)—Programa de Pós-Graduação em Computação-UFRGS, Porto Alegre, RS, 2002.

CRANDALL, R.; KLIVINGTON, J.; MITCHELL, D. Large-scale ffts and convolutions on apple hardware. **Y2010/unit4/FFTapps**, v. 20090909, 2009.

DIRLIK, S. **A comparison of FFT processor designs**. Dissertation (Master) — University of Twente, 2013.

DIRLIK, S. **A comparison of FFT processor designs**. [S.l.]: Department of EEMCS, University of Twente, 2013.

DONGARRA, J.; SULLIVAN, F. Guest editors' introduction: The top 10 algorithms. **Computing in Science & Engineering**, IEEE, v. 2, n. 1, p. 22–23, 2000.

EATON, J. W.; BATEMAN, D.; HAUBERG, S. **Gnu octave**. [S.l.]: Network thoery London, 1997.

FONSECA, M. B. et al. Exploração de operadores aritméticos na transformada rápida de fourier. Universidade Federal de Santa Maria, 2010.

GINGOLD, T. Ghdl-where vhdl meets gcc. **Online, December**, 2011.

GROGINSKY, H. L.; WORKS, G. A. A pipeline fast fourier transform. **IEEE Transactions on Computers**, IEEE, v. 100, n. 11, p. 1015–1019, 1970.

HON, T. T.; MARSONO, M. N. Hardware design space exploration of cordic algorithm for run-time reconfigurable platform. In: THE SOCIETY OF DIGITAL INFORMATION AND WIRELESS COMMUNICATION. **The First International Conference on Green Computing, Technology and Innovation (ICGCTI2013)**. [S.l.], 2013. p. 1–7.

JOHNSON, S. G.; FRIGO, M. Implementing ffts in practice. **Fast Fourier Transforms**, Rice University, Houston TX: Connexions, 2008.

KAUR, S.; SINGH, K. Implementation of high speed fixed point cordic techniques. **International Journal of Computer Applications**, Foundation of Computer Science, v. 56, n. 3, 2012.

LAKSHMI, B.; DHAR, A. Cordic architectures: A survey. **VLSI design**, Hindawi Publishing Corp., v. 2010, p. 2, 2010.

LIU, G.; FENG, Q. Asic design of low-power reconfigurable fft processor. In: **7th International Conference on ASIC (ASICON2007)**. [S.l.: s.n.], 2007. p. 44–47.

LIU, Y.; FAN, L.; MA, T. A modified cordic fpga implementation for wave generation. **Circuits, Systems, and Signal Processing**, Springer, v. 33, n. 1, p. 321–329, 2014.

LUNDY, T.; BUSKIRK, J. V. A new matrix approach to real ffts and convolutions of length 2 k. **Computing**, Springer, v. 80, n. 1, p. 23–45, 2007.

MEHER, P. K.; PARK, S. Y. Cordic designs for fixed angle of rotation. **IEEE transactions on very large scale integration (VLSI) systems**, IEEE, v. 21, n. 2, p. 217–228, 2013.

MEHER, P. K. et al. 50 years of cordic: Algorithms, architectures, and applications. **IEEE Transactions on Circuits and Systems I: Regular Papers**, IEEE, v. 56, n. 9, p. 1893–1907, 2009.

MICHAEL, K. et al. **Low Power Methodology Manual for System-on-Chip Design**. [S.l.]: New York: Springer Publications, 2007.

MOORE, G. Moore's law. **Electronics Magazine**, v. 38, n. 8, p. 114, 1965.

NEJI, N. et al. Fpga implementation of the cordic algorithm for fingerprints recognition systems. **International Journal of Computer Applications**, Foundation of Computer Science, v. 63, n. 6, 2013.

NEUENFELD, R. H. **Otimização de estruturas de borboletas para arquitetura de transformada rápida de Fourier de baixa dissipação de potência**. Dissertation (Master) — Dissertação (Mestrado em Engenharia Eletrônica e Computação)—PGEEC-UCPel, Pelotas, RS, 2016.

PRINCY, R. M.; KUMAR, S. A. Implementation of binary fft using cordic algorithm. **Journal of Network Communications and Emerging Technologies (JNCET)**, EverScience Publications, v. 7, n. 3, p. 37–40, 2017.

QURESHI, F. **Optimization of rotations in FFTs**. Thesis (PhD) — Linköping University Electronic Press, 2012.

RICHARD, G. L. et al. Understanding digital signal processing. **Prentice Hall, ISBN**, v. 131089897, p. 121–125, 2004.

SAMSUNG. 2017. <https://www.samsungfoundry.com/foundry/homepage.do>. Acessado em 01/07/2018.

SAPPER, A. N. et al. Exploring the combination of number of bits and number of iterations for a power-efficient fixed-point cordic implementation. In: IEEE. **Electronics, Circuits and Systems (ICECS), 2017 24th IEEE International Conference on**. [S.l.], 2017. p. 302–305.

SATHASIVAM, S.; REDDY, G. V. Asic implementation of high throughput fft processor for scientific applications. **Indian Journal of Science and Technology**, v. 9, n. 5, p. 1–5, 2016.

SHALINI, J.; MANJUNATHA, Y. R. Fpga based efficient n-point fft architecture using cordic for advanced ofdm. **International Journal of Engineering Research and Technology**, v. 11, n. 1, p. 11–27, 2018.

SRIDHARAN, A.; VIJI, A. Low power hardware implementation of high speed fft core. In: **International Conference on Computer Engineering**. [S.l.: s.n.], 2010. p. 223–227.

SWAIN, S. R.; PATNAIK, S. K. Design and implementation of cordic algorithm using vhdl. **International Journal of Emerging Trends in Electrical and Electronics (IJETEE–ISSN: 2320-9569) Vol**, v. 11, 2015.

THOMPSON, C. D. Fourier transforms in vlsi. **IEEE Transactions on Computers**, IEEE, n. 11, p. 1047–1057, 1983.

TIWARI, B.; GOEL, N. Implementation of a fast hybrid cordic architecture. In: IEEE. **Computational Intelligence & Communication Technology (CICT), 2016 Second International Conference on**. [S.l.], 2016. p. 702–706.

TSMC. 2017. <http://www.tsmc.com/english/dedicatedFoundry/technology/future_rd.htm>. Accessado em 01/07/2018.

VOLDER, J. E. The cordic trigonometric computing technique. **IRE Transactions on electronic computers**, IEEE, n. 3, p. 330–334, 1959.

WANG, S.; PIURI, V. A unified view of cordic processor design. In: **Application Specific Processors**. [S.l.]: Springer, 1997. p. 121–160.

WINOGRAD, S. On computing the discrete fourier transform. **Mathematics of computation**, v. 32, n. 141, p. 175–199, 1978.

YAVNE, R. An economical method for calculating the discrete fourier transform. In: ACM. **Proceedings of the December 9-11, 1968, fall joint computer conference, part I**. [S.l.], 1968. p. 115–125.

ZHAO, Y.; ERDOGAN, A.; ARSLAN, T. A low-power and domain-specific reconfigurable fft fabric for system-on-chip applications. In: **19th IEEE International Parallel and Distributed Processing Symposium**. [S.l.: s.n.], 2005. p. 4.

APÊNDICE A — ESPECIFICAÇÃO DA FFT SEQUENCIAL - *RADIX*-2

Este capítulo apresenta os principais detalhes técnicos de implementação da FFT Sequencial *Radix*-2. A função deste capítulo é ser a referência técnica na implementação de modificações, de modo a organizar e orientar possíveis melhorias.

A.1 Introdução

Este trabalho apresenta os detalhes técnicos de implementação utilizados no projeto de uma FFT. A implementação de uma FFT pode tornar-se inviável a medida em que cresce (número de pontos), ainda que possua uma estrutura regular. Pensando nisto, um dos objetivos deste documento é orientar o desenvolvimento desta estrutura, facilitando a visualização e/ou a alteração das técnicas e soluções empregadas. Não é intenção deste documento esgotar os assuntos relacionados ao projeto de uma FFT, mas revelar o estágio atual de desenvolvimento de uma FFT parametrizável.

A elaboração deste "Gerador de FFTs" pretende explorar os diversos aspectos de projeto que são inerentes ao desenho de uma FFT, tornando-os elegíveis e/ou parametrizáveis. A principal vantagem disto é a possibilidade de atender diferentes requisitos de projeto, realizando-se, para isto, apenas a alteração de parâmetros genéricos no bloco de maior hierarquia. Isto permite uma rápida percepção numérica e de recursos computacionais entre as diferentes possibilidades de implementação de forma a viabilizar comparações e decisções de projeto.

De forma a exemplificar o que foi exposto, um dos recursos considerados na exploração arquitetural de uma FFT é o algoritmo CORDIC. Inúmeros trabalhos estão disponíveis na literatura, relatando a implementação de estruturas eficientes de circuitos para processamento de sianis utilizando este algoritmo. Não diferente, este trabalho pretende orientar a utilização do referido algoritmo com vistas a avaliar sua contribuição na arquitetura de uma FFT parametrizável.

A.2 Cenários de Simulação (Testbench) das FFTs

Uma etapa igualmente importante a etapa de codificação de um sistema digital em HDL é a simulação funcional. Esta etapa consiste em avaliar numericamente a precisão do sistema implementado. A figura abaixo representa uma estrutura básica genérica para a realização de

uma simulação funcional. Como pode ser visto, existem 3 elementos principais em tal estrutura: (*i*) o gerador de estímulos; (*ii*) o design que está sendo simulado (DUT) e o (*iii*) observador das saídas do *design*.

Para a simulação funcional das FFTs, idealizou-se o cenário representado na figura abaixo. A principal diferença em relação a estrutura acima é a geração "local" dos estímulos de entrada. Isto evita utilizar dados externos provenientes de arquivos ou memória, facilitando e agilizando o processo de avaliação funcional. Vale dizer que a utilização deste cenário prevê a inclusão do bloco IFFT.

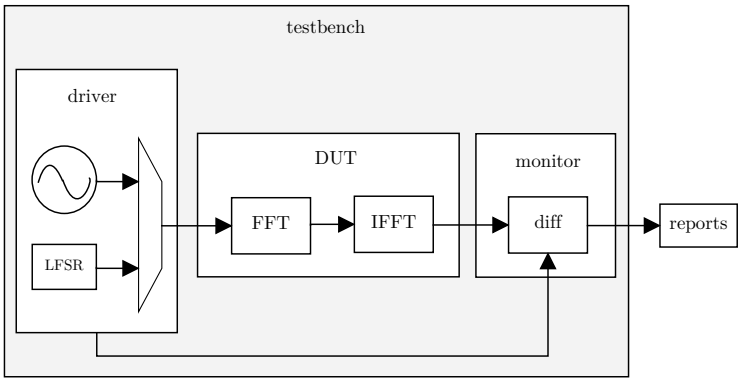

Figura A.1 – FFT Totalmente Sequencial – Testbench

Fonte: O Autor

A.3 FFT Totalmente Sequencial - Máquina de Estados (FSM) - Visão Detalhada

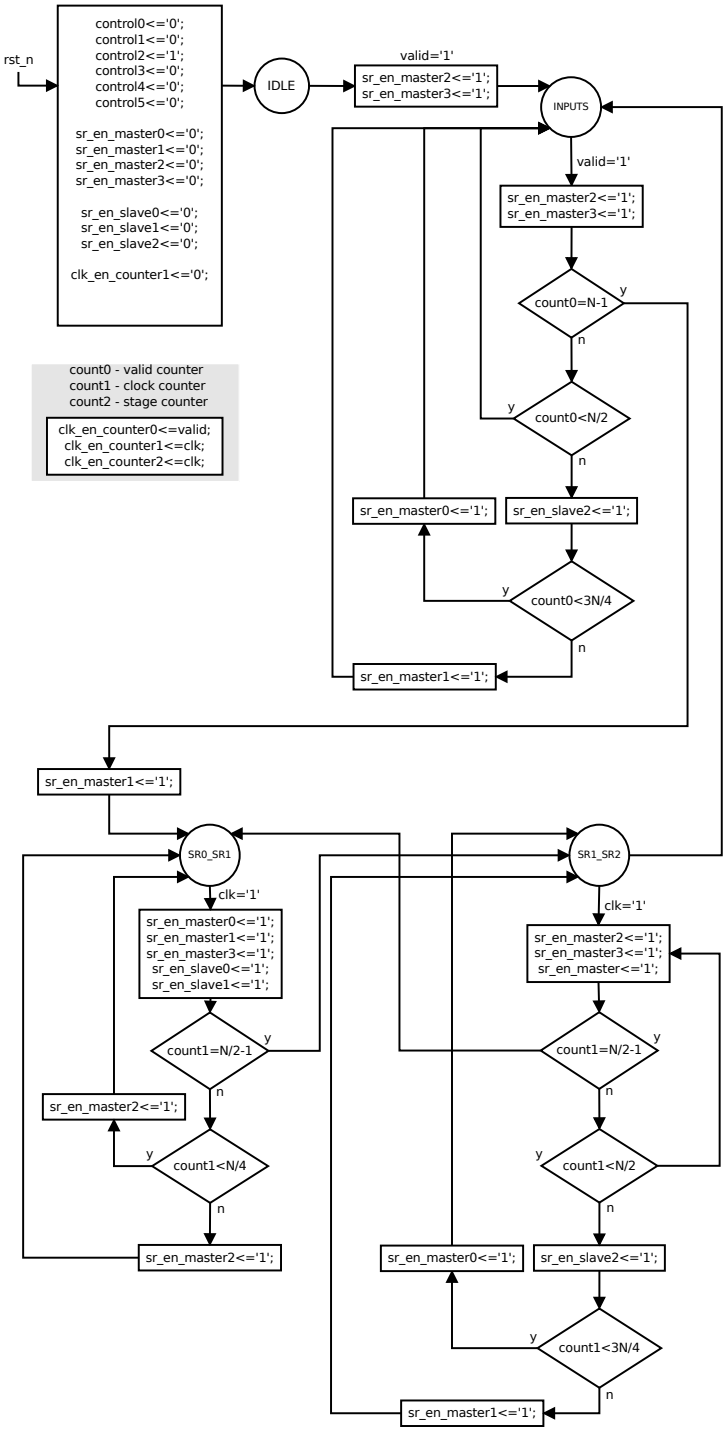

Figura A.2 – FFT Totalmente Sequencial − FSM − Visão detalhada

Fonte: O Autor