UNIVERSIDADE FEDERAL DO PAMPA

João Guilherme Nizer Rahmeier

Otimização de Área e Dissipação de Potência em Arquiteturas da Transformada Rápida de Fourier utilizando Multiplicação de Constantes Múltiplas

Alegrete, RS

Junho de 2016

João Guilherme Nizer Rahmeier

Otimização de Área e Dissipação de Potência em Arquiteturas da Transformada Rápida de Fourier utilizando Multiplicação de Constantes Múltiplas

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Sidinei Ghissoni

Alegrete, RS

Junho de 2016

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

R147o Rahmeier, João Guilherme Nizer
Otimização de Área e Dissipação de Potência em Arquiteturas da Transformada Rápida de Fourier utilizando Multiplicação de Constantes Múltiplas / João Guilherme Nizer Rahmeier.
83 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA ELÉTRICA, 2016.
"Orientação: Sidinei Ghissoni".

1. Transformada Rápida de Fourier. 2. Multiplicação de Constantes Múltiplas. 3. Otimização PPA. 4. Microeletrônica. I. Título.

João Guilherme Nizer Rahmeier

Otimização de Área e Dissipação de Potência em Arquiteturas da Transformada Rápida de Fourier utilizando Multiplicação de Constantes Múltiplas

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Trabalho aprovado. Alegrete, RS, 25 de Junho de 2016:

Prof. Dr. Sidinei Ghissoni
Orientador

Prof. Me. Crístian Müller
UNIPAMPA

Prof. Dr. Ewerson Luiz de Souza Carvalho
UNIPAMPA

Alegrete, RS
Junho de 2016

Dedico este trabalho à minha mãe Nubia Cristina Nizer
e ao meu pai Dalmir Rubens Rahmeier,
pelo apoio e incentivo em todos os momentos.

Agradecimentos

Gostaria de agradecer a toda a minha família que esteve sempre acompanhando minha trajetória, saindo do interior do Paraná para vir adquirir conhecimento no baita chão. Agradeço especialmente a minha mãe, Nubia Cristina Nizer, e ao meu pai, Dalmir Rubens Rahmeier, por todo o apoio financeiro e emocional; sem a ajuda de vocês dois essa realização seria impossível. Agradeço a minha namorada, Waleska Stephanie da Cruz, que conheci no intercâmbio e que me deu muita força em diversos momentos dessa trajetória, inclusive neste trabalho, me incentivando e me cobrando para fazer sempre o melhor.

Agradeço aos colegas de curso, parceiros, que estiveram presentes nas horas de confraternização, estudo, desespero e de alívio: Alian Engroff, Daniel Fernando Baú, Dionatas Longaretti, Haiglan Plotzki, John Jefferson Antunes Saldanha, Leonardo Tomazine Neto, Luiz da Silva Júnior, Marcelo Pereira Margalhães e Willian Deliberalli. Agradeço a todos os professores e colegas integrantes do Grupo de Arquitetura de Computadores e Microeletrônica - GAMA. Um agradecimento especial a todos os colegas e parceiros que contribuíram nos momentos de vivência no Alegrete. Também agradeço a todos os colegas de infância que ainda possuem laços de amizade e que sempre me quiseram bem. Um abraço especial ao grande amigo e compadre Gean Alchieri, que disponibilizou estadia durante a realização do estágio e, que acima de tudo, foi companheiro desde sempre. Agradeço também à Leticia Rieger, companheira de Rio Grande/Capanema, amiga sempre presente.

Gostaria também de agradecer a todos os amigos e amigas da chamada 127 do programa Ciência sem Fronteiras, que batalharam juntos na Florida A&M University em Tallahassee-FL. Desafio e superação são as palavras que definem nossa trajetória de intercâmbio. Obrigado por tudo!

Agradeço a todos, que de qualquer maneira contribuíram para que este trabalho se tornasse realidade!

Deus, obrigado!

"Os que se encantam com a prática
sem a ciência, são como os timoneiros que
entram no navio sem timão nem bússola,
nunca tendo certeza do seu destino".
(Leonardo da Vinci)

Resumo

Arquiteturas da Transformada Rápida de Fourier (FFT) são largamente utilizadas em sistemas de comunicação que utilizam multiplexação por divisão em frequências ortogonais. Esses sistemas compõem atualmente diversos padrões de comunicação, como: IEEE 802.11agjn, IEEE 802.20 acesso sem fio de banda larga móvel, linha digital assimétrica para assinante e transmissão de áudio e vídeo digital. Assim, este trabalho propõem uma metodologia utilizando soluções de Multiplicações de Constantes Múltiplas (MCM) para otimizar arquiteturas de estágio único, *radix-2* com decimação no tempo da FFT. Utiliza-se o algoritmo proposto por Cooley e Tukey para determinação dos coeficientes da FFT e o algoritmo proposto por Aksoy *et. al*, para resolver o problema MCM. Elabora-se um algoritmo para síntese automática das arquiteturas em linguagem de hardware SystemVerilog. Cada arquitetura é verificada funcionalmente e sintetizada logicamente utilizando a biblioteca de células XFAB $0.18\mu m$. A utilização dessa metodologia visa otimizar resultados de área e dissipação de potência para arquiteturas de 8 a 256 pontos.

Palavras-chaves: Microeletrônica, Transformada Rápida de Fourier, Multiplicação de Constantes Múltiplas, Otimização de PPA.

Abstract

Digital implementations of the Fast Fourier Transform (FFT) are widely used in communication systems using Orthogonal frequency-division multiplexing. These systems currently comprise several communication standards such as: IEEE 802.11agjn, IEEE 802.20 mobile wireless broadband access, asymmetric digital subscriber lines and digital audio and video broadcasting. Thus, this work proposes a methodology using a multiple constant multiplication (MCM) solution to optimize single stage, radix-2 decimation-in-time FFT processors. It uses the algorithm proposed by Cooley and Tukey to determine the coefficients of the FFT and the algorithm proposed by Aksoy *et. al* to solve the MCM problem. Also, is developed an algorithm that automatically elaborates the FFT architectueres using the hardware description language SystemVerilog. Each architecture is functionally verified and logically synthesized utilizing XFAB 0.18μmcells library. The usage of this methodology aims to optimize results in area and power comsumption in architectures from 8 up to 256 points.

Keywords: Microelectronics, Fast Fourier Transform, Multiple Constant Multiplication, PPA Optimization.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ADSL Linha Digital Assimétrica para Assinante, do Inglês *Assymetric Digital Subscriber Line*

BFS Busca em amplitude e profundidade, do Inglês *Breadth-First Search*

CSD Digito Canônico Sinalizado, do Inglês *Canonic Signed Digit*

CSE Eliminação de Subexpressões Comuns, do Inglês *Common Subexpression Elimination*

DAB Transmissão de Áudio Digital, do Inglês *Digital Audio Broadcasting*

DC software para síntese lógica, Design Compiler.

DFT Transformada Discreta de Fourier, do Inglês *Discrete Fourier Transform*

DIF Decimação em frequência, do Inglês *decimation-in-frequency.*

DIT Decimação no tempo, do Inglês *decimation-in-time*

DSP Processamento digital de sinais, do Inglês *digital signal processing.*

DVB Transmissão de Vídeo Digital, do Inglês *Digital Video Broadcasting*

FFT Transformada Rápida de Fourier, do Inglês *Fast Fourier Transform*

FIR Resposta Finita ao Impulso, do Inglês *Finite Impulse Response*

FSM Máquina de Estados finitos, do Inglês *Finite-State Machine*

MBWA Acesso sem fio de Banda Larga Móvel, do Inglês *Mobile Broadband Wireless Access*

MCM Multiplicação de Constantes Múltiplas, do Inglês *Multiple Constant Multiplication*

MDC Comutador por Atraso de Caminhos Múltiplos, do Inglês *Multi-path Delay Commutator*

MSD Dígito Mínimo Sinalizado, do Inglês *Minimal Signed Digit*

MSE Erro médio quadrático, do Inglês Mean-squared error

OFDM Multiplexação por Divisão de Frequências Ortogonais, do Inglês *Orthogonal Frequency-division Multiplexing*

PPA performance área e consumo de energia, do Inglês *performance, power and area.*

SDF Realimentação por atraso de caminho único, do Inglês Single-path Delay Feedback

SNR Razão Sinal-Ruído, do Inglês Singal-to-noise Ratio.

Sumário

1 Introdução

A otimização de performance, potência e área (PPA) em sistemas de processamento digital de sinais (DSP) tem ganhado atenção especial ao longo dos anos, principalmente devido à proliferação de dispositivos eletrônicos móveis. Uma das principais operações presentes em DSP é a transformada discreta de Fourier (DFT).

A DFT tem como propriedade transformar sinais no domínio do tempo para o domínio da frequência. Projetistas tiveram grande interesse nessa transformada por possibilitar que a convolução no domínio do tempo tornasse uma multiplicação ponto a ponto no domínio da frequência[1]. Como característica da convolução, sua implementação direta em hardware tem custos elevados. Além disso, por volta de 1971, Weinsten e Ebert sugeriram sua utilização para a demodulação em banda base [2].

Entretanto, a DFT requer um elevado custo computacional para sua implementação. Contudo, a criação de algoritmos que implementam-na de forma eficiente, as transformadas rápidas de Fourier (FFT), facilitaram sua utilização tanto em nível de software quanto no de hardware, contribuindo para os avanços nos sistemas digitais de sinais. Dentre os principais sistemas, encontram-se filtros digitais e sistemas de multiplexação por divisão de frequências ortogonais (OFDM), como: IEEE 802.11a,g,j,n, IEEE 802.20 acesso sem fio de banda larga móvel (MBWA), linha digital assimétrica para assinante (ADSL), transmissão de áudio e vídeo digital (DAB e DVB). Como reapresentado na Figura 1, a FFT encontra-se totalmente presente em nosso dia a dia: na transmissão de dados e acesso à redes WiFi, em centrais multimídias, smartphones, transmissão de dados via 4G entre outras. Otimizar essa transformada impacta diretamente nosso estilo de vida.

(a) Central multimídia.
Fonte: www.dvb.org/news.

(b) Recepção de vídeo digital.
Fonte: blog.toyota.co.uk.

(c) Redes WiFi.

Figura 1 – Aplicações da FFT em sistemas digitais.

Devido a grande utilização da FFT nos sistemas de comunicação modernos, metodologias para otimização dos circuitos que a implementam são importantes. As principais linhas de pesquisa estão relacionadas à: redução de memória para armazenamento de coeficientes (denominados em Inglês:*twiddle factors*), técnicas de otimização de multipli-

cadores complexos, implementações sem multiplicadores, uso de *pipeline* para otimizar desempenho, implementações com número de pontos adaptáveis, geração de coeficientes em hardware evitando uso de memórias, entre outras.

O presente trabalho aborda uma metodologia voltada na substituição de multiplicadores em arquiteturas de estágio único da FFT, com decimação no tempo (DIT), *radix-2*, visando otimização em área e consumo de energia do circuito integrado.

Como a implementação da FFT pode ser modelada como um problema de Multiplicação de Constantes Múltiplas (MCM), utiliza-se um algoritmo que encontra uma solução otimizada para o problema, substituindo os multiplicadores complexos. Estes novos blocos utilizam apenas somadores, compartilhamento de resultados parciais e deslocamentos binários.

Para obtenção dos dados de área e potência, as arquiteturas serão implementadas digitalmente utilizando linguagem de hardware SystemVerilog [4], e sintetizadas com tecnologia XFAB 0.18μm, utilizando ferramentas comerciais da Synopsys para projeto de circuitos integrados.

Em virtude da dificuldade e particularidade existente em mapear índices da FFT e codificar, em linguagem de hardware, arquiteturas com número de pontos (N) elevado, é interessante a proposta de uma ferramenta que faça isso de forma automática. Portanto, foram elaborados através da ferramenta Matlab, algoritmos que irão efetuar esse mapeamento, bem como converter a solução MCM para SystemVerilog e, de modo geral, descrever, instanciar e organizar todos os blocos das unidades FFT.

1.1 Motivação

A motivação deste trabalho está relacionada com a massiva presença de arquiteturas FFTs nos sistemas de comunicação digitais atuais, principalmente os que envolvem OFDM. Otimizar esses circuitos em área, potência e aumento na capacidade de processamento é importante para atender requisitos de portabilidade em dispositivos móveis e também contribuir para o aumento nas taxas de transmissão de dados.

1.2 Objetivos

Como objetivo geral deste trabalho consta:

- Desenvolver um algoritmo para a síntese automática de uma arquitetura em estágio único da FFT de N pontos, utilizando o método MCM para otimização de PPA.

Os objetivos específicos deste trabalho são:

- Elaborar um modelo no Matlab para a metodologia empregada;
- Implementar um algoritmo para converter a solução do problema MCM diretamente à linguagem de hardware SystemVerilog;
- Descrever um algoritmo para definição do mapeamento de índices da FFT para N pontos;
- Programar um algoritmo de síntese automática da arquitetura de estágio único, *radix-2* em DIT, através da seleção do número de pontos (N) e valor de quantização (Q) das entradas e coeficientes;
- Verificar funcionalmente as arquiteturas;
- Sintetizar logicamente as arquiteturas, utilizando softwares industriais, com a tecnologia XFAB 0.18μm;
- Analisar resultados de PPA das arquiteturas em nível lógico.

1.3 Organização do Trabalho

O restante deste trabalho encontra-se organizado da seguinte forma. No Capítulo 2 serão apresentados conceitos gerais sobre a transformada discreta de Fourier, seus custos computacionais e a maneira de realizá-la utilizando algoritmos rápidos da transformada de Fourier. O problema de multiplicação de constantes múltiplas é definido no Capítulo 3, juntamente com as considerações para se encontrar uma solução otimizada, explorando em detalhes o algoritmo utilizado neste trabalho. As etapas para elaboração do trabalho, considerações e alterações em algoritmo, bem como etapas de otimização e descrição das arquiteturas em linguagem de hardware são esclarecidas no Capítulo 4. Os resultados referentes a PPA, otimização em quantidade de operações e análises de erro e razão sinal-ruído nas arquiteturas são apresentados no Capítulo 5. As considerações finais e trabalhos futuros são explanadas no Capítulo 6.

2 Transformada de Fourier

Este capítulo fundamenta a DFT e apresenta suas particularidade quanto a utilização de recursos para implementação em hardware. São abordadas as formas de decimação da transformada, no tempo e na frequência, o que são os coeficientes e a maneira de obtê-los, além de explicar sobre *radix-r*. Por fim, será definido o algoritmo utilizado para a implementação da FFT, desenvolvido por Cooley e Tukey.

2.1 Transformada Discreta de Fourier - DFT

A DFT definida na Eq. (2.1), é uma aproximação da transformada de Fourier no domínio contínuo para funções discretas. Tem como objetivo transformar um conjunto de dados de entrada $x[n]$, no domínio do tempo, em um conjunto de dados de saída $X[k]$, no domínio da frequência[1].

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi kn/N}, \qquad 0 \leq k \leq N-1 \tag{2.1}$$

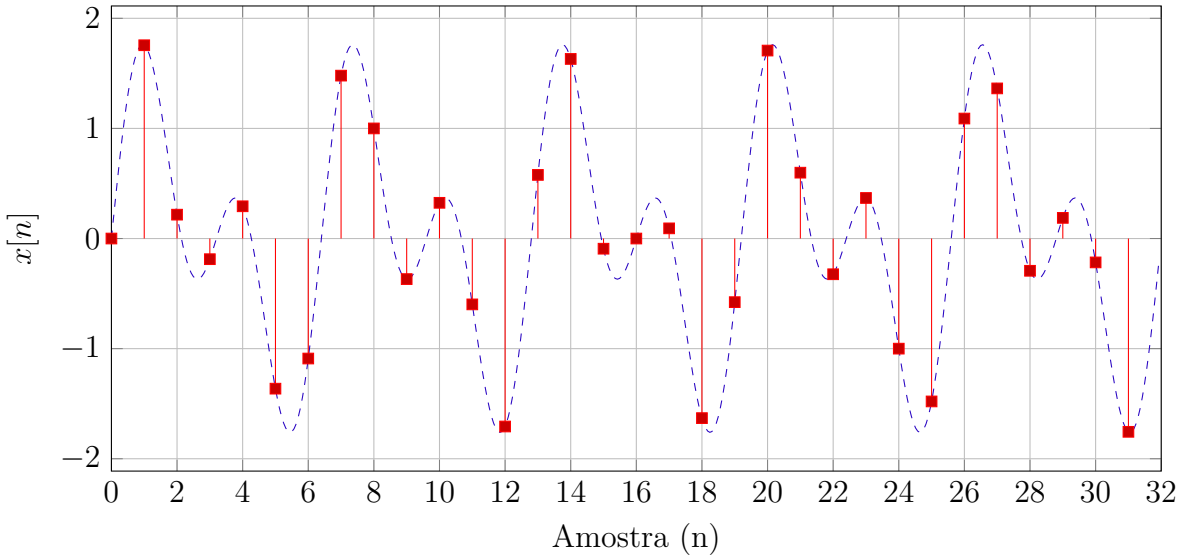
Onde, $X[k]$ é o k-ésimo resultado da transformada; $x[n]$ corresponde a n-ésima amostra; N é número de pontos da FFT; e i corresponde a $\sqrt{-1}$. Usualmente a componente exponencial é denominada *twiddle factor* ou raiz unitária, aqui referida como coeficiente. Esse coeficiente, definido na Eq. (2.2), é um número complexo ($\mathbb{C}$) com magnitude unitária.

$$W_N^{kn} = e^{-i2\pi kn/N} \tag{2.2}$$

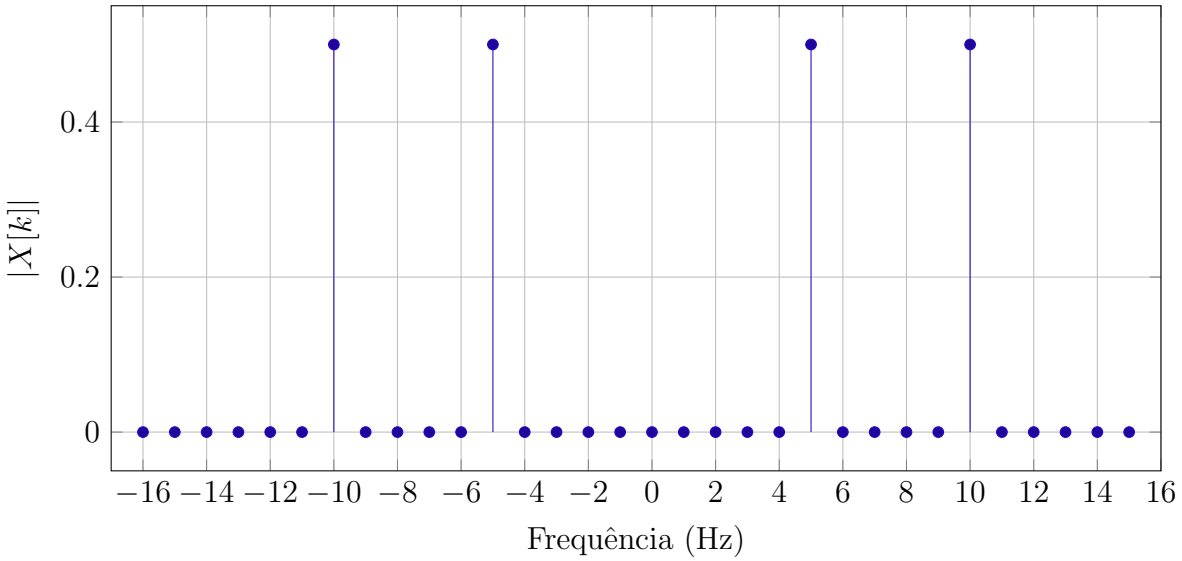
Assim como na transformada contínua de Fourier, o dado de entrada é decomposto em componentes senoidais ortogonais. Contudo, no caso discreto as frequências que compõem o sinal são representadas por pulsos discretos, geralmente referidos como funções delta de Dirac.

Esse comportamento pode ser observado na Figura 2. Em 2a, tem-se um sinal composto pela soma de dois sinais senoidais de frequência $f_1 = 5Hz$ e $f_2 = 10Hz$, ambos com amplitude $A = 1$, amostrado com frequência $f_s = 32Hz$. Aplicando a DFT com $N = 32$ pontos, obtêm-se o gráfico apresentado na Figura 2b. O resultado da transformada apresenta quatro deltas nas frequências: -10, -5, 5 e 10Hz. Comprovando que o sinal de entrada é composto unicamente por sinais com frequências fundamentais de $5Hz$ e $10Hz$.

Como discutido em [5], o espaçamento entre os pontos, no eixo-x, está relacionado com o número de pontos N e a frequência de amostragem f_s. Cada variação no eixo da



(a) Amostras do sinal $x[n]$ em vermelho e envoltória em azul.



(b) Magnitude do sinal $X[k]$.

Figura 2 – Exemplo da aplicação da DFT em um sinal senoidal.

frequência, do gráfico da transformada corresponde a $\Delta_{fe} = \frac{f_s}{N}$ Hz. Entretanto, este espaçamento não está relacionado com a resolução da DFT. Esta é dependente da quantidade de dados a serem transformados, denominado M, logo, $\Delta_f = \frac{f_s}{M}$ Hz. Neste trabalho será utilizado número de pontos igual ao número de dados. Assim, $M = N$ e $\Delta_f = \Delta_{fe}$.

2.1.1 Análise Computacional

Embora as funcionalidades da DFT serem interessantes, sua utilização se torna inviável quando aplicada para um número de pontos muito elevado. Considere a Equação

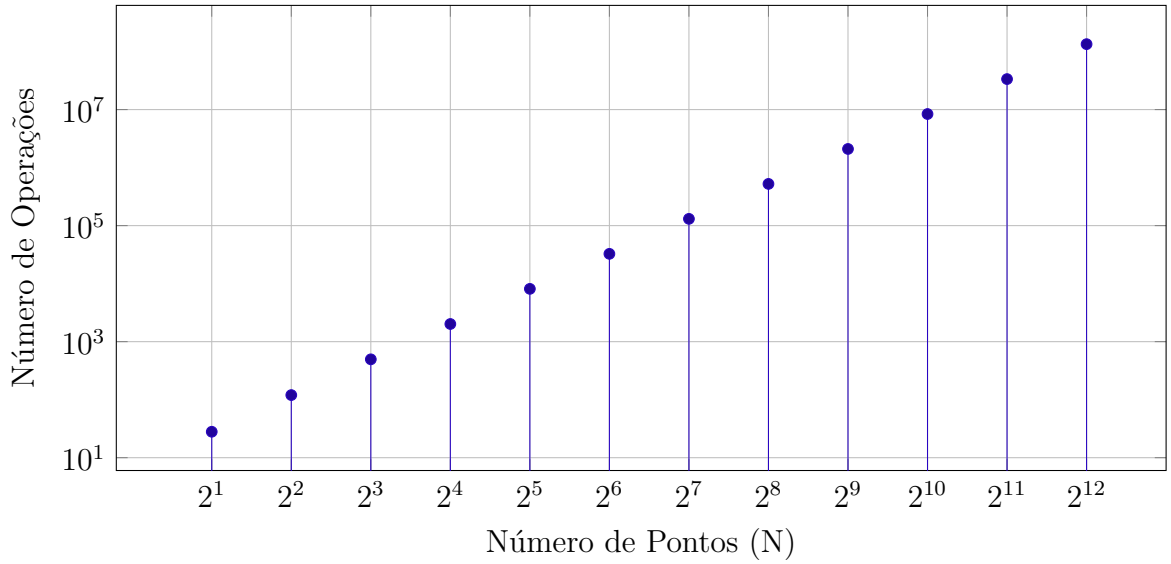


Figura 3 – Quantidade de operações (somas + multiplicações) para diferentes números de pontos N.

(2.3), onde a equação da DFT (2.1), é reescrita na forma matricial.

$$\begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-2] \\ X[N-1] \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & \cdots & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_N^0 & W_N^{N-2} & W_N^{2(N-2)} & \cdots & W_N^{(N-1)(N-2)} \\ W_N^0 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-2] \\ x[N-1] \end{bmatrix} \tag{2.3}$$

Analisando-se o lado direito da equação, verifica-se que a quantidade de multiplicações complexas necessárias é de N^2, seguida de $N(N-1)$ somas complexas [1]. Agora, considere que cada multiplicação complexa requer: quatro multiplicações reais e duas somas reais; e que uma soma complexa requer: duas somas reais. Ao total serão necessárias $4N^2$ multiplicações reais e $4N^2 - 2N$ adições reais.

Como apresentado na Figura 3, a quantidade de operações (somas + multiplicações) necessárias para a computação da DFT torna-a pouco atraente. Requisitos de PPA nas implementações em hardware são diretamente afetados, devido a quantidade de recursos necessários. É necessária a utilização de técnicas e métodos mais eficazes para a realização da DFT.

2.2 Transformada Rápida de Fourier - FFT

Como abordado na seção anterior, a DFT necessita de muitas operações, tornando-a ineficiente para implementações, especialmente em hardware. Baseado nisso, desenvolvem-se algoritmos que procuram implementá-la de forma mais eficiente. Esses algoritmos são

denominados transformadas rápidas de Fourier, usualmente referidos por FFTs. Neste trabalho será abordado o algoritmo proposto por Cooley e Tukey [6].

O método desenvolvido por Cooley e Tukey, baseia-se no princípio de simetria e periodicidade dos coeficientes. Desta maneira, simplificações podem ser obtidas na matriz de coeficientes, diminuindo de N^2 operações para menos de $2N \log_2 N$. Por ser um algoritmo clássico utilizado na literatura para implementação da FFT o mesmo será abordado neste trabalho.

2.2.1 Algoritmo de Cooley e Tukey

Nesta subseção serão apresentadas as etapas de simplificação da computação da DFT na forma de uma FFT, baseada no método de Cooley e Tukey.

Inicialmente, considere representar o número de pontos N como sendo composto pela multiplicação de dois números inteiros r_1 e r_2, assim, $N = r_1.r_2$. Agora, considere reescrever os índices k e n da Eq. (2.1), como:

$$\begin{array}{lll} k = k_1.r_1 + k_0 & k_0 = 0, 1, \cdots, r_1 - 1 & k_1 = 0, 1, \cdots r_2 - 1 \\ n = n_1.r_2 + n_0 & n_0 = 0, 1, \cdots, r_2 - 1 & n_1 = 0, 1, \cdots r_1 - 1 \end{array} \tag{2.4}$$

Obtêm-se que:

$$X[k_0, k_1] = \sum_{n_0=0}^{r_2-1} \sum_{n_1=0}^{r_1-1} x[n_0, n_1] W_N^{k(n_1 r_2 + n_0)} \tag{2.5}$$

Usando a propriedade de exponenciais, em que a soma de expoentes é equivalente a multiplicação de exponenciais elevadas aos componentes da soma, pode-se reescrever o coeficiente na seguinte forma:

$$W_N^{k(n_1 r_2 + n_0)} = W_N^{k n_1 r_2} W_N^{k n_0} \tag{2.6}$$

Atentando para a primeira parte do coeficiente reescrito e, substituindo k, como definido na Eq. (2.4), tem-se que:

$$W_N^{k n_1 r_2} = W_N^{k_1 n_1 r_2 r_1 + k_0 n_1 r_2} = W_N^{k_1 n_1 N} W_N^{k_0 n_1 r_2} \tag{2.7}$$

Da Eq. (2.7), verifica-se a parte importante da simplificação, pois o termo $W_N^{k_1 n_1 N}$, possui parte real unitária e imaginária zero. Desta maneira:

$$W_N^{k n_1 r_2} = 1 W_N^{k_0 n_1 r_2} \tag{2.8}$$

A partir dessa primeira simplificação pode-se reescrever a Eq. (2.5), como sendo:

$$X[k_0, k_1] = \sum_{n_0=0}^{r_2-1} \sum_{n_1=0}^{r_1-1} x[n_0, n_1] W_N^{k_0 n_1 r_2} W_N^{k n_0} \tag{2.9}$$

Agora, se considerado o somatório mais interno, conclui-se que ele depende unicamente das variáveis k_0 e n_0, portanto, definimos uma nova função $A[k_0, n_0]$ como:

$$A[k_0, n_0] = \sum_{n_1=0}^{r_1-1} x[n_0, n_1] W_N^{k_0 n_1 r_2} \tag{2.10}$$

A função $A[k_0, n_0]$, possui N elementos que necessitam de r_1 operações para serem computados. Após este passo, a Eq. (2.9) pode ser reescrita de acordo com a Eq. (2.11):

$$X[k_0, k_1] = \sum_{n_0=0}^{r_2-1} A[k_0, n_0] W_N^{(k_1 r_1 + k_0) n_0} \tag{2.11}$$

Possuindo os valores de $A[k_0, n_0]$, pode-se computar a Eq. (2.11) que, por sua vez, requer mais $N r_2$ operações. Como resultado final, o número total de operações necessárias para computar a DFT de N pontos, T_{op}, é:

$$T_{op} = N(r_1 + r_2) \tag{2.12}$$

Aplicando esse método de simplificação sucessivamente, iniciando na Eq. (2.10), tem-se um algoritmo de s estágios, que irão requerer um total de operações igual a:

$$T_{op} = N(r_1 + r_2 + \cdots r_s) \tag{2.13}$$

Adotando um valor de r_j fixo, igual a r, conclui-se que a FFT irá possuir quantidade de estágios correspondente á:

$$s = \log_r(N) \tag{2.14}$$

Usualmente, para facilitar implementações em hardware, escolhe-se um valor de r e N na base 2. Desta maneira, a quantidade final de operações, para a computação da DFT via FFT é:

$$T_{op} = rN \log_r(N) \tag{2.15}$$

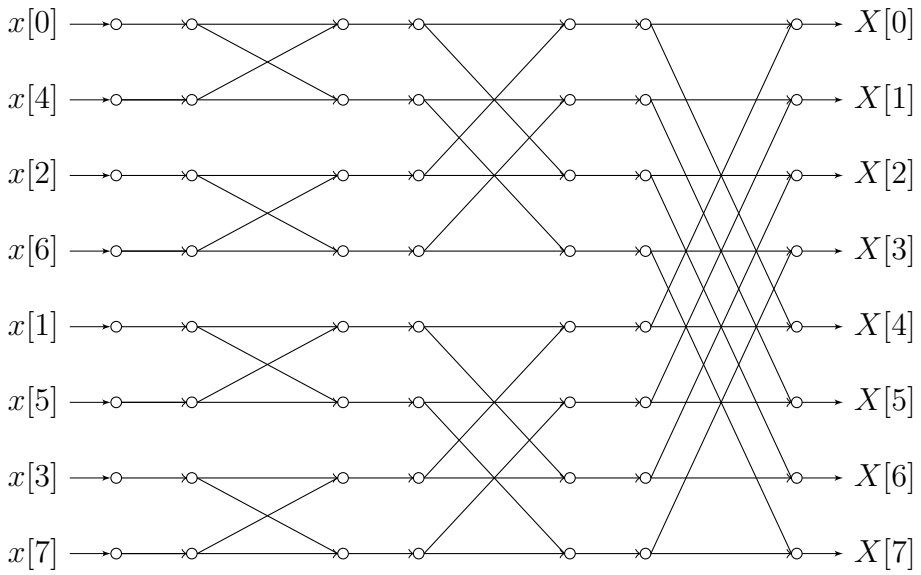


Figura 4 – Fluxo de dados para uma FFT DIT *radix-2* com N=8.

Fonte: Autor.

2.2.2 Formas de Decimação

Basicamente, o termo decimação se refere à decomposição de um sinal em duas bandas polifásicas. Os algoritmos da FFT podem ser classificados em dois tipos de conjuntos de acordo com a decimação: no tempo e na frequência.

2.2.2.1 Decimação no Tempo

Diz-se decimação no tempo (DIT) quando as divisões sucessivas do sinal em bandas polifásicas ocorre no conjunto de dados de entrada $x[n]$, como executado no algoritmo de Cooley e Tukey, e representado pela Eq. (2.16) com divisão em duas bandas.

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_{\frac{N}{2}}^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_{\frac{N}{2}}^{nk} \tag{2.16}$$

Como característica da FFT em DIT, demonstrada na Figura 4, os dados de entrada, $x[k]$, são ordenados na forma de *bit-reverse*. A Tabela 1 demonstra essa ordenação. O índice de um dado de entrada, in, pode ser representado em sua forma binária como $b_2 b_1 b_0$, assim, o correspondente índice na saída do algoritmo terá índice igual a out, na forma binária $b_0 b_1 b_2$ [1]. Isso deve ser considerado pelos projetistas para que a correta ordenação dos dados de entrada e saída seja efetuada.

Tabela 1 – Exemplo de indexação das entradas e saídas de uma FFT em DIT, para $N = 8$.

in		***out***	
0	000	000	**0**
4	100	001	**1**
2	010	010	**2**
6	110	011	**3**
1	001	100	**4**
5	101	101	**5**
3	011	110	**6**
7	111	111	**7**

2.2.2.2 Decimação em Frequência

O método de decimação em frequência (DIF) aplica a divisão do sinal de saída $X[k]$ em bandas polifásicas. A Eq. (2.17) demonstra essa divisão:

$$\begin{aligned} X[2l] &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] + x[\tfrac{N}{2} + n]) W_{\frac{N}{2}}^{2nl} \\ X[2l+1] &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x[\tfrac{N}{2} + n]) W_N^n W_{\frac{N}{2}}^{nl}, \qquad 0 \leq l \leq \tfrac{N}{2} - 1 \end{aligned} \tag{2.17}$$

Análogo ao método DIT, ocorre uma reordenação, na forma *bit-reverse*, dos dados de saída $X[k]$. Esse modelo de reordenação é apresentado na Tabela 2.

Tabela 2 – Exemplo de indexação das entradas e saídas de uma FFT em DIF, para $N = 8$.

in		***out***	
0	000	000	**0**
1	001	100	**4**
2	010	010	**2**
3	011	110	**6**
4	100	001	**1**
5	101	101	**5**
6	110	011	**3**
7	111	111	**7**

Na Figura 5 pode-se observar a maneira que ocorre o fluxo de dados em uma FFT DIF de 8 pontos.

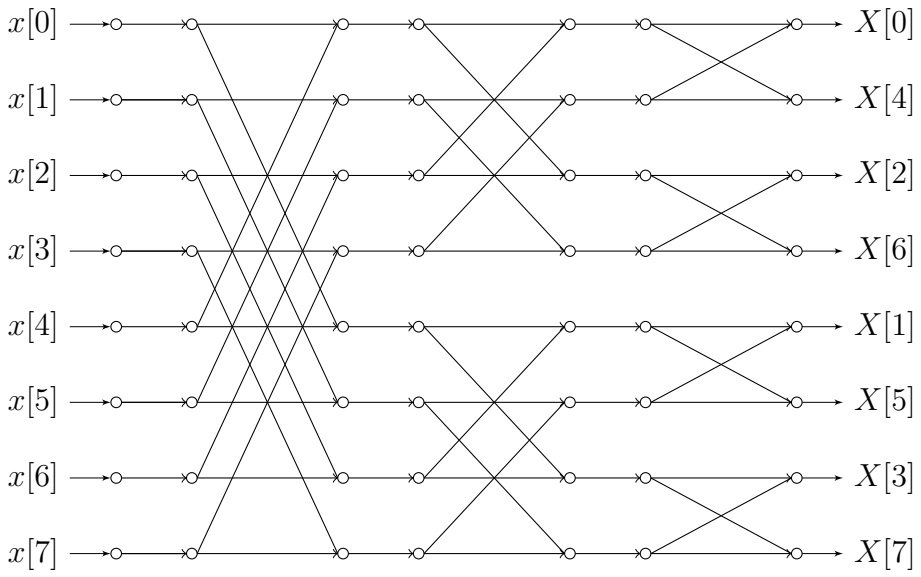


Figura 5 – Fluxo de dados para uma FFT DIF *radix-2* com N=8.

Fonte: Autor.

2.2.3 Definição de *radix*

A definição de *radix* está relacionada à quantidade de bandas em que o sinal é divido em cada estágio. Existem basicamente três categorias de algoritmos da FFT de acordo com o *radix* [1].

Se todos os estágios da FFT dividem o sinal em r bandas polifásicas, o algoritmo é denominado *radix-r*. Como exemplo, as FFTs apresentadas nas Figuras 4 e 5 são denominadas FFTs *radix-2* DIT e DIF respectivamente, por dividirem os sinais de entrada ou de saída em duas bandas. Neste trabalho utiliza-se *radix-2* que é o mesmo *radix* utilizado no algoritmo clássico de Cooley e Tukey.

Baseado também no trabalho de Cooley e Tukey, foi desenvolvido um método de decimação que utiliza mais de um *radix* para implementar o algoritmo da FFT. Esse método, proposto por [7], é denominado *mixed-radix*.

Outra maneira de executar a decimação é denominada *split-radix*, implementada por [8]. Esse método procura diminuir ainda a quantidade de operações, pelo fato de decompor a banda par utilizando *radix-2* e *radix-4* na ímpar. De forma simples, esse tipo de decomposição é descrito na Eq. (2.18).

$$X[k] = \sum_{n_2=0}^{\frac{N}{2}-1} x_2[n_2] W_{\frac{N}{2}}^{n_2 k} + W_N^k \sum_{n_4=0}^{\frac{N}{4}-1} x_4[n_4+1] W_{\frac{N}{4}}^{n_4 k} + W_N^{3k} \sum_{n_4=0}^{\frac{N}{4}-1} x_4[n_4+3] W_{\frac{N}{4}}^{n_4 k} \quad (2.18)$$

2.3 Implementação em Hardware da FFT

Nesta seção serão abordadas arquiteturas e métodos para a implementação da FFT em hardware. Inicialmente, é definido uma estrutura base, denominada borboleta, utilizada na maioria das implementações. Os métodos normalmente utilizados são definidos como: *pipelined*, baseadas em memória e de estágio único. O foco deste trabalho está voltado para as arquiteturas de estágio único.

2.3.1 Definição da Borboleta

A definição de borboleta, neste trabalho, refere-se a uma estrutura componente da FFT DIT. A borboleta complexa, apresentada na Figura 6, é um módulo base para implementação do algoritmo DIT *radix-2* [1].

Essa arquitetura implementa uma multiplicação complexa da entrada B por um coeficiente genérico TW. Após essa multiplicação, ocorre o cruzamento de dados e a realização de uma soma complexa de $A + B.TW$ e a subtração complexa de $A - B.TW$.

Como resultado dessas operações, obtêm-se os resultados intermediários da FFT, que são reutilizados através de indexamento e, quando no último processamento, representam os resultados finais da transformada.

Nos algoritmos *radix-2* esta estrutura é utilizada $N/2$ vezes em cada estágio, tornando-se um alvo para otimizações em diversos trabalhos [9][10][11].

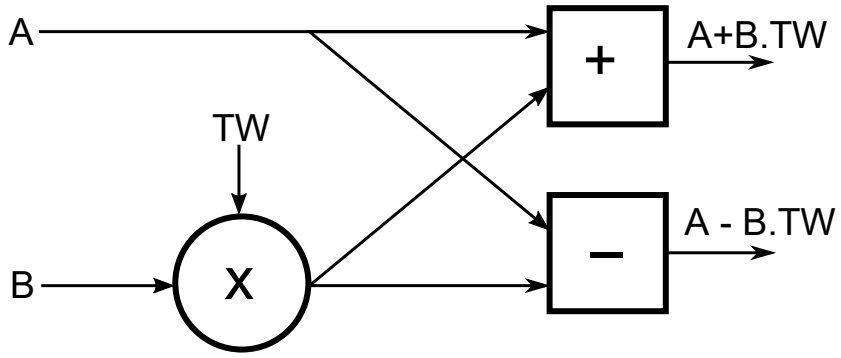


Figura 6 – Fluxo de dados de uma borboleta para FFT DIT *radix-2*.

Fonte: Autor.

2.3.2 Arquiteturas *Pipelined*

As arquiteturas *pipelined*são implementações que possuem grande capacidade de processamento de dados. Além disso, comumente são inseridos estágios de *by-pathing* que possibilitam a computação de FFTs de diferentes pontos em um mesmo chip. Como desvantagens, essas arquiteturas necessitam da implementação de blocos de memória para

armazenar os resultados obtidos entre um estágio e outro. Neste trabalho serão abordados dois tipos especiais de arquiteturas *pipelined*, os modelos de realimentação por atraso de caminho único (SDF) e comutador por atraso de caminhos múltiplos (MDC).

2.3.2.1 Realimentação por Atraso de Caminho Único - SDF

A Figura 7 demonstra um modelo de implementação *pipelined*do tipo SDF. São largamente utilizadas por necessitarem de uma quantidade de recursos limitados e também apresentarem grande desempenho para aplicações em tempo-real [12].

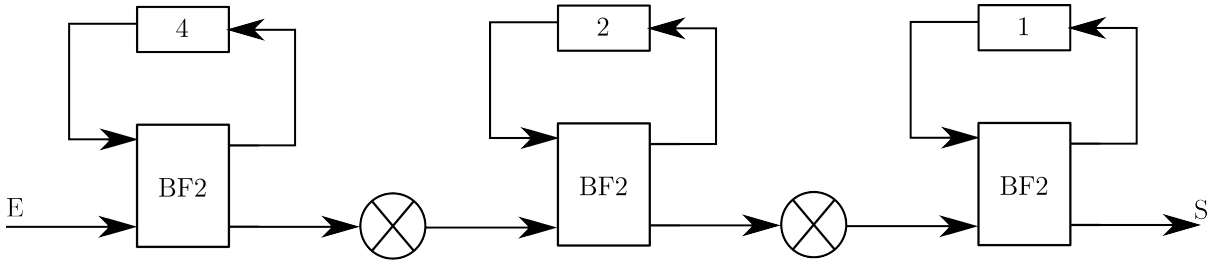


Figura 7 – Fluxo de dados de uma FFT *pipelined*SDF com 8 pontos, *radix-2*.

Fonte: Adaptado de [13].

As arquiteturas SDF tem como característica utilizar de forma eficiente os registradores. Cada conjunto de dados é disponibilizado para o multiplicador uma vez em cada estágio. A quantidade de multiplicadores complexos é baseada no tipo de borboleta utilizada, correspondente ao *radix* selecionado. Em [14], uma arquitetura SDF é proposta para evitar o uso de ciclos inativos durante a reconfiguração da quantidade de pontos, quando processando FFTs de tamanhos diferentes em sequência.

Outra metodologia para implementação de uma FFT *radix-16* SDF é apresentada em [15], buscando diminuir a quantidade de multiplicadores e somadores complexos, para aplicações WiMAX 802.16a. O trabalho proposto em [16] utiliza somadores e deslocamentos binários para substituir os multiplicadores em uma FFT *radix-2* SDF.

2.3.2.2 Comutador por Atraso de Caminhos Múltiplos - MDC

O modelo de implementação MDC procura dividir os dados de entrada em caminhos, grupos de dados, que são passados para a borboleta de acordo com atrasos coordenados pelo comutador. A Figura 8 apresenta esse modelo, para uma FFT *radix-2* e $N = 8$ pontos.

Em [17] é apresentada uma arquitetura MDC multi-*radix* para $N = 128$. O esquema de multiplicações é simplificado nos estágio 4 e 6, que requerem multiplicações apenas por $-i$. Além disso, componentes que exigem multiplicadores constantes utilizam da decomposição dos valores na forma de dígito canônico sinalizado (CSD). Essa mesma

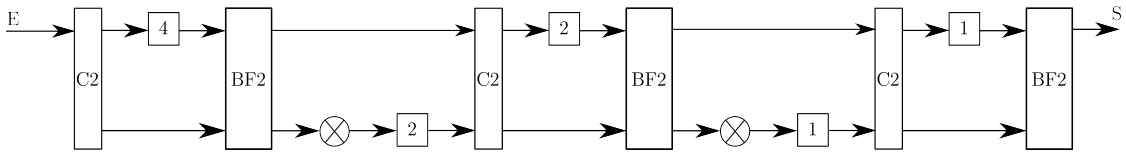


Figura 8 – Fluxo de dados de uma FFT *pipelined*MDC com 8 pontos, *radix-2*.

Fonte: Adaptado de [13].

metodologia é apresentada em [18], para uma FFT de $N = 256$, com simplificações nos estágios 5 e 8, de forma que, apenas no estágio 7, sejam utilizados multiplicadores complexos.

2.3.3 Arquiteturas Baseadas em Memória

As arquiteturas baseadas em memória tem como característica a implementação de uma única borboleta para processamento e o uso intenso de blocos de memória para armazenar resultados intermediários e também coeficientes. Na Figura 9 é apresentado um modelo genérico da organização dessa arquitetura.

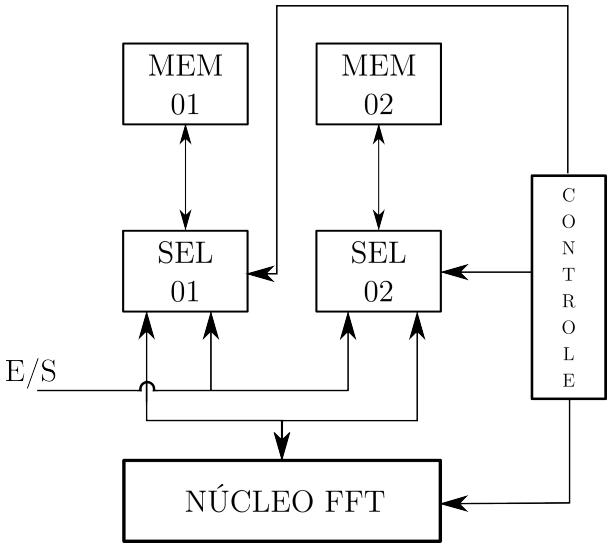


Figura 9 – Fluxo de dados de uma FFT baseada em memória.

Fonte: Autor.

Inicialmente, os dados são coletados e armazenados em memórias para processamento do primeiro estágio. Ao processar cada borboleta os dados são rearmazenados na memória. Na última etapa, os resultados são disponibilizados diretamente para a saída da arquitetura.

Como forma de otimização desta arquitetura, pesquisadores desenvolvem diferentes maneiras de armazenar e gerar os coeficientes em arquiteturas específicas. O esquema

proposto em [19] mostra que a quantidade de coeficientes necessários para computar uma FFT de N-pontos é igual a $N/8 + 1$, bastando utilizar um circuito simples para troca de dados entre real e imaginário e blocos que implementam o complemento de 2.

Nas arquiteturas propostas em [20] [21] [22], são apresentadas otimizações no acesso dos dados nas memórias, procurando diminuir o conflito e acelerar a disponibilização dos mesmos para reprocessamento ou nas saídas. Também são propostas alternativas para diminuir o tamanho das memórias.

Outra forma de otimização no uso de memórias é a proposta por arquiteturas que utilizam o algoritmo CORDIC para gerar os coeficientes, como em [23][24]. Esse algoritmo implementa através de somas e deslocamentos binários os valores do cosseno e do seno do ângulo do coeficiente, que irão corresponder, respectivamente a parte real e imaginária, utilizadas para multiplicação nos multiplicadores complexos. Entretanto, ainda são necessárias memórias e sistemas de indexação para leitura e escrita dos resultados intermediários.

2.3.4 Arquiteturas de Estágio Único

Uma arquitetura de estágio único foi inicialmente proposta em [25] no intuito de atingir velocidade de processamento necessária para sistemas de comunicação Wireless utilizando OFDM. Após essa primeira idealização, o trabalho proposto em [26] apresenta novamente um método para implementação deste modelo, embora, sem ainda sintetizá-lo, apenas demonstrando que otimizações no desempenho e potência do chip podem ser obtidas.

Essas arquiteturas, de forma geral, possuem a estrutura definida na Figura 10. Para este trabalho será considerado os sistemas de conversão serial/paralelo e paralelo/serial apenas quando se tratando da fabricação do chip de 8 pontos, nas demais implementação essas etapas são desconsideradas, assumindo que os dados já estão na forma paralela, disponíveis na entrada das arquiteturas.

Deste modo, os dados são disponibilizados na entrada da FFT, que inicialmente irá processar o primeiro estágio e em seguida realimentar o resultado, para a computação do estágio seguinte. Esse processo se repete até o último estágio, que como resultado disponibiliza os dados na saída da FFT.

As FFTs de estágio único tem como característica a utilização de N/r borboletas, onde r é o *radix* adotado. Além disso, a velocidade de processamento da FFT está ligada com a quantidade de pontos da mesma, de forma que, a quantidade de ciclos para processamento total é igual a quantidade de estágios s definida por $\log_r(N)$.

Este modelo apresenta um *tradeoff* intermediário entre área e desempenho. É necessário uma quantidade $\frac{N}{r}$ maior de borboletas comparadas com as baseadas em

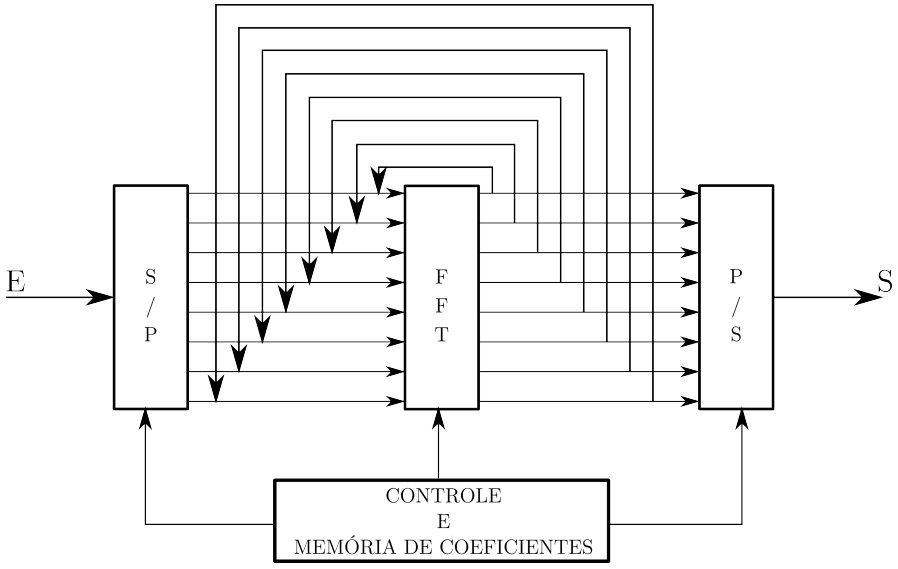


Figura 10 – Fluxo de dados genérico de uma FFT de estágio único para 8 pontos.

Fonte: Adaptado de [25].

memória e $\frac{N}{rlog_r(N)}$ nos modelos *pipelined*. Contudo, o desempenho é $\frac{N}{r}$ vezes maior do que o baseado em memória e $\frac{N}{rlog_r(N)}$ que o *pipelined*, considerando a mesma frequência de operação dos circuitos integrados.

Comparando a quantidade de memória utilizada, para a arquitetura de estágio único, utiliza-se N registradores para efetuar a realimentação. Na arquitetura baseada em memória utilizam-se $2N$ palavras e na arquitetura *pipelined*$\log_r(N)$ blocos de $\frac{N}{2^s}$ palavras.

Este trabalho irá utilizar esse modelo de implementação da FFT, buscando obter resultados de PPA através da síntese lógica dos processadores, além de aplicar uma técnica para otimização dos circuitos multiplicadores e também de diminuição das multiplicações implementadas.

3 Multiplicação de Constantes Múltiplas

Neste capítulo será abordado o problema de multiplicação de constantes múltiplas (MCM). Para isso, será definido o problema e sua presença em sistemas DSP, discutido as maneiras utilizadas para encontrar sua solução, bem como suas aplicações para a implementação de arquiteturas da FFT em hardware.

3.1 Definição do Problema MCM

Os problemas MCM estão presentes em diversas estruturas utilizadas em sistemas DSP, como: filtros de resposta finita ao impulso (FIR) e FFTs. De forma simplificada, o problema MCM pode ser definido na forma da Eq. (3.1). Onde Y corresponde à matriz de dados de saída do sistema, X às entradas do sistema e M corresponde à matriz de constantes (coeficientes).

$$Y = M.X \tag{3.1}$$

Esse problema consiste na busca de implementar um conjunto de multiplicações sucessivas por constantes, utilizando a menor quantidade de somas e deslocamento binários. Como definido em [27], esse tipo de problema é considerado NP-completo. Assim, pode-se utilizar versões restritas do problema para encontrar soluções de forma eficiente ou desenvolver heurísticas que buscam resultados sub-ótimos.

Analisando-se a Eq. (2.3) que apresenta a DFT em forma matricial, verifica-se que a mesma pode ser modelada como um problema MCM, bem como a sua implementação através da FFT, quando reescrevendo as equações (2.10) e (2.11).

Como descrito anteriormente, a FFT é uma operação que requer uma quantidade elevada de multiplicações: blocos que afetam consideravelmente os resultados de PPA nos circuitos integrados. A possibilidade de modelar a FFT como um problema MCM tem como benefício eliminar esses blocos e substituí-los por somadores/subtratores e deslocamentos binários.

3.2 Particularidades da Solução

O problema MCM, embora definido de forma simples, apresenta uma série de particularidades, podendo ser divididas em dois tópicos: representação dos coeficientes e compartilhamento de subexpressões.

3.2.1 Representação dos Coeficientes

Geralmente a representação dos coeficientes se dá de três formas: binária, dígito canônico sinalizado (CSD) e dígito mínimo sinalizado (MSD).

Na representação binária os números são representados pelo conjunto $\{0, 1\}$, tendo como propriedade que cada número possui uma única representação.

As formas CSD e MSD tem como conjunto de representação $\{1, 0, \overline{1}\}$, onde $\overline{1}$ representa -1. Embora possuam conjuntos iguais, ambas apresentam propriedades exclusivas, abordadas em [28]. A representação CSD tem como propriedades: a) um número possui uma única representação; e b) a multiplicação entre dígitos adjacentes é sempre 0. Para MSD, descarta-se a segunda propriedade, ou seja, um número pode possuir dois dígitos adjacentes não-zeros, o que possibilita uma gama de representações para um mesmo número decimal.

Na Tabela 3 são apresentados números decimais e suas respectivas representações binária, CSD e MSD.

Tabela 3 – Exemplos de números decimais representados na forma binária, CSD e MSD.

Decimal	**Binário**	**CSD**	**MSD**
0	0000	0000	0000
1	0001	0001	0001, $01\overline{11}$
5	0101	0101	0101, $10\overline{11}$
7	0111	$100\overline{1}$	0111, $100\overline{1}$

3.2.2 Compartilhamento de Subexpressões

A implementação direta dos coeficientes na forma de somas e deslocamentos pode ser aprimorada utilizando técnicas de eliminação de subexpressões comuns (CSE). Essas metodologias buscam encontrar padrões nos códigos que representam os coeficientes e reutilizá-los, eliminando redundâncias.

No trabalho apresentado em [29], um algoritmo iterativo é proposto para identificar padrões no conjunto de coeficientes que compõem o problema MCM. Essa proposta procura implementar coeficientes intermediários que são reutilizados com mais frequência, reduzindo assim o número total de adições necessárias.

Outra forma para otimizar o compartilhamento de subexpressões é utilizando um algoritmo exaustivo. Em [30], uma proposta é apresentada para otimização de filtros FIR, onde o ponto-chave do algoritmo está em escolher padrões que possuam a menor

quantidade de bits. Essa escolha resulta em somadores menores que, por consequência, facilitam implementações em silício.

A proposta apresentada em [31], modela o problema CSE através de programação linear inteira, utilizando representação MSD para os coeficientes. Esta técnica se mostra mais otimizada que o modelo exaustivo proposto anteriormente por [30].

3.3 Algoritmo de Aksoy *et al.*

Este trabalho utiliza o algoritmo proposto por Aksoy *et al.* em [32] e utilizado por Ghissoni para implementação e otimização de FFTs nos trabalhos [33][34][35] e [11]. O algoritmo encontra uma solução exata para o problema MMC através de uma busca em amplitude e profundidade, denominada BFS. Ao mesmo tempo em que o algoritmo procura soluções em diferentes iterações (profundidade), aumenta-se a quantidade de conjuntos analisados para encontrar soluções (amplitude). A característica desse algoritmo é ser independente do tipo de representação das constantes.

Inicialmente, o conjunto de coeficientes que compõem o problema MCM é denominado por T e, a quantidade de coeficientes por $|T|$. Outra definição são as matrizes W_R e W_T. As matrizes W_R consistem de valores de coeficientes ou constantes intermediárias que já foram implementadas e que estão prontas para serem utilizadas para implementar outros coeficientes. Já as matrizes W_T, correspondem aos coeficientes que ainda não foram implementados, e que necessitam de outras constantes intermediárias não pertencentes aos conjuntos W_R.

Antes de iniciar o algoritmo, existe uma fase de pré-processamento, que tem como função tornar todos os coeficientes positivos e únicos dentro do conjunto T. Além disso, é determinado o número maior de bits utilizado para representar os coeficientes, denominado bw. Após o pré-processamento, existem duas funções principais que compõem o algoritmo: *BFSearch* e *Synthesize*.

3.3.1 Função *Synthesize*

A função *Synthesize*, descrita em pseudocódigo no Listing 3.1, tem como dados de entradas duas matrizes, uma do tipo W_R e outra do tipo W_T. Consiste de um loop sobre todos os elementos de W_T, buscando sintetizá-los utilizando os elementos do conjunto W_R, até que todos sejam sintetizados ou nenhum elemento de W_T possa ser sintetizado com os elementos de W_R.

Ao final, essa função retorna outras duas matrizes, uma do tipo W_R contendo os elementos sintetizados e outra do tipo W_T que, ou possui os elementos que ainda não foram sintetizados, ou pode estar vazia, significando que todos os elementos foram sintetizados.

```
1   Synthesize(R, T)
    repeat
3     adicionado = 0
      for k = 1 to |T| do
5       if tk pode ser sintetizado com os elementos de R então
          adicionado = 1
7         R ← R ∪ {tk}
          T ← T \ {tk}
9   until adicionado = 0
    return (R, T)
11
```

Listing 3.1 – Pseudocódigo da função *Synthesize*. Fonte: Adaptado de [32].

3.3.2 Função *BFSearch*

A função *BFSearch*, descrita em pseudocódigo no Listing 3.2, é a principal do algoritmo, tendo como dados de entrada o conjunto de coeficientes T e o número de bits máximo bw. O primeiro passo dessa função é definir um conjunto de soluções denominado R, onde é atribuído o número 1 como constituinte. Em seguida, é chamada a função *Synthesize* que irá tentar implementar os coeficientes T com o conjunto R. Caso a resposta retorne um conjunto T nulo, todos os coeficiente foram implementados, caso contrário, há a necessidade de encontrar coeficientes intermediários para implementar o restante dos coeficientes.

Para encontrar esses coeficientes intermediários, é implementado um loop infinito de forma que o algoritmo só encerra seu processo quando todos os coeficientes são implementados. Dentro desse loop, são definidas novas matrizes X, que são inicializadas a cada iteração com os valores de W. Além disso, são inseridos dois *for* loops aninhados.

O primeiro *for* itera em i de 1 até m, onde m corresponde à n que é a dimensão das matrizes W tratadas como pares. O segundo *for* itera em j de 1 até $2^{bw+1} - 1$ com passo 2. Se o valor de j pertence aos conjuntos X_{R_i} e X_{T_i}, deve-se iterar j novamente, senão, tenta implementá-lo utilizando a função *Synthesize* tendo como argumentos X_{R_i} e j, e saída as matrizes temporárias A e B, para coeficientes prontos e não implementados, respectivamente. Caso B não seja nulo itera novamente j, caso contrário significa que j é implementado com os coeficientes previamente determinados e desta forma é adicionado ao conjunto X_{R_i}. Em seguida, incrementa em 1 a dimensão n e também se computam os valores de X_{R_n} e X_{T_n} na função *Synthesize* com os valores de entrada X_{R_i} e X_{T_i}.

Se a matriz X_{t_n} for nula o algoritmo chega ao fim, pois não restaram coeficientes para implementar. Caso contrário, atualizam-se os valores de m e X repetindo o processo.

```
BFSearch(T, bw)
  R ← {1}
  (R, T) = Synthesize(R, T)
  if T = Ø then
    return R
  else
    n = 1, W_R1 ← R, W_T1 ← T
    while 1 do
      m = n, X_R = W_R, X_T = W_T
      n = 0, W_R = W_T = [ ]
      for i = 1 até m do
        for j = 1 até 2^(bw+1)−1 passo 2 do
          if j ∉ X_Ri and j ∉ X_Ti then
            (A, B) = Synthesize(X_Ri, {j})
            if B = Ø then
              X_Ri ← X_Ri ∪ {j}
              n = n + 1
              (W_Rn , W_Tn ) = Synthesize(X_Ri, X_Ti )
              if W_Tn = Ø then
                return W_Rn
```

Listing 3.2 – Pseudocódigo da função *BFSearch*. Fonte: Adaptado de [32].

3.3.3 Apresentação da Solução

Após encontrada a solução MCM, isto é, os coeficientes únicos que precisam ser implementados para gerar as multiplicações, ocorre o processo de impressão da solução na forma de equações, que é realizado sempre que um coeficiente é implementado, utilizando os coeficientes necessários para sua composição. Essa impressão é realizada dentro do algoritmo que busca a solução de forma exaustiva, descrito anteriormente.

As equações são dividas em três tipos: subexpressões, expressões de saídas e expressões primárias. Definem-se subexpressões (S_i) as equações que resultam em um coeficiente intermediário. As expressões de saídas (O_j), podem resultar diretamente em um coeficiente ou então em um valor que necessite de deslocamentos binários para representar o coeficiente. Já as expressões primárias (P_k) correspondem aos coeficientes e são geradas a partir do recebimento direto de uma expressão de saída ou da mesma com os respectivos deslocamentos.

Como exemplo considere o problema MCM definido na Eq. (3.2). A Figura 11 apresenta-o na forma gráfica, onde em a) é apresentado o problema, utilizando multiplicadores; em b) é apresentada uma solução não otimizada e em c) o resultado do algoritmo descrito e utilizado no trabalho. Repare que, no último ocorre a otimização da quantidade de somadores necessários. O código gerado pela ferramenta na forma de equações é

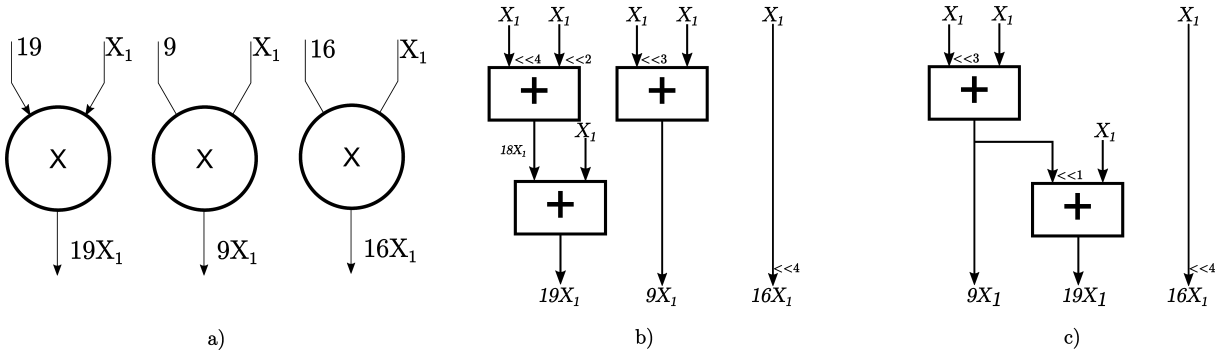


Figura 11 – Exemplo de problema e soluções MCM. Em a) apresentação do problema; b) solução não otimizada; e c) solução apresentada pelo algoritmo de *Aksoy et al.*

apresentado na Eq. (3.3).

$$Y = \begin{bmatrix} 9 \\ 19 \\ 16 \end{bmatrix} X_1 \tag{3.2}$$

$$\begin{gathered} -- \textit{Expressões Primárias} -- \\ P_1 : (+09 * X_1) = +(+09 * X_1) << 0 \\ P_2 : (+19 * X_1) = +(+19 * X_1) << 0 \\ P_3 : (+16 * X_1) = +(+01 * X_1) << 4 \\ -- \textit{Subexpressões e Expressões de Saídas} -- \\ O_1 : (+09 * X_1) = +X_1 << 0 + X_1 << 3 \\ O_2 : (+19 * X_1) = +O_1 << 1 + X_1 << 0 \end{gathered} \tag{3.3}$$

4 Metodologia

Neste capítulo serão apresentados os passos para elaboração das arquiteturas. Será abordada a arquitetura da FFT de estágio único DIT *radix-2*, dividida em duas organizações principais: borboletas e FFT de estágio único. Baseado nessas organizações, determina-se a quantidade de bits utilizadas para representar os dados nas arquiteturas, em função da realimentação e consequente truncamento entre estágios.

Após definida a arquitetura, são computados os coeficientes da FFT. Posteriormente, os mesmos são utilizados na busca da solução MCM, pelo algoritmo de Aksoy *et. al*, fornecido por Ghissoni para utilização neste trabalho. A resposta do algoritmo é então processada por um outro algoritmo, desenvolvido neste trabalho, que converte as equações da solução para a linguagem de hardware SystemVerilog.

Detalhes como o indexamento dos dados nas etapas de realimentação da arquitetura também serão discutidos. Por fim, serão detalhadas as etapas de verificação e síntese lógica das arquiteturas na tecnologia XFAB 0.18μm.

Os algoritmos aqui utilizados são descritos em duas linguagens. Para o desenvolvimento da arquitetura e busca da solução MCM, utiliza-se a linguagem de programação Matlab. Para as etapas de verificação funcional e síntese lógica das arquiteturas são utilizados códigos descritos em TCL.

4.1 Organização da Borboleta

A borboleta, genericamente definida na Seção 2.3.1, é a estrutura na FFT DIT que realiza uma multiplicação complexa, seguida de duas somas complexas. Após a decomposição dessas operações complexas em apenas reais, encontra-se a utilização de quatro multiplicações e seis somas.

Na Figura 12, é definida, de forma mais específica, a organização da borboleta de uma arquitetura de 16 pontos. Existem as entradas A e B, dividas em componentes reais: A_{re} e B_{re}; e em imaginárias: A_{im} e B_{im}.

A entrada B contém os dados que serão multiplicados pelos coeficientes, logo, sua conexão se da diretamente aos blocos MCM. Como saídas dos blocos MCM, tem-se o resultado das multiplicações pelos coeficientes. Já na saída do bloco, existem multiplexadores, que são responsáveis por selecionar as saídas corretas em função do estágio a ser computado.

Após a escolha das saídas nos blocos multiplexadores, as mesmas passam por um

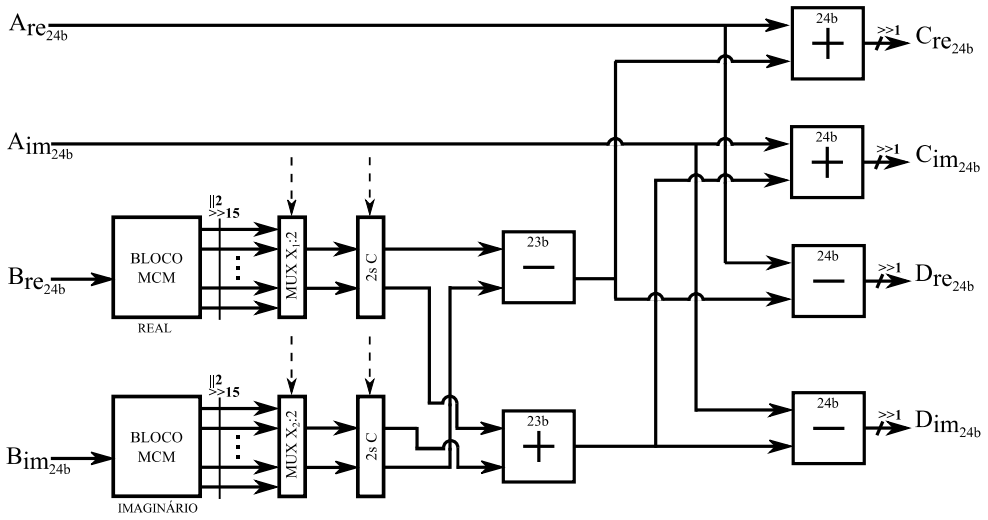


Figura 12 – Organização da borboleta de uma FFT DIT *radix-2* com N=16 pontos e Q=16 bits.

bloco de complemento de 2, que realiza a troca de sinal, quando necessário, de acordo com o valor original do coeficiente. Em seguida, os dados são subtraídos/somados gerando os resultados da multiplicação complexa.

Por fim, são implementadas as últimas quatro somas/subtrações, que finalizam a computação da borboleta. Esses dados serão, então, realimentados na arquitetura da FFT ou disponibilizados à saída, quando no último estágio.

4.1.1 Análise de Bits para Representação de Entradas e Saídas

Considere que os dados de entradas e coeficientes estão quantizados utilizando Q bits, ou seja, os valores de entrada e coeficientes, compreendidos de $1 \geq x \geq -1$ são multiplicados por 2^{Q-1}. Tomando como referência o fluxo de dados apresentado na organização da borboleta da Figura 12, pode-se deduzir a quantidade de bits final utilizada pelos dados de saída.

No primeiro estágio, os dados de entrada possuem Q bits e B é multiplicado pelo coeficiente que também possui Q bits. O resultado é um valor de $2Q$ bits, contudo, para manter a magnitude do dado e, considerando que a magnitude dos coeficientes é sempre 1, deve-se truncar $Q-1$ bits, resultando em $Q+1$ bits. Continuando o fluxo de dados, esse resultado passa por mais duas somas/subtrações, que adicionam mais 2 bits. Ao final, o resultado possui $Q+3$ bits.

Para diminuir a quantidade de bits utilizados entre estágios, optou-se ainda por realizar um último truncamento, descartando o bit menos significativo na saída de cada somador/subtrator. Assim, tem-se que em cada estágio ocorre a adição de 2 bits.

De forma geral, para esse tipo de truncamento, pode-se definir o número máximo de bits necessários para cada entrada e saída, como na Eq. (4.1):

$$IO_b = 2.\log_2(N) + Q \tag{4.1}$$

Determinado o número total de bits, IO_b, pode-se projetar todos os somadores subsequentes da arquitetura MCM e das borboletas para comportarem essa quantidade. Ressalta-se aqui que a borboleta precisa ser definida para esse número de bits correspondente ao último estágio, pois não há a possibilidade de alterar o tamanho dos blocos à medida que um estágio próximo seja computado.

Aplicado ao exemplo de N=16 pontos, as entradas A e B precisarão de 24 bits ao final. Portanto, os somadores da multiplicação complexa terão tamanho igual a 23 bits ($IO_b - 1$) e, os somadores finais da borboleta terão tamanho 24 bits (IO_b). Dessa maneira, evitam-se casos de *overflow* na arquitetura.

4.2 Organização da FFT

Após definidas as borboletas, as mesmas são instanciadas na arquitetura da FFT. Além das borboletas, essa arquitetura é composta dos seguintes blocos: multiplexadores de entrada e saída, multiplexadores de mapeamento de índices e um circuito de controle. O fluxo de dados dessa composição pode ser visualizado na Figura 13, ainda com exemplo para N=16 pontos.

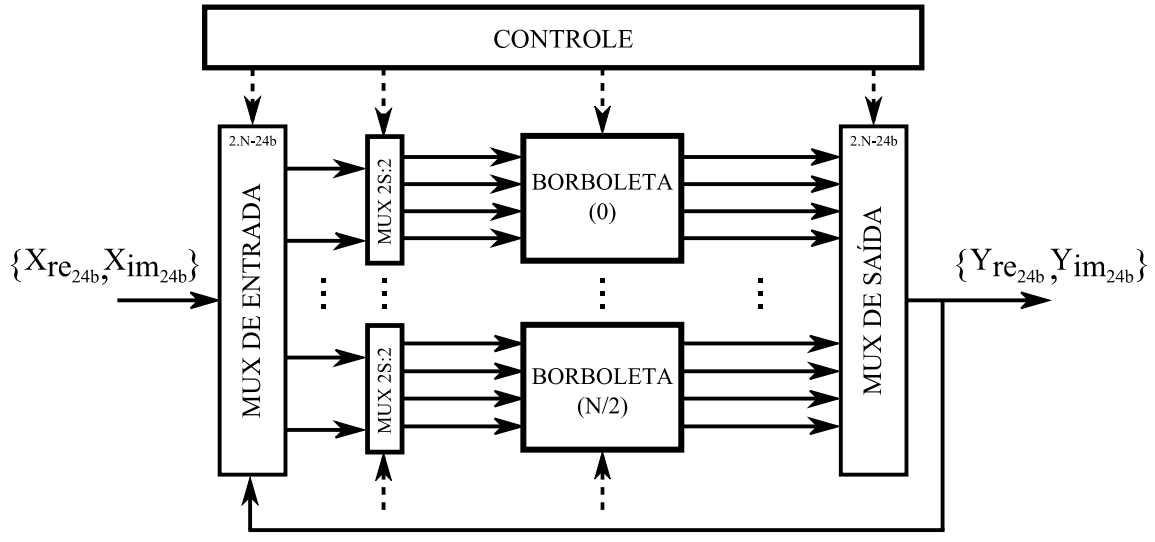


Figura 13 – Organização da FFT de estágio único para N = 16 pontos e Q = 16 bits.

4.2.1 Multiplexadores de Entrada e Saída

O multiplexador de entrada tem por finalidade escolher entre os dados de entrada X_re e X_im, quando no estágio inicial, ou então selecionar os dados de realimentação em estágios subsequentes.

A seleção dos dados para realimentação ou disponibilização na saída é efetuada pelo multiplexador de saída. Quando no último estágio, além de conectar os dados à saída, é necessário realizar um último indexamento nas saídas, que consiste em separar os dados em dois grupos: pares e ímpares.

Os resultados pares são disponibilizados nas saídas 0 à $N/2 - 1$ e os resultados ímpares nas saídas $N/2$ à $N - 1$. Esse último indexamento se deve em função da maneira que os dados são indexados durante a realimentação entre estágios.

4.2.2 Multiplexadores de Mapeamento de Índices

Esses multiplexadores, localizados após o multiplexador de entrada, tem por finalidade alimentar as borboletas com as entradas corretas em relação ao estágio. Para mapear esse comportamento, foi desenvolvido um algoritmo que gera os índices na entrada de cada borboleta em cada estágio. Esse algoritmo, com a função *index_mapping* pode ser apreciado no Listing 4.1. Como dado de entrada é disponibilizado o número de pontos da FFT. Com esse valor, são computados a quantidade de estágios s e também reservado uma matriz $N \times s$ para armazenar os índices.

Para a primeira etapa, como definido anteriormente, é utilizado a representação bit-reverse. Assim, na linha 4 são atribuídos diretamente o ordenamento através da função *bitrevorder*, disponível no Matlab. Em seguida, inicia-se o *for loop* para determinar os índices dos estágios restantes. Pode-se dividir o algoritmo em três etapas: par, ímpar e replicação.

A etapa par consiste em designar os índices pares do ordenamento e é descrita nas linhas 7-13. Inicia-se o processo com o valor de $par = 0$. Em cada *loop* são definidos os índices das posições c e $c + 1$ do estágio k, que correspondem a par e $par + 2^{(k-1)}$, respectivamente. Ao final, os valores de c e par são incrementados em 2. Quando $i = 2^{(k-1)}$, termina-se esta etapa.

Nas linhas 14-20 é definida a etapa ímpar. É inicializada a variável $impar = 1$ e c possui o valor final da etapa anterior. Novamente, a cada iteração em i são determinados índices para as posições c e $c + 1$, correspondentes à $impar$ e $impar + 2^{(k-1)}$. Os índices, antes de efetuar a etapa de replicação, podem ser observados na Tabela 4 a).

Para finalizar a etapa de replicação, nas linhas 21-23, são replicados os índices acrescendo um fator de conjunto igual a 2^k. Assim, o resultado final para uma FFT de

Tabela 4 – Índices em cada linha: a) antes de efetuar a replicação e b) após replicação.

N=16					
Estágio		**1**	**2**	**3**	**4**
Linha	**1**	0	0	0	0
		8	2	4	8
	2	4	1	2	2
		12	3	6	10
	3	2	0	1	4
		10	0	5	12
	4	6	0	3	6
		14	0	7	14
	5	1	0	0	1
		9	0	0	9
	6	5	0	0	3
		13	0	0	11
	7	3	0	0	5
		11	0	0	13
	8	7	0	0	7
		15	0	0	15

a)

N=16					
Estágio		**1**	**2**	**3**	**4**
Linha	**1**	0	0	0	0
		8	2	4	8
	2	4	1	2	2
		12	3	6	10
	3	2	4	1	4
		10	6	5	12
	4	6	5	3	6
		14	7	7	14
	5	1	8	8	1
		9	10	12	9
	6	5	9	10	3
		13	11	14	11
	7	3	12	9	5
		11	14	13	13
	8	7	13	11	7
		15	15	15	15

b)

N=16 pontos pode ser observado na Tabela 4 b).

Com os índices de cada linha determinados, é utilizado um algoritmo para descrever esses blocos em SystemVerilog, alternando as entradas dos mesmos em relação aos índices.

4.2.3 Circuito de Controle

O circuito de controle é elaborado utilizando uma máquina de estados (FSM). Essa FSM possui $s + 1$ estados, que correspondem a: s estágios da FFT mais um IDLE. Esses estados podem ser visualizados de forma gráfica na Figura 14.

Durante o estado IDLE, a arquitetura fica inativa, aguardando que um sinal de início seja enviado para a FFT iniciar sua computação, como em casos onde a transmissão dos dados se dá via serial. Quando todos os dados de entrada são recebidos, esse sinal de início é ativado e a arquitetura começa a computar os estágios da FFT.

A troca de estados após iniciado o primeiro estágio se dá durante a borda de subida do sinal de relógio (clk). Dentro de cada um dos estágios existem variáveis de controle que são assinaladas, são elas: sel_mcm, sel_mux_in, sel_mux_out, sel_cmp_re, sel_cmp_im e *done*. A funcionalidade de cada um desses sinais e seus respectivos tamanhos são descritos na Tabela 5.

```
function [ordem]=index_mapping(N)
s=log2(N);
ordem=zeros(N,s);
ordem(:,1)=bitrevorder([0:1:N-1]);
for k=1:s
  c=1;
  par=0;
  for i=1:2:2^(k-1)
    ordem(c,k)=par;
    ordem(c+1,k)=par+2^(k-1);
    c=c+2;
    par=par+2;
  end
  impar=1;
  for i=1:2:2^(k-1)
    ordem(c,k)=impar;
    ordem(c+1,k)=impar+2^(k-1);
    impar=impar+2;
    c=c+2;
  end
  for i=1:N/(2^(k))-1
    ordem(i*2^(k)+1:(i+1)*2^(k),k)=ordem((i-1)*2^(k)+1:i*2^(k),k)+2^(k);
  end
end
```

Listing 4.1 – Código para determinar os índices de mapeamento. Linguagem: Matlab.

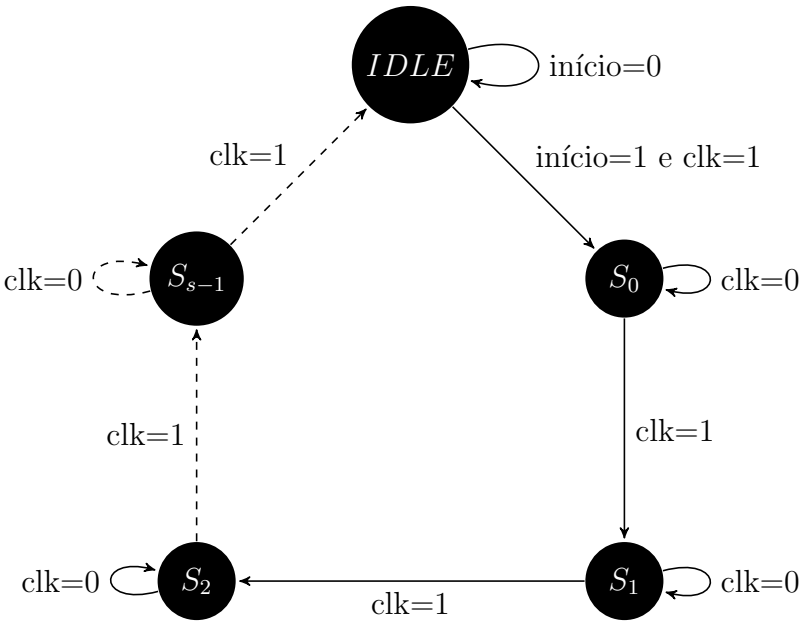


Figura 14 – Máquina de estados para o circuito de controle da FFT.

Tabela 5 – Funcionalidade dos sinais de controle na FSM.

Sinal	Funcionalidade	Tamanho
sel_mux_in	Escolhe entre sinal de entrada ou realimentação.	1 bit
sel_mux_out	disponibiliza dados para realimentação ou para a saída.	1 bit
sel_mcm	Seleciona a saída correta nos blocos MCMs das borboletas.	Inteiro maior ou igual que $\log_2(s)$
sel_cmp_re	Designa a realização ou não do complemento de dois na parte real do coeficiente.	N/2 bits
sel_cmp_im	Designa a realização ou não do complemento de dois na parte imaginária do coeficiente.	N/2 bits
done	Nível lógico alto quando o resultado está pronto.	1 bit

Além destas variáveis, a FSM é sensível ao sinal de reset (reset_n), que quando em nível lógico baixo retorna a arquitetura para o estado IDLE e, quando em nível lógico alto possibilita a computação da FFT.

4.3 Determinando os Coeficientes

Os valores dos coeficientes são definidos baseados no algoritmo de Cooley e Tukey e na maneira em que os mesmos são dispostos nos estágios da arquitetura. A disposição dos coeficientes interfere, posteriormente, no indexamento das entradas durante a realimentação dos estágios.

Considerando a Equação (4.2), R é denominado como o coeficiente rotacional, utilizado para determinar o valor dos coeficientes. O valor correspondente rotacional é computado de acordo com o o estágio da FFT e com a linha. No Listing 4.2 é demonstrado o algoritmo utilizado para computar cada um dos coeficientes rotacionais.

Inicialmente, computa-se o número de estágios s como sendo $\log_2(N)$. Após isso, utilizando dois *loops* for é determinado o valor do ângulo rotacional na matriz tw_{angle}. O valor do rotacional é incrementado em 2^{s-k} para cada valor de j e, toda vez que o contador *count* possui valor igual a 2^{k-1}, esse valor é resetado para 0. Os resultados de R para $N = 4$, $N = 8$ e $N = 16$ podem ser observados na Tabela 6.

$$e^{-i2\pi.R/N} \tag{4.2}$$

Ao final, com os coeficientes rotacionais computados, são determinados os valores dos coeficientes W_N^R, como definido na linha 16 do código. Os valores aqui obtidos são separados em parte real e imaginária e agrupados novamente em uma única matriz de valores reais.

```
function [tw_values]=tw_2(N)
s=log2(N);
for k =2:1:s
  value = 0;
  count=0;
  for j=1:N/2
    tw_angle(j,k)=value;
    value=value+2^(s-k);
    count=count+1;
    if count==2^(k-1)
      value=0;
      count=0;
    end
  end
end
tw_values=exp(-1i*2*pi.*tw_angle/N);
end

```

Listing 4.2 – Código para determinação dos coeficientes. Linguagem: Matlab.

Tabela 6 – Valores dos coeficientes rotacionais em relação aos estágios para N=4, 8 e 16.

		N=4		**N=8**			**N=16**			
Estágio		**1**	**2**	**1**	**2**	**3**	**1**	**2**	**3**	**4**
Linha	**1**	0	0	0	0	0	0	0	0	0
	2	0	1	0	2	1	0	4	2	1
	3			0	0	2	0	0	4	2
	4			0	2	3	0	4	6	3
	5						0	0	0	4
	6						0	4	2	5
	7						0	0	4	6
	8						0	4	6	7

Ainda representados em ponto flutuante, esses valores precisam ser preprocessados para que apenas os valores únicos, positivos e inteiros de cada linha da matriz sejam utilizados no algoritmo de solução MCM.

Esse processamento é efetuado quantizando os valores para 2^{Q-1}, onde Q representa o número de bits de quantização. Em função da representação dos dados em complemento de 2, utilizada neste trabalho, apresentar valores apenas de $+2^{Q-1} - 1 \geq x \geq -2^{Q-1}$, os valores de $\pm 2^{Q-1}$ são substituídos por $\pm 2^{Q-1} \mp 1$. Essa substituição insere um erro de $\frac{1}{2^{Q-1}} * 100\%$ na representação do coeficiente, contudo, evita-se a utilização de hardware extra para computação de coeficientes obtidos com rotacionais 0 e $N/4$, presentes na mesma linha, condição esta que ocorre em $N/2 - 1$ linhas da tabela de rotacionais.

```
for j=1:2:N/2
C_u = unique(coeff(j,:));
[C_file]=Convert_to_file(C_u);
[sys, prim_expr, out_zero] = MCM_solve(C_file);
[sub,adders]=MCM_to_SV(sys, prim_expr, out_zero);
[sub_u,adders_u]=Sub_Adders_Unique(sub, adders, sub_u, adders_u);
end
```

Listing 4.3 – Código para busca e conversão da solução MCM. Linguagem: Matlab.

4.4 Busca da Solução MCM

Após processados e determinados os coeficientes quantizados, incia-se a busca por uma solução MCM, ou seja, as equações que implementam as multiplicações únicas em cada linha da FFT.

No Listing 4.3 é demonstrado as etapas para a busca da solução MCM. Primeiro os coeficientes da linha são feitos únicos. Então, é gerado um arquivo de texto, que contém como dados o número de linhas (.r) da FFT. Esse valor é baseado na quantidade de valores únicos na linha da FFT. Em seguida, é inserida a quantidade de colunas (.c), que neste trabalho será constante igual a 1. Após, são impressos os coeficientes únicos. Por fim, encerra-se esse arquivo com um indicador (.e).

Os arquivos com os dados das linhas da FFT são enviados ao algoritmo MCM, que os processa um de cada vez, gerando a solução MCM correspondente. Essa solução, originalmente gerada na forma textual, agora é descrita por três tipos de matrizes: sistema de equações (sys), expressões primárias (prim_expr) e saídas zero (out_zero). Com essas três matrizes desenvolve-se um algoritmo para interpretá-las e convertê-las em respectivos arquivos de descrição em linguagem de hardware.

O resultado dessa conversão é um arquivo .sv, da solução MCM e também matrizes com tamanhos dos somadores/subtratores utilizados na arquitetura. Esses valores são armazenados e também unificados, para que após terminado o processamento de todas as soluções, arquiteturas únicas de somadores e subtratores sejam geradas, para posterior instanciamento.

Ao final, obtêm-se arquiteturas no formato representado pelo Listing 4.4. Composto pela entrada $X1$ e saídas correspondentes aos resultados das multiplicações, Out_n. Para este exemplo, fora utilizado apenas um instanciamento de subtrator de 22 bits.

4.4.1 Função *Convert_to_file*

A função *Convert_to_file* tem como dados de entrada os valores únicos de coeficientes de uma linha da FFT. A primeira etapa dessa função identifica a quantidade de linhas (.r)

```
module mcm_8_0 (
  input  logic     [13:0] X1,
  output logic     [12:0] Out_1,
  output logic     [12:0] Out_2);

  logic [21:0]O1;

  sub_22b     OUT1  ({X1[13], X1,7'b0000000}, {X1[13], X1[13], X1[13], X1
  [13], X1[13], X1[13], X1[13], X1[13], X1}, O1);

  assign Out_1 = {O1[19:7]};
  assign Out_2 = 13'b0;
endmodule

```

Listing 4.4 – Resultado de uma arquitetura MCM após ser processada. Linguagem: SystemVerilog.

```
.r 2
.c 1
0
127
.e

```

Listing 4.5 – Exemplo de arquivo de texto contendo os dados de linha, coluna e coeficientes. Linguagem: Texto.

e a escreve na primeira linha do arquivo texto.

Em seguida, é setado o valor de colunas, especificado em 1 (.c). Apos isso, são impressos os coeficientes em si, um em cada linha. Por último, é inserido um identificador de fim de arquivo (.e).

O resultado, por exemplo, para a primeira linha de uma FFT de 8 pontos quantizados em Q=8, pode ser observado no Listing 4.5, que neste caso, é composto pelos coeficientes 0 e 127.

Para continuar utilizando o formato de leitura dos coeficientes dentro da função que encontra a solução MCM, manteve-se esse sistema de escrita em arquivo de texto. Embora, por questões de otimização de algoritmo, os mesmos pudessem serem lidos diretamente de uma matriz no próprio software Matlab. O que por hora, não é o foco do trabalho.

4.4.2 Função *MCM_solve*

Com o arquivo de texto gerado para a linha da FFT, é possível determinar a solução MCM, utilizando o algoritmo descrito na Seção 3.3. Entretanto, são elaboradas algumas alterações quanto ao formato dos dados de saída. Essas alterações buscam facilitar a transcrição das equações resultantes para um posterior código em linguagem de hardware.

```
[sys, prim_expr, out_zero] = MCM_solve(C_file)
  r=linhas(C_file);
  c=colunas(C_file);
  out_zero  = zeros(r,c);
  prim_expr = zeros(r,c);
  for i=1:c
    for j= 1:r
      C=get_coeff(C_file);
      if C(i,j) == 0
        out_zero(j) = 1;
      else
        prim_expr(j)=power_2(C(i,j));
    end
  end
  [sys]=BFSearch[C,Q];
end

```

Listing 4.6 – Código simplificado para determinar as soluções MCM. Linguagem: Matlab.

Tabela 7 – Relação entre equações da solução MCM e os valores da matriz *sys*.

Primeira ou Segunda Entrada				**Saída**	
Sinal	**Tipo**	**Índice**	**Deslocamento**	**Tipo**	**Índice**
0 – (+)	0 – X	0 – X_0, S_0 ou O_0	D - << D	0 – Subexpressão	0 – S_0 ou O_0
1 – (-)	1 – S	⋮		1 – Exp. Prim.	N – S_N, O_N
	2 – O	N – X_N, S_N, O_N			

De forma resumida, esse algoritmo pode ser acompanhado no Listing 4.6. Primeiro, são adquiridos os valores de linhas e colunas, que determinam a quantidade de coeficientes. Depois, esses valores são verificados se são uma potência de dois, gerando o que define-se como as expressões primárias (prim_expr). Caso seja identificada uma saída 0, o índice da mesma é armazenado na matriz de saídas zero (out_zero).

Posteriormente, é utilizada a função descrita na Seção 3.3.2, para determinar a solução MCM. Essa solução, antes representada na forma de equações, agora é mapeada dentro de uma matriz denomina *sys*, que possui as equações mapeadas de acordo com as características descritas na Tabela 7.

A Eq. (3.3) pode ser descrita no formato da matriz *sys*, que é apresentado na Tabela 8. Dessa maneira, é facilitada a manipulação da solução para a sua conversão em uma arquitetura em linguagem de hardware.

Tabela 8 – Resultado da transcrição para forma de matriz *sys* das expressões da Eq. 3.3.

	1ª Entrada				**2ª Entrada**				**Saída**	
Índices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	0	1	0	0	0	1	3	1	1
2	0	2	1	1	0	1	1	0	1	2

4.4.3 Função *MCM_to_SV*

A função *MCM_to_SV* tem como papel, converter os dados das matrizes *sys*, *prim_expr* e *out_zero*, em uma arquitetura MCM descrita em linguagem de hardware SystemVerilog. Como resposta, identificar os somadores e subtratores que são utilizados nessa arquitetura.

A primeira condição do algoritmo é se a matriz *sys* for nula, não possuir elementos, então é gerada uma arquitetura de saída única assinalada em zero.

Caso contrário, é efetuada a reorganização de *sys* de acordo com o tipo de saída: primeiro são ordenadas as subexpressões e em seguida, as expressões primárias, de acordo com o valor da coluna 9.

Em seguida, é analisada a quantidade de bits necessária para representar a saída de cada equação, escolhendo o valor máximo de bits entre as suas duas entradas. Esse processo é realizado através de uma varredura da matriz *sys*, obtendo-se o tamanho do primeiro e do segundo termo, de acordo com o tipo:

- **Tipo X:**

  tamanho($S|O_n$,1|2)= $IO_b + Deslocamento + 1$

- **Tipo S:**

  máximo(tamanho(S_n,1),tamanho(S_n,2)) + Deslocamento +1

- **Tipo O:**

  máximo(tamanho(O_n,1),tamanho(O_n,2)) + Deslocamento +1

O tamanho é sempre acrescido em 1 para evitar a ocorrência de *overflow*, visto que, não são implementadas *flags* para sinalizar essa condição.

Definido o tamanho de bits máximo de cada expressão, segue por definir o tipo de operação realizada: soma ou subtração. Isso é definido de forma simples, somando os dois sinais de cada entrada da subexpressão (colunas 1 e 5). Se o resultado for 0, corresponde a uma soma, se o resultado for 1, uma subtração.

Com os dados de quantidade de bits de cada expressão e do tipo de operação, é definido o tamanho de cada somador e subtrator utilizado em cada arquitetura MCM. Além disso, pode-se iniciar o processo de descrição da arquitetura, apresentado nos itens a seguir:

- **Cabeçalho**

  Descreve o nome do módulo, de acordo o índice da arquitetura MCM sendo implementada.

- **Declarar Sinais de Entrada e Saída**

  Define o tamanho dos sinais de entrada ($IO_b - 1$) e saída ($IO_b - 2$).

- **Declarar Sinais Intermediários**

  Define os sinais de saídas das subexpressões e expressões primárias de acordo com o número máximo de bits determinados anteriormente.

- **Instanciar Somadores/Subtratores**

  Nesta etapa, as expressões são representadas na forma de instanciamento, que envolve a utilização de um somador ou subtrator. O tamanho do bloco é definido de acordo com o número máximo de bits, e a ordem das entradas é definida de acordo com o sinal de cada entrada, excepcionalmente para subtratores. Se o sinal negativo está na primeira entrada, ocorre a inversão na ordem das mesmas.

- **Assinalar as Saídas**

  Durante esta etapa, as saídas são assinaladas. Os truncamentos, que mantém a magnitude dos sinais após a multiplicação pelos coeficientes, são efetuados e os deslocamentos primários inseridos.

Ao fim destas etapas, a arquitetura está completa, conforme apresentado anteriormente, no Listing 4.3. Ao total, são geradas N/2 arquiteturas MCM, cada uma delas passando por esta etapa de conversão.

4.4.4 Função *Sub_Adders_Unique*

Toda vez que uma arquitetura MCM é gerada através da função *MCM_to_SV*, conjuntos de somadores e subtratores são gerados, contendo seus respectivos tamanhos em número de bits.

Para evitar que arquiteturas de somadores e subtratores sejam gerados em redundância, essa função gera conjuntos de somadores/subtratores únicos.

Ao final, essas arquiteturas são geradas de forma comportamental. Utilizando a descrição comportamental nestes blocos, facilita-se que, posteriormente, ferramentas de síntese identifiquem-nos e os implementem utilizando blocos otimizados. Mais detalhes sobre essa escolha serão apresentados nas etapas de descrição da síntese dos circuitos.

4.5 Composição das Arquiteturas

Depois de todos os blocos serem definidos, existe uma última função em matlab que elabora a entidade principal denominada *FFT_N*. Na forma de etapas, esse algoritmo é descrito a seguir:

- **Cabeçalho**

  São impressos dados da arquitetura, inserida a escala de tempo (*'timescale*) e incluídos (*'include*) as instâncias, blocos que serão utilizados na entidade principal.

- **Definição do Módulo**

  Define o nome do módulo e declara os sinais de entrada e saída, de acordo com número de pontos.

- **Definição de Sinais Intermediários**

  Nesta etapa, os sinais intermediários utilizados para interconexão de blocos são declarados. Na arquitetura da FFT esses sinais compreendem diretivas de controle, caminhos de realimentação do circuito e conexões de multiplexadores.

- **Instanciamento dos Blocos**

  O instanciamento dos blocos é feito nesta etapa, e segue uma ordem definida de cima para baixo, da esquerda para direita, no fluxo de dados da FFT: FSM, multiplexador de entrada, multiplexadores de mapeamento, borboletas e multiplexador de saída.

  Durante a etapa de instanciamento dos multiplexadores de mapeamento, é utilizado o algoritmo *index_mapping* descrito na Seção 4.2.2. As borboletas são instanciadas utilizando como entradas os sinais de saída (real, imaginário) destes blocos. Por fim, essas borboletas alimentam os sinais de entrada do multiplexador de saída.

- **Realimentação**

  A etapa de realimentação do circuito é realizada no ciclo positivo do sinal de relógio. É definida por um bloco *always*, após o instanciamento dos blocos da FFT, descritos na etapa anterior.

A entidade principal, descrita em SystemVerilog, de uma arquitetura FFT de estágio único, *radix-2* DIT de 8 pontos pode ser observada no Listing A.1 no Apêndice A, onde as etapas descritas anteriormente são definidas através de comentários no código.

4.6 Verificação Funcional

A verificação funcional é uma etapa pré-síntese lógica que tem o intuito de conferir se a arquitetura atende às funcionalidades as quais ela foi projetada.

Normalmente, a verificação tenta abranger o máximo possível de caminho de dados ou possibilidades de entrada. Entretanto, como a FFT possui uma quantidade relativamente alta de combinações de entrada, serão propostos três casos principais para teste nas arquiteturas:

- **Caso 1**

  É inserido um sinal CC na entrada da FFT. Todas as entradas, real e imaginária, possuem magnitude de 2^{Q-1}. Esse sinal irá resultar em um delta exatamente no índice zero no espectro de magnitude.

- **Caso 2**

  Um sinal com maior frequência possível é inserido na entrada da arquitetura. De tal maneira, a entrada $x_{re}[n] = x_{im}[n] = 2^{Q-1}$ e $x_{re}[n+1] = x_{im}[n+1] = -2^{Q-1}$. Como resultado, espera-se obter um delta exatamente na saída que representa a maior frequência, no gráfico de magnitude do sinal de saída.

- **Caso 3**

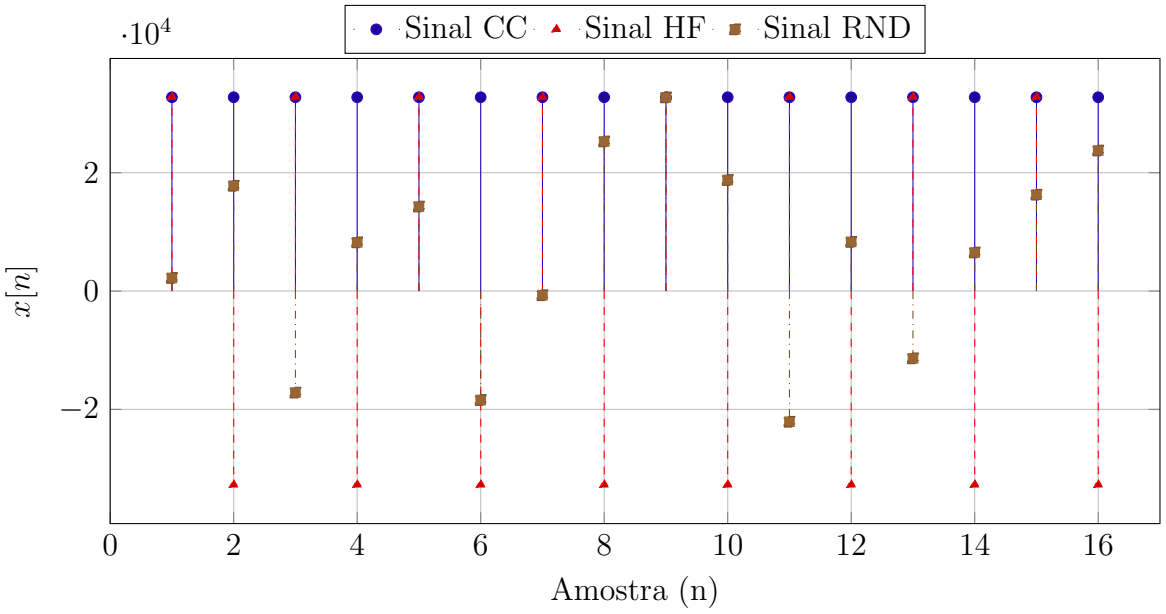
  Serão inseridos 10 conjuntos de sinais randômicos, com amplitude dentro da faixa de -1 a 1, quantizados por Q. Esse sinal tem por característica excitar todas as saídas da FFT. Assim, apresentam-se componentes em todo o espectro de magnitude.

A verificação utiliza um *testbench*, também descrito em SystemVerilog, que contém uma instância da FFT, um sistema de leitura dos vetores de entradas, e um de armazenamento de resultados. As entradas são geradas via Matlab, salvas em um arquivo .txt. Os resultados são salvos novamente em um arquivo .txt, lidos no Matlab e comparados com os resultados da função *fft*, disponível no ambiente Matlab.

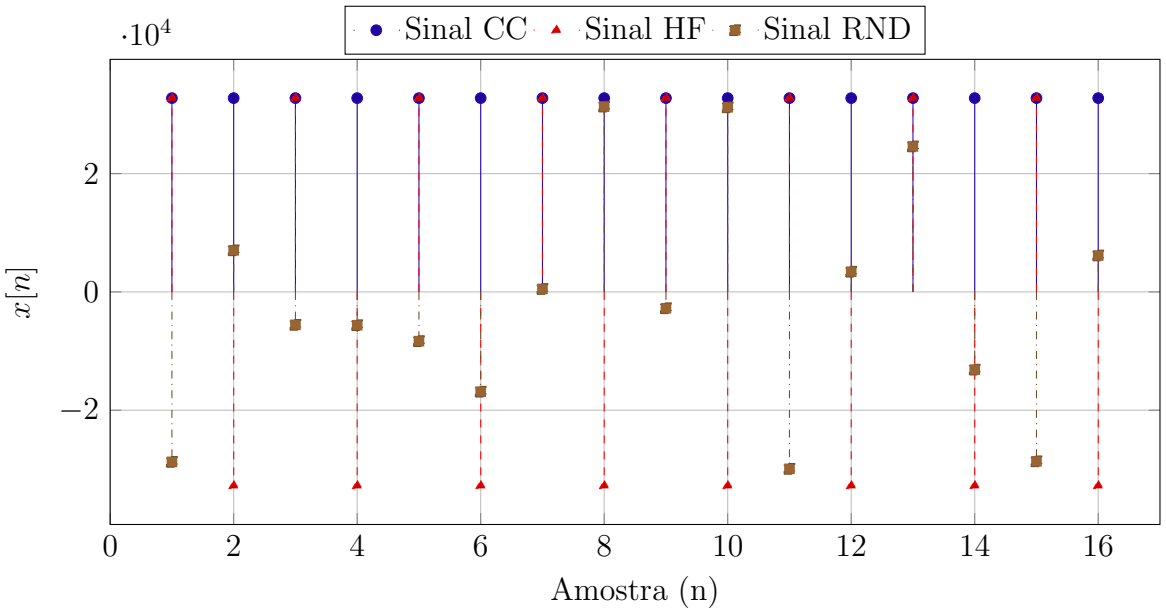
Para realizar a verificação funcional das arquiteturas é utilizado o software industrial VCS da Synopsys. As amostras dos sinais de entrada utilizados podem ser observadas na Figura 15. Em 15a, a componente real e em 15b a componente imaginária dos três tipos de sinais: contínuo (CC), alta frequência (HF) e randômico (RND).

Com os resultados desses testes é determinado o erro médio quadrático (MSE), representado pela Eq. (4.3). Para esse cálculo são separados valores reais e imaginários, como se todas as saídas fossem reais.

$$MSE(\%) = \frac{\sqrt{(fft_n - X_n)^2}}{2N.2^{Q-1}}.100 \tag{4.3}$$



(a) Componente real.



(b) Componente imaginária.

Figura 15 – Amostra dos sinais utilizados para verificação.

Onde:

fft_n - são os resultados de saída da função *fft* no Matlab;

X_n - são os resultados de saída da arquitetura FFT sob verificação.

O cálculo do MSE é realizado para cada um dos casos. Contudo, para os conjuntos randômicos é efetuada uma média dos erros considerando os dez conjuntos.

Os dados randômicos possibilitam ainda calcular a razão sinal-ruído (SNR), utilizando a Eq. (4.4) e considerando os sinais de saída na forma complexa. Para a FFT, o conceito de SNR é considerado de forma diferente: é uma razão entre o sinal teórico de

saída e a diferença entre sinal teórico e o valor resultante da arquitetura, considerando como entrada um mesmo sinal randômico uniformemente distribuído.

$$SNR(dB) = 10.\log_{10} \cdot \left[\frac{\sum_{n=0}^{N-1} |fft_n|^2}{\sum_{n=0}^{N-1} |fft_n - X_n|^2} \right] \tag{4.4}$$

4.7 Síntese Lógica

Após a verificação de cada uma das arquiteturas, pode-se iniciar a síntese lógica das mesmas arquiteturas. Esse processo tem como objetivo mapear todas funcionalidades e blocos, descritos em SystemVerilog, para um código correspondente utilizando células contidas na biblioteca de uma *foundry*.

Neste trabalho é utilizado a biblioteca de tecnologia da XFAB Semiconductor Foundries AG, de $0.18\mu m$. São blocos fabricados em tecnologia CMOS que tem como características baixa potência e tensão de alimentação em 1,8V [36].

Para efetuar a síntese lógica, utiliza-se do software industrial Design Compiler (DC) da Synopsys. Dentro desse ambiente, são informadas as restrições e características de cada arquitetura. Para facilitar a síntese, a mesma foi dividida em duas etapas: Leitura e Compilação.

4.7.1 Leitura

A etapa de leitura inclui as bibliotecas para realizar o mapeamento das arquiteturas XH018 e dw_foundation. A primeira consiste no arquivo disponibilizado pela *foundry*, que contém a descrição das células na tecnologia $0.18\mu m$. A segunda é uma biblioteca disponível no DC, que tem como parte de suas funcionalidades mapear descrições comportamentais (operadores matemáticos) para blocos estruturais otimizados.

Posteriormente, são realizadas as leituras dos arquivos em SystemVerilog de todos os blocos que compõem o projeto: multiplexadores, borboletas, controle, blocos MCM, somadores, subtratores, complemento de 2 e a própria FFT.

4.7.2 Compilação

A parte de compilação consiste em efetuar propriamente a síntese. Inicialmente, é configurado o projeto atual (*current_design*). Depois, é definido o sinal de relógio com um determinado período em nanosegundos, vinculado ao sinal de relógio da arquitetura em questão. Define-se a restrição de área como sendo 0, para que a ferramente priorize otimização em área. Por fim, é definido um modelo de carga nos fios, de acordo a tecnologia utilizada.

O passo seguinte é realizar a compilação, que irá converter os arquivos SystemVerilog para a tecnologia, incluindo uma série de etapas de otimização de potência e área, procurando atingir as especificações de tempo das células.

Por fim, é gerada uma arquitetura ainda em linguagem de hardware, agora Verilog. É também gerado um arquivo que contém todas as restrições do projeto (.sdc) e um arquivo que contém todas as informações de tempo da arquitetura (.sdf). Além disso, são gerados três relatórios finais contendo dados de área, potência e tempo.

Uma atenção especial deve ser dada para o relatório de tempo, pois é nele que encontra-se o *slack*. Esse dado informa se o período configurado anteriormente é suficiente para que o sinal de saída seja atualizado, em relação à alteração dos valores de entrada. Caso esse valor seja negativo, é necessário efetuar novamente a síntese alterando o período do sinal de relógio ou então configurando para que a arquitetura não possui uma restrição tão elevada em área.

5 Resultados e Análises

Este capítulo apresenta os resultados obtidos durante as etapas de otimização das arquiteturas FFT. Na seção 5.1, são apresentados os dados de redução na quantidade de coeficientes implementados. Os resultados de área e dissipação de potência entre arquiteturas de borboletas utilizando MCM e somente multiplicadores são apresentados na seção 5.2. Dados de MSE e SNR de cada uma das arquiteturas, obtidos na etapa de verificação funcional, são expostos na seção 5.3.

Os resultados finais pós-síntese lógica de área, dissipação de potência e frequência de operação de cada uma das arquiteturas são apresentados e discutidos na seção 5.4.

5.1 Operações Implementadas

A utilização dos blocos MCM possibilita que coeficientes redundantes sejam eliminados durante a elaboração das arquiteturas das borboletas. Na etapa para determinação dos coeficientes, o algoritmo identifica esses coeficientes redundantes e elimina-os, considerando que em estágios posteriores os resultados são obtidos através de multiplexação.

A exclusão desses coeficientes redundantes acarreta na diminuição de hardware para implementação de multiplicações reais. Na Figura 16 é possível analisar a redução percentual na quantidade de multiplicações reais implementadas em cada uma das arquiteturas.

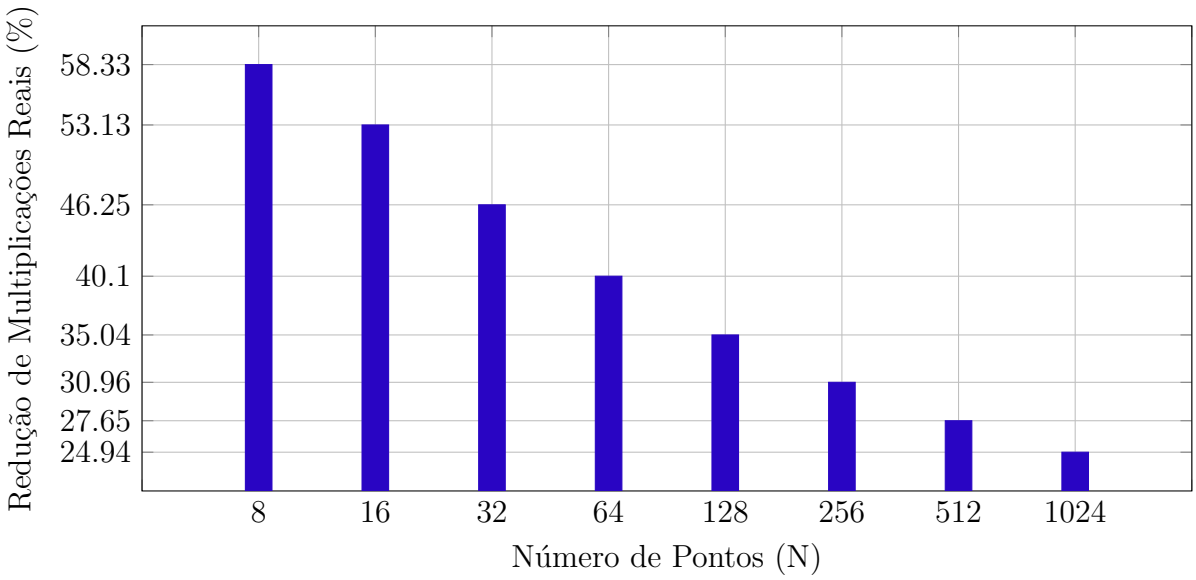


Figura 16 – Gráfico de redução na quantidade de multiplicações reais implementadas.

5.2 Otimização das Borboletas

Para realizar uma comparação entre implementação utilizando somadores ou multiplicadores, elaboraram-se duas arquiteturas, uma com quatro somadores e outra com quatro multiplicadores. Efetuou-se a síntese lógica de cada um desses blocos para obter os valores de potência e área totais. A escolha de quatro multiplicadores está relacionada com a quantidade de multiplicações reais necessárias para implementar a multiplicação complexa dentro de uma borboleta. Considerou-se arquiteturas de 32 bits e frequência de operação 25MHz. Os resultados podem ser observados na Tabela 9.

Tabela 9 – Resultados de área e potência para as arquiteturas de comparação.

	Área (mm^2)	**Potência (mW)**
Somadores	0,0083	0,337
Multiplicadores	0,0626	1,6122

Tabela 10 – Média de somadores presentes nas multiplicações complexas das borboletas implementadas via MCM.

N	**Somadores**
	$\overline{x}$
8	5,00
16	8,00
32	12,25
64	16,62
128	19,03
256	25,28
512	29,38
1024	32,21

A razão de potência e área entre as arquiteturas é 7,54 e 4,78, respectivamente. Dessa forma, considerando-se que o bloco somador possui quatro somadores, multiplicam-se as razões por quatro e arredonda-se o resultado para o inteiro abaixo. Obtêm-se que, por critério de potência podem-se utilizar até 19 somadores para constituir uma borboleta e por área até 30.

Considerando esses dados, foi obtido o número médio de somadores para implementação da multiplicação complexa via MCM em cada borboleta, de 8 até 1024 pontos, apresentados na Tabela 10, desconsiderando os quatro últimos somadores que são comuns em todas as borboletas. Conclui-se que, para *radix-2*, até 128 pontos a metodologia obtém

otimização em potência e área e, até 512 otimização em área. A arquitetura de 1024 pontos não se torna viável para implementação com a metodologia apresentada, testes com diferentes *radixes* podem ser futuramente explorados.

5.3 Análise de MSE e SNR

Durante a etapa de verificação funcional extraíram-se resultados das arquiteturas em função dos tipos de sinais de entrada. Utiliza-se dos indicadores de MSE e SNR para mensurar o efeitos causados nos resultados em relação aos truncamentos e arredondamento dos coeficientes.

A partir dos dados da Tabela 11, é possível apreciar os resultados de MSE para as arquiteturas. Nota-se que o erro é crescente em relação ao número de pontos das arquiteturas. Isso ocorre em função da propagação do erro de truncamento.

Tabela 11 – Resultados de MSE das arquiteturas para sinais randômicos, contínuo e de alta frequência.

N	**Randômico**		**CC**	**HF**
	$\overline{\mathbf{x}}$ (%)	σ (%)	(%)	(%)
8	0,0189	0,0031	0,0092	0,0046
16	0,0416	0,0032	0,0122	0,0122
32	0,132	0,0087	0,0153	0,0122
64	0,2966	0,0224	0,0183	0,0183
128	0,5982	0,0136	0,0214	0,0183
256	1,1731	0,0434	0,0244	0,0244
512	2,1929	0,0395	0,0275	0,0244

No caso randômico, em que ocorre o maior acionamento de saídas das FFTs, esse erro se mostra relevante para arquiteturas acima de 512 pontos, chegando a $5,9\%$ para a arquitetura de 1024 pontos. Foram utilizados 10 conjuntos de vetores randômicos, que possibilitaram calcular uma média e também o desvio padrão, que foi menor que $0,05\%$ em todos os casos, mostrando que $99,7\%$ dos dados apresentam erro entre a média: $\overline{x} \pm 0,15\%$. Nas arquiteturas abaixo de 512 pontos, os resultados foram bastante satisfatórios, apresentando erros médios menores que $1,5\%$.

Com os vetores de teste CC e HF, a quantidade de sinais de saída ativados é relativamente baixa e a diferença entre resultados teóricos e de verificação também é pequena, assim, obtêm-se indicadores baixos. Nota-se que, até mesmo para a arquitetura de 512 pontos os resultados ficaram em torno de $0,020\%$.

Os resultados de SNR possuem comportamento inverso com relação ao MSE, quanto maior, melhor, pois representam a relação entre a potência total do sinal teórico e a potência total do erro entre os resultados teóricos e os observados nas saídas das arquiteturas. Esse dados podem ser analisados na Tabela 12.

Tabela 12 – Resultados de SNR das arquiteturas para sinais randômicos, contínuo e de alta frequência.

N	**Randômico**		**CC**	**HF**
	$\overline{\mathbf{x}}$ **(dB)**	σ **(dB)**	**(dB)**	**(dB)**
8	76,52	2,22	80,77	88,55
16	72,23	0,86	78,27	81,28
32	65,37	0,52	76,33	81,28
64	61,03	0,52	74,75	77,76
128	57,58	0,18	73,41	77,76
256	54,77	0,31	72,25	75,26
512	52,11	0,20	71,22	75,26

Os resultados relacionados aos sinais randômicos são mais relevantes que os de sinais CC e HF. Isso se deve, pois, os sinais randômicos tem como característica proporcionarem uma elevada atividade nos sinais de saída das arquiteturas.

A característica dos sinais randômicos proporcionarem uma atividade elevada nas saídas das arquiteturas torna a análise de SNR extremamente relvante. Neste caso as arquiteturas apresentaram SNR na faixa de 76,52-52,11dB, que estão dentro do padrão de arquiteturas atuais que apresentam SNR acima dos 40dB [23].

Nota-se um declínio elevado da SNR conforme o aumento do número de pontos e consequente número de estágios das arquiteturas. Novamente, isso ocorre em função dos erros de truncamento nas arquiteturas, que fazem com que a potência do erro se torne elevada também.

Os resultados para sinais CC e HF dão uma ideia de limites máximos de SNR, pois apresentam um mínimo número de saídas ativas nas arquiteturas, com amplitudes máximas.

5.4 Análise de PPA

Após a etapa de síntese lógica dos circuitos são gerados relatórios que contém os dados de área e potência. Esses resultados são apresentados na Tabela 13. Repare que são

Tabela 13 – Resultados de área das arquiteturas FFT de estágio único.

N	Comb. (mm^2)	Não-Comb. (mm^2)	Interc. (mm^2)	Total (mm^2)
8	0,0502	0,0224	0,0030	0,0756
16	0,3019	0,0764	0,0129	0,3912
32	0,9485	0,1646	0,0392	1,1525
64	2,6374	0,3539	0,1158	3,1073
128	6,7624	0,7578	0,2898	7,8102
256	16,1191	1,6161	0,8711	18,607

apresentados resultados para as arquiteturas até 256 pontos, acima desse número houve problemas com recursos computacionais para finalizar etapas de otimização e mapeamento das arquiteturas, fazendo com que a ferramenta DesignCompiler apresentasse erro crítico e finalizasse o processo de síntese.

O dado de área total apresentado é composto por três categorias de áreas: combinacional, não-combinacional e interconexão de células. A parte combinacional representa em torno de 70% a 85% da área total e está relacionada com blocos somadores e circuitos de multiplexação. A parte não-combinacional está relacionada com blocos e variáveis de controle e registradores utilizados para etapas de realimentação e compreende área relativa em torno de 26% a 11% da total. A área de interconexão das células utiliza o modelo de fios definido na etapa de síntese para estimar a área necessária para realizar as conexões entre os blocos e compreende cerca de 4% da área total nas arquitetura propostas.

A área total dos circuitos que vai de $0.0756 - 18.607mm^2$ pode ser considerada elevada para arquiteturas atuais de FFTs. Isso se explica pelo *radix* adotado e também pelo formato semiparalelo vertical da arquitetura. Arquiteturas com *radix* elevados possibilitam utilizar menos borboletas e como demonstrado no gráfico da Figura 17, a razão área/borboleta das arquiteturas varia de forma logarítmica em relação ao número de pontos. Contudo, com um *radix* maior, pode-se obter um decréscimo em área proporcional ao de borboletas, embora as mesmas se tornem mais complexas. A estrutura vertical das arquiteturas contribui para o desempenhos das mesmas, todavia, sacrifica área quando comparadas com modelos *pipelined*e baseados em memória.

Todavia, como demonstrando anteriormente, essas arquiteturas apresentariam área muito mais elevada se considerada a utilização massiva de multiplicadores, buscando manter a característica de desempenho das arquiteturas.

Os resultados de potência demonstrados na Tabela 14, são divididos em três tipos:

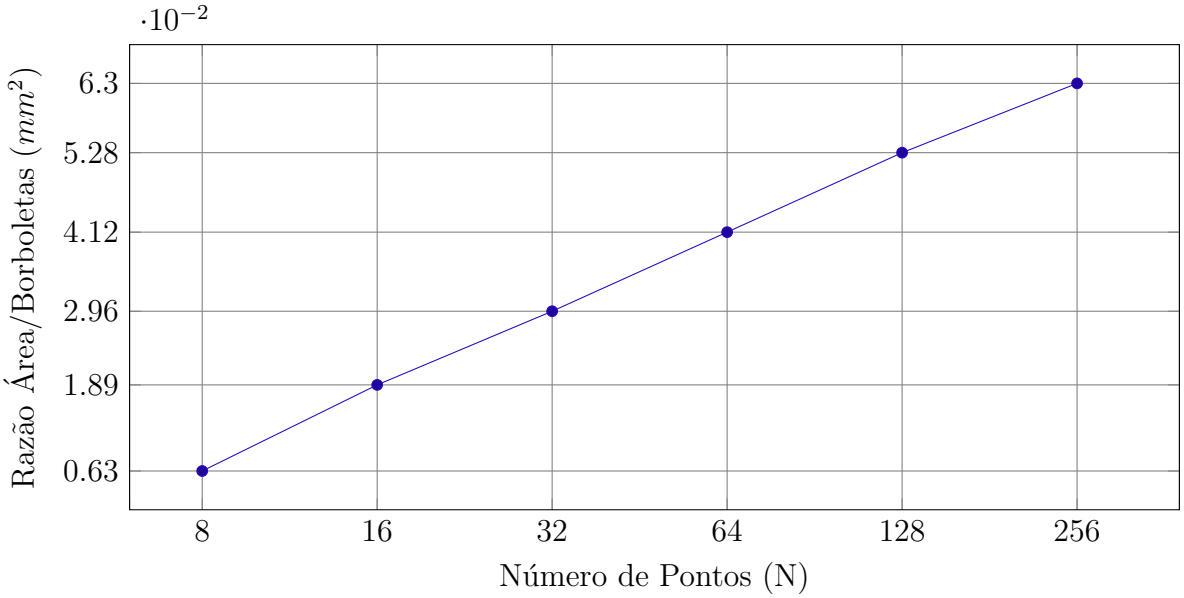


Figura 17 – Razão da área em relação a quantidade de borboletas nas arquiteturas.

Escala X: $\log_2$.

interna, chaveamento (dinâmicas) e *leakage* (estática). A potência interna esta relacionada com o carregamento de cargas internas e também com a corrente de curto-circuito entre transistores do tipo P e N no instante que ambos estão ligados. A potência de chaveamento relaciona o consumo de energia com a frequência de chaveamento sobre a capacitância de saída das células. A potência de *leakage* é dependente da temperatura, tensão e estado dos transistores, tendo sua maior parte originada por correntes de fuga entre fonte e dreno.

A potência interna é a componente com maior parcela da potência total em função da variação dos estados nos transistores. Atrelado a essa característica de mapeamento entre estágios da FFT, mas considerando acima de tudo a frequência de operação dos circuitos, obtêm-se a segunda maior parcela da potência total: a de chaveamento. Para as arquiteturas, ambos os tipos de potência tiveram um incremento proporcional ao número de pontos, e consequente quantidade de borboletas.

A menor parcela é a potência de *leakage* que, por características da tecnologia utilizada, tende a ser baixa. No geral, representa menos de 0.00003% da potência total. Essa parcela se torna relevante em tecnologia abaixo de $0.90\mu m$, onde as parcelas de potência interna e de chaveamento apresentam valores absolutos menores, fazendo com que a mesma contribua de forma mais expressiva no percentual total.

Com relação ao desempenho das arquiteturas, optou-se por sintetizá-las com frequência de operação em 25MHz. Essa decisão foi tomada em função da disponibilidade de recursos computacionais, visto que com requisitos de área mínima, a escolha de frequências elevadas resultava em elevados tempos e etapas de otimização na ferramenta (dias) e também a possibilidade de o *slack* (folga) não ser atingido e ficar negativo. Baseado

Tabela 14 – Resultados de potência das arquiteturas FFT de estágio único.

N	**Potência**			
	Interna (mW)	**Chaveamento (mW)**	***Leakage* (nW)**	**Total (mW)**
8	1,2295	0,4739	0,3115	1,7034
16	5,4588	2,8764	1,5936	8,3352
32	13,8476	8,3449	4,6629	22,1925
64	31,8782	20,9894	12,5090	52,8674
128	77,1938	56,1033	31,2580	133,2975
256	191,0473	167,6294	73,858	358,6802

nessa frequência de operação é possível determinar alguns parâmetros de desempenho das arquiteturas como demonstrados na Tabela 15.

Tabela 15 – Dados de desempenho das arquiteturas FFT de estágio único.

N	$\mathbf{t_{FFT}}$ (μ**s**)	**FFT/s (milhões)**	**Amostras/s (milhões)**
8	16	6,25	50
16	20	5	80
32	24	4,17	133
64	28	3,57	229
128	32	3,13	400
256	36	2,78	711

O indicador t_{FFT} representa o tempo necessário para que as arquiteturas executem a FFT de um conjunto de dados de entrada. Para isto foi utilizado o período do relógio multiplicado pelo número de estágios acrescido em um (ciclo de saída estável). Dessa maneira, conclui-se que o tempo de processamento das arquiteturas é unicamente dependente do período de operação e quantidade de estágios. Como esperado, a arquitetura com menor tempo de processamento é a de 8 pontos, com $16\mu s$, em função de ser a arquitetura com borboletas simples e possuir a menor quantidade de estágios. De forma análoga a que possui o maior tempo de processamento é a de 256 pontos, novamente por possuir número de estágios mais elevado e borboletas mais complexas.

A quantidade de FFTs computadas por segundo (FFT/s) é calculada através do inverso do indicador t_{FFT}. Assim sendo, arquiteturas com menos estágios concluem mais

FFTs do que aquelas com uma quantidade maior, operando em mesma frequência.

Considerando que o número de amostras processadas por cada FFT é igual ao número de pontos da mesma (não utilizar *zero padding*[1]), é possível calcular a quantidade de amostras processadas por segundo através da multiplicação do indicador FFT/s pela quantidade de pontos. Neste caso, a FFT de 256 pontos é a que possui maior indicador com 711 milhões de amostras por segundo e a de 8 pontos o menor com 50 milhões de amostras por segundo.

Em geral, todos os indicadores podem ser melhorados, incluindo área e potência, escolhendo uma tecnologia de fabricação menor. Tecnologias menores possibilitam frequências de operação maiores, contribuindo para o desempenho, além de ocuparem menor área e diminuírem o consumo de potência dinâmica. Com a tecnologia utilizada neste trabalho, o aumento da frequência de operação elevaria resultados de desempenho e afetaria negativamente a área e consumo de energia das arquiteturas.

[1] O ato de preencher entradas sem amostras de uma FFT com zeros, é denominado *zero padding*. Normalmente utilizado quando o número de amostras não é uma potência de 2 e se deseja utilizar um algoritmo que utiliza quantidades de entradas nesta potência. Consultar [1] para maiores informações.

6 Considerações Finais

Este trabalho apresentou uma metodologia para implementação e otimização de arquiteturas FFT de estágio único, DIT, *radix-2* utilizando MCM. Foram substituídos blocos multiplicadores por blocos equivalentes que utilizam apenas somadores, subtratores e deslocamentos binários. Essa substituição foi obtida através de um algoritmo de busca de solução MCM.

O desenvolvimento de um algoritmo para descrição automática das arquiteturas se tornou viável pelo fato de que erros de descrição nas arquiteturas puderam ser corrigidos de forma eficaz, diminuindo o tempo que esse processo levaria caso fosse efetuado de forma manual. De maneria similar, a integração do algoritmo que gera as arquiteturas e o que disponibiliza as soluções MCM reduziu drasticamente o tempo necessário para transcrição das soluções à linguagem de hardware SystemVerilog.

A escolha de isolar coeficientes únicos por linha da FFT possibilitou a redução na quantidade de multiplicações reais implementadas, contribuindo para a diminuição de área na arquitetura de cada borboleta. Ainda assim, por questão de comparação, foram desenvolvidas arquiteturas testes, compostas exclusivamente por multiplicadores e comparadas com implementações de somadores. Obtém-se dos testes, os limites de aplicabilidade da metodologia em radix-2, em termos de otimização em área e potência com relação ao número de pontos.

Durante a etapa de verificação funcional, foi possível determinar as consequências dos truncamentos realizados entre estágios e do arredondamento dos coeficientes. Essas consequências foram quantificadas através dos cálculos de MSE e SNR, que se mostraram satisfatórios para arquiteturas até 512 pontos. Embora, para a de 512 pontos, o MSE fosse um pouco elevado. Como solução para este problema, sugere-se a implementação de arquiteturas com um nível de truncamento menor, excluindo, por exemplo, o truncamento realizado ao final dos últimos somadores em cada borboleta.

Após a etapa de síntese lógica, os resultados de área e dissipação de potência para cada uma das arquiteturas foram apresentados. A maior parcela da potência total é definida por blocos combinacionais, representados pelos dispositivos somadores/subtratores, constituintes das borboletas. Ainda assim, nos blocos de realimentação utilizam-se registradores que contribuem com uma parcela de área não-combinacional. E, não menos importante, pois é uma característica das arquiteturas FFTs, tem-se a parcela de interconexão de blocos que resultou numa média de 4% da área total.

A potência dinâmica, definida por potência de chaveamento e interna das células, foi a que apresentou parcela quase integral na potência total. Isso se deve em função da

característica da tecnologia de fabricação XFAB 0.18μmapresentar potência de *leakage* praticamente desprezível.

Apresentou-se também, indicadores de desempenho das arquiteturas como: o tempo necessário para se computar uma FFT, a quantidade de FFTs executadas por segundo e a quantidade de amostras processadas por segundo. Relacionou-se esses indicadores em função da frequência de operação dos circuitos e também em relação a quantidade de estágios em cada arquitetura.

Conclui-se que, pelo fato de o *radix* utilizado ser baixo, requer-se uma quantidade elevada de borboletas, que por consequência impactam diretamente nos resultados de PPA. É sugerido então, implementar arquiteturas com diferentes *radixes* (4, 8 e 16), almejando que a diminuição da quantidade de borboletas resulte em dados de PPA ainda melhores. E o aumento na complexidade das borboletas pode ser considerado como um fator benéfico para o algoritmo MCM, pois, o aumento do número de coeficientes resulta em maiores possibilidades de compartilhamento de subexpressões.

6.1 Trabalhos Futuros

Para dar continuidade neste trabalho são sugeridas algumas atividades:

- Analisar efeitos de truncamento utilizando diferentes níveis de quantização nas entradas e nos coeficientes;
- Distinguir níveis de quantização em relação a quantidade de pontos (e estágios);
- Descrever implementações com valores diferentes de quantização entre entradas e coeficientes (Q_E e Q_C);
- Realizar análises de área para implementações com níveis de truncamento menores;
- Alterar os algoritmos de indexamento, obtenção de coeficientes e descrição das arquiteturas para utilizarem *radix-r*;
- Verificar o impacto do aumento na complexidade das borboletas *radix-r* nos resultados de PPA;
- Sintetizar as arquiteturas em tecnologias mais atuais, abaixo de 90μm, para que análises mais específicas possam ser realizadas em relação a potência de *leakage*, e também para que maiores frequências de operação possam ser atingidas.

6.2 Prototipação

O autor gostaria de agradecer o Interuniversity Microelectronics Center - IMEC que através do programa "Free IC Fabrication minisic Program" para Universidades Brasileiras, possibilitou a prototipação de uma FFT de estágio único, DIT, *radix-2* de 8 pontos. Essa arquitetura possui um bloco para receber os dados de forma serial, outro para enviar os dados de forma serial e ainda dois blocos para as conversões serial/paralelo e paralelo/serial. Esse projeto já está em fase de fabricação e futuramente serão realizados testes de funcionalidade no chip físico.

Além da FFT, foram implementados no mesmo chip o microcontrolador Pampium, amplificadores operacionais, filtros ativos, fontes de referência e transistores de teste. A tecnologia utilizada é CMOS MS/RF de 0.18μmcom 6 metais da TSMC. O chip possui área total igual a $1660 \times 1660 \mu m^2$ com 48 pinos. A Figura 18 apresenta todas as arquiteturas implementadas com a FFT destacada dentro do quadrado preto. Os dados referente a FFT podem ser acompanhados na Tabela 16.

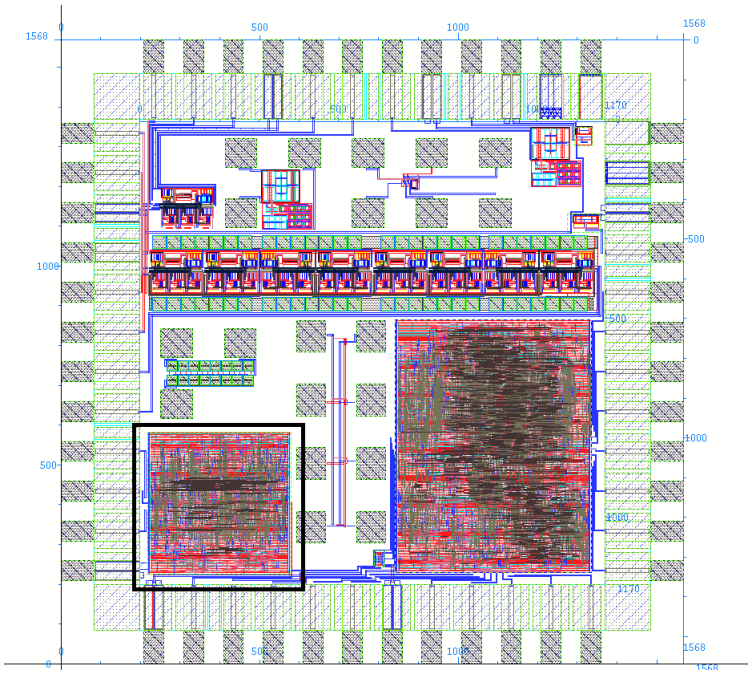


Figura 18 – Layout (quadrado preto) da FFT de estágio único, DIT, radix-2 de 8 pontos.

Tabela 16 – Resultados da Implementação da FFT de estágio único de 8 pontos.

Área Total	0,1324	mm^2
Potência Total	3,586	mW
Frequência	10	MHz

REFERÊNCIAS

1 MITRA, S. K. *Digital Signal Processing: A computer-based approach*. 4a ed.. ed. [S.l.]: McGraw Hill Education, 2011. Citado 7 vezes nas páginas 23, 27, 29, 32, 34, 35 e 72.

2 HEISKALA, J.; D, J. T. P. *OFDM wireless LANs: A theoretical and practical guide*. [S.l.]: Sams, 2001. Citado na página 23.

3 IEEE Standard for Information technology — Telecommunications and information exchange between systems Local and metropolitan area networks — Specific requirements. *IEEE Std. 802.11-2012*, p. i – 2793, 2012. Nenhuma citação no texto.

4 SUTHERLAND, S.; MILLS, D. Synthesizing systemverilog busting the myth that systemverilog is only for verification. *SNUG Silicon Valley*, 2013. Citado na página 24.

5 DRESSLER, K. Sinusoidal extraction using an efficient implementation of a multi-resolution fft. In: CITESEER. *Proc. of 9th Int. Conf. on Digital Audio Effects (DAFx-06)*. [S.l.], 2006. p. 247–252. Citado na página 27.

6 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. Citado na página 30.

7 SINGLETON, R. C. An algorithm for computing the mixed radix fast fourier transform. *Audio and Electroacoustics, IEEE Transactions on*, IEEE, v. 17, n. 2, p. 93–103, 1969. Citado na página 34.

8 DUHAMEL, P. Implementation of" split-radix" fft algorithms for complex, real, and real-symmetric data. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, IEEE, v. 34, n. 2, p. 285–295, 1986. Citado na página 34.

9 LIU, Z.; ZHONG, S.; CHEN, Y.; CHU, W. Design and optimization on reconfigurable butterfly core for a real-time fft processor. In: *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*. [S.l.: s.n.], 2009. p. 847–850. Citado na página 35.

10 FONSECA, M. B.; MARTINS, J. B. S.; COSTA, E. A. C. da. Design of pipelined butterflies from radix-2 fft with decimation in time algorithm using efficient adder compressors. In: *Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on*. [S.l.: s.n.], 2011. p. 1–4. Citado na página 35.

11 GHISSONI, S.; COSTA, E.; REIS, R. Reusing smaller optimized fft blocks for the realization of larger power-efficient radix-2 ffts. In: *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2015 25th International Workshop on*. [S.l.: s.n.], 2015. p. 169–176. Citado 2 vezes nas páginas 35 e 43.

12 LI, J.; LIU, F.; LONG, T.; MAO, E. Research on pipeline r22sdf fft. In: *Radar Conference, 2009 IET International*. [S.l.: s.n.], 2009. p. 1–5. ISSN 0537-9989. Citado na página 36.

13 HE, S.; TORKELSON, M. Design and implementation of a 1024-point pipeline fft processor. In: *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998.* [S.l.: s.n.], 1998. p. 131–134. Citado 2 vezes nas páginas 36 e 37.

14 POLYCHRONAKIS, N.; REISIS, D.; TSILIS, E. A continuous-flow, variable-length fft sdf architecture. In: *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on.* [S.l.: s.n.], 2010. p. 730–733. Citado na página 36.

15 FAN, C. P.; LEE, M. S.; SU, G. A. A low multiplier and multiplication costs 256-point fft implementation with simplified radix-24 sdf architecture. In: *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on.* [S.l.: s.n.], 2006. p. 1935–1938. Citado na página 36.

16 ALGNABI, Y. S.; ALDAAMEE, F. A.; TEYMOURZADEH, R.; OTHMAN, M.; ISLAM, M. S. Novel architecture of pipeline radix 22 sdf fft based on digit-slicing technique. In: *Semiconductor Electronics (ICSE), 2012 10th IEEE International Conference on.* [S.l.: s.n.], 2012. p. 470–474. Citado na página 36.

17 KIM, M. G.; SHIN, S. K.; SUNWOO, M. H. New parallel mdc fft processor with efiicient scheduling scheme. In: *Circuits and Systems (APCCAS), 2014 IEEE Asia Pacific Conference on.* [S.l.: s.n.], 2014. p. 667–670. Citado na página 36.

18 JANG, J. K.; KIM, M. G.; SUNWOO, M. H. Efficient scheduling scheme for eight-parallel mdc fft processor. In: *2015 International SoC Design Conference (ISOCC).* [S.l.: s.n.], 2015. p. 277–278. Citado na página 37.

19 HASAN, M.; ARSLAN, T. Scheme for reducing size of coefficient memory in fft processor. *Electronics Letters*, v. 38, n. 4, p. 163–164, Feb 2002. ISSN 0013-5194. Citado na página 38.

20 CHANG, C.-H.; WANG, C.-L.; CHANG, Y.-T. A novel memory-based fft processor for dmt/ofdm applications. In: *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on.* [S.l.: s.n.], 1999. v. 4, p. 1921–1924 vol.4. ISSN 1520-6149. Citado na página 38.

21 CHANG, C.-K.; HUNG, C.-P.; CHEN, S.-G. An efficient memory-based fft architecture. In: *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on.* [S.l.: s.n.], 2003. v. 2, p. II–129–II–132 vol.2. Citado na página 38.

22 HSIAO, C. F.; CHEN, Y.; LEE, C. Y. A generalized mixed-radix algorithm for memory-based fft processors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, v. 57, n. 1, p. 26–30, Jan 2010. ISSN 1549-7747. Citado na página 38.

23 ZHANG, J.; LIU, H.; CHEN, T.; ZHANG, B. et al. Enhanced hardware efficient fft processor based on adaptive recoding cordic. *Elektronika ir Elektrotechnika*, v. 19, n. 4, p. 97–103, 2012. Citado 2 vezes nas páginas 38 e 68.

24 PADEKAR, A. S.; BELSARE, S. Design of a cordic based radix-4 fft processor. *International Journal of Computer Science and Information Technologies*, Citeseer, v. 5, 2014. Citado na página 38.

25 SADAT, A.; MIKHAEL, W. B. Fast fourier transform for high speed ofdm wireless multimedia system. In: IEEE. *Circuits and Systems, 2001. MWSCAS 2001. Proceedings of the 44th IEEE 2001 Midwest Symposium on*. [S.l.], 2001. v. 2, p. 938–942. Citado 2 vezes nas páginas 38 e 39.

26 JABER, M. A.; MASSICOTTE, D. The radix-r one stage fft kernel computation. In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. [S.l.: s.n.], 2008. p. 3585–3588. ISSN 1520-6149. Citado na página 38.

27 CAPPELLO, P.; STEIGLITZ, K. Some complexity issues in digital signal processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 32, n. 5, p. 1037–1041, Oct 1984. ISSN 0096-3518. Citado na página 41.

28 PARK, I.-C.; KANG, H.-J. Digital filter synthesis based on minimal signed digit representation. In: ACM. *Proceedings of the 38th annual Design Automation Conference*. [S.l.], 2001. p. 468–473. Citado na página 42.

29 POTKONJAK, M.; SRIVASTAVA, M. B.; CHANDRAKASAN, A. P. Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, IEEE, v. 15, n. 2, p. 151–165, 1996. Citado na página 42.

30 PAŠKO, R.; SCHAUMONT, P.; DERUDDER, V.; VERNALDE, S.; ĎURAČKOVÁ, D. A new algorithm for elimination of common subexpressions. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, IEEE, v. 18, n. 1, p. 58–68, 1999. Citado 2 vezes nas páginas 42 e 43.

31 GUSTAFSSON, O.; WANHAMMAR, L. Ilp modelling of the common subexpression sharing problem. In: IEEE. *Electronics, Circuits and Systems, 2002. 9th International Conference on*. [S.l.], 2002. v. 3, p. 1171–1174. Citado na página 43.

32 AKSOY, L.; GUNES, E. O.; FLORES, P. An exact breadth-first search algorithm for the multiple constant multiplications problem. In: IEEE. *NORCHIP, 2008.* [S.l.], 2008. p. 41–46. Citado 3 vezes nas páginas 43, 44 e 45.

33 GHISSONI, S.; COSTA, E.; LAZZARI, C.; MONTEIRO, J.; AKSOY, L.; REIS, R. Radix-2 decimation in time (dit) fft implementation based on a matrix-multiple constant multiplication approach. In: *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*. [S.l.: s.n.], 2010. p. 859–862. Citado na página 43.

34 GHISSONI, S.; COSTA, E.; MONTEIRO, J.; REIS, R. Combination of constant matrix multiplication and gate-level approaches for area and power efficient hybrid radix-2 dit fft realization. In: *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*. [S.l.: s.n.], 2011. p. 567–570. Citado na página 43.

35 GHISSONI, S.; COSTA, E.; MONTEIRO, J.; REIS, R. Efficient area and power multiplication part of fft based on twiddle factor decomposition. In: *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*. [S.l.: s.n.], 2012. p. 657–660. Citado na página 43.

36 AG, X.-F. S. F. *XH018*. [S.l.], 2012. Disponível em: <http://www.xfab.com/fileadmin/X-FAB/Download_Center/Technology/CMOS/XH018_HV_CMOS_Data_Sheet.pdf>. Citado na página 63.

APÊNDICE A – Entidade Principal da FFT de Estágio Único de 8 Pontos

No Listing A.1 é definida a entidade principal de uma FFT de estágio único, *radix-2*, DIT de 8 pontos. São ressaltadas, através de comentários, as etapas durante a elaboração da entidade: cabeçalho, definição do módulo, definição de sinais intermediários, instanciamento dos blocos e realimentação. A linguagem utilizada é SystemVerilog.

```
//// CABECALHO ////
`timescale 1ns/1ps

`include "./mux_mapp_3_1.sv"
`include "./mux_mcm_3_2.sv"
`include "./cmp2_13b.sv"
`include "./out_mux_FFT_8.sv"
`include "./in_mux_FFT_8.sv"
`include "./fsm_fft_8.sv"
`include "./btfl_0.sv"
`include "./btfl_1.sv"
`include "./btfl_2.sv"
`include "./btfl_3.sv"
`include "./mcm_8_0.sv"
`include "./mcm_8_1.sv"
`include "./mcm_8_2.sv"
`include "./mcm_8_3.sv"
`include "./sub_17b.sv"
`include "./sub_22b.sv"
`include "./sub_13b_btfl.sv"
`include "./sub_14b_btfl.sv"
`include "./adder_13b_btfl.sv"
`include "./adder_14b_btfl.sv"

//// DEFINICAO DO MODULO ////
module FFT_8 (
      input logic clk,
      input logic rst_n,
      input logic go,
      input logic [7:0][13:0]X_re,
      input logic [7:0][13:0]X_im,
      output logic [7:0][13:0]Y_re,
      output logic [7:0][13:0]Y_im,
      output logic done);
```

```
//// DEFINICAO DOS SINAIS INTERMEDIARIOS ////
logic [1:0]       sel_mcm;
logic [3:0]       c2_re;
logic [3:0]       c2_im;
logic         sel_mux_in;
logic         sel_mux_out;
logic [7:0][13:0] feedback_re;
logic [7:0][13:0] feedback_im;
logic [7:0][13:0] in_s_re;
logic [7:0][13:0] in_s_im;
logic [7:0][13:0] mx_o_re;
logic [7:0][13:0] mx_o_im;
logic [7:0][13:0] btfl_o_re;
logic [7:0][13:0] btfl_o_im;

//// INSTANCIAMENTO DOS BLOCOS ////
fsm_fft_8      FSM      (clk, rst_n, go, done, sel_mux_in, sel_mux_out,
    sel_mcm, c2_re, c2_im);

in_mux_FFT_8  INMX     (clk, sel_mux_in, X_re, X_im, feedback_re,
    feedback_im, in_s_re, in_s_im);

mux_mapp_3_1  MAPP_MX0  (sel_mcm, in_s_re[0], in_s_im[0], in_s_re[0],
    in_s_im[0], in_s_re[0], in_s_im[0], mx_o_re[0], mx_o_im[0]);
mux_mapp_3_1  MAPP_MX1  (sel_mcm, in_s_re[4], in_s_im[4], in_s_re[2],
    in_s_im[2], in_s_re[4], in_s_im[4], mx_o_re[1], mx_o_im[1]);
mux_mapp_3_1  MAPP_MX2  (sel_mcm, in_s_re[2], in_s_im[2], in_s_re[1],
    in_s_im[1], in_s_re[2], in_s_im[2], mx_o_re[2], mx_o_im[2]);
mux_mapp_3_1  MAPP_MX3  (sel_mcm, in_s_re[6], in_s_im[6], in_s_re[3],
    in_s_im[3], in_s_re[6], in_s_im[6], mx_o_re[3], mx_o_im[3]);
mux_mapp_3_1  MAPP_MX4  (sel_mcm, in_s_re[1], in_s_im[1], in_s_re[4],
    in_s_im[4], in_s_re[1], in_s_im[1], mx_o_re[4], mx_o_im[4]);
mux_mapp_3_1  MAPP_MX5  (sel_mcm, in_s_re[5], in_s_im[5], in_s_re[6],
    in_s_im[6], in_s_re[5], in_s_im[5], mx_o_re[5], mx_o_im[5]);
mux_mapp_3_1  MAPP_MX6  (sel_mcm, in_s_re[3], in_s_im[3], in_s_re[5],
    in_s_im[5], in_s_re[3], in_s_im[3], mx_o_re[6], mx_o_im[6]);
mux_mapp_3_1  MAPP_MX7  (sel_mcm, in_s_re[7], in_s_im[7], in_s_re[7],
    in_s_im[7], in_s_re[7], in_s_im[7], mx_o_re[7], mx_o_im[7]);

btfl_0       BTFL0     (sel_mcm, c2_re[0], c2_im[0], mx_o_re[0], mx_o_im[0],
     mx_o_re[1], mx_o_im[1], btfl_o_re[0], btfl_o_im[0], btfl_o_re[1],
    btfl_o_im[1]);
btfl_1       BTFL1     (sel_mcm, c2_re[1], c2_im[1], mx_o_re[2], mx_o_im[2],
     mx_o_re[3], mx_o_im[3], btfl_o_re[2], btfl_o_im[2], btfl_o_re[3],
    btfl_o_im[3]);
btfl_2       BTFL2     (sel_mcm, c2_re[2], c2_im[2], mx_o_re[4], mx_o_im[4],
     mx_o_re[5], mx_o_im[5], btfl_o_re[4], btfl_o_im[4], btfl_o_re[5],
```

```
    btfl_o_im[5]);
btfl_3      BTFL3      (sel_mcm, c2_re[3], c2_im[3], mx_o_re[6], mx_o_im[6],
     mx_o_re[7], mx_o_im[7], btfl_o_re[6], btfl_o_im[6], btfl_o_re[7],
    btfl_o_im[7]);

out_mux_FFT_8   OUTMX      (clk, sel_mux_out, btfl_o_re, btfl_o_im, Y_re,
    Y_im);

//// REALIMENTACAO ////
always_ff @(posedge clk) begin
     feedback_re <= btfl_o_re;
     feedback_im <= btfl_o_im;
end
endmodule
```

Listing A.1 – Entidade principal de uma FFT de estágio único, *radix-2*, DIT de 8 pontos. Linguagem: SystemVerilog.