UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

CHRYSTIAN ALEXANDER CRUZ PATRIOTA

DESENVOLVIMENTO DE UMA UNIDADE DE PONTO FLUTUANTE EM FPGA COM APLICAÇÕES EM PROCESSAMENTO DIGITAL DE SINAIS

PAU DOS FERROS, RN
2023

CHRYSTIAN ALEXANDER CRUZ PATRIOTA

DESENVOLVIMENTO DE UMA UNIDADE DE PONTO FLUTUANTE EM FPGA COM APLICAÇÕES EM PROCESSAMENTO DIGITAL DE SINAIS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Pedro Thiago Valério de Souza, Prof. Dr.

PAU DOS FERROS, RN

2023

P314d Patriota, Chrystian Alexander Cruz.
Desenvolvimento De Uma Unidade De Ponto Flutuante Em Fpga Com Aplicações Em Processamento Digital De Sinais / Chrystian Alexander Cruz Patriota. - 2023.
44 f. : il.

Orientador: Pedro Thiago Valerio De Souza.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia de Computação, 2023.

1. Processamento digital. 2. Ponto flutuante. 3. FPGA. 4. Fourier. 5. Goertzel. I. Souza, Pedro Thiago Valerio De , orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

CHRYSTIAN ALEXANDER CRUZ PATRIOTA

DESENVOLVIMENTO DE UMA UNIDADE DE PONTO FLUTUANTE EM FPGA COM APLICAÇÕES EM PROCESSAMENTO DIGITAL DE SINAIS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Defendida em: 18 de maio de 2023.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br PEDRO THIAGO VALERIO DE SOUZA
Data: 25/05/2023 18:37:53-0300
Verifique em https://validar.iti.gov.br

Pedro Thiago Valério de Souza, Prof. Dr. (UFERSA)
Presidente

Documento assinado digitalmente
gov.br KENNEDY REURISON LOPES
Data: 25/05/2023 22:30:05-0300
Verifique em https://validar.iti.gov.br

Kennedy Reurison Lopes, Prof. Dr. (UFERSA)
Membro Examinador

Documento assinado digitalmente
gov.br FRANCISCO CARLOS GURGEL DA SILVA SE
Data: 25/05/2023 22:20:58-0300
Verifique em https://validar.iti.gov.br

Francisco Carlos Gurgel da Silva Segundo, Prof. Dr. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço aos meus pais, Nancy Lucia Alves da Cruz e José Bezerra Patriota, por sempre me proverem tudo que eu precisava para estar nesta instituição e nunca me deixaram faltar nada. Sempre foram a minha base e são as pessoas que sempre vou levar de inspiração para o resto da vida.

A minha namorada, Vyrna Dias de Alcântara, pelo apoio emocional quando eu mais precisei e por sempre estar na torcida por mim em tudo que eu me envolvia durante toda a graduação para que eu sempre pudesse continuar crescendo na minha carreira profissional.

Ao meu orientador e professor, Pedro Thiago Valerio De Souza, por todo ensinamento, auxílio, paciência e pontualidade durante todo esse semestre atual e anteriores para a elaboração desta pesquisa.

Aos meus professores durante a graduação, por compartilharem todo seu conhecimento, possibilitando a elaboração desta pesquisa. Em especial aos professores: Segundo, Cecílio, Adller e Wagner.

Aos meus amigos, por sempre terem me dado suporte durante toda minha graduação. Em especial a Felipe Cardoso Pimenta que sempre foi um grande irmão para mim, com seu suporte em tudo que eu fiz dentro e fora da universidade.

Deixo aqui todos os meus agradecimentos a todas essas pessoas queridas para mim.

RESUMO

A FPGA (Field-Programmable Gate Array) é uma ferramenta amplamente usada na área acadêmica quando se estuda assuntos relacionado a circuitos digitais. Contudo, a FPGA não possui uma unidade aritmética de ponto flutuante, na qual limita o seu uso para atividade mais complexas que demandam uma maior precisão. Esse trabalho teve o objetivo de implementar essas operações em ponto flutuante de 16 bits e testar a sua eficácia através de simulações e testes práticos. Como aplicações para a unidade de ponto flutuante, foram implementados alguns algoritmos clássicos no processamento digital de sinais, como o algoritmo de Goertzel, a FFT (Fast Fourier Transform) e FFT inversa. Os resultados obtidos para a unidade de ponto flutuante foram satisfatórios para todas as operações aritméticas implementadas, com a exceção da divisão, que apresentou erros em 25% dos testes feitos. Em relação as simulações utilizando os algoritmos de processamento digital de sinal, os resultados foram satisfatórios levando em conta a quantidade de operação que são realizadas por cada algoritmo e a comparação com uma referência.

Palavras-chave: Processamento digital; Ponto flutuante; FPGA; Fourier; Goertzel; Aritmética.

ABSTRACT

The FPGA (Field-Programmable Gate Array) is a tool widely used in academia when studying subjects related to digital circuits. However, the FPGA does not have a floating point arithmetic unit, which limits its use to more complex activities that demand greater precision. This work aimed to implement these operations in 16-bit floating point and test their effectiveness through simulations and practical tests. As applications for the floating point unit, some classic algorithms in digital signal processing were implemented, such as the Goertzel algorithm, the FFT (Fast Fourier Transform) and the inverse FFT. The results obtained for the floating point unit were satisfactory for all implemented arithmetic operations, with the exception of division, which presented errors in 25% of the tests performed. Regarding the simulations using the digital signal processing algorithms, the results were satisfactory taking into account the amount of operation that are performed by each algorithm and the comparison with a reference.

Keywords: Digital processing; Floating point; FPGA; Fourier; Goertzel; Arithmetic.

LISTA DE FIGURAS

Figura 1 - Exemplo de dizimação para uma FFT com 16 amostras. 19
Figura 2 - Diagrama de borboleta da FFT 20
Figura 3 - Fluxo da FFT para uma sequência de 8 pontos 20
Figura 4 - Representação de um número em ponto flutuante de 16 bits. 22
Figura 5 - *Altera DE2-115 Board* 23
Figura 6 - Fluxograma da operação de soma e subtração. 24
Figura 7 - Fluxograma da operação de multiplicação 26
Figura 8 - Fluxograma da operação de divisão. 29
Figura 9 - Exemplo em forma de onda 35
Figura 10 - Resultados para a adição. 36
Figura 11 - Resultados para a subtração. 36
Figura 12 - Resultados para a multiplicação. 37
Figura 13 - Resultados para a divisão. 37

LISTA DE QUADROS

Quadro 1 - Pseudocódigo para a implementação de algoritmo de Goertzel. 31
Quadro 2 - Pseudocódigo para a implementação da FFT. 32
Quadro 3 - Sequências utilizadas para produzir resultados. 38
Quadro 4 - DFT das sequências utilizadas para os resultados utilizando a biblioteca NumPy do Python. 38
Quadro 5 - Resultados do algoritmo de Goertzel. 39
Quadro 6 - Resultados da FFT para cada sequência 39
Quadro 7 - DFT da sequência 1 e 2 combinadas.....41
Quadro 8 - DFT das duas sequências obtidas a partir do uso da propriedade de simetria da DFT entre as duas sequências. 41
Quadro 9 - Sequências 1 e 2 obtidas a partir da FFT inversa. 42

LISTA DE ABREVIATURAS E SIGLAS

DFT	*Discrete Fourier Transform*
DFS	*Discrete Fourier Series*
FFT	*Fast Fourier Transform*
IEEE	*Institute of Electrical and Electronics Engineers*
FPGA	*Field-Programmable Gate Array*
UFERSA	Universidade Federal Rural do Semi-Árido
NaN	*Not a Number*
LUT	*Lookup table*
FPU	*Float Point Unit*

SUMÁRIO

1. INTRODUÇÃO 12
2. OBJETIVOS 14
2.1. Objetivo Geral 14
2.2. Objetivos Específico 14
3. FUNDAMENTAÇÃO TEÓRICA 15
3.1. Transformada de Fourier Discreta 15
3.2. Implementação Computacional da Transformada de Fourier Discreta 17
3.3. Algoritmo de Goertzel 17
3.4. Transformada Rápida de Fourier 18
3.5. Ponto Flutuante 21
4. ARQUITETURA DO SISTEMA 23
4.1. Implementação em FPGA 23
4.2. Operações de Adição e Subtração em Ponto Flutuante 24
4.3. Operação de Multiplicação em Ponto Flutuante 26
4.4. Operação de Divisão em Ponto Flutuante 28
4.5. Implementação do Algoritmo de Goertzel 30
4.6. FFT e FFT inversa 32
5. RESULTADOS 35
6. CONCLUSÕES 43
REFERÊNCIAS 44

1. INTRODUÇÃO

Em geral, as operações aritméticas realizadas em *hardware* através de FPGA são implementadas utilizando ponto fixo, já que sua representação de seus valores em ponto fixo é simples e, em muitos casos, ésuficiente para atender às necessidades requeridas (TE EWE; HEUNG; CONSTANTINIDES,2004).

Em certos casos, o uso da representação em ponto fixo pode ser suficiente para implementar uma função a nível de *hardware*. Contudo, em algumas situações, essa abordagem pode tornar-se obsoleta devido à falta de precisão e à ocorrência frequente de erros de *overflow* ou *underflow*. Hettiarachchi, Davuluru e Balster (2020) discutem algumas dessas limitações do ponto fixo em seus estudos.

Em certas situações, como na implementação de filtros digitais com alto ganho, o uso do ponto fixo pode não ser adequado, uma vez que sua resolução não é suficientepara fornecer a precisão necessária, resultando em uma saída instável (OPPENHEIM, 1970).

Desta forma, é fundamental avaliar cuidadosamente as limitações e vantagens de diferentes abordagens de representação numérica ao escolher uma estratégia de implementação em *hardware*. Assim, pode-se garantir a eficiência e a precisão do sistema em sua aplicação específica.

Para evitar esse tipo de problema ao utilizar ponto fixo, muitas vezes é adotada a abordagem do uso de ponto flutuante, que é padronizada pelo padrão IEEE 754. De acordo com Barrois e Sentieys (2017), uma das vantagens do ponto flutuante é a sua dinamicidade, além da sua estrutura já ser bem documentada pelo padrão IEEE 754. Vale ressaltar que a maioria dos computadores já possui uma unidade de operações aritméticas utilizando ponto flutuante. Por outro lado, o ponto fixo é uma solução mais simples, rápida e energeticamente eficiente. Contudo, é preciso levar em conta o custo de tempo necessário para criar o *design* do projeto e projetar o ponto fixo com a precisão suficiente para atender ao fim desejado, além da possibilidade de ocorrerem erros de *overflow* e *underflow*.

Swartzlander e Saleh (2010) afirmam que a aritmética de ponto flutuante é uma solução para esses casos, pois oferece uma ampla faixa dinâmica e evita problemas de *overflow* e *underflow* que são comuns na aritmética de ponto fixo.

Vale salientar que a representação em ponto flutuante permite que o número seja representado por um valor significativo e um expoente, o que permite que uma gama mais ampla de valores seja representada com alta precisão. Essa abordagem é especialmente adequada para aplicações que envolvem cálculos matemáticos complexos ou grandes quantidades de dados numéricos, como simulações físicas, processamento de imagens e análise

de dados científicos (PATTERSON; HENNESSY, 1994).

Por outro lado, é importante observar que o uso da aritmética de ponto flutuante pode ter um custo maior em termos de desempenho e complexidade de *hardware*. É fundamental avaliar cuidadosamente os requisitos de precisão e desempenho da aplicação para determinar se a aritmética de ponto flutuante é a melhor escolha para a implementação do *hardware* (PATTERSON; HENNESSY, 1994).

O objetivo deste trabalho é implementar todas as operações aritméticas básicas (soma, subtração, multiplicação e divisão) utilizando ponto flutuante, conforme especificado pelo padrão IEEE 754 para *half-precision floating-point format* (16 bits). Para isso, será utilizada a linguagem de descrição de hardware *Verilog*. Além disso, as implementações recém-criadas serão utilizadas para executar três algoritmos de processamento digital de sinais diferentes com o intuído de validar a sua implementação: o algoritmo de Goertzel, a FFT e a FFT inversa.

2. OBJETIVOS

2.1. Objetivo Geral

Executar a implementação a nível de *hardware* das quatro principais operações aritméticas (soma, subtração, multiplicação e divisão) em ponto flutuante de 16 bits, testá-las em uma FPGA e realizar a avaliação do seu desempenho a partir dos seus resultados gerados na execução da FFT e a FFT inversa.

2.2. Objetivos Específico

- Desenvolver as quatros operações aritméticas básicas, utilizando a abordagem de ponto flutuante de 16 *bits* definidas pelo IEEE 754;
- Descrever o algoritmo utilizado em cada operação e suas limitações com o intuito de auxiliar implementações futuras;
- Realizar a validação das operações em uma FPGA.
- Implementar o algoritmo de Goertzel utilizando as operações aritméticas implementadas;
- Implementar o algoritmo de FFT sobre uma sequência de 8 amostras utilizando as operações implementadas;
- Reutilizar a implementação da FFT para produzir a FFT inversa, realizando o mínimo de alterações possíveis;
- Realizar a validação da descrição de *hardware* dos algoritmos propostos, assim como das operações aritméticas;

3. FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, é abordado toda a fundamentação teórica necessária para a realização dos algoritmos propostos. A primeiro momento é discutido de forma breve sobre a operação de transformada discreta de Fourier, do inglês Discrete Fourier Transform (DFT), e logo em seguida são abordadas as formas de implementação computacional da DFT, como o algoritmo de Goertzel e o algoritmo transformada rápida de Fourier, do inglês Fast Fourier Transform (FFT) e da FFT inversa. Por fim, é argumentado sobre a representação em ponto flutuante definida pelo IEEE 754 apenas para o half-precision floating-point format (16 bits).

3.1. Transformada de Fourier Discreta

Antes de iniciar a explicação sobre o que é a DFT, se faz necessário existir uma breve apresentação sobre a transformada de Fourier de tempo discreto. Essa é uma transformada feita em sinais discretos que transforma a sua representação em amostras para uma representação continua em frequência. A DTFT (*Discrete-Time Fourier Transform*) de uma sequência $x[n]$ é definida como (OPPENHEIM; SCHAFER; BUCK, 1998):

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} \tag{1}$$

Por sua vez, a DTFT inversa é calculada como:

$$x[n] = \int_{-\pi}^{\pi} X\left(e^{j\omega}\right) e^{j\omega n} d\omega \tag{2}$$

De acordo com Oppenheim, Schafer e Buck (1998), a transformada de Fourier traz informações importante a respeito da sequência, como a composição de frequências existentes em um determinado sinal. Via de regra, $X\left(e^{j\omega}\right)$ é um número complexo, podendo ser representado na forma polar como:

$$X\left(e^{j\omega}\right) = \left|X\left(e^{j\omega}\right)\right| e^{j\angle X\left(e^{j\omega}\right)} \tag{3}$$

em que $\left|X\left(e^{j\omega}\right)\right|$ é o espectro de magnitude e $\angle X\left(e^{j\omega}\right)$ é o espectro de fase. Além disso, $X\left(e^{j\omega}\right)$ é uma função continua em ω, periódica com período de 2π, ou seja:

$$X\left(e^{j\omega}\right) = X\left(e^{j(\omega+2\pi)}\right) \tag{4}$$

A DTFT é uma importante ferramenta de análise no campo de processamento digital de sinais. Além de permitir a análise frequencial de sinais, a DTFT possui um conjunto importante de propriedades úteis na análise de sistemas lineares invariantes no tempo. Todavia, a implementação computacional da DTFT enfrenta uma barreira, visto que $X\left(e^{j\omega}\right)$ é uma função continua em ω. Como a memória em um computador não é infinita, não seria possível calcular $X\left(e^{j\omega}\right)$ para todos os valores de ω, mas sim apenas para um conjunto discreto e finito de valores. Desta forma, se faz necessário utilizar uma outra abordagem para obter a representação em frequência de uma sequência. Com isso dito, é apresentado por Oppenheim, Schafer e Buck (1998) uma outra forma de representação alternativa de Fourier, chamada de DFT.

A DFT de uma sequência é uma representação em frequência com o mesmo número de pontos da sequência original, representando uma amostragem linearmente espaçada da DTFT desta mesma sequência. Sua derivação vem das series discretas de Fourier, do inglês *Discrete Fourier Series* (DFS) (OPPENHEIM; SCHAFER; BUCK, 1998).

Sendo x[n] uma sequência de duração finita causal, tal que $x[n] = 0$ para $n < 0$ e $n > N - 1$. Matematicamente, a DFT de uma sequência $x[n]$ definida como:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \tag{5}$$

e a DFT inversa como:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] W_N^{-nk} \tag{6}$$

em que $W_N^{nk} = e^{j\left(\frac{2\pi}{N}\right)}$.

Analisando a Equação (5), observa-se que a DFT de *x*[*n*] consiste na amostragem da DTFT de *x*[*n*], ou seja:

$$X[k] = X\left(e^{j\omega}\right)\Big|_{\omega=\frac{2\pi k}{N}} \tag{7}$$

As amostras da DFT estão esqui-espaçadas em frequência de $\Delta = \frac{2\pi}{N}$, o qual denomina-se de resolução espectral.

3.2. Implementação Computacional da Transformada de Fourier Discreta

Considerando um sinal *x*[*n*] real e que $W_N^{nk} = e^{j\left(\frac{2\pi}{N}\right)} = \cos\left(\frac{2\pi}{N}\right) + j\sin\left(\frac{2\pi}{N}\right)$, a Equação (5) pode ser re-escrita como:

$$X[k] = \sum_{n=0}^{N-1} x[n]\cos\left(\frac{2\pi nk}{N}\right) - j\sum_{n=0}^{N-1} x[n]\sin\left(\frac{2\pi nk}{N}\right) \tag{8}$$

Como dito por Oppenheim, Schafer e Buck (1998), a Equação (8) é a implementação direta da DFT, e sua complexidade é de $O(N^2)$. Porém, outros tipos de algoritmos podem diminuir a complexidade original da transformada direta, a exemplo da FFT e do algoritmo de Goertzel.

3.3. Algoritmo de Goertzel

O algoritmo de Goertzel é um exemplo de como a periodicidade da DFT pode ser explorada. Baseia-se em utilizar equações recursivas para calcular a DFT em um valor de frequência específica. A vantagem deste algoritmo é de possibilitar a computação de apenas uma frequência da DFT, assim, diminuindo a complexidade que originalmente era de $O(N^2)$ agora se tornando de $O(N)$ (OPPENHEIM; SCHAFER; BUCK, 1998).

O algoritmo utiliza um par de equações de diferença para calcular a magnitude de uma frequência específica de uma sequência, sem a necessidade de calcular a transformada de

Fourier completa. Ele é especialmente útil quando é necessário calcular a magnitude de uma frequência específica em tempo real, como em aplicações de comunicação de áudio.

No caso, o algoritmo de Goertzel consiste em resolver recursivamente a seguinte equação de diferenças (OPPENHEIM; SCHAFER; BUCK, 1998):

$$v_k[n] = x[n] + 2\cos\left(\frac{2\pi k}{N}\right)v_k[n-1] - v_k[n-2] \tag{9}$$

em que k é o índice da DFT no qual deseja-se calcular. A equação de diferenças é resolvida recursivamente até a amostra $n = N$, no qual pode-se determinar $X[k]$ como sendo:

$$X[k] = v_k[N] - W_N^k v_k[N-1] \tag{10}$$

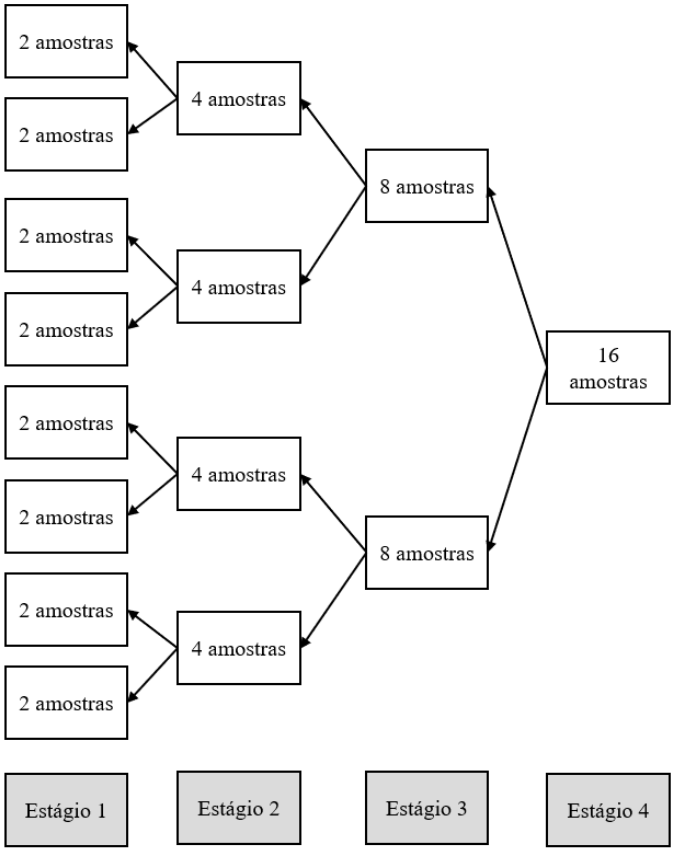
No método de Goertzel, não se precisa avaliar todos os diferentes valores de $X[k]$. De fato, pode-se, em geral, avaliar $X[k]$ para quaisquer M valores de k. Neste caso, a complexidade computacional é proporcional a NM. Para valores maiores de M, o algoritmo da FFT é mais eficiente em termos computacionais.

3.4. Transformada Rápida de Fourier

Assim como o algoritmo de Goertzel, a FFT também explora a simetria que existe nas exponenciais complexas. Todavia, diferente do algoritmo de Goertzel, a FFT obtém todos os pontos da DFT e isso com uma complexidade de $O(N \log_2 N)$. Seu uso pode não parecer tão significativo para sequência de tamanho N significativamente pequena, mas para sequencias bastante longas, a FFT se demostra muito eficiente para obtenção da DFT (OPPENHEIM; SCHAFER; BUCK, 1998).

A FFT pode ser implementada de diferentes formas, mas nesse texto utiliza-se o método chamado de dizimação no tempo. No caso, uma sequência de largura N, onde N é um valor de potência de 2, é dividida em duas sequencias de tamanho $N/2$. Isso é feito diversas vezes até que seja obtido sequencias de largura de 2 pontos. Desta forma, a FFT pode ser entendida como um cálculo em estágios, conforme ilustrado na Figura 1, em que em cada estágio ocorre a dizimação da sequência em duas sequências de tamanho igual à metade do estágio posterior. A dizimação encerra quando as sequências têm 2 pontos, e portanto, o estágio 1 da FFT corresponde a DFTs de tamanho 2.

Figura 1 - Exemplo de dizimação para uma FFT com 16 amostras.



Fonte: Autoria própria.

As DFT em dois pontos são calculadas de forma óbvia como (OPPENHEIM; SCHAFER; 1975):

$$\begin{aligned} V[0] &= v[0] + v[1] \\ V[1] &= v[0] - v[1] \end{aligned} \tag{11}$$

o resultado da DFT em dois pontos é utilizado para determinar as DFT do estágio seguinte, e assim por diante. Desta forma, as DFT calculadas no estágio *m* são utilizadas para calcular a DFT no estágio *m*+1.

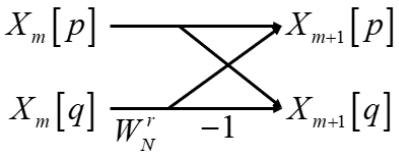
Demonstra-se que as DFT do estágio *m*+1 podem ser determinadas a partir das DFT do estágio *m* como (OPPENHEIM; SCHAFER; BUCK, 1998):

$$\begin{aligned} X_{m+1}[p] &= X_m[p] + W_N^r X_m[q] \\ X_{m+1}[q] &= X_m[p] - W_N^r X_m[q] \end{aligned} \tag{12}$$

em que *p*, *q* e *r* são valores que dependem do estágio e *N* é a quantidade de amostras da DFT.

Analisando-se a Equação (12), pode-se elaborar uma estrutura para a computação da FFT entre os estágios. Essa estrutura é denominada de borboleta, Figura 2.

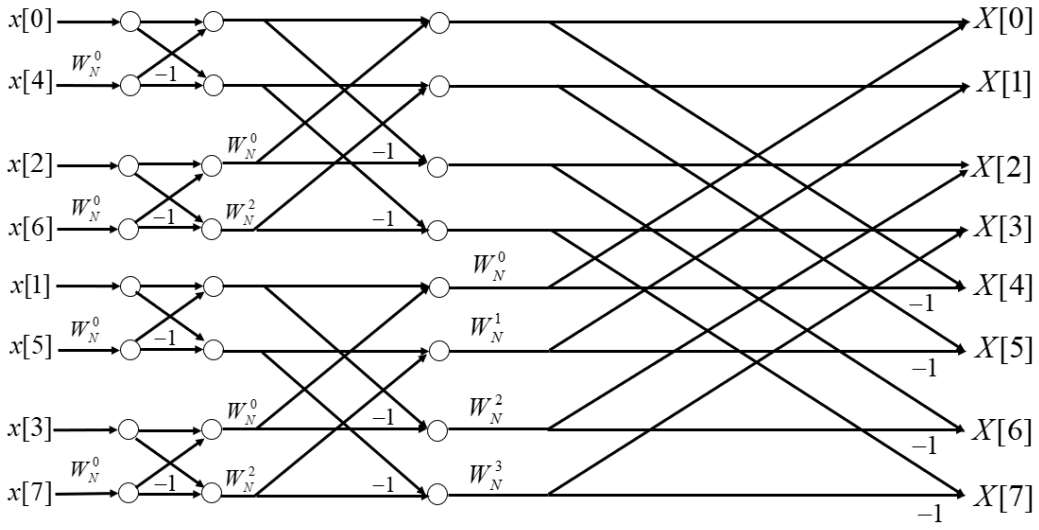
Figura 2 - Diagrama de borboleta da FFT



Fonte: Autoria Própria.

O intuito deste trabalho é realizar a implementação de uma FFT para 8 pontos, logo um diagrama maior é necessário para descrever todo o fluxo. A Figura 2, baseada do livro do Oppenheim & Schafer (1975) é uma ótima ilustração do fluxo da FFT que se planeja implementar a nível de *hardware*.

Figura 3 - Fluxo da FFT para uma sequência de 8 pontos



Fonte: Autoria própria. Baseado em Oppenheim & Schaffer (1975).

Como pode ser notado da Figura 2, há um total de 12 operações de borboletas descritas na Figura 1, todos com valores de entrada e de *r* diferentes. Outro ponto importante de salientar é que a entrada é permutada, essa permutação é chamada de *bit-reversal permutation* (OPPENHEIM; SCHAFER; BUCK, 1998).

Para realizar a FFT inversa, é possível utilizar a mesma borboleta apresentado na Figura 2. No entanto, algumas alterações devem ser feitas. Primeiramente, ao invés de utilizar o fator de escala W_N^r, deve-se utilizar o seu inverso, W_N^{-r}. Além disso, as entradas do algoritmo devem ser os valores da DFT de uma sequência. O processo de rearranjo de bits (*bit-reversal permutation*) deve ser mantido para que a saída seja a sequência original. Por fim, todas as saídas devem ser normalizadas por um fator de 1/*N*, onde *N* é o número total de amostras, que neste caso é igual a 8.

Em tópicos seguintes deste trabalho será trabalhado melhor como foi realizado a implementação destes algoritmos em *Verilog*.

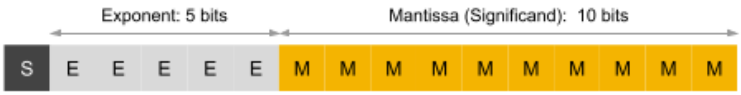
3.5. Ponto Flutuante

A representação em ponto flutuante fornece uma extensiva gama de formas de representar um número real através da notação cientifica em binário e tem o seu uso em diversas aplicações de computação cientifica, grande parte sendo em processamento digital de sinal. O ponto flutuante tem como objetivo representar os números reais a partir de três características, sendo elas: O sinal do número, o expoente e a mantissa. A forma de ponto flutuante mais utilizada é o definido pela IEEE 754 (WOODS *et al*., 2008).

Esse trabalho vai se limitar a apenas apresentar o ponto flutuante de 16 bits, também conhecido como *half-precision floating-point format*, porém, a norma da IEEE 754 não se limita apenas a 16 *bits* (WOODS *et al*., 2008). O uso por 16 *bits* neste trabalho se dá pelo fato de ter uma menor complexidade em comparação aos outros formatos.

De acordo com Scott (1991), a representação com 16 *bits* é definida da seguinte forma: O *bit* mais significativo é o *bit* do sinal, logo após o *bit* do sinal, os próximos 5 *bits* dizem respeito ao expoente. O expoente pode variar de “00000” até “11111”, dentro deste intervalo são encontrados os números normalizados, os números não normalizados, as representações de zeros, as representações de infinito e os *Not a Number* (NaN). Os 10 *bits* menos significativos da formatação são a mantissa, é definida como sendo o número em binário que multiplica o dois elevado ao expoente da notação cientifica. Vale salientar que valores de expoente negativo na notação cientifica são representados na formatação em 16 *bits* como sendo um número positivo, pois o expoente negativo é somado por um *offset* que no caso da formatação de 16 *bits*, o *offset* é de 15. A Figura 4 apresenta uma imagem da representação informando a posição de cada componente da formatação.

Figura 4 - Representação de um número em ponto flutuante de 16 bits.



Fonte: Google Cloud (2021)

Scott (1991) continua dizer que expoente de “00000” é onde se encontra os números não normalizados e as representações de zero. Caso a mantissa seja completa por zeros, então o número que o formato está representando é o zero e o sinal vai indicar se é zero positivo ou negativo. Caso a mantissa tenha pelo menos um *bit* diferente de zero, então o número que o formato que está representando é o número não normalizado, dado que o número não possui o *bit* escondido igual 1 mas sim a 0 e seu expoente é sempre -14 em notação cientifica. Para o expoente no intervalo de “00001” até “11110” representam todos os números normalizados com seu expoente variando de -14 em “00001” até 15 em “11110”.

O expoente de “11111” é onde se encontra as duas representações de infinito. No caso, a mantissa é composta totalmente de zeros, e apenas o *bit* de sinal que diferencia o infinito negativo do infinito positivo. Para a mantissa com pelo menos um *bit* 1, temos a representação dos NaN. O NaN pode ter diversos significados dependendo da sua mantissa, mas para o escopo deste trabalho, o NaN é usando para representar o resultado de uma operação invalida com a sequência de bits “1 11111 1111111111” ou “0 11111 1111111111” (SCOTT, 1991).

Em ponto flutuante, existe a questão de arredondamento, para este trabalho, a forma de arredondamento escolhida foi de arredondar em direção ao número par, de acordo com Muller et al. (2010), está é o modo padrão de arredondamento definido pela norma IEEE 754.

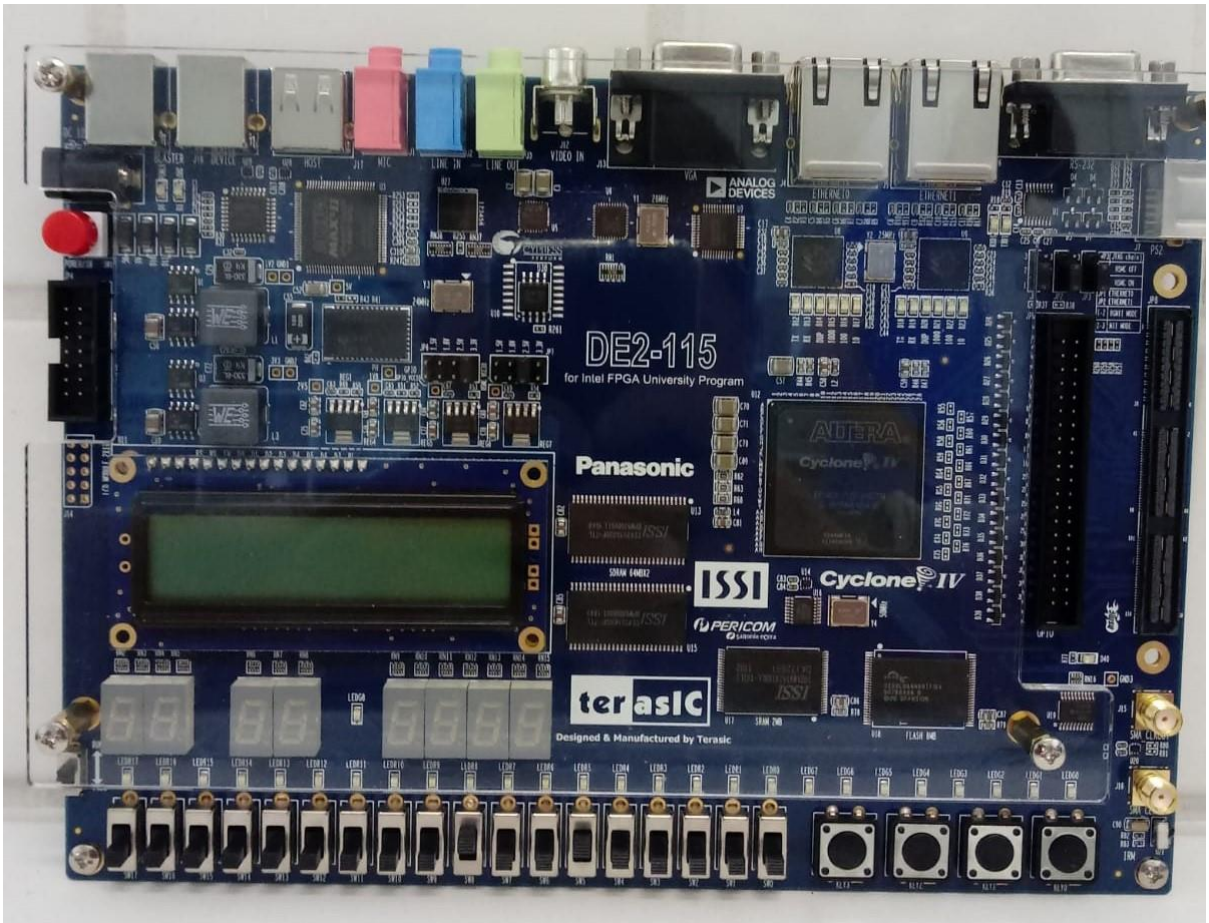
4. ARQUITETURA DO SISTEMA

Neste capítulo, é discorrido a respeito das implementações que foram feitas em *Verilog* para as operações aritméticas e para os algoritmos propostos.

4.1. Implementação em FPGA

Neste trabalho, as operações em ponto flutuante e os algoritmos de cálculo da DFT foram implementados em FPGA ilustrado na Figura 5. A FPGA é um tipo de dispositivo de circuito integrado reconfigurável que pode ser programado para executar funções específicas. As FPGA são diferentes dos *chips* de circuito integrado convencionais, que têm uma função específica e não podem ser reconfigurados depois de fabricados. São projetadas com uma matriz de blocos lógicos configuráveis e interconexões programáveis, permitindo que o designer configure o dispositivo para realizar a lógica desejada (WOODS et al., 2008).

Figura 5 - *Altera DE2-115 Board*



Fonte: Autoria própria.

A placa utilizada neste trabalho foi a *Altera DE2-115 Board* com a FPGA *Altera Cyclone® IV 4CE115* disponibilizada pela UFERSA. A utilização da FPGA neste trabalho foi para realizar a validação das operações aritmética implementadas utilizando ponto flutuante de 16 *bits*. Infelizmente, por limitações de tempo, não foi possível realizar a mesma validação a respeito dos algoritmos implementados, eles tiveram de ser validados apenas por simulações, porém utilizá-la para as operações aritméticas foi uma ótima forma de validar e encontrar comportamentos inesperados nas operações aritméticas que logo foram corrigidos.

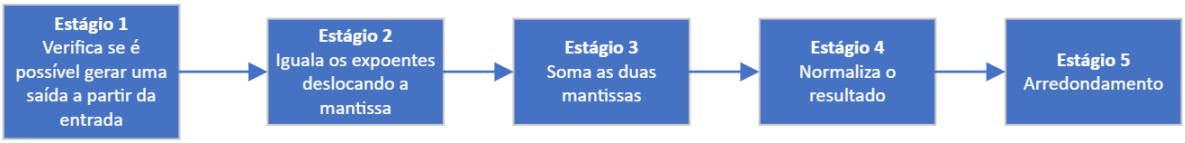
Esta placa contém diversos periféricos para utilizar em diversas aplicações, além da FPGA já mencionada anteriormente, a placa fornece memoria como 2MB SRAM, 2 64MB SDRAM, 8MB *flash memory* e SD *card socket*. São disponibilizados também 18 *switches* e 4 botões sem travas, 18 LEDs vermelhos e 9 LEDs verdes, um *clock* de 50MHz e diversos outros periféricos que não foram utilizados neste trabalho.

Devido utilizar uma FPGA, será necessária uma Linguagem de Descrição de *Hardware*. Neste trabalho, optou-se por utilizar *Verilog*, haja vista a sua versatilidade e padronização na indústria de semicondutores.

4.2. Operações de Adição e Subtração em Ponto Flutuante

As operações de adição e subtração foram implementadas utilizando a linguagem de descrição de *hardware*, *Verilog*. Estas operações, assim como as seguintes, são compostas por um conjunto de circuitos sequenciais em uma forma de *pipeline*, que realizam uma certa tarefa dependendo do estágio que se encontram no momento. Para a adição, assim como a subtração, a Figura 6 apresenta os cinco estágios utilizados para implementar ambas as operações.

Figura 6 - Fluxograma da operação de soma e subtração.



Fonte: Autoria própria.

Na Figura 6, o primeiro estágio tem a finalidade de gerar um resultado a partir da combinação das duas entradas. Um exemplo do primeiro estágio executando sua função é a

soma de um número normalizado com a representação de zero, neste caso, a saída já pode ser dita que será o número normalizado da entrada não importando qual ele seja (MULLER et al., 2010). Outros exemplos existem como, soma com representação de infinito é infinito exceto em casos em que a outra entrada é um NaN ou ambas são infinitas com sinais opostos, qualquer operação com NaN gera um novo NaN, dentre outras.

Uma outra vantagem que o estágio 1 traz é a tratamento de entradas para o estágio 2, pois representação de zeros, infinitos e NaN são todos tratadas no primeiro estágio. Com isso dito, é garantido dizer que se a operação chegar ao estágio 2, as duas entradas são dois números normalizados ou dois números não normalizados ou a combinação de um número normalizado e outro não normalizado.

O estágio 2 é aplicado quando os expoentes são diferentes. Assim como em notação cientifica, não é possível somar dois números que não tenham o mesmo expoente. Então no estágio 2 o número em ponto flutuante com o menor expoente tem sua mantissa deslocada para a direita até que o seu expoente se iguale com o do outro número em ponto flutuante e seja possível realizar a operação, de acordo com Muller et al. (2010) esse passo se chama *significand alignment*.

Neste estágio 2, assim como no estágio 1, tem uma forma de finalizar precocemente a operação para que não precise passar pelos estágios subsequentes. Isso ocorre quando é realizado mais de 11 deslocamentos no que resultaria em uma mantissa zerada de um dos números, assim a saída desta operação é o número com o maior expoente assim como ele entrou na operação.

O estágio 3 é bastante direto, dependendo de qual seja a operação e dos sinais dos valores de entrada, é realizada a soma ou a subtração das duas mantissas que agora tem expoentes iguais. O resultado é mandado para o estágio 4. Para o sinal do resultado da soma, em caso de que tenha ocorrido a soma das mantissas, é conservado o sinal durante a soma, em caso de que tenha ocorrido a subtração das mantissas, é conservado o sinal da entrada maior em magnitude, isso é notável pela entrada que tem o maior expoente.

O estágio 4 é um estágio comum para todas as operações aritméticas, após alguns estágios, o número em ponto flutuante resultante das operações pode ter sido modificado de uma forma que já não esteja no formato de IEEE 754, logo este estágio tem a responsabilidade gerar um valor que seja o número em formato de ponto flutuante assim como definido pelo padrão. Esse estágio realizar esta atividade deslocando a mantissa até que ela tenha apenas um bit 1 antes da virgula para o caso do resultado ser um número normalizado e para números não

normalizados, até que o expoente seja igual a -14 em decimal. Neste estágio também é possível pular o último estágio caso ocorra um *overflow* durante a normalização.

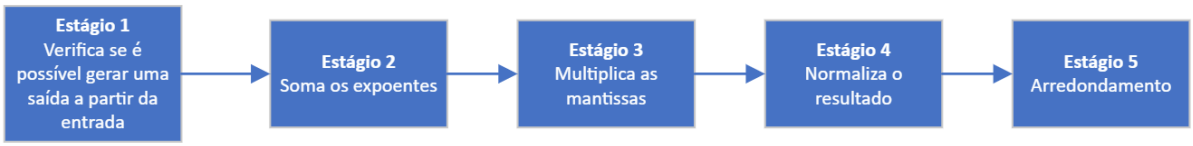
Para o último estágio, estágio 5, ele tem a ênfase de realizar o a arredondamento do número normalizado no estágio 4. Como já mencionado no capítulo anterior, a forma de arredondamento é arredondar em direção ao número par, todas as regras sobre esse tipo de arredondamento foram aplicadas para este estágio.

Ao fim do estágio 5, é obtido o resultado da operação. Para o caso da subtração, todos os estágios da Figura 6 se aplicam para a operação, porém o segundo valor na entrada desta operação tem o seu sinal trocado. Assim como dito por Muller et al. (2010), como podemos rescrever a subtração como $x + (-y)$, assim podemos ver as duas operações como uma operação só, a adição.

4.3. Operação de Multiplicação em Ponto Flutuante

Assim como a soma, a multiplicação foi desenvolvida utilizando estágios como já foi mencionado na seção anterior. A implementação da multiplicação recebe dois valores em ponto flutuante de entrada e na saída gera um outro ponto flutuante que é a multiplicação entre os dois pontos flutuantes de entrada. A Figura 7 apresenta os cinco estágios que foram necessários para implementar a lógica de *hardware* que consegue multiplicar os dois números.

Figura 7 - Fluxograma da operação de multiplicação



Fonte: Autoria própria.

O estágio 1 tem o objetivo igual ao estágio 1 da operação de adição e subtração, porém, aplicando-se regras diferentes. A exemplo da representação do zero multiplicado por um número normalizado qualquer que seja, na adição, o resultado é o número normalizado, na multiplicação, o resultado é a representação de zero (MULLER et al., 2010). Outros exemplos podem ser mencionados, de acordo com Muller et al. (2010), a representação de infinito multiplicado com qualquer outra representação, exceto a representação do zero e NaN, é a própria representação de infinito. Isso vale também para a representação de zero multiplicado

por qualquer outra representação, exceto a representação do infinito e NaN, é a própria representação de zero, para o caso destas exceções, o resultado da multiplicação é a o NaN. Caso já seja possível gerar um resultado a partir das entradas, a operação é finalizada de forma precoce e o resultado são as combinações mencionadas.

O estágio 2 tem a finalidade de realizar a soma dos expoentes, fazendo uma analogia à multiplicação entre dois números em notação cientifica, um dos primeiros passos é a soma dos seus expoentes. Vale salientar que um número em ponto flutuante tem o seu expoente real somado com um *offset*, logo quando os dois expoentes dos dois números em ponto flutuante são somados, deve-se realizar a subtração de um dos *offsets* para ter o valor do expoente do resultado da multiplicação com apenas um *offset*. A Equação (13) apresenta esse processo (MULLER et al., 2010). Na Equação (13) o e_r representa o expoente do resultado da soma de expoente, o e_1 e o e_2 são os valores do expoente da entrada 1 e 2 com o *offset* incluído e o *offset* é a constante inerente ao formato IEEE 754, para o caso atual, ponto flutuante de 16 *bits*, o seu valor é 15.

$$e_r = e_1 + e_2 - \text{offset} \tag{13}$$

Os números não normalizados têm o expoente com o valor binário de “00000”, porém, o expoente utilizado na Equação (13) deve ser o “00001”, já que o expoente real de um número não normalizado é o mesmo dos números normalizados com expoente “00001” no formato ponto flutuante. Além disso, dependendo do valor resultante da Equação (13), é possível finalizar a operação de forma precoce por ter ocorrido um *overflow* caso o expoente ultrapasse a maior expoente que o formato pode representar ou *underflow* caso o expoente tenha ficado com um valor muito pequeno que não pode ser representado nem mesmo com os números não normalizados.

O estágio 3 realiza a multiplicação entre a mantissa da entrada 1 e 2 e seus respectivos *bits* escondidos, que para o caso de um número normalizado, o *bit* escondido é de 1 e para o caso de um número não normalizado, o *bit* escondido é de 0. De acordo com Muller et al. (2010), a multiplicação de uma cadeia de *bits* de largura *L* por outra cadeia de largura *L* gera um resultado de largura 2*L*. Logo para essa multiplicação que envolve os 10 *bits* da mantissa e mais 1 *bit* escondido, formando uma cadeia de 11 *bits* para cada entrada, o resultado obtido é uma cadeia de 22 *bits*.

O estágio 4 realiza a normalização dos resultados de expoente e mantissa obtidos no estágio 2 e 3. Assim como a adição e subtração, a mantissa é deslocada até que ela tenha apenas 1 *bit* à esquerda virgula para o caso de um resultado normalizado ou até que o expoente real seja igual ao de -14, no qual significa que o número é não normalizado caso ele não tenha *bit* escondido. Vale salientar que neste estágio pode ocorrer *overflow* e *underflow* caso o expoente fique igual ou maior que “11111” ao final da normalização para um número normalizado ou a mantissa seja preenchida de zeros para o caso de um número não normalizado.

O estágio 5 tem o fim de realizar o arredondamento da mantissa assim como na operação de adição e subtração utilizando o arredondamento escolhido para este trabalho.

Com relação ao sinal do ponto flutuante resultante desta operação, é apenas necessário realizar a porta lógica XOR entre o sinal das duas entradas (MULLER et al., 2010).

4.4. Operação de Divisão em Ponto Flutuante

De acordo com Overton (2001) a operação de divisão é a operação mais complexa a ser implementada. Além disso, o autor completa reforçando que é a operação que leva a maior quantidade de tempo para ser executada por completo. Para a implementação desta operação neste trabalho, foi decidido utilizar LUT já que o circuito para implementar a operação de forma direta não é viável na prática (OVERTON, 2001).

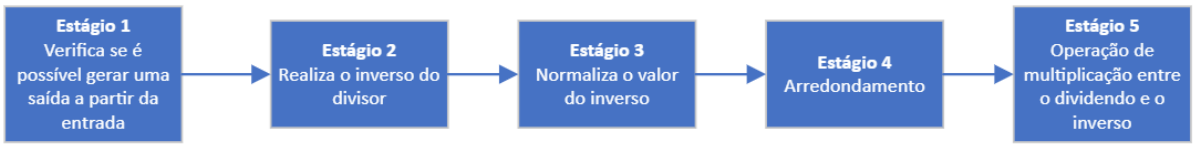
Para utilizar LUT, foi calculado que seria necessária uma quantidade de 4194304 entradas já existe 20 *bits* de mantissa e 2 *bits* escondidos de cada número em ponto flutuante da entrada. Essa quantidade de entrada é impraticável de ser implementado, então outra abordagem, ainda utilizando LUT foi executada.

A divisão pode ser realizada através do cálculo do inverso do divisor da divisão e utilizar esse inverso para multiplicar o dividendo. A Equação (14) apresenta a forma que a divisão foi implementada.

$$Q = x\frac{1}{y} \tag{14}$$

Com isso em mão, a LUT para realizar o inverso do divisor teve exatamente 2048 entradas, fazendo que a sua implementação seja viável, porém ainda com uma quantidade considerável de entradas. A Figura 8 mostra o fluxograma seguido para ser possível ser realizado a implementação da divisão.

Figura 8 - Fluxograma da operação de divisão.



Fonte: Autoria própria

Assim como todas as operações apresentadas até o momento, o estágio 1 sempre foi utilizando como uma forma de tratamento, visando finalizar a execução da operação antecipadamente por já poder entregar um resultado a partir das entradas. Além disso, a execução continua para o estágio 2 se e somente se as entradas forem números em ponto flutuante normalizado ou não normalizado, quaisquer outras combinações de entrada já possuem informação suficiente para decidir a saída sem precisar entrar em outros estágios da divisão.

Alguns exemplos de combinações de entrada podem ser apresentados na qual fariam a execução ser finalizada de forma precoce. De acordo com Muller et al. (2010), elas são: dividendo sendo um número normalizado ou não normalizado e o divisor ser a representação de zero - isso geraria uma saída sendo a representação de infinito; o dividendo ser a representação de um número normalizado ou não normalizado e o divisor ser a representação de infinito - isso geraria a representação de zero. Operações invalidas também podem ser mencionadas como: ambas entradas serem a representação de infinito ou ambas entradas serem a representação de zero, na qual isso geraria um NaN.

O estágio 2 visa calcular o inverso do divisor da operação, que é executada através da consulta na LUT para realizar o inverso para a mantissa e realizar um cálculo para também obter o inverso do expoente. Na questão do inverso da mantissa, é consultada a LUT e esta consulta é necessária de 11 *bits*: 10 da mantissa e 1 *bit* escondido, já que as representações neste estágio podem ser de um número normalizado ou um número não normalizado. A segunda tarefa a ser realizada é de realizar o inverso do expoente do divisor, neste caso é mais simples já que é apenas necessário trocar o sinal do expoente. Como na representação em ponto flutuante existe a questão do *offset*, então esse fator também teve de ser levado em consideração ao realizar o inverso do expoente. A Equação (15) apresenta o cálculo para se obter o inverso do expoente, onde e_o é o expoente do divisor antes da de ser calculado o inverso, *offset* é a

constante inerente ao formato, que no caso deste trabalho é de 15 e o e_i é o expoente inverso que se deseja obter.

$$e_i = 2(\text{offset}) - e_0 \tag{15}$$

Dependendo do número não normalizado, o seu inverso poderia causar um *overflow* na operação e diferente dos outros estágios das outras operações, que quando existia a ocorrência de *overflow* já era a indicação que a operação poderia ser finalizada, neste caso, apenas indica-se que o inverso não pode ser representado por outro valor que não seja a representação de infinito. Para evitar isso, o inverso de um número que causa esse comportamento é representado com o expoente de "11110", que é o expoente do número muito próximo de um *overflow* e utilizando a propriedade da associatividade da multiplicação, todo o outro expoente que deveria ter ido para o inverso é inserido no expoente do dividendo. Caso o dividendo também ocorra um *overflow* por causa do aumento em seu expoente, então isso é prova suficiente de que a operação em si é para ser um *overflow* e o execução pode ser interrompida tendo a representação de infinito na saída.

O estágio 3 tem a finalidade de realizar a normalização do inverso do divisor para que o inverso fique no formato de ponto flutuante de 16 *bits*. Logo após isso é realizado o arredondamento no estágio 4.

O estágio 5 é onde ocorre a multiplicação do dividendo com o inverso do divisor e todos os passos desta multiplicação foram descritos no tópico onde é apresentado sobre a operação de multiplicação. Assim como a operação de multiplicação o sinal do número em ponto flutuante na saída é igual ao realizar a porta logica XOR entre o sinal do dividendo e divisor.

4.5. Implementação do Algoritmo de Goertzel

A implementação deste algoritmo foi através de um pseudocódigo apresentado por Sysel e Rajmic (2012) na qual é demostrado de forma simplificada o algoritmo a ser implementado. Além disso, dentro desta implementação vai existir chamada para as operações aritméticas de adição, subtração e multiplicação. O Quadro 1 apresenta o pseudocódigo para a implementação do algoritmo de Goertzel.

Quadro 1 - Pseudocódigo para a implementação de algoritmo de Goertzel.

```
algoritmo Goertzel2(x,N,k)
     vk_m1 ← 0
     vk_m2 ← 0
     Const ← 2*cos(2*pi*k/N)
     Wn ← exp(-j*2*pi/N*k)
     para n de 0 até N-1 faça
          vk ← x[n] + Const*vk_m1 – vk_m2;
          vk_m2 ← vk_m1
          vk_m1 ← vk
     fim-para
     vk ← Const*vk_m1 – vk_m2
     Xk ← vk – Wn*vk_m1
     retorne Xk
fim-algoritmo
```

Fonte: Sysel e Rajmic (2012)

A implementação do algoritmo foi feita com um circuito sequencial, em que o circuito recebe 8 entradas em ponto flutuante de 16 *bits*, que representam a sequência no qual deseja-se calcular a DFT, e mais uma cadeia de 3 *bits*, que informa qual a frequência que se deseja a DFT. A saída são duas cadeias de 16 *bits* cada, onde uma representa a parte real da DFT e a segunda representa a parte imaginaria. Porém, algumas adaptações foram necessárias e serão elencadas a seguir.

O primeiro ponto é em relação às constantes que estão presentes na implementação. A primeira constante apresentada no Quadro 1 é `Const`, para qual foi criado uma LUT com todos os valores de *k* variando de 0 até 7 para uma rápida consulta destas possíveis constantes. A outra constante a se considerar está na variável `Wn` que diz respeito a exponencial $e^{-j2\pi\frac{k}{N}}$. Utilizando a fórmula de Euler, é possível obter duas constantes na qual fazem parte do cálculo final que calcula a parte real a parte imaginaria da saída, conforme apresentado na Equação (16). Dessa forma, foram feitas mais duas LUT para as constantes $\cos(2\pi\frac{k}{N})$ e $\sin(2\pi\frac{k}{N})$ com ambos os valores de *k* variando 0 até 7.

$$W_N^k = \cos\left(\frac{2\pi k}{N}\right) + j\sin\left(\frac{2\pi k}{N}\right) \tag{16}$$

O algoritmo apresentado no Quadro 1 é executado de forma recursiva. Ao sair do laço, o resultado pode ser calculado utilizando a Equação (17) e (18) para calcular, respectivamente, a parte real e imaginária respectivamente onde *k* é o que índica a frequência escolhida e *N* é o número total de pontos.

$$X_R[k] = \mathrm{Re}\{X[k]\} = v_k[N] - \cos\left(\frac{2\pi k}{N}\right) v_k[N-1] \tag{17}$$

$$X_I[k] = \mathrm{Im}\{X[k]\} = -\sin\left(\frac{2\pi k}{N}\right) v_k[N-1] \tag{18}$$

4.6. FFT e FFT inversa

Diferentemente do algoritmo de Goertzel, que recebe apenas sequências reais, na implementação da FFT optou-se por abranger sequências complexas de entrada. Dessa forma, o algoritmo recebe uma sequência complexa com oito amostras, sendo que cada amostra é representada por dois valores, um valor real e outro valor complexo. Dessa forma, ambos os algoritmos têm 16 entradas. A saída também possui 16 saídas que correspondem a 8 amostras complexas, cada amostra possui 2 valores para representar a parte real e a parte imaginária da DFT.

Quadro 2 - Pseudocódigo para a implementação da FFT.

```
algoritmo FFT(x,N)
     se N == 1 então
          retorne x;
     xp ← x (index pares);
     xi ← x (index ímpares);
     yp ← FFT(xp, N/2);
     yi ← FFT(xi, N/2);
     y ← vetor[0:N-1];
     para n de 0 até (N-1)/2 faça
          y[n] ← yp[n] + (Wn)^n*yi[n];
          y[n + N/2] ← yp[n] - (Wn)^n*yi[n];
     fim-para
```

```
    retorne y;
fim-algoritmo
```

Fonte: Oppenheim, Schafer e Buck (1998)

Para a sua implementação, foi escrito a descrição de hardware para o algoritmo utilizando *Verilog* com o método *in place*. O circuito produzido foi um circuito sequencial, que a cada certo momento, realizava umas das operações em formato de borboleta que foi apresentada pela Figura 2.

Assim como apresentado no algoritmo de Goertzel, algumas constantes tiveram LUT criadas para facilitar o acesso a seu valor, para o caso da FFT, a constante se encontra no cálculo do diagrama de borboleta, são as exponenciais complexas. Utilizando a fórmula de Euler, pode-se separar a exponencial complexas em funções seno e cosseno que são armazenadas em duas LUTs.

A partir da Equação (12), os valores da saída das borboletas podem ser calculados separando a exponencial complexa em seno e cosseno e dividindo as saídas em real e imaginário tais como apresentadas na Equação (19).

$$\begin{aligned}
\mathrm{Re}\{X_{m+1}[p]\} &= \mathrm{Re}\{X_m[p]\} + \mathrm{Re}\{X_m[q]\}\cos\left(\frac{2\pi k}{N}\right) + \mathrm{Im}\{X_m[q]\}\sin\left(\frac{2\pi k}{N}\right) \\
\mathrm{Im}\{X_{m+1}[p]\} &= \mathrm{Im}\{X_m[p]\} + \mathrm{Im}\{X_m[q]\}\cos\left(\frac{2\pi k}{N}\right) - \mathrm{Re}\{X_m[q]\}\sin\left(\frac{2\pi k}{N}\right) \\
\mathrm{Re}\{X_{m+1}[q]\} &= \mathrm{Re}\{X_m[p]\} - \mathrm{Re}\{X_m[q]\}\cos\left(\frac{2\pi k}{N}\right) - \mathrm{Im}\{X_m[q]\}\sin\left(\frac{2\pi k}{N}\right) \\
\mathrm{Im}\{X_{m+1}[q]\} &= \mathrm{Im}\{X_m[p]\} - \mathrm{Im}\{X_m[q]\}\cos\left(\frac{2\pi k}{N}\right) + \mathrm{Re}\{X_m[q]\}\sin\left(\frac{2\pi k}{N}\right)
\end{aligned} \tag{19}$$

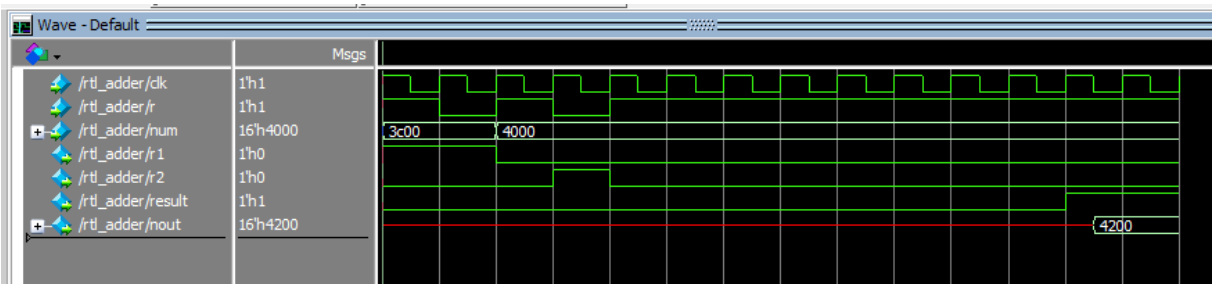
Para a FFT inversa, todos os passos são os mesmos feitos pela FFT, os únicos pontos que deve ser alterado é em relação à Equação (19) que devem ser modificadas. Primeiramente, ao invés de utilizar o fator de escala W_N^r, deve-se utilizar o seu inverso, W_N^{-r}. Por fim, todas as saídas devem ser normalizadas por um fator de $1/N$, onde N é o número total de amostras, que neste caso é igual a 8. Neste trabalho, a alteração foi feita para que não seja necessário realizar trocar a estrutura de borboleta, resultando na Equação (20).

$$
\begin{aligned}
\mathrm{Re}\{x_{m+1}[p]\} &= \frac{1}{2}\left[\mathrm{Re}\{x_m[p]\} + \mathrm{Re}\{x_m[q]\}\cos\left(\frac{2\pi k}{N}\right) + \mathrm{Im}\{x_m[q]\}\sin\left(-\frac{2\pi k}{N}\right)\right] \\
\mathrm{Im}\{x_{m+1}[p]\} &= \frac{1}{2}\left[\mathrm{Im}\{x_m[p]\} + \mathrm{Im}\{x_m[q]\}\cos\left(\frac{2\pi k}{N}\right) - \mathrm{Re}\{X_m[q]\}\sin\left(-\frac{2\pi k}{N}\right)\right] \\
\mathrm{Re}\{x_{m+1}[q]\} &= \frac{1}{2}\left[\mathrm{Re}\{x_m[p]\} - \mathrm{Re}\{x_m[q]\}\cos\left(\frac{2\pi k}{N}\right) - \mathrm{Im}\{x_m[q]\}\sin\left(-\frac{2\pi k}{N}\right)\right] \\
\mathrm{Im}\{x_{m+1}[q]\} &= \frac{1}{2}\left[\mathrm{Im}\{x_m[p]\} - \mathrm{Im}\{x_m[q]\}\cos\left(\frac{2\pi k}{N}\right) + \mathrm{Re}\{x_m[q]\}\sin\left(-\frac{2\pi k}{N}\right)\right]
\end{aligned}
\tag{20}
$$

5. RESULTADOS

Neste capítulo, será discutido sobre os resultados dos testes feitos sobre as operações aritméticas individualmente e após isso é apresentados resultados feitos utilizando os algoritmos propostos para cálculo da DFT. A Figura 9 ilustra um exemplo de forma de onda do resultado da soma de 1 mais 2 que no resultado deu 3.

Figura 9 - Exemplo em forma de onda



Fonte: Autoria própria

As operações matemáticas foram testadas em FPGA. Para cada operação, foi realizado 16 diferentes expressões aritméticas, os resultados foram comparados com uma ferramenta disponível na internet que realiza operações aritméticas com ponto flutuante de 16 *bits* e contém o mesmo tipo de arredondamento deste trabalho. Os resultados estão expostos nas seguintes Figuras: Na Figura 10 apresenta os resultados da adição, na Figura 11 apresenta os resultados da subtração, na Figura 12 apresenta os resultados da multiplicação e pôr fim a Figura 13 apresenta o resultado da divisão.

Figura 10 - Resultados para a adição.

EXPECTATIVA	RESULTADO
31C0	31C0
3CB1	3CB1
F9E0	F9E0
F33F	F33F
498D	498D
E330	E330
CE3C	CE3C
F557	F557
E996	E996
682A	682A
1FA3	1FA3
FFFF	FFFF
7C00	7C00
0002	0002
FFFF	FFFF
0000	0000

Fonte: Autoria própria

Figura 11 - Resultados para a subtração.

EXPECTATIVA	RESULTADO
ABEA	ABEA
BB62	BB62
79E0	79E0
733F	733F
498F	498F
6330	6330
CE3C	CE3C
7557	7557
6998	6998
6828	6828
201D	201D
FFFF	FFFF
7C00	7C00
0000	0000
7C00	7C00
0000	0000

Fonte: Autoria própria

Figura 12 - Resultados para a multiplicação.

EXPECTATIVA	RESULTADO
1F4B	1F4B
302F	302F
7C00	7C00
6559	6559
AA94	AA94
BCA0	BCA0
1440	1440
FC00	FC00
ED4D	ED4D
6DE3	6DE3
8026	8026
FFFF	FFFF
FFFF	7FFF
0000	0000
FC00	FC00
8000	8000

Fonte: Autoria própria

Figura 13 - Resultados para a divisão.

EXPECTATIVA	RESULTADO
37CF	37D0
2FA0	2F9F
08D4	08D5
0068	0068
E8B0	E8B1
8017	8017
7C00	7C00
894C	894C
916E	916E
61E1	61E1
CEB3	CEB3
FFFF	FFFF
7C00	7C00
3C00	3C00
FFFF	FFFF
FFFF	FFFF

Fonte: Autoria própria

Como pode ser visto pelas Figuras 10, 12, 13 e 14, as operações de adição, subtração e multiplicação tiveram um acerto de 100%, porém, a divisão teve um acerto de 75%. O motivo na qual pode ter influenciado esse resultado foi a forma de realizar a divisão, que não foi feita de forma direta, utilizando o valor inverso do divisor e depois uma multiplicação com o dividendo. Vale salientar também que os erros cometidos pela divisão só estão errados por apenas um *bit* de diferença e esse *bit* é o menos significativo.

Para testar o algoritmo de Goertzel e os algoritmos relacionados com a FFT foi primeiro obtido duas sequências de 8 números em ponto flutuante amostrados de uma distribuição

normal de média 0 e desvio padrão de 1. Os valores podem ser consultados no Quadro 3 que apresenta as duas sequencias em decimal e sua representação em ponto flutuante em hexadecimal.

Quadro 3 - Sequências utilizadas para produzir resultados.

	Sequência 1		Sequência 2	
	Decimal	Ponto flutuante (Hexadecimal)	Decimal	Ponto flutuante (Hexadecimal)
x[0]	2,3	409A	-2,906	C1D0
x[1]	-1,343	BD5F	0,3503	359B
x[2]	-1,198	BCCB	2,027	400E
x[3]	-0,9053	BB3E	0,665	3952
x[4]	0,3508	359D	0,5054	380B
x[5]	-0,4297	B6E0	0,3896	363C
x[6]	0,9663	3BBB	-1,797	BF30
x[7]	-0,367	B5DF	1,095	3C61

Fonte: Autoria própria

Como uma forma de comparação, foi utilizado a linguagem de programação Python em conjunto com a biblioteca NumPy para realizar a DFT das duas sequências presente no Quadro 3. Um ponto negativo desta abordagem é que a Biblioteca NumPy não trabalha com número em ponto flutuante de 16 bits, logo, foi feito a DFT das sequências em ponto flutuante de 64 bits e depois convertido em 16 bits. O Quadro 4 apresenta a DFT que será usada como referência daqui para frente.

Quadro 4 - DFT das sequências utilizadas para os resultados utilizando a biblioteca NumPy do Python.

	Sequência 1 (DFT)			Sequência 2 (DFT)		
	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)
X[0]	-0,625	B900	0000	0,3294	3545	0000
X[1]	1,686+j3,19	3EBE	4261	-3,135-j3,493	C245	C2FC
X[2]	2,885+j0,5	41C5	3800	-2,631+j1,02	C143	3C14
X[3]	2,215-j1,139	406E	BC8E	-3,688+j4,16	C360	4428
X[4]	5,465	4577	0000	-4,672	C4AC	0000
X[5]	2,215+j1,139	406E	3C8E	-3,688-j4,16	C360	C428

X[6]	2,885-j0,5	41C5	B800	-2,631-j1,02	C143	BC14
X[7]	1,686-j3,19	3EBE	C261	-3,135+j3,493	C245	42FC

Fonte: Autoria própria

O primeiro algoritmo a ser testado foi o algoritmo de Goertzel. O algoritmo foi executado 16 vezes para que pudesse obter cada valor da DFT para cada sequência. Os resultados para o algoritmo de Goertzel estão dispostos no Quadro 5.

Quadro 5 - Resultados do algoritmo de Goertzel.

	Sequência 1 (DFT)			Sequência 2 (DFT)		
	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)
X[0]	-0,628	B906	0000	0,3282	3540	0000
X[1]	1,678+j3,19	3EB6	4261	-3,135-j3,491	C245	C2FB
X[2]	2,885+j0,5	41C5	3800	-2,633+j1,02	C144	3C14
X[3]	2,215-j1,142	406E	BC91	-3.688+j4,16	C360	4428
X[4]	5.47	4577	0000	-4.67	C4AB	0000
X[5]	2,215+j1,142	406E	3C91	-3.688-j4,16	C360	C428
X[6]	2,885-j0,5	41C5	B800	-2,633-j1,02	C144	BC14
X[7]	1,678-j3,19	3EB6	C261	-3,135+j3,491	C245	42FB

Fonte: Autoria própria

Se compararmos o Quadro 3 com o Quadro 4, veremos que a DFT produzida pela referência e a DFT produzida pelo algoritmo de Goertzel são muito semelhantes na maioria dos pontos. O cálculo de erro absoluto em relação a parte decimal entre o algoritmo e a referência tem erros que chegam apenas até terceira casa decimal de 0,008, caracterizando bons resultados tendo em vista que o formato ponto flutuante de 16 *bits* é bastante limitado em suas representações.

O segundo algoritmo que foi testado foi a FFT das sequências 1 e 2 presentes no Quadro 3. O intuído deste teste é verificar se a DFT realizada pelo algoritmo FFT é capaz de gerar as mesmas ou próximos valores presentes na Quadro 4. O Quadro 6 apresenta os resultados para o algoritmo FFT.

Quadro 6 - Resultados da FFT para cada sequência

	Sequência 1 (DFT)			Sequência 2 (DFT)		
	Decimal	Ponto flutuante	Ponto flutuante	Decimal	Ponto flutuante	Ponto flutuante

		Parte real (Hexadecimal)	Parte Imaginária (Hexadecimal)		Parte real (Hexadecimal)	Parte Imaginária (Hexadecimal)
X[0]	-0,625	B900	0000	0,3301	3548	0000
X[1]	1,686+j3,19	3EBE	4261	-3,137-j3,493	C246	C2FC
X[2]	2,885+j0,5	41C5	3800	-2,631+j1,02	C143	3C14
X[3]	2,215-j1,139	406E	BC8E	-3,688+j4,16	C360	4428
X[4]	5.465	4577	0000	-4.672	C4AC	0000
X[5]	2,215+j1,139	406E	3C8E	-3,688-j4,16	C360	C428
X[6]	2,885-j0,5	41C5	B800	-2,631-j1,02	C143	BC14
X[7]	1,686-j 3,19	3EBE	C261	-3,137+j3,493	C246	42FC

Fonte: Autoria própria

Comparando o Quadro 4 e Quadro 6 é notável que o algoritmo FFT obteve resultado melhores em relação ao algoritmo de Goertzel. Apenas na sequência 2 que houve valores que divergiram se comparados aos valores da referência e essa divergência não teve um erro absoluto maior que a terceira casa decimal. Isso mostra que a FFT implementada mostrou bastante robustez na produção de seus resultados.

Ainda no algoritmo implementado da FFT, como este algoritmo foi implementado de forma genérica na qual é possível inserir na entrada valor complexos, pode-se utilizar uma propriedade da DFT para calcular a DFT de duas sequências reais ao mesmo tempo. Sendo a entrada $x[n] = x_1[n] + jx_2[n]$ com DFT $X[k]$, é possível obter a DFT da sequência $x_1[n]$ e da sequência $x_2[n]$ através das Equações (21) e (22) (OPPENHEIM; SCHAFER; BUCK, 1998).

$$X_1[k] = \frac{X^*[k] + X\left[\left((N+k)\right)_N\right]}{2} \tag{21}$$

$$X_2[k] = -j\,\frac{X^*[k] - X\left[\left((N+k)\right)_N\right]}{2} \tag{22}$$

em que $X\left[\left((N+k)\right)_N\right] = X\left[(N+k) \text{ módulo } N\right]$.

Realizando a DFT utilizando a FFT implementada da entrada combinada da sequência 1 e 2 foi obtido os valores no Quadro 7.

Quadro 7 - DFT da sequência 1 e 2 combinadas

	Sequência 1 e 2 (DFT)		
	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)
X[0]	-0.625+j0.3294	B900	3545
X[1]	5.176+j0.0552	452D	2B10
X[2]	1.864-j2.131	3F74	C043
X[3]	-1.941-j4.825	BFC3	C4D3
X[4]	5.465-j4.67	4577	C4AB
X[5]	6.372-j2.55	465F	C119
X[6]	3.903-j3.131	43CE	C243
X[7]	-1.808-j6.33	BF3B	C654

Fonte: Autoria própria

Utilizando as Equações (21) e (22) sobre os valores do Quadro 7, foi obtido os seguintes valores de DFT para a sequência 1 e 2 que estão dispostos no Quadro 8.

Quadro 8 - DFT das duas sequências obtidas a partir do uso da propriedade de simetria da DFT entre as duas sequências.

	Sequência 1 (DFT)			Sequência 2 (DFT)		
	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)	Decimal	Ponto flutuante Parte real (Hexadecimal)	Ponto flutuante Parte Imaginária (Hexadecimal)
X[0]	-0.625	B900	0000	0,3301	3548	0000
X[1]	1.684+j3.192	3EBC	4262	-3,137-j3,493	C246	C2FC
X[2]	2,885+j0,5	41C5	3800	-2,631+j1,02	C143	3C14
X[3]	2,215-j1,139	406E	BC8E	-3.69+j4,16	C361	4428
X[4]	5.465	4577	0000	-4.672	C4AC	0000
X[5]	2,215+j1,139	406E	3C8E	-3.69-j4,16	C361	C428
X[6]	2,885-j0,5	41C5	B800	-2,631-j1,02	C143	BC14
X[7]	1.684-j3.192	3EBC	C262	-3,137+j3,493	C246	42FC

Fonte: Autoria própria

Realizando uma comparação entre o Quadro 8 com o Quadro 4, é visto que assim como o cálculo da DFT utilizando a FFT de forma direta, o resultado gerado é uma DFT bastante semelhante ao da referência.

O último resultado que foi produzido foi a utilização da FFT inversa nos valores encontrados no Quadro 6 com a expectativa de gerar os valores no qual foi iniciado esse capítulo

que estão dispostos no Quadro 3. No Quadro 9 são apresentados todos estes resultados da FFT inversa, vale salientar de que foi omitido a colunas para valores imaginários já que o algoritmo implementado gerou representações de zeros para essas saídas.

Quadro 9 - Sequências 1 e 2 obtidas a partir da FFT inversa.

	Sequência 1		Sequência 2	
	Decimal	Ponto flutuante (Hexadecimal)	Decimal	Ponto flutuante (Hexadecimal)
x[0]	2,3	409A	-2,906	C1D0
x[1]	-1.342	BD5E	0.3501	359A
x[2]	-1,198	BCCB	2,027	400E
x[3]	-0,9053	BB3E	0,665	3952
x[4]	0.3511	359E	0.505	380A
x[5]	-0.4302	B6E2	0.3902	363E
x[6]	0.966	3BBA	-1,797	BF30
x[7]	-0.3672	B5E0	1,095	3C61

Fonte: Autoria própria

O Quadro 9 mostra os resultados obtidos da FFT inversa. Em uma rápida comparação entra os resultados do Quadro 9 e os valores escolhidos para entrar para os resultados que estão no Quadro 3, é visto o mesmo comportamento que vem sendo observado até o momento de que os resultados estão sendo produzidos no limite das representações em ponto flutuante e com uma quantidade insignificante de erros que não são suficientes para invalidar os resultados obtidos até então.

Outra causa para a ocorrência destes erros pode ser vista no livro de Goldberg (1991), no qual o autor enfatiza a existência de erros de arredondamento. Ao realizar uma sequência de operações aritméticas de ponto flutuante, o acúmulo de erros de arredondamento pode levar à perda de precisão no resultado.

6. CONCLUSÕES

O propósito deste trabalho foi procurar soluções para possibilitar o uso de uma ferramenta mais robusta para realizar cálculos aritméticos simples a nível de *hardware* na qual a solução por ponto fixo já não era viável. Com isso em mente, foi proposto a implementação de uma unidade de ponto flutuante, na qual conteria todas as operações aritméticas em ponto flutuante de 16 *bits* e a sua eficácia seria testada através de alguns algoritmos de processamento digital de sinal.

O primeiro objetivo deste trabalho foi o estudo sobre o ponto flutuante normalizado pela IEEE 754 para ter um completo entendimento de como funciona e como é encapsulado diversas representações de números reais em apenas 16 bits. Com o completo conhecimento sobre o formato, os próximos passos foram em implementar as quatro operações básicas utilizando ponto flutuante na qual os resultados destas operações foram bastantes satisfatórias com exceção da divisão, na qual ela apresentou algumas divergências em seus resultados em comparação a uma mesma unidade de ponto flutuante com as mesmas especificações deste trabalho.

Além desta testes, foram realizados outros tipos de teste mais complexos, na qual o objetivo era de realizar cálculo utilizados na área de processamento digital de sinal a exemplo do algoritmo de Goertzel, FFT e FFT inversa. Nestes testes, foram utilizados sequencias aleatórias e foram gerados resultados com todos esses três algoritmos. Os resultados de todos os algoritmos foram diferentes em muitos pontos ao da referência, porém a ordem do erro não foi algo na qual invalidasse os resultados já que se mostraram bastante baixo.

As operações aritméticas implementadas se mostraram bastante robustas em todos os testes feitos até o momento. Para trabalhos futuros, é recomendado procurar uma melhor abordagem para a divisão, já que a abordagem utilizada neste trabalho contém certas operações na qual sua saída não foi a desejada mesmo errando apenas por 1 *bit* de diferença. Outro ponto que pode ser melhorado é na questão de otimização da FPU, utilizar mais algoritmos complexos para testar a sua eficácia e tentar implementar uma nova FPU com uma quantidade maior de *bits*, a exemplos do ponto flutuante de 32 *bits* e 64 *bits*.

REFERÊNCIAS

BARROIS, B.; SENTIEYS, O. Customizing fixed-point and floating-point arithmetic — A case study in K-means clustering. Disponível em: <https://ieeexplore.ieee.org/document/8109980>.

EWE, C. Te; CHEUNG, P. Y. K.; CONSTANTINIDES, G. A. Dual Fixed-Point: An Efficient Alternative to Floating-Point Computation. In: BECKER, J.; PLATZNER, M.; VERNALDE, S. (Eds.). Field Programmable Logic and Application. FPL 2004. Berlin, Heidelberg: Springer, 2004. Capítulo 22, p. 221-230. Lecture Notes in Computer Science, vol 3203. DOI: 10.1007/978-3-540-30117-2_22.

GOLDBERG, David. Numerical Computing with IEEE Floating Point Arithmetic. Philadelphia: Society for Industrial and Applied Mathematics, 1991.

HETTIARACHCHI, D. L. N.; DAVULURU, V. S. P.; BALSTER, E. J. Integer vs. Floating-Point Processing on Modern FPGA Technology. In: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). Las Vegas, USA, 2020. p. 1-6. DOI: 10.1109/CCWC47524.2020.9031118.

MULLER, J.-M., BRISEBARRE, N., de DINECHIN, F., JEANNEROD, C.-P., LEREVRE, V., MELQUIOND, G., REVOL, N., STEHLE, D.,TISON, S. Handbook of Floating-Point Arithmetic. Birkhäuser, 2010.

OPPENHEIM, Alan V.; Realization of digital filters using block-floating-point arithmetic. IEEE Transactions on Audio and Electroacoustics, v. 18, n. 2, p. 130–136, jun. 1970.

OPPENHEIM, Alan V.; SCHAFER, Ronald W. Discrete-Time Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

OPPENHEIM, Alan V.; SCHAFER, Ronald W. Discrete-Time Signal Processing: United States Edition. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1998.

OVERTON, Michael L. Numerical Computing with IEEE Floating Point Arithmetic. Philadelphia: Society for Industrial and Applied Mathematics, 2001.

PATTERSON, D.A.; HENNESSY, J.L. Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann, 1994.

SCOTT, Thomas J. Mathematics and Computer Science at Odds over Real Numbers. In: SIGCSE '91 Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education. New York: Association for Computing Machinery, 1991. p. 130-139. DOI: https://doi.org/10.1145/107004.107029.

SYSEL, P.; RAJMIC, P. Goertzel algorithm generalized to non-integer multiples of fundamental frequency. EURASIP Journal on Advances in Signal Processing, [s.l.], v. 2012, n. 1, p. 56, 2012. DOI: 10.1186/1687-6180-2012-56.

SWARTZLANDER, E. E.; SALEH, H. H. M. FFT Implementation with Fused Floating-Point Operations. IEEE Transactions on Computers, v. 61, n. 2, p. 284-288, Feb. 2012. DOI: 10.1109/TC.2010.271.

WOODS, Roger; McALLISTER, John; TURNER, Richard; YI, Ying; LIGHTBODY, Gaye. FPGA-based Implementation of Signal Processing Systems. 1st ed. New York: Wiley-Interscience, 2008.