Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia Eletrônica

Implementação de uma FFT em FPGA para processamento de sinais phase-OTDR aplicado em sensores de aceleração distribuídos

Autor: Hachid Habib Cury

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda (FGA/UnB)

Brasília, DF

2021

Hachid Habib Cury

Implementação de uma FFT em FPGA para processamento de sinais phase-OTDR aplicado em sensores de aceleração distribuídos

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda (FGA/UnB)

Brasília, DF

2021

Hachid Habib Cury

Implementação de uma FFT em FPGA para processamento de sinais phase-OTDR aplicado em sensores de aceleração distribuídos/ Hachid Habib Cury. – Brasília, DF, 2021-

78 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda (FGA/UnB)

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2021.

1. FFT, FPGA, SoC. 2. RADIX-2, Phase-OTDR. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda (FGA/UnB). II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação de uma FFT em FPGA para processamento de sinais phase-OTDR aplicado em sensores de aceleração distribuídos

CDU 02:141:005.6

Hachid Habib Cury

Implementação de uma FFT em FPGA para processamento de sinais phase-OTDR aplicado em sensores de aceleração distribuídos

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 21 de maio de 2021 – Data da aprovação do trabalho:

Prof. Dr. Daniel Mauricio Muñoz Arboleda (FGA/UnB)
Orientador

Prof. Dr. Carlos Humberto Llanos Quintero (ENM/FT/UnB)
Convidado 1

Prof. Dr. Fabiano Araújo Soares (FGA/UnB)
Convidado 2

Brasília, DF
2021

Agradecimentos

Gostaria de agradecer,

Aos meus pais, Rita de Cassia Souza Cury e Rabib de Souza Cury, por todo amor, apoio e compreensão durante todos os momentos da minha vida.

A minha noiva Tayná Lins Portela, pela paciência e amor dedicado, por estar sempre ao meu lado, cuidando e me apoiando.

As minhas irmãs Samira de Souza Cury e Madiha de Souza Cury pela preocupação e atenção dedicadas sempre quando precisei.

Agradecimentos especiais à Carolina Franciscangelis (Unicamp) e Walter Margulis (Acreo Swedish ICT) pelo compartilhamento dos dados experimentais coletados com o sensor phase-OTDR, no âmbito da cooperação Brasil-Suécia através dos projetos de pesquisa financiados pelo CNPq/CISB/Saab.

Ao meu orientador Daniel Mauricio Muñoz Arboleda por sua confiança, dedicação e paciência durante a elaboração deste trabalho.

Também quero agradecer à todos meus amigos e familiares pelos momentos descontraídos, que me fizeram esquecer momentaneamente das minha obrigações e me deram forças para continuá-las posteriormente.

"A análise matemática é tão extensa como a própria natureza; define todas as relações perceptíveis, mede tempos, espaços, forças, temperaturas; esta difícil ciência se forma lentamente, mas preserva todos os princípios que uma vez adquiriu; ela cresce e se fortalece incessantemente no meio das muitas variações e erros da mente humana. Seu principal atributo é a clareza; não tem marcas para expressar noções confusas. Reúne fenômenos dos mais diversos, e descobre as analogias ocultas que os unem. " *(Joseph Fourier)*

Resumo

Sensores de fibra óptica distribuídos possuem a tecnologia chave para o desenvolvimento de estruturas inteligentes, uma vez que são capazes de aferir diversos parâmetros como temperatura, vibração, tensão mecânicas e podem ser totalmente integrados aos materiais da estrutura. Com o objetivo de obter a frequência de vibração para o monitoramento da saúde estrutural, foi proposto o desenvolvimento de um módulo em hardware reconfigurável para cálculo da transformada de Fourier dos sinais obtidos. No presente trabalho é abordado o desenvolvimento de um módulo FFT em ponto flutuante, baseado no algorítimo Cooley–Tukey de base 2 (Radix-2), desenvolvido para o processamento de sinais em um SoC FPGA. As multiplicações complexas em ponto flutuante serão efetuadas utilizando o algorítimo CORDIC para economia de hardware, e a unidade de geração de endereço pode ser configurada para o cálculo de uma FFT de 16 a 2048 pontos, a fim de englobar um número maior de sinais. O envio de dados entre o processador e o módulo desenvolvido utiliza o protocolo AXI-Stream com DMA, para leitura e escrita rápida da memória DDR. Utilizou-se de recursos do FPGA 13146 LUTs, 6256 FFs e 4,5 BRAMs, o dispositivo escolhido foi uma ZedBoard da família Zynq-7000 da Xilinx, a qual conta com um processador ARM Cortex-A9, dual-core e uma arquitetura de FPGA Artix-7.

Palavras-chave: FFT, FPGA, SoC, RADIX-2, Phi-OTDR.

Abstract

Distributed fiber optic sensors have the key technology to develop an intelligent structures, since they are capable of measuring various parameters such as temperature, vibration, mechanical stress and can be fully integrated with the materials of the structure. In order to obtain the frequency of vibration to the monitoring of structural health, it was proposed to develop a module in reconfigurable hardware for calculating the Fourier transform of the obtained signals. This work discusses the development of a floating point FFT module, based on the Cooley - Tukey base 2 (Radix-2) algorithm, developed for signal processing in an FPGA SoC. Complex floating point multiplications performed using the CORDIC algorithm for hardware savings, and an address generation unit can be configured to calculate an FFT from 16 to 2048 points, in order to encompass a greater number of signals. The sending of data between the processor and the developed module uses the AXI-Stream protocol with DMA, for fast reading and writing of the DDR memory. Utilizing the features of the FPGA 13146 LUTs, 6256 FFs and 4.5 BRAMs, the chosen device was a ZedBoard from the Zynq-7000 family from Xilinx, a qualified ARM Cortex-A9 processor, dual-core and an Artix-7 FPGA architecture.

Key-words: FFT, FPGA, SoC, RADIX-2, Phi-OTDR.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

FPGAs	Field Programmable Gate Arrays
VHDL	Very high speed integrated circuits Hardware Description Language
SDK	Software Development Kit
LUTs	Look Up Tables
FFs	Flip-Flops
BRAMs	Block RAMs
RAM	Random Access Memory
ROM	Read-Only Memory
SDP-RAM	Simple Dual-Port RAM
S-ROM	Single-port ROM
DSPs	Digital Signal Processing
IOB	Input/Output Bound
MSE	Mean Square Error
DDR	Double Data Rate
DAC	Digital to Analog Converter
CAD	Analog to Digital Converter
APD	Avalanche Photodiode
LEIA	Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
DFT	Discrete Fourier Transform
IDFT	Inverse Discrete Fourier Transform
DFS	Discrete Fourier Series

DTFT	Discrete-Time Fourier Transform
FT	Fourier continuous Transform
FS	Fourier Series
phase-OTDR	phase-sensitive Optical Time Domain
SoC	System On Chip
ARM	Advanced RISC Machine
CORDIC	COordinate Rotation DIgital Computer
RADIX-2	algorítimo Cooley–Tukey de base 2
AXI	Advanced eXtensible Interface
PS	Processing System
PL	Programmable Logic
ZedBoard	Zynq Evaluation and Development Board
IP	Intellectual Property
FSM	Finite State Machine

Sumário

1 Introdução

A fibra óptica representam uma tecnologia chave para o desenvolvimento de estruturas inteligentes, uma vez que podem ser totalmente integrados aos materiais da estrutura (ROSSETTO, 2004, pp.1). Exemplos de aplicações incluem: o monitoramento da deformação mecânica em grandes estruturas como prédios, pontes, barragens, tanques de armazenagem, navios, plataformas de petróleo, aviões, foguetes, dutos com longas distâncias. Distribuição de temperatura em transformadores elétricos de potência, geradores, sistemas reatores, fornos, sistema de controle de processos, sistema de detecção de incêndio, dutos de perfuração. E sensores embutidos em materiais compósitos para aplicações na obtenção, em tempo real, da deformação mecânica, vibração e temperatura em estruturas e fuselagens, especialmente na indústria aeronáutica (ROSSETTO, 2004, pp.13).

Na indústria aeronáutica a estrutura interna de uma aeronave possui diversas áreas de interesse para testes estruturais, a capacidade de medir a frequência de vibrações é desejável para testes de fadiga do material e do estudo de ocorrência de fenômenos, como flutter (FRANCISCANGELIS, 2017). Obter o valor das principais frequências de um sinal se torna muito mais natural ao analisar o espectro do mesmo no domínio da frequência.

A transformada discreta de Fourier (DFT) é a ferramenta mais amplamente usada em sistemas de processamento digital de sinal (DSP). Ela tem papel indispensável em muitas aplicações, como fala, processamento de áudio e imagem, análise de sinal, sistemas de comunicação e muitos outros. Ela mapeia a sequência do domínio do tempo para uma sequência do domínio da frequência do mesmo comprimento, enquanto a transformada discreta inversa de Fourier (IDFT) realiza o oposto (KUMAR; SAHOO; MEHER, 2019).

Porém a aplicação direta da DFT em um algoritmo para sinais com grande comprimento, podem consumir muito tempo devido a quantidade de cálculos a serem realizados, sendo proibitivos mesmo em computadores de alto desempenho. Segundo (LATHI, 2007, pp.719) o algoritmo de transformada rápida de Fourier (FFT) é o que torna a transformada de Fourier acessível para o processamento digital de sinais, reduz o número de cálculos da ordem de N_0^2 para $N_0 \times log_2(N_0)$, para uma DFT de comprimento N_0.

Os cálculos realizados, são somas e multiplicações complexas em ponto flutuante, sendo que as multiplicações complexas consomem muito silício e recursos escassos, como DSP. Como alternativa a esse problema, é utilizado o algoritmo CORDIC, que realiza as operações de multiplicação complexas através de somas e deslocamentos.

Os processadores de software e FPGAs (*Field Programmable Gate Arrays*) são os núcleos de trabalho da maioria dos sistemas embarcados. Os Sistemas em Chip (SoC) FPGAs surgiram da necessidade do processamento de softwares em projetos de hardware, unindo processadores e FPGAs em um único chip, possibilitando o gerenciamento de alto nível de processadores e as operações de processamento paralelo de dados de um FPGA. Dessa forma, a criação de um módulo FFT em FPGA para acelerar o cálculo da transformada de Fourier em um sistema embarcado com suporte Linux, é bastante útil para área de processamento digital de sinal, vistos as vantagens de possuir um sistema operacional completo realizando leitura e escrita de arquivos, se comunicando com diversas interfaces, realizando leitura de sensores e armazenamento em memória, juntamente com o processamento desses dados de forma paralela e otimizada em um FPGA.

1.1 Contextualização

Segundo (BARRIAS; CASAS; VILLALBA, 2016), citado por (FRANCISCANGELIS, 2017, pp.15), os sensores de fibra óptica distribuídos foram usados para monitorar vários parâmetros, tais como temperatura, vibração, tensão mecânica, campo magnético e corrente elétrica.

Esses sensores têm a vantagem sobre os elétricos por apresentarem maior sensibilidade, imunidade a interferências eletromagnéticas, ampla faixa dinâmica e podem medir de forma distribuída (PRIETO; ALEJANDRO, 2019, pp.10).

Segundo (FRANCISCANGELIS, 2017), a capacidade de medir vibrações entre 5 a 60 Hz em superfície de controle de vôo, como os ailerons, o leme ou o elevador de aviões é de interesse ao estudo da ocorrência de fenômenos indesejáveis, como flutter. Além disso, a capacidade de medir frequências mais altas também pode ser de interesse, uma vez que os testes de fadiga do material podem ocorrer em faixas de frequência além de 100 Hz.

A fibra óptica é uma ótima opção para essas aplicações, uma vez que podem ser totalmente integrados aos materiais da estrutura, servindo como sensores distribuídos com alta resolução, na ordem de dezenas de centímetros(MUNOZ et al., 2017).

Na indústria aeronáutica, a estrutura interna de uma aeronave possui centenas de áreas de interesse para testes estruturais. Algumas estruturas são móveis, como flaps, portas e trem de pouso. Isso requer o agrupamento cuidadoso de centenas de fios de instrumentação (VULLIEZ, 2013). O sensoriamento com fibra óptica oferece a capacidade de monitorar estruturas ao longo da fibra reduzindo a fiação volumosa exigida por sensores eletrônicos tradicionalmente utilizados, já que oferecem alta capacidade de multiplexação por medir de forma distribuída, como mostrado na figura 1.

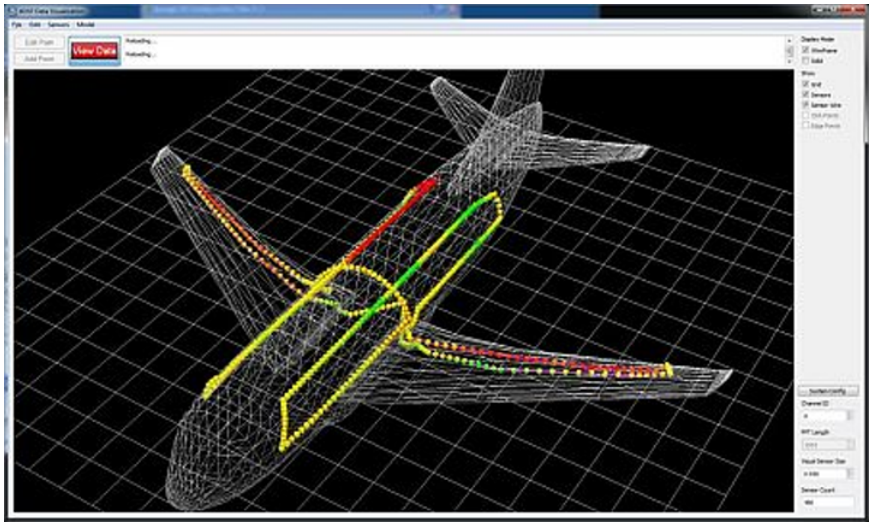


Figura 1 – Sensor de fibra distribuída pela estrutura de um avião comercial.
Fonte:(VULLIEZ, 2013)

1.2 Justificativa

Como visto anteriormente, a capacidade de identificar pontos de vibração ao longo de uma estrutura e obter suas frequências através do processamento digital dos sinais coletados é de grande interesse à industria. A utilização da fibra ótica para tal finalidade possui vantagens, como integração aos matérias da estrutura, imunidade a interferências eletromagnética e medições distribuídas.

A frequência de vibração é obtida ao analisar o ponto de pertubação de um conjunto de traços de medições consecutivas. A fim de executar tal procedimento de forma autônoma, realizando o gerenciamento de alto nível dos dados coletados, a comunicação com interfaces externas e o aceleramento dos cálculos da FFT em hardware, foi pensado na utilização de um dispositivo SoC FPGA para esta implementação.

1.3 Objetivos

1.3.1 Objetivo geral

Construir um módulo em um dispositivo SoC FPGA para a realização do cálculo da transformada rápida de fourier (FFT) em ponto flutuante, baseada no algorítimo Cooley–Tukey de base 2 (Radix-2) e no algorítimo CORDIC para economia de DSPs. De forma a utiliza-lo para obter os valores das frequências sofridas ao longo da fibra óptica para uma série de experimentos realizados com um sensor phase-OTDR.

1.3.2 Objetivos específicos

- Projetar um módulo multiplicador por constante em ponto flutuante sem utilização dos recursos DSPs da FPGA, de forma a compensar o ganho do CORDIC;
- Modificar o código do CORDIC fornecido para cálculo de senos e cossenos, de forma a implementar multiplicações complexas;
- Criar o módulo borboleta de decimação em frequência utilizando os somadores em ponto flutuante usados pelo CORDIC;
- Projetar um endereçador configurável, que realize a localização das entradas e saídas do módulo borboleta em memória para uma FFT de 16,32,64,128,256,512,1024 ou 2048 pontos.
- Projetar os blocos controladores para a realização dos cálculos da FFT utilizando 1 e 8 módulos borboletas;
- Desenvolver o block design, ligando o processador ZYNQ ao módulo projetado, utilizando a comunicação AXI4-Stream para o recebimento de dados e envio dos resultados.

2 Fundamentação teórica

2.1 Transformadas de Fourier

A série de Fourier foi desenvolvida pelo matemático Jean-Baptiste Joseph Fourier e publicada em um trabalho intitulado Mémoire sur la propagation de la chaleur dans les corps solides, em 1822. Sua descoberta foi perceber que sinais mais complexos poderiam ser representados através da simples soma de senoides, resultando em quatro representações de Fourier distintas, cada uma aplicável a uma classe de sinal. Os sinais podem ser classificados como sinais contínuos ou discretos e sinais periódicos ou aperiódicos, conforme mostrado na tabela abaixo.

Tabela 1 – Relação entre tipo de sinal e sua representação de Fourier

Tempo	Periódico	Não periódico
Contínuo	Série de Fourier (FS) (Discreta e Não periódica)	Transformada de Fourier contínua (FT) (Contínua e Não Periódica)
Discreto	Série Discreta de Fourier (DFS/DFT) (Discreta e Periódica)	Transformada de Fourier de Tempo Discreto (DTFT) (Contínua e Periódica)

Para sinais não periódicos, o espectro de Fourier é contínuo. Essa continuidade significa que o sinal é representado pela soma de senóides ou exponenciais de todas as frequências em um intervalo contínuo de frequência. Sendo assim, para processar os sinais com um computador, os mesmos devem necessariamente que ter duração finita e devem estar na forma digital (discretos).

A representação por série de Fourier de um sinal periódico de tempo contínuo em geral requer infinitas exponenciais complexas harmonicamente relacionadas, enquanto a série de Fourier para qualquer sinal de tempo discreto com período N requer apenas N exponenciais complexas harmonicamente relacionadas. (OPPENHEIM; SCHAFER, 2013, pp.368)

Sendo assim, a única representação de Fourier que poderia ser utilizada na prática para calculo em computadores seria a Série Discreta de Fourier (DFT), já que resposta da mesma é um sinal discreto de duração finita. Porém a aplicação direta da DFT em um algoritmo consome muito tempo devido a quantidade de cálculos a serem realizados.

Em meados da década de 1960, um algoritmo, conhecido atualmente como transformada rápida de Fourier, ou FFT (Fast Fourier Transform), foi apresentado. Esse algoritmo-mo, que foi descoberto independentemente por Cooley e Tukey em 1965 [...].

O que tornou sua descoberta moderna tão importante foi o fato de que a FFT mostrou-se perfeitamente adequada para uma eficiente implementação digital o que reduziu em algumas ordens de grandeza o tempo necessário para calcular as transformadas. Com essa ferramento, muitas ideias interessantes, mas anteriormente impraticáveis, utilizando a série e a transformada de Fourier de tempo discreto, de repente se tornaram realizáveis (OPPENHEIM; WILLSKY; NAWAB, 2010, p.107).

Esse algoritmo, chamado de transformada rápida de Fourier (FFT), reduz o número de cálculos da ordem de N_0^2 para $N_0 \times log_2(N_0)$. Para um N_0 grande, esses cálculos podem consumir muito tempo, sendo proibitivos, mesmo em um computador muito rápido. O algoritmo de FFT é o que torna a transformada de Fourier acessível para o processamento digital de sinais (LATHI, 2007, pp.719)

2.1.1 Série Discreta de Fourier (DFS/DFT)

Os sinais periódicos de tempo discreto podem ser incorporados dentro da estrutura da transformada de Fourier de tempo discreto, interpretando a transformada de um sinal periódico como um trem de impulsos no domínio da frequência.

Considerando uma sequência $X[n]$ periódica com período N, de modo que X[n] = X[n+rN] para quaisquer valores inteiros de n e r. Assim, tal sequência pode ser representada por uma série de Fourier correspondente a uma soma de sequências exponenciais complexas com frequências que são múltiplos inteiros da frequência fundamental $2\pi/N$. Essas exponenciais complexas podem ser representadas por fasores complexos

$$e_k[n] = e^{\pm jw_0 kn} = e^{\pm j(2\pi/N)kn} = e_k[n + rN] \tag{2.1}$$

Como citado anteriormente, diferente da série de Fourier de tempo contínuo que em geral requer infinitas exponenciais complexas relacionadas, a série de Fourier para qualquer sinal de tempo discreto com período N requer apenas N exponenciais complexas harmonicamente relacionadas. Assim, a representação por série de Fourier de uma sequência periódica x[n] precisa apenas conter N dessas exponenciais complexas. Por conveniência de notação, escolhemos k no intervalo de 0 a N-1.

Segundo (OPPENHEIM; WILLSKY; NAWAB, 2010, pp.112), a série de Fourier de um sinal em tempo contínuo é representada pelo seguinte par de expressão:

$$^{1}x(t) = \frac{1}{T} \sum_{k=-\infty}^{+\infty} a_k e^{jkw_0 t} \tag{2.2}$$

$$a_k = \int_T x(t) e^{-jkw_0 t} dt \tag{2.3}$$

[1] A constante multiplicativa 1/N é incluída na Equação 2.2 por conveniência. Ela também poderia ser incorporada na definição de a_k

Sendo a Equação 2.2 denominada como equação de síntese, e a Equação 2.3, como equação de análise da série de Fourier de tempo contínuo.

De forma análoga, podemos escrever a equação de síntese para a série de Fourier de tempo discreto trocando a variável de tempo contínuo t para a de tempo discreto n e ajustando o número de elementos do somatório de infinito para N elementos. O mesmo pode ser feito para a equação análise ao substituir o período T por N e trocar a integral do período por um somatório de N elementos. Obtendo assim as seguintes equações:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} a_k e^{jkw_0 n} \tag{2.4}$$

$$a_k = \sum_{n=0}^{N-1} x[n] e^{-jkw_0 n} \tag{2.5}$$

Para adequação da notação é definido:

$$a_k \Rightarrow X[k]$$

$$W_N = e^{-j(2\pi/N)} = e^{-jw_0} \tag{2.6}$$

Com essa notação, o par síntese-análise da DFT é representado da seguinte forma conforme (OPPENHEIM; SCHAFER, 2013, pp.369):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \tag{2.7}$$

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.8}$$

Sendo a Equação 2.7 denominada como equação de síntese da série de Fourier de tempo discreto, e a Equação 2.8, como equação de análise da série de Fourier de tempo discreto.

2.1.2 Transformada rápida de Fourier (FFT)

Segundo (OPPENHEIM; SCHAFER, 2013, pp.426) os algoritmos de FFT são baseados no princípio fundamental da decomposição do cálculo da transformada de Fourier discreta de uma sequência de comprimento N em transformadas de Fourier discretas de comprimento menor, que são combinadas para formar a transformada de N pontos. Essas transformadas de comprimento menor podem ser calculadas por métodos diretos, ou então podem ser decompostas novamente em transformadas ainda menores.

Os primeiros a criarem um algoritmo foram Cooley e Tukey, desde então, uma variedade de algoritmos FFT foram desenvolvidos, como o radix-4, split-radix e a Transformada discreta de Hartley (FHT). Segundo (UZUN et al., 2003), que implementou os

algoritmos FFT mencionados acima usando Handel-C para compilar os programas diretamente no hardware FPGA. O algoritmo Radix-4 fornece melhores desempenho em termos de tempo de computação enquanto radix-2 supera os outros algoritmos em termos de área e frequência máxima do sistema, como mostrado na figura 2.

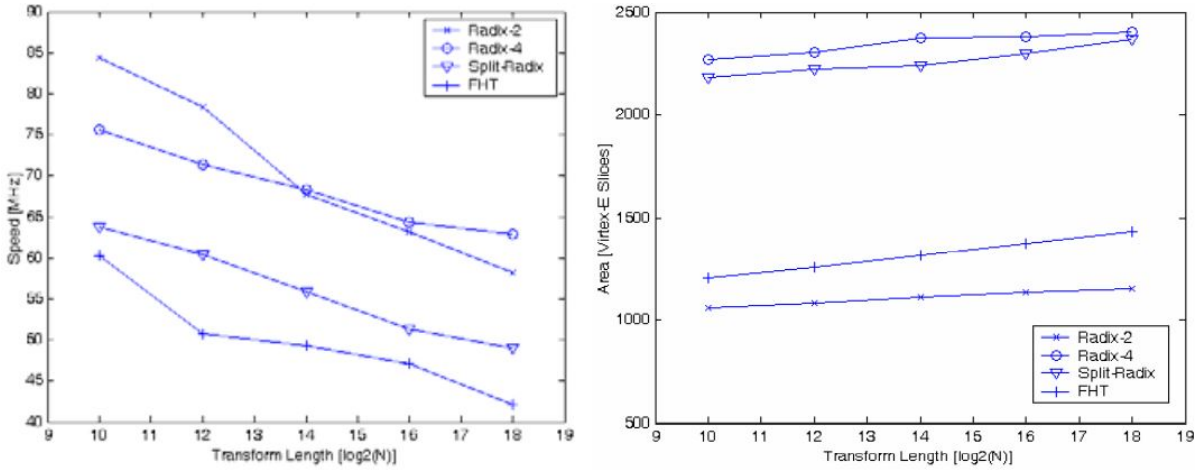


Figura 2 – Comparação de algoritmos em velocidade e área.
Fonte:(UZUN et al., 2003, pp.620)

Por outro lado, temos a contribuição de (GANAPATHIRAJU et al., 1999), que apresentou uma análise rigorosa dos algoritmos acima mencionados, em processadores de uso geral. Foi analisado de cada algoritmo o número de operações matemáticas, tempo de computação e requisitos de memória, apresentados nas tabelas 2, 3 e 4. Como resultado, Ganapathiraju concluiu que o FHT era o melhor algoritmo geral em todas as plataformas, oferecendo o tempo de execução mais rápido e exigindo quantidades razoavelmente pequenas de memória.

Tabela 2 – Número de operações envolvidos no cálculo de uma FFT de 1024 pontos.
Fonte:(GANAPATHIRAJU et al., 1999, pp.21)

Algorithm	Float Adds	Float Mults	Integer Adds	Integer Mults	Binary Shifts
RAD2	14336	20480	19450	2084	1023
RAD4	8960	14336	12902	3071	277
SRFFT	**5861**	**5522**	12664	2542	1988
FHT	7420	8841	**3235**	2048	**12**

Tabela 3 – Tempo de computação (em microssegundos) de diversos algoritmos
Fonte:(GANAPATHIRAJU et al., 1999, pp.21)

Algorithm	FFT Order					
	16	**64**	**256**	**1024**	**4096**	**16384**
RAD2	20	60	260	1960	6800	30500
RAD4	20	60	300	1800	6940	29000
SRFFT	20	40	140	660	3700	17260
FHT	**20**	**40**	**120**	**560**	**3240**	**14020**

Tabela 4 – Uso de memória e tamanho do código para cálculo de uma FFT de 1024 pontos
Fonte:(GANAPATHIRAJU et al., 1999, pp.22)

Algorithm	Memory Usage (Bytes)	Object Code (Bytes)
RAD2	**72440**	**5190**
RAD4	72536	5293
SRFFT	72508	6275
FHT	72652	11506

O algoritmo FHT ocupa pouca área em silício e apresentou vantagens em número de operações e velocidade. Porém sua implementação em hardware ficou limitada a uma baixa frequência de operação conforme (UZUN et al., 2003). Sendo otimizado e aumentando a frequência de trabalho, este algoritmo parece ser um bom candidato para implementação em hardware em trabalhos futuros. Devido a baixa complexidade, uma maior frequência máxima e por ocupar uma menor área em silício, o algoritmo de Radix-2 foi escolhido para ser implementado neste trabalho.

O algoritmo de Tukey-Cooley pode ser agrupados em dois tipos básicos: decimação em tempo e decimação em frequência. Abaixo será demonstrado o método da decimação em tempo, conforme (LATHI, 2007, pp.720), considerando que a quantidade de amostras de entrada N_0 igual a uma potência de 2.

2.1.3 Decimação em tempo

Primeiramente é divido a sequência de dados x_n com N_0 pontos em duas sequências de $(N_0/2)$ pontos constituídas de amostras de números pares e ímpares, respectivamente. Logo a Equação 2.8 poderá ser representada da seguinte forma:

$$X_k = \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0}^{2nk} + \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0}^{(2n+1)k} \tag{2.9}$$

Sabendo que

$$W_{N_0/2} = e^{-j(2\pi/(N_0/2))} = e^{-j2(2\pi/(N_0))} = W_{N_0}^2 \tag{2.10}$$

temos

$$X_k = \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0/2}^{nk} + W_{N_0}^{k} \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0/2}^{nk} \tag{2.11}$$

$$X_k = G_k + W_{N_0}^{k} H_k \ \{k \in N \ \mid \ 0 \leq k \leq N_0 - 1\} \tag{2.12}$$

Na qual G_k e H_k são as DFTs de $(N_0/2)$ pontos das sequências de números pares e ímpares, respectivamente. Além disso, são periódicas, com o período $(N_0/2)$. Logo,

$$\begin{aligned} G_{k+(N_0/2)} &= G_k \\ H_{k+(N_0/2)} &= H_k \end{aligned} \tag{2.13}$$

Assim

$$W_{N_0}^{k+(N_0/2)} = W_{N_0}^{k} W_{N_0}^{N_0/2} = e^{-j\pi} W_{N_0}^{k} = -W_{N_0}^{k} \tag{2.14}$$

Obtendo assim as equações para o calculo da FFT. Essas equações podem ser representadas convenientemente pelo gráfico de fluxo de sinal mostrado na Fig. 3 conhecida como borboleta.

$$\begin{aligned} X_k &= G_k + W_{N_0}^{k} H_k \\ X_{k+(N_0/2)} &= G_k - W_{N_0}^{k} H_k \end{aligned} \tag{2.15}$$

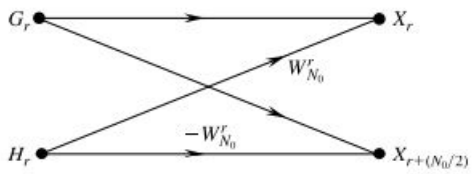


Figura 3 – Gráfico borboleta do fluxo de sinal para decimação em tempo.
Fonte:(LATHI, 2007, pp.721)

O processo é repetido, dividindo g_n e h_n em duas sequências de $(N_0/4)$ pontos correspondentes às amostras de números pares e ímpares. Continuamos, então, esse processo até termo alcançado a DFT de um ponto. Esses passos para o caso de $N_0 = 8$ estão mostrados na Fig.4

2.1.4 Decimação em frequência

Os algoritmos de FFT com dizimação no tempo são baseados em estruturar o cálculo da DFT formando subsequências cada vez menores da sequência de entrada x[n]. Alternativamente, podemos considerar a divisão da sequência da DFT X[k] em subsequências cada vez menores, da mesma maneira. Algoritmos de FFT baseados nesse procedimento são comumente chamados de algoritmos de dizimação na frequência. (OPPENHEIM; SCHAFER, 2013, pp.434)

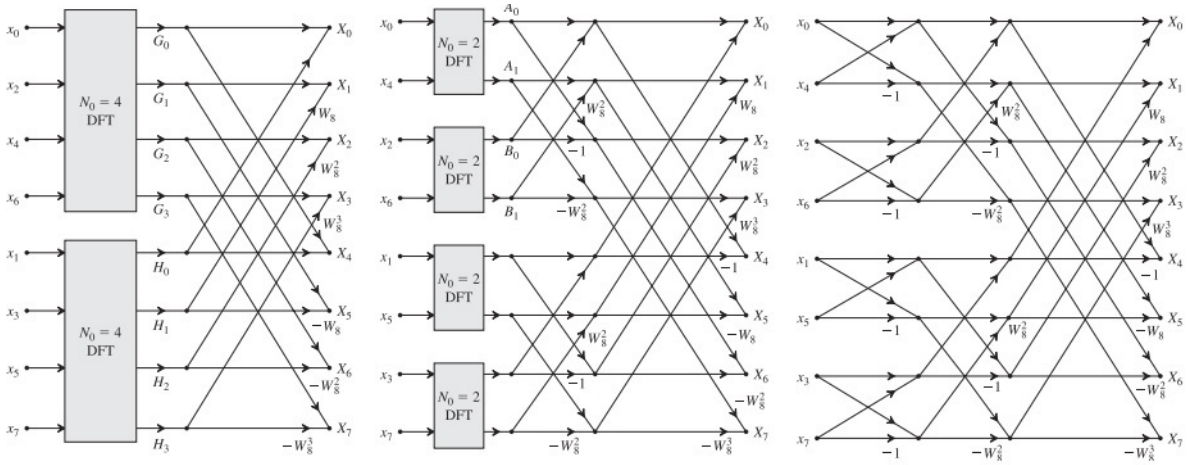


Figura 4 – Passos sucessivos em uma FFT de 8 pontos.
Fonte:(LATHI, 2007, pp.722)

A equação para operação da borboleta foi retirada de (OPPENHEIM; SCHAFER, 2013, pp.437), foi realizado modificações na notação para padronização com as equações anteriores e estão apresentadas abaixo.

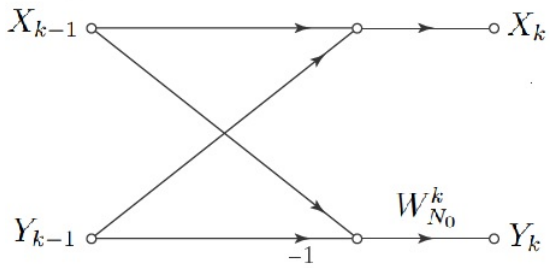


Figura 5 – Gráfico borboleta do fluxo de sinal para decimação em frequência.
Modificado de:(OPPENHEIM; SCHAFER, 2013)

$$\begin{aligned} X_k &= X_{k-1} + Y_{k-1} \\ Y_k &= (X_{k-1} - Y_{k-1}) \quad W_{N_0}^k \end{aligned} \tag{2.16}$$

Diferentemente da decimação em tempo, na frequência a ordem de entrada dos sinais na FFT ficam ordenados de forma crescente, e a saída ordenada no esquema Bit-Reverse. Como mostrado na figura 6.

2.2 Algorítimo CORDIC

Coordinate Rotation Digital Computer é abreviado como CORDIC. O conceito principal da aritmética CORDIC é baseado nos princípios simples e antigos da geometria bidimensional. Mas a formulação iterativa de um algoritmo computacional para sua implementação foi descrita pela primeira vez em 1959 por Jack Volder para o cálculo de funções trigonométricas, multiplicação e divisão (MEHER et al., 2009, pp.1893).

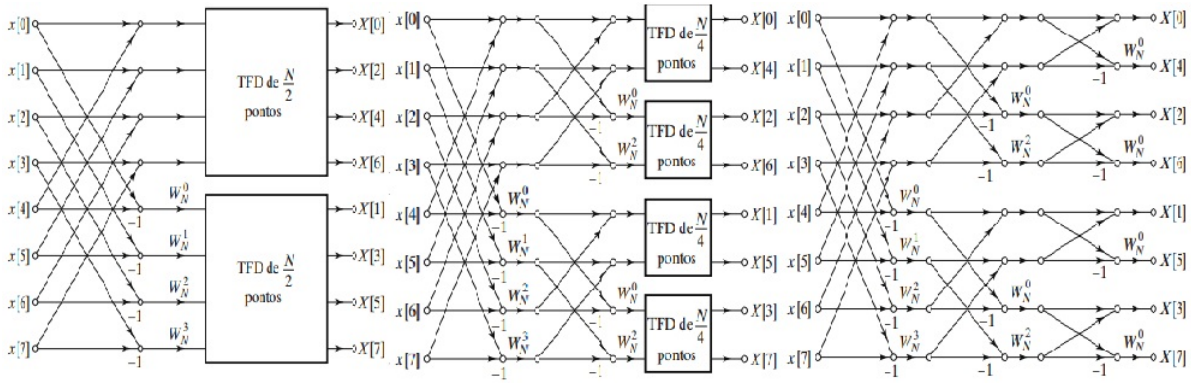


Figura 6 – Passos sucessivos em uma FFT de 8 pontos para decimação em frequência.
Fonte:(OPPENHEIM; SCHAFER, 2013, pp.437)

Segundo (VOLDER, 2000), a técnica de computação CORDIC foi desenvolvida especialmente para uso no computador de bordo do jato supersônico B-58. Neste computador, a maior parte da computação envolvia a solução descontínua e programada das relações trigonométricas das equações de navegação em tempo real.

A computação baseada em CORDIC recebeu maior atenção em 1971, quando John Walther, mostrou que, variando alguns parâmetros simples, poderia ser usado como um único algoritmo para implementação unificada de uma ampla gama de funções transcendentais elementares envolvendo logaritmos, exponenciais e raízes quadradas junto com aquelas sugeridas por Volder (MEHER et al., 2009, pp.1893).

Por razões de simplicidade, as operações trigonométricas no computador CORDIC podem ser funcionalmente descritas como o equivalente digital de um resolver analógico. Semelhante à operação desse resolver, existem dois modos de computação, rotação e vetorização. Em essência, a técnica de computação básica usada nos modos rotação e vetorização no CORDIC é uma sequência passo a passo de pseudo rotações que resultam em uma rotação geral através de um determinado ângulo (rotação) ou resultam em um argumento angular final de zero (vetorização)(VOLDER, 1959, pp.331).

Na figura 7 são apresentados as saídas em relação as entradas x, y e z, para cada configuração no modo vetorização. Sendo em 7a) a configuração hiperbólica, em 7b) a configuração linear e em 7c) a configuração circular.

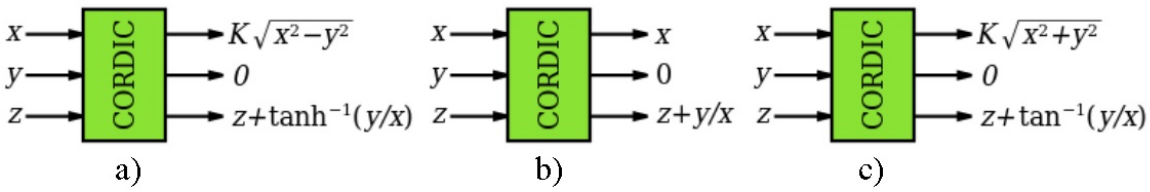


Figura 7 – CODIC modo vetorização: a)Hiperbólica; b)Linear; e C)Circular
Fonte:(WIKIPéDIA, 2020)

Na figura 8 são apresentados as saídas em relação as entradas x, y e z, para cada configuração no modo rotação. Sendo em 8a) a configuração hiperbólica, em 8b) a configuração linear e em 8c) a configuração circular.

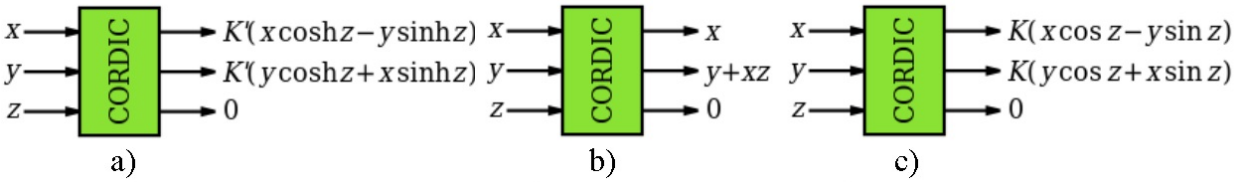


Figura 8 – CODIC modo rotação: a)Hiperbólica; b)Linear; e C)Circular
Fonte: (WIKIPéDIA, 2020)

Jack Volder define em sua obra (VOLDER, 1959, pp.332), que o valor da constante K é dado pela seguinte equação

$$K = \prod_{i=0}^{23} \sqrt{1 + 2^{-2i}} = 1,646760255 \tag{2.17}$$

Para o desenvolvimento deste trabalho, foi utilizado o CORDIC em modo rotação circular, conforme a figura 8c). Nas subseções 2.2.1 e 2.2.2 será explicado o funcionamento deste algoritmo e sua utilização no cálculo da FFT.

2.2.1 CORDIC modo rotação circular

Considere um ponto $P_0 = [x_0, y_0]$ que será rotacionado com um angulo θ até um ponto $P_n = [x_n, y_n]$, conforme mostra a figura abaixo.

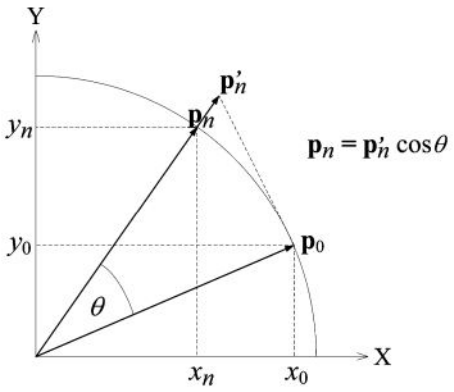


Figura 9 – Rotação de um vetor no plano 2D Fonte: (MEHER et al., 2009)

O valor de x_n e y_n pode ser calculado com a seguinte equação:

$$\begin{aligned} x_n &= x_0 cos(\theta) - y_0 sen(\theta) \\ y_n &= y_0 cos(\theta) + x_0 sen(\theta) \end{aligned} \tag{2.18}$$

Ao colocar o $cos(\theta)$ em evidência, temos

$$\begin{aligned} x_n &= cos(\theta)[x_0 - y_0 tan(\theta)] \\ y_n &= cos(\theta)[y_0 + x_0 tan(\theta)] \end{aligned} \tag{2.19}$$

Jack Volder define então em (VOLDER, 2000, pp.101) que o valor de $tan(\theta)$ seria restrito a um número binário, logo

$$tan(\theta) = 2^{-i} \;\rightarrow\; \theta = arctan(2^{-i}) \tag{2.20}$$

Dessa forma, multiplicar por $tan(\theta)$ seria equivalente a uma operação de deslocamento de bits. Substituindo a equação 2.20 na equação 2.19, obtemos

$$\begin{aligned} x_n &= cos(arctan(2^{-i}))[x_0 - y_0 2^{-i}] \\ y_n &= cos(arctan(2^{-i}))[y_0 + x_0 2^{-i}] \end{aligned} \tag{2.21}$$

Para simplificar a Eq.2.21, faremos a seguintes definições

$$\begin{aligned} \omega = tan(\phi) \rightarrow \phi = arctan(\omega) \\ \omega = \frac{sen(\phi)}{cos(\phi)} \rightarrow \omega^2 = \frac{sen(\phi)^2}{cos(\phi)^2} \\ \omega^2 + 1 = \frac{sen(\phi)^2 + cos(\phi)^2}{cos(\phi)^2} = \frac{1}{cos(\phi)^2} \\ \frac{1}{\sqrt{\omega^2 + 1}} = cos(\phi) \end{aligned}$$

$$cos(arctan(\omega)) = \frac{1}{\sqrt{\omega^2 + 1}} \tag{2.22}$$

Substituindo então a Eq.2.22 na Eq.2.21, é encontrado a fórmula do CORDIC

$$\begin{aligned} x_{i+1} &= \varepsilon_i[x_i - d_i.y_i 2^{-i}] \\ y_{i+1} &= \varepsilon_i[y_i + d_i.x_i 2^{-i}] \\ \varepsilon_i &= cos(arctan(2^{-i})) = \frac{1}{\sqrt{2^{-2i} + 1}} \end{aligned} \tag{2.23}$$

Dessa forma, cada nova iteração do algoritmo será igual a iteração anterior rotacionada com a metade da rotação anterior, conforme Eq.2.20. Para achar a rotação do angulo desejado, essa nova rotação poderá ser tanto positiva quanto negativo, por isso, é adicionado a variável de controle d_i, sendo $d_i = \pm 1$. Esse processo é exemplificado na figura 10a), no qual mostra como é calculado a rotação de um ponto a_0 até a_n no CORDIC.

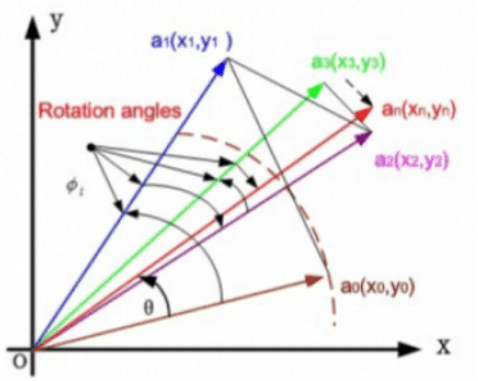


Figura 10 – Iterações do CORDIC de 0 a N conforme Eq.2.23
Fonte: (NEVES, 2016)

Portanto, torna-se necessário a criação de uma nova equação para o cálculo do ângulo. Lembrando que os valores de θ são delimitados na Eq.2.20, consequentemente

$$z_{i+1} = z_i - d_i arctan(2^{-i}) \quad \begin{cases} d_i = -1 & , z_i < 0 \\ d_i = +1 & , z_i \geq 0 \end{cases} \tag{2.24}$$

O valor do próximo ângulo será igual ao valor do ângulo anterior menos $arctan(2^{-i})$, perceba que quanto mais z_{i+1} tende a 0, mais próximo o ponto de iteração esterá do ponto desejado. Dessa forma, a variável de controle d_i sempre irá alterar seu sinal conforme o ângulo calculado esteja em relação ao zero.

Muitas vezes o valor ε_i é ignorado no processo iterativo e então aplicado posteriormente com um fator de escala,

$$\varepsilon(n) = \prod_{i=0}^{n-1} \varepsilon_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2^{-2i}+1}}$$

$$\varepsilon = \lim_{n\to\infty} \varepsilon(n) \approx 0,607252935008881 \tag{2.25}$$

$$\varepsilon = \frac{1}{K}$$

Sendo que ε possui como função recuperar esse valores originais, servindo então como uma constante de normalização. Seu valor com 18 iterações é 0,607252935032446, obtendo dez casas após o ponto decimal.

Em resumo, para a implementação do algoritmo em hardware utilizamos as seguintes equações:

$$
\begin{aligned}
x_{i+1} &= x_i - d_i.y_i 2^{-i} \\
y_{i+1} &= y_i + d_i.x_i 2^{-i} \\
z_{i+1} &= z_i - d_i arctan(2^{-i}) \\
d_i &= \begin{cases} -1 & , z_i < 0 \\ +1 & , z_i \geq 0 \end{cases}
\end{aligned}
$$

Assim, ao invés de realizar a multiplicação a cada iteração de x_{i+1} e y_{i+1} com ε_i, é feito apenas uma multiplicação com as entradas x e y a fim de normalizar a saída do CORDIC, como mostrado na figura abaixo.

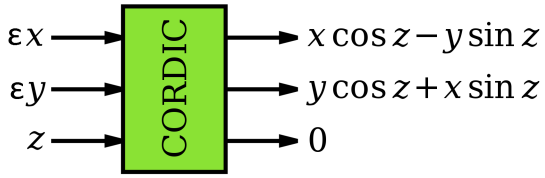


Figura 11 – CORDIC Block, Circular, Rotation.
Modificado de: (WIKIPéDIA, 2020)

2.2.2 Relação com algorítimo Cooley–Tukey (FFT)

Como visto no subseção 2.1.2, o algorítimo Cooley–Tukey de base 2, conhecido também como Radix-2, deve implementar a seguinte equação para calcular uma iteração da FFT (Eq.2.16).

$$
\begin{aligned}
X_k &= X_{k-1} + Y_{k-1} \\
Y_k &= (X_{k-1} - Y_{k-1}) \quad W_{N_0}^k
\end{aligned}
$$

Segundo a fórmula de Euler, $W_{N_0}^k$ é definido como:

$$
W_{N_0}^k = e^{-jkw_0} = cos(kw_0) - jsen(kw_0) \tag{2.26}
$$

Lembrando que X e Y são números complexos, logo a operação $C_k W_{N_0}^k$ será representada pela equação abaixo

$$
C_k = X_{k-1} - Y_{k-1} = A_k + jB_k
$$

$$
C_k W_{N_0}^k = (cos(kw_0) - jsen(kw_0))(A_k + jB_k)
$$

$$
= A_k cos(kw_0) + B_k sen(kw_0) + j(B_k cos(kw_0) - A_k sen(kw_0)) \tag{2.27}
$$

A Eq. 2.27 pode ser obtida ao fazermos as seguintes modificações nas variáveis da figura 11.

$$\begin{aligned} x &= B_k = (X_{k-1} - Y_{k-1})_{IMAG} \\ y &= A_k = (X_{k-1} - Y_{k-1})_{REAL} \\ z &= kw_0 \end{aligned}$$

Assim, na primeira saída será obtido a parte imaginária da multiplicação complexa da Eq.2.16 e na segunda saída a parte real da mesma.

2.3 Sistemas em um Chip (Soc)

Processadores e FPGAs (*Field Programmable Gate Arrays*) são os núcleos de trabalho da maioria dos sistemas embarcados. Integrando a funcionalidade de gerenciamento de alto nível de processadores e as operações rigorosas em tempo real, extremas processamento de dados ou funções de interface de um FPGA em um único dispositivo forma um plataforma de computação integrada ainda mais poderosa.(ALTERA, 2014)

Segundo (SANTOS, 2015, pp.29), com a necessidade de processamento em software para resolução de problemas em projetos de hardware, por exemplo, em projetos que necessitavam de processamento e interfaces exteriores, uma das alternativas encontradas foi a emulação de processadores em FPGA. Como exemplo temos o MicroBlaze, o processador da Xilinx disponível em todas as suas ferramentas de desenvolvimento para FPGAs. Porém a estrutura de processador emulada implementada por blocos lógicos não possui a mesma eficiência encontrada nos cores de processadores atualmente desenvolvidos. Por isso, assim como os demais dispositivos alocados internamente nos FPGAs, os processadores passaram a ser acoplados nestes dispositivos.

De acordo com o manual (XILINX, 2017, pp.26), os dispositivos SoC da família Zynq-7000 da Xilinx são dividido em duas partes, a parte de processamento de software (PS) e a de estrutura de hardware com lógica programável (PL). No qual, em PS, temos um processador ARM Cortex-A9, dual ou single-core, sendo capaz de rodar sistemas operacionais completos. Já em PL, temos uma arquitetura de FPGA da série 7 da Xilinx, seus recursos variam entre os dispositivos, sendo assim, capazes de atender a uma ampla gama de aplicações. Na figura 12 é mostrado sinais e interfaces dos dispositivos da família Zynq-7000 para as partes PS e PL.

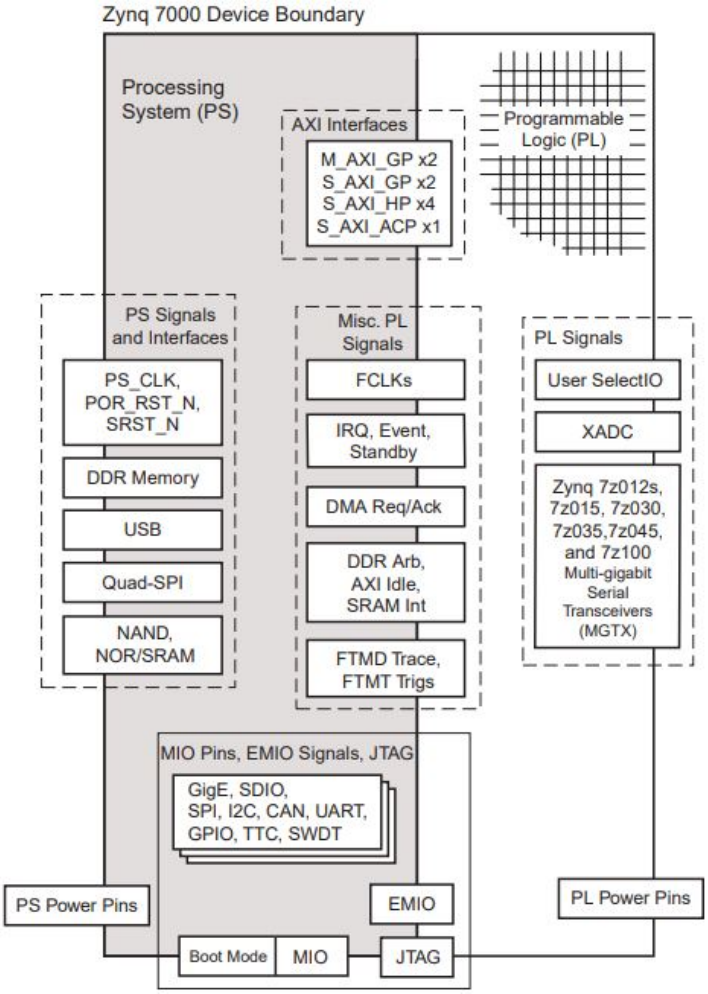


Figura 12 – Plataforma Zynq
Fonte: (XILINX, 2017)

Para estabelecer a ponte de comunicação entre o PL e o PS, a família Zynq-7000 utiliza como padrão o barramento AXI (Advanced eXtensible Interface). Essa interface permite estabelecer um fluxo de dados sincronizados entre PS e PL, de ambos os sentidos, suportando inclusive o disparo de interrupções de ambos os sistemas.

Devido as diferentes configurações entre cada dispositivos SoC da Xilinx, será dado mais detalhes entre as partes de processamento de software e a de estrutura de hardware com lógica programável na seção 3.1, no qual é escolhido o kit de desenvolvimento.

2.4 Protocolo AXI4

Conforme o guia de referência sobre o protocolo de comunicação AXI4, fornecido de Xilinx (XILINX, 2011). AXI faz parte da ARM AMBA, uma família de barramentos de microcontroladores introduzidos pela primeira vez em 1996. A primeira versão do AXI foi incluída pela primeira vez no AMBA 3.0, lançado em 2003. AMBA 4.0, lançado em 2010, inclui a segunda versão do AXI, AXI4. Existem três tipos de interfaces AXI4:

- AXI4 - Para requisitos de mapeamento de memória de alto desempenho, permite um envio de até 256 ciclos de transferência de dados com apenas uma única fase de endereço.
- AXI4-Lite - Para comunicação mapeada em memória simples e de baixo rendimento, é leve e permite uma única transação.
- AXI4-Stream - Para dados de streaming de alta velocidade, elimina a necessidade de uma fase de endereço e permite um envio ilimitado de transferência de dados. As interfaces e transferências AXI4-Stream não têm fases de endereço e, portanto, não são consideradas mapeadas na memória

Com o objetivo de armazenar e calcular a FFT de até 2048 pontos de forma rápida. Foi adotado para o projeto a comunicação AXI4-Stream, no qual, deverá realizar o stream de dados da memória DDR da ZedBoard para o IP criado. Dessa forma, combinando os protocolos AXI4-Stream com AXI4 de Memória Mapeada, descrita com mais detalhes nos subtópicos abaixo.

2.4.1 AXI4-Stream

Segundo o manual AMBA®4 AXI4-Stream Protocol (ARM, 2010). O protocolo AXI4-Stream é usado como uma interface padrão para conectar componentes que desejam trocar dados. A interface pode ser usada para conectar um único mestre, que gera dados, a um único escravo, que recebe dados. O protocolo também pode ser usado ao conectar um número maior de componentes mestres e escravos.

Além dos sinais de clock e reset, o protocolo é composto por 6 sinais principais, o TVALID, TREADY, TDATA, TLAST, TSTRB e TKEEP. Na tabela 5 temos as principais funções de cada um conforme descrito na documentação (ARM, 2010, pp.18).

Tabela 5 – Lista de sinais da interface AXI4-Stream

SINAL	FONTE	DESCRIÇÃO
ACLK	Clock source	O sinal de clock global. Todos os sinais são amostrados na borda ascendente.
ARESETn	Reset source	O sinal de reset global, ativo em baixo.
TVALID	Master	TVALID indica que o mestre está conduzindo uma transferência válida. Uma transferência ocorre quando tanto tvalid quanto TREADY são afirmados.
TREADY	Slave	Tready indica que o escravo pode aceitar uma transferência no ciclo atual.
TDATA[(8n-1):0]	Master	TDATA é a carga principal que é usada para fornecer os dados que estão passando através da interface.
TLAST	Master	O TLAST indica o final de um pacote.
TSTRB[(n-1):0]	Master	TSTRB indica se o conteúdo do byte associado do TDATA é processado como um byte de dados ou um byte de posição.
TKEEP[(n-1):0]	Master	TKEEP é o qualificatório byte que indica se o conteúdo do byte associado do TDATA é processado como parte do fluxo de dados.

A transmissão de dados começa através de um handshake, entre os sinais TVALID e TREADY Para que isso ocorra, o mestre deve primeiramente acionar a flag TVALID, neste momento o canal TDATA já armazena o valor da primeira saída, de forma que a comunicação comece imediatamente na hora em que o escravo aceite a transferência. Assim que TREADY seja ativado pelo escavo, a comunicação começa, sendo transferido dado a dado a cada borda de subida do clock ACLK. Quando o ultimo dado for ser enviado, o sinal TLAST deverá ir para alto durante o este ciclo de clock, assim como mostrado na figura 13.

Figura 13 – Comunicação AXI4-Stream de 16 dados capturada pelo Analisador Lógico Integrado (ILA). Fonte:(Autor, 2021)

2.4.2 Combinando AXI4 Stream com AXI4 de Memória Mapeada

Em muitos casos tem-se que utilizar sistemas que consistem na combinação de módulos de interface AXI4-Stream com módulos de AXI4 de Memória Mapeada. Esses sistemas funcionam em conjunto graças aos módulos AXI DMA, do inglês Data Memory Access. Os módulos AXI DMA servem para transformar dados vindos da interface AXI4 de Memória Mapeada para dados que podem ser transmitidos em interface AXI4- Stream, e vice-versa (NETO, 2017, pp.46).

O uso mais simples de um DMA seria transferir dados de uma parte da memória para outra, no entanto, um mecanismo de DMA pode ser usado para transferir dados de qualquer produtor de dados (por exemplo, uma ADC) para uma memória, ou de uma memória para qualquer consumidor de dados (por exemplo, um DAC) (Johnson, Jeff, 2014).

Com o intuito, então, de receber um conjunto de dados vindo da DDR da Zed-Board, calcular a transformada rápida de Fourier deste sinal e armazenar os resultados novamente na DDR, ficando os mesmos disponível para o processador. É implementado o diagrama de blocos da figura 14, no qual utiliza a interface AXI-lite para que o processador se comunique com o DMA AXI para configurar, iniciar e monitorar transferências de dados. A interface AXI4 para que o módulo DMA escreva e leia os dados da memória

DDR. E por fim, a interface AXI4-Stream para que o DMA envie para o módulo AXI FFT os dados lidos da memória, e faça o inverso para os dados de saída do módulo AXI FFT.

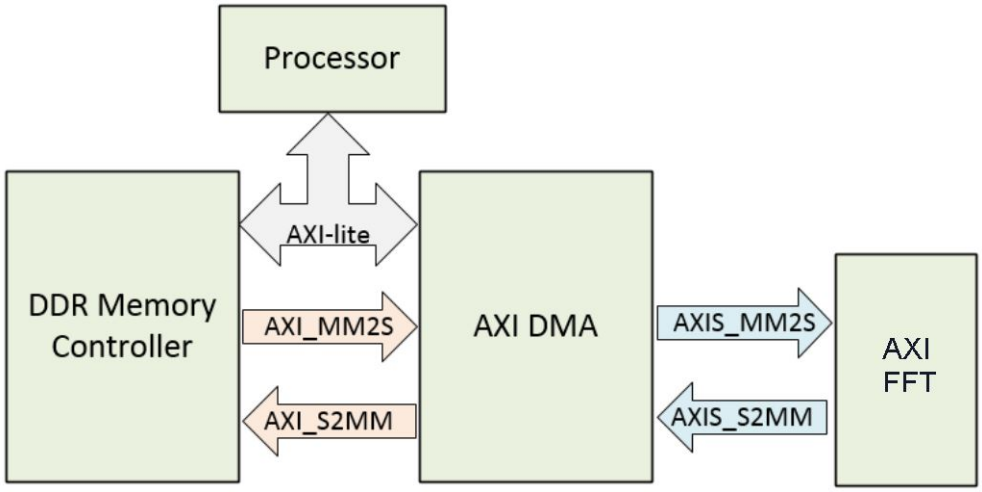


Figura 14 – Diagrama blocos da comunicação AXI4-Stream para a DDR
Modificado de: (Johnson, Jeff, 2014)

2.5 Revisão do estado da arte

Para fins de referência, foi realizado um levantamento de publicações sobre a implementação da transformada rápida de Fourier em dispositivos FPGAs. As famílias FPGAs escolhidas, os algoritmos e recursos utilizados, requisitos e resultados dos trabalhos pesquisados estão dispostos na tabela 6.

Tabela 6 – Trabalhos correlatos

REFERÊNCIA	FPGA	REPRESENTAÇÃO	MÉTODO	BITs	PONTOS	LUTs	FFs	DSPs	BRAMs	FREQ (MHz)	LATÊNCIA (CLKs)
(NGUYEN et al., 2018)	VIRTEX-7	P.FIXO	RADIX-2	-	1024	12737	2715	0	0	200	520
(BARBOSA, 2018)	ARTIX-7	P.FIXO	RADIX-2	16	1024	17021	22178	0	16	-	1728
(WANG et al., 2015)	VIRTEX-5	P.FIXO	RADIX-2	16	1024	2804	1589	16	3	298	2094
(SÁNCHEZ et al., 2008)	VIRTEX-2	P.FIXO	RADIX-2	8	1024	-	20873	0	0	186	1682
(INGEMARSSON et al., 2017)	VIRTEX-4	P.FIXO	$R2^2$ SDF	18	1024	-	1142	16	6	400	1068
(INGEMARSSON et al., 2017)	VIRTEX-6	P.FIXO	$R2^2$ SDF	18	1024	-	220	16	3	600	1049
(ZHOU; PENG; HWANG, 2009)	VIRTEX-4	P.FIXO	$R2^2$ SDF	16	1024	-	2256	16	8	236	1042
(DERAFSHI; FROUNCHI; TAGHIPOUR, 2010)	VIRTEX-4	P.FIXO	RADIX-2	16	1024	-	2472	10	-	100	-
(KUMAR; SAHOO; MEHER, 2019)	VIRTEX-6	P.FIXO	RADIX-2	32	1024	-	722	17	8	385	6320
(ROLIM, 2009)	VIRTEX-5	P.FLOAT	RADIX-2	32	8	39590	34129	32	-	163,1	-
(ROLIM, 2009)	VIRTEX-5	P.FLOAT	RADIX-2	64	8	93097	75810	64	-	128,9	-
(Mou; Yang, 2007)	VIRTEX-5	P.FLOAT	RADIX-2	32	1024	-	2452	16	18	150	5220
(CHEN et al., 2016)	VIRTEX-5	P.FLOAT	RADIX-2	32	8192	53558	13889	-	-	125	13329
(CHEN et al., 2016)	VIRTEX-5	P.FLOAT	RADIX-2	32	1024	53558	13889	-	-	125	1297
(BHAKTHAVATCHALU; Abdul Kareem; ARYA, 2011)	VIRTEX-4	-	RADIX-2	-	16	27742	13934	-	-	469,3	-
(ZHANG; CHEN, 2004)	-	-	RADIX-4	8	128	5740	5115	-	-	219	-

3 Descrição da Implementação

3.1 Kit de desenvolvimento

O kit de desenvolvimento escolhido foi a placa (Zynq Evaluation and Development) ZedBoard da Xilinx.

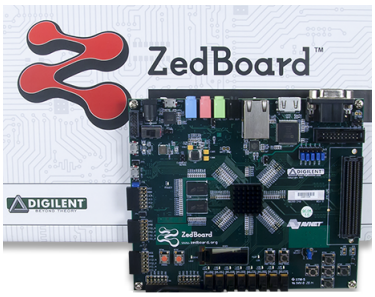


Figura 15 – Placa ZedBoard
Fonte: (XILINX, 2014b)

A placa ZedBoard é um dispositivo SoC da família Zynq-7000 da Xilinx, possui memória 512MB DDR3, 4GB SD card, programação USB-JTAG integrada, Display 128×32 OLED, saídas HDMI, 9 chaves seletoras, 7 Push-buttons, USB OTG 2.0 entre outros periféricos como mostrado na figura 16.

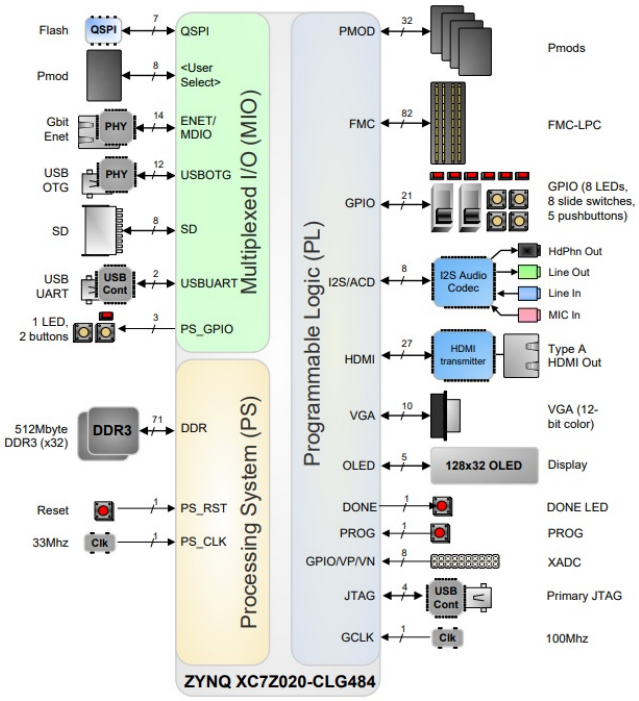


Figura 16 – Diagrama de blocos ZedBoard
Fonte: (XILINX, 2014b)

O chip utilizado é o XC7Z020-CLG484, com processador dual core ARM Cortex-A9 MPCore, RAM 256KB, uma arquitetura de FPGA Artix-7 com 85K de células lógicas, 104 BRAMs e 220 DSPs. O kit dispõem também de dois osciladores, para a parte de Sistema de Processamento (PS) temos um oscilador de 667 Mhz, e na estrutura de hardware com Lógica Programável (PL) um de 100Mhz, conforme o manual do fabricante (XILINX, 2014b).

Na tabela 7 é apresentado os recursos da ZedBoard separada em sistema de processamento e da lógica programável, conforme (XILINX, 2014a)

Tabela 7 – Recursos da ZedBoard

Sistema de processamento (PS)		**Lógica Programável (PL)**	
Núcleo do processador	ARM Dual-Core Cortex-A9 (866MHz)	Série 7 Equivalente	Artix-7
Cache L1	512KB	Células Lógicas	85K
Cache L2	256KB	Look-Up Tables (LUTs)	53200
Memória On-Chip	DDR3, DDR3L, DDR2, LPDDR2	Flip-Flops	106400
Canais DMA	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO	Block RAM (36Kb Blocks)	4,9 Mb (140)
Periféricos	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO	DSP Slices	220
Portas de interface lógica programável	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP		
Interrupções	16		

3.2 Setup experimental Phase-OTDR

Carolina Franciscangelis em sua tese "TOWARDS VIBRATION SENSING APPLICATIONS BASED ON PHASE-OTDR TECHNIQUES"(FRANCISCANGELIS, 2017), apresenta uma série de experimentos com o sensor phase-OTDR desenvolvido. Tais experimentos são utilizados para detecção de pontos de vibração para sistemas de vigilância e medição da frequência de vibração em material compósito.

Como base de dados deste trabalho, será utilizado os valores aferidos no experimento do Capítulo 4 "Vibration measurement on composite material with embedded optical fiber based on phase-OTDR". a fim de obter as frequências das vibrações utilizadas no procedimento.

Para a realização do experimento foi incorporado uma fibra óptica de 2,5 m e 125μm em um compósito de teste feito com lâminas de fibra de carbono. A perturbação foi realizada utilizando agitadores sob o compósito e os dados foram obtidos conforme mostrado na figura 17.

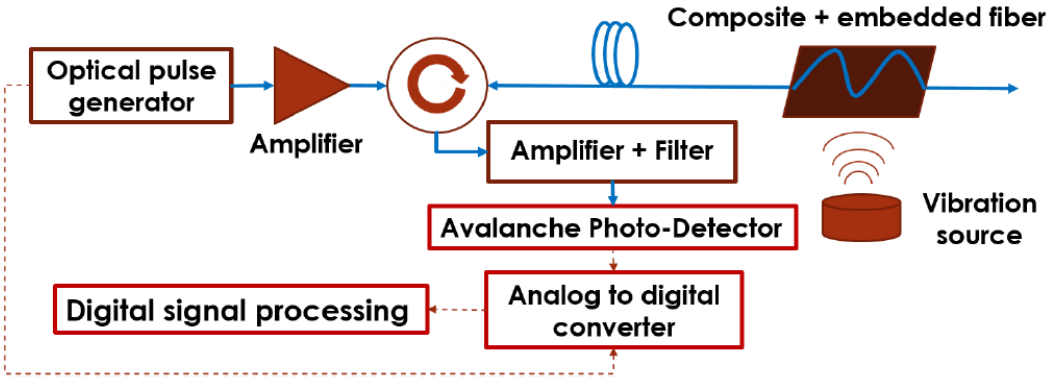


Figura 17 – Diagrama experimental
Fonte: (FRANCISCANGELIS, 2017)

A taxa de repetição dos pulsos ópticos utilizado foi de 5 kHz com a largura de 8 ns, limitando assim a resolução espacial a 80 cm. O sensor construído por Carolina Franciscangelis redireciona o sinal retroespalhado para o lado receptor da configuração que amplifica e filtra o sinal. Em seguida, um conjunto fotodetector de avalanche (APD) e uma placa de aquisição TSW14J56 armazenam até 2G de amostras com resolução de 16 bits em uma memória DDR3 e depois um FPGA da Altera realiza a leitura das amostras com taxas de transmissão que podem ser configuráveis entre 0.6Gbps à 12.5Gbps. Em particular, os dados experimentais usados neste trabalho foram adquiridos por um osciloscópio digital com 2,5 GSa/s, 300 MHz e 9 bits de resolução vertical.

Na figura 18 é mostrado a diferença quadrática dos traços phase-OTDR obtido, no qual é possível notar que a faixa perturbada da fibra está entre 45,5 e 48 m (seção da fibra embutida no compósito).

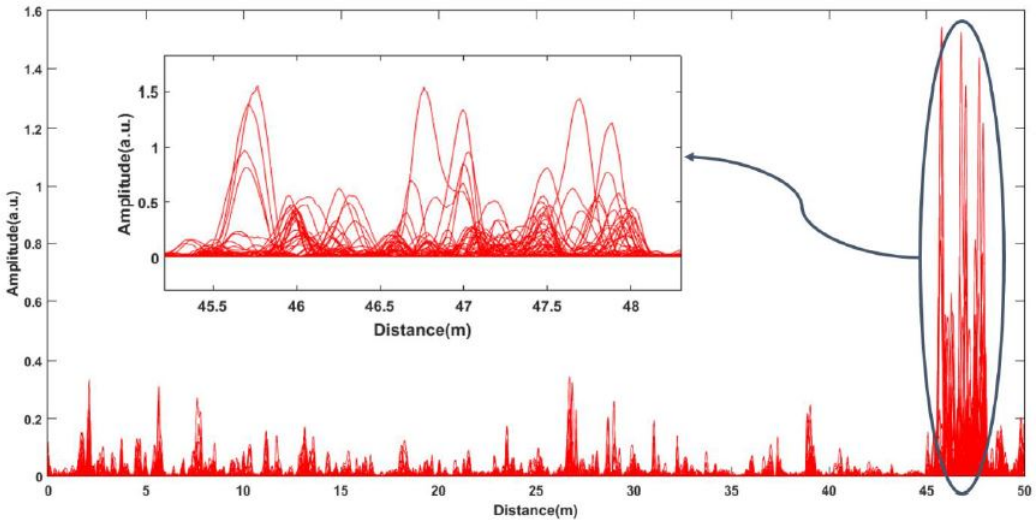


Figura 18 – Diferença quadrática dos valores
Fonte: (FRANCISCANGELIS, 2017)

O sinal no domínio do tempo pode ser obtido amostrando os traços de phase-OTDR em um ponto entre a seção perturbada. Por exemplo, a Figura 19 exibe o sinal de 1000 Hz recuperado no domínio do tempo, com a amostragem de 393 traços de phase-OTDR consecutivos.

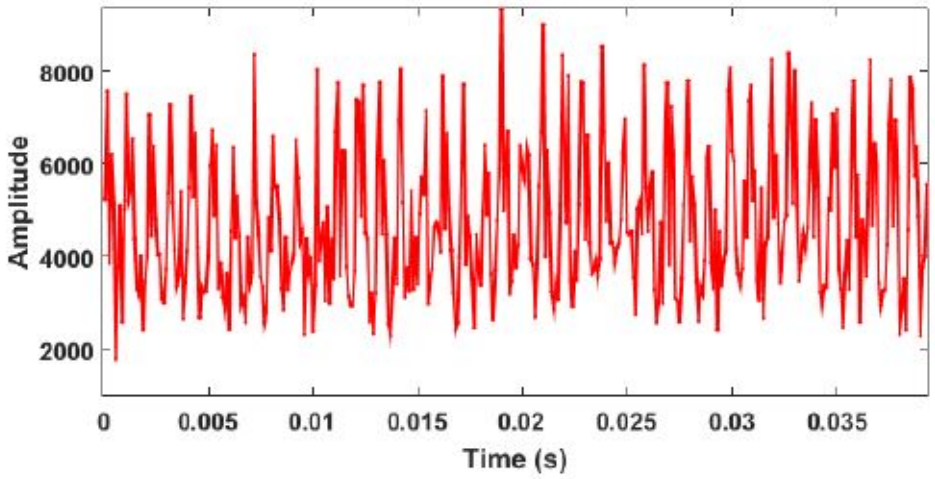


Figura 19 – Sinal de perturbação recuperado no domínio do tempo
Fonte: (FRANCISCANGELIS, 2017)

Para obter o sinal no domínio da frequência, uma transformada rápida de Fourier é aplicada ao seu sinal no domínio do tempo. Na figura 20 obtemos as transformadas de Fourier para cada frequência de vibração aplicada à placa de compósito.

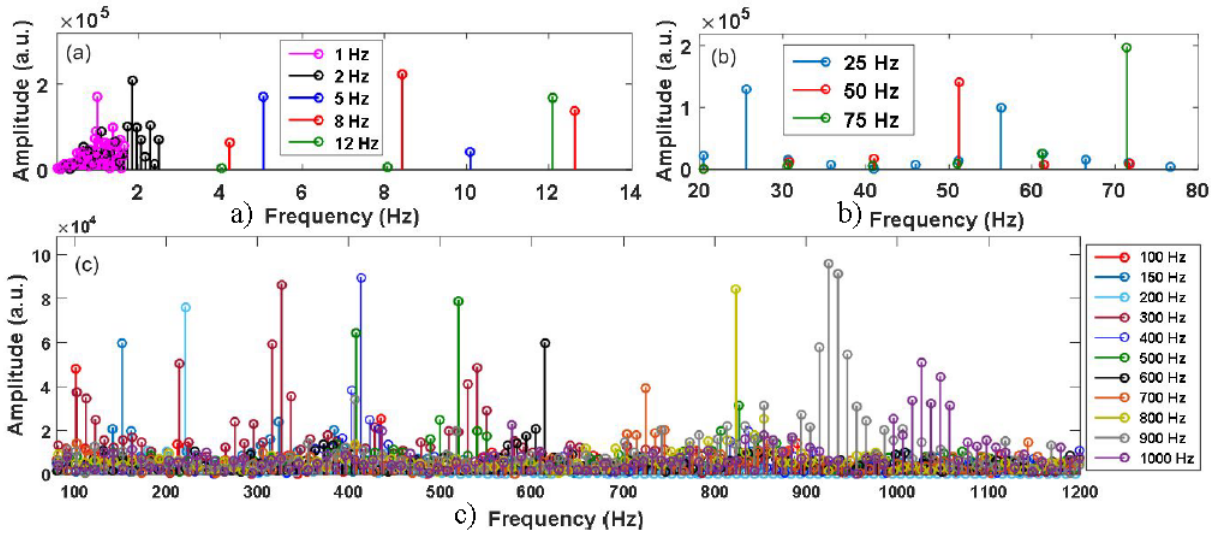


Figura 20 – Medições da frequência de vibração usando a técnica de fase-OTDR: a) 0 a 12 Hz; b)250 a 75 Hz e c) 100 a 1000 Hz
Fonte: (FRANCISCANGELIS, 2017)

A tabela 8 resume as principais características do sensor Phase-OTDR usado no trabalho de Carolina Franciscangelis (FRANCISCANGELIS, 2017).

Tabela 8 – Características do sensor phase-OTDR

Frequência de amostragem	Configurável
Resolução do conversor	16 bits
Comprimento da fibra *	19 Km
Resolução espacial **	80 cm
Frequência de interesse ***	até 60 Hz

* Com um período do pulso ótico de 200 ms e duração de 8 ns (MUNOZ et al., 2017);
** Depende do comprimento do pulso ótico. Pulsos de 8 ns permitem resolução espacial de 80 cm (MUNOZ et al., 2017);
*** Geralmente baixas frequências, de 1 a 60 Hz para aplicações aeronáuticas (KRISHNAMURTHY, 2010).

3.3 Sistema proposto

O sistema projetado neste trabalho para realização do cálculo da FFT é composto por um bloco de controle, um módulo endereçador, um módulo para o cálculo da borboleta, uma memória simple dual port RAM e uma memória ROM, como mostrado no diagrama de blocos da figura abaixo. Para melhor visualização, as entradas de cada módulo foi representado de preto e as saídas em negrito azul.

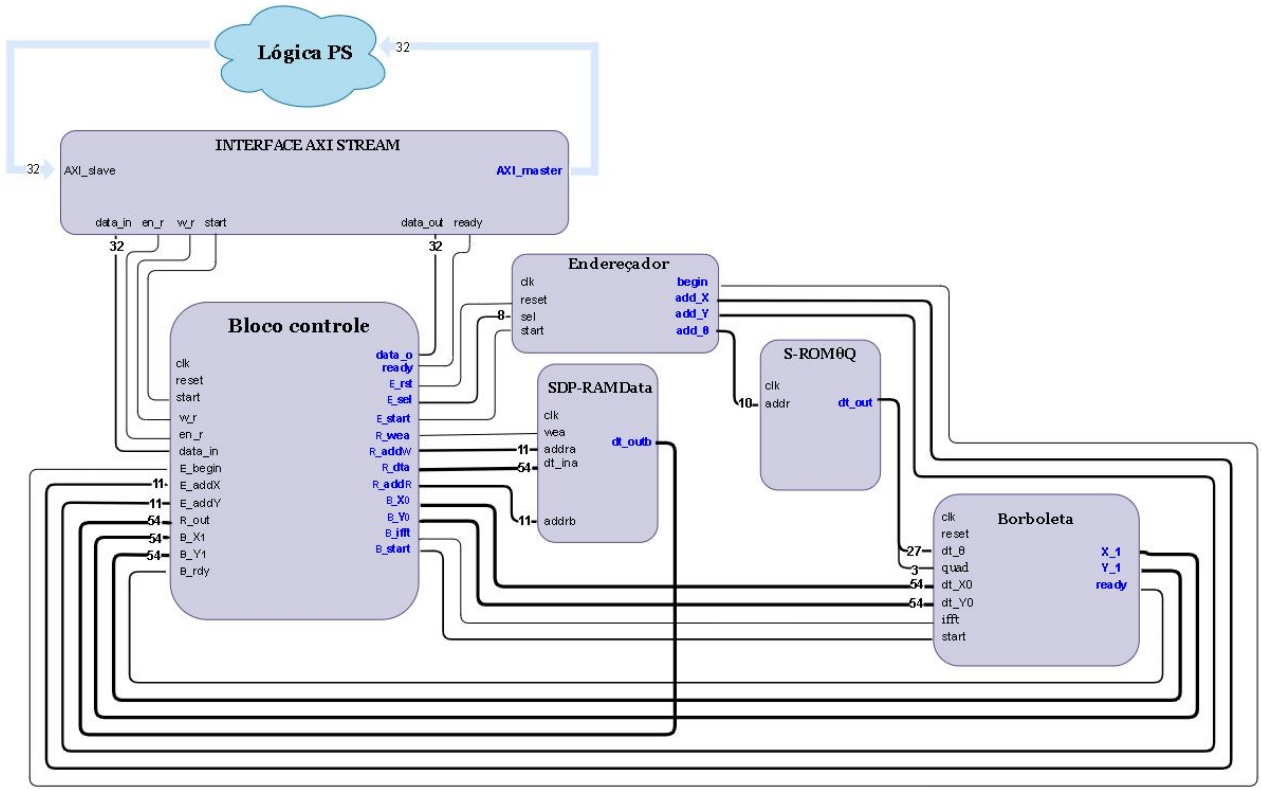


Figura 21 – Diagrama de blocos do sistema proposto
Fonte:(Autor, 2020)

A memória SDP-RAM possui 54 bits de largura, sendo 27 para a parte real e 27 para a parte imaginária, e 2048 palavras de profundidade, sendo ela utilizada para armazenar todos os dados enviados para o cálculo da FFT e todos os resultados intermediários. Na memória ROM será gravado os valores dos 1024 ângulos utilizados no cálculo da multiplicação complexa da FFT de 2048 pontos. Cada palavra possui 30 bits, sendo os 27 primeiros, o valor do ângulo em radianos e os 3 restantes, o quadrante.

O módulo endereçador funciona de forma semelhante a um contador, a cada pulso de *start* são gerados os endereços das entradas X_0, Y_0 e θ_quad utilizadas pelo módulo borboleta. As entradas θ_quad são armazenadas na memória ROM e ficam disponíveis para o módulo borboleta 2 ciclos de clock após o acionamento do *start* do endereçador, porém as entradas X_0 e Y_0 são enviadas para o bloco de controle para que o mesmo consulte na RAM estes valores, escreva-os no módulo e acione o *start*. Porém esse percurso todo pode ser realizado diversas vezes antes mesmo do módulo borboleta terminar o cálculo, já que a latência do mesmo é de 54 ciclos de clock. Essa característica é melhor explorada na subseção 3.3.5 ao utilizar um número maior de módulos borboletas podemos obter um novo valor em poucos ciclos de clock, ao acionar sequencialmente cada borboleta, os dados são calculados paralelamente gerando uma fila de resultados ao fazer este ciclo continuamente.

O bloco de controle irá comandar todo o cálculo realizado e o processo de leitura e

escrita na memória RAM durante este processo, além disso, irá receber os dados vindo via AXI stream de PS e devolver os mesmos quando o cálculo for finalizado. O diagrama de blocos da figura 21 foi proposto para a realização da FFT utilizando 1 módulo borboleta, comparado com o diagrama da figura 22, é visto que as principais diferenças entre os dois diagramas é a utilização de um banco de borboletas, que o bloco controlador do mesmo possui 7 canais a mais para a entrada de dados B_rdy e 7 canais para a saída B_start, e a inclusão de um bloco seletor para enviar para entradas B_X1 e B_Y1 do bloco de controle, as saídas $X1$ e $Y1$ do banco de borboletas. Por ser determinístico a ordem e tempo para o acionamento de cada borboleta, dado que cada módulo possui um período de trabalho fixo, as entradas de dados dos módulos borboletas são compartilhados e acionados pelo sinal B_start em períodos distintos. Nas subseções 3.3.4 e 3.3.5 será descrito com mais detalhes o projeto do bloco de controle para utilização de 1 ou 7 módulos borboletas respectivamente.

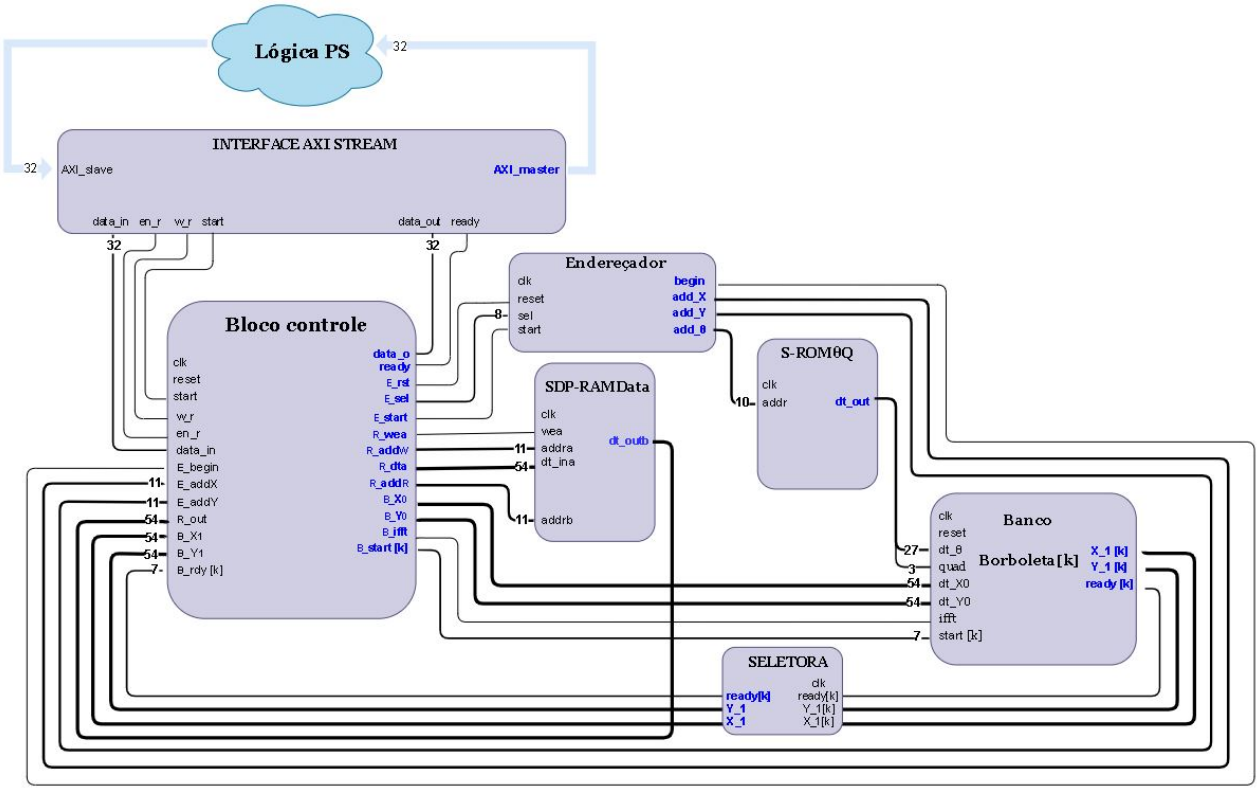


Figura 22 – Diagrama de blocos do sistema proposto para 7 borboletas
Fonte:(Autor, 2020)

3.3.1 Módulo borboleta RADIX-2

Para a criação do módulo borboleta, foi utilizado o IP somador em ponto flutuante (MUÑOZ et al., 2010b), para as operações de soma e subtração e o IP CORDIC (MUÑOZ et al., 2010a) para o cálculo da multiplicação complexa, como visto na seção 2.2.2.

Daniel M. Muñoz em seu trabalho "FPGA BASED FLOATING-POINT LIBRARY

FOR CORDIC ALGORITHMS" cria quatro módulos de CORDIC diferentes, o módulo circular que calcula senos e cossenos com ângulos de $-\pi/2$ a $\pi/2$ radianos e arco tangente com ângulos de -90 a 90, o módulo extended circular que calcula senos e cossenos com ângulos de -50 a 50, o módulo hyperbolic exp que calcula a exponencial com argumento de -1 a 1 e o módulo extended hyperbolic que calcula a exponencial com argumento de -20 a 20. Para este trabalho foi escolhido o módulo CORDIC extended circular, fazendo algumas modificações.

O módulo CORDIC extended circular ajusta o ângulo de entrada para $-\pi/2$ a $\pi/2$ e a um quadrante, esse ajuste automático utiliza DSP48. A fim de economizar recursos DSP48, os ângulos utilizados foram anteriormente calculados, ajustados e armazenados em uma memoria ROM juntamente com seus respectivos quadrantes, dessa forma, foi possível retirar o ajuste automático e inserir uma entrada quad, utilizando assim a lógica criada para o ajuste da resposta ao quadrante. Outra forma também utilizada para economia de recursos, foi a utilização dos próprios somadores em ponto flutuante do CORDIC para o calculo da soma e subtração complexa do módulo borboleta.

Para o utilização deste módulo CORDIC, foi necessário a criação de um multiplicador por constante em ponto flutuante de 27 bits. Sendo este, responsável para multiplicar as entradas do CORDIC por $\approx 0,607252935008881$, como visto anteriormente na figura 11 da seção 2.2.2.

O funcionamento da borboleta segue o diagrama lógico da figura 23. Após o acionamento da entrada *start*, é solicitado a soma da parte real das entradas X_0 e Y_0 (27 bits mais significativos) no somador 1, a subtração da parte real e imaginária das mesmas nos somadores 2 e 3. Em seguida, os *starts* dos somadores são zerados e é verificado o final da soma, caso verdade, o valor da soma real é armazenado em um registrador, e é solicitado a soma da parte imaginária das entradas, o resultado da subtração complexa vai para o multiplicador por constante, realizando assim o seguinte cálculo.

$$(X_{0REAL} - Y_{0REAL})0,60725293500888$$
$$(X_{0IMAG} - Y_{0IMAG})0,60725293500888$$

Após a verificação do termino do cálculo, o resultado da soma imaginária é registrada e o valor da parte real da multiplicação vai para a entrada Y do CORDIC e a parte imaginária para X, o *start* é acionado. Com o fim das iterações do CORDIC obtemos a multiplicação complexa da entrada pelo fator twiddle, sendo a saída $X_1CORDIC$ a parte imaginária da multiplicação complexada e a saída $Y_1CORDIC$ a parte real da mesma.

Dessa forma, é encontrado a equação de iteração na decimação em frequência, sendo a saída do módulo borboleta X_k = registro da soma real concatenado com o registro da soma imaginária e a saída Y_k = saída $Y_1CORDIC$ concatenado com a saída

$X_1 CORDIC$, conforme a Eq (Eq.2.16).

$$X_k = X_{k-1} + Y_{k-1}$$
$$Y_k = (X_{k-1} - Y_{k-1}) \quad W_{N_0}^k$$

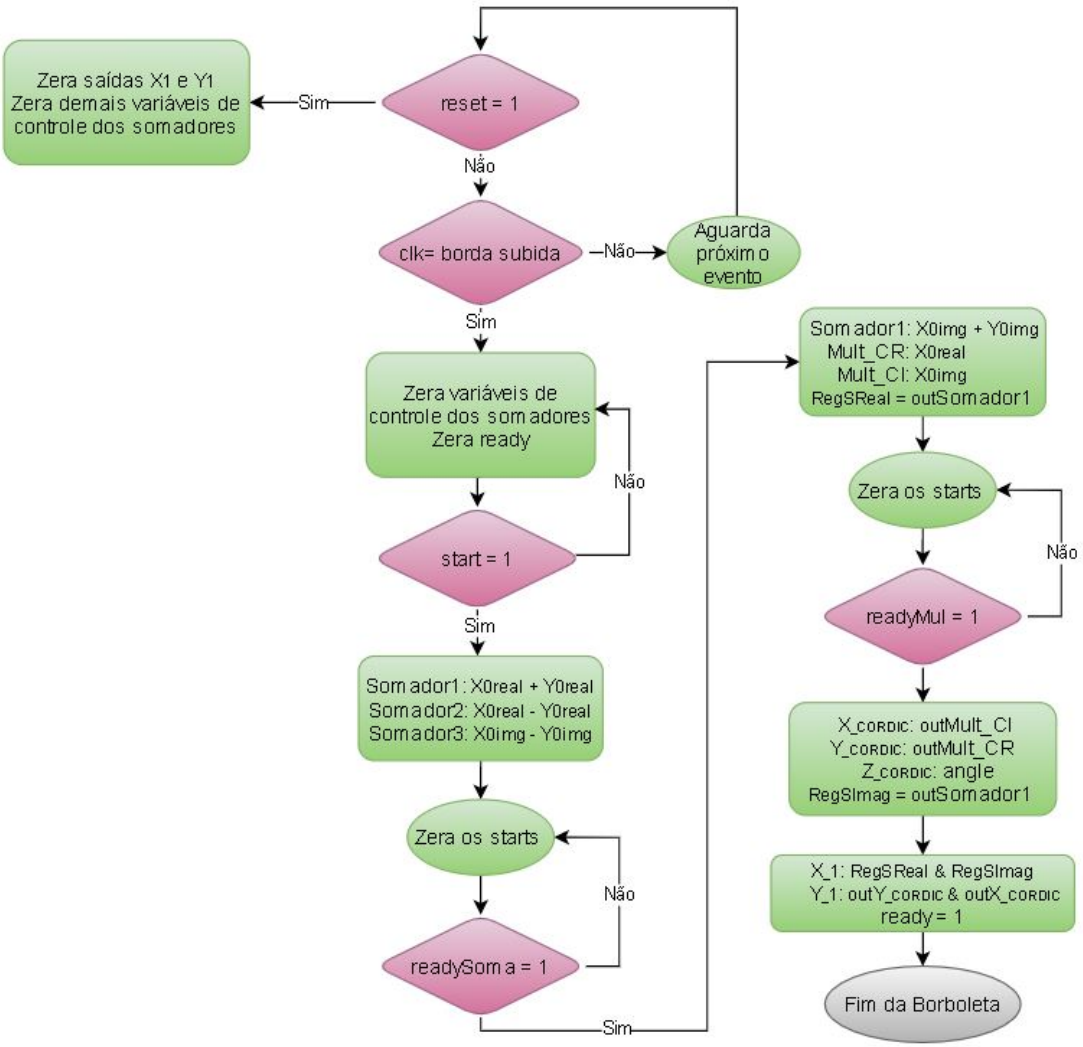


Figura 23 – Diagrama lógico do módulo borboleta. Fonte:(Autor, 2020)

A latência para o cálculo do módulo borboleta depende do número de iterações do CORDIC, por exemplo, para 25 iterações espera-se uma latência de 54 ciclos de relógio.

3.3.2 Módulo multiplicador por constante

Neste trabalho foi utilizado uma representação numérica em ponto flutuante de 27 bits, sendo 8 bits para o expoente, 18 para a mantissa e 1 para o bit de sinal, conforme mostrado na figura 25. Essa representação é interessante para hardware pelo fato de que a multiplicação da mantissa em FPGA pode ser realizadas com apenas 1 DSP, mesmo que o atual trabalho não utilize blocos DSPs, este padrão foi adotado visando aplicações futuras.

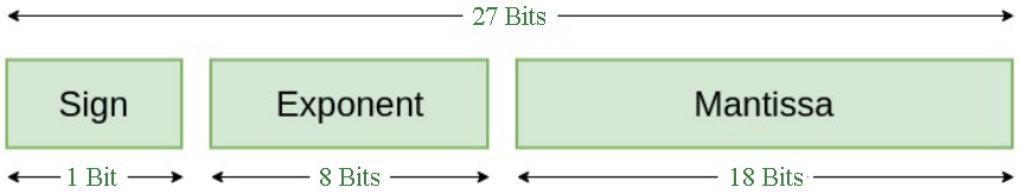


Figura 24 – Representação em ponto flutuante utilizada
Modificado de:(GEEKSFORGEEK, 2020)

Para transformar um número decimal em sua representação em ponto flutuante é feito a conversão para binário puro do número, e são realizados deslocamentos da virgula até encontrar o bit mais significativo, realizando uma normalização. São representados 256 valores diferentes para o expoente, sendo 128 para valores maiores que 1 e 128 para valores entre 0 e 1. Dessa forma, a representação do expoente de um número normalizado será feita somando o número de deslocamento a esquerda por um BIAS igual a 127, para a representação de 8 bits de expoente. De forma geral, um número é representado no padrão adotado conforme a seguinte equação.

$$(-1)^{S}.1, M.2^{E-127} \tag{3.1}$$

O valor da constante de interesse é $\approx 0,607252935008881_{10} \approx 0,10011011011101001112_{2}$, realizando a normalização temos que

$$Valor = 1,001101101110100111_2 \quad Deslocamento = -1$$
$$S_\varepsilon = 0_2; \quad E_\varepsilon = 127 - 1 = 01111110_2; \; e \; M_\varepsilon = 00110110111010011 1_2$$
$$\varepsilon \approx 0,6072529 = 00111111000110110111010011 1$$

Dessa forma, ao realizar a multiplicação de um número em ponto flutuante com a constante ε, temos

$$(-1)^{S_k}.1, M_k.2^{Exp_k} \times (-1)^{0}.1, M_\varepsilon.2^{E\varepsilon-127} = (-1)^{S_{mul}}.1, M_{mul}.2^{Exp_{mul}}$$
$$S_{mul} = S_k; \; e \; M_{mul} = 1, M_k \times 1, M_\varepsilon;$$

Nesta representação o valor de M_{mul} estará entre as seguintes faixas:

$$(1 \leq M_{mul} < 2) \tag{3.2}$$
$$(2 \leq M_{mul} < 4) \tag{3.3}$$

Para o caso de 3.2 não é preciso realizar a normalização da mantissa, dessa forma temos que:

$$Exp_{mul} = Exp_k + E\varepsilon - 127 = Exp_k - 1 \tag{3.4}$$

Para caso 3.3, é realizado um deslocamento para a direita na mantissa, deixando na forma padrão $1.M_{mul}$

$$Exp_{mul} = Exp_k + E\varepsilon - 127 + 1 = Exp_k \tag{3.5}$$

Para calcular $M_{mul} = 1.M_k \times 1.M_\varepsilon$, é realizado somas e deslocamentos em $(1.M_k)$ onde a posição de $(1.M_\varepsilon) = 1$, conforme o trecho de código abaixo.

```
mult_aux<=std_logic_vector(
            unsigned(mult)+
            shift_left(unsigned(mult),1)+
            shift_left(unsigned(mult),2)+
            shift_left(unsigned(mult),5)+
            shift_left(unsigned(mult),7)+
            shift_left(unsigned(mult),8)+
            shift_left(unsigned(mult),9)+
            shift_left(unsigned(mult),11)+
            shift_left(unsigned(mult),12)+
            shift_left(unsigned(mult),14)+
            shift_left(unsigned(mult),15)+
            shift_left(unsigned(mult),18)
            );
```

Figura 25 – Código de multiplicação das mantissas
Fonte:(Autor,2020)

Antes de aplicar a Eq 3.4, é verificado as condições de contorno para números menores que a representação utilizada (underflow). Primeiramente é verificado se o expoente de Exp_k = zero, caso verdadeiro, será verificado se a mantissa M_k é maior ou igual a $9,6886 \times 10^{-39}$, que é o menor número multiplicável por 0,6072529 e representável na notação adotada. Caso verdadeiro, a mantissa sofrerá um descolamento para a direita, compensando assim a operação de subtração por 1 no expoente. Caso falso, será considerado multiplicação por zero.

Sempre que o expoente de $Exp_k >$ zero, as condições 3.2 e 3.3 serão verificadas e as operações 3.4 e 3.5 serão realizadas, respectivamente. As condições de contorno para números muito grandes (overflow) não são verificadas já que o resultado da multiplicação pela constante ε sempre será um número menor que a entrada N_k.

3.3.3 Módulo de endereçamento

O módulo endereçador foi projetado para funcionar de forma similar a um contador, que a cada iteração gera os novos endereços para localização das variáveis X_0, Y_0 e θ.

O módulo possui 3 entradas, o *start* que fará o incremento no endereçador, o *reset* que zera todas as saídas e leva o contador para fora do ciclo de contagem, sendo o próximo estágio o primeiro estado do contador, e a entrada *sel*, que atribui valores para as variáveis internas a fim de configurar o ciclo de contagem para o endereçamento de uma FFT de 16 a 2048 pontos.

Uma FFT-RADIX2 com N pontos, possui $\log_2(N)$ níveis e cada nível realiza $N/2$ operações, como mostrado na figura 26 com $N = 16$ pontos, 4 níveis e 8 operações por nível.

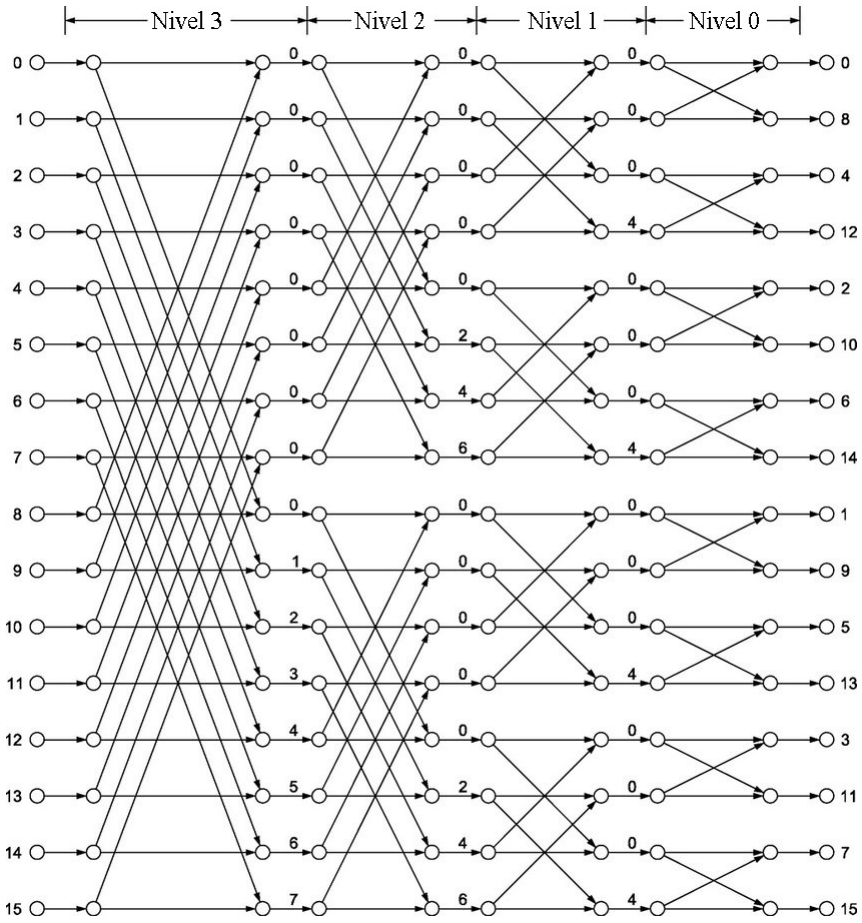


Figura 26 – Decimação na frequência para uma FFT-RADIX2 de 16 pontos.
Modificado de:(MOOKHERJEE; DEBRUNNER; DEBRUNNER, 2015)

As entradas X e Y de cada operação da borboleta possuem uma separação entre sí de 2^{nivel}, sendo o nível 3 uma separação de 8, nível 2 uma separação de 4 até o nível 0 com uma separação de 1. A cada iteração os endereços de X_k e Y_k são incrementados em 1 unidade até que o número da operação da borboleta seja igual a 2^{nivel}, quando isso acontece o novo valor de X_k passa a ser o valor passado de Y_k +1, ou seja, $X_k = Y_{k-1} + 1$ e o $Y_k = Y_{k-1} + 2^{nivel} + 1$. De forma similar é obtido o valor de θ, que sempre é iniciado com o valor 0, a cada iteração da borboleta θ é somado com $N \times 2^{-nivel}$ até que o número da operação da borboleta seja igual a 2^{nivel}, quando isso acontece o novo valor de $\theta = 0$.

A lógica citada anteriormente é implementada no diagrama da figura 45 utilizando dois contadores como referência, o contador *repet* que tem um ciclo de contagem de 1 até $N/2$, e é usado para a troca de nível. E o contador i que tem o ciclo de contagem de 0 até $2^{nivel} - 1$ e é usado para controle lógico dentro no nível

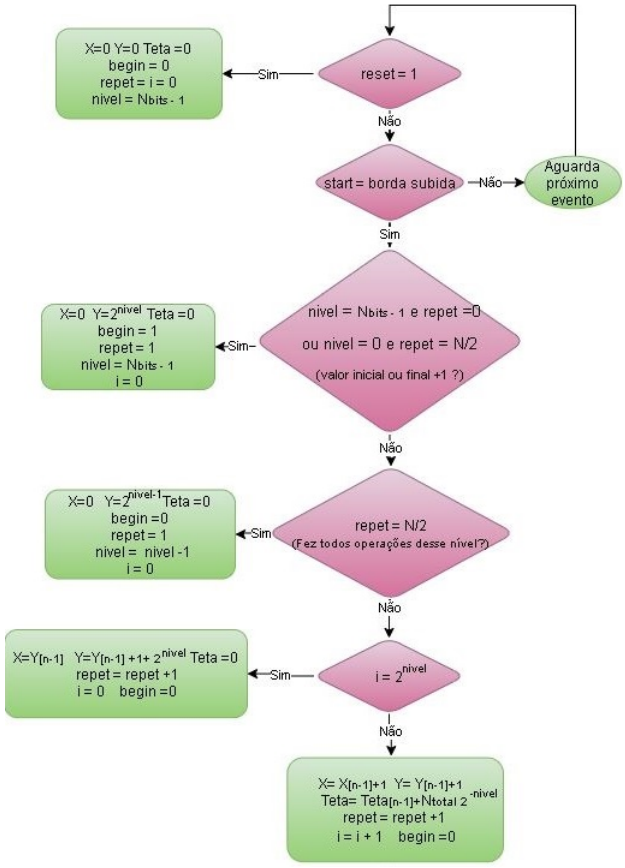


Figura 27 – Diagrama lógico do módulo endereçador por decimação na frequência.
Fonte:(Autor, 2020)

3.3.4 Bloco de controle - 1 borboleta

O bloco de controle irá receber os dados da parte PS via AXI, escreve-los na memória RAM, contá-los e iniciar os cálculos da FFT. Primeiramente é acionando o endereçador, os endereços obtidos são consultados na memoria e enviados para o módulo borboleta, após o término do cálculo, os resultados da operação são escritos novamente na RAM. Esse processo é executado de forma consecutiva até que o endereçador envie um sinal de *begin*, informando assim que já passou por todos estágios de endereço, conforme o diagrama lógico 28.

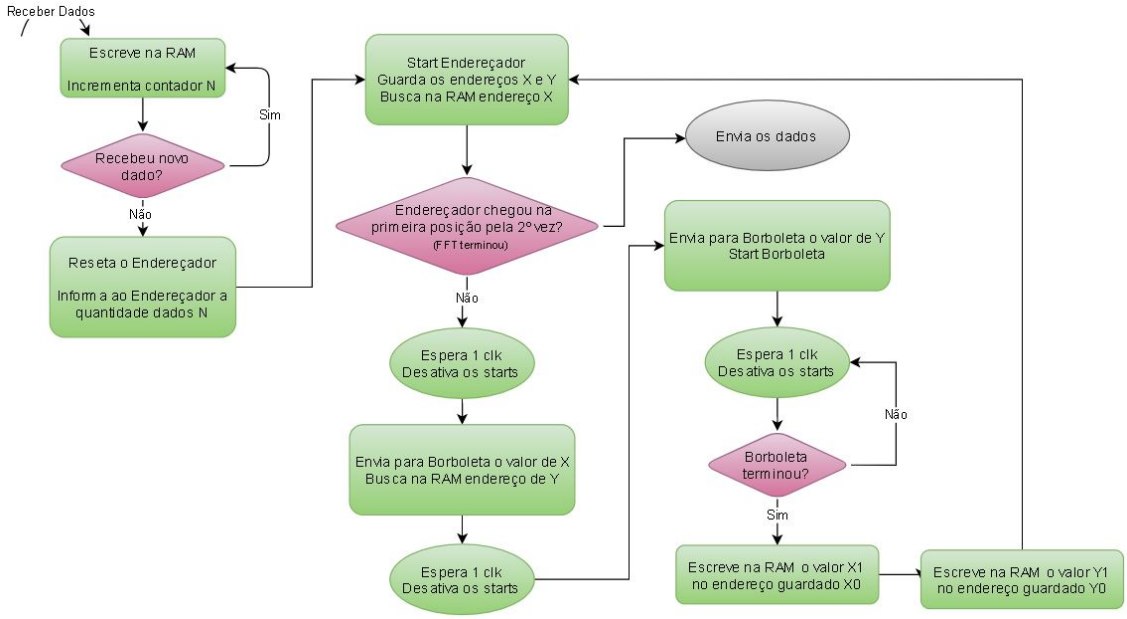


Figura 28 – Diagrama lógico do bloco de controle para cálculo da FFT com 1 borboleta
Fonte:(Autor, 2020)

Após a realização do cálculo, os endereços são lidos na ordem bit reversa e os dados são enviados novamente para a PS, como os dados possuem parte real e imaginária cada um com 27 bits e a arquitetura AXI stream aceita o envio de dados de 32 bits por vez, então é realizado uma concatenação com zeros, é enviado a parte real e depois a parte imaginária, conforme o diagrama lógico 29

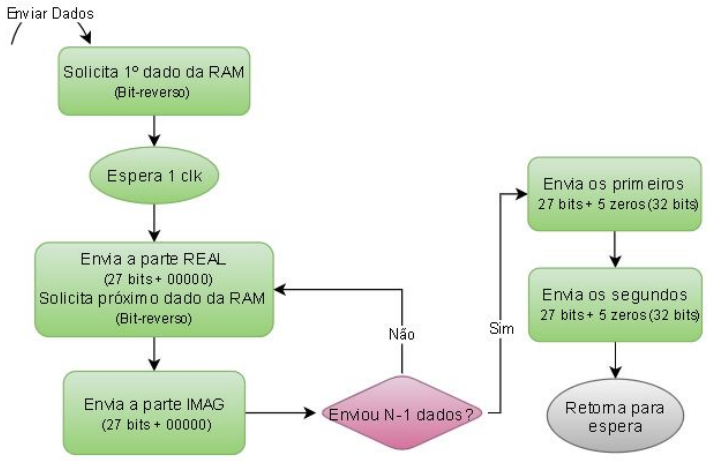


Figura 29 – Diagrama lógico do bloco de controle para leitura dos resultados
Fonte:(Autor, 2020)

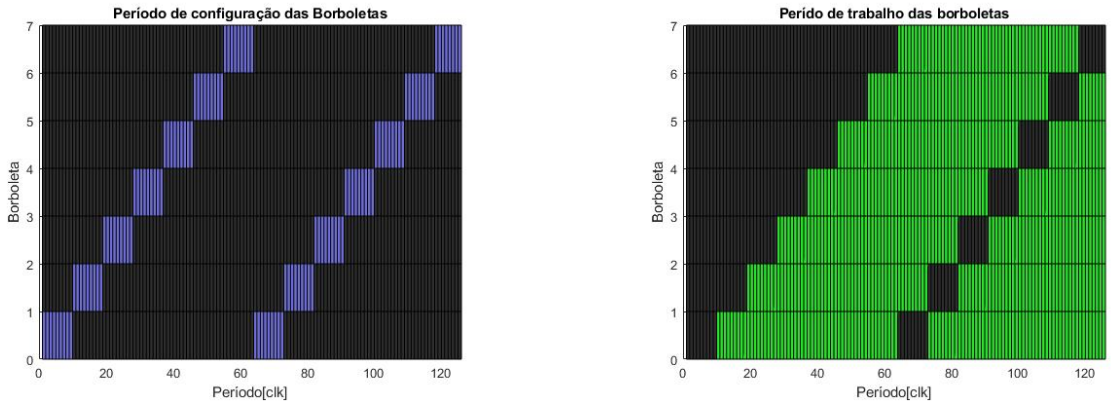
Espera-se que a latência para a configuração de uma borboleta seja de 7 ciclos de relógio. Adicionalmente, são necessários dois ciclos de relógio para armazenar os resultados do cálculo da borboleta em blocos BRAM. Dessa forma, a latência total é de 9 ciclos de relógio.

3.3.5 Bloco de controle - 7 borboletas

Levando em consideração que a latência para o cálculo de uma borboleta é de 54 ciclos de clock e que a latência para configuração da mesma é de 9 ciclos de clock, pode-se aproveitar o paralelismo do algoritmo RADIX-2 de forma a configurar sequencialmente 6 borboletas enquanto uma borboleta é usada. Assim, um total de 7 módulos borboletas foram usados neste trabalho.

A arquitetura de 7 borboletas permite que o tempo de espera para a utilização de uma borboleta seja menor e, portanto, os resultados de cada operação serão entregues ao controlador mais rapidamente. As únicas modificações entre as duas arquiteturas foi a inclusão de mais módulos borboletas, um ajuste nos sinais de controle desses módulos no bloco de controle e a criação de um bloco seletor para entregar ao bloco de controle os últimos resultados calculados.

O bloco de controle foi projetado para que durante o calculo da primeira borboleta fosse configurado outras 6 borboletas, como ilustrado na figura 30a. Cada configuração gasta um total de 9 ciclos de clocks para registrar os resultados da borboleta e envia os novos valores, como a configuração de 6 módulos é exatamente o tempo gasto para um módulo borboleta terminar seu cálculo, temos a entrega de resultados ao bloco de controle a cada 9 ciclos de clock, como ilustrado na figura 30b. Essa abordagem equivale em latência a utilização de 6 módulos borboletas em paralelo, evitando a complexidade de enviar e receber os dados ao mesmo tempo.



(a) **(b)**

Figura 30 – Período em clock de a) configuração das borboletas; b) trabalho das borboletas. Fonte:(Autor, 2021)

De acordo com o diagrama lógico 31, após o recebimento dos dados da parte PS e escrita dos mesmos na memória RAM, é acionado o endereçador, zerado o contador K e solicitado à RAM os primeiros dados para o cálculo. As primeiras 7 operações são realizadas com um delay de clocks, a fim de compensar o processo de escrita na RAM das operações seguintes. Nas operações posteriores é verificado após o acionamento de uma

borboleta se a borboleta seguinte terminou seus cálculos, caso verdadeiro, os resultados da operação são enviados para a memória RAM nos endereços armazenados anteriormente e novos valores passam a serem enviados para esta borboleta. Esse processo será repetido diversas vezes até que o endereçador envie o sinal de begin, informando assim que todos os endereços já foram lidos e a FFT terminou, iniciando o envio dos resultados para PS.

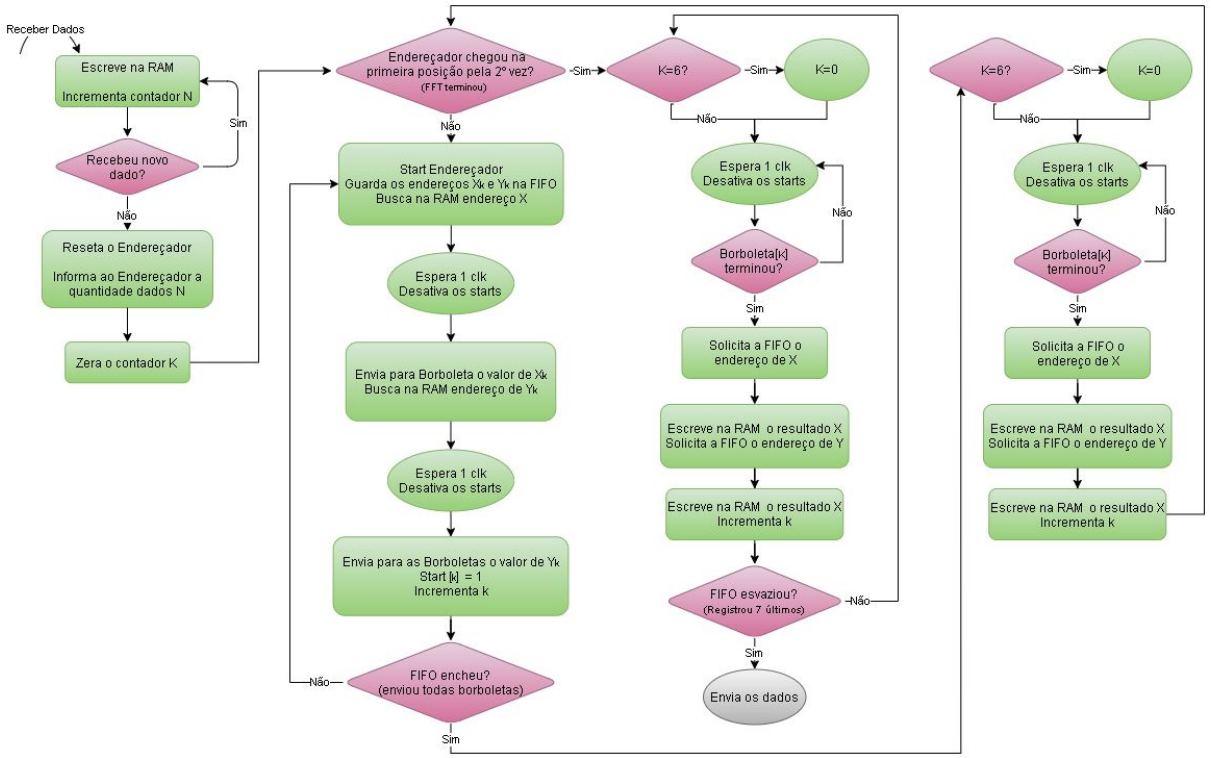


Figura 31 – Diagrama lógico do bloco de controle para cálculo da FFT com 7 borboletas
Fonte:(Autor, 2020)

O diagrama lógico foi implementado em forma de uma máquina de estados finitos (FSM), representado na figura 32. Os estados em azul são para a configuração e recebimento de dados, em verde para realização dos cálculos da FFT e em vermelho para o envio dos resultados.

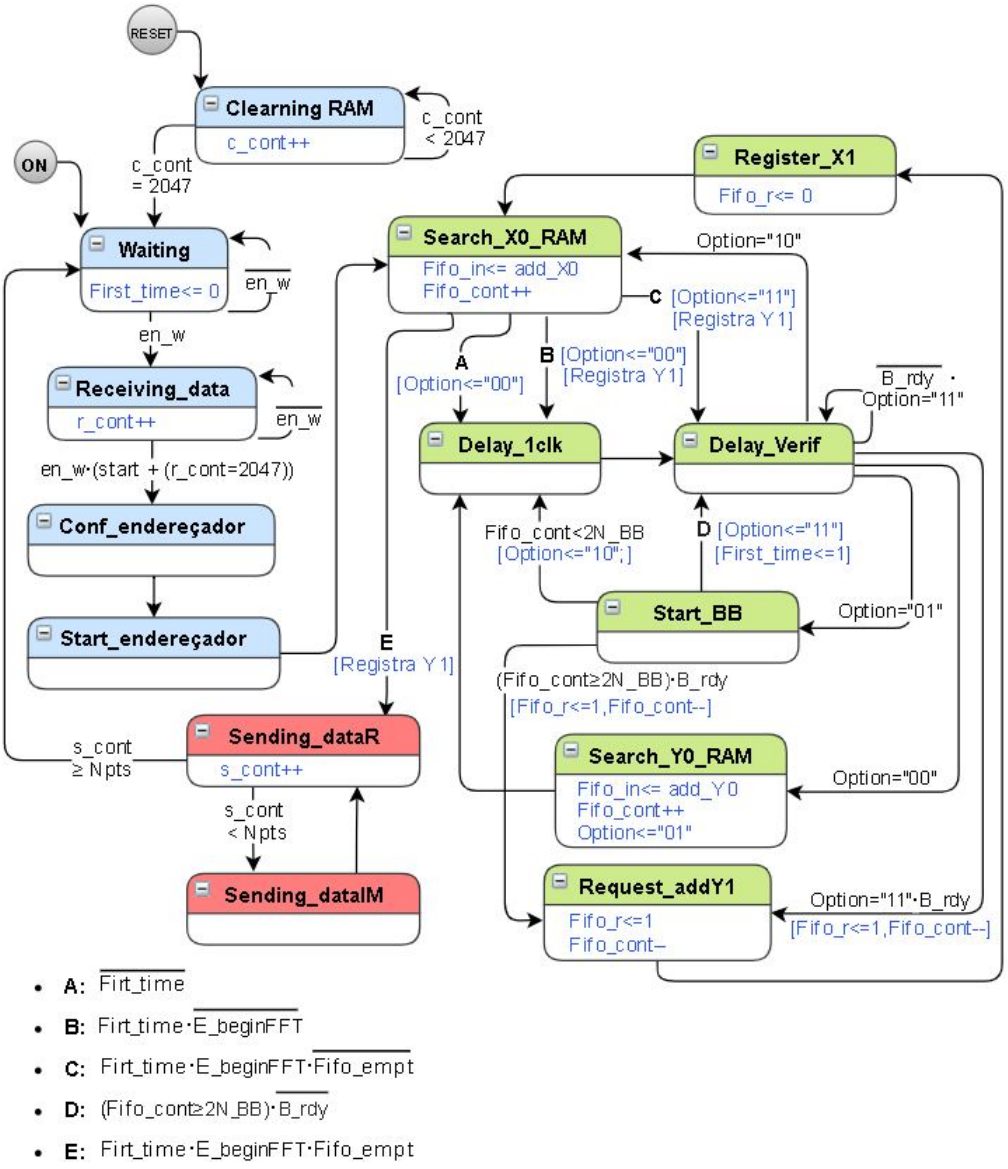


Figura 32 – FSM do bloco de controle
Fonte:(Autor, 2020)

Segue abaixo a descrição de cada estado:

- ClearningRAM
  - Zera todos os valores da memória RAM.
- Waiting
  - Aguarda a solicitação para recebimento de dados via AXI-Stream.
- Receiving_data
  - Registra os valores recebidos via AXI-Stream na memória RAM até que o Start seja acionado ou que já tenha chegado na sua capacidade máxima de armazenamento.

- Conf_endereçador
  - Realiza as configurações iniciais no módulo endereçador informando o número de dados lidos para o cálculo da FFT.
- Start_endereçador
  - Inicia o Endereçador
- Search_X0_RAM
  - Solicita à memoria RAM o dado a ser enviado pela entrada X_0 da borboleta e registra o endereço da entrada Y_0 da borboleta;
  - Caso seja os primeiro 7 cálculos, vai para estado Delay_1clk com Option = "00";
  - Caso não seja os primeiro e nem os últimos 7 cálculos, vai para estado Delay_1clk com Option = "00"e registra na RAM o resultado Y1 da ultima borboleta que terminou;
  - Caso seja os penúltimos 6 cálculos, vai para estado Delay_Verif com Option = "11"e registra na RAM o resultado Y1 da ultima borboleta que terminou;
  - Caso seja o último cálculo, vai para estado Sending_dataR e registra na RAM o resultado Y1 da ultima borboleta que terminou.
- Delay_1clk
  - Gasta 1 ciclo de clock para aguardar a leitura da memória RAM.
- Delay_Verif
  - Caso Option = "00", vai para Search_Y0_RAM;
  - Caso Option = "01", vai para Start_BB;
  - Caso Option = "10", vai para Search_X0_RAM;
  - Caso Option = "11"e o módulo borboleta não tenha terminado, permaneça em Delay_Verif;
  - Caso Option = "11"e o módulo borboleta tenha terminado, vai para Request_addY1 e solicita a FIFO o endereço para armazenar a saída X1 da borboleta.
- Search_Y0_RAM
  - Solicita à memoria RAM o dado a ser enviado pela entrada Y_0 da borboleta.
- Start_BB

  - Envia os valores para as entradas das borboletas o dá o sinal de Start para o módulo borboleta a ser utilizado;
  - Caso tenha apenas uma borboleta livre e nenhuma tenha terminado, vai para Delay_Verif com Option = "11"e seta o valor First_time para informar que os 7 primeiro cálculos já forma realizados;
  - Caso tenha apenas uma borboleta livre e alguma tenha terminado, vai para Request_addY1 e solicita a FIFO o endereço para armazenar a saída X1 da borboleta;
  - Caso seja as 6 primeiras operações, vai para Delay_1clk com Option = "10"para compensar o registro dos resultados na operações seguintes.

- Request_addY1
  - Solicita a FIFO o endereço para armazenar a saída Y1 da borboleta
- Register_X1
  - Registra na RAM a saída X1 da borboleta que terminou no endereço informado pela FIFO.
- Sending_dataR
  - Solicita da RAM os resultados em ordem bit reversa e envia via AXI-Stream a parte real;
  - Caso não tenha terminado de ler todos os valores, vai para Sending_dataIM e incrementa o contador de dados enviados;
  - Caso tenha terminado de ler todos os valores, vai para Waiting.
- Sending_dataIM
  - Envia via AXI-Stream a parte imaginária.

3.3.6 Interconexão do IP com o processador

Para realizar a troca de dados do IP criado com o processador, foi desenvolvido o block desing da figura 33, conforme especificado na subseção 2.4.2. A configuração e inicialização da transferência de dados do IP axi_dma_0 vem pela porta S_AXI_LITE do mesmo. Os dados lidos e escritos da memória DDR são enviados para o axi_dma_0 pelas portas M_AXI_MM2S e M_AXI_S2MM, e enviado para o módulo FFT pelas portas stream M_AXIS_MM2S e lidos pela porta S_AXIS_S2MM. O módulo DMA também possui duas saídas que estão ligadas a IRQ_F2P do processador ZYNQ, responsável pelas interrupções do processador quando for iniciado uma transferência de dados.

O sistema proposto foi implementado e encapsulado no IP FFT_UNB_ZED_0, no qual foi incluso uma interface AXI4-Stream escrava, no qual, recebe os valores enviados de PS, os contabiliza e os armazena em sua memória SDP_RAM. O IP recebe seu start quando tiver lido 2048 valores ou quando o ultimo valor é enviado, ou seja, quando o S00_TLAST for setado. O envio dos resultados é realizado pela interface AXI4-Stream mestre, que após levantar a flag M00_TVALID, começa enviar os dados armazenados na memória SDP_RAM a cada ciclo de clock em ordem bit reverse, enviando primeiro o valor real e depois o valor imaginário.

A figura 33 contém os principais IP's do projeto desenvolvido, ficando de fora apenas o Clocking Wizard e Processor System Reset, responsável pelo clock e reset do sistema. O block desing completo pode ser encontrado apêndice B, figura 46.

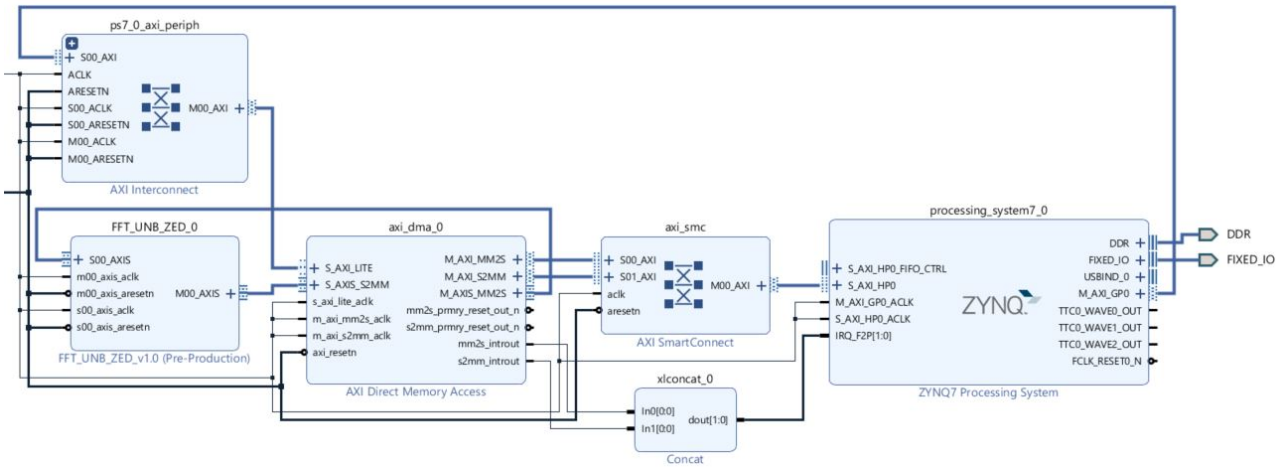


Figura 33 – Principais IP's do Block Desing
Fonte:(Autor, 2021)

Todos os arquivos, códigos VHDL, modelos em matlab e base de dados deste projeto, estão versionados no git do Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados da Universidade de Brasília (LEIA).

4 Resultados e Discussões Parciais

4.1 Validação da arquitetura

Para a validação do multiplicador por constante foram feitos algumas simulações utilizando os softwares Vivado e Matlab, resultando nos gráficos da figura 34 e 35. Para simulação do primeiro foram utilizado 1000 valores aleatório entre -100 a 100. Essa faixa foi adotada já que os valores de amplitude de entrada para obter a transformada de Fourier dos sinais phase-OTDR não possuem grande relevância, sendo representada em unidade arbitrária(a.u), como visto anteriormente na figura 18. Os gráficos gerados foram calculados no software Matlab, utilizando uma precisão double, o erro quadrático médio (MSE) obtido para essa faixa foi igual a $4,445739 \times 10^{-9}$.

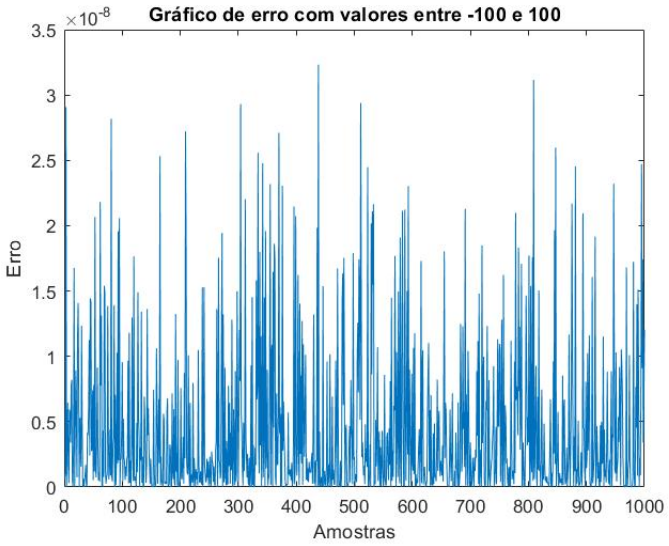


Figura 34 – Gráfico de erro do multiplicador por constante, com valores de entrada entre -100 e 100 Fonte:(Autor, 2020)

Conforme o valor da entrada do multiplicador aumenta, os erros das casas decimais tendem a serem maior dado a representação em ponto flutuante. Dessa forma, foi pensado em uma simulação de varredura dos valores de entrada, começando em 0 e indo até 10000 com passos de 0,5 e precisão double. Para valores de entrada inferiores a 930,5 é obtido erros máximos na ordem 10^{-7}, para valores inferiores a 3373 é obtido erros máximos na ordem 10^{-6} e para valores inferiores a 6746 é obtido erros máximos na ordem 10^{-5}. O erro quadrático médio obtido para essa faixa de 0 e indo até 10000 foi de $1,390304 \times 10^{-5}$, sendo de forma geral, aceito em termos de engenharia.

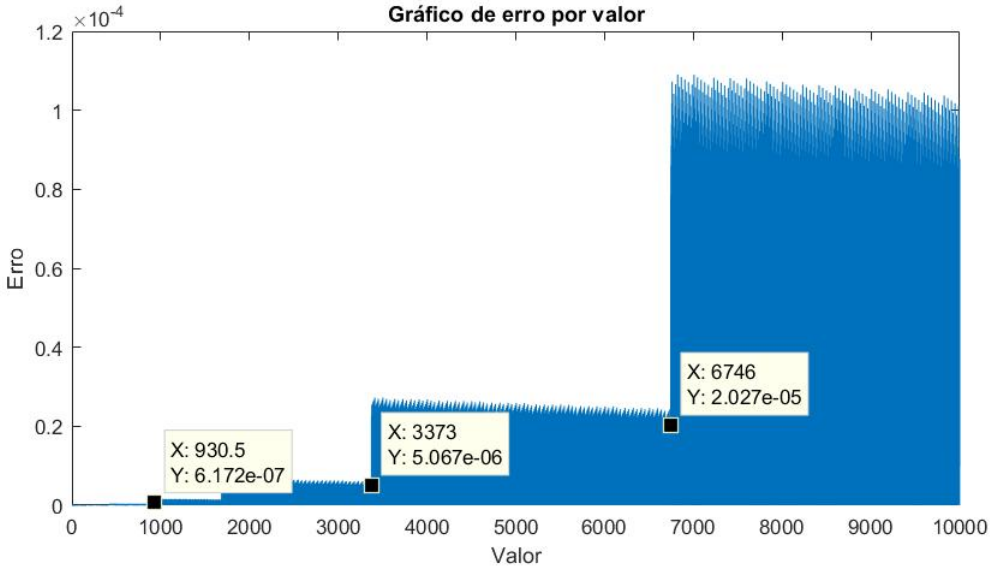
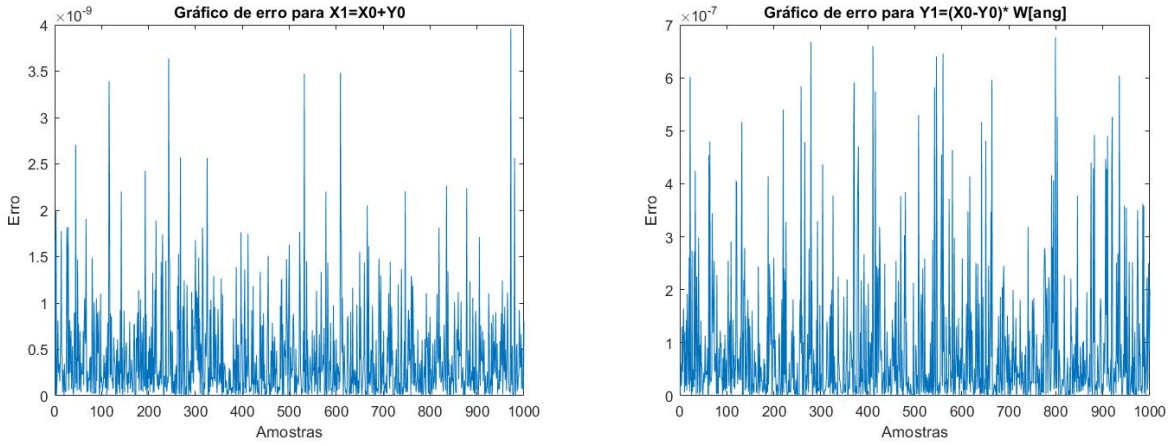


Figura 35 – Gráfico de erro do multiplicador por constante, com valores de entrada entre 0 e 10^4 Fonte:(Autor, 2020)

A validação do módulo borboleta foi realizado de forma similar ao do módulo multiplicador, gerando os gráficos de erro 36a e 36b. Sendo o primeiro referente a saída $X1$, que realiza a soma complexa em ponto flutuante das entradas $X0$ e $Y0$, e o segundo referente a saída $Y1$ que realiza o cálculo $(X0 - Y0) \times W[ANG]$.

O módulo FFT projetado poderá calcular uma transformada de um sinal de até 2048 pontos, tendo um total de 1024 valores diferentes para as entradas dt_θ e *quad*, dessa forma, é obtido que o menor ângulo que será utilizado é de $\pi/1024$. A simulação realizou o cálculo da borboleta 1000 vezes, para as entradas do módulo $X0$ e $Y0$ foram utilizados valores aleatórios entre -10 e 10, para a entrada dt_θ foi gerado valores aleatórios entre $\pi/1024$ e π, sendo que quando o valor de fosse maior que $\pi/2$, a entrada dt_θ era subtraída de π e *quad* era rotacionado para o 3º quadrante, caso contrário, o valor de dt_θ se mantinha e *quad* era rotacionado para o primeiro quadrante do circulo trigonométrico. Dessa forma, obtendo valores mais próximos aos que serão utilizados nos cálculos e atendendo o requisito do CORDIC para ângulos entre $-\pi/2$ à $\pi/2$.

O erro quadrático médio foi encontrado comparando a simulação em hardware e o algoritmo em software de dupla precisão, foi realizado o somatório das diferenças entre as partes reais com as partes imaginárias, elevado ao quadrado e dividido pela quantidade de pontos. Para o gráfico da figura 36a, foi encontrado um MSE_{X1}=$4,082997 \times 10^{-10}$, e para os valores do gráfico da figura 36b foi encontrado MSE_{Y1}=$8,419200 \times 10^{-8}$.



(a) **(b)**

Figura 36 – Gráfico de erro do módulo borboleta: a)Saída $X1$; b)Saída $Y1$.
Fonte:(Autor, 2021)

O teste de validação para o módulo endereçador foi realizado de forma diferente, tratando de um circuito lógico que deve informar ao bloco de controle a posição em memória de cada valor utilizado para o cálculo de FFT, cada erro é visto como erro de lógica e não de cálculo, sendo inaceitável no projeto. Dessa forma, foi realizado uma simulação comparando os valores de saída do módulo com as posições em vetor das variáveis X_0, Y_0 e θ em um algoritmo em Matlab, contabilizando a quantidade de erros para sinais de 16 a 2048 pontos. O módulo foi aprovado quando atingiu zero erros para todos os casos de teste.

Para validação do sistema proposto foram realizados dois tipos de testes. No primeiro tipo (tipo I), temos a construção e envio de sinais senoidais de 16 a 2048 amostras com frequências bem definidas, no segundo (tipo II) temos a utilização de sinais reais phase-OTDR obtidos dos ensaios com o sensor de fibra ótica. Sendo assim, primeiramente foi escolhido o sinal $cos(\omega_s \cdot f_1 \cdot n) + cos(\omega_s \cdot f_2 \cdot n) + cos(\omega_s \cdot f_3 \cdot n)$ para utilização nos testes de tipo I, sendo que $F_s = 16Hz$, $f_1 = 1Hz$, $f_2 = 2Hz$ e $f_3 = 3,5Hz$. Na figura 37 pode-se ver os quatro sinais utilizados, no sinal da figura 37a, n varia de 0 à 31; no sinal da figura 37b, n varia de 0 à 127; no sinal da figura 37c, n varia de 0 à 511; e o sinal da figura 37d, n varia de 0 à 2047.

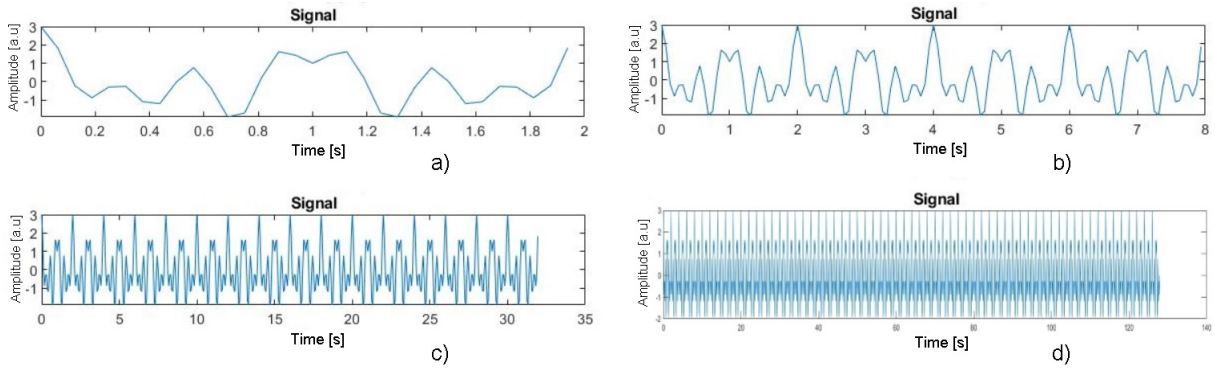


Figura 37 – Sinal de entrada para calculo da FFT: a) 32 pontos; b) 128 pontos; c) 512 pontos ; d) 2048 pontos. Fonte:(Autor, 2021)

Os sinais criados foram desenvolvidos no *SoftwareDevelopmentKit* (SDK) da *Xilinx*, e enviados para o IP desenvolvido através do DMA (*Direct Memory Acess*) e uma interface AXI Stream, garantindo que cada dado é enviado a cada ciclo de relógio. Os dados enviados e recebidos são capturados pelo *IntegratedLogicAnalyzer* (ILA) e analisados pelo software Matlab, que realiza a FFT dos dados de entrada do hardware e grafíca os resultados para análise. Na figura 38 observa-se a resposta em software e hardware para cada vetor de teste enviado.

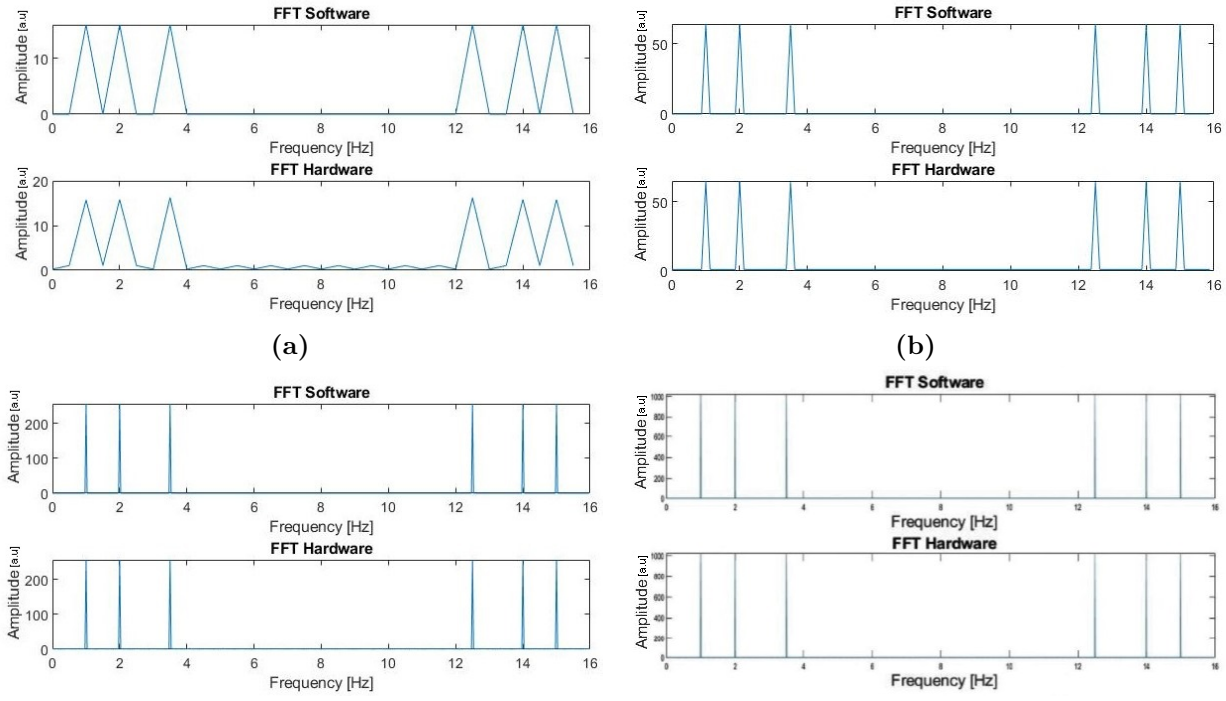


Figura 38 – Comparação software/hardware para FFT:a) 32 pontos; b) 128 pontos; c) 512 pontos ; d) 2048 pontos. Fonte:(Autor, 2021)

Os resultados entre a FFT no hardware são exatamente as mesmas que na calculada em software em quesito da localização das frequências de interesse, porém a amplitude das mesmas possuem diferença. Na tabela 9 temos os valores médios e medianos da raiz do erro quadrático para cada teste realizado, nota-se que o erro quadrático médio aumenta conforme a número de pontos do sinal, no qual a precisão de 27 bits tem sua parcela de contribuição dado o aumento do número de cálculos necessários para realização da FFT, porém, o número de pontos também aumenta a amplitude da resposta, aumentando assim o valor final da média. De forma comparativa, percebemos que a mediana do erro quadrático permanece com valores próximos e não sofre tanta alteração com o número de pontos, mostrando que a maior parte do erro quadrático da amplitude permanece baixo.

Tabela 9 – Raiz do erro quadrático para o teste de tipo I

N.PONTOS	MÉDIA	MEDIANA
32	6,94	1,04
128	14,06	1,17
512	27,70	1,82
2048	55,31	0,82

Para realizar um teste mais próximo do real, foi escolhido um *dataset* de um experimento no qual foi aplicado uma vibração de 2Hz no final da fibra, sendo a frequência de amostragem igual a 5Hz. Na figura 39 foi calculado a diferença entre os valores do período passado e presente, esse valor é elevado ao quadrado de forma a ficar mais visual as perturbações ocorridas. Como esperado, observa-se um alto destacamento na amplitude dos sinais no final da fibra.

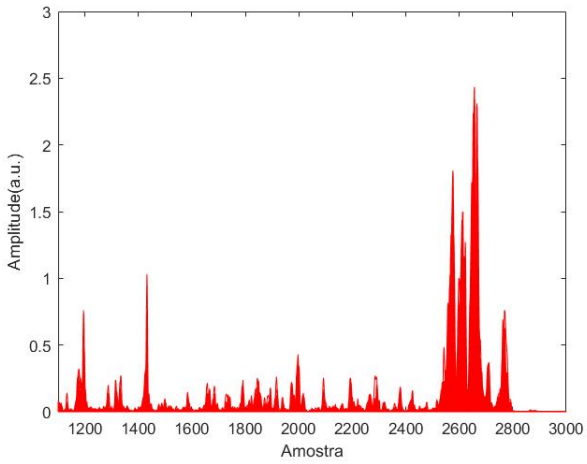


Figura 39 – Dados experimentais sensor phase-OTDR
Fonte:(Autor, 2021)

Então foi escolhido o ponto da amostra 2576 para obter os valores ao decorrer do tempo. O sinal obtido pode ser visto na figura abaixo.

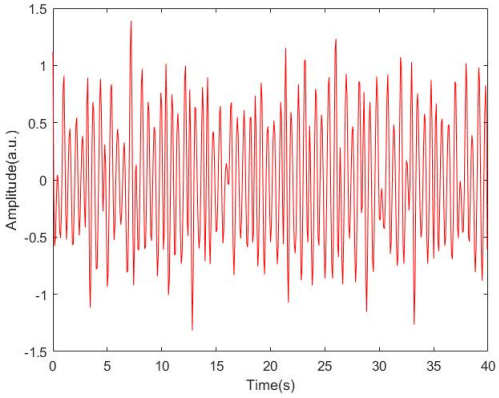


Figura 40 – Sinal da amostra 2576
Fonte:(Autor, 2021)

O sinal obtido foi ajustado para possuir 512 amostras, copiado no SDK e enviado para o módulo FFT como feito anteriormente. Na figura 41 temos o resultado do cálculo do sinal, no qual observa-se um pico próximo a frequência de 2Hz como esperado. A raiz do erro quadrático médio da amplitude foi de 12,0 e a mediana igual 1,19, sendo que esses valores no final não possuem grande relevância para esta aplicação, dado que o valor de interesse é a localização dos pontos de maior energia para a identificação das frequências de vibração da fibra óptica.

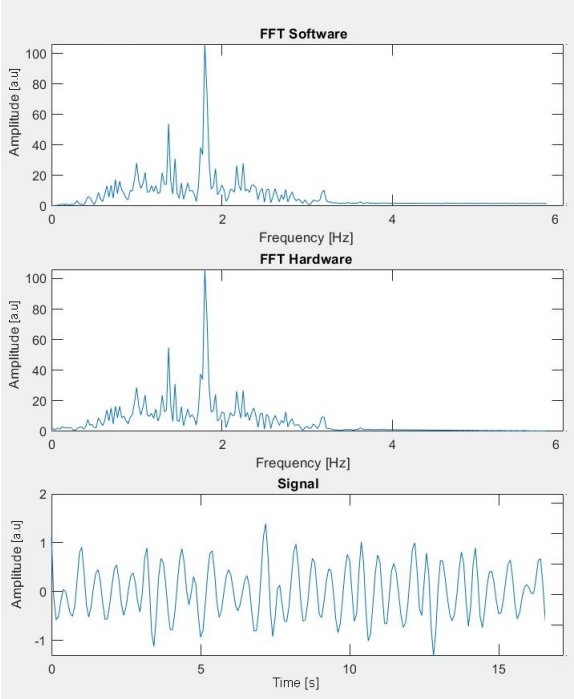


Figura 41 – Resultado do teste com sinais phase OTDR
Fonte:(Autor, 2021)

4.2 Análise de consumo de recursos

No quesito de utilização de recursos do FPGA, os módulos criados utilizam apenas Look Up Tables (LUTs) e Flip-Flops (FFs) já que os módulos multiplicador por constante e CORDIC, realizam os cálculos das multiplicações com somas e descolamentos, não utilizando assim os recursos de DSPs. Os valores dos recursos das tabelas 10 e 11 foram obtidos no software Vivado após a implementação física.

Tabela 10 – Utilização de recursos dos módulos criados

Módulo	LUTs	FFs	BRAM
Mult Const	243	78	0
Borboleta	1661	702	0
Endereçador	190	81	0
B. Controle	464	354	0,5

Na tabela 11 se apresentam os recursos que serão utilizados pelos módulos SDP-RAM, S-ROM e FIFO, os quais são blocos de Propriedade Intelectual (IP) desenvolvidos pela Xilinx e disponibilizados no catalogo de IPs do software Vivado.

Tabela 11 – Utilização de recursos dos módulos de IP

Módulo	LUTs	FFs	BRAM
SDP-RAM	0	0	3
S-ROM	0	0	1
FIFO	22	24	0,5

A figura 42 apresenta o layout da utilização de recursos da ZedBoard para a implementação do sistema proposto, conforme o block desing da figura 33, destacando na legenda os blocos com maiores consumo em área de silício.

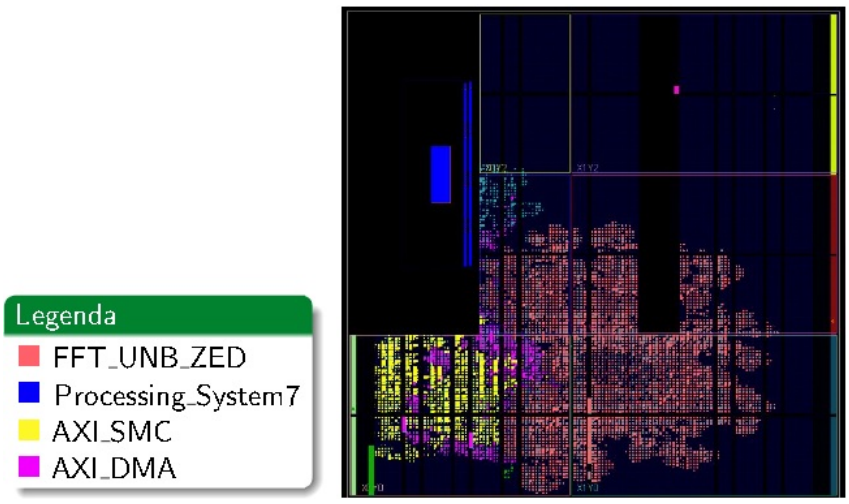


Figura 42 – Layout de utilização da arquitetura implementada
Fonte:(Autor, 2021)

A partir de simulações comportamentais foram extraídas as equações de latência do sistema para uma FFT de tamanho N_0, confirmando as estimativas de latência mencionadas no capítulo 3. Utilizando a equação 4.1 é obtido o número de ciclos gastos para realizar os cálculos da FFT, com a equação 4.2 obtemos o número de ciclos gastos para ler, calcular e enviar a FFT.

$$FFT_{clk} = \frac{9}{2} N_0 log_2(N_0) + 60 \tag{4.1}$$

$$total_{clk} = \frac{9}{2} N_0 log_2(N_0) + 3N_0 + 60 \tag{4.2}$$

O cálculo da FFT de tamanho N_0 utilizando o método Radix-2 é realizado em $N_0 \cdot log_2(N_0)$ cálculos, sendo 9/2 por demorar 9 ciclos de clock para obter os resultados de cada borboleta e que cada borboleta realiza 2 cálculos por vez, $3N_0$ para o ler N_0 valores e enviar $2N_0$ resultados, e por último temos a constante de 60 ciclos de clock para as configurações iniciais e execução da primeira borboleta (54 ciclos de clock).

Na tabela 12 é mostrado o número de ciclos de clocks gasto pelo módulo criado para o cálculo da FFT conforme o número de amostras recebidas.

Tabela 12 – Número de clocks necessários por etapa

N.PONTOS FFT	RECEBER	ENVIAR	FFT	TOTAL
2048	2048	4096	101436	107580
1024	1024	2048	46140	49212
512	512	1024	20796	22332
256	256	512	9276	10044
128	128	256	4092	4476
64	64	128	1788	1980
32	32	64	780	876
16	16	32	348	396

Observamos na tabela abaixo o resultado de tempo de execução necessário implementando a FFT Radix2 no hardware e em software. Esse valores foram obtidos utilizando a biblioteca time.h no ARM do FPGA, sendo a frequência de trabalho do processador igual a 667 MHz e do IP criado de 100 MHz.

Tabela 13 – Comparação do tempo execução do Radix2 em hardware e em software

N.PONTOS FFT	HARDWARE [μs]	SOFTWARE [μs]	GANHO	Freq HW [KH_Z]
2048	1014,36	3222,61	3,2	0,98
1024	461,40	1734,02	3,7	2,17
512	207,96	798,62	3,8	4,81
256	92,76	358,08	3,9	10,78
128	40,92	158,55	3,9	24,44
64	18,84	75,32	4,0	53,08
32	7,80	35,41	4,5	128,20
16	3,48	17,95	5,1	287,36

Devido a alta frequência de operação do processador ARM, temos um grande aumento no consumo de potência do circuito, de 0.166W para 1.901W. Dos 1,753W consumido na arquitetura utilizando ARM, 86% é destinado para o processador, como mostrado na figura 43a. Porém, a utilização de um processador para gerenciamento dos dados e comunicação com outros módulos e interfaces é bastante atraente. Caso o consumo de potência seja crucial e a aplicação não necessite de tanta performance de um processador, talvez a utilização de um processador *soft*, como o Microblaze, seja uma boa solução.

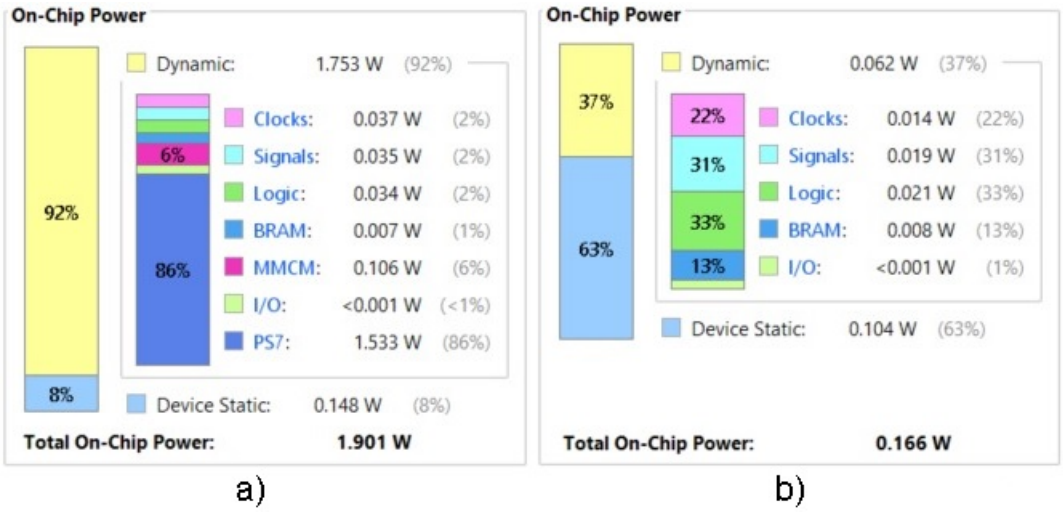


Figura 43 – Consumo de potência a)Com ARM; b)Sem ARM.
Fonte:(Autor, 2021)

De forma comparativa, a tabela 14 mostra o consumo de recursos de algumas arquiteturas similares encontradas na literatura científica. A escolha dos trabalhos listados se baseou na utilização do método Radix-2 para o cálculo da FFT, por ser capaz de processar pelo menos 1024 pontos com palavras de pelo menos 8 bits.

Tabela 14 – Comparação da utilização de recursos

REFERÊNCIA	REPRESENTAÇÃO	BITs	PONTOS	LUTs	FFs	DSPs	BRAMs	FREQ (MHz)	LATÊNCIA (CLKs)
(BARBOSA, 2018)	P.FIXO	16	1024	17021	22178	0	16	-	1728
(WANG et al., 2015)	P.FIXO	16	1024	2804	1589	16	3	298	2094
(SÁNCHEZ et al., 2008)	P.FIXO	8	1024	-	20873	0	0	186	1682
(KUMAR; SAHOO; MEHER, 2019)	P.FIXO	32	1024	-	722	17	8	385	6320
(Mou; Yang, 2007)	P.FLOAT	32	1024	-	2452	16	18	150	5220
(CHEN et al., 2016)	P.FLOAT	32	8192	53558	13889	-	-	125	13329
(CHEN et al., 2016)	P.FLOAT	32	1024	53558	13889	-	-	125	1297
(Autor, 2021)	P.FLOAT	27	2048	13146	6256	0	4,5	100	101436

Nos trabalhos de (WANG et al., 2015), (KUMAR; SAHOO; MEHER, 2019) e (Mou; Yang, 2007) é utilizado de DSPs para o cálculo das multiplicações, resultando em uma economia de LUTs e FFs. O demais projetos utilizam o algoritmo CORDIC para realizar as multiplicações complexas exigidas pelo método Radix-2, priorizando assim a economia dos recursos DSPs.

De forma geral, a arquitetura proposta neste trabalho mostrou-se mais econômica em relação a flip-flops e LUTs do que as demais arquiteturas que utilizam CORDIC para realizar o calculo da FFT, por outro lado, foi a que gastou mais ciclos de clock para isso. No trabalho (BARBOSA, 2018) foi utilizado 8 borboletas em paralelo e otimizado o CORDIC para concluir seus cálculos a cada 3 iterações; (SÁNCHEZ et al., 2008) usa uma palavra com poucos bits e aplicou uma abordagem mais paralela, cascateando estados; (CHEN et al., 2016) consegue esse feito também dedicando-se a redução de iteração do CORDIC e na utilização de 4 borboletas em paralelo. Neste trabalho, o cálculo da borboleta é realizado em ponto flutuante com 54 ciclos de clock, a abordagem de acionamento das 7 borboletas reduzem a espera de resultado de 54 ciclos para 9 ciclos. A redução do número de iterações do CORDIC e a utilização de mais borboletas trabalhando em paralelo seria interessante para diminuir a latência do projeto. Porém, mesmo com uma maior latência comparado a outras arquitetura em hardware o sistema mostrou-se mais rápido que o mesmo método implementado em software, sendo suficiente para a aplicação proposta já que os sinais de interesse são de baixa frequência.

O sistema desenvolvido por (CHEN et al., 2016) possui capacidade de realizar a FFT de uma sinal até 4 vezes maior do que realizado pela proposta deste trabalho, para obter esse resultado seria preciso utilizar mais iterações no CORDIC implementado, afim de aumentar sua precisão para poder usar um valor de ângulo menor que $\pi/1024$. Apesar de não precisar utilizar uma quantidade muito maior de LUTs e FFs, seria utilizado mais BRAMs para poder armazenar estes valores e o cálculo de cada borboleta demoraria mais ciclos para ser completado, não sendo interessante para a aplicação deste projeto, dado que a transformada de 512 traças do sinal phase-OTDR já são suficientes para a determinação da frequência de vibração, como visto anteriormente na seção 3.2.

5 Conclusões

Sendo a transformada discreta de Fourier uma das ferramentas mais amplamente usadas em processamento digital de sinal, sua implementação em sistemas em chip é de suma importância, dado as funcionalidades de gerenciamento de alto nível dos processadores e as operações extremo processamento de dados e paralelismo de um FPGA. A implementação do processamento dos dados do sensor phase-OTDR neste ambiente possibilita realizar em software a automatização, o gerenciamento dos dados coletados e a comunicação com interfaces externas, juntamente com o aceleramento dos cálculos da FFT de forma paralela em hardware.

Com objetivo de desenvolver este ambiente, foi apresentado fundamentações teóricas sobre a transformada de Fourier e suas aplicações com o algoritmo de Tukey-Cooley de base 2 através da decimação em tempo e em frequência. Foi apresentado também o algoritmo CORDIC para o cálculos trigonométricos, e sua aplicação para o calcular as multiplicações complexas exigidas, utilizando somas e deslocamentos. Depois tivemos o estudo do protocolo de comunicação AXI4 adotado pela Xilinx, sendo ele necessário para realizar a comunicação entre PS e PL, e entre IPs. Por ultimo, foi tratado dos sistemas em chip (SoC) e suas vantagens em sistemas embarcados.

Após tratar da base do projeto, foi escolhido o kit de desenvolvimento e falado sobre a base de dados experimental. Com a explicação da base de dados experimental e a técnica utilizada para obter os valores das frequências, foi proposto um sistema que utiliza 5 blocos principais, sendo 3 deles implementados no projeto e outros 2 IPs da Xilinx. Dessa forma, é apresentado a lógica de implementação dos módulos do bloco de controle, endereçador e borboleta para 2 arquiteturas distintas. A primeira a ser desenvolvida é a FFT radix-2 utilizando 1 borboleta para o cálculo da transformada. Para a segunda é proposto modificações no bloco de controle para executar 7 borboletas no cálculo da FFT, a fim de agilizar a entrega de dados para o bloco de controle de 54 ciclos de clock para 9 ciclos de clock.

Os módulos implementados utilizaram apenas flip-flops, look up tables e BRAM devido ao uso do algoritmo CORDIC e a criação do módulo multiplicador por constante. Em geral, o consumo de recurso do sistema proposto utilizando 7 borboletas foi de 13146 LUTs e 6256 FFs e 4,5 BRAM. Consumindo 1,753W de potência dinâmica e com uma latência total de 396 ciclos de clock para 16 pontos da FFT a 107580 ciclos de clock para 2048 pontos da FFT. Foi visto que ao operar em sua frequência máxima, entrega os resultados em média 4 vezes mais rápido que processadores ARM utilizando a mesma arquitetura.

Em relação ao consumo de recurso, o sistema desenvolvido consome menos em comparação aos trabalhos correlatos, mas possui uma latência mais alta, gastando mais ciclos de clock para entregar os resultados. Porém, possui uma arquitetura totalmente em ponto flutuante e mostrou-se eficiente em relação a identificação das frequências de vibração em sensores phase-OTDR, como proposto.

Conforme os resultados de latência do sistema proposto apresentado na tabela 13, é possível utilizar uma frequência de amostragem de até 2 MHz para 2048 pontos, de forma que o tempo para capturar esse vetor de dados seria maior que o tempo para calcular a FFT de 2048 pontos. Sendo assim, o projeto desenvolvido poderia identificar frequências de até 1 MHz de forma continua e obedecendo o teorema de Nyquist. Devido ao baixo consumo de recursos de hardware e com consumo de potência dinâmica de apenas 0.062 W sem ARM, o sistema desenvolvido permite embarcar a solução proposta em aplicações com restrições de consumo energético e de portabilidade(tamanho e peso), por exemplo, em drones, satélites e submarinos.

Além de identificar os pontos e frequências de vibração, a classificação da natureza dessa pertubação pode ser interessante. Como proposta para trabalhos futuros pode-se explorar a implementação em hardware de uma rede neural artificial alimentada por uma FFT, a fim de classificar padrões de vibração, ou para resolução de sistemas que envolvem equações diferenciais parciais, assim como apresentado na figura 44.

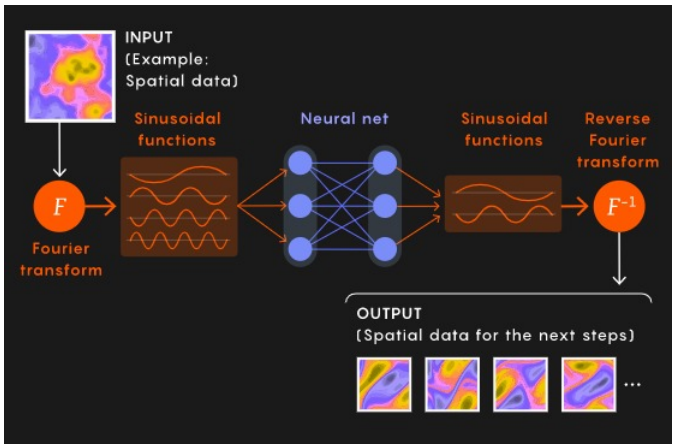


Figura 44 – Trabalhos futuros
Fonte:(MINARI, 2021)

Um aspecto importante para a continuação deste trabalho é a criação de um gerador de código VHDL a partir de uma ferramenta de modelagem de alto nível, por exemplo, desenvolvida em Python. Dessa forma, poderia se estimar a latência e o consumo de recursos em função do número de borboletas e de iterações do CORDIC. Outro parâmetro de projeto que pode ser explorado nessa ferramenta é a precisão numérica, variando o tamanho da mantissa e/ou do expoente a fim de garantir a precisão necessária para uma aplicação específica.

Referências

ALTERA. What is an soc fpga? 2014. Disponível em: <https://www.intel.co.jp/content/dam/www/programmable/us/en/pdfs/literature/ab/ab1_soc_fpga.pdf>. Citado na página 32.

ARM. *AMBA 4 AXI4-Stream Protocol*. [S.l.], 2010. 42 p. Disponível em: <https://developer.arm.com/documentation/ihi0051/a/Interface-Signals/Transfer-signaling/Handshake-process?lang=en>. Citado na página 34.

BARBOSA, C. S. IMPLEMENTAÇÃO DO ALGORITMO RADIX-2 PARA CÁLCULO DA FFT EM FPGA. 2018. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/16425>. Citado 2 vezes nas páginas 36 e 66.

BARRIAS, A.; CASAS, J. R.; VILLALBA, S. A review of distributed optical fiber sensors for civil engineering applications. *Sensors (Switzerland)*, v. 16, n. 5, 2016. ISSN 14248220. Citado na página 17.

BHAKTHAVATCHALU, R.; Abdul Kareem, N.; ARYA, J. Comparison of reconfigurable FFT processor implementation using CORDIC and multipliers. *2011 IEEE Recent Advances in Intelligent Computational Systems, RAICS 2011*, n. 1, p. 343–347, 2011. Citado na página 36.

CHEN, J. et al. Configurable floating-point FFT accelerator on FPGA based multiple-rotation CORDIC. *Chinese Journal of Electronics*, v. 25, n. 6, p. 1063–1070, 2016. ISSN 10224653. Citado 2 vezes nas páginas 36 e 66.

DERAFSHI, Z. H.; FROUNCHI, J.; TAGHIPOUR, H. A high speed FPGA implementation of a 1024-point complex FFT processor. *2nd International Conference on Computer and Network Technology, ICCNT 2010*, p. 312–315, 2010. Citado na página 36.

FRANCISCANGELIS, C. Towards Vibration Sensing Applications Based on Phase-Otdr Techniques . 2017. Citado 6 vezes nas páginas 16, 17, 38, 39, 40 e 41.

GANAPATHIRAJU, A. et al. A comparative analysis of fft algorithms. 02 1999. Citado 2 vezes nas páginas 23 e 24.

GEEKSFORGEEK. *IEEE Standard 754 Floating Point Numbers*. 2020. Disponível em: <https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/>. Citado na página 46.

INGEMARSSON, C. et al. Efficient FPGA Mapping of Pipeline SDF FFT Cores. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 25, n. 9, p. 2486–2497, 2017. ISSN 10638210. Citado na página 36.

Johnson, Jeff. *Using the AXI DMA in Vivado*. 2014. Disponível em: <https://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html/>. Acesso em: 07 maio 2021. Citado 2 vezes nas páginas 35 e 36.

KRISHNAMURTHY, T. Frequencies and flutter speed estimation for damaged aircraft wing using scaled equivalent plate analysis. In: . [S.l.: s.n.], 2010. ISBN 978-1-60086-961-7. Citado na página 41.

KUMAR, G. G.; SAHOO, S. K.; MEHER, P. K. 50 years of fft algorithms and applications. 2019. Citado 3 vezes nas páginas 16, 36 e 66.

LATHI, B. *Sinais e Sistemas Lineares - 2.ed.* Bookman, 2007. ISBN 9788560031139. Disponível em: <https://books.google.com.br/books?id=ySxoo2TVeeYC>. Citado 5 vezes nas páginas 16, 21, 24, 25 e 26.

MEHER, P. K. et al. 50 years of CORDIC: Algorithms, architectures, and applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, v. 56, n. 9, p. 1893–1907, 2009. ISSN 15498328. Citado 3 vezes nas páginas 26, 27 e 28.

MINARI, G. *Redes neurais podem ajudar a resolver as equações mais difíceis do mundo*. 2021. Disponível em: <https://canaltech.com.br/inovacao/redes-neurais-podem-ajudar-a-resolver-as-equacoes-mais-dificeis-do-mundo-184218/>. Citado na página 68.

MOOKHERJEE, S.; DEBRUNNER, L.; DEBRUNNER, V. A low power radix-2 fft accelerator for fpga. 11 2015. Citado na página 48.

Mou, S.; Yang, X. Design of a high-speed fpga-based 32-bit floating-point fft processor. In: *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*. [S.l.: s.n.], 2007. v. 1, p. 84–87. Citado 2 vezes nas páginas 36 e 66.

MUNOZ, D. M. et al. Low latency disturbance detection using distributed optical fiber sensors. *Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control, ICNSC 2017*, p. 372–377, 2017. Citado 2 vezes nas páginas 17 e 41.

MUÑOZ, D. M. et al. FPGA BASED FLOATING-POINT LIBRARY FOR CORDIC ALGORITHMS Departments of 1 Mechanical Engineering , 2 Computer Science University of Brasilia 70910-900 , Brasilia , D . F ., Brazil. p. 55–60, 2010. Citado na página 43.

MUÑOZ, D. M. et al. Tradeoff of FPGA design of a floating-point library for arithmetic operators. *Journal of Integrated Circuits and Systems*, v. 5, n. 1, p. 42–52, 2010. ISSN 18071953. Citado na página 43.

NETO, J. C. d. O. AnÁlise do protocolo de comunicaÇÃo advanced extensible interface utilizado em sistemas de um chip. p. 73, 2017. Disponível em: <https://www.monografias.ufma.br/jspui/handle/123456789/1591>. Citado na página 35.

NEVES, F. *CORDIC – Introdução*. 2016. Disponível em: <https://www.embarcados.com.br/cordic-parte-1/>. Citado na página 30.

NGUYEN, N. H. et al. A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms. *Microprocessors and Microsystems*, Elsevier B.V., v. 60, p. 96–106, 2018. ISSN 01419331. Disponível em: <https://doi.org/10.1016/j.micpro.2018.04.003>. Citado na página 36.

OPPENHEIM, A.; SCHAFER, R. *Processamento Em Tempo Discreto De Sinais*. PEARSON BRASIL, 2013. ISBN 9788581431024. Disponível em: <https://books.google.com.br/books?id=g72vnQEACAAJ>. Citado 5 vezes nas páginas 20, 22, 25, 26 e 27.

OPPENHEIM, A.; WILLSKY, A.; NAWAB, S. *Sinais e sistemas*. Prentice-Hall, 2010. ISBN 9788576055044. Disponível em: <https://books.google.com.br/books?id=ZOg9bwAACAAJ>. Citado na página 21.

PRIETO, J.; ALEJANDRO, M. Diseño Y Construcción De Un Sensor Phi-Otdr Para Detectar Perturbaciones En. 2019. Citado na página 17.

ROLIM, A. U. A. *Desenvolvimento de uma FFT utilizando ponto flutuante para FPGA*. Tese (Doutorado), 2009. Disponível em: <https://repositorio.ufpe.br/handle/123456789/1765>. Citado na página 36.

ROSSETTO, J. F. Sensores distribuidos utilizando efeitos não-lineares em fibras opticas para aplicação em estruturas inteligentes. 2004. Disponível em: <http://repositorio.unicamp.br/jspui/handle/REPOSIP/260254>. Citado na página 16.

SÁNCHEZ, M. A. et al. Implementing FFT-based digital channelized receivers on FPGA platforms. *IEEE Transactions on Aerospace and Electronic Systems*, v. 44, n. 4, p. 1567–1585, 2008. ISSN 00189251. Citado 2 vezes nas páginas 36 e 66.

SANTOS, H. M. D. CONCEÇÃO DE UM TUTORIAL DE INICIAÇÃO AO CO-PROJETO DE HARDWARE/SOFTWARE. 2015. Disponível em: <https://core.ac.uk/download/pdf/47142679.pdf>. Citado na página 32.

UZUN, I. et al. Towards a general framework for an fpga-based fft coprocessor. In: *Seventh International Symposium on Signal Processing and Its Applications, 2003. Proceedings.* [S.l.: s.n.], 2003. v. 1, p. 617–620 vol.1. Citado 3 vezes nas páginas 22, 23 e 24.

VOLDER, J. The CORDIC computing technique. *Proceedings of the Western Joint Computer Conference, IRE-AIEE-ACM 1959*, p. 257–261, 1959. Citado 2 vezes nas páginas 27 e 28.

VOLDER, J. The birth of CORDIC. *The Journal of VLSI Signal Processing*, n. 1, p. 101–105, 2000. Disponível em: <http://www.springerlink.com/index/X40006671445XL56.pdf>. Citado 2 vezes nas páginas 27 e 29.

VULLIEZ, P. Fiber-optic sensing: distributed fiber-optic sensing solves real-world problems. 2013. Disponível em: <https://www.laserfocusworld.com/fiber-optics/article/16560884/fiberoptic-sensing-distributed-fiberoptic-sensing-solves-realworld-problems>. Citado 2 vezes nas páginas 17 e 18.

WANG, Z. et al. A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 23, n. 5, p. 973–977, 2015. ISSN 10638210. Citado 2 vezes nas páginas 36 e 66.

WIKIPéDIA. *CORDIC*. 2020. Disponível em: <https://pt.wikipedia.org/wiki/CORDIC#Refer%C3%AAncias>. Citado 3 vezes nas páginas 27, 28 e 31.

XILINX. *AXI Reference Guide*. [S.l.], 2011. 82 p. Disponível em: <https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf>. Citado na página 33.

XILINX. Zynq-7000 soc, product selection guide. 2014. Disponível em: <https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf>. Citado na página 38.

XILINX. *(Zynq Evaluation and Development) Hardware User's Guide*. [S.l.], 2014. 38 p. Disponível em: <http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf?_ga=2.252203018.265037859.1605642073-791013538.1598733782>. Citado 2 vezes nas páginas 37 e 38.

XILINX. *Zynq-7000 All Programmable SoC*. [S.l.], 2017. 1845 p. Disponível em: <https://wiki.bu.ost.ch/infoportal/_media/embedded_systems/zynq7000/zynq_technicalreferencemanual.pdf>. Citado 2 vezes nas páginas 32 e 33.

ZHANG, G.; CHEN, F. Parallel FFT with CORDIC for ultra wide band. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, v. 2, n. 3, p. 1173–1177, 2004. Citado na página 36.

ZHOU, B.; PENG, Y.; HWANG, D. Pipeline FFT Architectures Optimized for FPGAs. *International Journal of Reconfigurable Computing*, v. 2009, p. 1–9, 2009. ISSN 1687-7195. Citado na página 36.

Apêndices

APÊNDICE A – Primeiro Apêndice

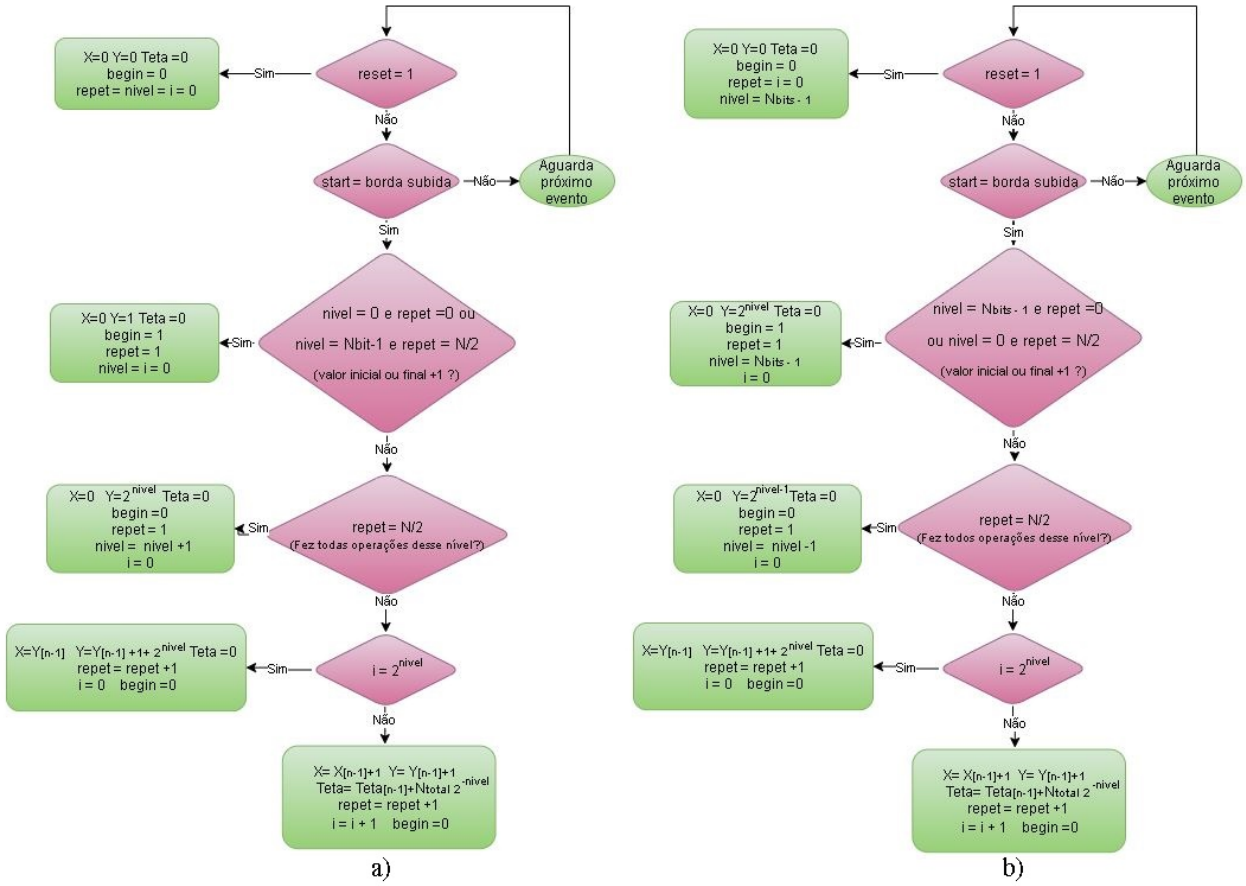


Figura 45 – Diagrama lógico do módulo endereçador: a)Decimação no Tempo ; b)Decimação na Frequência. Fonte:(Autor, 2020)

APÊNDICE B – Segundo Apêndice

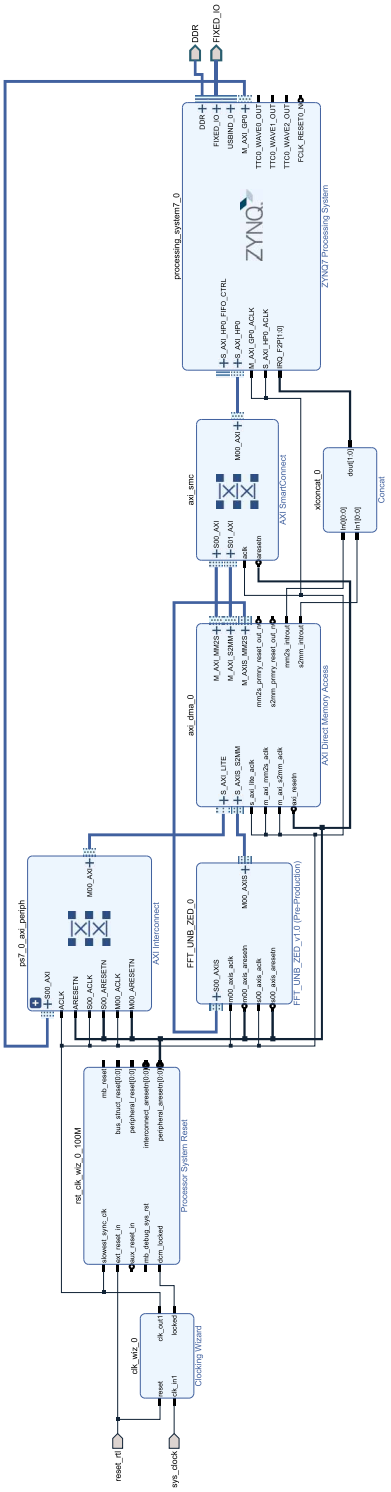


Figura 46 – Block Design completo.
Fonte:(Autor, 2021)

Anexos

ANEXO A – Primeiro Anexo

ANEXO B – Segundo Anexo